Department of Statistical Science, UCL

A Theoretical and Methodological Framework for Machine Learning in Survival Analysis

Enabling Transparent and Accessible Predictive Modelling on Right-Censored Time-to-Event Data

Raphael Edward Benjamin Sonabend

A dissertation submitted in partial fulfilment
of the requirements for the degree of

Doctor of Philosophy

of

University College London

April 2021

I, Raphael Sonabend confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Abstract

Survival analysis is an important field of Statistics concerned with making time-to-event predictions with 'censored' data. Machine learning, specifically supervised learning, is the field of Statistics concerned with using state-of-the-art algorithms in order to make predictions on unseen data. This thesis looks at unifying these two fields as current research into the two is still disjoint, with 'classical survival' on one side and supervised learning (primarily classification and regression) on the other. This PhD aims to improve the quality of machine learning research in survival analysis by focusing on transparency, accessibility, and predictive performance in model building and evaluation. This is achieved by examining historic and current proposals and implementations for models and measures (both classical and machine learning) in survival analysis and making novel contributions.

In particular this includes: i) a survey of survival models including a critical and technical survey of almost all supervised learning model classes currently utilised in survival, as well as novel adaptations; ii) a survey of evaluation measures for survival models, including key definitions, proofs and theorems for survival scoring rules that had previously been missing from the literature; iii) introduction and formalisation of composition and reduction in survival analysis, with a view on increasing transparency of modelling strategies and improving predictive performance; iv) implementation of several R software packages, in particular **mlr3proba** for machine learning in survival analysis; and v) the first large-scale benchmark experiment on right-censored time-to-event data with 24 survival models and 66 datasets.

Survival analysis has many important applications in medical statistics, engineering and finance, and as such requires the same level of rigour as other machine learning fields such as regression and classification; this thesis aims to make this clear by describing a framework from prediction and evaluation to implementation.

Impact Statement

Survival analysis is found throughout many fields of study, including engineering, finance, and healthcare. As such the field has huge benefits throughout academic and non-academic sectors.

Inside academia, this research contributes to the growing interest in machine learning applied to fields outside of regression and classification. Historically there is a clear relationship between: i) ease of understanding a field; ii) its accessibility; iii) the power of predictive models; iv) the rigour and attention with which the field is given; and v) general academic interest. This thesis attempts to improve the rigour and attention within academia by increasing accessibility and understanding. Specifically there has been a division in academia between those studying medical statistics and those interested in machine learning, this thesis attempts to bridge this gap to harness knowledge and expertise from both sides. A full list of the academic contributions that are directly related to the work in this thesis is presented below.

Outside academia, this research is directly applicable to public health and policy, financial institutions, and engineering. This thesis raises and tackles ethical considerations for the correct usage of machine learning in survival analysis, which includes highlighting where this has failed in the past. Failures of this kind are not acceptable when deployed in a public health sector where transparency is of the utmost importance. Similarly it is an ethical question whether the models currently deployed in the public sector are up to the standard expected by modern machine learning techniques. This thesis tackles both problems by increasing the accessibility of machine learning in survival analysis. The impact of this research is already demonstrated by over 44,000 downloads of **mlr3proba**, the machine learning survival analysis package introduced in this thesis. Additionally, **mlr3proba** has been utilised in several research projects, including predicting time-to-death from COVID-19, time-to-falling in Italian rehabilitation centres, and time-to-death from non-small cell lung cancer for Roche.

Acknowledgements

I have three people to thank for pushing me towards attempting and ultimately completing a PhD in Statistics. Firstly, my sixth form Statistics teacher who told me that I would never succeed and therefore motivated me to try. Secondly, Dr. Paul Northrop whose very first lecture on my undergraduate course convinced me to pursue Statistics and who supported me throughout my undergraduate and PhD. Paul's support ensured I was able to complete my PhD and for that I'm incredibly grateful. Thirdly, Dr. Franz Király. Franz taught me everything I know about machine learning, formal mathematics, and software engineering. Franz also provided me with many opportunities for teaching, mentoring, consulting, and external research. My sincere thanks to Franz, whose guidance saved me from a life of accountancy.

I'd also like to thank my second supervisor, Dr Gareth Ambler, for reviewing and proof-reading parts of my thesis, and for very helpful discussions about classical survival models and measures.

Continual support was provided by several peers, to all I am very grateful. This includes Ahmed Guecioueur for discussions about measures, Jeremy Sellier for helping me with my formal mathematics, and István Papp for discussions about ML methods and helping review my thesis. I also want to thank Markus Löning for discussions about losses, toolboxes and software design, helping review my thesis, and many useful conversations. My thanks also to Nurul Ain Toha for discussions about software design, collaborating on several R packages with me, helping with tricky mathematics, and regular calls that have helped keep me grounded. When I was stuck in a writing rut and forgot to exercise, Ekaterina Shatalina helped get me out the flat for socially distanced walks and gave me the space to vent about the stresses of life, as well as reviewing my thesis introduction, many thanks to Ekaterina. Finally, my thanks to Dr. Bilal Mateen, who asked to be acknowledged as my personal mentor. Bilal guided me through my academic journey so far and has provided a huge amount of support, advice, and many opportunities to work on interesting projects alongside great collaborators.

Whilst working on **mlr3proba**, I was warmly welcomed to be a part of *mlr* by Prof. Dr. Bernd Bischl, Dr. Michel Lang, Andreas Bender, Patrick Schratz, and the whole team. Thanks to the team for allowing me the opportunity to contribute to their universe of packages, as well as inviting me to take part in their workshops, and for providing informal training and education in formal code writing and deployment.

During my PhD I was a visiting researcher at The Alan Turing Institute and for three months I took a break from my thesis to work as a full-time researcher for The Turing consulting for Roche. I am very grateful to The Turing for the unlimited supply of food and drink, a tap of hot coffee, and the opportunity to measure 100 ears. Most importantly, I met Dr. Sebastian Vollmer and Dr Ioanna Manolopoulou, both of whom have provided support, great conversation, and further opportunities for work and collaboration.

Whilst finishing editing my thesis I have been employed at Imperial College London and I'd like to thank the entire Department of Infectious Disease Epidemiology for being so welcoming and in particular Dr. Samir Bhatt, Dr. Seth Flaxman, Dr. Marc Baguelin, Professor Neil Ferguson, and the whole COVID-19 Real Time Modelling team.

There have been many moments in the past few years where I could have dropped out of my undergraduate and PhD and so will be forever indebted to my family who helped keep me on track and put up with my mood swings throughout the highs, and more importantly, throughout the lows.

DJ, WK, OB, MK and SM (HTJI), have all improved my mental health whenever it's been low, whether they were aware of it or not.

Thanks to the Engineering and Physical Sciences Research Council (EPSRC) [grant EP/R513143/1] for supporting this thesis as part of the EPSRC Standard Research Studentship (DTP).

Finally, my everlasting thanks to Stephanie Friend. For always encouraging me, being by my side, and helping me complete this journey. Also, for helping edit this thesis, for proof-reading, and for removing un-needed, un-motivated and poorly constructed words and sentences (but not this one). Thank you for supporting me now and always. I promise I'm almost finished working. ILY&ILY.

For John and Spirit,
the doctors that inspired me.

Academic Contributions

The following articles and software are directly related to the work in this thesis.

Journal Articles

Published

- Raphael Sonabend et al. "mlr3proba: An R Package for Machine Learning in Survival Analysis". In: *Bioinformatics* (2021). ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btab039. URL: https://cran.r-project.org/package=mlr3proba
- Raphael Sonabend and Franz J. Kiraly. "set6: R6 Mathematical Sets Interface". In: *Journal of Open Source Software* 5.55 (2020), p. 2598. ISSN: 2475-9066. DOI: 10.21105/joss.02598. URL: https://cran.r-project.org/package=set6

Accepted

- Raphael Sonabend and Franz Kiraly. "distr6: The Complete R6 Probability Distributions Interface". In: *The R Journal* (2021). arXiv: 2009.02993. URL: https://cran.r-project.org/package=distr6

Preprint

- Franz J. Király et al. "Designing Machine Learning Toolboxes: Concepts, Principles and Patterns". In: *arXiv* (2021). arXiv: 2101.04938. URL: http://arxiv.org/abs/2101.04938

R Software

Relevant R software that has not been included in one of the articles above are listed below.

- Raphael Sonabend. *param6: A Fast and Lightweight R6 Parameter Interface*. 2021. URL: https://cran.r-project.org/package=param6
- Raphael Sonabend and Florian Pfisterer. *mlr3benchmark: Benchmarking analysis for 'mlr3'*. 2020. URL: https://cran.r-project.org/package=mlr3benchmark

- Raphael Sonabend and Patrick Schratz. *mlr3extralearners: Extra Learners For mlr3*. 2020. URL: https://github.com/mlr-org/mlr3extralearners
- Raphael Sonabend. *survivalmodels: Models for Survival Analysis*. 2020. URL: https://cran.r-project.org/package=survivalmodels
- Raphael Sonabend. *R62S3: Automatic Method Generation from R6*. 2019. URL: https://cran.r-project.org/package=R62S3

Other Publications

The following academic contributions were also made whilst funded by EPSRC and affiliated with UCL.

- Enora Gandon et al. "Cultural transmission and perception of vessel shapes among Hebron potters". In: *Journal of Anthropological Archaeology* (2021)
- John M Dennis et al. "Type 2 Diabetes and COVID-19–Related Mortality in the Critical Care Setting: A National Cohort Study in England, March–July 2020". In: *Diabetes Care* (2020), p. dc201444. DOI: 10.2337/dc20-1444. URL: http://care.diabetesjournals.org/content/early/2020/10/23/dc20-1444.abstract
- Enora Gandon et al. "Assessing the influence of culture on craft skills: A quantitative study with expert Nepalese potters". In: *PLOS ONE* 15.10 (2020), e0239139. URL: https://doi.org/10.1371/journal.pone.0239139
- Bilal A Mateen et al. "The role of impulsivity in neurorehabilitation: A prospective cohort study of a potential cognitive biomarker for fall risk?" In: *Journal of Neuropsychology* n/a.n/a (2020). ISSN: 1748-6645. DOI: https://doi.org/10.1111/jnp.12239. URL: https://doi.org/10.1111/jnp.12239
- Bilal Mateen and Raphael Sonabend. "All I want for Christmas is... Rigorous validation of predictive models to prevent hasty generalisations". In: *Significance* 16.6 (2019), pp. 20–24. ISSN: 1740-9705. DOI: 10.1111/j.1740-9713.2019.01336.x. URL: https://doi.org/10.1111/j.1740-9713.2019.01336.x
- Franz J Király, Bilal Mateen, and Raphael Sonabend. "NIPS - Not Even Wrong? A Systematic Review of Empirically Complete Demonstrations of Algorithmic Effectiveness in the Machine Learning and Artificial Intelligence Literature". In: *arXiv* (2018). arXiv: 1812.07519. URL: http://arxiv.org/abs/1812.07519

Contents

List of Figures

List of Tables

Symbols and Notation

The most common symbols and notation used throughout this thesis are presented below; in rare cases where different meanings are intended within the thesis, this will be made clear.

Cases, Fonts, and Symbols

Lower-case letters, x, refer to fixed ('realised', 'observed') values and upper-case letters, X, refer to random variables. For example X is a random variable (r.v.) taking values in (t.v.i.) the set $\mathcal{X}$ if, $X : \Omega \to \mathcal{X}$ where Ω is the sample space of all possible outcomes; then $x \in \mathcal{X}$ is a possible realised value from X. A lower-case (Greek or Latin) letter, x, refers to either a single element or a vector, which will be clear from context. Calligraphic letters, $\mathcal{X}$, are used to refer to sets. A lower-case bold-face letter, $\mathbf{x}$, refers to a matrix. If x is a vector then x_i refers to the ith element in this vector. If $\mathbf{x}$ is a matrix then x_i refers to the ith row of the matrix, $x_{:j}$ refers to the jth column of the matrix, and x_{ij} refers to the ith row of the jth column of matrix $\mathbf{x}$. Unless otherwise stated, a 'vector' is used to refer to a column vector. An element with a 'hat', $\hat{x}$, refers to the prediction or estimation of the variable without the hat, x. Inline code and datasets will use `this font` and package names will use **this font**. Finally, any dates will be presented in the ISO format: YYYY-MM-DD.

Italicised text emphasises a word or phrase that is the focus of the sentence or definition. 'Single quotation marks' are most often utilised to signify that the word or phase will either be defined later in the thesis, or to identify when a word should be taken in an English and not mathematical sense, for example 'a good model' would signify that the phrase does not refer to a particular mathematical definition of a model being good. "Double quotation marks" are reserved for direct quotes and are always followed by the associated citation.

Distributions and Random Variables

Two separate notations are used to represent probability distributions and random variables. The first is the 'standard' notation: let X be a random variable following some distribution ζ, then f_X is the probability density function of X.

The second notation instead associates distribution functions directly with the distribution and not the variable. So if ζ is a distribution then $\zeta.f$ is the probability density function of ζ; analogously for other distribution defining functions. This notation is described in full detail when first introduced in the thesis.

Variables

The majority of variables will be defined when required however below are some that are commonly used throughout this thesis.

$\mathbb{R}$	Set of Reals.
$\mathbb{R}_{>0}$	Set of Positive Reals (excluding zero).
$\mathbb{R}_{\geq 0}$	Set of Non-Negative Reals (including zero).
$\bar{\mathbb{R}}$	Set of Extended Reals, equal to $\mathbb{R} \cup \{-\infty, +\infty\}$.
$\mathbb{N}_0$	Set of Naturals (including zero).
$\mathbb{N}_{>0}$	Set of Positive Naturals (excluding zero).
$\mathcal{N}$	Normal distribution.
$\mathcal{U}$	Uniform distribution.
$x, \mathbf{x}, X, \mathcal{X}$	Vector, matrix, random variable, and set of features.
$y, \mathbf{y}, Y, \mathcal{Y}$	Vector, matrix, random variable, and set of true outcomes.
$t, \mathbf{t}, T, \mathcal{T}$	Vector, matrix, random variable, and set of observed time outcomes.
δ, Δ	Vector and random variable of survival/censoring indicators.
β	Vector of model coefficients, or weights.
η	Linear predictor, $X\beta$.
$\zeta.f$	Probability density function of distribution ζ.
$\zeta.F$	Cumulative distribution function of distribution ζ.
$\zeta.h$	Hazard function of distribution ζ.
$\zeta.H$	Cumulative hazard function of distribution ζ.
$\zeta.S$	Survival function of distribution ζ.
$\mathcal{L}$	Likelihood function.

The indicator function, $\mathbb{I}(\cdot)$, expects a well-defined logical statement $(\cdot)$ and equals 1 when this statement is true, and 0 otherwise. Any distribution function with a '0' in the subscript refers to the 'baseline' function, e.g. h_0, S_0 are the baseline hazard and baseline survival functions respectively.

Functions

$\text{Distr}(\mathcal{D})$	Space of distributions over the set $\mathcal{D}$.
$\lvert x \rvert$	Absolute value of x.
$\lVert x \rVert$	Euclidean norm of vector x, $\sqrt{\lvert x_1 \rvert^2 + ... + \lvert x_n \rvert^2}$.
$\bar{x}$	Sample mean of vector x, $\frac{1}{n}\sum_{i=1}^{n} x_i$.
$\mathbb{E}(X)$	Expectation of random variable X.
$\text{Var}(X)$	Variance of random variable X.

Let $f : \mathcal{X} \to \mathcal{Y}$ be any function with domain $\mathcal{X}$ and codomain $\mathcal{Y}$. Then the *function signature* of f is $\mathcal{X} \to \mathcal{Y}$. Arguments and parameters are separated in function signatures by a pipe, '|', where variables to the left are parameters (free variables) and those to the right are arguments (fixed). For example let f be an indicator function that 'checks' if the parameter, ϕ, is below the fixed argument, θ, then f is fully defined by

$$f : \mathbb{R} \times \mathbb{R} \to \{0, 1\}; \quad (\phi|\theta) \mapsto \mathbb{I}(\phi < \theta)$$

Traditionally arguments are not included in the formal signature and the above could be expressed as: Let $\theta \in \mathbb{R}$ then $f : \mathbb{R} \to \{0, 1\}$; $(\phi) \mapsto \mathbb{I}(\phi < \theta)$. The first notation is preferred as it clearly specifies all variables included in the function with their domains, whether they are free or fixed, and cleanly extends to multiple parameters and arguments.

Acronyms

Below is a table of acronyms used throughout this thesis (styled as they appear in the text), these are all fully defined the first time they are used.

AFT	Accelerated Failure Time
APT	Accessible, Performant, Transparent
ANN	Artificial Neural Network
AUC	Area Under the Curve
cdf	Cumulative Distribution Function
chf	Cumulative Hazard Function
CPH	Cox Proportional Hazards
GBM	Gradient Boosting Machine
GLM	Generalised Linear Model
IGS	Integrated Graf Score
IPC(W)	Inverse Probability of Censoring (Weighted)
I(S)LL	Integrated (Survival) Log Loss
KM	Kaplan-Meier
LHS	Left Hand Side
MAE	Mean Absolute Error
ML	Machine Learning
pdf	Probability Density Function
PH	Proportional Hazards
PO	Proportional Odds
RHS	Right Hand Side
(R)MSE	(Root) Mean Squared Error
ROC	Receiver Operating Characteristic
R(S)F	Random (Survival) Forest
r.v.	Random Variable
(S)SVM	(Survival) Support Vector Machine
s.t.	Such That
TNR	True Negative Rate
TPR	True Positive Rate
t.v.i.	Taking Values In
w.r.t.	With Respect To
(W)(S)DLL	(Weighted) (Survival) Density Log Loss

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). This thesis seeks to set the foundations for clear workflows, good practice, and precise results for 'machine learning survival analysis'.

Section 1.1 will elaborate further on the motivation and objectives behind this PhD; research objectives and contributions are then presented in section 1.2.

1.1. Motivations and Objectives

Experiments throughout the literature demonstrate that machine learning survival models often perform worse (or at least no better) than classical statistical models [102, 154, 233, 241] (also see chapter 7).[1] This thesis sets out to explore why this is the case and how this has potential to be improved. The following questions, based on observations of the field, motivated this thesis:

Why are regression and classification more popular than survival analysis in machine learning? There is no doubt that this is the case, for example the 'bibles of machine learning' [23, 118, 145] discuss classification and regression in detail but survival analysis is never discussed. Survival analysis has important applications in healthcare, finance, engineering and more, all fields that directly impact upon individual lives on a day-to-day basis, and should perhaps be considered as important as classification and regression. The result of this gap in interest, is the erroneous assumption that one field can be directly applied to another. For example there is evidence of researchers treating censoring as a nuisance to be ignored and using regression models instead [268]. Censoring is indeed a challenge and may contribute to making survival analysis less accessible than other fields, but this need not be the case; a clear unification of terminology and presentation of methods may help make 'machine learning survival analysis' more accessible. Added accessibility could lead to more academics (and non-academics) engaging with the field and promoting good standards of practice, as well as developing more novel models and measures.

Why are probabilistic survival predictions important? Development of survival models appears to be skewed towards 'ranking models', which predict the relative risk of an event occurring (section 2.3). In many applications these predictions are sufficient, for example in randomised control trials if assessing the increased/decreased risk of an event after treatment. However, there are many use-cases where predicting an individual's survival probability distribution is required. Take, for example, an engineer calculating the lifetime of a plane's

[1] The distinction between a 'classical' and 'machine learning' model used in this thesis is provided in chapter 3.

engine.[1] There are three important reasons to replace a jet engine at the optimal time: i) financial: jet engines are very expensive and replacing one sooner than required is a waste of money; ii) environmental: an engine being replaced too early is a waste of potential usage; and iii) safety: if the engine is replaced too late then there is a risk to passengers. Now consider examples for the three possible 'prediction types' the engineer can make:

i) A 'relative risk prediction': This engine is twice as likely to fail as another.

ii) A 'survival time prediction': The engine is expected to fail in 30 days.

iii) A 'survival distribution prediction': The lifetime of the engine is distributed according to the probability distribution ζ.

The first prediction type is not useful as the underlying relative risk may be unknown and the engineer is concerned with the individual lifetime. The second prediction type provides a useful quantity for the engineer to work with however there is no uncertainty captured in this prediction. The third prediction type can capture the uncertainty of failure over the entirety of the positive Reals (though usually only a small subset is possible and useful). With this final prediction type, the engineer can create safe decisions: 'replace the engine at time τ, where τ is the time when the predicted probability of survival drops below 60%, $S(\tau) = 0.6$'. There are ethical, economic, and environmental reasons for a good survival distribution prediction and this thesis considers a distribution prediction to be the most important prediction type.

How are survival models evaluated? Evaluating predictions from survival models is of the utmost importance. This is especially important as survival models are often deployed in the public domain, particularly in healthcare. Physical products in healthcare, such as new vaccines, undergo rigorous testing and research in randomised control trials before being publically deployed; the same level of rigour should be expected for the evaluation of survival models that are used in life-and-death situations. Evaluation measures for regression and classification are well-understood with important properties, however survival measures have not undergone the same treatment. For example many survival models are still being evaluated solely with concordance indices that have been repeatedly criticised [105, 246, 265]. This paper argues for the use of scoring rules (section 4.6), which simultaneously assess predictions of distribution and relative risk.

Motivated by these questions, this thesis attempts to unify the two fields of machine learning and survival analysis to make the intersection of the two ('machine learning survival analysis') more concise and accessible. This aim is guided by three key themes: Accessibility, Transparency, and Performance. These are now briefly described to explain why they have been identified as key principles for this thesis.

[1] In this engineering context, survival analysis is usually referred to as reliability analysis.

1.1.1. Accessibility, Transparency, and Predictive Performance

In all critical analyses there must be a metric with which to judge the surveyed objects. For example, machine learning models may be judged by predictive performance, i.e. does one model outperform another? Or estimators may be judged according to bias and consistency properties. As this thesis compares multiple different types of objects, a more universal criteria is applied for the reviews, surveys, and comparisons. These are: Accessibility, Transparency, and (predictive) Performance. A model that satisfies all three criteria may be considered APT (accessible, transparent, performant). These key themes are now briefly described and then further discussion is given to why all must be satisfied for this thesis to consider a model or measure to be 'good'. These are primarily explained in terms of a 'model', though all extend naturally to other objects.

A model is termed *accessible* if there either exists an open-source implementation of the model, or sufficient infrastructure and published mathematics for the model to be implementable.[1] For example, a novel neural network without an open-source implementation can still be accessible if the model's architecture is clearly described and can therefore be implemented with neural network packages such as TensorFlow [2].

A model is called *transparent* if its properties are well-understood, its use and manipulation of data is clear, and its predictions have a precise interpretation. The word 'transparent' does not refer to the inner workings of the model and therefore a transparent model could still be a 'black-box'.[2] For example, random forests (section 3.3) are built of hundreds or thousands of individual predictive models, thus making it impossible to fully identify how the final prediction is created. However the model is considered transparent as it is mathematically clear and intuitive how it utilises the individual components to produce its prediction.

A model has good predictive *performance* if its predictions are notably improved over some baseline model [111]. Unlike transparency and accessibility, it is possible to quantify performance and compare this between models (chapter 4). Whilst there is often a trade-off between predictive performance and model interpretability (e.g. compare neural networks and linear regression), this is not the case for predictive performance and transparency. When considering non-predictive objects, such as measures, then performance instead refers to verifying other established performance properties, for example consistency, unbiasedness, and robustness. An object with good performance is called 'performant'.

Performance is traditionally the primary metric by which models (and measures) are judged, but this thesis only considers a model to be 'good' (or APT) if all three of these themes are satisfied. In fact, it can be demonstrated that if even one of these conditions is not satisfied a model can be dishonest or inefficient.

[1] The term 'accessible' is slightly more general than terms such as 'off-shelf' as accessibility is defined to include objects that are not off-shelf but that can be implemented given information provided in the literature.

[2] Therefore the term 'transparent' here does not refer to the concept of a 'glass-box' model, which is the opposite of a black-box model.

By example, take the model that always predicts the height of a person as 42cm. This model is very accessible and transparent but has terrible predictive performance, the model is therefore useless. Now consider a patented model without open-source implementation that not only makes perfect predictions but is also clearly described. In this case as no accessible implementation exists, the model cannot be used and tested by the community and more importantly cannot be externally validated, leading to ethical questions about commercial implementation and even whether the results can be trusted. Finally, in the case of an accessible model with strong predictive performance but without clear description in a paper or reader-friendly code/documentation, there can only be limited trust in the model's performance, especially with respect to future performance.

1.1.2. Research Objectives

This PhD aims to understand how machine learning has been used in survival analysis historically, at present, and how it could, or even should, be utilised in the future. This is achieved by the following concrete objectives:

O1) Mathematically define predictive formulations for 'machine learning survival analysis' with unified notation and terminology from the separate fields of machine learning and survival analysis.

O2) Complete a critical survey of machine learning survival models to determine which are APT.

O3) Survey metrics for evaluation of survival models ('survival measures'). Where required, provide novel definitions and proofs to extend or critique capabilities of survival measures, in particular for 'scoring rules'.

O4) Formalise the concepts of 'composition and reduction' in survival analysis. Design and implement strategies for survival analysis that help further the three themes.

O5) Improve accessibility of machine learning survival analysis via implementation of an open-source R package, **mlr3proba**.

O6) Perform the first large-scale benchmark experiment of commonly utilised classical and machine learning survival models.

The next section details the contributions in this thesis that address these research objectives.

1.2. Contributions and Thesis Structure

Contributions made by the author and external collaborators are first listed separately, and then the thesis structure is detailed.

1.2.1. Personal Contributions

Key contributions made by the author are listed below.

- *Chapter 1*: Derivation of the accessibility, transparency, and predictive performance themes for reviewing models and measures.
- *Chapter 2*: Formalisation of survival analysis as a machine learning task, including identification and formalisation of distinct survival predict types and tasks [162].
- *Chapter 3*: Survey of machine learning models for survival analysis with a return type taxonomy and with a survey focus on the three identified themes. Proposals of novel adaptations for machine learning methods, primarily focusing on optimisation of distribution predictions.
- *Chapter 4*: Literature review of in-sample measures. Survey of evaluation measures for evaluating survival models. Definitions for survival losses, approximate survival losses, and properness properties of survival losses. Survey of survival scoring rules utilised in the literature and definition of novel losses. Proofs that common survival scoring rules are improper. Proof of theorem relating strictly proper regression and survival losses. Conjectures on the strict properness of survival losses to guide future research.
- *Chapter 5*: Design and formalisation of 5 composition and 8 reduction strategies for survival analysis.
- *Chapter 6*: Design and implementation of software packages: **R62S3**, **set6**, **distr6**, **mlr3proba**, **mlr3extralearners**, **mlr3benchmark**, **survivalmodels**, **param6**. First author on three publications included in the chapter for **set6** [278], **distr6** [277], and **mlr3proba** [281].
- *Chapter 7*: Conducted first large-scale benchmark experiment of machine learning survival models. Derived simulated data for benchmark experiments. Collected real datasets for experiments.

1.2.2. External Contributions

Key contributions made by external collaborators are listed below.

- *Chapter 5*: Composition strategies (C1) and (C3) were initially discussed with Dr. Franz Király (FK) and Prof. Dr. Bernd Bischl (BB). Reduction strategies (R7) and (R8) were originally derived as part of a project for Nuffield Health. Initial formalisation of (R7) and (R8) were written up in the author's BSc dissertation.
- *Chapter 6*
  - *Section 6.2*: Parts of this section: the use-cases and parts of the introduction, were directly copied from a paper in the *Journal of Open*

Source Software [278], which was written with FK. Notable bottlenecks in the software were identified by Jakob Richter (JR), solving these lead to significant improvements in speed of construction and comparison of sets.

- *Section 6.3*: This section is copied directly from a paper accepted in *The R Journal* [277], which was written with FK. FK introduced the novel distribution notation and contributed significantly to the introduction and the section on composite distributions. The designs of **distr6** were based on the **distr** [258] package built by Prof. Dr. Peter Ruckdeschel and Prof. Dr. Matthias Kohl, who both contributed to design discussions for the package. BB also helped informed some early design choices. The kernel interface in **distr6** was primarily implemented by Nurul Ain Toha. Several distributions were implemented by a team of undergraduates at UCL: Shen Chen, Jordan Deenichin, Chengyang Gao, Chloe Zhaoyuan Gu, Yunjie He, Xiaowen Huang, Shuhan Liu, Runlong Yu, Chijing Zeng and Qian Zhou.
- *Section 6.4*: Parts of this section were copied from a paper in *Bioinformatics* [281], written with FK, Andreas Bender (AB), BB, and Michel Lang (ML); these were: the motivating example, significant parts of the introduction, the 'Related software' section, and parts of the 'Return Types' paragraph. **mlr3proba** was originally built from **mlr3survival**[1], created by ML. **mlr3survival** contained five learners (coxph, glmnet, ranger, rpart), two measures (harrellc, unoc), and original designs for the survival prediction, learner, and task classes. Since merging **mlr3survival** into **mlr3proba**, these have been modified however some of the original code remains. Design discussions for **mlr3proba** were held with FK, BB, ML, JR, and Martin Binder. Implementation of some learners were made by AB.

- *Chapter 7*: Discussions about the analysis of the real data experiments were held with FK, BB, ML, and Florian Pfisterer. The code to analyse these experiments was primarily written by the package author and implemented in **mlr3benchmark**, with some code written by ML and Patrick Schratz.

In addition to direct contributions, discussions with the primary supervisor, Dr Franz Király, and secondary supervisor, Dr Gareth Ambler, contributed to work in all chapters.

1.2.3. Thesis Structure

- *Chapter 2* introduces the survival and machine learning settings separately. First a mathematical overview to survival analysis is provided (section 2.1) and then the scope of this thesis is identified and justified (section 2.2). Survival prediction types are then mathematically defined for use throughout the thesis (section 2.3). This is then mirrored for machine learning by first introducing supervised learning and important machine learning methods

[1] https://github.com/mlr-org/mlr3survival

(section 2.4) and then defining survival analysis as a machine learning task (section 2.5).

- *Chapter 3* reviews classical survival models (section 3.1) and surveys machine learning survival models (sections 3.2-3.6). Only a short review is provided for the classical setting as this has been covered extensively in the literature over the past few decades. This thesis takes a novel approach by focusing the classical review on model prediction types, in order to gain clarity in understanding how the models can and cannot be utilised. The rest of the chapter critically surveys the use of machine learning in survival analysis. The survey is first split by machine learning classes, and then further categorised again by model prediction types.

- *Chapter 4* discusses how to evaluate the models introduced in the previous chapter. This starts (section 4.1) with a general discussion about the importance of evaluation and how survival measures must be selected to relate to the correct survival task. The chapter continues with a full review of the different types of survival measures, how these relate, and what properties exist for the most common of these. Extensive discussion is given to survival scoring rules (section 4.6) including introducing and completing novel definitions (section 4.6.2) and proofs (section 4.6.4).

- *Chapter 5* introduces the concepts of composition and reduction to survival analysis. The chapter beings with an introduction to composition (section 5.2) as an abstract concept before identifying how it is already prevalent in survival analysis. The closely related concept of reduction is then introduced (section 5.3). Concrete composition (section 5.4) and reduction (section 5.5) strategies are then detailed.

- *Chapter 6* focuses entirely on software engineering and introduces the packages that have been published to the *Comprehensive R Archive Network* (CRAN)[1]. The chapter begins (section 6.1) with a general overview to the ecosystem that the packages live in, as well as their original motivations. Then three packages are discussed in greater detail (sections 6.2-6.4).

- *Chapter 7* draws together results from all previous chapters in a large-scale benchmark experiment of survival models for right-censored survival data.

- *Chapter 8* concludes the thesis and details potential future plans and research.

[1] `https://cran.r-project.org/`

1.3. Code and Reproducibility

Finally, some brief words on the programming present in this thesis.

Programming Languages This thesis includes simulations and figures generated in R and the benchmark experiments in chapter 7 are also conducted in R. Some Python implementations are considered in chapter 3. Only R and Python are considered as they are the two most popular open-source programming languages that intersect classical statistics and machine learning. Further discussion on these languages is provided in chapter 6.

Reproducibility The R code for any figures or experiments in this thesis are freely available in a public[1] GitHub repository[2] under an MIT license.[3] For any code that requires specific software packages, these are listed when required alongside version numbers. All R scripts have set seeds for reproducibility. The code used in this thesis was run using various R versions from 3.6 to 4.0.2 and whilst this should not affect reproducibility, this cannot be guaranteed.

[1] This repository will remain private until the experiment in chapter 7 has been released as a pre-print (target: 2021-05-01). Until this time access will be granted upon request.

[2] https://github.com/RaphaelS1/thesis_supplementary

[3] https://opensource.org/licenses/MIT

Chapter 2

Survival Analysis and Machine Learning

In their broadest and most basic definitions, survival analysis is the study of temporal data from a given origin until the occurrence of one or more events or 'end-points' [55], and machine learning is the study of models and algorithms that learn from data in order to make predictions or find patterns [118]. Reducing either field to these definitions is ill-advised.

This chapter collects terminology utilised in survival analysis (section 2.1) and machine learning (section 2.4) in order that this thesis can cleanly discuss 'machine learning survival analysis' (section 2.5). Once the mathematical setting is set up, the thesis scope is fully presented in section 2.2. Whilst the content of this chapter is not novel with respect to either survival analysis or machine learning separately, this does appear to be the first formulation of the survival analysis machine learning 'task' [162].

2.1. Survival Analysis

Survival analysis is the field of Statistics concerned with the analysis of time-to-event data, which consists of covariates, a categorical (often binary) outcome, and the time until this outcome takes place (the 'survival time'). As a motivating example of time-to-event data, say 100 patients are admitted to a COVID-19 ward and for each patient the following covariate data are collected: age, weight and sex; additionally for each patient the time until death or discharge is recorded. In the time-to-event dataset, which takes a standard tabular form, each of the 100 patients is a row, with columns consisting of age, weight, and sex measurements, as well as the outcome (death or discharge) and the time to outcome.

Survival analysis is distinct from other areas of Statistics due to the incorporation of 'censoring', a mechanism for capturing uncertainty around when an event occurs in the real-world. Continuing the above example, if a patient dies of COVID-19 five dies after admittance, then their outcome is exactly known: they *died* after five days. Consider now a patient who is discharged after ten days. As death is a guaranteed event they have a true survival time but this may

be decades later, therefore they are said to be *censored* at ten days. This is a convenient method to express that the patient survives up to ten days and their survival status at any time after this point is unknown. Censoring is a unique challenge to survival analysis that attempts to incorporate as much information as possible without knowing the true outcome. This is a 'challenge' as statistical models usually rely on learning from observed, i.e. known, outcome data; therefore censoring requires special treatment.

Whilst survival analysis occurs in many fields, for example as 'reliability analysis' in engineering and 'duration analysis' in economics, in this thesis the term 'survival' will always be used. Moreover the following terminology, analogous to a healthcare setting, are employed: survival analysis (or 'survival' for short) refers to the field of study; the event of interest is the 'event', or 'death'; an observation that has not experienced an event is 'censored' or 'alive'; and observations are referred to as 'observations', 'subjects', or 'patients'.

Some of the biggest challenges in survival analysis stem from an unclear definition of a 'survival analysis prediction' and different (sometimes conflicting) common notations. This thesis attempts to make discussions around survival analysis clearer and more precise by first describing the mathematical setting for survival analysis in section 2.1.1 and only then defining the prediction types to consider in section 2.3.

2.1.1. Survival Data and Definitions

Survival analysis has a more complicated data setting than other fields as the 'true' data generating process is not directly modelled but instead engineered variables are defined to capture observed information. Let,

- X t.v.i. $\mathcal{X} \subseteq \mathbb{R}^p, p \in \mathbb{N}_{>0}$ be the generative random variable representing the data *features/covariates/independent variables*.
- Y t.v.i. $\mathcal{T} \subseteq \mathbb{R}_{\geq 0}$ be the (unobservable) *true survival time*.
- C t.v.i. $\mathcal{T} \subseteq \mathbb{R}_{\geq 0}$ be the (unobservable) *true censoring time*.

It is impossible to fully observe both Y and C. This is clear by example: if an observation drops out of a study then their censoring time is observed but their event time is not, whereas if an observation dies then their true censoring time is unknown. Hence, two engineered variables are defined to represent observable outcomes. Let,

- $T := \min\{Y, C\}$ be the *observed outcome time*.
- $\Delta := \mathbb{I}(Y = T) = \mathbb{I}(Y \leq C)$ be the *survival indicator* (also known as the *censoring* or *event* indicator).[1]

Together (T, Δ) is referred to as the *survival outcome* or *survival tuple* and they form the dependent variables. The survival outcome provides a concise

[1]Indicators are usually named to reflect a positive condition in the function (in this case the event when $Y = T$), but counter to this convention the 'censoring indicator' is possibly the most common term.

mechanism for representing the time of the *observed* outcome and indicating which outcome (death or censoring) took place.

Now the full generative template for survival analysis is given by (X, Δ, C, Y, T) t.v.i. $\mathcal{X} \times \{0,1\} \times \mathcal{T} \times \mathcal{T} \times \mathcal{T}$ and with $(X_i, \Delta_i, C_i, Y_i, T_i)$ jointly i.i.d. A *survival dataset* is defined by $\mathcal{D} = \{(X_1, T_1, \Delta_1), ..., (X_n, T_n, \Delta_n)\}$ where $(X_i, T_i, \Delta_i) \overset{i.i.d.}{\sim} (X, T, \Delta)$ and X_i is a p-vector, $X_i = (X_{i;1}, ..., X_{i;p})$. Though unobservable, the true outcome times are defined by $(Y_1, C_1), ..., (Y_n, C_n)$ where $(Y_i, C_i) \overset{i.i.d.}{\sim} (Y, C)$.

Table 1 exemplifies a random survival dataset with n observations (rows) and p features.

Table 1: Theoretical time-to-event dataset. (Y, C) are 'hypothetical' as they can never be directly observed. Rows are individual observations, X columns are features, T is observed time-to-event, Δ is the censoring indicator, and (Y, C) are hypothetical true survival and censoring times.

X			T	Δ	Y	C
X_{11}	$\cdots$	X_{1p}	T_1	Δ_1	Y_1	C_1
$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
X_{n1}	$\cdots$	X_{np}	T_n	Δ_n	Y_n	C_n

Table 2 exemplifies an observed survival dataset with a modified version of the `rats` dataset [291].

Table 2: `rats` [291] time-to-event dataset with added hypothetical columns (Y, C). Rows are individual observations, X columns are features, T is observed time-to-event, Δ is the censoring indicator, and (Y, C) are hypothetical (here arbitrary values dependent on (T, Δ)) true survival and censoring times.

X			T	Δ	Y	C
litter $(X_{\cdot;1})$	**rx** $(X_{\cdot;2})$	**sexF** $(X_{\cdot;3})$	**time**	**status**	**survTime**	**censTime**
1	1	1	101	0	105	101
1	0	1	49	1	49	55
1	0	1	104	0	200	104
2	1	0	91	0	92	91
2	0	0	104	1	104	104
2	0	0	102	1	102	120

Both datasets includes two extra columns, on the right of the triple vertical line, which imagine hypothetical data for the unobserved true survival and censoring times.

Finally the following terms are used frequently throughout this report. Let $(T_i, \Delta_i) \overset{i.i.d.}{\sim} (T, \Delta), i = 1, ..., n$, be random survival outcomes. Then,

i) The *set of unique* or *distinct time-points* refers to the set of time-points in which at least one observation dies or is censored, $\mathcal{U}_O := \{T_i\}_{i \in \{1,...,n\}}$.

ii) The *set of unique death times* refers to the set of unique time-points in which death (and not censoring) occurred, $\mathcal{U}_D := \{T_i : \Delta_i = 1\}_{i \in \{1,\dots,n\}}$.

iii) The *risk set* at a given time-point, τ, is the set of subjects who are known to be alive (not dead or censored) just before that time, $\mathcal{R}_\tau := \{i : T_i \geq \tau\}$ where i is a unique row/subject in the data.

iv) The *number of observations alive* at τ is the cardinality of the risk set, $|\mathcal{R}_\tau|$, and is denoted by $n_\tau := \sum_i \mathbb{I}(T_i \geq \tau)$.

v) The *number of observations who die* at τ is denoted by $d_\tau := \sum_i \mathbb{I}(T_i = \tau, \Delta_i = 1)$.

vi) The Kaplan-Meier estimate of the average survival function of the training data *survival distribution* is the Kaplan-Meier estimator (section 3.1.1) fit (section 2.4.1) on training data (T_i, Δ_i) and is denoted by $\hat{S}_{KM}$.

vii) The Kaplan-Meier estimate of the average survival function of the training data *censoring distribution* is the Kaplan-Meier estimator fit on training data $(T_i, 1 - \Delta_i)$ and is denoted by $\hat{G}_{KM}$.

Notation and definitions will be recapped at the start of each chapter for convenience.

2.1.2. Censoring

Censoring is now discussed in more detail and important concepts introduced. Given the survival generating process (X, T, Δ) with unobservable (Y, C), the event is experienced if $Y \leq C$ and $\Delta = 1$ or censored if $\Delta = 0$.

Censoring 'Location' Right-censoring is the most common form of censoring in survival models and it occurs either when a patient drops out (but doesn't experience the event) of the study before the end and thus their outcome is unknown, or if they experience the event at some unknown point after the study end. Formally let $[\tau_l, \tau_u]$ be the study period for some, $\tau_l, \tau_u \in \mathbb{R}_{\geq 0}$. Then right-censoring occurs when either $Y > \tau_u$ or when $Y \in [\tau_l, \tau_u]$ and $C \leq Y$. In the first case $T = C = \tau_u$ and censoring is due to the true time of death being unknown as the observation period has finished. In the latter case, a separate censoring event, such as drop-out or another competing risk, is observed.

Left-censoring is a rarer form of censoring and occurs when the event happens at some unknown time before the study start, $Y < \tau_l$. Interval-censoring occurs when the event takes place in some interval within the study period, but the exact time of event is unknown. Figure 1 shows a graphical representation of right-censoring.

Censoring 'Dependence' Censoring is often defined as *uninformative* if $Y \perp\!\!\!\perp C$ and *informative* otherwise however these definitions can be misleading as the term 'uninformative' appears to be imply that censoring is independent of both X and Y, and not just Y. Instead the following more precise definitions are used in this report.

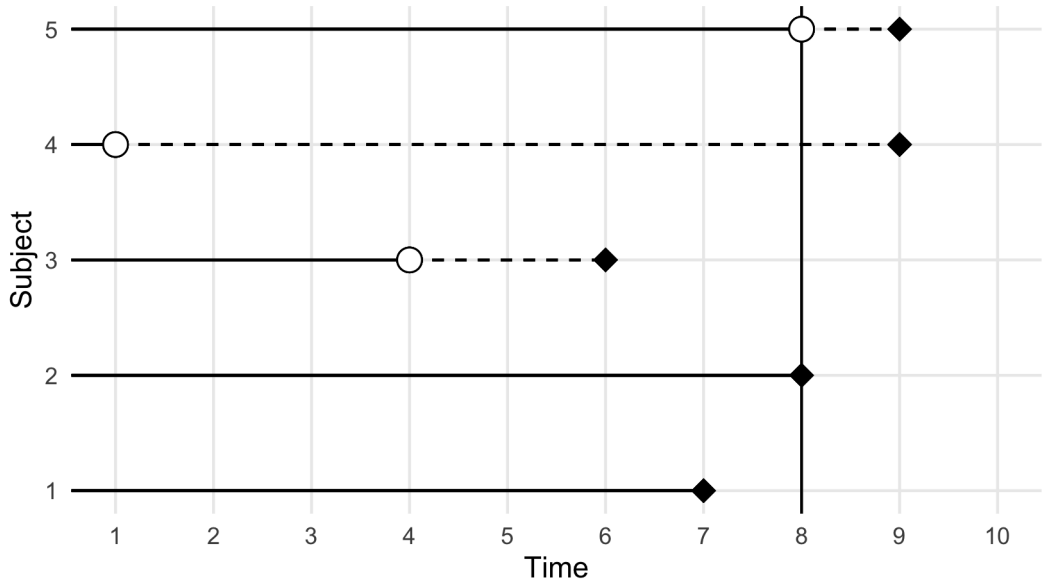


Figure 1: Dead and censored subjects (y-axis) over time (x-axis). Black diamonds indicate true death times and white circles indicate censoring times. Vertical line is the study end time. Subjects 1 and 2 die in the study time. Subject 3 is censored in the study and (unknown) dies within the study time. Subject 4 is censored in the study and (unknown) dies after the study. Subject 5 dies after the end of the study.

Definition 2.1.1. Let (X, T, Δ, Y, C) be defined as above, then

i) If $C \perp\!\!\!\perp X$, censoring is *feature-independent*, otherwise censoring is *feature-dependent*.

ii) If $C \perp\!\!\!\perp Y$, then censoring is *event-independent*, otherwise censoring is *event-dependent*.

iii) If $(C \perp\!\!\!\perp Y)|X$, censoring is conditionally independent of the event given covariates, or *conditionally event-independent*.

iv) If $C \perp\!\!\!\perp (X, Y)$ censoring is *uninformative*, otherwise censoring is *informative*.

Non-informative censoring can generally be well-handled by models as true underlying patterns can still be detected and the reason for censoring does not affect model inference or predictions. However in the real-world, censoring is rarely non-informative as reasons for drop-out or missingness in outcomes tend to be related to the study of interest. Event-dependent censoring is a tricky case that, if not handled appropriately (by a competing-risks framework), can easily lead to poor model development; the reason for this can be made clear by example: Say a study is interested in predicting the time between relapses of stroke but a patient suffers a brain aneurysm due to some separate neurological condition, then there is a high possibility that a stroke may have occurred if the aneurysm had not. Therefore a survival model is unlikely to distinguish the censoring event (aneurysm) from the event of interest (stroke) and will confuse predictions. In practice, the majority of models and measures assume that censoring is conditionally event-independent and hence censoring can be predicted by the covariates whilst not directly depending on the event. For example if studying the survival

time of ill pregnant patients in hospital, then dropping out of the study due to pregnancy is clearly dependent on how many weeks pregnant the patient is when the study starts (for the sake of argument assume no early/late pregnancy due to illness).

Type I Censoring Type I and Type II censoring are special-cases of right-censoring, only Type I is discussed in this thesis as it is more common in simulation experiments. Type I censoring occurs if a study has a set end-date, or maximum survival time, and a patient survives until the end of the study. If survival times are dependent on covariates (i.e. not random) and the study start date is known (or survival times are shifted to the same origin) then Type I censoring will usually be informative as censored patients will be those who survived the longest.

2.2. Thesis Scope

Now that the mathematical setting has been defined, the thesis scope is provided. For time and relevance the scope of this thesis is narrowed to the most parsimonious setting that is genuinely useful in modelling real-world scenarios. This is the setting that captures all assumptions made by the majority of proposed survival models and therefore is practical both theoretically and in application. This setting is defined by the following assumptions (with justifications):

i) Let p be the proportion of censored observations in the data, then $p \in (0, 1)$. This open interval prevents the case when $p = 0$, which is simply a regression problem (section 2.4.2.2), or the case when $p = 1$, in which no useful models exist (as the event never occurs).

ii) Only right-censoring is observed in the data, no left- or interval-censoring. This accurately reflects most real-world data in which observations that have experienced the event before the study start (left-censoring) are usually not of interest, and close monitoring of patients means that interval-censoring is unlikely in practice. It is acknowledged that left-truncation is a common problem in medical datasets though this is often handled not by models but by data pre-processing, which is not part of the workflow discussed in this thesis.

iii) There is only one event of interest, an observation that does not experience this event is censored. This eliminates the 'competing risk' setting in which multiple events of interest can be modelled.

iv) The event can happen at most once. For example the event could be death or initial diagnosis of a disease however cannot be recurrent such as seizure. In the case where the event could theoretically happen multiple times, only the time to one (usually the first) occurrence of the event is modelled.

v) The event is guaranteed to happen at least once. This is an assumption implicitly made by all survival models as predictions are for the time until the true event, Y, and not the observed outcome, T.

For both the multi-event and recurrent-event cases, simple reductions exist such that these settings can be handled by the models discussed in this paper however this is not discussed further here.

No assumptions are made about whether censoring is dependent on the data but when models and measures make these assumptions, they will be explicitly discussed.

The purpose of any statistical analysis is dependent on the research question. For example techniques are available for data analysis, imputation, exploration, prediction, and more. This thesis focuses on the predictive setting; other objectives, such as model inspection and data exploration can be achieved post-hoc via interpretable machine learning techniques [219].

Finally, the methods in this thesis are restricted to frequentist statistics. Bayesian methods are not discussed as the frequentist setting is usually more parsimonious and additionally there are comparatively very few off-shelf implementations of Bayesian survival methods. Despite this, it is noted that Bayesian methods are particularly relevant to the research in this thesis, which is primarily concerned with uncertainty estimates and predictions of distributions. Therefore, a natural extension to the work in this thesis would be to fully explore the Bayesian setting.

2.3. Survival Prediction Problems

This section continues by defining the survival problem narrowed to the scope described in the previous section. Defining a single 'survival prediction problem' (or 'task') is important mathematically as conflating survival problems could lead to confused interpretation and evaluation of models. Let (X, T, Δ) and $\mathcal{D}$ be as defined above. A general survival prediction problem is one in which: i) a survival dataset, $\mathcal{D}$, is split (section 2.4.1) for training, $\mathcal{D}_0$, and testing, $\mathcal{D}_1$; ii) a survival model is fit on $\mathcal{D}_0$; and iii) the model predicts some representation of the unknown true survival time, Y, given $\mathcal{D}_1$.

The process of 'fitting' is model-dependent, and can range from simple maximum likelihood estimation of model coefficients, to complex algorithms. The model fitting process is discussed in more abstract detail in section 2.4 and then concrete algorithms are discussed in chapter 3. The different survival problems are separated by 'prediction types' or 'prediction problems', these can also be thought of as predictions of different 'representations' of Y. Four prediction types are discussed in this paper, these may be the only possible survival prediction types and are certainly the most common as identified in chapters 3 and 4. They are predicting:

i) The *relative risk* of an individual experiencing an event – A single continuous ranking.

ii) The *time until an event* occurs – A single continuous value.

iii) The *prognostic index* for a model – A single continuous value.

iv) An individual's *survival distribution* – A probability distribution.

The first three of these are referred to as *deterministic* problems as they predict a single value whereas the fourth is *probabilistic* and returns a full survival distribution. Definitions of these are expanded on below.

Survival predictions differ from other fields in two respects. Firstly, the predicted outcome, Y, is a different object than the outcome used for model training, (T, Δ). This differs from, say, regression in which the same object (a single continuous variable) is used for fitting and predicting. Secondly, with the exception of the time-to-event prediction, all other prediction types do not predict Y but some other related quantity.

Survival prediction problems must be clearly separated as they are inherently incompatible. For example it is not meaningful to compare a relative risk prediction from one observation to a survival distribution of another. Whilst these prediction types are separated above, they can be viewed as special cases of each other. Both (1) and (2) may be viewed as variants of (3); and (1), (2), and (3) can all be derived from (4); this is elaborated on below.

Relative Risk/Ranking This is perhaps the most common survival problem and is defined as predicting a continuous rank for an individual's 'relative risk of experiencing the event'. For example, given three patients, $\{i, j, k\}$, a relative risk prediction may predict the 'risk of event' as $\{0.1, 0.5, 10\}$ respectively. From these predictions, the following types of conclusions can be drawn:

i) Conclusions comparing patients. e.g. i is at the least risk; the risk of j is only slightly higher than that of i but the risk of k is considerably higher than j; the corresponding ranks for i, j, k, are $1, 2, 3$.

ii) Conclusions comparing risk groups. e.g. thresholding risks at 1.0 places i and j in a 'low-risk' group and k in a 'high-risk' group

So whilst many important conclusions can be drawn from these predictions, the values themselves have no meaning when not compared to other individuals. Interpretation of these rankings has historically been conflicting in implementation, with some software having the interpretation 'higher ranking implies higher risk' whereas others may indicate 'higher ranking implies lower risk' (section 6.4.4.2). In this thesis, a higher ranking will always imply a higher risk of event (as in the example above).

Time to Event Predicting a time to event is the problem of predicting the deterministic survival time of a patient, i.e. the amount of time for which they are predicted to be alive after some given start time. Part of the reason this problem is less common in survival analysis is because it borders regression – a single continuous value is predicted – and survival – the handling of censoring is required – but neither is designed to solve this problem directly. Time-to-event predictions can be seen as a special-case of the ranking problem as an individual with a predicted longer survival time will have a lower overall risk, i.e. if t_i, t_j and r_i, r_j are survival time and ranking predictions for patients i and j respectively, then $t_i > t_j \rightarrow r_i < r_j$.

Prognostic Index Given covariates, $x \in \mathbb{R}^{n \times p}$, and a vector of model coefficients, $\beta \in \mathbb{R}^p$, the linear predictor is defined by $\eta := x\beta \in \mathbb{R}^n$. The 'prognostic index' is a term that is often used in survival analysis papers that usually refers to some transformation (possibly identity), ϕ, on the linear predictor, $\phi(\eta)$. Assuming a predictive function (for survival time, risk, or distribution defining function (see below)) of the form $g(\varphi)\phi(\eta)$, for some function g and variables φ where $g(\varphi)$ is constant for all observations (e.g. Cox PH (section 3.1.2)), then predictions of η are a special case of predicting a relative risk, as are predictions of $\phi(\eta)$ if ϕ is rank preserving. A higher prognostic index may imply a higher or lower risk of event, dependent on the model structure.

Survival Distribution Predicting a survival distribution refers specifically to predicting the distribution of an individual patient's survival time, i.e. modelling the distribution of the event occurring over $\mathbb{R}_{\geq 0}$. Therefore this is seen as the probabilistic analogue to the deterministic time-to-event prediction, these definitions are motivated by similar terminology in machine learning regression problems (section 2.4). The above three prediction types can all be derived from a probabilistic survival distribution prediction (section 5.4).

A survival distribution is a mathematical object that is estimated by predicting a *representation* of the distribution. Let W be a continuous random variable t.v.i. $\mathbb{R}_{\geq 0}$ with probability density function (pdf), $f_W : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$, and cumulative distribution function (cdf), $F_W : \mathbb{R}_{\geq 0} \to [0, 1]; (\tau) \mapsto P(W \leq \tau)$. The pdf, $f_W(\tau)$, is the likelihood of an observation dying in a small interval around time τ, and $F_W(\tau) = \int_0^\tau f_W(\tau)$ is the probability of an observation being dead at time τ (i.e. dying at or before τ). In survival analysis, it is generally more interesting to model the risk of the event taking place or the probability of the patient being alive, leading to other distribution representations of interest.

The survival function is defined as

$$S_W : \mathbb{R}_{\geq 0} \to [0, 1]; \quad (\tau) \mapsto P(W \geq \tau) = \int_\tau^\infty f_W(u)\, du \tag{2.3.1}$$

and so $S_W(\tau) = 1 - F_W(\tau)$. This function is known as the survival function as it can be interpreted as the probability that a given individual survives until some point $\tau \geq 0$.

Another common representation is the hazard function,

$$h_W : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}; \quad (\tau) \mapsto \frac{f_W(\tau)}{S_W(\tau)} \tag{2.3.2}$$

The hazard function is interpreted as the instantaneous risk of death given that the observation has survived up until that point; note this is not a probability as h_W can be greater than one.

The cumulative hazard function (chf) can be derived from the hazard function by

$$H_W : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}; \quad (\tau) \mapsto \int_0^\tau h_W(u)\, du \tag{2.3.3}$$

The cumulative hazard function relates simply to the survival function by

$$H_W(\tau) = \int_0^\tau h_W(u)\ du = \int_0^\tau \frac{f_W(u)}{S_W(u)}\ du = \int_0^\tau -\frac{S'_W(u)}{S_W(u)}\ du = -\log(S_W(\tau)) \tag{2.3.4}$$

Any of these representations may be predicted conditionally on covariates for an individual by a probabilistic survival distribution prediction. Once a function has been estimated, predictions can be made conditional on the given data. For example if n survival functions are predicted, $\hat{S}_1, ..., \hat{S}_n$, then $\hat{S}_i$ is interpreted as the predicted survival function given covariates of observation i, and analogously for the other representation functions.

2.4. Machine Learning

This section begins with a very brief introduction to machine learning and a focus on regression and classification; the survival machine learning task is then introduced (section 2.5). Of the many fields within machine learning (ML), the scope of this thesis is narrowed to supervised learning. Supervised learning is the sub-field of ML in which predictions are made for outcomes based on data with observed dependent and independent variables. For example predicting someone's height is a supervised learning problem as data can be collected for features (independent variables) such as age and sex, and outcome (dependent variable), which is height. Predictive survival analysis problems fall naturally in the supervised learning framework as there are identifiable features and (multiple types of) outcomes.

2.4.1. Terminology and Methods

Common supervised learning methods are discussed in a simplified setting with features X t.v.i. $\mathcal{X}$ and outcomes Y t.v.i. $\mathcal{Y}$; usually outcomes are referred to as 'targets' (a 'target for prediction'). Let $\mathcal{D}_0 = \{(X_1, Y_1), ..., (X_n, Y_n)\}$ be a (training) dataset where $(X_i, Y_i) \overset{i.i.d.}{\sim} (X, Y)$. The methods below extend naturally to the survival setting.

Strategies and Models In order to clearly separate between similar objects, several terms for machine learning are now introduced and clearly distinguished.

Let $g : \mathcal{X} \to \mathcal{Y}$ be the true (but unknown) mapping from the features to outcomes, referred to as the *true prediction functional*. Let $\mathcal{G}$ be the set of *prediction functionals* such that $\forall \Upsilon \in \mathcal{G}, \Upsilon : \mathcal{X} \to \mathcal{Y}$. A *learning* or *fitting algorithm* is defined to be any function of the form $\mathcal{A} : \mathcal{X}^n \times \mathcal{Y}^n \to \mathcal{G}$. The goal of supervised learning is to *learn* g with a learning algorithm *fit* on (i.e. the input to the algorithm is) training data, $\hat{g} := \mathcal{A}(\mathcal{D}_0) \in \mathcal{G}$. Note that $\hat{g}$ may take hyper-parameters that can be set or tuned (see below). The learning algorithm is 'good' if $\hat{g}(X) \approx g(X)$ (see 'Evaluation' below).

The learning algorithm is determined by the chosen *learning strategy* and *model*, where a model is a complete specification of a learning strategy including hyper-parameters. These terms are more clearly illustrated by example:

i) Learning strategy – simple linear regression

ii) Model – $y = \beta_0 + \beta_1 x$ where $x \in \mathbb{R}$ is a single covariate, $y \in \mathbb{R}$ is the target, and $\beta_0, \beta_1 \in \mathbb{R}$ are model coefficients.

iii) Learning algorithm (model fitting) – Minimise the residual sum of squares: $(\hat{\beta}_0, \hat{\beta}_1) := \text{argmin}_{\beta_0,\beta_1}\{\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i)^2\}$ for $(x_i, y_i) \in \mathcal{D}_0, i = 1, ..., n$.

iv) Prediction functional – $\hat{g}(x) = \hat{\beta}_0 + \hat{\beta}_1 x$

To further illustrate the difference between learning strategy and model, note that the same learning strategy 'simple linear regression' could either utilise the model above or instead a model without intercept, $y = \beta x$, in which case the learning algorithm and prediction functional would also be modified.

The model in (ii) is called *unfitted* as the model coefficients are unknown and the model cannot be used for prediction. After step (iii) the model is said to be fit to the training data and therefore the model is *fitted*.[1] It is common to refer to the learning algorithm (and associated hyper-parameters) as the unfitted model and to refer to the prediction functional (and associated hyper-parameters) as the fitted model.

Evaluation Models are *evaluated* by evaluation measures called *losses* or *scores*,[2] $L : \mathcal{Y} \times \mathcal{Y} \to \bar{\mathbb{R}}$. Let $(X^*, Y^*) \sim (X, Y)$ be test data (i.e. independent of $\mathcal{D}_0$) and let $\hat{g} : \mathcal{X} \to \mathcal{Y}$ be a prediction functional fit on $\mathcal{D}_0$, then these evaluation measures determine how closely predictions, $\hat{g}(X^*)$, relate to the truth, Y^*, thereby providing a method for determining if a model is 'good'.[3]

Task A machine learning *task* is a simple mechanism to outline the problem of interest by providing: i) the data specification; ii) the definition of learning; iii) the definition of success (when is a prediction 'good'?) [162]. All tasks in this paper have the same definitions of learning and success. For (ii), the aim is to learn the true prediction functional, g, by fitting the learning algorithm on training data, $\hat{g} := \mathcal{A}(\mathcal{D}_0)$. For (iii), a predicted functional is considered 'good' if the *expected generalization error*, $\mathbb{E}[L(Y^*, \hat{g}(X^*))]$, is low, where $(X^*, Y^*) \sim (X, Y)$ is independent of the training data $\mathcal{D}_0$, and L is some loss that is chosen according to the domain of interest (regression, classification, survival).

Resampling Models are *tested* on their ability to make predictions. In order to avoid 'optimism of training error' [145] – overconfidence caused by testing the model on training data – models are tested on previously unseen or 'held-out' data. *Resampling* is the procedure of splitting one dataset into two or more for separated training and testing. In this paper only two resampling methods are utilised: *holdout* and *cross-validation*. Holdout is the process of splitting a

[1] The terms 'fitted' and 'unfitted' are used instead of 'fit' and 'unfit' to prevent confusion with words such as 'suitable' and 'unsuitable'.

[2] The term 'loss' is usually utilised to refer to evaluation measures to be minimised, whereas 'scores' should be maximised, this is returned to in chapter 4.

[3] Here evaluation refers specifically to predictive ability; other forms of evaluation and further discussion of the area are provided in chapter 4.

primary dataset into training data for model fitting and testing data for model predicting. This is an efficient method but may not accurately estimate the expected generalisation error for future model performance, instead this is well-estimated by K-fold cross-validation (KCV) [118]. In KCV, data is split into $K \in \mathbb{N}_{>0}$ 'folds' such that $K-1$ of the folds are used for model training and the final Kth fold for testing. The testing fold is iterated over all K folds, so that each at some point is used for testing and then training (though never at the same time). In each iteration the model is fit on the training folds, and predictions are made and evaluated on the testing fold, giving a loss $L_k := L(\hat{g}(X^k), Y^k)$, where (X^k, Y^k) are data from the kth fold. A final loss is defined by, $L^* := \frac{1}{K}\sum_{k=1}^{K} L_k$. Commonly $K = 5$ or $K = 10$ [32, 166].

Model Performance Benchmarking Whilst *benchmarking* often refers to speed tests, i.e. the time taken to complete an operation, it can also refer to any experiment in which objects (mathematical or computational) are compared. In this report, a benchmark experiment will either refer to the comparison of multiple models' predictive abilities, or comparison of computational speeds and object sizes for model fitting; which of these will be clear from context.

Model Comparison Models can be analytically compared on how well they make predictions for new data. Model comparison is a complex topic with many open questions [67, 69, 223] and as such discussion is limited here. When models are compared on multiple datasets, there is more of a consensus in how to evaluate models [67] and this is expanded on further in chapter 7. Throughout this thesis there are small simulation experiments for model comparison on single datasets however as these are primarily intended to aid exposition and not to generalise results, it suffices to compare models with the conservative method of constructing confidence intervals around the sample mean and standard error of the loss when available [223].

Hyper-Parameters and Tuning A *hyper-parameter* is a model parameter that can be set by the user, as opposed to coefficients that are estimated as part of model fitting. A hyper-parameter can be set before training, or it can be tuned. *Tuning* is the process of choosing the optimal hyper-parameter value via automation. In the simplest setting, tuning is performed by selecting a range of values for the hyper-parameter(s) and treating each choice (combination) as a different model. For example if tuning the number of trees in a random forest (section 3.3), m_r, then a range of values, say $100, 200, 500$ are chosen, and three models $m_{r100}, m_{r200}, m_{r500}$ are benchmarked. The optimal hyper-parameter is given by whichever model is the best performing. *Nested resampling* is a common method to prevent overfitting that could occur from using overlapping data for tuning, training, or testing. Nested resampling is the process of resampling the training set again for tuning.

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 $\hat{g}$.

Any other survival prediction type falls within one of these tasks above, for example predicting log-survival time is the deterministic task and predicting prognostic index or linear predictor is the ranking task. Removing the separation between the prognostic index and ranking prediction types is due to them both making predictions over the Reals; their mathematical difference lies in interpretation only. In general, the survival task will assume that $\mathcal{T} \subseteq \mathbb{R}_{\geq 0}$, and the terms 'discrete' or 'reduced survival task' will refer to the case when $\mathcal{T} \subseteq \mathbb{N}_0$. Unless otherwise specified, the 'survival task', will be used to refer to the probabilistic survival task.[1]

Survival Analysis and Regression Survival and regression tasks are closely related as can be observed from their respective definitions. Both are specified by $g : \mathcal{X} \to \mathcal{S}$ where for probabilistic regression $\mathcal{S} \subseteq \text{Distr}(\mathbb{R})$ and for survival $\mathcal{S} \subseteq \text{Distr}(\mathbb{R}_{\geq 0})$. Furthermore both settings can be viewed to use the same generative process. In the survival setting in which there is no censoring then data is drawn from (X, Y) t.v.i. $\mathcal{X} \times \mathcal{T}, \mathcal{T} \subseteq \mathbb{R}_{\geq 0}$ and in regression from (X, Y) t.v.i. $\mathcal{X} \times \mathcal{Y}, \mathcal{Y} \subseteq \mathbb{R}$, so that the only difference is whether the outcome data ranges over the Reals or positive Reals.

These closely related tasks are discussed in more detail in 5.3, with a particular focus on how the more popular regression setting can be used to solve survival tasks. In chapter 3 the models are first introduced in a regression setting and then the adaptations to survival are discussed, which is natural when considering that historically machine learning survival models have been developed by adapting regression models.

2.6. Summary

This chapter has introduced survival analysis and machine learning, defined each and provided consistent notation and terminology for use throughout this thesis.

The problem of censoring in survival analysis was presented and it was demonstrated why this is a helpful but challenging feature for survival modelling. The primary survival prediction types in the classical setting were identified then separately defined and the scope of this thesis was presented.

The machine learning scope of this thesis is narrowed to the supervised learning setting with a view to make predictions either over the Reals (for rankings), the positive Reals (time to event), or for a full probability distribution over the positive Reals. Key machine learning methods were introduced and will be referred to throughout this paper. Finally classification, regression, and survival tasks were defined and the close connections between regression and survival were identified. These will be exploited fully in chapters 3 and 5.

[1] These definitions are given in the most general case where the time variable is over $\mathbb{R}_{\geq 0}$. In practice, all models instead assume time is over $\mathbb{R}_{>0}$ and any death at $T_i = 0$ is set to $T_i = \epsilon$ for some very small $\epsilon \in \mathbb{R}_{>0}$. Analogously for the discrete survival task. This assumption may not reflect reality as a patient could die at the study start however models cannot typically include this information in training.

The next chapter will use the terminology and notation from this chapter to review and survey both classical and machine learning survival models.

Chapter 3

A Critical Survey of Survival Analysis Models

This chapter provides a brief review of classical survival models and then a critical survey of machine learning survival models. The terms 'classical', 'machine learning', and even 'model' have hazy definitions that will be further specified to make clear how they apply in this paper.

Recall (section 2.4.1) the separation between the following terms:

- Learning strategy – A method for estimating the true prediction functional, g
- Fitting algorithm, $\mathcal{A}$ – A function mapping the training data, $\mathcal{D}_0$, to an estimate of the true prediction functional, $\hat{g} := \mathcal{A}(\mathcal{D}_0)$. The choice of fitting algorithm is determined by the learning strategy.
- (Unfitted) Model – The complete specification of a learning strategy with hyper-parameters and any other components such as pre-processing
- Fitted Model/Prediction functional, $\hat{g} : \mathcal{X} \to \mathcal{Y}$ – Function, possibly with hyper-parameters, for making predictions on unseen data

'Classical' models are defined with a very narrow scope in this thesis: low-complexity models that are either non-parametric or have parameters that can be fit with maximum likelihood estimation (or an equivalent method). In contrast, 'machine learning' (ML) models require more intensive model fitting procedures such as recursion or iteration. The classical models in this paper are fast to fit and highly interpretable, though can be inflexible and may make unreasonable assumptions. Whereas the ML models are more flexible with hyper-parameters however are computationally more intensive (both in terms of speed and storage), require tuning to produce 'good' results, and are often a 'black-box' with difficult interpretation.

This chapter investigates models for predictive survival analysis with a focus on whether a model is APT (section 1.1.1). As classical survival models have been studied extensively for decades, these are separated from the ML models in this chapter and reduced to a smaller literature review in section 3.1. The rest of

this chapter then surveys each of the primary machine learning classes separately. The scope of the models discussed in this chapter is limited to the general thesis scope (section 2.2), i.e. single event with right-censoring and no competing-risks, though in some cases these are discussed.

Novel adaptations for each of the ML models are suggested at the end of each section, these primarily serve as interesting avenues to explore for future research but none have been studied for theoretical properties or implemented in software packages, though most have been informally explored to demonstrate some 'proof-of-concept'.

Notation and Terminology The notation introduced in chapter 2 is recapped for use in this chapter: the generative template for the survival setting is given by (X, T, Δ, Y, C) t.v.i. $\mathcal{X} \times \mathcal{T} \times \{0,1\} \times \mathcal{T} \times \mathcal{T}$ where $\mathcal{X} \subseteq \mathbb{R}^p$ and $\mathcal{T} \subseteq \mathbb{R}_{\geq 0}$, where C, Y are unobservable, $T := \min\{Y, C\}$, and $\Delta = \mathbb{I}(Y = T)$. Random survival data is given by $(X_i, T_i, \Delta_i, Y_i, C_i) \overset{i.i.d.}{\sim} (X, T, \Delta, Y, C)$. Usually data will instead be presented as a training dataset, $\mathcal{D}_0 = \{(X_1, T_1, \Delta_1), ..., (X_n, T_n, \Delta_n)\}$ where $(X_i, T_i, \Delta_i) \overset{i.i.d.}{\sim} (X, T, \Delta)$. For simplicity only a single testing observation needs to be defined to effectively write about the prediction functional, this test observation is given by $\mathcal{D}_1 = (X^*, T^*, \Delta^*) \sim (X, T, \Delta)$.

For regression models the generative template is given by (X, Y) t.v.i. $\mathcal{X} \subseteq \mathbb{R}^p$ and $Y \subseteq \mathbb{R}$. As with the survival setting, a regression training set is given by $\{(X_1, Y_1), ..., (X_n, Y_n)\}$ where $(X_i, Y_i) \overset{i.i.d.}{\sim} (X, Y)$ and a testing observation by $\mathcal{D}_1 = (X^*, Y^*) \sim (X, Y)$.

Finally recall: the set of unique time-points, $\mathcal{U}_O := \{T_i\}_{i \in \{1,...,n\}}$, the set of unique death times, $\mathcal{U}_D := \{T_i : \Delta_i = 1\}_{i \in \{1,...,n\}}$, the risk set at τ is $\mathcal{R}_\tau := \{i : T_i \geq \tau\}$, the number of observations alive or at risk at τ is $n_\tau := \sum_i \mathbb{I}(T_i \geq \tau)$, and the number of observations that die at τ is $d_\tau := \sum_i \mathbb{I}(T_i = \tau, \Delta_i = 1)$.

3.1. A Review of Classical Survival Models

This section provides a literature review of 'classical' models proposed for survival analysis. There are several possible taxonomies for categorising statistical models, these include:

i) Parametrisation Type: One of non-, semi-, or fully-parametric.
Non-parametric models assume that the data distribution cannot be specified with a finite set of parameters. In contrast, fully-parametric models assume the distribution can be specified with a finite set of parameters. Semi-parametric models are a hybrid of the two and are formed of a finite set of parameters *and* an infinite-dimensional 'nuisance' parameter.

ii) Conditionality Type: One of unconditional or conditional. A conditional prediction is one that makes use of covariates in order to condition the prediction on each observation. Unconditional predictors, which are referred to below as 'estimators', ignore covariate data and make the same prediction for all individuals.

iii) Prediction Type: One of ranking, survival time, or distribution (section 2.3).

Table 3 summarises the models discussed below into the taxonomies above for reference. Note that the Cox model is listed as predicting a continuous ranking, and not a survival distribution, which may appear inconsistent with other definitions. The reason for this is elaborated upon in 5.4.1. Though the predict-type taxonomy is favoured throughout this thesis, it is clearer to review classical models in increasing complexity, beginning with unconditional estimators before moving onto semi-parametric continuous ranking predictions, and finally conditional distribution predictors. The review is brief with mathematics limited to the model fundamentals but not including methods for parameter estimation. Also the review is limited to the 'basic' model specification and common extensions such as regularization are not discussed though they do exist for many of these models.

All classical models are highly transparent and accessible, with decades of research and many off-shelf implementations. Predictive performance of each model is briefly discussed as part of the review and then again in chapter 7.

Table 3: Table of models discussed in this literature review, classified by parametrisation, prediction type, and conditionality.

Model[1]	**Parametrisation**[2]	**Prediction**[3]	**Conditionality**
Kaplan-Meier	Non	Distr.	Unconditional
Nelson-Aalen	Non	Distr.	Unconditional
Akritas	Non	Distr.	Conditional
Cox PH	Semi	Rank	Conditional
Parametric PH	Fully	Distr.	Conditional
Accelerated Failure Time	Fully	Distr.	Conditional
Proportional Odds	Fully	Distr.	Conditional
Flexible Spline	Fully	Distr.	Conditional

1. All models are implemented in the R package **survival** [291] with the exception of flexible splines, implemented in **flexsurv** [141], and the Akritas estimator in **survivalmodels** [275].
2. Non = non-parametric, Semi = semi-parametric, Fully = fully-parametric.
3. Distr. = distribution, Rank = ranking.

3.1.1. Non-Parametric Distribution Estimators

Unconditional Estimators Unconditional non-parametric survival models assume no distribution for survival times and estimate the survival function using simple algorithms based on observed outcomes and no covariate data. The two most common methods are the Kaplan-Meier estimator [153], which estimates the average survival function of a training dataset, and the Nelson-Aalen estimator [1, 226], which estimates the average cumulative hazard function of a training dataset.

The Kaplan-Meier estimator of the survival function is given by

$$\hat{S}_{KM}(\tau|\mathcal{D}_0) = \prod_{t \in \mathcal{U}_O, t \leq \tau} \left(1 - \frac{d_t}{n_t}\right) \tag{3.1.1}$$

As this estimate is so important in survival models, this thesis will always use the symbol $\hat{S}_{KM}$ to refer to the Kaplan-Meier estimate of the average survival function fit on training data (T_i, Δ_i). Another valuable function is the Kaplan-Meier estimate of the average survival function of the *censoring* distribution, which is the same as above but estimated on $(T_i, 1 - \Delta_i)$, this will be denoted by $\hat{G}_{KM}$.

The Nelson-Aalen estimator for the cumulative hazard function is given by

$$\hat{H}(\tau|\mathcal{D}_0) = \sum_{t \in \mathcal{U}_O, t \leq \tau} \frac{d_t}{n_t} \tag{3.1.2}$$

The primary advantage of these models is that they rely on heuristics from empirical outcomes only and don't require any assumptions about the form of the data. To train the models they only require (T_i, Δ_i) and both return a prediction of $\mathcal{S} \subseteq \text{Distr}(\mathcal{T})$ (box 3). In addition, both simply account for censoring and can be utilised in fitting other models or to estimate unknown censoring distributions. The Kaplan-Meier and Nelson-Aalen estimators are both consistent estimators for the survival and cumulative hazard functions respectively.

Utilising the relationships provided in section 2.3, one could write the Nelson-Aalen estimator in terms of the survival function as $\hat{S}_{NA} = \exp(-\hat{H}(\tau|\mathcal{D}_0))$. It has been demonstrated that $\hat{S}_{NA}$ and $\hat{S}_{KM}$ are asymptotically equivalent, but that $\hat{S}_{NA}$ will provide larger estimates than $\hat{S}_{KM}$ in smaller samples [57]. In practice, the Kaplan-Meier is the most widely utilised non-parametric estimator in survival analysis and is the simplest estimator that yields consistent estimation of a survival distribution; it is therefore a natural, and commonly utilised, 'baseline' model [19, 123, 134, 317]: estimators that other models should be 'judged' against to ascertain their overall performance (chapter 4).

Not only can these estimators be used for analytical comparison, but they also provide intuitive methods for graphical calibration of models (section 4.5.2). These models are never stuidied for prognosis directly but as baselines, components of complex models (section 5.4.1), or graphical tools [112, 144, 218]. The reason for this is due to them having poor predictive performance as a result of omitting explanatory variables in fitting. Moreover, if the data follows a particular distribution, parametric methods will be more efficient [317].

Conditional Estimators The Kaplan-Meier and Nelson-Aalen estimators are simple to compute and provide good estimates for the survival time distribution but in many cases they may be overly-simplistic. Conditional non-parametric estimators include the advantages described above (no assumptions about underlying data distribution) but also allow for conditioning the estimation on the covariates. This is particularly useful when estimating a censoring distribution that may depend on the data (chapter 4). However predictive performance of conditional non-parametric estimators decreases as the number of covariates increases, and these models are especially poor when censoring is feature-dependent [98].

The most widely used conditional non-parametric estimator for survival anal-

ysis is the Akritas estimator [5] defined by[1]

$$\hat{S}(\tau|X^*, \mathcal{D}_0, \lambda) = \prod_{j:T_j \leq \tau, \Delta_j = 1} \left(1 - \frac{K(X^*, X_j|\lambda)}{\sum_{l=1}^{n} K(X^*, X_l|\lambda)\mathbb{I}(T_l \geq T_j)}\right) \tag{3.1.3}$$

where K is a kernel function, usually $K(x, y|\lambda) = \mathbb{I}(|\hat{F}_X(x) - \hat{F}_X(y)| < \lambda), \lambda \in (0, 1]$, $\hat{F}_X$ is the empirical distribution function of the training data, $X_1, ..., X_n$, and λ is a hyper-parameter. The estimator can be interpreted as a conditional Kaplan-Meier estimator which is computed on a neighbourhood of subjects closest to X^* [24]. To account for tied survival times, the following adaptation of the estimator is utilised [24]

$$\hat{S}(\tau|X^*, \mathcal{D}_0, \lambda) = \prod_{t \in \mathcal{U}_O, t \leq \tau} \left(1 - \frac{\sum_{j=1}^{n} K(X^*, X_j|\lambda)\mathbb{I}(T_j = t, \Delta_j = 1)}{\sum_{j=1}^{n} K(X^*, X_j|\lambda)\mathbb{I}(T_j \geq t)}\right) \tag{3.1.4}$$

If $\lambda = 1$ then $K(\cdot|\lambda) = 1$ and the estimator is identical to the Kaplan-Meier estimator.

The non-parametric nature of the model is highlighted in eq. (3.1.4), in which both the fitting and predicting stages are combined into a single equation. A new observation, X^*, is compared to its nearest neighbours from a training dataset, $\mathcal{D}_0$, without a separated fitting procedure. One could consider splitting fitting and predicting in order to clearly separate between training and testing data. In this case, the fitting procedure is the estimation of $\hat{F}_X$ on training data and the prediction is given by eq. (3.1.4) with $\hat{F}_X$ as an argument. This separated fit/predict method is implemented in **survivalmodels** [275]. As with other non-parametric estimators, the Akritas estimator can still be considered transparent and accessible. With respect to predictive performance, the Akritas estimator has more explanatory power than non-parametric estimators due to conditioning on covariates, however this is limited to a very small number of variables and therefore this estimator is still best placed as a conditional baseline.

3.1.2. Continuous Ranking and Semi-Parametric Models: Cox PH

The Cox Proportional Hazards (CPH) [59], or Cox model, is likely the most widely known semi-parametric model and the most studied survival model [112, 218, 248, 317]. The Cox model assumes that the hazard for a subject is proportionally related to their explanatory variables, $X_1, ..., X_n$, via some baseline hazard that all subjects in a given dataset share ('the PH assumption'). The hazard function

[1] Arguments and parameters are separated in function signatures by a pipe, '|', where variables to the left are parameters (free variables) and those to the right are arguments (fixed). In this equation, τ is a parameter to be set by the user, and $X^*, \mathcal{D}_0, \lambda$ are fixed arguments. This could therefore be simplified to $\hat{S}(\tau)$ to only include free variables.

in the Cox PH model is defined by

$$h(\tau|X_i) = h_0(\tau)\exp(X_i\beta) \tag{3.1.5}$$

where h_0 is the non-negative *baseline hazard function* and $\beta = \beta_1, ..., \beta_p$ where $\beta_i \in \mathbb{R}$ are coefficients to be fit. Note the proportional hazards (PH) assumption can be seen as the estimated hazard, $h(\tau|X_i)$, is directly proportional to the model covariates $\exp(X_i\beta)$. Whilst a form is assumed for the 'risk' component of the model, $\exp(X_i\beta)$, no assumptions are made about the distribution of h_0, hence the model is semi-parametric.

The coefficients, β, are estimated by maximum likelihood estimation of the 'partial likelihood' [60], which only makes use of ordered event times and does not utilise all data available (hence being 'partial'). The partial likelihood allows study of the informative β-parameters whilst ignoring the nuisance h_0. The predicted linear predictor, $\hat{\eta} := X^*\hat{\beta}$, can be computed from the estimated $\hat{\beta}$ to provide a ranking prediction.

Inspection of the model is also useful without specifying the full hazard by interpreting the coefficients as 'hazard ratios'. Let $p = 1$ and $\hat{\beta} \in \mathbb{R}$ and let $X_i, X_j \in \mathbb{R}$ be the covariates of two training observations, then the *hazard ratio* for these observations is the ratio of their hazard functions,

$$\frac{h(\tau|X_i)}{h(\tau|X_j)} = \frac{h_0(\tau)\exp(X_i\hat{\beta})}{h_0(\tau)\exp(X_j\hat{\beta})} = \exp(\hat{\beta})^{X_i - X_j} \tag{3.1.6}$$

If $\exp(\hat{\beta}) = 1$ then $h(\tau|X_i) = h(\tau|X_j)$ and thus the covariate has no effect on the hazard. If $\exp(\hat{\beta}) > 1$ then $X_i > X_j \rightarrow h(\tau|X_i) > h(\tau|X_i)$ and therefore the covariate is positively correlated with the hazard (increases risk of event). Finally if $\exp(\hat{\beta}) < 1$ then $X_i > X_j \rightarrow h(\tau|X_i) < h(\tau|X_i)$ and the covariate is negatively correlated with the hazard (decreases risk of event).

Interpreting hazard ratios is known to be a challenge, especially by clinicians who require simple statistics to communicate to patients [260, 284]. For example the full interpretation of a hazard ratio of '2' for binary covariate X would be: 'assuming that the risk of death is constant at all time-points then the instantaneous risk of death is twice as high in a patient with X than without'. Simple conclusions are limited to stating if patients are at more or less risk than others in their cohort. Further disadvantages of the model also lie in its lack of real-world interpretabilitity, these include [248]: i) the PH assumption may not be realistic and the risk of event may not be constant over time; ii) the estimated baseline hazard from a non-parametric estimator is a discrete step-function resulting in a discrete survival distribution prediction despite time being continuous; and iii) the estimated baseline hazard will be constant after the last observed time-point in the training set [94].

Despite these disadvantages, the model has been demonstrated to have excellent predictive performance and routinely outperforms (or at least does not underperform) sophisticated ML models [95, 204, 306] (and chapter 7). It's simple form and wide popularity mean that it is also highly transparent and accessible.

The next class of models address some of the Cox model disadvantages by

making assumptions about the baseline hazard.

3.1.3. Conditional Distribution Predictions: Parametric Linear Models

Parametric Proportional Hazards The CPH model can be extended to a fully parametric PH model by substituting the unknown baseline hazard, h_0, for a particular parameterisation. Common choices for distributions are Exponential, Weibull and Gompertz [151, 317]; their hazard functions are summarised in table 4 along with the respective parametric PH model. Whilst an Exponential assumption leads to the simplest hazard function, which is constant over time, this is often not realistic in real-world applications. As such the Weibull or Gompertz distributions are often preferred. Moreover, when the shape parameter, γ, is 1 in the Weibull distribution or 0 in the Gompertz distribution, their hazards reduce to a constant risk (fig. 2). As this model is fully parametric, the model parameters can be fit with maximum likelihood estimation, with the likelihood dependent on the chosen distribution.

Table 4: Exponential, Weibull, and Gompertz hazard functions and PH specification.

Distribution[1]	$h_0(\tau)$[2]	$h(\tau\|X_i)$[3]
$\text{Exp}(\lambda)$	λ	$\lambda \exp(X_i\beta)$
$\text{Weibull}(\gamma, \lambda)$	$\lambda\gamma\tau^{\gamma-1}$	$\lambda\gamma\tau^{\gamma-1} \exp(X_i\beta)$
$\text{Gompertz}(\gamma, \lambda)$	$\lambda \exp(\gamma\tau)$	$\lambda \exp(\gamma\tau) \exp(X_i\beta)$

1. Distribution choices for baseline hazard. γ, λ are shape and scale parameters respectively.
2. Baseline hazard function, which is the (unconditional) hazard of the distribution.
3. PH hazard function, $h(\tau|X_i) = h_0(\tau) \exp(X_i\beta)$.

In the literature, the Weibull distribution tends to be favoured as the initial assumption for the survival distribution [95, 112, 124, 244, 246], though Gompertz is often tested in death-outcome models for its foundations in modelling human mortality [104]. There exist many tests for checking the goodness-of-model-fit (section 4.2) and the distribution choice can even be treated as a model hyper-parameter. Moreover it transpires that model inference and predictions are largely insensitive to the choice of distribution [55, 248]. In contrast to the Cox model, fully parametric PH models can predict absolutely continuous survival distributions, they do not treat the baseline hazard as a nuisance, and in general will result in more precise and interpretable predictions if the distribution is correctly specified [248, 256].

Whilst misspecification of the distribution tends not to affect predictions too greatly, PH models will generally perform worse when the PH assumption is not valid. PH models can be extended to include time-varying coefficients or model stratification [59] but even with these adaptations the model may not reflect reality. For example, the predicted hazard in a PH model will be either monotonically increasing or decreasing but there are many scenarios where this is not realistic, such as when recovering from a major operation where risks tends to increase in the short-term before decreasing. Accelerated failure time models overcome this disadvantage and allow more flexible modelling, discussed next.

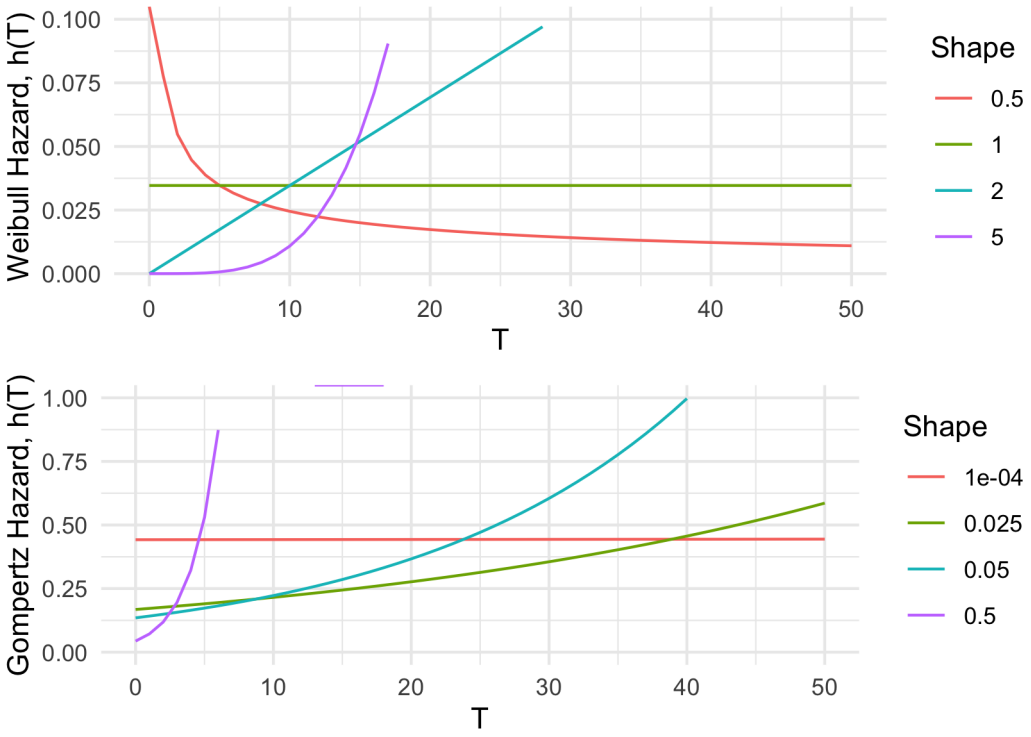


Figure 2: Comparing the hazard curves under Weibull and Gompertz distributions for varying values of the shape parameter; scale parameters are set so that each parametrisation has a median of 20. x-axes are time and y-axes are Weibull (top) and Gompertz (bottom) hazards as a function of time.

Accelerated Failure Time In contrast to the PH assumption, where a unit increase in a covariate is a multiplicative increase in the hazard rate, the Accelerated Failure Time (AFT) assumption means that a unit increase in a covariate results in an acceleration or deceleration towards death (expanded on below). The hazard representation of an AFT model demonstrates how the interpretation of covariates differs from PH models,

$$h(\tau|X_i) = h_0(\exp(-X_i\beta)\tau)\exp(-X_i\beta) \tag{3.1.7}$$

where $\beta = (\beta_1, ..., \beta_p)$ are model coefficients. In contrast to PH models, the 'risk' component, $\exp(-X_i\beta)$, is the exponential of the *negative* linear predictor and therefore an increase in a covariate value results in a decrease of the predicted hazard. This representation also highlights how AFT models are more flexible than PH as the predicted hazard can be non-monotonic. For example the hazard of the Log-logistic distribution (fig. 3) is highly flexible depending on chosen parameters. Not only can the AFT model offer a wider range of shapes for the hazard function but it is more interpretable. Whereas covariates in a PH model act on the hazard, in an AFT they act on time, which is most clearly seen in the log-linear representation,

$$\log Y_i = \mu + \alpha_1 X_{i1} + \alpha_2 X_{i2} + ... + \alpha_p X_{ip} + \sigma\epsilon_i \tag{3.1.8}$$

where μ and σ are location and scale parameters respectively, $\alpha_1, ..., \alpha_p$ are model coefficients, and ϵ_i is a random error term. In this case a one unit increase in covariate X_{ij} means a α_j increase in the logarithmic survival time. For example if $\exp(X_i\alpha) = 0.5$ then i 'ages' at double the baseline 'speed'. Or less abstractly if studying the time until death from cancer then $\exp(X_i\alpha) = 0.5$ can be interpreted as 'the entire process from developing tumours to metastasis and eventual death in subject i is twice as fast than the normal', where 'normal' refers to the baseline when all covariates are 0.

Specifying a particular distribution for ϵ_i yields a fully-parametric AFT model. Common distribution choices include Weibull, Exponential, Log-logistic, and Log-Normal [151, 317]. The Buckley-James estimator [36] is a semi-parametric AFT model that non-parametrically estimates the distribution of the errors however this model has no theoretical justification and is rarely fit in practice [320]. The fully-parametric model has theoretical justifications, natural interpretability, and can often provide a better fit than a PH model, especially when the PH assumption is violated [234, 243, 330].

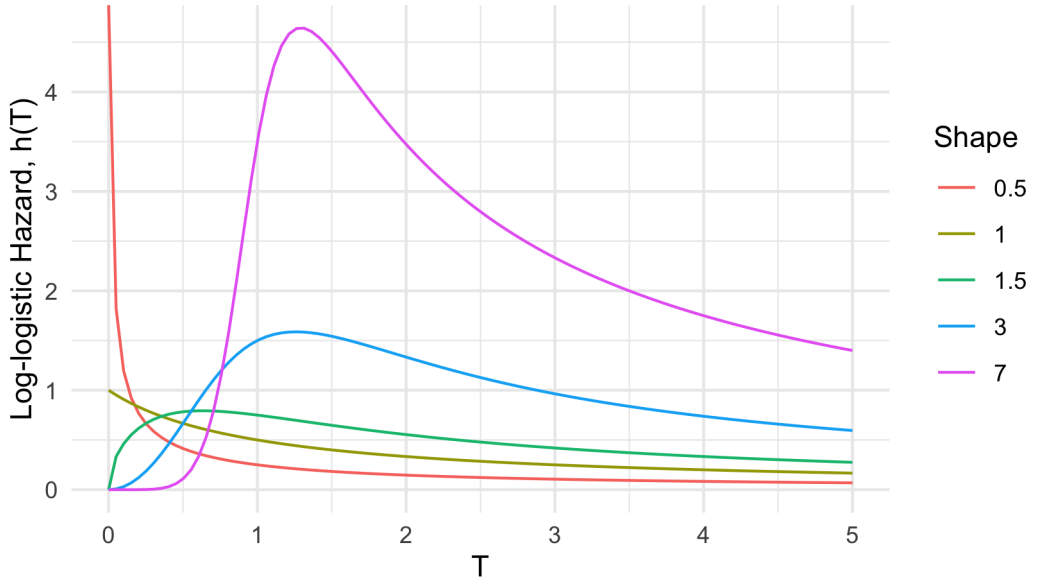


Figure 3: Log-logistic hazard curves with a fixed scale parameter of 1 and a changing shape parameter. x-axis is time and y-axis is the log-logistic hazard as a function of time.

Proportional Odds Proportional odds (PO) models [15] fit a proportional relationship between covariates and the odds of survival beyond a time τ,

$$O_i(\tau) = \frac{S_i(\tau)}{F_i(\tau)} = O_0(\tau)\exp(X_i\beta) \tag{3.1.9}$$

where O_0 is the baseline odds.

In this model, a unit increase in a covariate is a multiplicative increase in the odds of survival after a given time and the model can be interpreted as estimating the log-odds ratio. There is no simple closed form expression for the partial

likelihood of the PO model and hence in practice a Log-logistic distribution is usually assumed for the baseline odds and the model is fit by maximum likelihood estimation on the full likelihood [15].

Perhaps the most useful feature of the model is convergence of hazard functions [163], which states $h_i(\tau)/h_0(\tau) \to 1$ as $\tau \to \infty$. This property accurately reflects real-world scenarios, for example if comparing chemotherapy treatment on advanced cancer survival rates, then it is expected that after a long period (say 10 years) the difference in risk between groups is likely to be negligible. This is in contrast to the PH model that assumes the hazard ratios are constant over time, which is rarely a reflection of reality.

In practice, the PO model is harder to fit and is less flexible than PH and AFT models, both of which can also produce odds ratios. This may be a reason for the lack of popularity of the PO model, in addition there is limited off-shelf implementations [55]. Despite PO models not being commonly utilised, they have formed useful components of neural networks (section 3.6) and flexible parametric models (below).

Flexible Parametric Models – Splines Royston-Parmar flexible parametric models [256] extend PH and PO models by estimating the baseline hazard with natural cubic splines. The model was designed to keep the form of the PH or PO methods but without the semi-parametric problem of estimating a baseline hazard that does not reflect reality (see above), or the parametric problem of misspecifying the survival distribution.

To provide an interpretable, informative and smooth hazard, natural cubic splines are fit in place of the baseline hazard. The crux of the method is to use splines to model time on a log-scale and to either estimate the log cumulative Hazard for PH models, $\log H(\tau|X_i) = \log H_0(\tau) + X_i\beta$, or the log Odds for PO models, $\log O(\tau|X_i) = \log O_0(\tau) + X_i\beta$, where β are model coefficients to fit, H_0 is the baseline cumulative hazard function and O_0 is the baseline odds function. For the flexible PH model, a Weibull distribution is the basis for the baseline distribution and a Log-logistic distribution for the baseline odds in the flexible PO model. $\log H_0(\tau)$ and $\log O_0(\tau)$ are estimated by natural cubic splines with coefficients fit by maximum likelihood estimation. The standard full likelihood is optimised, full details are not provided here. Between one and three internal knots are recommended for the splines and the placement of knots does not greatly impact upon the fitted model [256].

Advantages of the model include being: interpretable, flexible, can be fit with time-dependent covariates, and it returns a continuous function. Moreover many of the parameters, including the number and position of knots, are tunable, although Royston and Parmar advised against tuning and suggest often only one internal knot is required [256]. A recent simulation study demonstrated that even with an increased number of knots (up to seven degrees of freedom), there was little bias in estimation of the survival and hazard functions [29]. Despite its advantages, a 2018 review [229] found only twelve instances of published flexible parametric models since Royston and Parmar's 2002 paper, perhaps because it is more complex to train, has a less intuitive fitting procedure than alternatives, and has limited off-shelf implementations; i.e. is less transparent and accessible than parametric alternatives.

The PH and AFT models are both very transparent and accessible, though require slightly more expert knowledge than the CPH in order to specify the 'correct' underlying probability distribution. Interestingly whilst there are many papers comparing PH and AFT models to one another using in-sample metrics (section 4.2) such as AIC [97, 112, 218, 330], no benchmark experiments could be found for out-of-sample performance. PO and spline models are less transparent than PH and AFT models and are even less accessible, with very few implementations of either. No conclusions can be drawn about the predictive performance of PO or spline models due to a lack of suitable benchmark experiments.

3.2. A Survey of Machine Learning Models for Survival Analysis

These next sections provide a technical, critical survey of machine learning models proposed for survival analysis with the focus on the 'simpler' setup of non-competing risks. Models are separated into their different 'classes' (table 5), which exists as a natural taxonomy in machine learning. Each class review is then further separated by first discussing the simpler and more standard regression setting, before expanding into their survival framework. The focus is once again on the different predict types of the model, which enables clear exposition and discussion around how some areas have successfully dealt with the survival predictive problem, whereas others have fallen short.

This is not the first survey of machine learning models for survival analysis. A recent 2017 survey [317] focused on covering the breadth of machine learning models for survival analysis and this survey is recommended to the reader as a strong starting point to understand which ML models are available for survival analysis. However whilst this provides a comprehensive review and a 'big-picture' view, there is no discussion about how successful the discussed models are in solving the survival task.

A comprehensive survey of neural networks was presented by Schwarzer *et al.* (2000) [268] in which the authors collected the many ways in which neural networks have been 'misused' in the context of survival analysis. This level of criticism is vital in the context of survival analysis and healthcare data as transparency and understanding are often prioritised over predictive performance. Whilst the survey in this thesis will try not to be as critical as the Schwarzer review, it will aim to discuss models and how well they actually solve the survival problem.

In line with the core topic of this thesis, this survey aims to demonstrate if each model is APT (section 1.1.1). Historically, surveys have focused primarily on predictive performance, which is generally preferred for complex classification and regression tasks. However in the context of survival analysis, transparency is of the utmost importance and any model that does not solve the task it claims to, despite strong predictive performance, can be considered sub-optimal. The survey will also examine the accessibility of survival models. A model need not be open-source to be accessible, but it should be 'user-friendly' and not require expert

cross-domain knowledge. For example, a neural network may require knowledge of complex model building, but if set-up correctly could be handled without medical or survival knowledge. Whereas a Gaussian Process requires knowledge of the model class, simulation, (usually) Bayesian modelling, and also survival analysis.

Table 5 provides information about the models reviewed in this survey, including a model reference for use in the chapter 7 benchmark experiment, the predict types of the model, and in which R package it is implemented.

Table 5: Summarising the models discussed in section 3.2 by their model class and respective survival task.

Class[1]	Name (Page)[2]	Authors (Year)[3]	Task[4]	Implementation[5]
RF	RRT (p. 66)	LeBlanc and Crowley (1992) [188]	Rank	**rpart** [292]
RF	RSDF-DEV (p. 66)	Hothorn *et al.* (2004) [130]	Prob.	**ipred** [238]
RF	RRF (p. 66)	Ishwaran *et al.* (2004) [140]	Rank	-
RF	RSCIFF (p. 66)	Hothorn *et al.* (2006) [131]	Det., Prob.	**party** [127], **partykit** [129]
RF	RSDF-STAT (p. 67)	Ishwaran *et al.* (2008) [138]	Prob.	**randomForestSRC** [139], **ranger** [325]
GBM	GBM-COX (p. 72)	Ridgeway (1999) [249] & Buhlmann (2007) [38]	Prob.	**mboost** [132], **xgboost** [45], **gbm** [110]
GBM	CoxBoost (p. 73)	Binder & Schumacher (2008) [19]	Prob.	**CoxBoost** [20]
GBM	GBM-AFT (p. 75)	Schmid & Hothorn (2008) [263]	Det.	**mboost**, **xgboost**
GBM	GBM-BUJAR (p. 76)	Wang & Wang (2010) [319]	Det.	**bujar** [318]
GBM	GBM-GEH (p. 76)	Johnson & Long (2011) [149]	Det.	**mboost**
GBM	GBM-UNO (p. 77)	Mayr & Schmid (2014) [212]	Rank	**mboost**
SVM	SVCR (p. 82)	Shivaswamy *et al.* (2007) [273]	Det.	**survivalsvm** [84]
SVM	SSVM-Rank (p. 84)	Van Belle *et al.* (2007) [303]	Rank	**survivalsvm**
SVM	SVRc (p. 83)	Khan and Zubek (2008) [158]	Det.	-
SVM	SSVM-Hybrid (p. 81)	Van Belle (2011) [306]	Det.	**survivalsvm**
SVM	SSVR-MRL (p. 86)	Goli *et al.* (2016) [102, 103]	Det.	-
ANN	ANN-CDP (p. 94)	Liestøl *et al.* (1994) [197]	Prob.	-

Continued on next page...

Table 5: (continued)

Class[1]	Name (Page)[2]	Authors (Year)[3]	Task[4]	Implementation[5]
ANN	ANN-COX (p. 92)	Faraggi and Simon (1995) [78]	Rank	-
ANN	PLANN (p. 95)	Biganzoli *et al.* (1998) [18]	Prob.	-
ANN	COX-NNET (p. 93)	Ching *et al.* (2018) [49]	Prob.	**cox-nnet** * [48]
ANN	DeepSurv (p. 93)	Katzman *et al.* (2018) [156]	Prob.	**survivalmodels** [275]
ANN	DeepHit (p. 98)	Lee *et al.* (2018) [191]	Prob.	**survivalmodels**
ANN	Nnet-survival (p. 96)	Gensheimer & Narasimhan (2019) [96]	Prob.	**survivalmodels**
ANN	Cox-Time (p. 94)	Kvamme *et al.* (2019) [174]	Prob.	**survivalmodels**
ANN	PC-Hazard (p. 97)	Kvamme & Borgan (2019) [173]	Prob.	**survivalmodels**
ANN	RankDeepSurv (p. 98)	Jing *et al.* (2019) [148]	Det.	**RankDeepSurv** *,† [147]
ANN	DNNSurv (p. 97)	Zhao & Fend (2020) [332]	Prob.	**survivalmodels**

1. Model Class. RSF – Random Survival Forest; GBM – Gradient Boosting Machine; SVM – Support Vector Machine; ANN – Artificial Neural Network. There is some abuse of notation here as some of the RSFs are actually decision trees and some GBMs do not use gradient boosting.

2. Model identifier used in this section and chapter 7.

3. Authors and year of publication, for RSFs this is the paper most attributed to the algorithm.

4. Survival task type: Deterministic (Det.), Probabilistic (Prob.), Ranking (Rank).

5. If available in R then the package in which the model is implemented, otherwise '$*$' signifies a model is only available in Python. With the exception of DNNSurv, all ANNs in **survivalmodels** are implemented from the Python package **pycox** [172] with **reticulate** [301].

† – Code available to create model but not implemented 'off-shelf'.

3.3. Random Forests

3.3.1. Random Forests for Regression

Random forests are a composite algorithm built by fitting many simpler component models, decision trees, and then averaging the results of predictions from these trees. Decision trees are first briefly introduced before the key 'bagging' algorithm that composes these trees to a random forest. Woodland terminology is used throughout this subsection.

Decision Trees Decision trees are a common model class in machine learning and have the advantage of being (relatively) simple to implement and highly interpretable. A decision tree takes a set of inputs and a given *splitting rule* in order to create a series of splits, or branches, in the tree that culminates in a final *leaf*, or *terminal node*. Each terminal node has a corresponding prediction, which for regression is usually the sample mean of the training outcome data. This is made clearer by example, fig. 4 demonstrates a decision tree predicting the miles per gallon (`mpg`) of a car from the `mtcars` [122] dataset. With this tree a new prediction is made by feeding the input variables from the top to the bottom, for example given new data, $x = \{\texttt{wt} = 3, \texttt{disp} = 250\}$, then in the first split the right branch is taken as $\texttt{wt} = 3 > 2.32$ and in the second split the left branch is taken as $\texttt{disp} = 250 \leq 258$, therefore the new data point 'lands' in the final leaf and is predicted to have an `mpg` of 20.8. This value of 20.8 arises as the sample mean of `mpg` for the 11 (which can be seen in the box) observations in the training data who were sorted into this terminal node. Algorithmically, as splits are always binary, predictions are simply a series of conditional logical statements.

Splitting Rules Precisely how the splits are derived and which variables are utilised is determined by the splitting rule.[1] In regression, the most common splitting rule is to select the cut-off for a given variable that minimises the mean squared error in each hypothetical resultant leaf. The goal is to find the variable and cutoff that leads to the greatest difference between the two resultant leaves and thus the maximal homogeneity within each leaf. For all decision tree and random forest algorithms going forward, let L denote some leaf, then let L_{xy}, L_x, L_y respectively be the set of observations, features, and outcomes in leaf L. Let $L_{y;i}$ be the ith outcome in L_y and finally let $L_{\bar{y}} = \frac{1}{n}\sum_{i=1}^{n} L_{y;i}$. To simplify notation, $i \in L$ is taken to be equivalent to $i \in \{i : X_i \in L_X\}$, i.e. the indices of the observations in leaf L.

Let $c \in \mathbb{R}$ be some cutoff parameter and let $L^a_{xy}(j,c) := \{(X_i, Y_i)|X_{ij} < c, i = 1, ..., n\}, L^b_{xy}(j,c) = \{(X_i, Y_i)|X_{ij} \geq c, i = 1, ..., n\}$ be the two leaves containing the set of observations resulting from partitioning variable j at cutoff c. Then a split is determined by finding the arguments, (j^*, c^*) that minimise the sum of

[1] Other methods for growing trees such as pruning are not discussed here as they are less relevant to random forests, which are primarily of interest. Instead see (e.g.) Breiman (1984) [30].

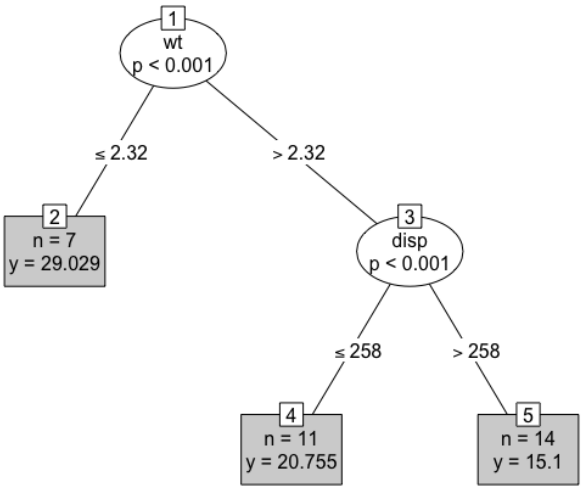


Figure 4: Demonstrating classification trees using the `mtcars` [122] dataset and the **party** [127] package. Ovals are leaves, which indicate the variable that is being split. Edges are branches, which indicate the cut-off at which the variable is split. Rectangles are terminal nodes and include information about the number of training observations in the node and the terminal node prediction.

the mean squared errors (MSE) in both leaves [145],

$$(j^*, c^*) = \underset{j,c}{\operatorname{argmin}} \sum_{y \in L_y^a(j,c)} (y - L_{\bar{Y}}^a(j,c))^2 + \sum_{y \in L_y^b(j,c)} (y - L_{\bar{Y}}^b(j,c))^2 \tag{3.3.1}$$

This method is repeated from the first branch of the tree down to the very last such that observations are included in a given leaf L if they satisfy all conditions from all previous branches; features may be considered multiple times in the growing process. This is an intuitive method as minimising the above sum results in the set of observations within each individual leaf being as similar as possible, thus as an observation is passed down the tree, it becomes more similar to the subsequent leaves, eventually landing in a leaf containing homogeneous observations. Controlling how many variables to consider at each split and how many splits to make are determined by hyper-parameter tuning.

Decision trees are a powerful method for high-dimensional data as only a small sample of variables will be used for growing a tree, and therefore they are also useful for variable importance by identifying which variables were utilised in growth (other importance methods are also available). Decision trees are also highly interpretable, as demonstrated by fig. 4. The recursive pseudo-algorithm in algorithm 1 demonstrates the simplicity in growing a decision tree (again methods such as pruning are omitted).

Stopping Rules The 'stopping rule' in algorithm 1 is usually a condition on the number of observations in each leaf such that leaves will continue to be split until some minimum number of observations has been reached in a leaf. Other

Algorithm 1 Fitting a decision tree.
Input Training data, $\mathcal{D}_0$. Splitting rule, SR.
Output Fitted decision tree, $\hat{g}$.

1: Compute (j^*, c^*) as the optimisers of SR (e.g. eq. (3.3.1)) to create the initial leaf and branches.
2: Repeat step 1 on all subsequent branches until a stopping rule is reached.
3: Return the fitted tree, $\hat{g}$, as the series of branches.

conditions may be on the 'depth' of the tree, which corresponds to the number of levels of splitting, for example the tree in fig. 4 has a depth of 2 (the first level is not counted).

Random Forests Despite being more interpretable than other machine learning methods, decision trees usually have poor predictive performance, high variance and are not robust to changes in the data. As such, *random forests* are preferred to improve prediction accuracy and decrease variance. Random forests utilise bootstrap aggregation, or *bagging* [31], to aggregate many decision trees. A pseudo fitting algorithm is given in algorithm 2.

Algorithm 2 Fitting a random forest.
Input Training data, $\mathcal{D}_0$. Total number of trees, $B \in \mathbb{N}_{>0}$.
Output Fitted random forest, $\hat{g}$.

1: **for** $b = 1, ..., B$ **do**
2: Create a bootstrapped sample of the data, D_b.
3: Grow a decision tree, $\hat{g}_b$, on D_b with algorithm 1.
4: **end for**
5: $\hat{g} \leftarrow \{\hat{g}_b\}_{b=1}^{B}$
6: **return** $\hat{g}$

Prediction from a random forest follows by making predictions from the individual trees and aggregating the results by some function σ (algorithm 3); σ is usually the sample mean for regression,

$$\hat{g}(X^*) = \sigma(\hat{g}_1(X^*), ..., \hat{g}_B(X^*)) = \frac{1}{B}\sum_{b=1}^{B} \hat{g}_b(X^*) \tag{3.3.2}$$

where $\hat{g}_b(X^*)$ is the terminal node prediction from the bth tree and B are the total number of grown trees ('B' is commonly used instead of 'N' to note the relation to bootstrapped data).

Usually many (hundreds or thousands) trees are grown, which makes random forests robust to changes in data and 'confident' about individual predictions. Other advantages include having several tunable hyper-parameters, including: the number of trees to grow, the number of variables to include in a single tree, the splitting rule, and the minimum terminal node size. Machine learning models with many hyper-parameters, tend to perform better than other models as they can be fine-tuned to the data, which is why complex deep learning models are often the best performing. Although as a caveat: too many parameters can lead

Algorithm 3 Predicting from a random forest.
Input Testing data $X^* \sim \mathcal{X}$, fitted forest $\hat{g}$ with $B \in \mathbb{N}_{>0}$ trees, aggregation method σ.
Output Prediction, $\hat{Y} \sim \mathcal{Y}$.

1: **for** $b = 1, ..., B$ **do**
2: 'Drop' X^* down the tree $\hat{g}_b$ individually to return a prediction $\hat{g}_b(X^*)$.
3: **end for**
4: $\hat{Y} \leftarrow \sigma(\hat{g}_1(X^*), ..., \hat{g}_B(X^*))$
5: **return** $\hat{Y}$

to over-fitting and tuning many parameters can take a long time and be highly intensive. Random forests lose the interpretability of decision trees and are considered 'black-box' models as individual predictions cannot be easily scrutinised.

3.3.2. Random Forests for Survival Analysis

Given time constraints and the scope of this thesis, this survey of random forests for survival analysis will primarily focus on 'traditional' decision trees and random forests and will not look at other sub-fields such as causal forests. A comprehensive review of random survival forests (RSFs) is provided in Bou-Hamad (2011) [27], which includes extensions to time-varying covariates and different censoring types. In order to prevent overlap, this survey will focus primarily on methods that have off-shelf implementations, their prediction types, and how successfully these methods handle the problem of censoring. Random forests and decision trees for survival are termed from here as Random Survival Forests (RSFs) and Survival Decision Trees (SDTs) respectively.

Unlike other machine learning methods that may require complex changes to underlying algorithms, individual components of a random forest can be adapted without altering the fundamental algorithm. The principle random forest algorithm is unchanged for RSFs, the difference is in the choice of splitting rule and terminal node prediction, which both must be able to handle censoring. Therefore instead of discussing individual algorithms, the different choices of splitting rules and terminal node predictions are discussed, then combinations of these are summarised into five distinct algorithms.

3.3.2.1. Splitting Rules

Survival trees and RSFs have been studied for the past four decades and whilst the amount of splitting rules to appear could be considered "numerous" [27], only two broad classes are commonly utilised and implemented [139, 235, 292, 325]. The first class rely on hypothesis tests, and primarily the log-rank test, to maximise dissimilarity between splits, the second class utilises likelihood-based measures. The first is discussed in more detail as this is common in practice and is relatively straightforward to implement and understand, moreover it has been demonstrated to outperform other splitting rules [27]. Likelihood rules are more complex and require assumptions that may not be realistic, these are discussed briefly.

Hypothesis Tests The log-rank test statistic has been widely utilised as the 'natural' splitting-rule for survival analysis [52, 138, 189, 269]. The log-rank test compares the survival distributions of two groups and has the null-hypothesis that both groups have the same underlying risk of (immediate) death, i.e. identical hazard functions.

Let L^A and L^B be two leaves then using the notation above let h^A, h^B be the (true) hazard functions derived from the observations in the two leaves respectively. The log-rank hypothesis test is given by $H_0 : h^A = h^B$ with test statistic [269],

$$LR(L^A) = \frac{\sum_{\tau \in \mathcal{U}_D}(d_\tau^A - e_\tau^A)}{\sqrt{\sum_{\tau \in \mathcal{U}_D} v_\tau^A}} \tag{3.3.3}$$

where d_τ^A is the observed number of deaths in leaf A at τ,

$$d_\tau^A := \sum_{i \in L^A} \mathbb{I}(T_i = \tau, \Delta_i = 1) \tag{3.3.4}$$

e_τ^A is the expected number of deaths in leaf A at τ,

$$e_\tau^A := \frac{n_\tau^A d_\tau}{n_\tau} \tag{3.3.5}$$

and v_τ^A is the variance of the number of deaths in leaf A at τ,

$$v_\tau^A := e_\tau^A \Big(\frac{n_\tau - d_\tau}{n_\tau}\Big)\Big(\frac{n_\tau - n_\tau^A}{n_\tau - 1}\Big) \tag{3.3.6}$$

where $\mathcal{U}_D$ is the set of unique death times across the data (in both leaves),
$n_\tau = \sum_i \mathbb{I}(T_i \geq \tau)$ is the number of observations at risk at τ in both leaves,
$n_\tau^A = \sum_{i \in L^A} \mathbb{I}(T_i \geq \tau)$ is the number of observations at risk at τ in leaf A, and
$d_\tau = \sum_i \mathbb{I}(T_i = \tau, \Delta_i = 1)$ is the number of deaths at τ in both leaves.

Intuitively these results follow as the number of deaths in a leaf is distributed according to Hyper$(n_\tau^A, n_\tau, d_\tau)$. The same statistic results if L^B is instead considered. Algorithm 1 follows for fitting decision trees with the log-rank splitting rule, SR, to be maximised.

The higher the log-rank statistic, the greater the dissimilarity between the two groups, thereby making it a sensible splitting rule for survival, moreover it has been shown that it works well for splitting censored data [189].[1] When censoring is highly dependent on the outcome, the log-rank statistic does not perform well and is biased [26], which tends to be true of the majority of survival models. Additionally, the log-rank test requires no knowledge about the shape of the survival curves or distribution of the outcomes in either group [26], making it ideal for an automated process that requires no user intervention.

The log-rank *score* rule [128] is a standardized version of the log-rank rule that could be considered as a splitting rule, though simulation studies have demon-

[1] The results of this experiment are actually in LeBlanc's unpublished 1989 PhD thesis and therefore it has to be assumed that LeBlanc is accurately conveying its results in this 1993 paper.

strated non-significant predictive performance when comparing the two [138].

Alternative dissimiliarity measures and tests have also been suggested as splitting rules, including modified Kolmogorov-Smirnov test and Gehan-Wilcoxon tests [53]. Simulation studies have demonstrated that both of these may have higher power and produce 'better' results than the log-rank statistic [81]. Despite this, these do not appear to be in common usage and no implementation could be found that include these.

Likelihood Based Rules Likelihood ratio statistics, or deviance based splitting rules, assume a certain model form and thereby an assumption about the data. This may be viewed as an advantageous strategy, as it could arguably increase interpretability, or a disadvantage as it places restrictions on the data. For survival models, a full-likelihood can be estimated with a Cox form by estimating the cumulative hazard function [188]. LeBlanc and Crowley (1992) [188] advocate for selecting the optimal split by maximising the full PH likelihood, assuming the cumulative hazard function, H, is known,

$$\mathcal{L} := \prod_{m=1}^{M} \prod_{i \in L^m} h_m(T_i)^{\Delta_i} \exp(-H_m(T_i)) \tag{3.3.7}$$

where M is the total number of terminal nodes, h_m and H_m are the (true) hazard and cumulative hazard functions in the mth node, and again L^m is the set of observations in terminal node m. Estimation of h_m and H_m are described with the associated terminal node prediction below.

The primary advantage of this method is that any off-shelf regression software with a likelihood splitting rule can be utilised without any further adaptation to model fitting by supplying this likelihood with required estimates. However the additional costs of computing these estimates may outweigh the benefits once the likelihood has been calculated, and this could be why only one implementation of this method has been found [27, 292].

Other Splitting Rules As well as likelihood and log-rank spitting rules, other papers have studied comparison of residuals [295], scoring rules [139], and distance metrics [107]. These splitting rules work similarly to the mean squared error in the regression setting, in which the score should be minimised across both leaves. The choice of splitting rule is usually data-dependent and can be treated as a hyper-parameter for tuning. However if there is a clear goal in prediction, then the choice of splitting rule can be informed by the prediction type. For example, if the goal is to maximise separation, then a log-rank splitting rule to maximise homogeneity in terminal nodes is a natural starting point. Whereas if the goal is to estimate the linear predictor of a Cox PH model, then a likelihood splitting rule with a Cox form may be more sensible.

3.3.2.2. Terminal Node Prediction

Only two terminal node predictions appear in common usage.

Predict: Ranking Terminal node ranking predictions for survival trees and forests have been limited to those that use a likelihood-based splitting rule and assume a PH model form [140, 188]. In model fitting the likelihood splitting rule model attempts to fit the (theoretical) PH model $h_m(\tau) = h_0(\tau)\theta_m$ for $m \in 1, ..., M$ where M is the total number of terminal nodes and θ_m is a parameter to estimate. The model returns predictions for $\exp(\hat{\theta}_m)$ where $\hat{\theta}_m$ is the estimate of θ_m. This is estimated via an iterative procedure in which in iteration $j+1$, $\hat{\theta}_m^{j+1}$ is estimated by

$$\hat{\theta}_m^{j+1} = \frac{\sum_{i \in L^m} \Delta_i}{\sum_{i \in L^m} \hat{H}_0^j(T_i)} \tag{3.3.8}$$

where as before L^m is the set of observations in leaf m and

$$\hat{H}_0^j(\tau) = \frac{\sum_{i:T_i \leq \tau} \Delta_i}{\sum_{m=1}^{M} \sum_{\{i:i \in \mathcal{R}_\tau \cap L^a\}} \hat{\theta}_m^j} \tag{3.3.9}$$

which is repeated until some stopping criterion is reached. The same cumulative hazard is estimated for all nodes however $\hat{\theta}_m$ varies across nodes. This method lends itself naturally to a composition to a full distribution (section 5.4.1) as it assumes a PH form and separately estimates the cumulative hazard and relative risk (section 3.3.3), though no implementation of this composition could be found.

Predict: Survival Distribution The most common terminal node prediction appears to be predicting the survival distribution by estimating the survival function, using the Kaplan-Meier or Nelson-Aalen estimators, on the sample in the terminal node [130, 138, 189, 269]. Estimating a survival function by a non-parametric estimator is a natural choice for terminal node prediction as these are natural 'baselines' in survival, similarly to taking the sample mean in regression. The prediction for SDTs is straightforward, the non-parametric estimator is fit on all observations in each of the terminal nodes. This is adapted to RSFs by bagging the estimator across all decision trees [130]. Using the Nelson-Aalen estimator as an example, let m be a terminal node in an SDT, then the terminal node prediction is given by,

$$\hat{H}_m(\tau) = \sum_{\{i:i \in L^m \cap T_i \leq \tau\}} \frac{d_i}{n_i} \tag{3.3.10}$$

where d_i and n_i are the number of events and observations at risk at time T_i in terminal node m. Ishwaran [138] defined the bootstrapped Nelson-Aalen estimator as

$$\hat{H}_{Boot}(\tau) = \frac{1}{B} \sum_{b=1}^{B} \hat{H}_{m,b}(\tau), \quad m \in 1, ..., M \tag{3.3.11}$$

where B is the total number of bootstrapped estimators, M is the number of terminal nodes, and $\hat{H}_{m,b}$ is the cumulative hazard for the mth terminal node in the bth tree. The bootstrapped Kaplan-Meier estimator is calculated analogously. More generally these can be considered as a uniform mixture of B distributions (section 5.4.4).

All implemented RSFs can now be summarised into the following five algorithms:

RRT
LeBlanc and Crowley's (1992) [188] survival decision tree uses a deviance splitting rule with a terminal node ranking prediction, which assumes a PH model form. These 'relative risk trees' (RRTs) are implemented in the package **rpart** [292]. This model is considered the least accessible and transparent of all discussed in this section as: few implementations exist, it requires assumptions that may not be realistic, and predictions are harder to interpret than other models. Predictive performance of the model is expected to be worse than RSFs as this is a decision tree; this is confirmed in chapter 7.

RRF
Ishwaran *et al.* (2004) [140] proposed a random forest framework for the relative risk trees, which makes a slight adaptation and applies the iteration of the terminal node prediction after the tree is grown as opposed to during the growing process. No implementation for these 'relative risk forests' (RRFs) could be found or any usage in the literature. Therefore RRFs are also considered not to be APT for the same reasons given to the RRTs, except that in this case the predictive performance of RRFs is simply unknown (though can reasonably be expected to outperform an RRT).

RSDF-DEV
Hothorn *et al.* (2004) [130] expanded upon the RRT by introducing a bagging composition thus creating a random forest with a deviance splitting rule, again assuming a PH form. However the ranking prediction is altered to be a bootstrapped Kaplan-Meier prediction in the terminal node. This is implemented in **ipred** [238]. This model improves upon the accessibility and transparency of the RRT by providing a more straightforward and interpretable terminal node prediction. However, as this is a decision tree, predictive performance is again expected to be worse than the RSFs.

RSCIFF
Hothorn *et al.* [131] studied a conditional inference framework in order to predict log-survival time. In this case the splitting rule is based on an IPC weighted loss function, which allows implementation by off-shelf classical random forests. The terminal node predictions are a weighted average of the log-survival times in the node where weighting is determined by the Kaplan-Meier estimate of the censoring distribution. This 'random survival conditional inference framework forest' (RSCIFF) is implemented in **party** [127] and **partykit** [129], which additionally includes a distribution terminal node prediction via the bootstrapped Kaplan-Meier estimator. The survival tree analogue (SDCIFT) is implemented in the same packages. Implementation of the RSCIFF is complex, which is likely why all implementations (in the above packages) are by the same authors. The complexity of conditional inference forests may also be the reason why several reviews, including this one, mention (or completely omit) RSCIFFs but do not include any comprehensive details that explain the fitting procedure [27, 316].

In this regard, it is hard to claim that RSCIFFs are transparent or accessible. Moreover the authors of the model state that random conditional inference forests are for "expert user[s] only and [their] current state is rather experimental" [129]. Finally with respect to model performance, there is evidence that they can outperform RSDFs (below) dependent on the data type [225] however no benchmark experiment could be found that compared them to other models.

RSDF-STAT
Finally Ishwaran *et al.* (2008) [138] proposed the most general form of RSFs with a choice of hypothesis tests (log-rank and log-rank score) and survival measure (Brier, concordance) splitting rules, and a bootstrapped Nelson-Aalen terminal node prediction. These are implemented in **randomForestSRC** [139] and **ranger** [325]. This final class of RSFs are likely the only class that can be considered APT. There are several implementations of these models across programming languages, and extensive details for the fitting and predicting procedures, which makes them very accessible. The models utilise a standard random forest framework, which makes them transparent and familiar to those without expert Survival knowledge. Moreover they have been proven to perform well in benchmark experiments, especially on high-dimensional data [123, 283].

3.3.3. Novel Adaptations

Based on this survey of RSFs, a couple of novel adaptations may be considered as natural extensions.

Parametric Terminal Node Predictions All probabilistic RSFs make use of a non-parametric estimator for the terminal node prediction. As an adaptation one could fit a semi- or fully-parametric model in the terminal nodes. However this could suffer from the problem of increased complexity/run-time, as well as overfitting, though is a sensible method worth considering. Alternatively a random forest for inference could be designed whereby a theoretical (say Weibull) survival distribution is assumed and the terminal node predictions are then MLE (or other inference method) estimates for the distribution parameters.

RRT and RRF Composition As discussed above, Ishwaran's Relative Risk Forest makes a relative risk prediction in each terminal node (section 3.3.2.2) by fitting

$$\hat{H}_{h;b}(\tau) = \hat{H}_{0;b}(\tau)\hat{\theta}_h \tag{3.3.12}$$

in which $\hat{H}_{0;b}(\tau)$ and $\hat{\theta}_h$ are iteratively updated and the final prediction is $\hat{\theta}_h$. A natural alternative would be to return the bootstrapped survival distribution prediction over $\hat{H}_{h;b}(\tau)$, instead of only returning $\hat{\theta}$. Ishwaran *et al.* allude to this prediction type in Section 3.2 of the 2004 paper [140], however this is not formalised or implemented. It would be natural to first consider this for RRTs (before extension to RRFs) and implementation would likely be straightforward as any software must first estimate $\hat{H}_{0;b}(\tau)$ and $\hat{\theta}_h$.

3.3.4. Conclusions

Random forests are a highly flexible algorithm that allow the various components to be adapted and altered without major changes to the underlying algorithm. The result is that relatively few R implementations of RSFs cover almost half a century's worth of developments. The only algorithm that does not seem to be implemented is the relative risk forest.

Of the methods reviewed, only one can be considered APT for survival predictions. A lack of accessibility, transparency, or proven performance makes RRT and RSDF-DEV a poor choice for model fitting. RSCIFF is potentially a powerful method with promising results in benchmark experiments, but even the authors recognise its complexity prevents it from being accessible. Ishwaran's RSFs on the other hand are APT and suitable for model fitting and deployment. Simulation studies have demonstrated that RSFs can perform well even with high levels of censoring and there is evidence that on some datasets these can outperform a Cox PH [138]. Despite only one of the five models discussed here being APT, Ishwaran's model is highly flexible, and its implementation in software packages reflects this. Therefore one can still confidently conclude that random forests are a powerful algorithm in regression, classification, and survival analysis.

3.4. Gradient Boosting Machines

3.4.1. Gradient Boosting Machines for Regression

Boosting is a machine learning strategy that can be applied to any model class. Similarly to random forests, boosting is an 'ensemble' method that creates a model from a 'committee' of learners. The committee is formed of 'weak' learners that make poor predictions individually, which creates a 'slow learning' approach (as opposed to 'greedy') that requires many iterations for a model to be a good fit to the data. Boosting models are similar to random forests in that both make predictions from a large committee of learners. However the two differ in how this committee is combined to a prediction. In random forest algorithms, each decision tree is grown independently and their predictions are combined by a simple mean calculation. In contrast, weak learners in a boosting model are fit sequentially and predictions are made by a linear combination of predictions from each learner. With respect to transparency, it is simpler to inspect 100 trees in a random forest, than it is to inspect 100 weak learners in a boosted model, though both are considered black-box models.

The best known boosting algorithm is likely AdaBoost [85], which is more generally a Forward Stagewise Additive Model (FSAM) with an exponential loss [118]. Today, the most widely used boosting model is the Gradient Boosting Machine (GBM) [88].

Training a GBM Pseudo-code for training a componentwise GBM is presented in algorithm 4. The term 'componentwise' is explained fully below, only this variation of GBM is presented as it is the most common in implementation [110,

132]. Line 1: the initial function is initialized as $g_0 = 0$;[1] Line 2: iterate over boosting steps $m = 1, ..., M$ and; Line 3: randomly sample the training data, $\mathcal{D}_0$, to a smaller sample, $\mathcal{D}_0^*$, this may be ignored if $\phi = 1$; Line 4: for all training observations in the reduced dataset, $i \in \{i : X_i \in \mathcal{D}_0^*\}$, compute the negative gradient, r_{im}, of the differentiable loss function, L, with respect to predictions from the previous iteration, $g_{m-1}(X_i)$; Line 5: fit one weak learner for each feature, $j = 1, ..., p$, in the training data, where the feature, $X_{:j}$, is the single covariate and r_{im} are the labels; Line 6: select the optimal weak learner as the one that minimises the squared error between the prediction and the true gradient; Line 7: update the fitted model by adding the optimal weak learner with a shrinkage penalty, ν; Line 9: return the model updated in the final iteration as the fitted GBM.

Algorithm 4 Training a componentwise Gradient Boosting Machine.

Input Training data, $\mathcal{D}_0 = \{(X_1, Y_1), ..., (X_n, Y_n)\}$, where $(X_i, Y_i) \overset{i.i.d.}{\sim} (X, Y)$. Differentiable loss, L. Hyper-parameters: sampling fraction, $\phi \in (0, 1]$; step-size, $\nu \in (0, 1]$; number of iterations, $M \in \mathbb{R}_{>0}$.

Output Boosted model, $\hat{g}$.

1: Initialize $g_0 \leftarrow 0$
2: **for** $m = 1, ..., M$ **do**
3: $\quad \mathcal{D}_0^* \leftarrow$ Randomly sample $\mathcal{D}_0$ w.p. ϕ
4: $\quad r_{im} \leftarrow -[\frac{\partial L(y_i, g_{m-1}(X_i))}{\partial g_{m-1}(X_i)}], i \in \{i : X_i \in \mathcal{D}_0^*\}$
5: $\quad$ Fit p weak learners, w_j to $(X_i, r_{im}), j = 1, .., p$
6: $\quad j^* \leftarrow \text{argmin}_{j=1,..,p} \sum_{i \in \{i: X_i \in \mathcal{D}_0^*\}} (r_{im} - w_j(X_i))^2$
7: $\quad g_m \leftarrow g_{m-1} + \nu w_{j^*}$
8: **end for**
9: $\hat{g} \leftarrow g_M$
10: **return** $\hat{g}$

Predicting with a GBM In general, predictions from a trained GBM are simple to compute as the fitted model (and all individual weak learners) take the same inputs, which are passed sequentially to each of the weak learners. In algorithm 4, the fitted GBM is a single model, which is a linear combination of weak learners. Instead one could think of the returned model as a collection of the optimal weak learners, i.e. let $w_{m;j^*}$ be the optimal weak learner from iteration m and let the fitted GBM (Line 9 algorithm 4) be $\hat{g} := \{w_{m;j^*}\}_{m=1}^M$.[2] With this formulation, making predictions from the GBM can be demonstrated simply in algorithm 5.

[1] Some algorithms may instead initialize g_0 by finding the value that minimises the given loss function, however setting $g_0 = 0$ appears to be the most common practice for componentwise GBMs.

[2] This formulation is computationally and mathematically identical to the formulation in algorithm 4 and is practically more convenient for implementation, indeed this is the implementation in **mboost** [132]. Despite this, the formulation in algorithm 4 is common in the literature, which often conflates model training and predicting.

Algorithm 5 Predicting from a Gradient Boosting Machine.
Input Fitted GBM, $\hat{g} := \{w_{m;j^*}\}_{m=1}^{M}$, trained with step-size ν. Testing data $X^* \sim \mathcal{X}$.
Output Prediction, $\hat{Y} \sim \mathcal{Y}$.

1: Initialize $\hat{Y} = 0$
2: **for** $m = 1, ..., M$ **do**
3: $\quad \hat{Y} \leftarrow \hat{Y} + \nu w_{m;j^*}(X^*)$
4: **end for**
5: **return** $\hat{Y}$

The biggest advantages of boosting are firstly relatively few hyper-parameters, which all have a meaningful and intuitive interpretation, and secondly its modular nature means that, like random forests, relatively few parts need to be updated to derive a novel model. First the model components will be discussed and then the hyper-parameters. Once this has been established, deriving survival variants can be simply presented.

3.4.1.1. Losses and Learners

Losses Building a GBM requires selection of the loss to minimise, L, selection of weak learners, w_j, and a method to compare the weak learners to the loss gradient. The only constraint in selecting a loss, L, is that it must be differentiable w.r.t. $g(X)$ [118]. Of course a sensible loss should be chosen (a classification loss should not be used for regression) and different choices of losses will optimise different tasks. L_2-losses have been demonstrated to be effective for regression boosting, especially with high-dimensional data [39]; this is referred to as L_2-boosting.

Weak Learners Algorithm 4 is specifically a *componentwise* GBM [39], which means that each of the p weak learners is fit on a single covariate from the data. This method simplifies selecting the possible choices for the weak learners to selecting the class of weak learner (below). Additionally, componentwise GBMs provide a natural and interpretable feature selection method as selecting the optimal learner (algorithm 4, line 6) corresponds to selecting the feature that minimises the chosen loss in iteration m.

Only three weak, or 'base', learner classes are commonly used in componentwise GBMs [132, 319]. These are linear least squares [88], smoothing splines [39], and decision stumps [39, 88]. Let L be a loss with negative gradient for observation i in the mth iteration, r_{im}, and let $\mathcal{D}_0$ be the usual training data. For linear least squares, an individual weak learner is fit by [88, 319],

$$w_j(\mathcal{D}_0) = X_{;j} \frac{\sum_{i=1}^{n} X_{ij} r_{im}}{\sum_{i=1}^{n} (X_{ij})^2} \tag{3.4.1}$$

For smoothing splines, usually cubic splines are implemented, these fit weak learners as the minimisers of the equation [39],

$$w_j := \underset{g \in \mathcal{G}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} (r_{im} - g(X_{ij}))^2 + \lambda \int (g''(u))^2 du \tag{3.4.2}$$

where g'' is the second derivative of g, $\mathcal{G}$ is the set of functions, $\mathcal{G} := \{g : g \text{ is twice continuously differentiable and } \int (g''(x))^2 dx < \infty\}$, and λ is a hyper-parameter usually chosen so that the number of degrees of freedom, df, is small, with df ≈ 4 suggested [39, 264, 319].

Finally for decision stumps (fig. 5), a decision tree, w_j, is grown (algorithm 1) on $(X_{;j}, r_m)$ to depth one (equivalently to two terminal nodes) for each of the $j = 1, ..., p$ covariates [88].

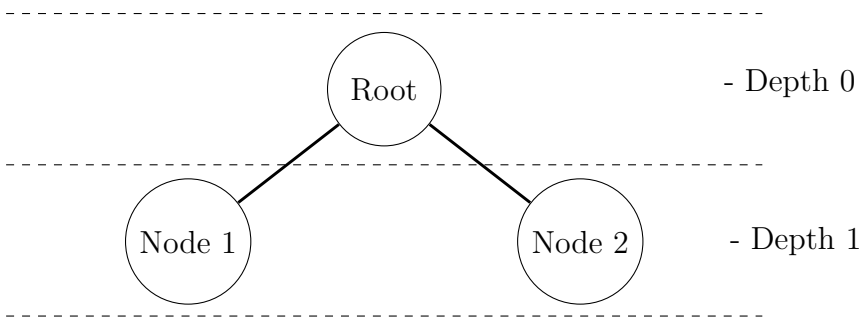


Figure 5: A decision tree of depth one, known as a decision stump. The root layer is separated at depth 0 from the terminal nodes at depth 1. A decision stump is defined by a decision tree with a single split at the root node.

3.4.1.2. Hyper-Parameters

The hyper-parameters in 4 are the 'step-size', ν, the sampling fraction, ϕ, and the number of iterations, M.

Number of iterations, M The number of iterations is often claimed to be the most important hyper-parameter in GBMs and it has been demonstrated that as the number of iterations increases, so too does the model performance (with respect to a given loss on test data) up to a certain point of overfitting [37, 118, 264]. This is an intuitive result as the foundation of boosting rests on the idea that weak learners can slowly be combined to form a single powerful model. This is especially true in componentwise GBMs as time is required to learn which features are important. Finding the optimal value of M is critical as a value too small will result in poor predictions, whilst a value too large will result in model overfitting. Two primary methods have been suggested for finding the optimal value of M. The first is to find the $M \in \mathbb{N}_{>0}$ that minimises a given measure based on the AIC [4], the second is the 'usual' empirical selection by nested cross-validation. In practice the latter method is usually employed.

Step-size, ν The step-size parameter (algorithm 4, line 7), ν, is a shrinkage parameter that controls the contribution of each weak learner at each iteration. Several studies have demonstrated that GBMs perform better when shrinkage is applied and a value of $\nu = 0.1$ is often suggested [38, 118, 88, 192, 264]. The optimal values of ν and M depend on each other, such that smaller values of ν require larger values of M, and vice versa. This is intuitive as smaller ν results in a slower learning algorithm and therefore more iterations are required to fit the model. Accurately selecting the M parameter is generally considered to be

of more importance, and therefore a value of ν is often chosen heuristically (e.g. the common value of 0.1) and then M is tuned by cross-validation and/or early-stopping.

Sampling Fraction, ϕ Motivated by the success of bagging in random forests, stochastic gradient boosting [86] randomly samples the data in each iteration. It appears that subsampling performs best when also combined with shrinkage [118] and as with the other hyper-parameters, selection of ϕ is usually performed by nested cross-validation.

3.4.2. Gradient Boosting Machines for Survival Analysis

In a componentwise GBM framework, adapting boosting to survival analysis requires only selecting a sensible choice of loss function L. Therefore fitting and predicting algorithms for componentwise survival GBMs are not discussed as these are fully described in algorithms 4 and 5 respectively. However, some GBMs in this section are not componentwise and therefore require some more detailed consideration. Interestingly, unlike other machine learning algorithms that historically ignored survival analysis, early GBM papers considered boosting in a survival context [249]; though there appears to be a decade gap before further considerations were made in the survival setting. After that period, several developments by Binder, Schmid, and Hothorn, adapted componentwise GBMs to a framework suitable for survival analysis. Their developments are covered exhaustively in the R packages **gbm** [110] and **mboost** [132]. This survey continues with the predict type taxonomy.

3.4.2.1. Cox Survival Models

All survival GBMs make ranking predictions and none are able to directly predict survival distributions. However, the GBMs discussed in this section all have natural compositions to distributions as they are modelled in the semi-parametric proportional hazards framework (section 5.4.1). The models discussed in the next section can also be composed to distributions though the choice of composition is less clear and therefore they are listed as pure 'ranking' models.

GBM-COX
The 'GBM-COX' aims to predict the distribution of data following the PH assumption by estimating the coefficients of a Cox model in a boosting framework [249]. The model attempts to predict $\hat{g}(X^*) = \hat{\eta} := X^*\hat{\beta}$, by minimising a suitable loss function. As the model assumes a PH specification, the natural loss to optimise is the Cox partial likelihood [59, 60], more specifically to minimise the negative partial log-likelihood, $-l$, where

$$l(\beta) = \sum_{i=1}^{n} \Delta_i \Big[\eta_i \; - \; \log \Big(\sum_{j \in \mathcal{R}_{t_i}}^{n} \exp(\eta_i) \Big) \Big] \tag{3.4.3}$$

where $\mathcal{R}_{t_i}$ is the set of patients at risk at time t_i and $\eta_i = X_i\beta$. The gradient of $-l(\beta)$ at iteration m is

$$r_{im} := \Delta_i - \sum_{j=1}^{n} \Delta_j \frac{\mathbb{I}(T_i \geq T_j)\exp(g_{m-1}(X_i))}{\sum_{k\in\mathcal{R}_{t_j}} \exp(g_{m-1}(X_k))} \tag{3.4.4}$$

where $g_{m-1}(X_i) = X_i\beta_{m-1}$.

Algorithm 4 now follows with the loss $L := -l(\beta)$.[1]

The GBM-COX is implemented in **mboost** [132] and has been demonstrated to perform well even when the data violates the PH assumption [149]. Despite being a black-box, GBMs are well-understood and individual weak learners are highly interpretable, thus making GBMs highly transparent. Several well-established software packages implement GBM-COX and those that do not tend to be very flexible with respect to custom implementations. GBM-COX is therefore considered an APT survival model.

CoxBoost

The CoxBoost algorithm boosts the Cox PH by optimising the penalized partial-log likelihood; additionally the algorithm allows for mandatory (or 'forced') covariates [19]. In medical domains the inclusion of mandatory covariates may be essential, either for model interpretability, or due to prior expert knowledge. This is not a feature usually supported by boosting. CoxBoost deviates from algorithm 4 by instead using an offset-based approach for generalized linear models [297]. This model has a non-componentwise and componentwise framework but only the latter is implemented by the authors [20] and discussed here. Let $\mathcal{I}_{mand}$ be the indices of the mandatory covariates to be included in all iterations, $m = 1, ..., M$, then for an iteration m the indices to consider for fitting are the set

$$I_m = \{\{1\} \cup \mathcal{I}_{mand}, ..., \{p\} \cup \mathcal{I}_{mand}\}/\{\{j\} \cup \mathcal{I}_{mand} : j \in \mathcal{I}_{mand}\} \tag{3.4.5}$$

i.e. in each iteration the algorithm fits a weak learner on the mandatory covariates and one additional (non-mandatory) covariate (hence still being componentwise).

In addition, a penalty matrix $\mathbf{P} \in \mathbb{R}^{p\times p}$ is considered such that $P_{ii} > 0$ implies that covariate i is penalized and $P_{ii} = 0$ means no penalization. In practice this is usually a diagonal matrix [19] and by setting $P_{ii} = 0, i \in I_{mand}$ and $P_{ii} > 0, i \notin I_{mand}$, only optional (non-mandatory) covariates are penalized. The penalty matrix can be allowed to vary with each iteration, which allows for a highly flexible approach, however in implementation a simpler approach is to either select a single penalty to be applied in each iteration step or to have a single penalty matrix [20].

At the mth iteration and the kth set of indices to consider ($k = 1, ..., p$), the

[1] Early implementations and publications of the GBM algorithm [86, 88] included an additional step to the algorithm in which a step size is estimated by line search. More recent research has determined that this additional step is unneccesary [38] and the line search method does not appear to be used in practice.

loss to optimize is the penalized partial-log likelihood given by

$$l_{pen}(\gamma_{mk}) = \sum_{i=1}^{n} \Delta_i \Big[\eta_{i,m-1} + X_{i,\mathcal{J}_{mk}} \gamma_{mk}^T\Big] - \\ \Delta_i \log\Big(\sum_{j=1}^{n} \mathbb{I}(T_j \leq T_i) \exp(\eta_{i,m-1} + X_{i,\mathcal{J}_{mk}} \gamma_{mk}^T\Big) - \lambda \gamma_{mk} \mathbf{P}_{mk} \gamma_{mk}^T \tag{3.4.6}$$

where $\eta_{i,m} = X_i \beta_m$, γ_{mk} are the coefficients corresponding to the covariates in $\mathcal{J}_{mk}$ which is the possible set of candidates for a subset of total candidates $k = 1, ..., p$, $\mathbf{P}_{mk}$ is the penalty matrix, and λ is a penalty hyper-parameter to be tuned or selected.[1]

In each iteration, all potential candidate sets (the union of mandatory covariates and one other covariate) are updated by

$$\hat{\gamma}_{mk} = \mathbf{I}_{pen}^{-1}(0) U(0) \tag{3.4.7}$$

where $U(\gamma) = \partial l / \partial \gamma(\gamma)$ and $\mathbf{I}_{pen}^{-1} = \partial^2 l / \partial \gamma \partial \gamma^T (\gamma + \lambda \mathbf{P}_{mk})$ are the first and second derivatives of the unpenalized partial-log-likelihood. The optimal set is then found as

$$k^* := \underset{k}{\operatorname{argmax}}\, l_{pen}(\gamma_{mk}) \tag{3.4.8}$$

and the estimated coefficients are updated with

$$\hat{\beta}_m = \hat{\beta}_{m-1} + \gamma_{mk^*}, \quad k^* \in \mathcal{J}_{mk} \tag{3.4.9}$$

The step size, ν, is then one, but this could potentially be altered.

The algorithm deviates from algorithm 4 as l_{pen} is directly optimised and not its gradient, additionally model coefficients are iteratively updated instead of a more general model form. The algorithm is implemented in **CoxBoost** [20]. Experiments suggest that including the 'correct' mandatory covariates may increase predictive performance [19]. CoxBoost is less accessible than other boosting methods as it requires a unique boosting algorithm, as such only one off-shelf implementation appears to exist and even this implementation has been removed from CRAN as of 2020-11-11. CoxBoost is also less transparent as the underlying algorithm is more complex, though is well-explained by the authors [19]. There is good indication that CoxBoost is performant, which is seen in chapter 7. In a non-medical domain, where performance may be the most important metric, then perhaps CoxBoost can be recommended as a powerful model. However, when sensitive predictions are required, CoxBoost is currently not APT. Further papers studying the model and more off-shelf implementations could change this in the future.

[1] On notation, note that $\mathbf{P}_{ij}$ refers to the penalty matrix in the ith iteration for the jth set of indices, whereas P_{ij} is the (i, j)th element in the matrix $\mathbf{P}$.

3.4.2.2. Ranking Survival Models

The ranking survival models in this section are all unified as they make predictions of the linear predictor, $\hat{g}(X^*) = X^*\hat{\beta}$.[1]

GBM-AFT

Schmid and Hothorn (2008) [263] published a GBM for accelerated failure time models in response to PH-boosted models that may not be suitable for non-PH data. Their model fits into the GBM framework by assuming a fully-parametric AFT and simultaneously estimating the linear predictor, $\hat{g}(X_i) = \hat{\eta}$, and the scale parameter, $\hat{\sigma}$, controlling the amount of noise in the distribution. The (fully-parametric) AFT is defined by

$$\log Y = \eta + \sigma W \tag{3.4.10}$$

where W is a random variable independent of the covariates that follows a given distribution and controls the noise in the model. By assuming a distribution on W, a distribution is assumed for the full parametric model. The full likelihood, $\mathcal{L}$, is given by

$$\mathcal{L}(\mathcal{D}_0|\mu,\sigma,W) = \prod_{i=1}^{n}\Big[\frac{1}{\sigma}f_W\Big(\frac{\log(T_i)-\mu}{\sigma}\Big)\Big]^{\Delta_i}\Big[S_W\Big(\frac{\log(T_i)-\mu}{\sigma}\Big)\Big]^{(1-\Delta_i)} \tag{3.4.11}$$

where f_W, S_W is the pdf and survival function of W for a given distribution. By setting $\mu := g(X_i)$, σ is then rescaled according to known results depending on the distribution [164]. The gradient of the negative log-likelihood, $-l$, is minimised in the mth iteration where

$$\begin{aligned} l(\mathcal{D}_0|\hat{g},\hat{\sigma},W) = \sum_{i=1}^{n}\Delta_i\Big[-\log\sigma + \log f_W\Big(\frac{\log(T_i)-\hat{g}_{m-1}(X_i)}{\hat{\sigma}_{m-1}}\Big)\Big] + \\ (1-\Delta_i)\Big[\log S_W\Big(\frac{\log(T_i)-\hat{g}_{m-1}(X_i)}{\hat{\sigma}_{m-1}}\Big)\Big] \end{aligned} \tag{3.4.12}$$

where $\hat{g}_{m-1}, \hat{\sigma}_{m-1}$ are the location-scale parameters estimated in the previous iteration. Note this key difference to other GBM methods in which two estimates are made in each iteration step. In order to allow for this, algorithm 4 is run as normal but in addition, after updating $\hat{g}_m$, one then updates $\hat{\sigma}_m$ as

$$\hat{\sigma}_m := \underset{\sigma}{\operatorname{argmin}} -l(\mathcal{D}_0|g_m,\sigma,W) \tag{3.4.13}$$

σ_0 is initialized at the start of the algorithm with $\sigma_0 = 1$ suggested [263].

This algorithm provides a ranking prediction without enforcing an often-unrealistic PH assumption on the data. This model is implemented in **mboost** and **xgboost**.

[1] This is commonly referred to as a 'linear predictor' as it directly relates to the boosted linear model (e.g. Cox PH), however it is more accurately a 'prognostic index' as the final prediction is not the true linear predictor.

Experiments indicate that this may outperform the Cox PH [263]. Moreover the model has the same transparency and accessibility as the GBM-COX and is therefore also considered APT.

GBM-GEH

The concordance index is likely the most popular measure of discrimination, this in part due to the fact that it makes little-to-no assumptions about the data (section 4.4). A less common measure is the Gehan loss, motivated by the semi-parametric AFT. Johnson and Long proposed the GBM with Gehan loss, here termed GBM-GEH, to optimise separation within an AFT framework [149].

The semi-parametric AFT is defined by the linear model,

$$\log Y = \eta + \epsilon \tag{3.4.14}$$

for some error term, ϵ.

The D-dimensional Gehan loss to minimise is given by,

$$G_D(\mathcal{D}_0, \hat{g}) = -\frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}\Delta_i(\hat{e}_i - \hat{e}_j)\mathbb{I}(\hat{e}_i \leq \hat{e}_j) \tag{3.4.15}$$

where $\hat{e}_i = \log T_i - \hat{g}(X_i)$. The negative gradient of the loss is,

$$r_{im} := \frac{\sum_{j=1}^{n}\Delta_j\mathbb{I}(\hat{e}_{m-1,i} \geq \hat{e}_{m-1,j}) - \Delta_i\mathbb{I}(\hat{e}_{m-1,i} \leq \hat{e}_{m-1,j})}{n} \tag{3.4.16}$$

where $\hat{e}_{m-1,i} = \log T_i - \hat{g}_{m-1}(X_i)$.

Algorithm 4 then follows naturally substituting the loss and gradient above. The algorithm is implemented in **mboost**. Simulation studies on the performance of the model are inconclusive [149] however the results in chapter 7 indicate strong predictive performance. Therefore this can tentatively be considered APT but further benchmark experiments would be preferred.

GBM-BUJAR

GBM-BUJAR is another boosted semi-parametric AFT. However the algorithm introduced by Wang and Wang (2010) [319] uses Buckley-James imputation and minimisation. This algorithm is almost identical to a regression GBM (i.e. using squared loss or similar for L), except with one additional step to iteratively impute censored survival times. Assuming a semi-parametric AFT model, the GBM-BUJAR algorithm iteratively updates imputed outcomes with the Buckley-James estimator [36],

$$T^*_{m,i} := \hat{g}_{m-1}(X_i) + e_{m-1,i}\Delta_i + (1-\Delta_i)\Big[\hat{S}_{KM}(e_{m-1,i})^{-1}\sum_{e_{m-1,j} > e_{m-1,i}} e_{m-1,j}\Delta_j\hat{p}_{KM}(e_{m-1,j})\Big] \tag{3.4.17}$$

where $\hat{g}_{m-1}(X_i) = \hat{\eta}_{m-1}$, and $\hat{S}_{KM}, \hat{p}_{KM}$ are Kaplan-Meier estimates of the survival and probability mass functions respectively fit on some training data, and $e_{m-1,i} := \log(T_i) - g_{m-1}(X_i)$. Once $T^*_{m,i}$ has been updated, algorithm 4 continues

from with least squares as with any regression model.

GBM-BUJAR is implemented in **bujar** [318] though without a separated fit/predict interface, its accessibility is therefore limited. There is no evidence of wide usage of this algorithm nor simulation studies demonstrating its predictive ability. Finally, there are many known problems with semi-parametric AFT models and the Buckey-James procedure [320], hence GBM-BUJAR is also not transparent.

GBM-UNO

Instead of optimising models based on a given model form, Chen *et al.* [47] studied direct optimisation of discrimination by Harrell's C whereas Mayr and Schmid [212] focused instead on Uno's C. Only an implementation of the Uno's C method could be found, this is therefore discussed here and termed 'GBM-UNO'.

The GBM-UNO attempts to predict $\hat{g}(X^*) := \hat{\eta}$ by optimising Uno's C (section 4.4.1),

$$C_U(\hat{g}, \mathcal{D}_0) = \frac{\sum_{i \neq j} \Delta_i \{\hat{G}_{KM}(T_i)\}^{-2} \mathbb{I}(T_i < T_j) \mathbb{I}(\hat{g}(X_i) > \hat{g}(X_j))}{\sum_{i \neq j} \Delta_i \{\hat{G}_{KM}(T_i)\}^{-2} \mathbb{I}(T_i < T_j)} \tag{3.4.18}$$

The GBM algorithm requires that the chosen loss, here C_U, be differentiable w.r.t. $\hat{g}(X)$, which is not the case here due to the indicator term, $\mathbb{I}(\hat{g}(X_i) > \hat{g}(X_j))$. Therefore a smoothed version is instead considered where the indicator is approximated by the sigmoid function [205],

$$K(u|\sigma) = (1 + \exp(-u/\sigma))^{-1} \tag{3.4.19}$$

where σ is a hyper-parameter controlling the smoothness of the approximation. The measure to optimise is then,

$$C_{USmooth}(\mathcal{D}_0|\sigma) = \sum_{i \neq j} \frac{k_{ij}}{1 + \exp\left[(\hat{g}(X_j) - \hat{g}(X_i))/\sigma\right]} \tag{3.4.20}$$

with

$$k_{ij} = \frac{\Delta_i (\hat{G}_{KM}(T_i))^{-2} \mathbb{I}(T_i < T_j)}{\sum_{i \neq j}^{n} \Delta_i (\hat{G}_{KM}(T_i))^{-2} \mathbb{I}(T_i < T_j)} \tag{3.4.21}$$

The negative gradient at iteration m for observation i can then be found,

$$r_{im} := -\sum_{j=1}^{n} k_{ij} \frac{-\exp(\frac{\hat{g}_{m-1}(X_j) - \hat{g}_{m-1}(X_i)}{\sigma})}{\sigma(1 + \exp(\frac{\hat{g}_{m-1}(X_j) - \hat{g}_{m-1}(X_i)}{\sigma}))} \tag{3.4.22}$$

Algorithm 4 can then be followed exactly by substituting this loss and gradient; this is implemented in **mboost**. One disadvantage of GBM-UNO is that C-index boosting is more insensitive to overfitting than other methods [211], therefore stability selection [214] can be considered for variable selection; this is possible with **mboost**. Despite directly optimising discrimination, simulation studies do not indicate that this model has better separation than other boosted

or lasso models [212]. GBM-UNO has the same accessibility, transparency, and performance (chapter 7) as previous APT boosting models and is therefore also considered APT.

3.4.3. Novel Adaptations

A clear theme emerging throughout this survey is a historical focus on predicting survival time or ranking, with less interest in direct optimisation of distributional predictions, which may be due to less off-shelf software for the task. Optimisation of a distribution is possible by considering a scoring rule (section 4.6) as the GBM loss. The integrated Graf score (IGS) is discussed below but others are also possible.

The Integrated Graf Score (IGS) is given by,

$$L_{IGS}(t,\delta,\hat{S}|\tau^*) = \int_0^{\tau^*} \frac{\hat{S}(\tau)^2\mathbb{I}(t\leq\tau,\delta=1)}{\hat{G}_{KM}(t)} + \frac{\hat{F}(\tau)^2\mathbb{I}(t>\tau)}{\hat{G}_{KM}(\tau)}\, d\tau \tag{3.4.23}$$

where τ^* is a threshold cut-off but in this case it is assumed $\tau^* = \max\{T_i : i = 1,..,n\}$. Differentiating with respect to $\hat{S}(\tau)$, the negative gradient in the mth iteration is given by

$$r_{im} := \int_0^{\tau^*} 2\hat{f}(\tau)\Big[\frac{\hat{F}(\tau)\mathbb{I}(t_i>\tau)}{\hat{G}_{KM}(\tau)} - \frac{\hat{S}(\tau)\mathbb{I}(t_i\leq\tau,\delta_i=1)}{\hat{G}_{KM}(t_i)}\Big]\, d\tau \tag{3.4.24}$$

where $\hat{f}$ is the estimated probability density function.

Algorithm 4 follows with these equations. The package **mboost** can be utilised to test these equations as a 'custom family'.

3.4.4. Conclusions

Componentwise gradient boosting machines are a highly flexible and powerful machine learning tool. They have proven particularly useful in survival analysis as minimal adjustments are required to make use of off-shelf software. The flexibility of the algorithm allows all the models above to be implemented in very few R (and other programming languages) packages.

Boosting is a method that often relies on intensive computing power and therefore dedicated packages, such as **xgboost** [45], exist to push CPU/GPUs to their limits in order to optimise predictive performance. This can be viewed as a strong advantage though one should be careful not to focus too much on predictive performance to the detriment of accessibility and transparency.

Boosting, especially with tree learners, is viewed as a black-box model that is increasingly difficult to interpret as the number of iterations increase. However, there are several methods for increasing interpretability, such as variable importance and SHAPs [202]. There is also evidence that boosting models can outperform the Cox PH [263] (not something all ML models can claim) and in general survival GBMs are considered APT.

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.

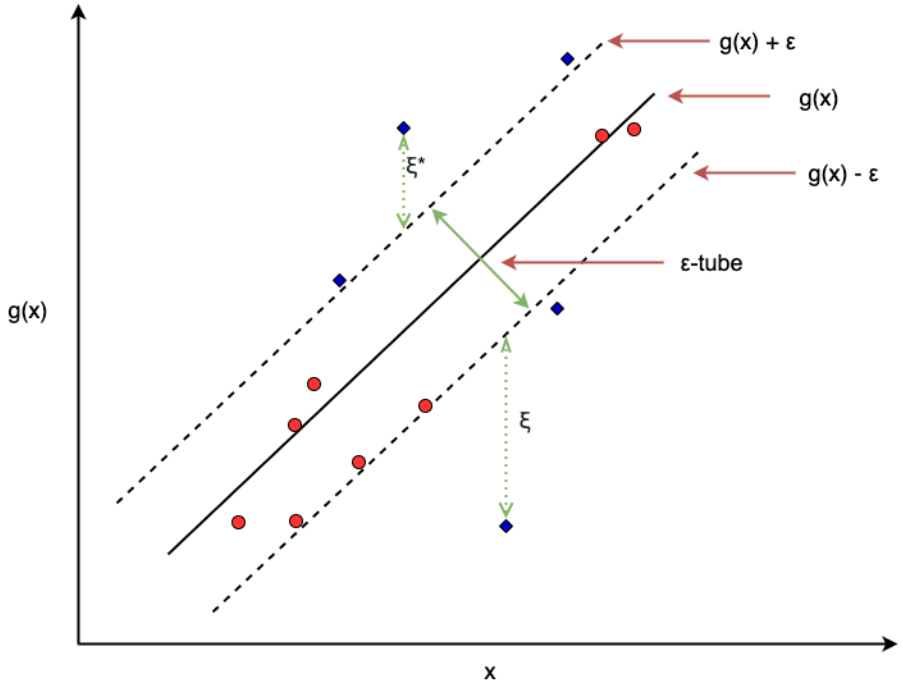


Figure 6: Visualising a support vector machine with an ϵ-tube and slack parameters ξ and ξ^*. Red circles are values within the ϵ-tube and blue diamonds are values outside the tube. x-axis is single covariate, x, and y-axis is $g(x) = x\beta + \beta_0$.

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,

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

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).

3.5.3. Novel Adaptations

Based on the above survey, one novel adaptation is proposed to merge the SSVM-Hybrid with SSVR-MRL. This is a simple addition in which one extra constraint (and associated penalty and slack parameter) is added in order to control for right-censored observations. The SSVM-Hybrid becomes,

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

$$\text{s.t.} \begin{cases} g(X_i) - g(X_{j(i)}) \geq T_i - T_{j(i)} - \xi_i \\ (1-\Delta_i)(g(X_i) - T_i - MRL(T_i|\hat{S})) \leq \xi_i'' \\ \Delta_i(g(X_i) - T_i) \leq \xi_i^* \\ T_i - g(X_i) \leq \xi_i' \\ \xi_i, \xi_i', \xi_i^*, \xi_i'' \geq 0, \quad \forall i = 1, ..., n \end{cases} \tag{3.5.18}$$

Where the ranking (blue) and regression (red) components are unchanged but the additional MRL (magenta) constraint is added for censored observations. One additional parameter should not impact upon fitting time or overfitting too greatly, though this should be tested on large datasets. As with the combination of hybrid and ranking models, the additional constraint can be automatically 'tuned out' of the model, or just manually removed, by setting $\gamma = 0$.

3.5.4. Conclusions

Several SSVMs have been proposed for survival analysis. These can generally be categorised into 'regression' models that adapt SVMs to account for censoring and predict a survival time, 'ranking' models that predict a relative ranking in order to optimise measures of discrimination, and 'hybrid' models that optimise measures of discrimination but make survival time predictions. Other SSVMs that lie outside of these groupings are not able to solve the survival task (e.g. [272]). Other SVM-type approaches could be considered, including relevance vector machines and import vector machines, however less work has been developed in these areas and further consideration is beyond the scope of this thesis.

The models that have received the most attention are SVCR, SSVM-Rank, and SSVM-Hybrid; the first two are special cases of SSVM-Hybrid. Judging if SSVM-Hybrid (and by extension SVCR and SSVM-Rank) is APT is not straightforward. On the one hand it could be considered transparent as SVMs have been studied for decades and the literature for SSVMs, especially from Van Belle, is extensive. On the other hand, the predictions from SSVM-Hybrid should be interpretable as survival times but first hand experience indicates that this is not the case (though this may be due to implementation), which calls into question whether the interpretation they claim to have is actually correct. For accessibility, there appears to be only one implementation of SSVMs in R [84], and also only one in Python [235], which may be due to SSVMs being difficult to implement, even when several optimisation solvers exist off-shelf. Finally, there is no evidence that SSVMs outperform the Cox PH or baseline models and moreover they often perform worse [84, 306], which is also seen in chapter 7. Yet one cannot dismiss SSVMs outright as they often require extensive tuning to perform well, even in classification settings, and no benchmark experiment has yet to emerge for testing SSVMs with the required set-up.[1] Therefore SSVMs may not be APT

[1] Though one is in progress as a result of the work in chapter 7.

for now but future developments will be worth paying attention to.

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,

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

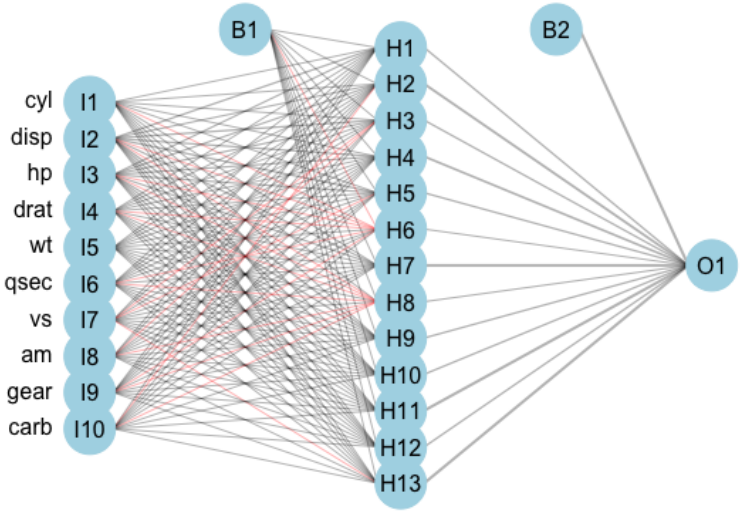


Figure 7: Single-hidden-layer artificial neural network with 13 hidden units fit on the `mtcars` [122] dataset using the **nnet** [222] package, and **gamlss.add** [286] for plotting. Left column are input variables, I1-I10, second column are 13 hidden units, H1-H13, right column is single output variable,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

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

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.

3.6.3. Novel Adaptations

In stark contrast to other model classes, the vast majority of survival ANNs have focused on optimising probabilistic predictions and not relative risks. There does not appear to a model that directly optimises separation via some concordance measure. One simple method could consider RankDeepSurv [148] without L_1. Interestingly, Jing *et al.* [148] compare RankDeepSurv to a model using L_1 only but not to a model using L_2 only, which would be similar to SSVM-Rank [306] (section 3.5.2). RankDeepSurv also likely suffers from the same computational problems as RANKSVMC [303]. However this could be resolved by employing the same nearest-neighbours methodology [304] as in SSVM-Rank. A disadvantage of RankDeepSurv is that the loss does not compare right-censored observations to possibly-correct predictions. Two possible methods to resolve this are either to include a hyper-parameter for 'mean time alive from censoring', ϵ, or an MRL

imputation method, similarly to SSVR-MRL [102] (section 3.5.2). With these adaptations, L_1 is given by

$$L_1(\hat{T}, \theta | \mathcal{D}_0, \epsilon) = \frac{1}{n} \sum_{i=1}^{n} \Delta_i (\hat{T}_i - T_i)^2 + (1 - \Delta_i)(\hat{T}_i - T_i - \epsilon_i)^2 \tag{3.6.33}$$

where either ϵ_i are hyper-parameters for tuning (all could be set equal to prevent overfitting) or $\epsilon_i = MRL(T_i)$. L_2 is given by,

$$\begin{aligned} L_2(\hat{T}, \theta | \mathcal{D}_0) &= \frac{1}{n} \sum_{\{i:I(i)=1\}}^{n} [(T_i - T_{j(i)}) - (\hat{T}_i - \hat{T}_{j(i)})]^2; \\ I(i) &= \begin{cases} 1, & T_i - T_{j(i)} > \hat{T}_i - \hat{T}_{j(i)} \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{3.6.34}$$

where $T_{j(i)}$ is the survival time of the nearest non-censored neighbour with the largest survival time smaller than T_i, the same definition as given by Van Belle *et al.* [306]. The adaptation to L_1 prevents predictions from being discarded in the loss and simultaneously increases penalization applied to under-predictions. The adapted L_2 should also have a faster run-time.

3.6.4. Conclusions

There have been many advances in neural networks for survival analysis. It is not possible to review all proposed survival neural networks without diverting too far from the thesis scope. This survey of ANNs should demonstrate two points: firstly that the vast majority (if not all) of survival ANNs are reduction models that either find a way around censoring via imputation or discretisation of time-intervals, or by focusing on partial likelihoods only; secondly that no survival ANN is APT.

Despite ANNs being highly performant in other areas of supervised learning, there is strong evidence that the survival ANNs above are inferior to a Cox PH when the data follows the PH assumption or when variables are linearly related [95, 204, 233, 241, 327, 328, 329, 332]. There are not enough experiments to make conclusions in the case when the data is non-PH. Experiments in chapter 7 support the finding that survival ANNs are not performant.

There is evidence that many papers introducing neural networks do not utilise proper methods of comparison or evaluation [161] and in conducting this survey, these findings are further supported. Many papers made claims of being 'superior' to the Cox model based on unfair comparisons (section 5.3.3 (RM2)) or miscommunicating (or misinterpreting) results (e.g. [82]). At this stage, it does not seem possible to make any conclusions about the effectiveness of neural networks in survival analysis. Moreover, even the authors of these models have pointed out problems with transparency [17, 197], which was further highlighted by Schwarzer *et al.* [268].

Finally, accessibility of neural networks is also problematic. Many papers

do not release their code and instead just state their networks architecture and available packages. In theory, this is enough to build the models however this does not guarantee the reproducibility that is usually expected. For users with a technical background and good coding ability, many of the models above could be implemented in one of the neural network packages in R, such as **nnet** [222] and **neuralnet** [90]; though in practice the only package that does contain these models, **survivalmodels**, does not directly implement the models in R (which is much slower than Python) but provides a method for interfacing the Python implementations in **pycox** [172].

3.7. Alternative Models

This survey has focused on reviewing machine learning models according to the three key themes of this thesis (section 1.1.1) and within the thesis scope (section 2.2). Therefore this survey has not exhaustively covered all machine learning models and entire model classes have been omitted; this short section briefly discusses these classes.

Bayesian Models As stated in the thesis scope, only frequentist frameworks are considered in this thesis. In terms of accessibility, many more off-shelf survival model implementations exist in the frequentist framework. Despite this, there is good evidence that Bayesian survival models, such as Bayesian neural networks [9, 79], can perform well [23] and a survey of these models may be explored in future work.

Gaussian Processes Gaussian Processes (GPs) are a class of model that naturally fit the survival paradigm as they model the joint distribution of random variables over some continuous domain, often time. The simplest extension from a standard Cox model to GP is given by the non-linear hazard

$$h(\tau|X_i) = h_0(\tau)\phi(g(\tau|X_i)); \quad g(\cdot) \sim \mathcal{GP}(0, k) \tag{3.7.1}$$

where ϕ is a non-negative link function, $\mathcal{GP}$ is a Gaussian process [247], and k is a kernel function with parameters to be estimated [159]. Hyper-parameters are learnt by evaluating the likelihood function [23] and in the context of survival analysis this is commonly performed by assuming an inhomogeneous Poisson process [80, 261, 312]. For a comprehensive survey of GPs for survival, see Saul (2016) [261]. There is evidence of GPs outperforming Cox and ML models [80]. GPs are excluded from this survey due to lack of implementation (thus accessibility) and poorer transparency. Future research could look at increasing off-shelf accessibility of these models.

Non-Supervised Learning As well as pure supervised learning, there are also survival models that use active learning [228], transfer learning, or treat survival analysis as a Markov process. As with GPs, none of these are currently available off-shelf and all require expert knowledge to be useful. These are not discussed

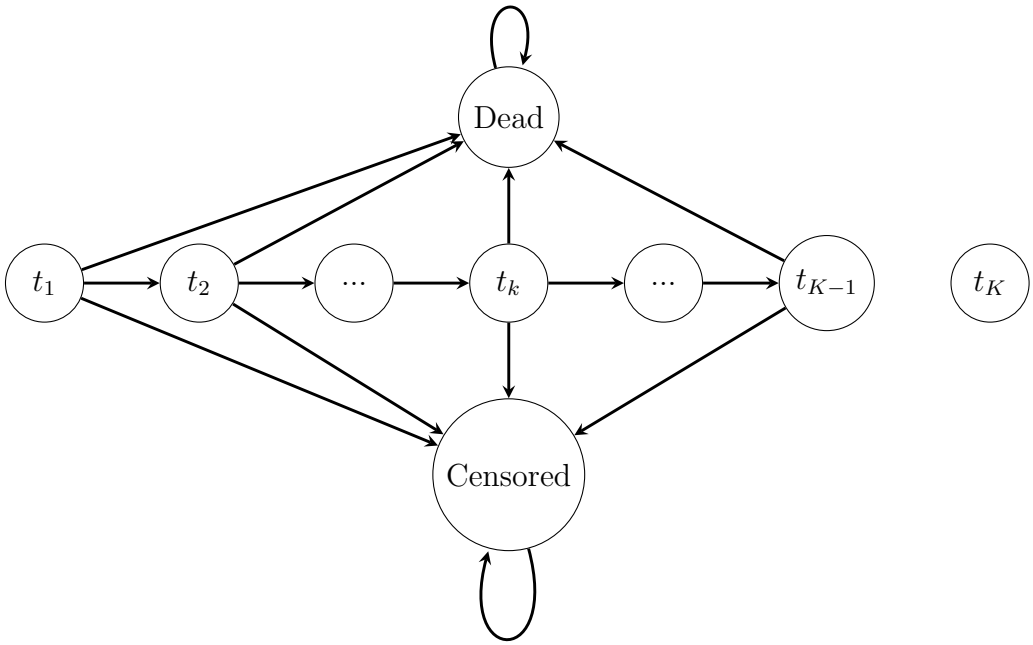


Figure 8: Markov survival process with probabilities suppressed. $t_1, ..., t_K$ are states representing time, 'Censored' and 'Death' are absorbing states corresponding to observed censoring indicator.

in detail here but a very brief introduction to the Markov Process (MP) set-up is provided to motivate further consideration for the area.

Figure 8 visualises the survival set-up as a Markov chain. In each discrete time-point $t_1, ..., t_{K-1}$, an individual can either move to the next time-point (and therefore be alive at that time-point), or move to one of the absorbing states ('Dead' and 'Censored'). The final time-point, t_K, is never visited as an individual must be dead or censored at the end of a study, and hence are last seen alive at t_{K-1}. In this set-up, data is assumed sequential and the time of death or censoring is determined by the last state at which the individual was seen to be alive, plus one, i.e. if an individual transitions from t_k to 'Death', then they died at t_{k+1}. This setting assumes the Markov property, so that the probability of moving to the 'next' state only depends on the current one. This method lends itself naturally to competing risks, which would extend the 'Dead' state to multiple absorbing states for each risk. Additionally, left-censoring can be naturally incorporated without further assumptions [3].

This set-up has been considered in survival both for Markov models and in the context of reinforcement learning [62], though the latter case is underdeveloped and future research could pursue this further.

3.8. Conclusions

This chapter has surveyed models for survival analysis with an emphasis on ML models. The survey was not exhaustive but demonstrates the status quo of accessible ML survival methods. In particular, the survey provides four primary contributions.

Firstly, the taxonomy by prediction type prevents erroneous comparisons between incompatible models, which has been seen in prior surveys.

Secondly, by focusing on transparency and accessibility (and not just predictive performance), the survey highlights how some model classes have seen extensive development and others less so. This is not a criticism of the research but a result of a lack of transparency, accessibility, and conflicting terminology. Unifying notation and terminology allows research to be guided in novel directions as opposed to focusing on pre-existing methodology.

Thirdly, by presenting a comprehensive overview to each ML class, this survey highlights how several models can be adapted by utilising research from other classes. For example, the novel ANN adaptation (section 3.6.3) followed naturally from the SSVM survey by identifying closely related model forms. This survey should allow more concise cross-referencing between model classes in order to guide future developments. Other possible examples for future research include guiding ANN development towards more concise use of reduction (formalised in chapter 5) and adapting random forests to more parametric estimators (a development seen in GBMs).

Finally, novel models are briefly discussed as adaptations to existing methods. In future research these will be studied further for their theoretical properties and analytically compared in benchmark experiments. Adaptations that are deemed 'sensible' (section 1.1.1) will be added to the package **survivalmodels** [275].

Previous surveys of ML models for survival analysis demonstrated problems in proposed methodology. For example in the context of ANNs, Schwarzer *et al.* went so far as to conclude "...these obvious deficiencies show that there is some danger that the fruitful development of statistical methodology for survival data during the last three decades may be wasted with these naive applications of neural networks to such data" [268]. Since this 2000 review, there has been more development in the area and it certainly appears that more robust implementation and methodology has been proposed; though some of the 'naive' applications discussed by Schwarzer *et al.* have also been observed.

No benchmark experiment has yet to emerge in the literature that comprehensively compares all these models on predictive performance, this is resolved in chapter 7.

Chapter 4

Evaluation Measures for Survival Models

This chapter studies how to evaluate the predictions arising from the surveyed models in the previous chapter. 'Model evaluation' is as vague a phrase as 'human evaluation'. A human could be evaluated by a series of exams, physical or neurological tests, aesthetics, etc. Likewise a model could be evaluated according to how well it fits to training data, the quality of predictions on new data, the average prediction, and many more methods. This chapter aims to provide a nuanced approach to defining, understanding, and examining model evaluation. Evaluation is defined in further detail in section 4.1 and throughout this chapter the definition will continue to be refined and specialised to specific sub-types of evaluation, including discrimination (section 4.4), calibration (section 4.5), and overall predictive performance (section 4.6).

Evaluation is a surprising source of disagreement in the literature with some arguing that the process can often be ignored completely [176, 324]. There is a larger divide in survival analysis as many believe that the primary (possibly only) goal is risk prediction [44, 227, 237] and thus other forms of evaluation are not required. These strict views can undermine an integral part of the model building and deployment process, and create more division than necessary. This thesis advocates for strict implementation of model evaluation as a critical part of the model building process as well as in continuous monitoring of deployed models. Without rigorous evaluation, a model cannot be 'trusted' to perform well and could be as useless as making random guesses for all predictions. This is critical in survival analysis, which has important applications in healthcare and finance, in these sectors models that have not been evaluated are potentially dangerous.

An infamous example of evaluation going wrong is the Google Flu Trends (GFT) model[1], which claimed to accurately predict future flu trends but was in fact deemed by many a complete failure as it significantly overestimated all predictions, in some cases doubling the true figures [187]. The GFT model was never utilised (at least openly) in policy and as such no lasting harm was created. However it is not hard to imagine the problems that would be caused by

[1] https://www.google.org/flutrends/about/

such a model if it was utilised and trusted during the time of COVID-19. On a more individual level, as machine learning is increasingly deployed in public sectors, major decisions for patients could become increasingly automated (or at least machine-assisted). Patients should expect their models to be as trained and tested as their doctors.

This chapter attempts to highlight the purpose and need of evaluation in survival analysis by first giving a high-level overview to evaluation as a concept, then providing a brief review of commonly-used survival measures and finally extensive treatment to scoring rules for evaluation of probabilistic predictions, including novel definitions and proofs for properness of scoring rules. The term *measure* will be used throughout this chapter to refer to functions or 'metrics' that quantify some aspect of model evaluation, this should not be confused with a mathematical measure.

The APT criteria will be utilised to survey these measures. For transparency and accessibility, these are straightforward to apply to measures with the same definitions as for models. For predictive performance this is more complicated as it depends on the model class. Therefore optimal measure performance definitions will be covered within each section.

Notation and Terminology The notation introduced in chapter 2 is recapped for use in this chapter. The generative template is given by (X, T, Δ, Y, C) t.v.i. $\mathcal{X} \times \mathcal{T} \times \{0,1\} \times \mathcal{T} \times \mathcal{T}$ where $\mathcal{X} \subseteq \mathbb{R}^p$ and $\mathcal{T} \subseteq \mathbb{R}_{\geq 0}$, where C, Y are unobservable, $T := \min\{Y, C\}$, and $\Delta = \mathbb{I}(Y = T)$. Specific survival data is given by training data, $\mathcal{D}_0 = \{(X_1, T_1, \Delta_1), ..., (X_n, T_n, \Delta_n)\}$ where $(X_i, T_i, \Delta_i) \overset{i.i.d.}{\sim} (X, T, \Delta)$, and test data, $\mathcal{D}_1 = \{(X_1^*, T_1^*, \Delta_1^*), ..., (X_m^*, T_m^*, \Delta_m^*)\}$ where $(X_i^*, T_i^*, \Delta_i^*) \overset{i.i.d.}{\sim} (X, T, \Delta)$.

4.1. Evaluation Overview

4.1.1. What is Evaluation?

Evaluation is the process of examining a model's relationship to data, which may refer to the model's relationship to training data, i.e. how well the model is 'fit' to this data, or the relationship to testing data, i.e. how 'good' are the predictions from the model. In this thesis, only three types of evaluation measure are considered and qualitative definitions of these are given here; more precise definitions appear later in the chapter.

- Discrimination – A model's discriminatory power refers to how well it separates observations that are at a higher or lower risk of event. Therefore discrimination is also sometimes referred to as *separation*. For example, a model with good discrimination will predict that (at a given time) a dead patient has a higher probability of being dead than an alive patient. These measures are the most common in survival and assess relative risk or rank predictions.

- Calibration – There is no single agreed upon definition of model calibration, with definitions varying from paper to paper [56, 117, 246, 308]. Generally, a model is said to be well-calibrated if the average predicted values from the model are in some 'agreement' (which is specified by the chosen measure) with the average true observed values.

- Predictive Performance – A model is said to have good predictive performance (or sometimes 'predictive accuracy') if its predictions for new data are 'close to' the truth.

These are referred to as measures of predictive ability as they draw conclusions about the ability of the model to make predictions.[1]

Using these definitions as a primary taxonomy for survival measures is problematic as without clear definitions there can be significant overlap between model 'classes'. Instead this thesis advocates for the same taxonomy as in the previous chapter and categorises measures by the return type that they evaluate: survival time, ranking, or survival distribution.

Goodness-of-fit measures are very briefly discussed in section 4.2 for completeness, however these are generally out of scope in this thesis as the vast majority (if any) cannot evaluate machine learning models.

4.1.2. Why are Models Evaluated?

A key element of the scientific method is experiments and validation. In the usual workflow of the scientific method: i) a hypothesis is proposed; ii) predictions are made; and iii) experiments are performed to test the hypothesis based on these predictions. For statistical models the same principles are upheld: i) a model is proposed (by manual or automated selection with possible tuning); ii) predictions are made either internally (cross-validation) or externally (held-out data); and iii) validation is performed on these predictions in order to infer something about the model's performance. The model can then be considered 'good' or 'bad' and either deployed, adjusted, or discarded. As these are models that are run on a computer (as opposed to experiments in the real-world), the process from fitting to validating is relatively quick and as such multiple proposed models can be evaluated and compared at the same time. This provides two key use-cases for evaluation: i) demonstrating model performance; and ii) model comparison/selection.

Resistance to model evaluation can be found in the machine learning community. One such example are proponents of inhomogeneous ensemble methods, which combine predictions from multiple different models into a single prediction. The arguments for these models are that: i) model evaluation can never be precise enough, or strong enough guarantees cannot be given [146]; and ii) ensemble methods can guarantee a better performance than the individual component models and therefore evaluation of the components is not required. For example,

[1] Measures of predictive ability measure a model's *ability* to make any form of prediction. Measures of predictive performance measure the *performance* of the predictions. In this section a model's predictive ability refers to all three of discrimination, calibration, and predictive performance.

'super learners' [176] are a class of such model and claim[1] to guarantee that a super learner will always perform as well as, if not better, than its component models: "...the super learner framework allows a researcher to try many prediction algorithms...knowing that the final combined super learner fit will either be the best fit or near the best fit" [239]. This has three problems, it: i) assumes that researchers will only fit sensible prediction algorithms; ii) advocates for complex ensemble models instead of transparent and parsimonious ones; and iii) assumes that a super learner is guaranteed to be the (near) 'best fit', which actively discourages simpler models being tested separately. Each of these problems can be resolved by researchers only fitting sensible models and opting for an Occam's Razor approach where inhomogeneous ensemble methods are used only if they outperform simpler models, thus requiring validation to test this.

By the parsimony principle, if two models have the same predictive performance (within some degree of confidence), then the simpler and more transparent model is preferred. Even a very slight gain in predictive performance could be outweighed by a large increase to complexity. All models, whether simple or complex, should be critically compared to many alternatives. At the very least a model should be compared to a baseline (section 4.6.5.1) as many performance measures are uninterpretable without a point of comparison [111].

4.1.3. How are Models Evaluated?

The process of evaluation in machine learning is briefly given as a key method in section 2.4 and relevant parts are repeated here. The evaluation process itself is a simple application of a suitable mathematical function to predictions and true data. Let L be some evaluation measure and for now assume L is a measure evaluating deterministic predictions (the following generalises to other types trivially). A model will either be evaluated on each prediction separately, in which case $L : \mathbb{R} \times \mathbb{R} \to \bar{\mathbb{R}}$ or the measure is calculated for all predictions simultaneously, in which case $L : \mathbb{R}^m \times \mathbb{R}^m \to \bar{\mathbb{R}}$. Specifically the loss parameters are observed (true) outcomes, Y, and predictions of this outcome, $\hat{Y}$. L is usually referred to as a *loss* when L should be minimised for optimal prediction, whereas a *score* is the term given when L should be maximised.

All evaluation measures discussed in this thesis are out-of-sample measures and therefore evaluation takes place after the model makes predictions on held-out test data.

Specific choices for L are now reviewed.

[1] Testing this claim is tangential so for now will be assumed true.

4.2. In-Sample Measures

In-sample measures are not examined in this thesis as no in-sample measures could be found that are applicable to all machine learning methods and therefore are out of scope for this thesis. Instead, the interested reader is referred to the papers and references listed below:

Residuals For discussion about model residuals, refer to texts on survival modelling fitting and goodness-of-fit such as:

- David Collett. *Modelling Survival Data in Medical Research*. 3rd ed. CRC, 2014. ISBN: 978-1584883258
- David W Hosmer Jr, Stanley Lemeshow, and Susanne May. *Applied survival analysis: regression modeling of time-to-event data*. Vol. 618. John Wiley & Sons, 2011. ISBN: 1118211588

Both provide a comprehensive overview to model residuals for semi- and fully-parametric low-complexity survival models.

R^2 measures R^2 type measures have been the focus of several reviews and surveys, in particular the following are recommended:

- Babak Choodari-Oskooei, Patrick Royston, and Mahesh K.B. Parmar. "A simulation study of predictive ability measures in a survival model I: Explained variation measures". In: *Statistics in Medicine* 31.23 (2012), pp. 2627–2643. ISSN: 02776715. DOI: 10.1002/sim.4242 — For a comprehensive review and simulation study of R^2 type measures
- John T. Kent and John O'Quigley. "Measures of dependence for censored survival data". In: *Biometrika* 75.3 (1988), pp. 525–534. ISSN: 00063444. DOI: 10.1093/biomet/75.3.525 — Defines the commonly utilised Kent and O'Quigley R^2 measure
- Patrick Royston and Willi Sauerbrei. "A new measure of prognostic separation in survival data". In: *Statistics in Medicine* 23.5 (2004), pp. 723–748. ISSN: 02776715. DOI: 10.1002/sim.1621 — Defines the commonly utilised Royston and Sauerbrei R^2 measure

Likelihood and Information Criteria Measures of likelihood and information criteria (e.g. AIC, BIC) are commonly utilised in in-sample model comparison of low-complexity survival models though in general are harder (if not impossible) to compute on ML alternatives.

These criterion are originally defined in:

- Hirotugu Akaike. "A New Look at the Statistical Model Identification". In: *IEEE Transactions on Automatic Control* 19.6 (1974), pp. 716–723. ISSN: 17451337. DOI: 10.1093/ietfec/e90-a.12.2762. arXiv: arXiv:1011.1669v3 — For the introduction of the AIC

- Gideon Schwarz. "Estimating the Dimension of a Model". In: *The Annals of Statistics* 6.2 (1978), pp. 461–464. ISSN: 0090-5364. DOI: 10.1214/aos/1176344136. arXiv: arXiv:1011.1669v3. URL: http://projecteuclid.org/euclid.aos/1176344136 — For the introduction of the BIC

These are discussed for survival analysis in:

- Chris T Volinsky and Adrian E Raftery. "Bayesian Information Criterion for Censored Survival Models". In: *International Biometric Society* 56.1 (2000), pp. 256–262 — For discussion on the BIC for survival models.
- CLIFFORD M HURVICH and CHIH-LING TSAI. "Regression and time series model selection in small samples". In: *Biometrika* 76.2 (1989), pp. 297–307. ISSN: 0006-3444. DOI: 10.1093/biomet/76.2.297. URL: https://doi.org/10.1093/biomet/76.2.297 — Definition of corrected AIC for survival models, AIC_C
- Hua Liang and Guohua Zou. "Improved AIC Selection Strategy for Survival Analysis". eng. In: *Computational statistics & data analysis* 52.5 (2008), pp. 2538–2548. ISSN: 0167-9473. DOI: 10.1016/j.csda.2007.09.003. URL: https://www.ncbi.nlm.nih.gov/pubmed/19158943https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2344147/ — 'Improved' AIC for survival models.

4.3. Evaluating Survival Time

There appears to be little research into measures for evaluating survival time predictions, which is likely due to this task usually being of less interest than the others (section 2.3). Common measures in survival analysis for survival time predictions are the same as regression measures but with an additional indicator variable to remove censoring. Three common regression measures are the mean absolute error (MAE), mean squared error (MSE), and root mean squared error (RMSE). These are respectively defined for survival analysis as

Definition 4.3.1. Let $\mathcal{T}^m \subseteq \mathbb{R}^m_{>0}$, $\hat{t} = \hat{t}_1, ..., \hat{t}_m, t = t_1, ..., t_m$, $\delta = \delta_1, ..., \delta_m$, and $d := \sum_{i=1}^m \delta_i$, then

i) The *censoring-adjusted mean absolute error*, MAE_C is defined by

$$MAE_C : \mathcal{T}^m \times \mathcal{T}^m \times \{0,1\}^m \to \mathbb{R}_{\geq 0}; (\hat{t}, t, \delta) \mapsto \frac{1}{d} \sum_{i=1}^{m} \delta_i |t_i - \hat{t}_i| \quad (4.3.1)$$

ii) The *censoring-adjusted mean squared error*, MSE_C is defined by

$$MSE_C : \mathcal{T}^m \times \mathcal{T}^m \times \{0,1\}^m \to \mathbb{R}_{\geq 0}; (\hat{t}, t, \delta) \mapsto \frac{1}{d} \sum_{i=1}^{m} \delta_i (t_i - \hat{t}_i)^2 \quad (4.3.2)$$

iii) The *censoring-adjusted root mean squared error*, $RMSE_C$ is defined by

$$RMSE_C : \mathcal{T}^m \times \mathcal{T}^m \times \{0,1\}^m \to \mathbb{R}_{\geq 0}; (\hat{t}, t, \delta) \mapsto \sqrt{MSE_C(t, \hat{t}, \delta)} \quad (4.3.3)$$

These are referred to as 'distance' measures as they measure the distance between the true, (t, δ), and predicted, $\hat{t}$, values. This approach is not ideal as the removal of censored observations results in increased bias as the proportion of censoring increases (section 4.4.1). Furthermore these measures make some assumptions that are likely not valid in a survival setting. For example these metrics assume that an over-prediction should be penalised equally as much as an under-prediction, whereas in survival data it is likely that a model should be overly-cautious and under-predict survival times, i.e. it is safer to predict a patient is more at risk and will die sooner rather than less risk and die later.

These measures are clearly transparent and accessible as off-shelf implementation is straightforward, though **mlr3proba** [281] was the only R package found to implement these. For performance, no conclusions can be drawn as no research could be found into the theoretical properties of these losses; despite this there is evidence of them being utilised in the literature [317].

4.4. Evaluating Continuous Rankings

The next category of survival measures assess predictive performance via discrimination for the evaluation of continuous ranking predictions. Assessment of continuous rankings are also possible by measures of calibration however few methods could be found that generalised to all (not just PH) model forms. Therefore this section exclusively discusses measures of discrimination. First time-independent concordance indices (section 4.4.1) are discussed and then time-dependent AUCs (section 4.4.2).

Measures of discrimination identify how well a model can separate patients into different risk groups. A model has perfect discrimination if it correctly predicts that patient i is at higher risk of death than patient j if patient i dies first. This risk of death is derived from the ranking prediction type. All discrimination measures are ranking measures, which means that the exact predicted value is irrelevant, only its relative ordering is required. For example given predictions $\{100, 2, 299.3\}$, only their rankings, $\{2, 1, 3\}$, are used by measures of discrimination.

4.4.1. Concordance Indices

The simplest form of discrimination measures are concordance indices, which in general measure the proportion of cases in which the model correctly separates a pair of observations into 'low' and 'high' risk.

Definition 4.4.1. Let (i, j) be a pair of observations with outcomes $\{(t_i, \delta_i), (t_j, \delta_j)\} \overset{i.i.d.}{\sim} (T, \Delta)$ and let $y_i, y_j \in \mathbb{R}$ be their respective risk predictions. Then (i, j) are called [115, 116]:

- *Comparable* if $t_i < t_j$ and $\delta_i = 1$;
- *Concordant* if $y_i > y_j$.[1]

A concordance index (C-index) is a weighted proportion of the number of concordant pairs over the number of comparable pairs. As such, a C-index value is between $[0, 1]$ with 1 indicating perfect separation, 0.5 indicating no separation, and 0 being separation in the 'wrong direction', i.e. all high risk patients being ranked lower than all low risk patients. Concordance measures may either be reported as a value in $[0, 1]$, a percentage, or as 'discriminatory power'. Discriminatory power refers to the percentage improvement of a model's discrimination above the baseline value of 0.5. For example if a model has a concordance of 0.8 then its discriminatory power is $(0.8 - 0.5)/0.5 = 60\%$. This representation of discrimination provides more information by encoding the model's improvement over some baseline although is often confused with reporting concordance as a percentage (e.g. reporting a concordance of 0.8 as 80%).

The most common concordance indices can be expressed as a general measure.

Definition 4.4.2. Let $\mathcal{T}^m \subseteq \mathbb{R}^m_{>0}$, $y = y_1, ..., y_m, t = t_1, ..., t_m$, $\delta = \delta_1, ..., \delta_m$, and let W be a weighting function. Then, the *survival concordance index* is defined by,

$$\begin{aligned} &C : \mathbb{R}^m \times \mathcal{T}^m \times \{0, 1\}^m \times \mathbb{R}_{\geq 0} \to [0, 1]; \\ &(y, t, \delta | \tau) \mapsto \frac{\sum_{i \neq j} W(t_i) \mathbb{I}(t_i < t_j, y_i > y_j, t_i < \tau) \delta_i}{\sum_{i \neq j} W(t_i) \mathbb{I}(t_i < t_j, t_i < \tau) \delta_i} \end{aligned} \tag{4.4.1}$$

for some cut-off time τ.

The choice of W specifies a particular evaluation measure (see below). To evaluate the discrimination of a prediction functional, $\hat{g}$, with predicted rankings from the model, $r = r_1, ..., r_m$, the concordance is calculated as $C(r, (T^*_1, ..., T^*_m), (\Delta^*_1, ..., \Delta^*_m) | \tau)$ for some choice of $\tau \in \mathbb{R}_{\geq 0}$. The use of the cut-off τ mitigates against decreased sample size over time due to the removal of censored observations. There are multiple methods for dealing with tied times but in practice a value of 0.5 is usually taken when $t_i = t_j$ [293]. The following weights have been proposed for the concordance index [293]:

- $W(t_i) = 1$ – This is Harrell's concordance index, C_H [115, 116], which is widely accepted to be the most common survival measure [56, 105, 246]. There is no cut-off in the original definition of C_H ($\tau = \infty$).

[1] Recall (section 2.3) this thesis defines the risk ranking such that a higher value implies higher risk of death and so a pair is concordant if $\mathbb{I}(t_i < t_j, y_i > y_j)$, whereas this would be $\mathbb{I}(t_i < t_j, y_i < y_j)$ if a higher value implied a lower risk of death.

- $W(t_i) = [\hat{G}_{KM}(t_i)]^{-2}$ – This is Uno's C, C_U [300]. $\hat{G}_{KM}$ is the Kaplan-Meier estimate of the survival function of the censoring distribution fit on training data. This is referred to as an Inverse Probability of Censoring Weighted (IPCW) measure as the estimated censoring distribution is utilised to weight the measure in order to compensate for removed censored observations.
- $W(t_i) = [\hat{G}_{KM}(t_i)]^{-1}$
- $W(t_i) = \hat{S}_{KM}(t_i)$. $\hat{S}_{KM}$ is the Kaplan-Meier estimator of the survival distribution.
- $W(t_i) = \hat{S}_{KM}(t_i)/\hat{G}_{KM}(t_i)$

All methods assume that censoring is conditionally-independent of the event given the features (section 2.1.2), otherwise weighting by $\hat{S}_{KM}$ or $\hat{G}_{KM}$ would not be applicable. It is assumed here that $\hat{S}_{KM}$ and $\hat{G}_{KM}$ are estimated on the training data and not the testing data (though the latter is often seen in implementation [291]).

With respect to being APT, all concordance indices are highly transparent and accessible, with many off-shelf implementations. With respect to performance, Choodari-Oskooei *et al.* (2012) [50] define a measure as performant if it is:[1] i) independent of censoring; ii) interpretable; and iii) robust against outliers. This second property is already covered by 'transparency'. The third property is guaranteed for all measures of concordance, which are ranking measures; all outliers are removed once ranks are applied to predictions. Therefore the first property, "a measure that is the least affected by the amount of censoring is generally preferred" [50], is now considered.

Several papers have shown that C_H is affected by the precense of censoring [169, 237, 254, 300] as the measure ignores pairs in which the shorter survival time is censored. Despite this, C_H is still the most widely utilised measure and moreover if a suitable cut-of τ is chosen, then all these weightings perform very similarly [246, 265].

Measures that utilise other weightings have been demonstrated to be less affected by censoring than C_H [246]. However if a poor choice is selected for τ then IPCW measures (which include $\hat{G}_{KM}$ in the weighting) can be highly unstable [246]. For example, the variance of C_U has been shown to drastically increase more than other measures with increased censoring [265].

None of these measures are perfect and all have been shown to be affected to some extent by censoring [265], which can lead to both under-confidence and over-confidence in the model's discriminatory ability. For example, C_U has been observed to report values as low as 0.2 when the 'true estimate' was 0.6 [265]. Therefore interpreting a value from these measures can be very difficult, for example naively reporting a concordance of 60% when $C_H = 0.6$ would be incorrect as this value may mean very different things for different amounts of censoring. Whilst intepreting these measures may be difficult, it is not impossible as

[1] This paper refers specifically to measures of explained variation and therefore only the properties that generalise to all measures are included here.

all these estimators tend to produce values around a similar range [246, 265]. Therefore this thesis advocates for multiple concordance indices being reported alongside expert interpretation that takes into account sample size and censoring proportions [265] as well as 'risk profiles' (how at risk patients are) [246].

For within-study model comparison, instability from censoring is not of concern as the measure will be affected equally across all models; though interpretation remains difficult. However a concordance from one study cannot be compared to that from another if the datasets differ greatly in the proportion of censoring. Future research could consider more robust concordance indices that can provide greater ease of interpretation.

As well as the concordance indices discussed here, another promiment alterntive was derived by Gönen and Heller (2005) [105]. However as this is only applicable to the Cox PH it is out of scope for this thesis, which is primarily concerned with generalisable measures for model comparison.

In simulation experiments, the concordance indices that tended to perform 'better' were those based on AUC-type measures, these are now discussed.

4.4.2. AUC Measures

AUC, or AUROC, measures calculate the Area Under the Receiver Operating Characteristic (ROC) Curve, which is a plot of the *sensitivity* (or true positive rate (TPR)) against $1-$*specificity* (or true negative rate (TNR)) at varying thresholds (described below) for the predicted probability (or risk) of event. Figure 9 visualises ROC curves for two classification models. The blue line is a featureless baseline that has no discrimination. The red line is a decision tree with better discrimination as it comes closer to the top-left corner.

In a classification setting with no censoring, the AUC has the same interpretation as Harrell's C [300]. AUC measures for survival analysis have been developed in order to provide a time-dependent measure of discriminatory ability [120]. The proposed concordance indices described above are time-independent, which is useful for producing a single statistic. However, in a survival setting it can reasonably be expected for a model to perform differently over time and therefore time-dependent measures are advantageous. First discussion around computation of TPR and TNR are provided and then how these are incorporated into the AUC equation.

The AUC, TPR, and TNR are derived from the *confusion matrix* in a binary classification setting. Let $b, \hat{b} \in \{0, 1\}$ be the true and predicted binary outcomes respectively. The confusion matrix is

	$b = 1$	$b = 0$
$\hat{b} = 1$	TP	FP
$\hat{b} = 0$	FN	TN

where $TN := \sum_i \mathbb{I}(b = 0, \hat{b} = 0)$ is the number of (#) true negatives, $TP := \sum_i \mathbb{I}(b = 1, \hat{b} = 1)$ is # true positives, $FP := \sum_i \mathbb{I}(b = 0, \hat{b} = 1)$ is # false

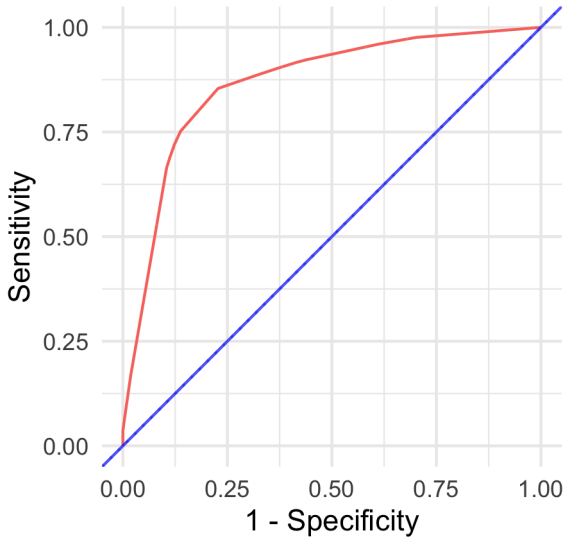


Figure 9: ROC Curves for a classification example. Red is a decision tree with good discrimination as it 'hugs' the top-left corner. Blue is a featureless baseline with no discrimination as it sits on $y = x$.

positives, and $FN := \sum_i \mathbb{I}(b = 1, \hat{b} = 0)$ is # false negatives. From these are derived

$$TPR := \frac{TP}{TP + FN} \tag{4.4.2}$$

$$TNR := \frac{TN}{TN + FP} \tag{4.4.3}$$

In classification, a probabilistic prediction of an event can simply be *thresholded* (or 'binarised') to obtain a deterministic prediction. For a predicted $\hat{p} := \hat{P}(b = 1)$, and threshold α, the thresholded binary prediction is given by $\hat{b} := \mathbb{I}(\hat{p} > \alpha)$. In survival analysis, this is complicated as either models only predict a continuous ranking (and not a probability of death), or a full survival distribution, which implies that the probability of death changes over time; it is the first of these that is utilised in AUC measures. Two primary methods for doing so have emerged, the first is to use an IPCW method to weight the thresholded linear predictor by an estimated censoring distribution at a given time, the second is to first classify cases and controls then compute estimators based on these classes. All measures of TPR, TNR and AUC are in the range $[0, 1]$ with larger values preferred.

Weighting the linear predictor was proposed by Uno *et al.* (2007) [299] and provides a method for estimating TPR and TNR via

$$\begin{aligned} &TPR_U : \mathbb{R}^m \times \mathbb{R}^m_{\geq 0} \times \{0,1\}^m \times \mathbb{R}_{\geq 0} \times \mathbb{R} \to [0,1]; \\ &(\hat{\eta}, t, \delta | \tau, \alpha) \mapsto \frac{\sum_{i=1}^m \delta_i \mathbb{I}(k(\hat{\eta}_i) > \alpha, t_i \leq \tau)[\hat{G}_{KM}(t_i)]^{-1}}{\sum_{i=1}^m \delta_i \mathbb{I}(t_i \leq \tau)[\hat{G}_{KM}(t_i)]^{-1}} \end{aligned} \tag{4.4.4}$$

and

$$\begin{aligned} &TNR_U : \mathbb{R}^m \times \mathbb{R}^m_{\geq 0} \times \mathbb{R}_{\geq 0} \times \mathbb{R} \to [0,1]; \\ &(\hat{\eta}, t|\tau, \alpha) \mapsto \frac{\sum_{i=1}^m \mathbb{I}(k(\hat{\eta}_i) \leq \alpha, t_i > \tau)}{\sum_{i=1}^m \mathbb{I}(t_i > \tau)} \end{aligned} \tag{4.4.5}$$

where τ is the time at which to evaluate the measure, α is a cut-off for the linear predictor, and k is a known, strictly increasing, differentiable function. k is chosen depending on the model choice, for example if the fitted model is PH then $k(x) = 1 - \exp(-\exp(x))$ [299]. Similarities can be drawn between these equations and Uno's concordance index, in particular the use of IPCW. Censoring is again assumed to be at least random once conditioned on features. Plotting TPR_U against $1 - TNR_U$ for varying values of α provides the ROC.

The second method, which appears to be more prominent in the literature, is derived from Heagerty and Zheng (2005) [121]. They define four distinct classes, in which observations are split into controls and cases.

An observation is a *case* at a given time-point if they are dead, otherwise they are a *control*. These definitions imply that all observations begin as controls and (hypothetically) become cases over time. Cases are then split into *incident* or *cumulative* and controls are split into *static* or *dynamic*. The choice between modelling static or dynamic controls is dependent on the question of interest. Modelling static controls implies that a 'subject does not change disease status' [121], and few methods have been developed for this setting [152], as such the focus here is on *dynamic* controls. The incident/cumulative cases choice is discussed in more detail below.[1]

The TNR for dynamic cases is defined as

$$TNR_D(y, N|\alpha, \tau) = P(y_i \leq \alpha | N_i(\tau) = 0) \tag{4.4.6}$$

where $y = (y_1, ..., y_n)$ is some deterministic prediction and $N(\tau)$ is a count of the number of events in $[0, \tau)$. Heagerty and Zheng further specify y to be the predicted linear predictor $\hat{\eta}$. Cumulative/dynamic and incident/dynamic measures are available in software packages 'off-shelf', these are respectively defined by

$$TPR_C(y, N|\alpha, \tau) = P(y_i > \alpha | N_i(\tau) = 1) \tag{4.4.7}$$

and

$$TPR_I(y, N|\alpha, \tau) = P(y_i > \alpha | dN_i(\tau) = 1) \tag{4.4.8}$$

where $dN_i(\tau) = N_i(\tau) - N_i(\tau-)$. Practical estimation of these quantities is not discussed here.

[1] All measures discussed in this section evaluate model discrimination from 'markers', which may be a *predictive* marker (model predictions) or a *prognostic* marker (a single covariate). This section always defines a marker as a ranking prediction, which is valid for all measures discussed here with the exception of one given at the end.

The choice between the incident/dynamic (I/D) and cumulative/dynamic (C/D) measures primarily relates to the use-case. The C/D measures are preferred if a specific time-point is of interest [121] and is implemented in several applications for this purpose [152]. The I/D measures are preferred when the true survival time is known and discrimination is desired at the given event time [121].

Defining a time-specific AUC is now possible with

$$AUC(y, N|\tau) = \int_0^1 TPR(y, N|1 - TNR^{-1}(p|\tau), \tau)\ dp \tag{4.4.9}$$

Finally, integrating over all time-points produces a time-dependent AUC and as usual a cut-off is applied for the upper limit,

$$AUC^*(y, N|\tau^*) = \int_0^{\tau^*} AUC(y, N|\tau) \frac{2\hat{p}_{KM}(\tau)\hat{S}_{KM}(\tau)}{1 - \hat{S}^2_{KM}(\tau^*)}\ d\tau \tag{4.4.10}$$

where $\hat{S}_{KM}, \hat{p}_{KM}$ are survival and mass functions estimated with a Kaplan-Meier model on training data.

Since Heagerty and Zheng's paper, other methods for calculating the time-dependent AUC have been devised, including by Chambless and Diao [42], Song and Zhou [282], and Hung and Chiang [136]. These either stem from the Heagerty and Zheng paper or ignore the case/control distinction and derive the AUC via different estimation methods of TPR and TNR. Blanche *et al.* (2012) [25] surveyed these and concluded "regarding the choice of the retained definition for cases and controls, no clear guidance has really emerged in the literature", but agree with Heagerty and Zeng on the use of C/D for clinical trials and I/D for 'pure' evaluation of the marker. Blanche *et al.* (2013) [24] published a survey of C/D AUC measures with an emphasis on non-parametric estimators with marker-dependent censoring, including their own Conditional IPCW (CIPCW) AUC,

$$AUC_B(y, t, \delta, \hat{G}|\tau) = \frac{\sum_{i \neq j} \mathbb{I}(y_i > y_j)\mathbb{I}(t_i \leq \tau, t_j > \tau)\frac{\delta_i}{m^2 \hat{G}(t_i|y_i)\hat{G}(\tau|y_j)}}{\left(\sum_{i=1}^m \mathbb{I}(t_i \leq \tau)\frac{\delta_i}{m\hat{G}(t_i|y_i)}\right)\left(\sum_{j=1}^m \mathbb{I}(t_j > \tau)\frac{1}{m\hat{G}(\tau|y_j)}\right)} \tag{4.4.11}$$

where $t = (t_1, ..., t_m)$, and $\hat{G}$ is the Akritas [5] estimator of the censoring distribution (section 3.1.1). It can be shown that setting the λ parameter of the Akritas estimator to 1 results in the IPCW estimators [24]. However unlike the previous measures in which a deterministic prediction can be substituted for the marker, this is not valid for this estimator and as such this cannot be used for predictions. This is clear from the weights, $\hat{G}(t|y)$, in the equation which are dependent on the prediction itself. The purpose of the CIPCW method is to adapt the IPCW weights to be conditioned on the data covariates, which is not the case when y is

a predictive marker. Hence the following adaptation is considered instead,

$$AUC^*_B(y, x, t, \delta, \hat{G}|\tau) = \frac{\sum_{i \neq j} \mathbb{I}(y_i > y_j)\mathbb{I}(t_i \leq \tau, t_j > \tau)\frac{\delta_i}{m^2 \hat{G}(t_i|x_i)\hat{G}(\tau|x_j)}}{\left(\sum_{i=1}^m \mathbb{I}(t_i \leq \tau)\frac{\delta_i}{m\hat{G}(t_i|x_i)}\right)\left(\sum_{j=1}^m \mathbb{I}(t_j > \tau)\frac{1}{m\hat{G}(\tau|x_j)}\right)} \tag{4.4.12}$$

where x are random covariates (possibly from a separate training dataset).

AUC measures are less transparent and less accessible than the simpler time-independent concordance indices, only the **survAUC** [240] package could be found that implements these measures. For performance, reviews of these measures have produced (sometimes markedly) different results [25, 195, 152] with no clear consensus on how and when these measures should be used. The primary advantage of these measures is to extend discrimination metrics to be time-dependent. However it is unclear how to interpret a threshold of a linear predictor and moreover if this is even the 'correct' quantity to threshold, especially when survival distribution predictions are the more natural object to evaluate over time. Methods for evaluating these distribution predictions are now discussed.

4.5. Evaluating Distributions by Calibration

The final discussed measures are for evaluating survival distributions. First measures of calibration are briefly discussed in this section and then extensive treatment is given to scoring rules (section 4.6).

Random Variable and Distribution Notation Throughout these next two sections, two different notations are utilised for random variables and distributions. The first is the 'standard' notation, for example if ζ is a continuous probability distribution and $X \sim \zeta$ is a random variable, then f_X is the probability density function of X. The second notation associates distribution functions directly with the distribution and not the variable. For example if ζ is a continuous probability distribution then $\zeta.f$ is the probability density function of ζ. Analogously for the probability mass, cumulative distribution, hazard, cumulative hazard, and survival functions of $X \sim \zeta$, $p_X/\zeta.p, F_X/\zeta.F, h_X/\zeta.h, H_X/\zeta.H, S_X/\zeta.S$. This notation (fully described and motivated in section 6.3.1) provides a clearer separation of probability distributions and random variables, which in turn allows for cleaner proofs involving probability distributions.

Measures of Calibration Few measures of calibration exist in survival analysis [246] and this is likely due to the meaning of calibration being unclear in this context [308]. This is compounded by the fact that calibration is often evaluated graphically, which can leave room for high subjectivity and thus may be restricted to expert interpretation. For these reasons, measures of calibration are only considered in this thesis with respect to accessibility and transparency as there is no clear meaning for what makes a calibration measure performant. Many methods of calibration are restricted to calibration and re-calibration of PH models [66,

308], none of these are considered here as they do not generalise to all (or at least many) survival models.

Point and Probabilistic Calibration Andres *et al.* (2018) [8] derived a taxonomy for calibration measures to separate measures that only evaluate distributions at a single time-point ('1-Calibration') and measures that evaluate distributions at all time-points ('distributional-calibration'). This section will use the same taxonomy but in keeping with machine learning terminology will refer to '1-Calibration' as 'Point Calibration' and 'distributional-calibration' as 'Probabilistic Calibration'.

All measures considered previously can be viewed as 'point' measures as they evaluate predictions at a single point, specifically comparing the predicted linear predictor (more generally relative risk) or survival time to the true time of death. However calibration measures and scoring rules instead evaluate predicted distributions and specifically functions that vary over time, hence it is often of more interest to evaluate these functions at multiple (all if discrete) time-points in order to derive a metric that captures changes over time. For example one may expect probabilistic predictions to be more accurate in the near-future and to steadily worsen as uncertainty increases over time (both mathematical (censoring) and real-world uncertainty), and therefore a measure that only evaluates distributions at a single (possibly early) time-point cannot assess the true variation in the prediction.

Mathematically this difference in measures may be considered as follows: Let $\mathcal{P}$ be a set of distributions over $\mathcal{T} \subseteq \mathbb{R}_{>0}$, then a point measure for evaluating distributions is given by,

$$L_1 : \mathcal{P} \times \mathcal{T} \times \{0,1\} \times \mathcal{T} \to \bar{\mathbb{R}}; \quad (\zeta, t, \delta|\tau) \mapsto g_1(\zeta.\rho(\tau), t, \delta) \tag{4.5.1}$$

and a probabilistic measure is given by,

$$L_P : \mathcal{P} \times \mathcal{T} \times \{0,1\} \times \mathbb{R}_{>0} \to \bar{\mathbb{R}}; \quad (\zeta, t, \delta|\tau^*) \mapsto \int_0^{\tau^*} g_P(\zeta.\rho(\tau), t, \delta)\ d\tau \tag{4.5.2}$$

or

$$L_P : \mathcal{P} \times \mathcal{T} \times \{0,1\} \times \mathbb{R}_{>0} \to \bar{\mathbb{R}}; \quad (\zeta, t, \delta|\tau^*) \mapsto \sum_{\tau=0}^{\tau^*} g_P(\zeta.\rho(\tau), t, \delta) \tag{4.5.3}$$

where τ^* is some cut-off for the measure to control uncertainty increasing over time, ρ is usually the survival function but may be any distribution-defining function, and g_1, g_P are functions corresponding to specific measures (some examples in next two sections). Note that τ is an argument (not a free variable) of L_1 as the fixed choice of τ is measure-dependent; usually $\tau = t$.

Less abstractly, a point-calibration measure will evaluate a function of the predicted distribution at a single time-point whereas a probabilistic measure evaluates the distribution over a range of time-points; in both cases the evaluated quantity is compared to the observed outcome, (T^*, Δ^*).

4.5.1. Point Calibration

Point calibration measures can be further divided into metrics that evaluate calibration at a single time-point (by reduction) and measures that evaluate an entire distribution by only considering the event time. The subtle difference significantly affects conclusions that can be drawn. In the first case, a calibration measure can only draw conclusions at that one time-point, whereas the second case can draw conclusions about the calibration of the entire distribution.

4.5.1.1. Calibration by Reduction

Point calibration measures are implicitly reduction methods as they attempt to evaluate a full distribution based on a single point only. For example given a predicted survival function $\zeta.S$, then one could select a time-point τ^* and calculate the survival function at this time, $\zeta.S(\tau^*)$, probabilistic classification calibration measures can then be utilised. Using this approach one may employ common calibration methods such as the Hosmer–Lemeshow test [125]. Calibration at a single point in this manner is not particularly useful as a model may be well-calibrated at one time-point and then poorly calibrated at all others [113]. To overcome this one could perform the Hosmer–Lemeshow test (or any other applicable test) multiple times at different values of $\tau^* \in \mathbb{R}_{\geq 0}$. However doing so is inefficient and can lead to problems with 'multiple testing'; hence these single-point methods are not considered further.

4.5.1.2. Houwelingen's α

Methods that evaluate entire distributions based on a single point may be more useful as conclusions can be drawn at the distribution level. One such method is termed here 'Houwelingen's α'. van Houwelingen proposed several measures [308] for calibration but only one generalises to all probabilistic survival models. This method evaluates the predicted cumulative hazard function, $\zeta_i.H$ (for some predicted distribution ζ_i), by comparing $\zeta_i.H$ to the 'true' hypothetical cumulative hazard, H. The test statistic, H_α, is defined by

$$H_\alpha := \frac{\sum_i H_i(T_i^*)}{\sum_i \zeta_i.H(T_i^*)} \approx \frac{\sum_i \Delta_i^*}{\sum_i \zeta_i.H(T_i^*)} \tag{4.5.4}$$

where $\zeta = (\zeta_1, ..., \zeta_m)$ are predicted distributions and $\{(T_1^*, \Delta_1^*), ..., (T_m^*, \Delta_m^*)\} \overset{i.i.d.}{\sim} (T, \Delta)$ is some test data. The model is therefore well-calibrated if $H_\alpha = 1$. This has standard error $SE(H_\alpha) = \exp(1/\sqrt{(\sum_i \Delta_i^*)})$.

The approximate equality is motivated by formulating survival data as a counting process and noting that in this setting the cumulative hazard function can estimate the number of events in a time-period [126]. No study could be found that utilised H_α for model comparison, possibly because graphical methods are favoured. This method can infer results about the calibration of an entire model and not just at a single point because the measure is calculated at a meaningful time (the event time) and utilises known results from counting processes to verify if the expected number of deaths equals the observed number of deaths.

However, as with the reduction method, the statistic is derived from a single point (the observed event time) for each individual and thus it is possible that the model is well-calibrated only for making predictions at the event time, but not over the full $\mathbb{R}_{>0}$ range.

4.5.2. Probabilistic Calibration

Unlike other areas of evaluation, graphical methods are favoured in calibration and possibly more so than numerical ones. Graphical methods compare the average predicted distribution to the expected distribution. As the expected distribution is itself unknown, this is often estimated with the Kaplan-Meier curve.

4.5.2.1. Kaplan-Meier Comparison

The simplest graphical comparison compares the average predicted survival curve to the Kaplan-Meier curve estimated on the testing data. Formally, let $\zeta_1.S, ..., \zeta_m.S$ be predicted survival functions, then the average predicted survival function is a mixture of these distributions, $\frac{1}{m}\sum_{i=1}^{m}\zeta_i.S(\tau)$. Plotting this mixture and the Kaplan-Meier on τ vs $S(\tau)$ allows a visual comparison of how closely these curves align. An example is given in fig. 10, the Cox model (CPH) is well-calibrated as it almost perfectly overlaps the Kaplan-Meier estimator, whereas predictions from the poorly-calibrated support vector machine (SVM) are far from this line.

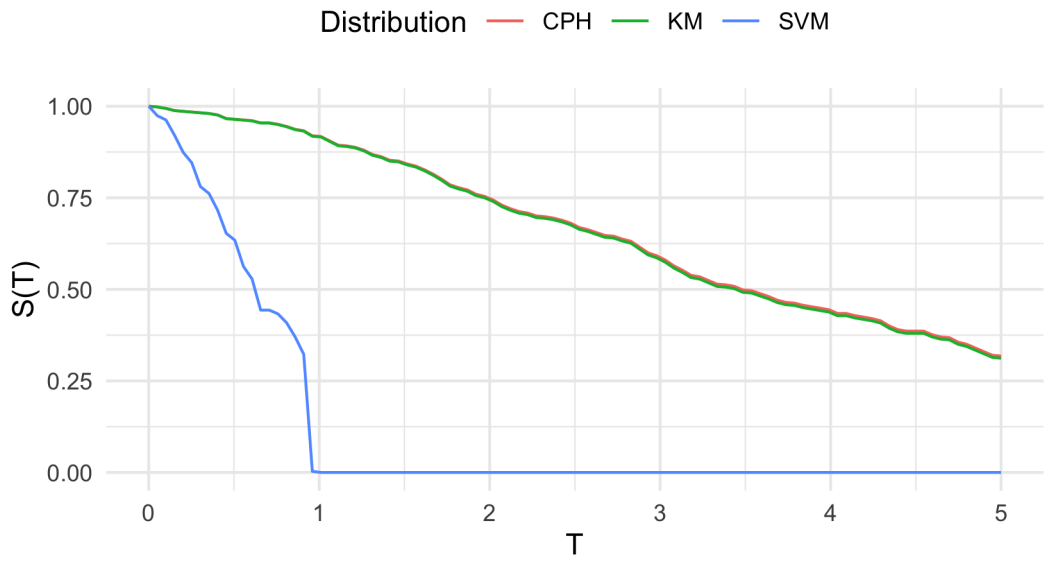


Figure 10: Assessing the calibration of a Cox PH (CPH) and SVM (with distribution composition by PH form and Kaplan-Meier (section 5.4.1)) by comparing the average survival prediction to a Kaplan-Meier (KM) estimate on the testing dataset. x-axis is time and y-axis is the predicted survival functions evaluated over time. The CPH (red line) is said to be well-calibrated as it almost perfectly overlaps the Kaplan-Meier (green line), whereas the SVM (blue line) is far from this line. Models trained and tested on randomly simulated data from the **simsurv** [35] package in **mlr3proba** (section 6.4).

This approach is both simple and interpretable. In the example above one can conclude: on average, the trained Cox PH predicts a distribution just as well as (or very close to) an unconditional estimator using the real test data. A major caveat is that conclusions are at an average *population* level with no individual-level measurement.

In order to capture finer information on a level closer to inidivduals, calibration can be applied to the predicted relative risks or linear predictor. One such approach is to bin the predictions to create different 'risk groups' from low-to-high risk [254]. These groups are then plotted against a stratified Kaplan-Meier estimator. This allows for a more nuanced approach to calibration and can simultaneously visualise a model's discrimination. However this method is far less transparent as it adds even more subjectivity around how many risk groups to create and how to create them [254].

4.5.2.2. D-Calibration

D-Calibration [8, 113] is a very recent method that aims to evaluate a model's calibration at all time-points in a predicted survival distribution. The D-calibration measure is identical to the χ^2 test-statistic, which is usually written as follows

$$\chi^2 := \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i} \tag{4.5.5}$$

where $O_1, ..., O_n$ is the observed number of events in n groups and $E_1, ..., E_n$ is the expected number of events. The statistic is utilised to determine if the underlying distribution of the observed events follows a theoretical/expected distribution.

The D-Calibration measure tests if predictions (observations) from the survival functions of predicted distributions, $\zeta_1.S, ..., \zeta_m.S$, follow the uniform distribution as expected. The following lemma motivates this test.

Lemma 4.5.1. *Let ζ be a continuous probability distribution and let $X \sim \zeta$ be a random variable. Let S_X be the survival function of X. Then $S_X(X) \sim \mathcal{U}(0, 1)$.*

Proof. Proof, in appendix C, follows by transformation of random variables $Y := S_X(X)$. □

In order to utilise the χ^2 test (for categorical variables), the $[0, 1]$ codomain of $\zeta_i.S$ is cut into B disjoint contiguous intervals ('bins') over the full range $[0, 1]$. Let m be the total number of observations in the test data. Then assuming a discrete uniform distribution as the theoretical distribution, the expected number of events is m/B.

The observed number of events in bin i, O_i, is defined as follows: Define b_i as the set of observations that die in the ith bin, formally defined by $b_i := \{j \in 1, ..., m : \lceil \zeta_j.S(T_j^*)B \rceil = i\}$, where $j = 1, ..., m$ are the indices of the test observations and $\zeta = (\zeta_1, ..., \zeta_m)$ are predicted distributions.[1] Then, $O_i = |b_i|, \forall i \in 1, ..., B$.

[1] This is a slightly simplified procedure which omits handling of censoring, but this is easily extended in the full algorithm, see Algorithm 2 of Haider *et al.* (2020) [113].

The D-Calibration measure, or χ^2 statistic, is now defined by,

$$D_{\chi^2}(\zeta, T^*) := \frac{\sum_{i=1}^{B}(O_i - \frac{m}{B})^2}{m/B} \tag{4.5.6}$$

This measure has several useful properties. Firstly, a p-value can be derived from χ^2_{B-1} to hypothesis test if a single model is 'D-calibrated'. Secondly, as a model is increasingly well-calibrated it holds that $D_{\chi^2} \to 0$ (as the number of observed events approach expected events), which motivates utilising the test for model comparison. Thirdly, the theory lends itself very nicely to an intuitive graphical calibration method:

If a model is D-calibrated, i.e. predicted distributions from the model result in a low D-calibration, then one expects,

$$p = \frac{\sum_i \mathbb{I}(T_i^* \leq \zeta_i.F^{-1}(p))}{|T^*|} \tag{4.5.7}$$

where $p \in [0, 1]$ and $\zeta_i.F^{-1}$ is the inverse cumulative distribution function of the ith predicted distribution. In words, if a model is D-calibrated then the number of deaths occurring at or before each quantile should be equal to the quantile itself, for example 50% of deaths should occur before their predicted median survival time. Therefore one can graphically test for D-calibration by plotting p on the x-axis and the RHS of eq. (4.5.7) on the y-axis. A D-calibrated model should result in a straight line on $x = y$. This is visualised in fig. 11 for the same models as in fig. 10. Again the SVM is terribly-calibrated but the CPH is better calibrated. In this case it is clearer that the D-calibration of the CPH is not perfect, especially at higher quantiles. Comparison to χ^2_9 indicates the CPH is D-calibrated whereas the SVM is not.

4.5.2.3. Transparency and Accessibility

It has already been stated that performance cannot be considered for calibration measures however it is unclear if any of these measures are even accessible or transparent as they often require expert interpretation to prevent erroneous conclusions. This is demonstrated by example using the same data and models as in fig. 11. The predictions from these models are evaluated with Harrell's C (section 4.4.1), the Integrated Graf Score (section 4.6.3), D-Calibration, and Houwelingen's α (table 6). All measures agree that the SVM performs poorly. In contrast, whilst the Cox PH (CPH) is well-calibrated according to both measures, its concordance is quite bad (barely above baseline). Haider *et al.* [113] claimed that if a model is D-Calibrated then a 'patient should believe the prediction from the survival curve', these results clearly demonstrate otherwise. Measures of calibration alone are clearly not sufficient to determine if a survival curve prediction should be 'believed' and should therefore be computed alongside measures of discrimination or scoring rules, discussed next.

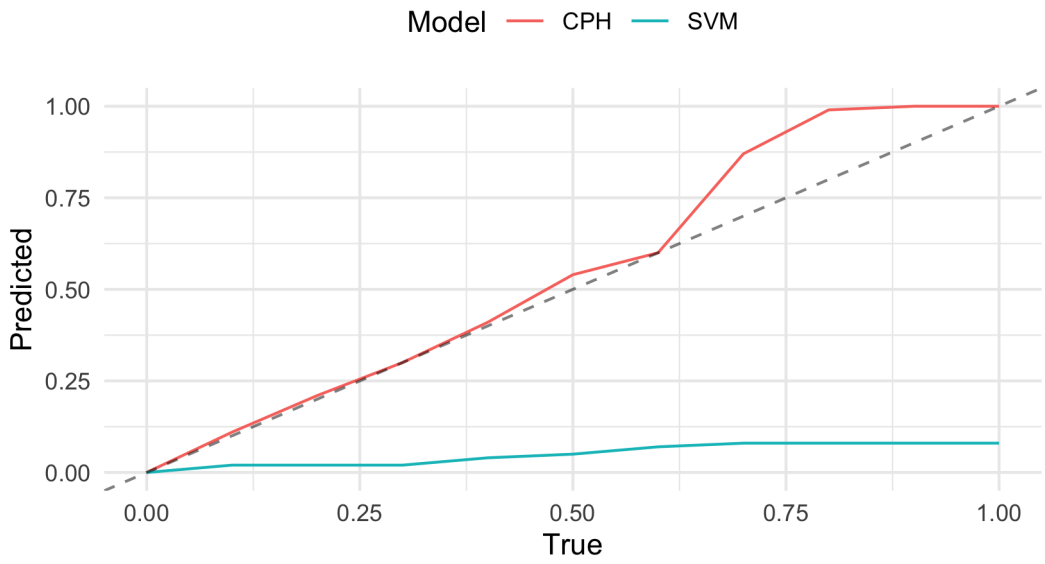


Figure 11: Assessing the D-calibration of the Cox PH (CPH) and SVM from the same data as fig. 10: models trained and tested on randomly simulated data from the **simsurv** [35] package in **mlr3proba** (section 6.4). x-axis are quantiles in $[0, 1]$ and y-axis are predicted quantiles from the models. The dashed line is $y = x$. Again the SVM is terribly calibrated and the CPH is better calibrated as it is closer to $y = x$.

4.6. Evaluating Distributions by Scoring Rules

Scoring rules evaluate probabilistic predictions and (attempt to) measure the overall predictive ability of a model, i.e. both calibration and discrimination [100, 220]. Scoring rules have been gaining in popularity for the past couple of decades since probabilistic forecasts were recognised to be superior than deterministic predictions for capturing uncertainity in predictions [63, 64]. Formalisation and development of scoring rules has primarily been due to Dawid [63, 64, 65] and Gneiting and Raftery [100]; though the earliest measures promoting "rational" and "honest" decision making date back to the 1950s [34, 106]. Whilst several scoring rules have been proposed for classification problems, fewer exist for probabilistic regression predictions [100] and even fewer for survival analysis. In practice, only three continuous scoring rules for regression are employed (though the last two of these are often conflated), the integrated Brier score [34], the log loss [106], and the integrated log loss.[1] In survival analysis only one scoring rule was found to be routinely employed. In fact, there is no recognised definition of a scoring rule in survival analysis, nor definitions for the fundamental scoring rule properties of (strict) properness. This section attempts to fill these gaps and to explore the proposed scoring rules for survival analysis.[2]

[1] These often appear under many different names. The Brier score is often referred to as the 'squared-error loss', or 'quadratic score', and the log loss often appears as the 'log score', 'logarithmic loss', 'cross-entropy loss', or 'negative log-likelihood'.

[2] In this section a 'scoring rule' refers to the general class of measures that evaluate a probabilistic prediction and a 'loss' refers to the specific function to be minimised. As all scoring rules are optimally minimised in this survey, the terms are used interchangeably.

Table 6: Comparison of numerical calibration metrics. Same models and data as in fig. 10: models trained and tested on randomly simulated data from the **simsurv** [35] package in **mlr3proba** (section 6.4).

Model	KM	CPH	SVM
C_H^1	0.5	0.52	0.45
L_{IGS}^2	0.18	0.18	0.52
H_α^3	0.99	1.00	15.42
$D_{\chi^2}^4$	2.23^*	7.03^*	1.02×10^{10}

1. Harrell's C (section 4.4.1).
2. Integrated Graf Score (section 4.6.3).
3. Houwelingen's α (section 4.5.1).
4. D-Calibration statistic. A '$*$' indicates the model is D-Calibrated according to a χ_9^2 test.

This survey of survival scoring rules covers: i) basic definitions for scoring rules and properties; ii) proposed scoring rules for survival analysis; iii) proofs for (strict) properness; and iv) baselines and standard errors for scoring rules. Key contributions include demonstrating that no commonly-utilised survival scoring rule is proper and deriving a class of strictly proper outcome-independent scoring rules with strict assumptions (see section 4.6.2 for definitions and section 4.6.4 for proofs).

Each of these subsections is built up in complexity, starting with binary classification, then probabilistic regression, and finally survival. This is required to demonstrate how the survival setting makes use of the other two for scoring rules.

To recap the notation from chapter 2, the three mathematical settings are defined by the generative processes:

- Regression: (X, Y) t.v.i. $\mathcal{X} \times \mathcal{Y}$ where $\mathcal{X} \subseteq \mathbb{R}^p$ and $\mathcal{Y} \subseteq \mathbb{R}$.
- Classification: (X, Y) t.v.i. $\mathcal{X} \times \mathcal{Y}$ where $\mathcal{X} \subseteq \mathbb{R}^p$ and $\mathcal{Y} = \{0, 1\}$.
- Survival: (X, T, Δ, Y, C) t.v.i. $\mathcal{X} \times \mathcal{T} \times \{0, 1\} \times \mathcal{T} \times \mathcal{T}$ where $X \subseteq \mathbb{R}^p$ and $\mathcal{T} \subseteq \mathbb{R}_{\geq 0}$, where C, Y are unobservable, $T := \min\{Y, C\}$, and $\Delta = \mathbb{I}(Y = T)$.

As the sections are clearly separated, the overloaded notation will be clear from context.

4.6.1. Classification and Regression Scoring Rules

Definitions and losses in the classification setting are first discussed and then the same in the regression setting.

4.6.1.1. Classification

All scoring rules were initially derived from the binary classification setting, in this case scoring rules are considered to have the form in box 4.

Box 4 (Binary classification loss)**.** Let $\mathcal{P}$ be some family of distributions over $\mathcal{Y} = \{0, 1\}$ containing at least two elements. Then for a predicted distribution in $\mathcal{P}$, any real-valued function with the signature $L : \mathcal{P} \times \mathcal{Y} \to \bar{\mathbb{R}}$ will be considered as a *binary classification loss*.

Any arbitrary function can be a binary classification loss as long as it satisfies the conditions in box 4, for example $L(\zeta, y) = 0$ is a valid loss for all $\zeta \in \mathcal{P}$ and all $y \in \mathcal{Y}$. Therefore a scoring rule is generally only considered useful if it satisfies the properties below [100].

Definition 4.6.1. A classification loss $L : \mathcal{P} \times \mathcal{Y} \to \bar{\mathbb{R}}$ is called:

i) *Proper* if: for any distributions p_Y, p in $\mathcal{P}$ and for any random variables $Y \sim p_Y$, it holds that

$$\mathbb{E}[L(p_Y, Y)] \leq \mathbb{E}[L(p, Y)] \tag{4.6.1}$$

ii) *Strictly proper* if in addition to being proper it holds, for the same quantification of variables, that

$$\mathbb{E}[L(p_Y, Y)] = \mathbb{E}[L(p, Y)] \Leftrightarrow p = p_Y \tag{4.6.2}$$

Proper scoring rules provide a method of model comparison as, by definition, predictions closest to the true distribution will result in lower expected losses.[1] On the other hand, if a scoring rule is not proper ('improper' [100]) then it has no meaningful comparison as it is unknown if the optimal model would have a lower or higher loss than any sub-optimal one. A strictly proper scoring rule has additional important uses such as in model optimisation, i.e. if a loss is strictly proper then minimisation of the loss will result in the 'optimum score estimator based on the scoring rule' [100]. Whilst properness is usually a minimal acceptable property for a scoring rule, it is generally not sufficient on its own. For example, take the following classification loss,

$$L : \mathcal{P} \times \mathcal{Y} \to \bar{\mathbb{R}}; \quad (\zeta, y) \mapsto 42 \tag{4.6.3}$$

This is proper as the loss, L, is always equal to 42 and therefore is minimised by the true distribution of Y but the loss is clearly useless. Properness and strict properness properties are utilised to determine if a scoring rule is performant and will be stated (if previously proved/disproved) or proved/disproved for all losses going forward.

Losses The two most widely used scoring rules for classification are the Brier score [34] and log loss [106].[2]

[1] Further details for model comparison are not provided here as the topic is complex and with many open questions, see e.g. [67, 69, 223].

[2] Despite being called a 'score', the Brier score is in fact a loss to be minimised.

The (binary classification) log loss is defined by

$$\begin{aligned} &L_{LL} : \mathcal{P} \times \mathcal{Y} \to \mathbb{R}_{\geq 0}; \\ &(\zeta, y) \mapsto -\mathbb{I}(y = 1) \log(\zeta.p(1)) - \mathbb{I}(y = 0) \log(\zeta.p(0)) \end{aligned} \tag{4.6.4}$$

or more simply

$$(\zeta, y) \mapsto -\log \zeta.p(y) \tag{4.6.5}$$

The (binary classification) Brier score is defined by

$$L_{BS} : \mathcal{P} \times \mathcal{Y} \to [0, 1]; \quad (\zeta, y) \mapsto (y - \zeta.p(y))^2 \tag{4.6.6}$$

These are both strictly proper scoring rules [65] and are visualised in fig. 12 to demonstrate their properties. The figure highlights the 'honesty' property of the scoring rules (i.e. their strict properness) as both losses are shown to be minimised when the true prediction is made. The plot also demonstrates baselines for interpretability (section 4.6.5.1). For the Brier score and log loss, any result below 0.25 and 0.693 respectively indicates a prediction better than a constant uninformed prediction of $\zeta.p(1) = 0.5$. Therefore classification scoring rules provide a method to simultaneously encourage honest predictions and have in-built informative baselines for external reference.

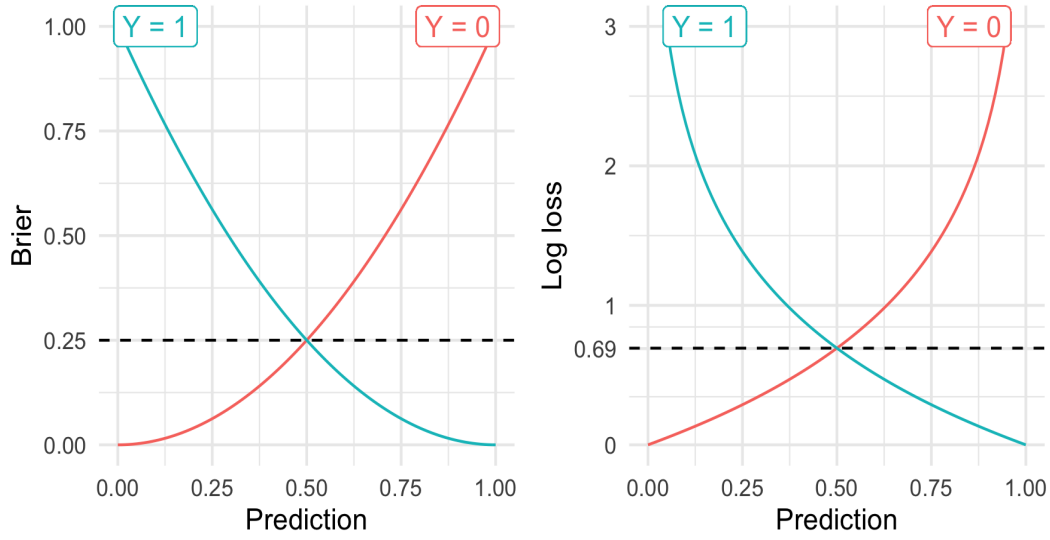


Figure 12: Brier and log loss scoring rules for a binary outcome and varying probabilistic predictions. x-axis is a probabilistic prediction in $[0, 1]$, y-axis is Brier score (left) and log loss (right). Blue lines are varying Brier score/log loss over different predicted probabilities when the true outcome is 1. Red lines are varying Brier score/log loss over different predicted probabilities when the true outcome is 0. Both losses are minimised with the correct prediction, i.e. if $\zeta.p(1) = 1$ when $y = 1$ and $\zeta.p(1) = 0$ when $y = 0$ for a predicted discrete distribution ζ.

4.6.1.2. Regression

The definition of a probabilistic regression scoring rule follows similarly to the classification setting after a re-specification of the target domain.

Box 5 (Probabilistic regression loss)**.** Let $\mathcal{P}$ be some family of distributions over $\mathcal{Y} \subseteq \mathbb{R}$ containing at least two elements. Then for a predicted distribution in $\mathcal{P}$, any real-valued function with the signature $L : \mathcal{P} \times \mathcal{Y} \to \bar{\mathbb{R}}$ will be considered as a *probabilistic regression loss*.

Definition 4.6.2. A probabilistic regression loss $L : \mathcal{P} \times \mathcal{Y} \to \bar{\mathbb{R}}$ is called:

i) *Proper* if: for any distributions p_Y, p in $\mathcal{P}$ and for any random variables $Y \sim p_Y$, it holds that

$$\mathbb{E}[L(p_Y, Y)] \leq \mathbb{E}[L(p, Y)] \tag{4.6.7}$$

ii) *Strictly proper* if in addition to being proper it holds, for the same quantification of variables, that

$$\mathbb{E}[L(p_Y, Y)] = \mathbb{E}[L(p, Y)] \Leftrightarrow p = p_Y \tag{4.6.8}$$

Losses In the regression setting, classification scoring rules are extended by instead considering distribution functions and integrating these over $\mathcal{Y} \subseteq \mathbb{R}$.

The Integrated Brier Score (IBS) is defined by,[1]

$$L_{IBS} : \mathcal{P} \times \mathcal{Y} \to [0, 1]; \quad (\zeta, y) \mapsto \int_{\mathcal{Y}} (\mathbb{I}(y \leq \tau) - \zeta.F(\tau))^2 \, d\tau \tag{4.6.9}$$

The extension from the classification Brier score is intuitive, instead of evaluating if the predicted pmf is 'correct' at a single point, the predicted cumulative distribution function is compared with the true event status over the entire distribution.

The log loss has two adaptations for continuous predictions. The first is analogous to the IBS and is termed the Integrated Log Loss (ILL)

$$\begin{aligned} &L_{ILL} : \mathcal{P} \times \mathcal{Y} \to \mathbb{R}_{\geq 0}; \\ &(\zeta, y) \mapsto -\int_{\mathcal{Y}} \mathbb{I}(y \leq \tau) \log[\zeta.F(\tau)] + \mathbb{I}(y > \tau) \log[\zeta.S(\tau)] \, d\tau \end{aligned} \tag{4.6.10}$$

This follows the 'longer' form of the binary classification log loss and considers the cumulative probability of events over all time-points. A second adaptation to the log loss instead considers the 'simpler' form and replaces the probability mass function with the probability density function. Again this measure is intuitive as a perfect distributional prediction will assign the highest point of density to

[1] also known as the Continuous Ranked Probability Score (CRPS).

the point at which the event occurs. This variant of the log loss does not have a specific name but it is termed here the 'density log loss', L_{DLL}, and is formally defined by,

$$L_{DLL} : \mathcal{P} \times \mathcal{Y} \to \mathbb{R}_{\geq 0}; \quad (\zeta, y) \mapsto -\log[\zeta.f(y)] \tag{4.6.11}$$

where $\mathcal{P}$ is a family of absolutely continuous distributions over $\mathcal{Y}$ with defined density functions.

All three of these losses are strictly proper [100, 111].

4.6.2. Survival Scoring Rule Definitions

Losses in the survival setting compare predicted survival distributions to the observed outcome tuple (time and censoring). A large class of survival losses additionally incorporate an estimator of the unknown censoring distribution, in order to attempt meaningful comparison. This second group of losses are termed here as 'approximate' losses as the true censoring distribution is never known and hence an estimate of the loss is approximate at best.

Box 6 (Survival loss)**.** Let $\mathcal{T} \subseteq \mathbb{R}_{\geq 0}$ and let $\mathcal{C}, \mathcal{P}$ be any two distinct families of distributions over $\mathcal{T}$, containing at least two elements. Then,

- Any real-valued function with the signature $L : \mathcal{P} \times \mathcal{T} \times \{0, 1\} \to \bar{\mathbb{R}}$ will be considered as a *survival loss*.
- Any real-valued function with the signature $L : \mathcal{P} \times \mathcal{T} \times \{0, 1\} \times \mathcal{C} \to \bar{\mathbb{R}}$ will be considered as an *approximate survival loss*.

Two separate novel definitions for (strict) properness are provided: the first captures the general case in which no assumptions are made about the censoring distribution; the second assumes that censoring is conditionally event-independent.

Definition 4.6.3. A survival loss $L : \mathcal{P} \times \mathcal{T} \times \{0, 1\} \to \bar{\mathbb{R}}$ is called:

i) *Proper* if: for any distributions p_Y, p in $\mathcal{P}$; and for any random variables $Y \sim p_Y$, and C t.v.i. $\mathcal{T}$; with $T := \min\{Y, C\}$ and $\Delta := \mathbb{I}(T = Y)$; it holds that,

$$\mathbb{E}[L(p_Y, T, \Delta)] \leq \mathbb{E}[L(p, T, \Delta)] \tag{4.6.12}$$

ii) *Strictly proper* if in addition to being proper it holds, for the same quantification of variables, that

$$\mathbb{E}[L(p_Y, T, \Delta)] = \mathbb{E}[L(p, T, \Delta)] \Leftrightarrow p = p_Y \tag{4.6.13}$$

iii) *Outcome-independent proper* if: for any distributions p_Y, p in $\mathcal{P}$; and for any random variables $Y \sim p_Y$, and C t.v.i. $\mathcal{T}$, where $C \perp\!\!\!\perp Y$; with $T := \min\{Y, C\}$ and $\Delta := \mathbb{I}(T = Y)$; it holds that,

$$\mathbb{E}[L(p_Y, T, \Delta)] \leq \mathbb{E}[L(p, T, \Delta)] \tag{4.6.14}$$

iv) *Outcome-independent strictly proper* if in addition to being outcome-independent proper it holds, for the same quantification of variables, that

$$\mathbb{E}[L(p_Y, T, \Delta)] = \mathbb{E}[L(p, T, \Delta)] \Leftrightarrow p = p_Y \tag{4.6.15}$$

These final two definitions are 'weaker' but provide a term for losses that are improper in general but are (strictly) proper under common (though possibly strict) assumptions about the censoring distribution. Note by definition that if a loss is:

i) (strictly) proper then it is also outcome-independent (strictly) proper;

ii) (outcome-independent) strictly proper then it is also (outcome-independent) proper

Analogous definitions are now provided for approximate survival losses.

Definition 4.6.4. An approximate survival loss $L : \mathcal{P} \times \mathcal{T} \times \{0, 1\} \times \mathcal{C} \to \bar{\mathbb{R}}$ is called:

i) *Proper* if: for any distributions p_Y, p in $\mathcal{P}$ and $c \in \mathcal{C}$; and for any random variables $Y \sim p_Y$ and $C \sim c$; with $T := \min\{Y, C\}$ and $\Delta := \mathbb{I}(T = Y)$; it holds that,

$$\mathbb{E}[L(p_Y, T, \Delta|c)] \leq \mathbb{E}[L(p, T, \Delta|c)] \tag{4.6.16}$$

ii) *Strictly proper* if in addition to being proper it holds, for the same quantification of variables, that

$$\mathbb{E}[L(p_Y, T, \Delta|c)] = \mathbb{E}[L(p, T, \Delta|c)] \Leftrightarrow p = p_Y \tag{4.6.17}$$

iii) *Outcome-independent proper* if: for any distributions p_Y, p in $\mathcal{P}$ and $c \in \mathcal{C}$; and for any random variables $Y \sim p_Y$ and $C \sim c$, where $C \perp\!\!\!\perp Y$; with $T := \min\{Y, C\}$ and $\Delta := \mathbb{I}(T = Y)$; it holds that,

$$\mathbb{E}[L(p_Y, T, \Delta|c)] \leq \mathbb{E}[L(p, T, \Delta|c)] \tag{4.6.18}$$

iv) *Outcome-independent strictly proper* if in addition to being outcome-independent proper it holds, for the same quantification of variables, that

$$\mathbb{E}[L(p_Y, T, \Delta|c)] = \mathbb{E}[L(p, T, \Delta|c)] \Leftrightarrow p = p_Y \tag{4.6.19}$$

As the true censoring distribution, c, can never be known exactly, this definition allows for approximate losses to be proper in the asymptotic (with infinite training data) if they include estimators of c that are convergent in distribution. Proper approximate losses are therefore useful in modern predictive settings in which 'big data' is very common and thus estimators, such as the Kaplan-Meier, can converge to the true censoring distribution. However approximate losses may provide misleading results when the sample size is small; future research should ascertain what 'small' means for individual losses.

4.6.3. Common Survival Scoring Rules

The IBS, ILL, and DLL are now extended to the survival setting by suitably incorporating censoring and their properness properties are then discussed in section 4.6.4. Measures are split into 'classes', which represent the basic form of the measure.

4.6.3.1. Squared Survival Losses

The analogue to the IBS for survival analysis is termed here as the Integrated Graf Score (IGS) as it was extensively discussed and promoted by Graf [108, 109].

Definition 4.6.5. The *integrated Graf score* (IGS) is defined by

$$\begin{aligned} &L_{IGS} : \mathcal{P} \times \mathcal{T} \times \{0,1\} \times \mathcal{C} \to [0,1]; \\ &(\zeta, t, \delta | \hat{G}_{KM}) \mapsto \int_0^{\tau^*} \frac{\zeta.S^2(\tau)\mathbb{I}(t \leq \tau, \delta = 1)}{\hat{G}_{KM}(t)} + \frac{\zeta.F^2(\tau)\mathbb{I}(t > \tau)}{\hat{G}_{KM}(\tau)} \, d\tau \end{aligned} \tag{4.6.20}$$

where $\zeta.S^2(\tau) = (\zeta.S(\tau))^2$, analogously for $\zeta.F^2$, and $\tau^* \in \mathcal{T}$ is an upper threshold to compute the loss up to.

The IGS consistently estimates the mean square error $L(t, S|\tau^*) = \int_0^{\tau^*} [\mathbb{I}(t > \tau) - S(\tau)]^2 d\tau$, where S is the correctly specified survival function, when censoring is uninformative only [98]. This is intuitive as the IGS utilises the marginal Kaplan-Meier estimator to estimate the censoring distribution. Therefore CIPCW estimates such as the Cox model or Akritas estimator could instead be considered for $\hat{G}_{KM}$ and these have been demonstrated to have less bias when censoring is informative [98]. However this raises concerns as now separate models have to be trained and predicted, which could need validation themselves, and therefore the final measure is even more difficult to interpret. Graf claimed that the IGS is strictly proper [109] however as no definition of properness was provided this claim cannot be validated. With the definition of properness provided in this thesis (definition 4.6.4), the IGS is not even proper (section 4.6.4.4).

One could instead consider extending the IBS by weighting by $\hat{G}_{KM}(t)$ only, giving the following loss.

Definition 4.6.6. Let $\mathcal{P}$ be a family of absolutely continuous distributions over $\mathcal{T}$ with defined density functions. Then the *reweighted Integrated Graf score* (IGS*) is defined by

$$\begin{aligned} &L_{IGS^*} : \mathcal{P} \times \mathcal{T} \times \{0,1\} \times \mathcal{C} \to \mathbb{R}_{\geq 0}; \\ &(\zeta, t, \delta | \hat{G}_{KM}) \mapsto \frac{\delta \int_{\mathcal{T}} (\mathbb{I}(t \leq \tau) - \zeta.F(\tau))^2 \, d\tau}{\hat{G}_{KM}(t)} \end{aligned} \tag{4.6.21}$$

IGS* is outcome-independent strictly proper (section 4.6.4.3).

4.6.3.2. Log Survival Losses

The ILL is similarly extended to the Integrated Survival Log Loss (ISLL) [109].

Definition 4.6.7. The *integrated survival log loss* (ISLL) is defined by

$$L_{ISLL} : \mathcal{P} \times \mathcal{T} \times \{0,1\} \times \mathcal{C} \to \mathbb{R}_{\geq 0};$$
$$(\zeta, t, \delta | \hat{G}_{KM}) \mapsto -\int_0^{\tau^*} \frac{\log[\zeta.F(\tau)]\mathbb{I}(t \leq \tau, \delta = 1)}{\hat{G}_{KM}(t)} + \frac{\log[\zeta.S(\tau)]\mathbb{I}(t > \tau)}{\hat{G}_{KM}(\tau)}\, d\tau$$

where $\tau^* \in \mathcal{T}$ is an upper threshold to compute the loss up to.

The ISLL is not a proper approximate survival loss (section 4.6.4.4). Again one could instead a different weighting in the denominator of the measure to give the following loss.

Definition 4.6.8. Let $\mathcal{P}$ be a family of absolutely continuous distributions over $\mathcal{T}$ with defined density functions. Then the *reweighted integrated survival log loss* (ISLL*) is defined by

$$L_{ISLL^*} : \mathcal{P} \times \mathcal{T} \times \{0,1\} \times \mathcal{C} \to \mathbb{R}_{\geq 0};$$
$$(\zeta, t, \delta | \hat{G}_{KM}) \mapsto -\frac{\delta \int_{\mathcal{T}} \mathbb{I}(t \leq \tau) \log[\zeta.F(\tau)] + \mathbb{I}(t > \tau) \log[\zeta.S(\tau)]\, d\tau}{\hat{G}_{KM}(t)} \tag{4.6.22}$$

ISLL* is an outcome-independent strictly proper scoring rule (section 4.6.4.3).

The DLL can be extended in one of two ways, the first simply removes all censored observations.

Definition 4.6.9. Let $\mathcal{P}$ be a family of absolutely continuous distributions over $\mathcal{T}$ with defined density functions. Then the *survival density log loss* (SDLL) is defined by

$$L_{SDLL} : \mathcal{P} \times \mathcal{T} \times \{0,1\} \to \mathbb{R}_{\geq 0}; \quad (\zeta, t, \delta) \mapsto -\delta \log[\zeta.f(t)] \tag{4.6.23}$$

The SDLL is not a proper scoring rule (section 4.6.4.2). The second extension to DLL adds the same IPC weighting as IGS* and ISLL*.

Definition 4.6.10. Let $\mathcal{P}$ be a family of absolutely continuous distributions over $\mathcal{T}$ with defined density functions. Then the *weighted survival density log loss* (SDLL*) is defined by

$$L_{SDLL^*} : \mathcal{P} \times \mathcal{T} \times \{0,1\} \times \mathcal{C} \to \mathbb{R}_{\geq 0}; \quad (\zeta, t, \delta | \hat{G}_{KM}) \mapsto -\frac{\delta \log[\zeta.f(t)]}{\hat{G}_{KM}(t)} \tag{4.6.24}$$

SDLL* is outcome-independent strictly proper (section 4.6.4.3).

4.6.3.3. Absolute Survival Losses

Whilst the IGS and ISLL appear to be the most common losses in the literature, there is one other class to briefly mention that is based on absolute error

functions. For example, the 'absolute Brier score' proposed by Schemper and Henderson [262] which is based on the mean absolute error. This takes a similar approach to the IGS and weights the loss at different time-points according to whether an observation is censored. Studies of this loss have demonstrated that it depends heavily on correct model specification and is biased when this is not the case [51, 266]. To prevent this bias, Schmid *et al.* [266] proposed the following robust approximate loss, termed here the 'Schmid score',

$$L(\zeta, t, \delta|\hat{G}_{KM}) = \int_0^{\tau^*} \frac{\zeta.S(\tau)\mathbb{I}(t \leq \tau, \delta = 1)}{\hat{G}_{KM}(t)} + \frac{\zeta.F(\tau)\mathbb{I}(t > \tau)}{\hat{G}_{KM}(\tau)}\, d\tau \qquad (4.6.25)$$

where $\hat{G}_{KM}$ and τ^* are as defined above. Analogously to the IGS, the Schmid score consistently estimates the mean absolute error when censoring is uninformative [266]. Both scores tend to yield similar results [266].

4.6.3.4. Comparing Weighting Methods

The IGS and ISLL are well-established survival losses however no discussion about IGS* and ISLL* could be found in the literature. On the surface these measures may look very similar but there are two important differences, which are illustrated below with the ISLL and ISLL*, recall these are defined as:

$$L_{ISLL^*}(\zeta, t, \delta|\hat{G}_{KM}) = -\int_0^{\tau^*} \frac{\log[\zeta.F(\tau)]\mathbb{I}(t \leq \tau, \delta = 1)}{\hat{G}_{KM}(t)} + \frac{\log[\zeta.S(\tau)]\mathbb{I}(t > \tau, \delta = 1)}{\hat{G}_{KM}(t)}\, d\tau$$

$$L_{ISLL}(\zeta, t, \delta|\hat{G}_{KM}) = -\int_0^{\tau^*} \frac{\log[\zeta.F(\tau)]\mathbb{I}(t \leq \tau, \delta = 1)}{\hat{G}_{KM}(t)} + \frac{\log[\zeta.S(\tau)]\mathbb{I}(t > \tau)}{\hat{G}_{KM}(\tau)}\, d\tau$$

The primary differences are (RHS of equations):

i) Always removing censored observations from L_{ISLL^*} (even when alive) whereas L_{ISLL} includes all observations when alive.

ii) L_{ISLL^*} weights alive and dead observations by $\hat{G}_{KM}(t)$ whereas L_{ISLL} weights dead observations by $\hat{G}_{KM}(t)$ and alive observations by $\hat{G}_{KM}(\tau)$

Analytically the difference between these weighting results has major implications as L_{ISLL^*} (and L_{IGS^*}) is outcome-independent strictly proper (section 4.6.4.3) whereas L_{ISLL} (and L_{IGS}) is not even proper (section 4.6.4.4). However whilst it has been demonstrated that the IGS consistently estimates the mean squared error [98], no theory exists for IGS*. Similarly no study has been made on ISLL* and SDLL*.

4.6.3.5. PECs

As well as evaluating probabilistic outcomes with integrated scoring rules, non-integrated scoring rules can also be utilised for evaluating distributions at a single point. For example, instead of evaluating a probabilistic prediction with the IGS over $\mathbb{R}_{\geq 0}$, instead one could compute the IGS at a single time-point, $\tau \in \mathbb{R}_{\geq 0}$,

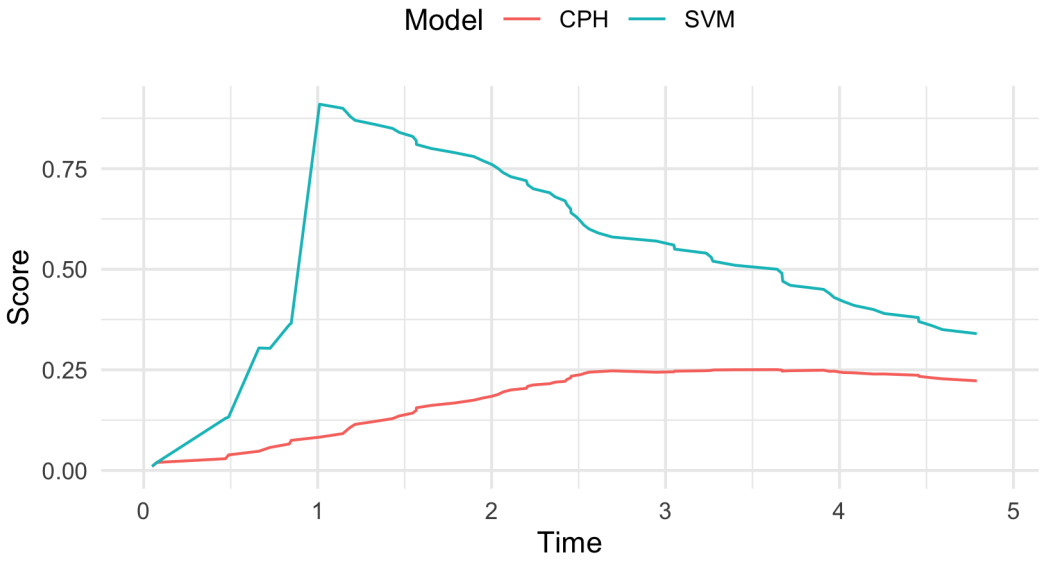


Figure 13: Prediction error curves for the CPH and SVM models from section 4.5. x-axis is time and y-axis is the IGS computed at different time-points. The CPH (red) performs better than the SVM (blue) as it scores consistently lower. Trained and tested on randomly simulated data from **mlr3proba**.

only. Plotting these for varying values of τ results in 'prediction error curves' (PECs), which provide a simple visualisation for how predictions vary over the outcome. PECs are especially useful for survival predictions as they can visualise the prediction 'over time'. PECs should only be used as a graphical guide and never for model comparison as they only provide information at a limited number of points. An example is provided in fig. 13 for the IGS; the CPH is consistently better performing than the SVM.

4.6.4. Properness of Survival Scoring Rules

As the IBS, ILL, and DLL are all strictly proper regression losses, one may assume the analogous survival losses are also strictly proper. No arguments could be found proving/disproving properness of the survival losses, which may be due to researchers assuming properness followed from the regression setting. Despite these estimators being demonstrated to have useful properties and to 'perform well' in simulation experiments [50, 51, 98], it transpires that none are proper. Key results in this section are collected in the following summary theorem.

Theorem 4.6.1. *Let $\mathcal{T} \subseteq \mathbb{R}_{>0}$ and let $\mathcal{C}, \mathcal{P}$ be two distinct families of distributions over $\mathcal{T}$ containing at least two elements and let $L_R : \mathcal{P} \times \mathcal{T} \to \bar{\mathbb{R}}$ be a regression scoring rule. Then the following statements are true:*

i) L_{SDLL} is not: a) outcome-independent proper; b) outcome-independent strictly proper; c) proper; d) strictly proper (proposition 4.6.8).

ii) Define the approximate survival loss,

$$L_S : \mathcal{P} \times \mathcal{T} \times \{0,1\} \times \mathcal{C} \to \bar{\mathbb{R}}; \quad (\zeta, t, \delta | \hat{G}_{KM}) \mapsto \frac{\delta L_R(\zeta, t)}{\hat{G}_{KM}(t)} \tag{4.6.26}$$

Then L_S is outcome-independent strictly proper if and only if L_R is strictly proper (theorem 4.6.10).

iii) $L_{SDLL^*}, L_{IGS^*}, L_{ISLL^*}$ are all outcome-independent strictly proper (proposition 4.6.11).

iv) L_{IGS} is not: a) outcome-independent proper; b) outcome-independent strictly proper; c) proper; d) strictly proper (proposition 4.6.12).

v) L_{ISLL} is not: a) outcome-independent proper; b) outcome-independent strictly proper; c) proper; d) strictly proper (proposition 4.6.13).

The following conjectures are also made:

i) No survival loss, $L : \mathcal{P} \times \mathcal{T} \times \{0,1\} \to \bar{\mathbb{R}}$, is: a) outcome-independent strictly proper; b) strictly proper (conjecture 4.6.9).

ii) No approximate survival loss, $L : \mathcal{P} \times \mathcal{T} \times \{0,1\} \times \mathcal{C} \to \bar{\mathbb{R}}$, is strictly proper (conjecture 4.6.14).

4.6.4.1. Definitions and Lemmas

Important proofs in this subsection follow after these definitions and lemmas.

Lemma 4.6.2. *Let $L : \mathcal{P} \times \mathcal{T} \times \{0,1\} \to \bar{\mathbb{R}}$ be a survival loss. Let $p_Y \in \mathcal{P}$, let $Y \sim p_Y$ and C t.v.i. $\mathcal{T}$ be random variables where $C \perp\!\!\!\perp Y$. Let $T := \min\{Y, C\}$ and $\Delta := \mathbb{I}(T = Y)$. Then if $\exists p \in \mathcal{P}, p \neq p_Y$, such that*

$$\mathbb{E}[L(p_Y, T, \Delta)] > \mathbb{E}[L(p, T, \Delta)] \tag{4.6.27}$$

Then, L is not:

i) outcome-independent proper;

ii) outcome-independent strictly proper;

iii) proper;

iv) strictly proper.

Proof. Proofs, in appendix C, follow trivially by definition. □

Lemma 4.6.3. *Let $L : \mathcal{P} \times \mathcal{T} \times \{0,1\} \times \mathcal{C} \to \bar{\mathbb{R}}$ be an approximate survival loss. Let $p_Y \in \mathcal{P}$ and let $c \in \mathcal{C}$. Let $Y \sim p_Y$ and C t.v.i.$\mathcal{T}$ be random variables. Let $T := min\{Y, C\}$ and $\Delta := \mathbb{I}(T = Y)$. Then if $\exists p \in \mathcal{P}, p \neq p_Y$, such that*

$$\mathbb{E}[L(p_Y, T, \Delta | c)] > \mathbb{E}[L(p, T, \Delta | c)] \tag{4.6.28}$$

Then: L is not,

i) proper;

ii) strictly proper.

Proof. Proofs, in appendix C, follow trivially by definition. □

Definition 4.6.11. Let $L : \mathcal{P} \times \mathcal{T} \times \{0,1\} \to \bar{\mathbb{R}}$ be a proper scoring rule and let p, p_Y be distributions in $\mathcal{P}$. Let $Y \sim p_Y$ and C t.v.i. $\mathcal{T}$ be random variables and let $T := \min\{Y, C\}$ and $\Delta := \mathbb{I}(T = Y)$. Then, [100]

i) $S_L(p_Y, p) := \mathbb{E}[L(p, T, \Delta)]$ is defined as the *expected penalty.*

ii) $H_L(p_Y) := S_L(p_Y, p_Y)$ is defined as the *(generalised) entropy of* $p_Y \in \mathcal{P}$.

iii) $D_L(p_Y, p) := S_L(p_Y, p) - H_L(p_Y)$ is defined as the *discrepancy* or *divergence* of $p \in \mathcal{P}$ from $p_Y \in \mathcal{P}$.

Similar definitions follow for the expected penalty, entropy, and divergence for an approximate survival loss $L : \mathcal{P} \times \mathcal{T} \times \{0,1\} \times \mathcal{C} \to \bar{\mathbb{R}}$.

Lemma 4.6.4. *Let $L : \mathcal{P} \times \mathcal{T} \times \{0,1\} \to \bar{\mathbb{R}}$ be a survival loss and let p_Y be a distribution in $\mathcal{P}$. Let $Y \sim p_Y$ and C t.v.i. $\mathcal{T}$ be random variables and let $T := \min\{Y, C\}$ and $\Delta := \mathbb{I}(T = Y)$. Then,*

- *$D_L(p_Y, p) \geq 0$ for all $p \in \mathcal{P}$ if L is proper*
- *$D_L(p_Y, p) > 0$ iff L is strictly proper and $p \neq p_Y$*

Proof. Proofs, in appendix C, follow trivially by definition. □

Definition 4.6.12. Let X be an absolutely continuous random variable and let Y be a discrete random variable. Then,

i) The *mixed joint density* of (X, Y) is defined by

$$f_{X,Y}(x, y) = f_{X|Y}(x|y)P(Y = y) \tag{4.6.29}$$

where $f_{X|Y}(x|y)$ is the conditional probability density function of X given $Y = y$.

ii) The *mixed joint cumulative distribution function* of (X, Y) is given by

$$F_{X,Y}(x, y) = \sum_{z \leq y} \int_{u=-\infty}^{x} f_{X,Y}(u, z)\, du \tag{4.6.30}$$

Lemma 4.6.5. *Let X, Y be jointly absolutely continuous random variables supported on the Reals with joint density function $f_{X,Y}(x, y)$ and let $Z = \mathbb{I}(X \leq Y)$, then the mixed joint density of (X, Z) is given by*

$$f_{X,Z}(x, z) = \begin{cases} \int_x^\infty f_{X,Y}(x, y)\, dy, & z = 1 \\ \int_{-\infty}^x f_{X,Y}(x, y)\, dy, & z = 0 \end{cases} \tag{4.6.31}$$

Proof. Proof, in appendix C, follows by transformation of random variables via the joint cdf. □

Corollary 4.6.6. *Let X, Y be jointly absolutely continuous random variables supported on the Reals with joint density function $f_{X,Y}(x,y)$ and let $Z = \mathbb{I}(X \leq Y)$. As a direct corollary to lemma 4.6.5, if X and Y are independent then the mixed joint density of (X,Z) is given by*

$$f_{X,Z}(x,z) = \begin{cases} f_X(x)S_Y(x), & z = 1 \\ f_X(x)F_Y(x), & z = 0 \end{cases} \tag{4.6.32}$$

Proof. Proof, in appendix C, follows from the definition of independent random variables. □

Lemma 4.6.7. *Let X, Y be jointly absolutely continuous random variables supported on the Reals with joint density function $f_{X,Y}(x,y)$ and let $Z = \mathbb{I}(X \leq Y)$, then the mixed joint density of (Y,Z) is given by*

$$f_{Y,Z}(y,z) = \begin{cases} \int_{-\infty}^{y} f_{X,Y}(x,y)\ dx, & z = 1 \\ \int_{y}^{\infty} f_{X,Y}(x,y)\ dx, & z = 0 \end{cases} \tag{4.6.33}$$

In addition if $X \perp\!\!\!\perp Y$, then

$$f_{Y,Z}(y,z) = \begin{cases} f_Y(y)F_X(y), & z = 1 \\ f_Y(y)S_X(y), & z = 0 \end{cases} \tag{4.6.34}$$

Proof. Proofs follow analogously to lemma 4.6.5 and corollary 4.6.6; further details are not provided. □

4.6.4.2. No Strictly Proper Survival Loss

First it is proved that the survival density log loss is not outcome-independent proper and then a conjecture is made on the strict properness of all non-approximate losses.

Proposition 4.6.8. *The survival density log loss is not:*

i) outcome-independent proper

ii) outcome-independent strictly proper

iii) proper

iv) strictly proper

Proof.

Proof of (i). Let $\mathcal{P}$ be a family of absolutely continuous distributions over the positive Reals with defined density functions and let ζ, ξ be some distributions in $\mathcal{P}$. Let Y be some random variable distributed according to ξ and let C be an r.v. t.v.i. $\mathcal{T}$ and let Y and C be independent. Let $T := \min\{Y, C\}$ and

$\Delta := \mathbb{I}(T = Y)$. Let $\hat{Y}$ be some random variable distributed according to ζ, independent of Y, C, T, and Δ. Finally define $q := P(\Delta = 1)$.

By lemma 4.6.4, the loss is improper if there exists some ζ, ξ such that $D_{SDLL}(\xi, \zeta) < 0$. Proof follows by demonstrating such ζ, ξ exist and therefore that the loss is improper. First calculating $S_{SDLL}(\xi, \zeta)$,

$$\begin{aligned}
S_{SDLL}(\xi, \zeta) &= \mathbb{E}\{-\Delta \log[f_{\hat{Y}}(T)]\} \\
&= q\mathbb{E}\{-\Delta \log[f_{\hat{Y}}(Y)]|\Delta = 1\} + (1-q)\mathbb{E}\{-\Delta \log[f_{\hat{Y}}(C)]|\Delta = 0\} \\
&= -q \int f_{Y|\Delta}(y|1) \log(f_{\hat{Y}}(y))\ dy \\
&= -\int f_{Y,\Delta}(y, 1) \log(f_{\hat{Y}}(y))\ dy \\
&= -\int \log(f_{\hat{Y}}(y)) \int_y^\infty f_{Y,C}(y, c)\ dc\ dy
\end{aligned}$$

Where the second equality is the law of total expectation and substituting $q = P(\Delta = 1)$, the third is definition of expectation on the LHS and the RHS follows as $\Delta = 0$, the fourth follows from definition of conditional probability, and the final from lemma 4.6.5. By similar reasoning,

$$H_{SDLL}(\xi) = -\int \log(f_Y(y)) \int_y^\infty f_{Y,C}(y, c)\ dc\ dy$$

The loss is improper if there is at least one $\zeta \neq \xi \in \mathcal{P}$ s.t. $S_{SDLL}(\xi, \zeta) < H_{SDLL}(\xi)$. Selecting such a counter-example let $\xi = \mathcal{U}(0, 3)$ and let $C \sim \xi$ so that $f_{Y,C}(y, c) = \frac{1}{9}, 0 \leq y, c \leq 3$ and $f_Y(y) = \frac{1}{3}, 0 \leq y \leq 3$. Then,

$$\begin{aligned}
H_{SDLL}(\xi) &= -\int \log(f_Y(y)) \int_y^\infty f_{Y,C}(y, c)\ dc\ dy \\
&= -\int_0^3 \log\left(\frac{1}{3}\right) \int_y^3 \frac{1}{9}\ dc\ dy \\
&= -\int_0^3 \log\left(\frac{1}{3}\right)\left(\frac{1}{3} - \frac{y}{9}\right)\ dy \\
&= -\left[\frac{y}{3}\log\left(\frac{1}{3}\right) - \frac{y^2}{18}\log\left(\frac{1}{3}\right)\right]_0^3 \\
&= -\log\left(\frac{1}{3}\right) + \frac{1}{2}\log\left(\frac{1}{3}\right) \\
&= \log(\sqrt{3})
\end{aligned}$$

Let $\zeta = \mathcal{U}(0, 2)$, then

$$\begin{aligned} S_{SDLL}(\xi, \zeta) &= -\int \log(f_{\hat{Y}}(y)) \int_y^{\infty} f_{Y,C}(y, c)\, dc\, dy \\ &= -\int_0^3 \log\left(\frac{1}{2}\right) \int_y^3 \frac{1}{9}\, dc\, dy \\ &= \log(\sqrt{2}) \end{aligned}$$

Now computing the divergence of ζ from ξ,

$$\begin{aligned} D_{SDLL}(\xi, \zeta) &= S_{SDLL}(\xi, \zeta) - H_{SDLL}(\xi) \\ &= \log(\sqrt{2}) - \log(\sqrt{3}) \\ &= -0.20 < 0 \end{aligned}$$

By lemma 4.6.4 as $D_{SDLL}(\xi, \zeta) < 0$ it follows that L_{SDLL} is improper. Furthermore, as it was assumed that $Y \perp\!\!\!\perp C$, it follows that the loss is not outcome-independent proper. ■

Proofs of (ii)-(iv). Proofs follow from (i) and lemma 4.6.2. ■

□

Not only is the L_{SDLL} not outcome-independent proper but the counter-example in the proof is not even a rare edge case. Accounting for the censoring distribution is attempted by approximate losses, which are explored after the following conjecture.

Conjecture 4.6.9. *Let $L : \mathcal{P} \times \mathcal{T} \times \{0, 1\} \to \bar{\mathbb{R}}$ be a survival loss, then L is not:*

i) outcome-independent strictly proper;

ii) strictly proper;

This conjecture is motivated by identifying that as the true censoring distribution is always unknown, a counter-example can likely always be identified to contradict the loss being strictly proper.[1]

4.6.4.3. Strictly Proper Approximate Survival Losses

By making strict assumptions about the data, some survival scoring rules can still be useful, these assumptions are:

i) survival times and censoring times are independent;

ii) the training dataset is large enough to approximate the censoring distribution

[1] This conjecture is being explored as part of a theorem in a paper with external collaborators.

With these assumptions, a large class of approximate losses can be outcome-independent strictly proper.

Theorem 4.6.10. *Let $L_R : \mathcal{P} \times \mathcal{T} \to \bar{\mathbb{R}}$ be a regression loss and define the approximate survival loss*

$$L_S : \mathcal{P} \times \mathcal{T} \times \{0,1\} \times \mathcal{C} \to \bar{\mathbb{R}}; \quad (\zeta, t, \delta | \hat{G}_{KM}) \mapsto \frac{\delta L_R(\zeta, t)}{\hat{G}_{KM}(t)} \tag{4.6.35}$$

Then L_S is outcome-independent strictly proper if and only if L_R is strictly proper.

Proof.

Proof L_S strictly proper $\Rightarrow$ L_R strictly proper. Let $\mathcal{P}$ be a family of absolutely continuous distributions over the positive Reals and let ζ, ξ be distinct distributions in $\mathcal{P}$. Let Y be some random variable distributed according to ξ and let C be an r.v. t.v.i. $\mathcal{T}$ with $Y \perp\!\!\!\perp C$. Let $T := \min\{Y, C\}$, $\Delta := \mathbb{I}(T = Y)$, and $q := P(\Delta = 1)$.

Proof follows by definition of strict properness,

$$\begin{aligned}
&\mathbb{E}[L_S(\xi, T, \Delta | \hat{G}_{KM})] < \mathbb{E}[L_S(\zeta, T, \Delta | \hat{G}_{KM})] \\
&\Rightarrow \mathbb{E}[\frac{\Delta L_R(\xi, T)}{\hat{G}_{KM}(T)}] < \mathbb{E}[\frac{\Delta L_R(\zeta, T)}{\hat{G}_{KM}(T)}] \\
&\Rightarrow q\mathbb{E}[\frac{\Delta L_R(\xi, Y)}{\hat{G}_{KM}(Y)} | \Delta = 1] + (1-q)\mathbb{E}[\frac{\Delta L_R(\xi, C)}{\hat{G}_{KM}(C)} | \Delta = 0] \\
&\qquad < q\mathbb{E}[\frac{\Delta L_R(\zeta, Y)}{\hat{G}_{KM}(Y)} | \Delta = 1] + (1-q)\mathbb{E}[\frac{\Delta L_R(\zeta, C)}{\hat{G}_{KM}(C)} | \Delta = 0] \\
&\Rightarrow q\mathbb{E}[\frac{L_R(\xi, Y)}{\hat{G}_{KM}(Y)} | \Delta = 1] < q\mathbb{E}[\frac{L_R(\zeta, Y)}{\hat{G}_{KM}(Y)} | \Delta = 1] \\
&\Rightarrow \int \frac{f_{Y|\Delta}(y|1) L_R(\xi, Y)}{\hat{G}_{KM}(y)} dy < \int \frac{f_{Y|\Delta}(y|1) L_R(\zeta, Y)}{\hat{G}_{KM}(y)} dy \\
&\Rightarrow \int \frac{f_Y(y) S_C(y) L_R(\xi, Y)}{\hat{G}_{KM}(y)} dy < \int \frac{f_Y(y) S_C(y) L_R(\zeta, Y)}{\hat{G}_{KM}(y)} dy \\
&\Rightarrow \int f_Y(y) L_R(\xi, Y) dy < \int f_Y(y) S_C(y) L_R(\zeta, Y) dy \\
&\Rightarrow \mathbb{E}[L_R(\xi, Y)] < \mathbb{E}[L_R(\zeta, Y)]
\end{aligned}$$

where the first inequality follows by definition of L_S being strictly proper, the second by definition of L_S, the third is conditional expectation, the fourth is substitution of Δ, the fifth is definition of expectation and cancelling the q terms, the sixth is corollary 4.6.6, the seventh from cancelling $\hat{G}_{KM}$ and S_C in the asymptotic with infinite training data, the eighth by definition of expectation.

As $\zeta \neq \xi$ and $Y \sim \xi$ it follows that L_R is strictly proper by definition, as required. ■

Proof L_R strictly proper $\Rightarrow$ L_S strictly proper. Again let $\mathcal{P}$ be a family of absolutely continuous distributions over the positive Reals and let ζ, ξ be distinct distributions in $\mathcal{P}$. Let Y be some random variable distributed according to ξ and let C be an r.v. t.v.i. $\mathcal{T}$ with $Y \perp\!\!\!\perp C$. Let $T := \min\{Y, C\}$ and $\Delta := \mathbb{I}(T = Y)$.

Proof follows by definition of strict properness,

$$
\begin{aligned}
&\mathbb{E}[L_R(\xi, Y)] < \mathbb{E}[L_R(\zeta, Y)] \\
&\Rightarrow \int f_Y(y) L_R(\xi, Y)\ dy < \int f_Y(y) S_C(y) L_R(\zeta, Y)\ dy \\
&\Rightarrow \int \frac{f_Y(y) S_C(y) L_R(\xi, Y)}{S_C(y)}\ dy < \int \frac{f_Y(y) S_C(y) L_R(\zeta, Y)}{S_C(y)}\ dy \\
&\Rightarrow \int \frac{f_Y(y) S_C(y) L_R(\xi, Y)}{\hat{G}_{KM}(y)}\ dy < \int \frac{f_Y(y) S_C(y) L_R(\zeta, Y)}{\hat{G}_{KM}(y)}\ dy \\
&\Rightarrow \int \frac{f_{Y|\Delta}(y|1) L_R(\xi, Y)}{\hat{G}_{KM}(y)}\ dy < \int \frac{f_{Y|\Delta}(y|1) L_R(\zeta, Y)}{\hat{G}_{KM}(y)}\ dy \\
&\Rightarrow q\mathbb{E}\Big[\frac{\Delta L_R(\xi, Y)}{\hat{G}_{KM}(Y)}|\Delta = 1\Big] < q\mathbb{E}\Big[\frac{\Delta L_R(\zeta, Y)}{\hat{G}_{KM}(Y)}|\Delta = 1\Big] \\
&\Rightarrow q\mathbb{E}\Big[\frac{\Delta L_R(\xi, Y)}{\hat{G}_{KM}(Y)}|\Delta = 1\Big] + 0 < q\mathbb{E}\Big[\frac{\Delta L_R(\zeta, Y)}{\hat{G}_{KM}(Y)}|\Delta = 1\Big] + 0 \\
&\Rightarrow q\mathbb{E}\Big[\frac{\Delta L_R(\xi, Y)}{\hat{G}_{KM}(Y)}|\Delta = 1\Big] + (1-q)\mathbb{E}\Big[\frac{\Delta L_R(\xi, C)}{\hat{G}_{KM}(C)}|\Delta = 0\Big] \\
&\qquad < q\mathbb{E}\Big[\frac{\Delta L_R(\zeta, Y)}{\hat{G}_{KM}(Y)}|\Delta = 1\Big] + (1-q)\mathbb{E}\Big[\frac{\Delta L_R(\zeta, C)}{\hat{G}_{KM}(C)}|\Delta = 0\Big] \\
&\Rightarrow \mathbb{E}\Big[\frac{\Delta L_R(\xi, T)}{\hat{G}_{KM}(T)}\Big] < \mathbb{E}\Big[\frac{\Delta L_R(\zeta, T)}{\hat{G}_{KM}(T)}\Big]
\end{aligned}
$$

where the first inequality follows by definition of L_R being strictly proper, the second by definition of expectation, the third by multiplying the numerator and denominator on both sides by $S_C(y)$, the fourth in the asymptotic given infinite training data, the fifth by corollary 4.6.6, the sixth by definition of conditional expectation and multiplying both sides by $q = P(\Delta = 1)$, the seventh by adding 0 to both sides, the eighth by substituting expressions equalling 0 to both sides, the final by law of total expectation.

As $\zeta \neq \xi$ and $Y \sim \xi$ it follows that $L_S : \mathcal{P} \times \mathcal{T} \times \{0, 1\} \times \mathcal{C} \to \bar{\mathbb{R}}$ is strictly proper by definition, as required. ■

□

Proposition 4.6.11. *The following approximate survival losses are outcome-independent strictly proper:*

i) L_{SDLL^*} – *eq.* (4.6.24)

ii) L_{IGS^*} – *eq.* (4.6.21)

iii) L_{ISLL^*} – *eq.* (4.6.22)

Proof. Proofs follow from theorem 4.6.10 and noting that:

$$L_{SDLL^*}(\zeta, t, \delta, \hat{G}_{KM}) = \frac{\delta L_{DL}(\zeta, t)}{\hat{G}_{KM}(t)} \tag{4.6.36}$$

$$L_{IGS^*}(\zeta, t, \delta, \hat{G}_{KM}) = \frac{\delta L_{IBS}(\zeta, t)}{\hat{G}_{KM}(t)} \tag{4.6.37}$$

$$L_{ISLL^*}(\zeta, t, \delta, \hat{G}_{KM}) = \frac{\delta L_{ILL}(\zeta, t)}{\hat{G}_{KM}(t)} \tag{4.6.38}$$

where $L_{DL} : \mathcal{P} \times \mathcal{T} \to \bar{\mathbb{R}}$ is the density log-loss, $L_{IBS} : \mathcal{P} \times \mathcal{T} \to \bar{\mathbb{R}}$ is the integrated Brier score, $L_{ILL} : \mathcal{P} \times \mathcal{T} \to \bar{\mathbb{R}}$ is the integrated log-loss (section 4.6.1.2), and all three are strictly proper [100, 111]. □

4.6.4.4. Non-Proper Approximate Survival Losses

From the previous proofs, it would be natural to assume that L_{IGS} and L_{ISLL} are also outcome-independent strictly proper, however this is not the case.

Proposition 4.6.12. *The integrated Graf score, L_{IGS}, is not:*

i) outcome-independent proper

ii) outcome-independent strictly proper

iii) proper

iv) strictly proper

Proof.

Proof of (i). Let $\mathcal{P}$ be a family of absolutely continuous distributions over the positive Reals and let ζ, ξ be some distributions in $\mathcal{P}$. Let Y be some random variable distributed according to ξ and let C be an r.v. t.v.i. $\mathcal{T}$ and let Y and C be independent. Let $T := \min\{Y, C\}$ and $\Delta := \mathbb{I}(T = Y)$. Let $\hat{Y}$ be some random variable distributed according to ζ, independent of Y, C, T, and Δ.

By lemma 4.6.4, the loss is improper if there exists some ζ, ξ such that $D_{SDLL}(\xi, \zeta) < 0$. Proof follows by demonstrating such ζ, ξ exist and therefore that the loss is improper. First calculating $S_{IGS}(\xi, \zeta)$,

$$
\begin{aligned}
S_{IGS}(\xi,\zeta) &= \mathbb{E}\{\int_0^{\tau^*} \frac{S^2_{\hat{Y}}(\tau)\mathbb{I}(T \leq \tau, \Delta = 1)}{\hat{G}_{KM}(T)} + \frac{F^2_{\hat{Y}}(\tau)\mathbb{I}(T > \tau)}{\hat{G}_{KM}(\tau)} \; d\tau\} \\
&= \mathbb{E}[\int_T^{\tau^*} \frac{S^2_{\hat{Y}}(\tau)\Delta}{\hat{G}_{KM}(T)} \; d\tau] + \mathbb{E}[\int_0^{T} \frac{F^2_{\hat{Y}}(\tau)}{\hat{G}_{KM}(\tau)} \; d\tau] \\
&= q\mathbb{E}[\int_T^{\tau^*} \frac{S^2_{\hat{Y}}(\tau)\Delta}{\hat{G}_{KM}(T)} \; d\tau|\Delta = 1] + q\mathbb{E}[\int_0^{T} \frac{F^2_{\hat{Y}}(\tau)}{\hat{G}_{KM}(\tau)} \; d\tau|\Delta = 1] + \\
&\quad (1-q)\mathbb{E}[\int_T^{\tau^*} \frac{S^2_{\hat{Y}}(\tau)\Delta}{\hat{G}_{KM}(T)} \; d\tau|\Delta = 0] + (1-q)\mathbb{E}[\int_0^{T} \frac{F^2_{\hat{Y}}(\tau)}{\hat{G}_{KM}(\tau)} \; d\tau|\Delta = 0] \\
&= q\mathbb{E}[\int_Y^{\tau^*} \frac{S^2_{\hat{Y}}(\tau)}{\hat{G}_{KM}(Y)} \; d\tau|\Delta = 1] + q\mathbb{E}[\int_0^{Y} \frac{F^2_{\hat{Y}}(\tau)}{\hat{G}_{KM}(\tau)} \; d\tau|\Delta = 1] + \\
&\quad (1-q)\mathbb{E}[\int_0^{C} \frac{F^2_{\hat{Y}}(\tau)}{\hat{G}_{KM}(\tau)} \; d\tau|\Delta = 0] \\
&= q\int f_{Y|\Delta}(y|1)\int_y^{\tau^*} \frac{S^2_{\hat{Y}}(\tau)}{\hat{G}_{KM}(y)} \; d\tau dy + q\int f_{Y|\Delta}(y|1)\int_0^{y} \frac{F^2_{\hat{Y}}(\tau)}{\hat{G}_{KM}(\tau)} \; d\tau dy + \\
&\quad (1-q)\int f_{C|\Delta}(c|0)\int_0^{c} \frac{F^2_{\hat{Y}}(\tau)}{\hat{G}_{KM}(\tau)} \; d\tau dc \\
&= \int f_{Y,\Delta}(y,1)\int_t^{\tau^*} \frac{S^2_{\hat{Y}}(\tau)}{\hat{G}_{KM}(y)} \; d\tau dy + \int f_{Y,\Delta}(y,1)\int_0^{y} \frac{F^2_{\hat{Y}}(\tau)}{\hat{G}_{KM}(\tau)} \; d\tau dy + \\
&\quad \int f_{C,\Delta}(c,0)\int_0^{c} \frac{F^2_{\hat{Y}}(\tau)}{\hat{G}_{KM}(\tau)} \; d\tau dc \\
&= \int f_Y(y)S_C(y)\int_y^{\tau^*} \frac{S^2_{\hat{Y}}(\tau)}{\hat{G}_{KM}(y)} \; d\tau dy + \int f_Y(y)S_C(y)\int_0^{y} \frac{F^2_{\hat{Y}}(\tau)}{\hat{G}_{KM}(\tau)} \; d\tau dy + \\
&\quad \int f_C(c)S_Y(c)\int_0^{c} \frac{F^2_{\hat{Y}}(\tau)}{\hat{G}_{KM}(\tau)} \; d\tau dc
\end{aligned}
$$

where the second equality is linearity of integration and expectation, the third is law of total expectation and substituting $q := P(\Delta = 1)$, the fourth is substituting Y for $T|\Delta = 1$, C for $T|\Delta = 0$, the fifth is definition of conditional expectation, the sixth is definition of conditional probability, the seventh is corollary 4.6.6 and lemma 4.6.7. To prove L_{IGS} is not proper, proof continues in the asymptotic as $n \to \infty$ and $\hat{G}_{KM} \to S_C$ [153],

$$
\begin{aligned}
S_{IGS}(\xi,\zeta) = &\int f_Y(y)\int_y^{\tau^*} S^2_{\hat{Y}}(\tau) \; d\tau dy + \int f_Y(y)S_C(y)\int_0^{y} \frac{F^2_{\hat{Y}}(\tau)}{S_C(\tau)} \; d\tau dy + \\
&\int f_C(c)S_Y(c)\int_0^{c} \frac{F^2_{\hat{Y}}(\tau)}{S_C(\tau)} \; d\tau dc
\end{aligned}
$$

In addition for this counter-example let $C \sim \xi$ so that the above reduces to,

$$S_{IGS}(\xi,\zeta) = \int f_Y(y) \int_y^{\tau^*} S^2_{\hat{Y}}(\tau)\, d\tau dy + 2\int f_Y(y) S_Y(y) \int_0^y \frac{F^2_{\hat{Y}}(\tau)}{S_Y(\tau)}\, d\tau dy$$

The loss is improper if there is at least one $\zeta \neq \xi \in \mathcal{P}$ s.t. $S_{IGS}(\xi,\zeta) < H_{IGS}(\xi)$. To find such a counter-example let $\xi = \text{Exp}(a)$ and let $\zeta = \text{Exp}(b)$. If some $X \sim \text{Exp}(\lambda)$ then $f_X(x) = \lambda e^{-\lambda x}, S_X(x) = e^{-\lambda x}, S^2_X(x) = e^{-2\lambda x}, F_X(x) = 1 - e^{-\lambda x}, F^2_X(x) = 1 - 2e^{-\lambda x} + e^{-2\lambda x}$. Continuing by substituting these results,

$$\begin{aligned} S_{IGS}(\xi,\zeta) &\Rightarrow \int (ae^{-ay}) \int_y^{\tau^*} (e^{-2b\tau})\, d\tau dy + 2\int (ae^{-2ay}) \int_0^y \frac{1 - 2e^{-b\tau} + e^{-2b\tau}}{(e^{-a\tau})}\, d\tau dy \\ &= ... \\ &= \frac{2}{a} - \frac{e^{-2b\tau^*}}{2b} + \frac{a}{2b(a-2b)} - \frac{2b^2}{a(a-b)(a-2b)} - \frac{4a}{(a+b)(a-b)} \end{aligned}$$

Full calculations are provided in appendix C.

Proof that the loss is not strictly proper only requires a single counter-example, hence for convenience select $\tau^* = 1$, which simplifies the above as follows

$$\begin{aligned} S_{IGS}(\xi,\zeta) &\Rightarrow \frac{2}{a} - \frac{e^{-2b}}{2b} + \frac{a}{2b(a-2b)} - \frac{2b^2}{a(a-b)(a-2b)} - \frac{4a}{(a+b)(a-b)} \\ &= \frac{a^2 - ae^{-2b}(a+b) - ab + 2b^2}{2ab(a+b)} \end{aligned}$$

To demonstrate the loss is improper let $a = 2$ and first let $b = a = 2$, then

$$\begin{aligned} H_{IGS}(\xi) &= \frac{a^2 - ae^{-2b}(a+b) - ab + 2b^2}{2ab(a+b)} \\ &= \frac{1 - e^{-2a}}{2a} \\ &= \frac{1 - e^{-4}}{4} \end{aligned}$$

Now when $\zeta \neq \xi$, let $b = \frac{3}{2}$,

$$\begin{aligned} S_{IGS}(\xi,\zeta) &= \frac{2^2 - 2e^{-2\frac{3}{2}}(2 + \frac{3}{2}) - 2\frac{3}{2} + 2\frac{3}{2}^2}{(2)(2)\frac{3}{2}(2 + \frac{3}{2})} \\ &= \frac{11}{42} - \frac{e^{-3}}{3} \end{aligned}$$

Now computing the divergence of ζ from ξ,

$$\begin{aligned} D_{IGS}(\xi, \zeta) &= S_{IGS}(\xi, \zeta) - H_{IGS}(\xi) \\ &= \frac{11}{42} - \frac{e^{-3}}{3} - \frac{1 - e^{-4}}{4} \\ &= -0.000112 < 0 \end{aligned}$$

By lemma 4.6.4 as $D_{IGS}(\xi, \zeta) < 0$ and as $Y \perp\!\!\!\perp C$, it follows that L_{IGS} is not outcome-independent proper. ■

Proof of (ii)-(iv). Proofs follow from (i) and lemma 4.6.2. ■

□

Whilst in this counter-example the value of $D_{IGS}(\xi, \zeta)$ is very close to zero, there will be other counter-examples with a more pronounced difference, though this is not required for the proof. Also note that again this is not a rare edge case, practically this example is reflected in any real-world scenario in which the prediction is close to the truth and when the censoring and survival times follow the same distribution.

Proposition 4.6.13. *The integrated survival log-loss, L_{ISLL}, is not:*

i) outcome-independent proper

ii) outcome-independent strictly proper

iii) proper

iv) strictly proper

Proof is not provided but follows with the same argumentation as the previous proposition and noting that a counter-example can always be found as C is unknown and cannot be removed from the equation.

Conjecture 4.6.14. *Let $L : \mathcal{P} \times \mathcal{T} \times \{0, 1\} \times \mathcal{C} \to \bar{\mathbb{R}}$ be an approximate survival loss, then L is not strictly proper.*

This conjecture is motivated by noting that the joint distribution of (Y, C) is always unknown and thus a suitable counter-example to strict-properness can likely always be derived.[1]

4.6.5. Baselines and ERV

A common criticism of scoring rules is a lack of interpretability, e.g. without context an IGS of 0.5 or 0.0005 have no meaning. The final part of this section very briefly looks at two methods that help increase the interpretability of scoring rules. Scoring rules may already be considered less transparent than, say, concordance indices, as the underlying mathematics is more abstract, and therefore interpretability of the measure can play a large role in increasing transparency.

[1] This conjecture is being explored as part of a theorem in a paper with external collaborators.

4.6.5.1. Baselines

A baseline is either a model or a value that can be utilised to provide a reference value for a scoring rule, they provide a universal method to judge all models of the same class by [111].

In classification, an analytical baseline value can be derived for measures, i.e. a baseline model does not actually need to be fit to know what a 'good' value for the loss is. For example it is generally known that a Brier score is considered 'good' if it is below 0.25 or a log loss if it is below 0.693 (section 4.6.1). Unfortunately simple analytical expressions are not possible in survival analysis as the losses are dependent on the distributions of both the survival and censoring time. Therefore all experiments in survival analysis must include a baseline model that can produce a reference value in order to derive meaningful results.

There is a clear consensus that the Kaplan-Meier estimator is the most sensible baseline model for survival modelling [108, 186, 254] as it is the simplest model that can consistently estimate the true survival function. One could also consider the Akritas estimator as a tunable conditional baseline (section 3.1.1).

Baseline models are often ignored in experiments when there is overconfidence in a particular model class, this is frequently the case in survival analysis in which a novel model class may only be compared to a Cox PH. This has practical and ethical implications. The calibration example in section 4.5.1 demonstrates how one sophisticated model (CPH) may outperform another (SVM) and still perform worse than the Kaplan-Meier. Not including Kaplan-Meier in every experiment could lead to over-confidence in a novel model that is no better than an unconditional estimator (with no individual predictive ability).

4.6.5.2. Explained Residual Variation

Baseline models can also be utilised to derive a potentially more useful representation of scoring rules. Any scoring rule can be utilised to derive a measure of explained residual variation (ERV) [167, 168] by standardising the loss with respect to a baseline, say Kaplan-Meier. For any survival loss L (analogously for an approximate survival loss), the ERV is,

$$\begin{aligned} &R_L : \mathcal{P} \times \mathcal{P} \times \mathbb{R}^m_{\geq 0} \times \{0,1\}^m \to [0,1]; \\ &(\zeta, \xi_0, t, \delta) \mapsto 1 - \frac{\frac{1}{m}\sum_{i=1}^m L(\zeta, t_i, \delta_i)}{\frac{1}{m}\sum_{i=1}^m L(\xi_0, t_i, \delta_i)} \end{aligned} \tag{4.6.39}$$

where $t = t_1, ..., t_m, \delta = \delta_1, ..., \delta_m$ and ζ should be a predicted distribution from a sophisticated (non-baseline) model and ξ_0 is a prediction from the Kaplan-Meier estimator.[1]

Representing a scoring rule in this manner improves interpretability by allowing for model comparison whilst simultaneously capturing the improvement from a baseline. Therefore instead of reporting some arbitrary loss value, say $L = 0.1$,

[1] Equation (4.6.39) assumes the numerator is always less than the denominator or more specifically that the sophisticated model is 'better' than the baseline; if this is not the case then $R^2_L < 0$. Therefore this representation should only be utilised when the model outperforms the baseline.

one can instead report $R_L = 70\%$ which demonstrates a clear improvement (of 70%) over the baseline.

4.7. Conclusions

This chapter briefly reviewed different classes of survival measures before focusing on the application of scoring rules to survival analysis.

One finding of note from the review of survival measures is the possibility that research and debate has become too focused on measures of discrimination. For example, many papers state the flaws of Harrell's C index [105, 246, 265, 299] however few acknowledge that simulation experiments have demonstrated that common alternatives yield very similar results to Harrell's C [246, 293] and moreover some promimnent alternatives, such as Uno's C [299], are actually harder to interpret due to very high variance [246, 265]. Whilst all concordance indices may be considered accessible and transparent, there is considerable doubt over their performance due to influence from censoring.

Focus on discrimination could be the reason for less development in survival time and calibration measures. There is evidence [317] of the censoring-adjusted RMSE, MAE, and MSE (section 4.3) being used in evaluation but without any theoretical justification, which may lead to questionable results. Less development in calibration measures is likely due to these measures being more widely utilised for re-calibration of models and not in model comparison. The new D-Calibration measure [8, 113] could prove useful for model comparison however independent simulation experiments and theoretical studies of the measure's properties would first be required. No calibration measures can be considered performant due to a lack of clear definition of a calibration measure for survival, moreover the reviewed measures may not even be transparent and accessible due to requiring expert interpretation.

The most problematic findings in this chapter lie in the survival scoring rules. Section 4.6.4 proved that no commonly used scoring rule is proper, which means that any results regarding model comparison based on these measures are thrown into question. It is also conjectured that no approximate survival loss can be strictly proper (in general), which is due to the joint distribution of the censoring and survival distribution always being unknown and impossible to estimate (though the marginal censoring distribution can be estimated). As demonstrated in section 4.6.1, a proper scoring rule is not necessarily a useful one and therefore is not enough for robust model validation.

As an important caveat to the findings in this chapter, this thesis presents one particular definition of properness for survival scoring rules. This definition is partially subjective and other definitions could instead be considered. Therefore these losses should not be immediately dismissed outright. As well as deriving new losses that are (strictly) proper with respect to the definitions provided here, research may also be directed towards finding other sensible definitions of properness, or in confirming that the definition here is the only sensible option. As these are open research questions, the scoring rules discussed in this chapter are still

utilised in evaluation for the benchmark experiment in chapter 7.

This chapter demonstrates that no survival measure on its own can capture enough information to fully evaluate a survival prediction. No measure is satisfactorily APT. This is a serious problem that will either lead (or already is leading) to less interest and uptake in survival modelling, or misunderstanding and deployment of sub-optimal models. Evaluation of survival models is still possible but currently requires expert interpretation to prevent misleading results. If the aim of a study is solely in assessing a model's discriminatory power, then measures of discrimination alone are sufficient, otherwise a range of classes should be included to capture all aspects of model performance. This thesis advocates reporting *all* of the below to evaluate model performance:

- **Calibration**: Houwelingen's α and [309] *and* D-calibration [113].
- **Discrimination**: Harrell's [115] *and* Uno's [300] C. By including two (or even more) measures of concordance, one can determine a feasible range for the 'true' discriminatory ability of the model instead of basing results on a single measure. Time-dependent AUCs can also be considered but these may require expert-interpretation and may only be advisable for discrimination-specific studies.
- **Scoring Rules**: When censoring is outcome-independent and a large enough training dataset is available, then the re-weighted integrated Graf score and re-weighted integrated survival log-loss (section 4.6.3). Otherwise the IGS *and* ISLL [109] which should be interpreted together to ensure consistency in results.

If survival time prediction is the primary goal then RMSE_C and MAE_C can be included in the analysis however these should not form the primary conclusions due to a lack of theoretical justification. Instead, scoring rules should be utilised as a distributional prediction can always be composed into a survival time prediction (section 5.4.3).

All measures discussed in this chapter, with the exception of the Blanche AUC, have been implemented in **mlr3proba** (chapter 6). The listed measures above are utilised in the benchmark experiment in chapter 7.

Chapter 5

Composition and Reduction

In this chapter, composition and reduction are formally introduced, defined and demonstrated within survival analysis. Neither of these are novel concepts in general or in survival, with several applications already seen in chapter 3 (particularly in neural networks), however a lack of formalisation has led to much repeated work and at times questionable applications (section 3.6). The primary purpose of this chapter is to formalise composition and reduction for survival and to unify references and strategies for future use. These strategies are introduced in the context of minimal 'workflows' and graphical 'pipelines' in order to maximise their generalisability. The pipelines discussed in this chapter are implemented in **mlr3proba** and returned to in section 6.4.

A *workflow* is a generic term given to a series of sequential operations. For example a standard ML workflow is fit/predict/evaluate, which means a model is fit, predictions are made, and these are evaluated. In this thesis, a *pipeline* is the name given to a concrete workflow. Section 5.1 demonstrates how pipelines are represented in this thesis.

Composition (section 5.2) is a general process in which an object is built (or composed) from other objects and parameters. Reduction (section 5.3) is a closely related concept that utilises composition in order to transform one problem into another. Concrete strategies for composition and reduction are detailed in sections 5.4 and 5.5.

Notation and Terminology The notation introduced in chapter 2 is recapped for use in this chapter: the generative survival template for the survival setting is given by (X, T, Δ, Y, C) t.v.i. $\mathcal{X} \times \mathcal{T} \times \{0,1\} \times \mathcal{T} \times \mathcal{T}$ where $\mathcal{X} \subseteq \mathbb{R}^p$ and $\mathcal{T} \subseteq \mathbb{R}_{\geq 0}$, where C, Y are unobservable, $T := \min\{Y, C\}$, and $\Delta = \mathbb{I}(Y = T)$. Random survival data is given by $(X_i, T_i, \Delta_i, Y_i, C_i) \overset{i.i.d.}{\sim} (X, T, \Delta, Y, C)$. Usually data will instead be presented as a training dataset, $\mathcal{D}_0 = \{(X_1, T_1, \Delta_1), ..., (X_n, T_n, \Delta_n)\}$ where $(X_i, T_i, \Delta_i) \overset{i.i.d.}{\sim} (X, T, \Delta)$, and some test data $\mathcal{D}_1 = (X^*, T^*, \Delta^*) \sim (X, T, \Delta)$.

For regression models the generative template is given by (X, Y) t.v.i. $\mathcal{X} \subseteq \mathbb{R}^p$ and $Y \subseteq \mathbb{R}$. As with the survival setting, a regression training set is given by $\{(X_1, Y_1), ..., (X_n, Y_n)\}$ where $(X_i, Y_i) \overset{i.i.d.}{\sim} (X, Y)$ and some test data $(X^*, Y^*) \sim (X, Y)$.

5.1. Representing Pipelines

Before introducing concrete composition and reduction algorithms, this section briefly demonstrates how these pipelines will be represented in this thesis.

Pipelines are represented by graphs designed in the following way: all are drawn with operations progressing sequentially from left to right; graphs are comprised of nodes (or 'vertices') and arrows (or 'directed edges'); a rounded rectangular node represents a process such as a function or model fitting/predicting; a (regular) rectangular node represents objects such as data or hyper-parameters. Output from rounded nodes are sometimes explicitly drawn but when omitted the output from the node is the input to the next.

These features are demonstrated in fig. 14. Say $y = 2$ and $a = 2$, then: data is provided ($y = 2$) and passed to the shift function ($f(x) = x + 2$), the output of this function ($y = 4$) is passed directly to the next ($h(x|a) = x^a$), this function requires a parameter which is also input ($a = 2$), finally the resulting output is returned ($y^* = 16$). Programmatically, $a = 2$ would be a hyper-parameter that is stored and passed to the required function when the function is called.

This pipeline is represented as a pseudo-algorithm in algorithm 6, though of course is overly complicated and in practice one would just code $(y + 2)\hat{}a$.

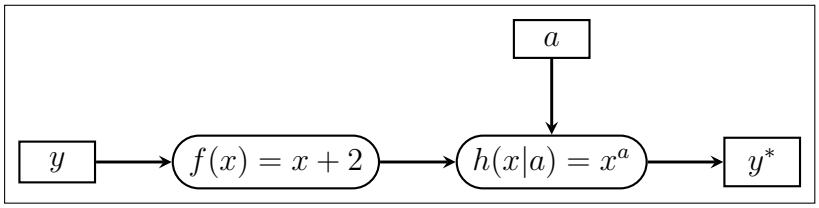


Figure 14: Example of a pipeline.

Algorithm 6 Example pipeline.
Input Data, $y \in \mathbb{R}$. Parameter, $a \in \mathbb{R}$.
Output Transformed data, $x \in \mathbb{R}$.

$x \leftarrow y$
$x \leftarrow x + 2$
$x \leftarrow x\hat{}a$
return x

5.2. Introduction to Composition

This section introduces composition, defines a taxonomy for describing compositors (section 5.2.1), and provides some motivating examples of composition in survival analysis (section 5.2.2).

In the simplest definition, a model (be it mathematical, computational, machine learning, etc.) is called a *composite model* if it is built of two or more constituent parts. This can be simplest defined in terms of objects. Just as objects in the real-world can be combined in some way, so can mathematical objects. The exact 'combining' process (or 'compositor') depends on the specific

composition, so too do the inputs and outputs. By example, a wooden table can be thought of as a composite object (fig. 15). The inputs are wood and nails, the combining process is hammering (assuming the wood is pre-chopped), and the output is a surface for eating. In mathematics, this process is mirrored. Take the example of a shifted linear regression model. This is defined by a linear regression model, $f(x) = \beta_0 + x\beta_1$, a shifting parameter, α, and a compositor $g(x|\alpha) = f(x) + \alpha$. Mathematically this example is overly trivial as this could be directly modelled with $f(x) = \alpha + \beta_0 + x\beta_1$, but algorithmically there is a difference. The composite model g, is defined by first fitting the linear regression model, f, and then applying a shift, α; as opposed to fitting a directly shifted model.

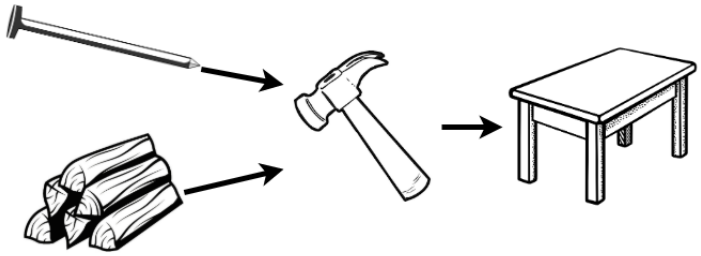


Figure 15: Visualising composition in the real-world. A table is a composite object built from nails and wood, which are combined with a hammer 'compositor'. Figure not to scale.

Why Composition? Tables tend to be better surfaces for eating your dinner than bundles of wood. Or in modelling terms, it is well-known that ensemble methods (e.g. random forests) will generally outperform their components (e.g. decision trees). All ensemble methods are composite models and this demonstrates one of the key use-cases of composition: improved predictive performance. The second key use-case is reduction, which is fully discussed in section 5.3. Section 5.2.2 motivates composition in survival analysis by demonstrating how it is already prevalent but requires formalisation to make compositions more transparent and accessible.

Composite Model vs. Sub-models A bundle of wood and nails is not a table and $1,000$ decision trees are not a random forest, both require a compositor. The compositor in a composite model combines the components into a single model. Considering a composite model as a single model enables the hyperparameters of the compositor and the component model(s) to be efficiently tuned whilst being evaluated as a single model. This further allows the composite to be compared to other models, including its own components, which is required to justify complexity instead of parsimony in model building (section 4.1.2).

5.2.1. Taxonomy of Compositors

Just as there are an infinite number of ways to make a table, composition can come in infinite forms. However there are relatively few categories that these can be grouped into. Two primary taxonomies are identified here. The first is the 'composition type' and relates to the number of objects composed:

i) Single-Object Composition (SOC) – This form of composition either makes use of parameters or a transformation to alter a single object. The shifted linear regression model above is one example of this, another is given in section 5.4.3.

ii) Multi-Object Composition (MOC) – In contrast, this form of composition combines multiple objects into a single one. Both examples in section 5.2.2 are multi-object compositions.

The second grouping is the 'composition level' and determines at what 'level' the composition takes place:

i) Prediction Composition – This applies at the level of predictions; the component models could be forgotten at this point. Predictions may be combined from multiple models (MOC) or transformed from a single model (SOC). Both examples in section 5.2.2 are prediction compositions.

ii) Task Composition – This occurs when one task (e.g. regression) is transformed to one or more others (e.g. classification), therefore always SOC. This is seen mainly in the context of reduction (section 5.3).

iii) Model Composition – This is commonly seen in the context of wrappers (section 5.5.7.4), in which one model is contained within another.

iv) Data Composition – This is transformation of training/testing data types, which occurs at the first stage of every pipeline.

5.2.2. Motivation for Composition

Two examples are provided below to demonstrate common uses of composition in survival analysis and to motivate the compositions introduced in section 5.4.

Example 1: Cox Proportional Hazards Common implementations of well-known models can themselves be viewed as composite models, the Cox PH is the most prominent example in survival analysis. Recall the model defined by

$$h(\tau|X_i) = h_0(\tau)\exp(\beta X_i) \tag{5.2.1}$$

where h_0 is the baseline hazard and β are the model coefficients.

This can be seen as a composite model as Cox defines the model in two stages [59]: first fitting the β-coefficients using the partial likelihood and then by suggesting an estimate for the baseline distribution. This first stage produces a linear predictor return type (section 2.3) and the second stage returns a survival

distribution prediction. Therefore the Cox model for linear predictions is a single (non-composite) model, however when used to make distribution predictions then it is a composite. Cox implicitly describes the model as a composite by writing "alternative simpler procedures would be worth having" [59], which implies a decision in fitting (a key feature of composition). This composition is formalised in section 5.4.1 as a general pipeline (C1). The Cox model utilises the (C1) pipeline with a PH form and Kaplan-Meier baseline.

Example 2: Random Survival Forests Fully discussed in section 3.3, random survival forests are composed from many individual decision trees via a prediction composition algorithm (algorithm 3). In general, random forests perform better than their component decision trees, which tends to be true of all ensemble methods. Aggregation of predictions in survival analysis requires slightly more care than other fields due to the multiple prediction types, however this is still possible and is formalised in section 5.4.4.

5.3. Introduction to Reduction

This section introduces reduction, motivates its use in survival analysis (section 5.3.1), details an abstract reduction pipeline and defines the difference between a complete/incomplete reduction (section 5.3.2), and outlines some common mistakes that have been observed in the literature when applying reduction (section 5.3.3).

Reduction is a concept found across disciplines with varying definitions. This report uses the Langford definition: reduction is "a complex problem decomposed into simpler subproblems so that a solution to the subproblems gives a solution to the complex problem" [184]. Generalisation (or induction) is a common real-world use of reduction, for example sampling a subset of a population in order to estimate population-level results. The true answer (population-level values) may not always be found in this way but very good approximations can be made with simpler sub-problems (sub-sampling).

Reductions are workflows that utilise composition. By including hyper-parameters, even complex reduction strategies can remain relatively flexible. To illustrate reduction by example, recall the table-building example (section 5.2) in which the task of interest is to acquire a table. The most direct but complex solution is to fell a tree and directly saw it into a table (fig. 16, top), clearly this is not a sensible process. Instead the problem can be reduced into simpler sub-problems: saw the tree into bundles of wood, acquire nails, and then use the 'hammer compositor' (fig. 15) to create a table (fig. 16, bottom).

In a modelling example, predicting a survival distribution with the Cox model can be viewed as a reduction in which two sub-problems are solved and composed: i) predict continuous ranking; ii) estimate baseline hazard; and iii) compose with (C1) (section 5.4.1). This is visualised as a reduction strategy in fig. 17. The entire process from defining the original problem, to combining the simpler sub-solutions (in green), is the reduction (in red).

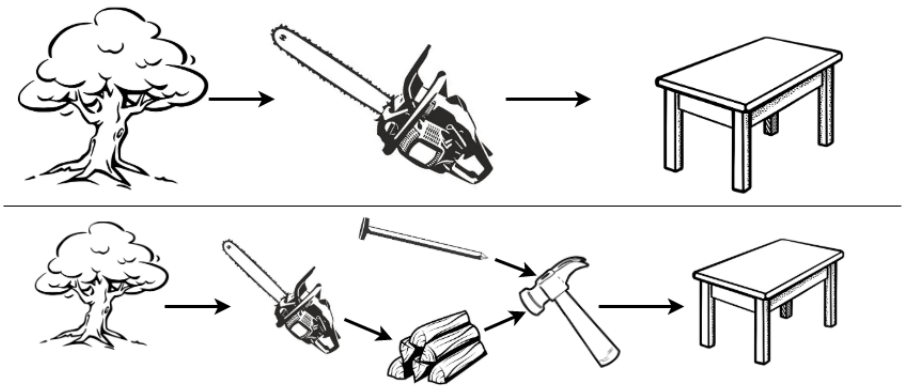


Figure 16: Visualising reduction in the real-world. The complex process (top) of directly sawing a tree into a table is inefficient and unnecessarily complex. The reduction (bottom) that involves first creating bundles of wood is simpler, more efficient, and yields the same result, though technically requiring more steps.

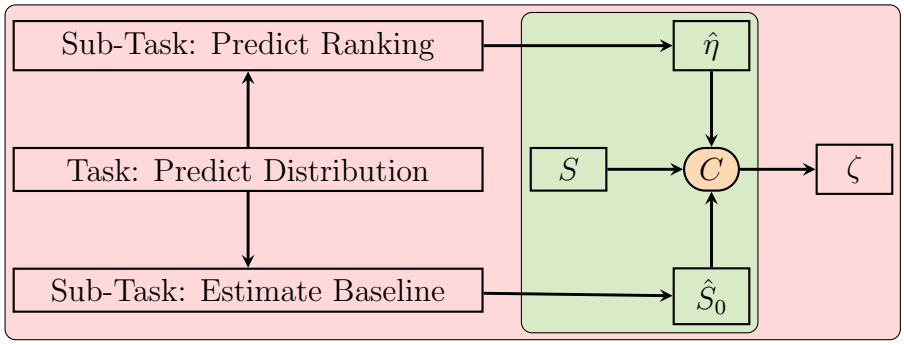


Figure 17: Solving a survival distribution task by utilising reduction and (C1) (section 5.4.1). S, $\hat{\eta}$, C, $\hat{S}_0$ are fully described in fig. 19. The nodes in the green area are part of the composite model, all nodes combined form the reduction.

5.3.1. Reduction Motivation

Formalisation of reduction positively impacts upon accessibility, transparency, and predictive performance. Improvements to predictive performance have already been demonstrated when comparing random forests to decision trees. In addition, a reduction with multiple stages and many hyper-parameters allows for fine tuning for improved transparency and model performance (usual overfitting caveat applies, as does the trade-off described in section 5.6).

The survey of ANNs (section 3.6) demonstrated how reduction is currently utilised without transparency. Many of these ANNs are implicitly reductions to probabilistic classification (section 5.5.7.6) however none include details about how the reduction is performed. Furthermore in implementation, none provide interface points to the reduction hyper-parameters. Formalisation encourages consistent terminology, methodology and transparent implementation, which can only improve model performance by exposing further hyper-parameters.

Accessibility is improved by formalising specific reduction workflows that previously demanded expert knowledge in deriving, building, and running these pipelines. All regression reductions in this chapter, are implemented in

mlr3proba [281] and can be utilised with any possible survival model.

Finally there is an economic and efficiency advantage to reduction. A reduction model is relatively 'cheap' to explore as they utilise pre-established models and components to solve a new problem. Therefore if a certain degree of predictive ability can be demonstrated from reduction models, it may not be worth the expense of pursuing more novel ideas and hence reduction can help direct future research.

5.3.2. Task, Loss, and Data Reduction

Reduction can be categorised into task, loss, and data reduction, often these must be used in conjunction with each other. The direction of the reductions may be one- or two-way; this is visualised in fig. 18. This diagram should not be viewed as a strict fit/predict/evaluation workflow but instead as a guidance for which tasks, T, data, D, models, M, and losses, L, are required for each other. The subscript O refers to the original object 'level' before reduction, whereas the subscript R is in reference to the reduced object.

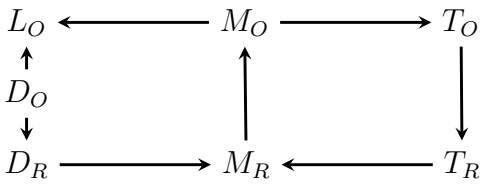


Figure 18: Task, loss, and data reduction to and from the original complex problem to sub-problems.

The individual task, model, and data compositions in the diagram are listed below, the reduction from survival to classification (section 5.5.7) is utilised as a running example to help exposition.

- $T_O \to T_R$: By definition of a machine learning reduction, task reduction will always be one way. A more complex task, T_O, is reduced to a simpler one, T_R, for solving. T_R could also be multiple simpler tasks. For example, solving a survival task, T_O, by classification, T_R (section 5.5.7).

- $T_R \to M_R$: All machine learning tasks have models that are designed to solve them. For example logistic regression, M_R, for classification tasks, T_R.

- $M_R \to M_O$: The simpler models, M_R, are used for the express purpose to solve the original task, T_O, via solving the simpler ones. To solve T_O, a compositor must be applied, which may transform one (SOC) or multiple models (MOC) at a model- or prediction-level, thus creating M_O. For example predicting survival probabilities with logistic regression, M_R, at times $1, ..., \tau^*$ for some $\tau^* \in \mathbb{N}_{>0}$ (section 5.5.7.4).

- $M_O \to T_O$: The original task should be solvable by the composite model. For example predicting a discrete survival distribution by concatenating probabilistic predictions at the times $1, ..., \tau^*$ (section 5.5.7.6).

- $D_O \to D_R$: Just as the tasks and models are reduced, the data required to fit these must likewise be reduced. Similarly to task reduction, data reduction can usually only take place in one direction, to see why this is the case take an example of data reduction by summaries. If presented with 10 data-points $\{1, 1, 1, 5, 7, 3, 5, 4, 3, 3\}$ then these could be reduced to a single point by calculating the sample mean, 3.3. Clearly given only the number 3.3 there is no strategy to recover the original data. There are very few (if any) data reduction strategies that allow recovery of the original data. Continuing the running example, survival data, D_O, can be binned (section 5.5.7.1) to classification data, D_R.

There is no arrow between D_O and M_O as the composite model is never fit directly, only via composition from $M_R \to M_O$. However, the original data, D_O, is required when evaluating the composite model against the respective loss, L_O.[1] Reduction should be directly comparable to non-reduction models, hence this diagram does not include loss reduction and instead insists that all models are compared against the same loss L_O.

A reduction is said to be *complete* if there is a full pipeline from $T_O \to M_O$ and the original task is solved, otherwise it is *incomplete*. The simplest complete reduction is comprised of the pipeline $T_O \to T_R \to M_R \to M_O$. Usually this is not sufficient on its own as the reduced models are fit on the reduced data, $D_R \to M_R$.

A complete reduction can be specified by detailing: i) the original task and the sub-task(s) to be solved, $T_O \to T_R$; ii) the original dataset and the transformation to the reduced one, $D_O \to D_R$ (if required); and iii) the composition from the simpler model to the complex one, $M_R \to M_O$.

5.3.3. Common Mistakes in Implementation of Reduction

In surveying models and measures, several common mistakes in the implementation of reduction and composition were found to be particularly prevalent and problematic throughout the literature. It is assumed that these are indeed mistakes (not deliberate) and result from a lack of prior formalisation. These mistakes were even identified 20 years ago [268] but are provided in more detail in order to highlight their current prevalence and why they cannot be ignored.

RM1) Incomplete reduction. This occurs when a reduction workflow is presented as if it solves the original task but fails to do so and only the reduction strategy is solved. A common example is claiming to solve the survival task by using binary classification, e.g. erroneously claiming that a model predicts survival probabilities (which implies distribution) when it actually predicts a five year probability of death (box 1). This is a mistake as it misleads readers into believing that the model solves a survival task (box 3) when it does not. This is usually a semantic not mathematical error and

[1] A complete diagram would indicate that D_O is split into training data, which is subsequently reduced, and test data, which is passed to L_O. All reductions in this section can be applied to any data splitting process.

results from misuse of terminology. It is important to be clear about model predict types (section 2.3) and general terms such as 'survival predictions' should be avoided unless they refer to one of the three prediction tasks.

RM2) Inappropriate comparisons. This is a direct consequence of (RM1) and the two are often seen together. (RM2) occurs when an incomplete reduction is directly compared to a survival model (or complete reduction model) using a measure appropriate for the reduction. This may lead to a reduction model appearing erroneously superior. For example, comparing a logistic regression to an RSF (section 3.3) for predicting survival probabilities at a single time using the accuracy measure is an unfair comparison as the RSF is optimised for distribution predictions. This would be non-problematic if a suitable composition is clearly utilised. For example a regression SSVM predicting survival time cannot be directly compared to a Cox PH. However the SSVM can be compared to a CPH composed with the probabilistic to deterministic compositor (C3), then conclusions can be drawn about comparison to the composite survival time Cox model (and not simply a Cox PH).

RM3) Naïve censoring deletion. This common mistake occurs when trying to reduce survival to regression or classification by simply deleting all censored observations, even if censoring is informative. This is a mistake as it creates bias in the dataset, which can be substantial if the proportion of censoring is high and informative. More robust deletion methods are described in section 5.4.5.

RM4) Oversampling uncensored observations. This is often seen when trying to reduce survival to regression or classification, and often alongside (RM3). Oversampling is the process of replicating observations to artificially inflate the sample size of the data. Whilst this process does not create any new information, it can help a model detect important features in the data. However, by only oversampling uncensored observations, this creates a source of bias in the data and ignores the potentially informative information provided by the proportion of censoring.

5.4. Composition Strategies for Survival Analysis

Though composition is common practice in survival analysis, with the Cox model being a prominent example, a lack of formalisation means a lack of consensus in simple operations. For example, it is often asked in survival analysis how a model predicting a survival distribution can be used to return a survival time prediction. A common strategy is to define the survival time prediction as the median of the predicted survival curve however there is no clear reason why this should be more sensible than returning the distribution mean, mode, or some random quantile. Formalisation allow these choices to be analytically compared both theoretically and practically as hyper-parameters in a workflow. Four prediction compositions

Table 7: Compositions formalised in section 5.4.

ID[1]	Composition	Type[2]	Level[3]
C1)	Linear predictor to distribution	MOC	Prediction
C2)	Survival time to distribution	MOC	Prediction
C3)	Distribution to survival time	SOC	Prediction
C4)	Survival model averaging	MOC	Prediction
C5)	Survival to regression	SOC	Data

1. ID for reference throughout this thesis.
2. Composition type. Multi-object composition (MOC) or single-object composition (SOC).
3. Composition level.

are discussed in this section (table 7), three are utilised to convert prediction types between one another, the fourth is for aggregating multiple predictions. One data composition is discussed for converting survival to regression data. Each is first graphically represented and then the components are discussed in detail. As with losses in the previous chapter, compositions are discussed at an individual observation level but extend trivially to multiple observations.

5.4.1. C1) Linear Predictor → Distribution

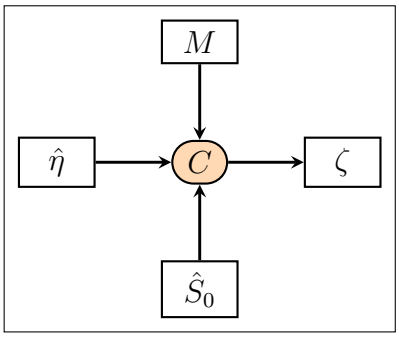


Figure 19: Linear predictor ($\hat{\eta}$) to survival distribution (ζ) composition. Parameters: M – Model form; $\hat{S}_0$ – Estimated baseline survival function.

This is a prediction-level MOC that composes a survival distribution from a predicted linear predictor and estimated baseline survival distribution. The composition (fig. 19) requires:

- $\hat{\eta}$: Predicted linear predictor. $\hat{\eta}$ can be tuned by including this composition multiple times in a benchmark experiment with different models predicting $\hat{\eta}$. In theory any continuous ranking could be utilised instead of a linear predictor though results may be less sensible (section 5.6).
- $\hat{S}_0$: Estimated baseline survival function. This is usually estimated by the Kaplan-Meier estimator fit on training data, $\hat{S}_{KM}$. However any model that can predict a survival distribution can estimate the baseline distribution (caveat: see section 5.6) by taking a uniform mixture of the predicted

individual distributions: say $\xi_1, ..., \xi_m$ are m predicted distributions, then $\hat{S}_0(\tau) = \frac{1}{m}\sum_{i=1}^{m} \xi_i.S(\tau)$. The mixture is required as the baseline must be the same for all observations. Alternatively, parametric distributions can be assumed for the baseline, e.g. $\xi = \text{Exp}(2)$ and $\xi.S(t) = \exp(-2t)$. As with $\hat{\eta}$, this parameter is also tunable.

- M: Chosen model form, which theoretically can be any non-increasing right-continuous function but is usually one of:
  - Proportional Hazards (PH): $S_{PH}(\tau|\eta, S_0) = S_0(\tau)^{\exp(\eta)}$
  - Accelerated Failure Time (AFT): $S_{AFT}(\tau|\eta, S_0) = S_0(\frac{\tau}{\exp(\eta)})$
  - Proportional Odds (PO): $S_{PO}(\tau|\eta, S_0) = \frac{S_0(\tau)}{\exp(-\eta)+(1-\exp(-\eta))S_0(\tau)}$

  Models that predict linear predictors will make assumptions about the model form and therefore dictate sensible choices of M, for example the Cox model assumes a PH form. This does not mean other choices of M cannot be specified but that interpretation may be more difficult (section 5.6). The model form can be treated as a hyper-parameter to tune.

- C: Compositor returning the composed distribution, $\zeta := C(M, \hat{\eta}, \hat{S}_0)$ where ζ has survival function $\zeta.S(\tau) = M(\tau|\hat{\eta}, \hat{S}_0)$.

Pseudo-code for training (algorithm 7) and predicting (algorithm 8) this composition as a model 'wrapper' with sensible parameter choices (section 5.6) is provided in appendix A.

5.4.2. C2) Survival Time → Distribution

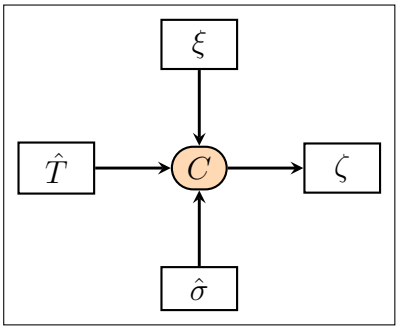


Figure 20: Survival time ($\hat{T}$) to distribution (ζ) composition. Parameters: $\hat{\sigma}$ – Estimated scale parameter; ξ – Assumed survival distribution.

This is a prediction-level MOC that composes a distribution from a predicted survival time and assumed location-scale distribution. The composition (fig. 20) requires:

- $\hat{T}$: A predicted survival time. As with the previous composition, this is tunable. In theory any continuous ranking could replace $\hat{T}$, though the resulting distribution may not be sensible (section 5.6).

- ξ: A specified location-scale distribution, $\xi(\mu, \sigma)$, e.g. Normal distribution.
- $\hat{\sigma}$: Estimated scale parameter for the distribution. This can be treated as a hyper-parameter or predicted by another model.
- C: Compositor returning the composed distribution $\zeta := C(\xi, \hat{T}, \hat{\sigma}) = \xi(\hat{T}, \hat{\sigma})$.

Pseudo-code for training (algorithm 9) and predicting (algorithm 10) this composition as a model 'wrapper' with sensible parameter choices (section 5.6) is provided in appendix A.

5.4.3. C3) Distribution → Survival Time Composition

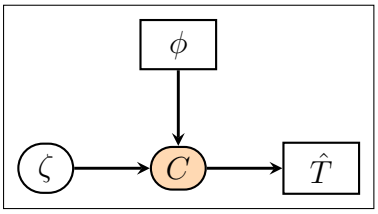


Figure 21: Distribution (ζ) to survival time ($\hat{T}$) composition. Parameters: ϕ – Summary method.

This is a prediction-level SOC that composes a survival time from a predicted distribution. Any paper that evaluates a distribution on concordance is implicitly using this composition in some manner. Not acknowledging the composition leads to unfair model comparison (section 5.3.3). The composition (fig. 21) requires:

- ζ: A predicted survival distribution, which again is 'tunable'.
- ϕ: A distribution summary method. Common examples include the mean, median and mode. Other alternatives include distribution quantiles, $\zeta.F^{-1}(\alpha)$, $\alpha \in [0, 1]$; α could be tuned as a hyper-parameter.
- C: Compositor returning composed survival time predictions, $\hat{T} := C(\phi, \zeta) = \phi(\zeta)$.

Pseudo-code for training (algorithm 11) and predicting (algorithm 12) this composition as a model 'wrapper' with sensible parameter choices (section 5.6) is provided in appendix A.

5.4.4. C4) Survival Model Averaging

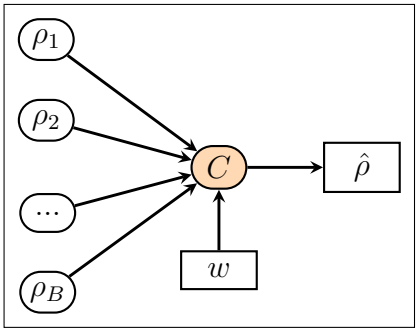


Figure 22: Survival model averaging composition. $\rho_1, ..., \rho_B$ are B predictions of the same return type (time, ranking, distribution) and $\hat{\rho}$ is the averaged prediction. Parameters: $w = w_1, ..., w_B$ – Weights summing to one.

Ensembling is likely the most common composition in machine learning. In survival it is complicated slightly as multiple prediction types means one of two possible compositions is utilised to average predictions. The (fig. 22) composition requires:

- $\rho = \rho_1, ..., \rho_B$: B predictions (not necessarily from the same model) of the same type: ranking, survival time or distribution; again 'tunable'.
- $w = w_1, ..., w_B$: Weights that sum to one.
- C: Compositor returning combined predictions, $\hat{\rho} := C(\rho, w)$ where $C(\rho, w) = \frac{1}{B}\sum_{i=1}^{B} w_i\rho_i$, if ρ are ranking of survival time predictions; or $C(\rho, w) = \zeta$ where ζ is the distribution defined by the survival function $\zeta.S(\tau) = \frac{1}{B}\sum_{i=1}^{B} w_i\rho_i.S(\tau)$, if ρ are distribution predictions.

Pseudo-code for training (algorithm 13) and predicting (algorithm 14) this composition as a model 'wrapper' with sensible parameter choices (section 5.6) is provided in appendix A.

5.4.5. C5) Survival to Regression Data

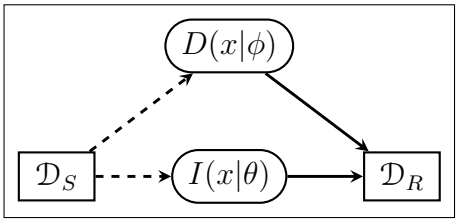


Figure 23: Survival ($\mathcal{D}_S$) to regression ($\mathcal{D}_R$) data composition. Dashed lines represent a choice in the workflow. Parameters: I – Imputation method (section 5.4.5.2); θ – Hyper-parameters of I; D – Deletion method (section 5.4.5.1); ϕ – Hyper-parameters of D.

This is a data-level SOC that transforms survival data to regression data by either removing censored observations or 'imputing' survival times. This composition is frequently incorrectly utilised (section 5.3.3) and therefore more detail is provided here than previous compositions. Note that the previous compositions were prediction-level transformations that occur after a survival model makes a prediction, whereas this composition is on a data-level and can take place before model training or predicting.

In Statistics, there are only two methods for removing 'missing' values: deletion and imputation; both of these have been attempted for censoring.

Censoring can be beneficial, harmful, or neutral; each will affect the data differently if deleted or imputed. Harmful censoring occurs if the reason for censoring is negative, for example drop-out due to disease progression. Harmful censoring indicates that the true survival time is likely soon after the censoring time. Beneficial censoring occurs if censoring is positive, for example drop-out due to recovery. This indicates that the true survival time is likely far from the censoring time. Finally neutral censoring occurs when no information can be gained about the true survival time from the censoring time. Whilst the first two of these can be considered to be dependent on the outcome, neutral censoring is often the case when censoring is independent of the outcome conditional on the data, which is a standard assumption for the majority of survival models and measures.

5.4.5.1. Deletion

Deletion is the process of removing observations from a dataset. This is usually seen in 'complete case analysis' in which observations with 'missingness', covariates with missing values, are removed from the dataset. In survival analysis this method is somewhat riskier as the subjects to delete depend on the outcome and not the features. Three methods are considered, the first two are a more brute-force approach whereas the third allows for some flexibility and tuning.

Complete Deletion Deleting all censored observations is simple to implement with no computational overhead. Complete deletion results in a smaller regression dataset, which may be significantly smaller if the proportion of censoring is high. If censoring is uninformative, the dataset is suitably large and the proportion of censoring suitably low, then this method can be applied without further consideration. However if censoring is informative then deletion will add bias to the dataset, although the 'direction' of bias cannot be known in advance. If censoring is harmful then censored observations will likely have a similar profile to those that died, thus removing censoring will artificially inflate the proportion of those who survive. Conversely if censoring is beneficial then censored observations may be more similar to those who survive, thus removal will artificially inflate the proportion of those who die.

Omission Omission is the process of omitting the censoring indicator from the dataset, thus resulting in a regression dataset that assumes all observations experienced the event. Complete deletion results in a smaller dataset of dead

patients, omission results in no sample size reduction but the outcome may be incorrect. This reduction strategy is likely only justified for harmful censoring. In this case the true survival time is likely close to the censoring time and therefore treating censored observations as dead may be a fair assumption.

IPCW If censoring is conditionally-outcome independent then deletion of censored events is possible by using Inverse Probability of Censoring Weights (IPCW). This method has been seen several times throughout this thesis in the context of models and measures. It has been formalised as a composition technique by Vock *et al.* (2016) [314] although their method is limited to binary classification. Their method weights the survival time of uncensored observations by $w_i = 1/\hat{G}_{KM}(T_i)$ and deletes censored observations, where $\hat{G}_{KM}$ is the Kaplan-Meier estimate of the censoring distribution fit on training data. As previously discussed, one could instead consider the Akritas (or any other) estimator for $\hat{G}_{KM}$.

Whilst this method does provide a 'safer' way to delete censored observations, there is not a necessity to do so. Instead consider the following weights

$$w_i = \frac{\Delta_i + \alpha(1 - \Delta_i)}{\hat{G}_{KM}(T_i)} \tag{5.4.1}$$

where $\alpha \in [0, 1]$ is a hyper-parameter to tune. Setting $\alpha = 1$ equally weights censored and uncensored observations and setting $\alpha = 0$ recovers the setting in which censored observations are deleted. It is assumed $\hat{G}_{KM}$ is set to some very small ϵ when $\hat{G}_{KM}(T_i) = 0$. When $\alpha \neq 0$ this becomes an imputation method, other imputation methods are now discussed.

5.4.5.2. Imputation

Imputation methods estimate the values of missing data conditional on non-missing data and other covariates. Whilst the true value of the missing data can never be known, by carefully conditioning on the 'correct' covariates, good estimates for the missing value can be obtained to help prevent a loss of data. Imputing outcome data is more difficult than imputing covariate data as models are then trained on 'fake' data. However a poor imputation should still be clear when evaluating a model as testing data remains un-imputed. By imputing censoring times with estimated survival times, the censoring indicator can be removed and the dataset becomes a regression dataset.

Gamma Imputation Gamma imputation [142] incorporates information about whether censoring is harmful, beneficial, or neutral. The method imputes survival times by generating times from a shifted proportional hazards model

$$h(\tau) = h_0(\tau) \exp(\eta + \gamma) \tag{5.4.2}$$

where η is the usual linear predictor and $\gamma \in \mathbb{R}$ is a hyper-parameter determining the 'type' of censoring such that $\gamma > 0$ indicates harmful censoring, $\gamma < 0$ indicates beneficial censoring, and $\gamma = 0$ is neutral censoring. This imputation

method has the benefit of being tunable as γ is a hyper-parameter and there is a choice of variables to condition the imputation. No independent experiments exist studying how well this method performs, nor discussing the theoretical properties of the method.

MRL The Mean Residual Lifetime (MRL) estimator has been previously discussed in the context of SVMs (section 3.5.2). Here the estimator is extended to serve as an imputation method. Recall the MRL function, $MRL(\tau|\hat{S}) = \int_\tau^\infty \hat{S}(u)\ du/\hat{S}(\tau)$, where $\hat{S}$ is an estimate of the survival function of the underlying survival distribution (e.g. $\hat{S}_{KM}$). The MRL is interpreted as the expected remaining survival time after the time-point τ. This serves as a natural imputation strategy where given the survival outcome (T_i, Δ_i), the new imputed time T_i' is given by

$$T_i' = T_i + (1 - \Delta_i)MRL(T_i|\hat{S}) \tag{5.4.3}$$

where $\hat{S}$ would be fit on the training data and could be an unconditional estimator, such as Kaplan-Meier, or conditional, such as Akritas. The resulting survival times are interpreted as the true times for those who died and the expected survival times for those who were censored.

Buckley-James Buckley-James [36] is another imputation method discussed earlier (section 3.4). The Buckley-James method uses an iterative procedure to impute censored survival times by the conditional expectation given censoring times and covariates [319]. Given the survival tuple for an outcome (T_i, Δ_i), the new imputed time T_i' is

$$T_i' = \begin{cases} T_i, & \Delta_i = 1 \\ X_i\hat{\beta} + \frac{1}{\hat{S}_{KM}(e_i)} \sum_{e_i < e_k} \hat{p}_{KM}(e_k)e_k & \Delta_i = 0 \end{cases} \tag{5.4.4}$$

where $\hat{S}_{KM}$ is the Kaplan-Meier estimator of the survival distribution estimated on training data and with associated pmf $\hat{p}_{KM}$ and $e_i = T_i - X_i\hat{\beta}$ where $\hat{\beta}$ are estimated coefficients of a linear regression model fit on (X_i, T_i). Given the least squares approach, more parametric assumptions are made than other imputation methods and it is more complex to separate model fitting from imputation. Hence, this imputation may only be appropriate on a limited number of data types.

Alternative Methods Other methods have been proposed for 'imputing' censored survival times though with either less clear discussion or to no benefit. Multiple imputation by chained equations (MICE) has been demonstrated to perform well with covariate data and even outcome data (in a non-survival setting). However no adaptations have been developed to incorporate censoring times into the imputation and therefore is less informative than Gamma imputation.

Re-calibration of censored survival times [313] uses an iterative update procedure to 're-calibrate' censoring times however the motivation behind the method is not sufficiently clear to be of interest in general survival modelling tasks outside of the authors' specific pipelines.

Finally parametric imputation is defined by making random draws from truncated probability distributions and adding these to the censoring time [253, 255]. Whilst this method is arguably the simplest method and will lead to a sufficiently random sample, i.e. not one skewed by the imputation process, in practice the randomness leads to unrealistic results, with some imputed times being very far from the original censoring times and some being very close.

5.4.5.3. The Decision to Impute or Delete

Deletion methods are simple to implement and fast to compute however they can lead to biasing the data or a significant sample reduction if used incorrectly. Imputation methods can incorporate tuning and have more relaxed assumptions about the censoring mechanism, though they may lead to over-confidence in the resulting outcome and therefore add bias into the dataset. In some cases, the decision to impute or delete is straightforward, for example if censoring is uninformative and only few observations are censored then complete deletion is appropriate. If it is unknown if censoring is informative then this can crudely be estimated by a benchmark experiment. Classification models can be fit on $\{(X_1, \Delta_1), ..., (X_n, \Delta_n)\}$ where $(X_i, \Delta_i) \in \mathcal{D}_0$. Whilst not an exact test, if any model significantly outperforms a baseline, then this may indicate censoring is informative. This is demonstrated in table 8, in which a logistic regression outperforms a featureless baseline in correctly predicting if an observation is censored when censoring is informative, but is no better than the baseline when censoring is uninformative.

Table 8: Estimating censoring dependence by prediction. The datasets are defined in section 7.2.6.2, `Sim1` is informative censoring and `Sim7` is uninformative. Logistic regression is compared to a featureless baseline with the Brier score with standard errors. Censoring can be significantly predicted to 95% confidence when informative (`Sim1`) but not when uninformative (`Sim7`).

Data	Baseline	Logistic Regression
`Sim1`	0.20 (0.14, 0.26)	0.02 (0.01, 0.03)
`Sim7`	0.19 (0.14, 0.24)	0.16 (0.13, 0.19)

5.5. Novel Survival Reductions

This section collects the various strategies and settings discussed previously into complete reduction workflows. Table 9 lists the reductions discussed in this section with IDs for future reference. All strategies are described by visualising a graphical pipeline and then listing the composition steps required in fitting and predicting.

This section only includes novel reduction strategies and does not provide a survey of pre-existing strategies. This limitation is primarily due to time (and page) constraints as every method has very distinct workflows that require complex exposition. Well-established strategies are briefly mentioned below and fu-

ture research is planned to survey and compare all strategies with respect to empirical performance (i.e. in benchmark experiments).

Two prominent reductions are 'landmarking' [309] and piecewise exponential models [89]. Both are reductions for time-varying covariates and hence outside the scope of this thesis. Relevant to this thesis scope is a large class of strategies that utilise 'discrete time survival analysis' [298]; these strategies include reductions (R7) and (R8). Methodology for discrete time survival analysis has been seen in the literature for the past three decades [197]. The primary reduction strategy for discrete time survival analysis is implemented in the R package **discSurv** [321]; this is very similar to (R7) except that it enforces stricter constraints in the composition procedures and forces a 'discrete-hazard' instead of 'discrete-survival' representation (section 5.5.7.2).

Table 9: Survival reductions in section 5.5. First column is a unique identifier for the strategy, second column is the original suvival task of interest, third column is the reduced task that will be solved as a surrogate in the workflow.

ID	Original Survival Task	Reduced Task
R1)	Probabilistic	Probabilistic Regression
R2)	Probabilistic	Deterministic Regression
R3)	Deterministic	Deterministic Regression
R4)	Deterministic	Probabilistic Distribution
R5)	Probabilistic	Deterministic Regression
R6)	Ranking	Deterministic Regression
R7)	Probabilistic	Probabilistic Classification
R8)	Deterministic	Probabilistic Classification

5.5.1. R1) Probabilistic Survival → Probabilistic Regression

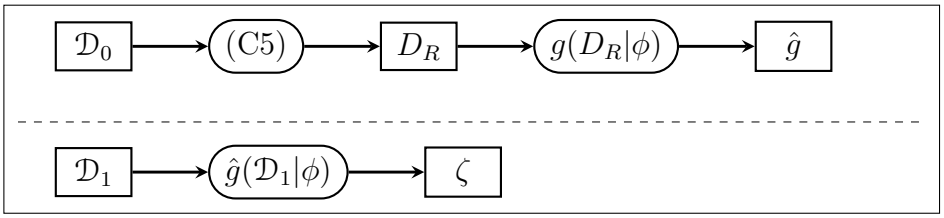


Figure 24: Probabilistic survival to probabilistic regression reduction. Top row is fitting step and bottom is predicting. Key: training data, $\mathcal{D}_0$; survival to regression data composition (section 5.4.5), (C5); transformed data, D_R; probabilistic regression model, g; model hyper-parameters, ϕ; fitted model, $\hat{g}$; testing data, $\mathcal{D}_1$; distribution predictions, ζ.

This is perhaps the most natural reduction strategy as the survival task can be thought of as probabilistic regression with censoring. Steps and compositions of the reduction (fig. 24):

Fit

F1) A survival dataset, $\mathcal{D}_0$, is composed with (C5) to a regression dataset, $\mathcal{D}_R$.

F2) A *probabilistic* regression model, g, with hyper-parameters, ϕ, is fit on the composed regression data. It is important to select a model that will only predict distributions supported over $\mathbb{R}_{\geq 0}$ in order to reflect the survival setting.

Predict

P1) Testing survival data, $\mathcal{D}_1$, is passed to the trained regression model, $\hat{g}$, without further data composition, and distributions are predicted, $\zeta = \zeta_1, ..., \zeta_m$.

5.5.2. R2) Probabilistic Survival → Deterministic Regression

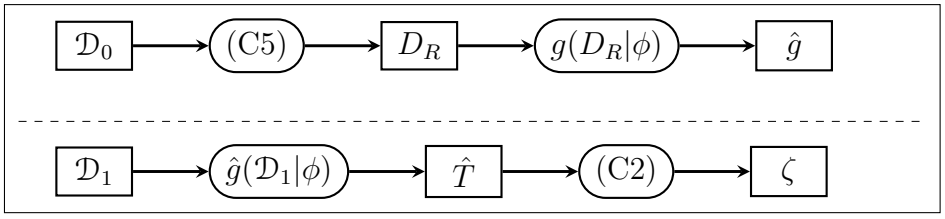


Figure 25: Probabilistic survival to deterministic regression reduction. Top row is fitting step and bottom row is predicting. Key: training data, $\mathcal{D}_0$; survival to regression data composition (section 5.4.5), (C5); transformed data, D_R; deterministic regression model, g; model hyper-parameters, ϕ; fitted model, $\hat{g}$; testing data, $\mathcal{D}_1$; survival time predictions, $\hat{T}$; time to distribution composition (section 5.4.2), (C2); distribution predictions, ζ.

This is almost identical to the previous reduction but utilises deterministic regression models and composition to distribution predictions. Steps and compositions of the reduction (fig. 25):

Fit

F1) A survival dataset, $\mathcal{D}_0$, is composed with (C5) to a regression dataset, $\mathcal{D}_R$.

F2) A *deterministic* regression model, g, with hyper-parameters, ϕ, is fit on the composed regression data. It is important to select a model that will only predict positive values in order to reflect the survival setting.

Predict

P1) Testing survival data, $\mathcal{D}_1$, is passed to the trained regression model, $\hat{g}$, without further data composition, and survival times are predicted, $\hat{T} = \hat{T}_1, ..., \hat{T}_m$.

P2) Survival times are composed with (C2) to distribution predictions $\zeta = \zeta_1, ..., \zeta_m$.

5.5.3. R3) Deterministic Survival → Deterministic Regression

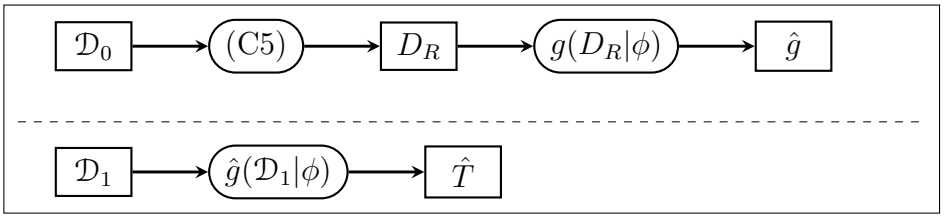


Figure 26: Deterministic survival to deterministic regression reduction. Top row is fitting step and bottom row is predicting. Key: training data, $\mathcal{D}_0$; survival to regression data composition (section 5.4.5), (C5); transformed data, D_R; deterministic regression model, g; model hyper-parameters, ϕ; fitted model, $\hat{g}$; testing data, $\mathcal{D}_1$; survival time predictions, $\hat{T}$.

This reduction is identical to (R2) except (P2) is omitted.

5.5.4. R4) Deterministic Survival → Probabilistic Regression

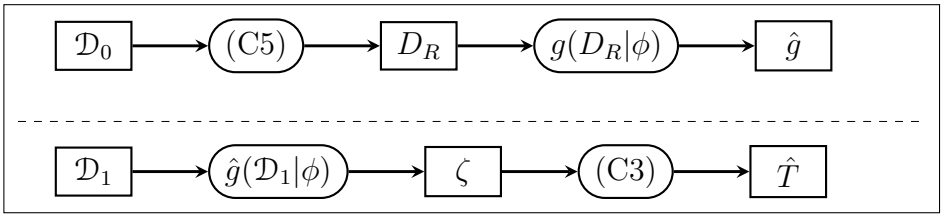


Figure 27: Deterministic survival to probabilistic regression reduction. Top row is fitting step and bottom row is predicting. Key: training data, $\mathcal{D}_0$; survival to regression data composition (section 5.4.5), (C5); transformed data, D_R; probabilistic regression model, g; model parameters, ϕ; fitted model, $\hat{g}$; testing data, $\mathcal{D}_1$; distribution predictions, ζ; distribution to survival time composition (section 5.4.3), (C3); survival time predictions, $\hat{T}$

This is identical to (R1) with an additional composition to survival time. Steps and compositions of the reduction (fig. 27):

Fit

F) Same as (R1).

Predict

P1) Testing survival data, $\mathcal{D}_1$, is passed to the trained regression model, $\hat{g}$, without further data composition, and distributions are predicted, $\zeta = \zeta_1, ..., \zeta_m$.

P2) Distributions are composed with (C3) to survival times $\hat{T} = \hat{T}_1, ..., \hat{T}_m$.

5.5.5. R5) Probabilistic Survival → Deterministic Regression (II)

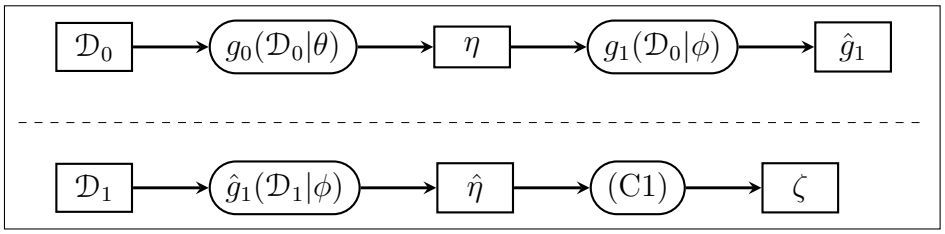


Figure 28: Probabilistic survival to deterministic regression reduction (II). Top row is fitting step and bottom row is predicting. Key: training data, $\mathcal{D}_0$; linear survival model g_0 with parameters θ; fitted linear predictor from g_0, η; deterministic regression model, g_1, with parameters, ϕ; fitted model, $\hat{g}_1$; testing data, $\mathcal{D}_1$; linear predictor predictions, $\hat{\eta}$; linear predictor to distribution composition (section 5.4.1), (C1); distribution prediction, ζ.

These next two reductions utilise deterministic regression to predict linear predictors. This first reduction additionally composes the linear predictor to a distribution prediction. Steps and compositions of the reduction (fig. 28):

Fit

F1) A survival model, g_0, with a linear predictor prediction type is fit on a survival dataset, $\mathcal{D}_0$.

F2) The model is inspected and the fitted linear predictors, η, are returned.

F3) A deterministic regression model, g, is fit on (X_i, η_i) with η_i as the target.

Predict

P1) Testing survival data, $\mathcal{D}_1$, is passed to the trained regression model, $\hat{g}$, and linear predictors are predicted, $\hat{\eta} = \hat{\eta}_1, ..., \hat{\eta}_m$.

P2) Linear predictors are composed with (C1) to survival distributions $\zeta = \zeta_1, ..., \zeta_m$. The most sensible choice of model form for the (C1) composition will be dictated by g_0, e.g. does it have an underlying PH form?

5.5.6. R6) Ranking Survival → Deterministic Regression

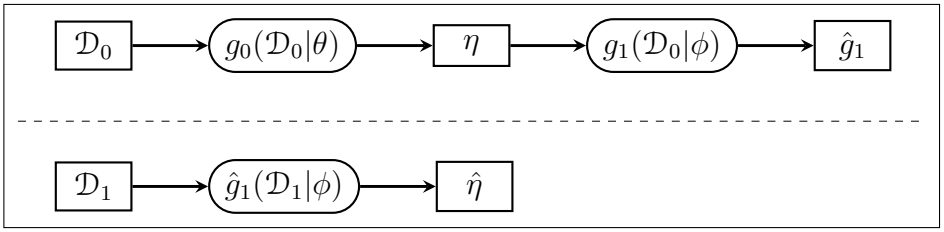


Figure 29: Ranking survival to deterministic regression reduction. Top row is fitting step and bottom row is predicting. Key: training data, $\mathcal{D}_0$; linear survival model g_0 with parameters θ; fitted linear predictor from g_0, η; deterministic regression model, g_1, with parameters, ϕ; testing data, $\mathcal{D}_1$; linear predictor predictions, $\hat{\eta}$.

This reduction is identical to (R5) except (P2) is omitted. Whilst this is categorised as solving a ranking task, the predicted quantities can be interpreted as linear predictors (given the model form specified by g_0).

5.5.7. R7-R8) Survival → Probabilistic Classification

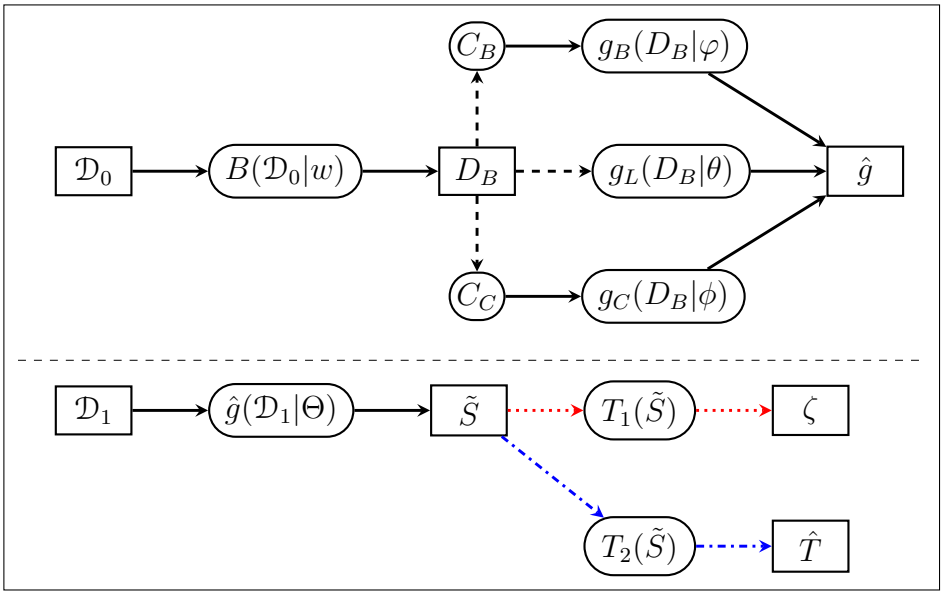


Figure 30: Survival to classification reduction. Top row is fitting and bottom row is predicting. Dashed lines represent a choice in the reduction (alternative compositions). Red dotted lines complete the probabilistic survival reduction (R7) and the blue dash-dotted lines complete the deterministic survival reduction (R8). Key: training data, $\mathcal{D}_0$; binning function, B, with weights, w; binned data, D_B; composition to binary-class classification, C_B; composition to multi-class classification, C_C; binary-class classifier, g_B, with parameters, φ; multi-label classifier, g_L, with parameters, θ; multi-class classifier, g_C, with parameters ϕ; trained classifier, $\hat{g}$ with parameters Θ; testing data, $\mathcal{D}_1$; pseudo-survival probabilities, $\tilde{S}$; composition, T_1, to distribution, ζ; composition, T_2, to survival time, $\hat{T}$.

Two separate reductions are presented in fig. 30 however as both are reductions to probabilistic classification and are only different in the very last step, both are presented in this section. Steps and compositions of the reduction (fig. 30):

Fit

F1) A survival dataset, $\mathcal{D}_0$, is binned, B, with a continuous to discrete data composition (section 5.5.7.1).

F2) A multi-label classification model, with adaptations for censoring, $g_L(D_B|\theta)$, is fit on the transformed dataset, D_B. Optionally, g_L could be further reduced to binary, g_B, or multi-class classification, g_c, (section 5.5.7.4).

Predict

P1) Testing survival data, $\mathcal{D}_1$, is passed to the trained classification model, $\hat{g}$, to predict pseudo-survival probabilities $\tilde{S}$ (or optionally hazards (section 5.5.7.2)).

P2a) Predictions can be composed, T_1, into a survival distribution prediction, $\zeta = \zeta_1, ..., \zeta_m$ (section 5.5.7.6); or,

P2b) Predictions can be composed, T_2, to survival time predictions, $\hat{T} = \hat{T}_1, ..., \hat{T}_m$ (section 5.5.7.7).

Further details for binning, multi-label classification, and transformation of pseudo-survival probabilities are now provided.

5.5.7.1. Composition: Binning Survival Times

An essential part of the reduction is the transformation from a survival dataset to a classification dataset, which requires two separate compositions. The first (discussed here) is to discretise the survival times ($B(\mathcal{D}_0|w)$ in fig. 30) and the second is to merge the survival time and censoring indicator into a single outcome (section 5.5.7.2).

Discretising survival times is achieved by the common 'binning' composition, in which a continuous outcome is discretised into 'bins' according to specified thresholds. These thresholds are usually determined by specifying the width of the bins as a hyper-parameter w.[1] This is a common transformation and therefore further discussion is not provided here. An example is given below with the original survival data on the left and the binned data on the right ($w = 1$).

X	Time (Cont.)	Died
1	1.56	0
2	2	1
3	3.3	1
4	3.6	0
5	4	0

X	Time (Disc.)	Died
1	[1, 2)	0
2	[2, 3)	1
3	[3, 4)	1
4	[3, 4)	0
5	[4, 5)	0

[1] Binning is described here with equal widths but generalises to unequal widths trivially.

5.5.7.2. Composition: Survival to Classification Outcome

The binned dataset still has the unique survival data format of utilising two outcomes for training (time and status) but only making a prediction for one outcome (distribution). In order for this to be compatible with classification, the two outcome variables are composed into a single variable.[1] This is achieved by casting the survival times into a 'wide' format and creating a new outcome indicator.[2] Two outcome transformations are possible, the first represents a discrete survival function and the second represents a discrete hazard function.[3]

Discrete Survival Function Composition In this composition, the data in the transformed dataset represents the discrete survival function. The new indicator is defined as follows,

$$Y_{i;\tau} := \begin{cases} 1, & T_i > \tau \\ 0, & T_i \leq \tau \cap \Delta_i = 1 \\ -1, & T_i \leq \tau \cap \Delta_i = 0 \end{cases} \tag{5.5.1}$$

At a given discrete time τ, an observation, i, is either alive ($Y_{i;\tau} = 1$), dead ($Y_{i;\tau} = 0$), or censored ($Y_{i;\tau} = -1$). Therefore $\hat{P}(Y_{i;\tau} = 1) = \hat{S}_i(\tau)$, motivating this particular choice of representation.

This composition is demonstrated below with the binned data (left) and the composed classification data (right).

X	Time (Disc.)	Died
1	[1, 2)	0
2	[2, 3)	1
3	[3, 4)	1
4	[3, 4)	0
5	[4, 5)	0

X	[1,2)	[2,3)	[3,4)	[4,5)
1	-1	-1	-1	-1
2	1	0	0	0
3	1	1	0	0
4	1	1	-1	-1
5	1	1	-1	-1

Discrete Hazard Function Composition In this composition, the data in the transformed dataset represents the discrete hazard function. The new indicator is defined as follows,

$$Y^*_{i;\tau} := \begin{cases} 1, & T_i = \tau \cap \Delta_i = 1 \\ -1, & T_i = \tau \cap \Delta_i = 0 \\ 0, & \text{otherwise} \end{cases} \tag{5.5.2}$$

[1] This is the first key divergence from other discrete-time classification strategies, which use the censoring indicator as the outcome and the time outcome as a feature.

[2] This is the second key divergence from other discrete-time classification strategies, which keep the data in a 'long' format.

[3] This is the final key divergence from other discrete-time classification strategies, which enforce the discrete hazard representation.

At a given discrete time τ, an observation, i, either experiences the event ($Y^*_{i;\tau} = 1$), experiences censoring ($Y_{i;\tau} = -1$), or neither ($Y_{i;\tau} = 0$). Utilising sequential multi-label classification problem transformation methods (section 5.5.7.4) results in $\hat{P}(Y^*_{i;\tau} = 1) = \hat{h}_i(\tau)$. If methods are utilised that do not 'look back' at predictions then $\hat{P}(Y^*_{i;\tau} = 1) = \hat{p}_i(\tau)$ (section 5.5.7.4).[1]

This composition is demonstrated below with the binned data (left) and the composed classification data (right).

X	Time (Disc.)	Died
1	[1, 2)	0
2	[2, 3)	1
3	[3, 4)	1
4	[3, 4)	0
5	[4, 5)	0

X	[1,2)	[2,3)	[3,4)	[4,5)
1	-1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	-1	0
5	0	0	0	-1

Multi-Label Classification Data In both compositions, survival data t.v.i. $\mathbb{R}^p \times \mathbb{R}_{\geq 0} \times \{0, 1\}$ is transformed to multi-label classification data t.v.i. $\mathbb{R}^p \times \{-1, 0, 1\}^K$ for K binned time-intervals. The multi-label classification task is defined in section 5.5.7.4 with possible algorithms.

The discrete survival representation has a slightly more natural interpretation and is 'easier' for classifiers to use for training as there are more positive events (i.e. more observations alive) to train on, whereas the discrete hazard representation will have relatively few events in each time-point. However the hazard representation leads to more natural predictions (section 5.5.7.6).

A particular bias that may easily result from the composition of survival to classification data is now discussed.

5.5.7.3. Reduction to Classification Bias

The reduction to classification bias is commonly known [333] but is reiterated briefly here as it must be accounted for in any automated reduction to classification workflow. This bias occurs when making classification predictions about survival at a given time and incorrectly censoring patients who have not been observed long enough, instead of removing them.

By example, say the prediction of interest is five-year survival probabilities after a particular diagnosis, clearly a patient who has only been diagnosed for three years cannot inform this prediction. The bias is introduced if this patient is censored at five-years instead of being removed from the dataset. The result of this bias is to artificially inflate the probability of survival at each time-point as an unknown outcome is treated as censored and therefore alive.

This bias is simply dealt with by removing patients who have not been alive 'long enough'.[2] Paradoxically, even if a patient is observed to die before the time-point of interest, they should still be removed if they have not been in the dataset

[1] This important distinction is not required in other discrete-time reduction strategies that automatically condition the prediction by including time as a feature.

[2] Accounting for this bias is only possible if the study start and end dates are known, as well as the date the patient entered the study.

'long enough' as failing to do so will result in a bias in the opposite direction, thus over-inflating the proportion of dead observations.

Accounting for this bias is particularly important in the multi-label reduction as the number of observable patients will decrease over time due to censoring.

5.5.7.4. Multi-Label Classification Algorithms

As the work in this section is completely out of the thesis scope, the full text is in appendix B. The most important contributions from this section are:

- Reviewing problem transformation methods [296] for multi-label classification;
- Identifying that only binary relevance, nested stacking, and classifier chains are appropriate in this reduction; and
- Generalising these methods into a single wrapper for any binary classifier, the 'LWrapper'.

5.5.7.5. Censoring in Classification

Classification algorithms cannot natively handle the censoring that is included in the survival reduction, but this can be incorporated using one of two approaches.

Multi-Class Classification All multi-label datasets can also handle multi-class data, hence the simplest way in which to handle censoring is to make multi-class predictions in each label for the outcome Y_τ t.v.i.$\{-1, 0, 1\}$. Many off-shelf classification learners can make multi-class predictions natively and simple reductions exist for those that cannot. As a disadvantage to this method, classifiers would then predict if an individual is dead or alive or censored (each mutually exclusive), and not simply alive or dead. Though this could be perceived as an advantage when censoring is informative as this will accurately reflect a real-world competing-risks set-up.

Subsetting/Hurdle Models For this approach, the multi-class task is reduced to two binary class tasks: first predict if a subject is censored or not (dead or alive) and only if the prediction for censoring is below some threshold, $\alpha \in [0, 1]$, then predict if the subject is alive or not (dead or censored). If the probability of censoring is high in the first task then the probability of being alive is automatically set to zero in the final prediction, otherwise the prediction from the second task is used. Any classifier can utilise this approach and it has a meaningful interpretation, additionally α is a tunable hyper-parameter. The main disadvantage is increases to storage and run-time requirements as double the number of models may be fit.

Once the datasets have been composed to classification datasets and censoring is suitably incorporated by either approach, then any probabilistic classification model can be fit on the data. Predictions from these models can either be composed to a distribution prediction (R7) or a survival time prediction (R8).

5.5.7.6. R7) Probabilistic Survival → Probabilistic Classification

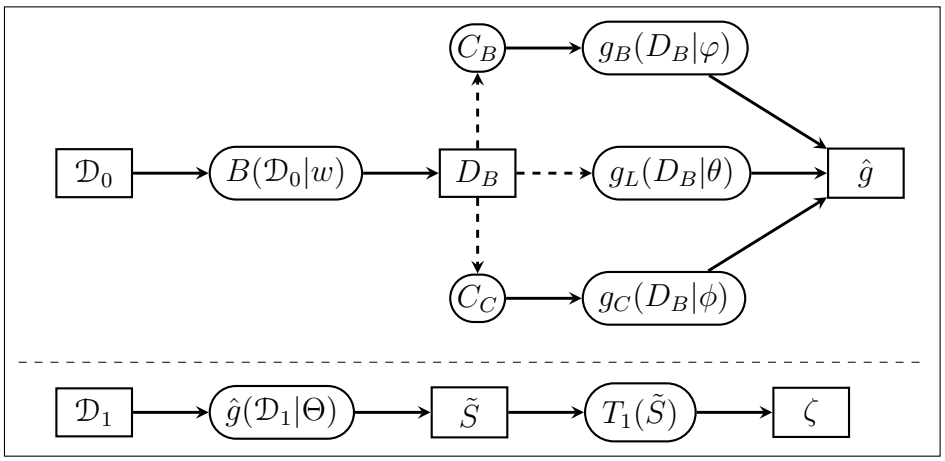


Figure 31: Probabilistic survival to probabilistic reduction. See fig. 30 for key.

This final part of the (R7) reduction is described separately for discrete hazard and survival representations of the data (section 5.5.7.2).

Discrete Hazard Representation In this representation recall that predictions of the positive class, $P(Y_\tau = 1)$, are estimating the quantity $h(\tau)$. These predictions provide a natural and efficient transformation from predicted hazards to survival probabilities. Let $\hat{h}_i$ be a predicted hazard function for some observation i, then the survival function for that observation can be found with a Kaplan-Meier type estimator,

$$\tilde{S}_i(\tau^*) = \prod_{\tau} 1 - \hat{h}_i(\tau) \tag{5.5.3}$$

Now predictions are for a pseudo-survival function, which is 'pseudo' as it is not right-continuous. Resolving this is discussed below.

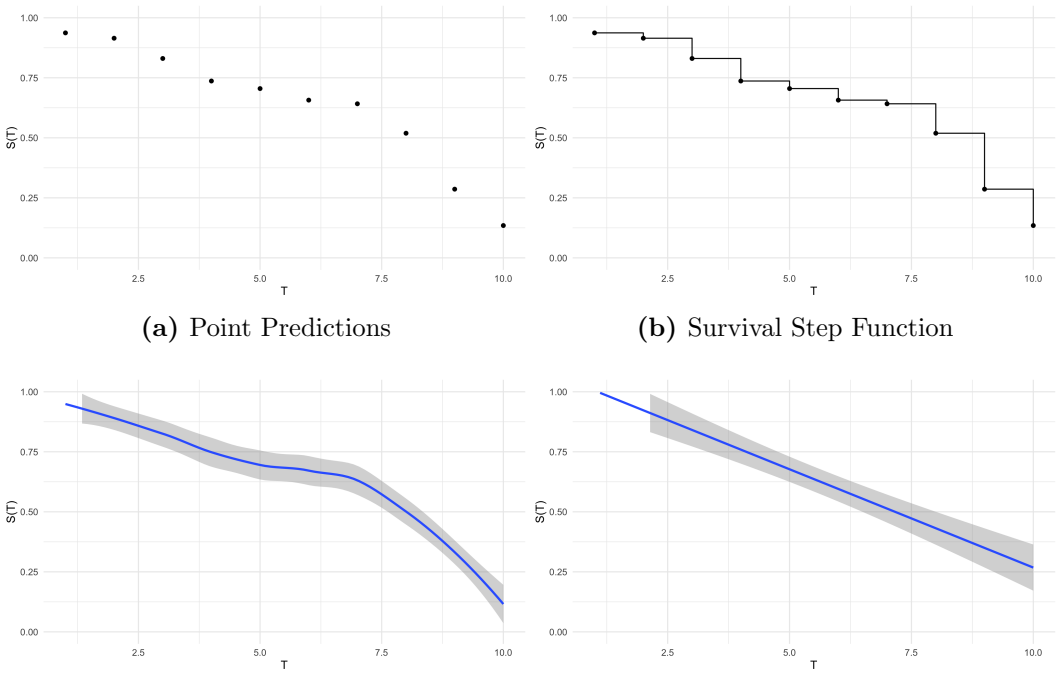
Discrete Survival Representation In this representation, $P(Y_\tau = 1)$ is estimating $S(\tau)$, which means that predictions from a classification model result in discrete point predictions and not a right-continuous function. More importantly, there is no guarantee that a non-increasing function will be predicted, i.e. there is no guarantee that $P(Y_j = 1) < P(Y_i = 1)$, for time-points $j > i$.

Unfortunately there is no optimal way of dealing with predictions of this sort and 'mistakes' of this kind have been observed in some software implementation. One point to note is that in practice these are quite rare as the probability of survival will always decrease over time. Therefore the 'usual' approach is quite 'hacky' and involves imputing increasing predictions with the previous prediction, formally,

$$\tilde{S}(i+1) := \min\{P(Y_{i+1} = 1), P(Y_i = 1)\}, \forall i = \mathbb{R}_{\geq 0} \tag{5.5.4}$$

assuming $\tilde{S}(0) = 1$. Future research should seek more robust alternatives.

Right-Continuous Survival Function From either representation, a non-increasing but non-continuous pseudo-survival function, $\tilde{S}$, is now predicted. Creating a right-continuous function ('$T_1(\tilde{S})$' in fig. 31) from these point predictions (fig. 32 (a)) is relatively simple and well-known with accessible off-shelf software. At the very least, one can assume a constant hazard rate between predictions and cast them into a step function (fig. 32 (b)). This is a fairly common assumption and is usually valid as bin-width decreases. Alternatively, the point predictions can be smoothed into a continuous function with off-shelf software, for example with polynomial local regression smoothing (fig. 32 (c)) or generalised linear smoothing (fig. 32 (d)). Whichever method is chosen, the survival function is now non-increasing right-continuous and the (R7) reduction is complete.



(a) Point Predictions

(b) Survival Step Function

(c) Local polynomial regression smoothing

(d) Generalised linear smoothing

Figure 32: Survival function as a: point prediction (a), step function assuming constant risk (b), local polynomial regression smoothing (c), and generalised linear smoothing (d). (c) and (d) computed with **ggplot2** [322].

5.5.7.7. R8) Deterministic Survival → Probabilistic Classification

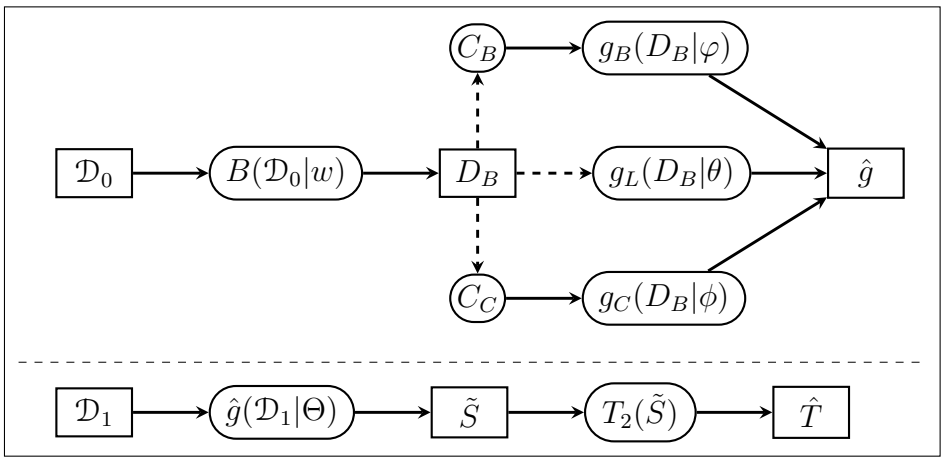


Figure 33: Deterministic survival to probabilistic reduction. See fig. 30 for key.

Predicting a deterministic survival time from the multi-label classification predictions is relatively straightforward and can be viewed as a discrete analogue to (C3) (section 5.4.3). For the discrete hazard representation, one can simply take the predicted time-point for an individual to be time at which the predicted hazard probability is highest however this could easily be problematic as there may be multiple time-points at which the predicted hazard equals 1. Instead it is cleaner to first cast the hazard to a pseudo-survival probability (section 5.5.7.6) and then treat both representations the same.

Let $\tilde{S}_i$ be the predicted multi-label survival probabilities for an observation i s.t. $\tilde{S}_i(\tau)$ corresponds with $\hat{P}(Y_{i;\tau} = 1)$ for label $\tau \in \mathcal{K}$ where $Y_{i;\tau}$ is defined in section 5.5.7.2 and $\mathcal{K} = \{1, ..., K\}$ is the set of labels for which to make predictions. Then the survival time transformation is defined by

$$T_2(\tilde{S}_i) = \inf\{\tau \in \mathcal{K} : \tilde{S}_i(\tau) \leq \beta\} \tag{5.5.5}$$

for some $\beta \in [0, 1]$.

This is interpreted as defining the predicted survival time as the first time-point in which the predicted probability of being alive drops below a certain threshold β. Usually $\beta = 0.5$, though this can be treated as a hyper-parameter for tuning. This composition can be utilised even if predictions are not non-increasing, as only the first time the predicted survival probability drops below the threshold is considered. With this composition the (R8) reduction is now complete.

5.6. Choices and Defaults

Before concluding the chapter, this brief section describes a common problem that occurs when programming pipelines and how this thesis (and implementation in **mlr3proba**) addresses this.

Many Choices Implementation of any of these pipelines leads to an important trade-off between user-choice and sensible decisions. When programming any software, the more choice that is given to the user, the higher the potential to make less sensible decisions; in the extreme as the number of user possibilities tends to infinity, the probability of a user selecting a sensible decision will tend to zero. On the other hand, if decisions are fully-restricted to sensible decisions then the user's choice is also fully-restricted by the subjective concept of 'sensible'.

To illustrate the problem, below are three possible choices that could be made with the compositors in section 5.4:

- A linear predictor predicted by a CPH could be composed with a PH-ANN-predicted baseline and AFT model form to a full distribution.
- A survival time predicted by a regression SSVM could be composed with a Gompertz baseline and PO model form to a full distribution.
- A survival time could be composed by taking the 42nd quantile from a survival distribution predicted by a random survival forest.

Each choice lacks a meaningful interpretation however there is no apriori reason why they should yield 'bad' predictions and all could be considered in a benchmark experiment. Dismissing these examples as 'not sensible' may lead to dismissing the optimal model with respect to predictive performance.

Sensible Defaults It has been demonstrated that the choice of defaults vastly influences human decision making [150], which is known as the '(endogenous) default effect'. This effect extends to computer science and parameter defaults. Setting sensible defaults for parameters encourages users towards using these defaults in their code and this 'sensible defaults' design principle is routinely used in programming software.[1]

This thesis advocates for a slight adaptation to the 'sensible defaults' design principle: non-proprietary open-source software should apply the sensible defaults principle whilst allowing users to make any choice that is possible (even if not sensible); whereas proprietary software should only allow sensible choices. This distinction is important from an ethical standpoint: in the latter case users may not be domain-experts and therefore the developer could be considered liable for negative consequences of building models from non-sensible choices.

[1] No specific reference for the 'sensible defaults' principle could be found, though it is often seen as a direct consequence of the 'convention over configuration' principle.

5.7. Conclusions

This chapter introduced composition and reduction to survival analysis and formalised specific strategies. Formalising these concepts allows for better quality of research and most importantly improved transparency. Clear interface points for hyper-parameters and compositions allow for reproducibility that was previously obfuscated by unclear workflows and imprecise documentation for pipelines.

Additionally, composition and reduction improves accessibility. Reduction workflows vastly increase the number of machine learning models that can be utilised in survival analysis, thus opening the field to those whose experience is limited to regression or classification. Formalisation of workflows allows for precise implementation of model-agnostic pipelines as computational objects, as opposed to functions that are built directly into an algorithm without external interface points.

Finally, predictive performance is also increased by these methods, which is most prominently the case for the survival model averaging compositor (C4) (as demonstrated by RSFs).

All compositions in this chapter, as well as (R1)-(R6), have been implemented in **mlr3proba** (section 6.4) with the **mlr3pipelines** [21] interface. The reductions to classification will be implemented in a near-future update. Additionally the **discSurv** package [321] will be interfaced as a **mlr3proba** pipeline to incorporate further discrete-time strategies.

The compositions (C1) and (C3) are included in the benchmark experiment in chapter 7 so that every tested model can make probabilistic survival distribution predictions as well as deterministic survival time predictions. Future research will benchmark all the pipelines in this chapter and will cover algorithm and model selection, tuning, and comparison of performance. Strategies from other papers will also be explored. The best performing models from these experiments will then be analytically compared to the best-performing models from chapter 7.

Chapter 6

Software Packages

The work in the previous chapters of this thesis are now consolidated for practical use in R [245] software packages. Several packages (table 10) have been developed over the course of this PhD, all of which are open source and freely available over CRAN and/or GitHub. This chapter begins with providing a setting for the most important of these packages and detailing why they were required, next the three primary packages are detailed in individual sections, each of which has been submitted for publication.

6.1. Introduction

This chapter introduces three software packages, or 'toolboxes'. The term 'toolbox' is often used synonymously with 'software package', here the term specifically identifies a package with a suite of functions, or 'tools', for use in a specific purpose. These three toolboxes are **set6** [278], **distr6** [277], and **mlr3proba** [281], respectively designed to solve the problems of: constructing and manipulating mathematical sets; constructing and manipulating probability distributions; and probabilistic supervised machine learning. As well as these three packages, a fourth, **R62S3** [274], was developed in order to further the R6 [43] object-oriented paradigm, this is briefly discussed in section 6.1.3.1. All packages developed for this thesis are listed with their GitHub organisation for reference and their dependencies on each other in table 10.

6.1.1. Overview to Packages and Their Relationships

mlr3proba (section 6.4) is a machine learning interface for probabilistic supervised learning and contains models and measures for survival analysis. The name **mlr3proba** is derived from 'mlr3', the universe in which it lives, and 'proba', for probabilistic (supervised learning). All models in **mlr3proba** solve probabilistic tasks that predict (or estimate) a probability distribution. As **mlr3proba** makes use of the R6 object-oriented paradigm (section 6.1.2), it was sensible to introduce R6 object-oriented probability distributions.

The '`dpqr`' functions are the primary methods for interacting with probability distributions in R. An alternative was provided by **distr** [258], which implements

Table 10: Published packages, respective GitHub organisation, and dependencies on each other.

Package	Org	Dependencies
R62S3	xoopR	None
set6	xoopR	None
param6 [276]	xoopR	**set6**
distr6	alan-turing-institute	**R62S3**, **set6**, **param6**
mlr3proba	mlr-org	**distr6**
mlr3extralearners [280]	mlr-org	**mlr3proba**, **survivalmodels**
mlr3benchmark [279]	mlr-org	None
survivalmodels [275]	RaphaelS1	None

distributions via the object-oriented programming (OOP) paradigm, S4. Utilising **distr** in **mlr3proba** would lead to a clash of paradigms, which is poor practice in OOP toolboxes. **distr6** (section 6.3) introduces probability distributions as R6 objects and allows construction, manipulation, and composition of probability distributions. Models in **mlr3proba** return predictions of probability distributions as **distr6** objects. **distr6** was designed to be modular and therefore not to cater to extraneous tasks, such as defining distribution and parameter domains and supports, which are naturally handled by mathematical sets. The parameter set interface is currently in the process of being transitioned to **param6**, which is not discussed further as it is still very new and requires further user-testing. Similarly to **mlr3proba** requiring an R6 implementation of distributions, **distr6** required an R6 implementation of mathematical sets.

distr6 required an R6 interface that could handle complex sets, such as multi-dimensional Cartesian products, which prior packages could not handle satisfactorily (section 6.2.2). **set6** (section 6.2) uses R6 to implement a scalable interface with lazy-evaluation and symbolic-representation, which allows handling of infinite (or very large) sets. The package supports finite and infinite sets, composition of sets, and set operations. In addition, **set6** includes methods for validation and containedness checks, which improves efficiency in **distr6** for setting parameter values.

All three packages utilise the relatively new R6 object-oriented paradigm, which introduces new syntax, structures, and important object-oriented concepts to R. As probability distributions and mathematical sets are such foundational concepts in mathematics and statistics, **R62S3** (section 6.1.3.1) was created to lessen the R6 learning curve. **R62S3** makes R6 packages more accessible by converting R6 class methods to S3 or S4 dispatch functions, thus providing new users with a familiar interface but maintaining an object-oriented structure.

6.1.2. R and R6

All of the implemented packages depend on the R6 object-oriented (OO) paradigm, this is a relatively new OO paradigm and is quite different from its predecessors S3 and S4. It is therefore important to demonstrate why OOP and R are chosen as well as why R6 is preferred to S3 and S4.

Why Object-Oriented Programming? There are several programming paradigms, but the most common in machine learning are procedural, object-oriented, and functional programming.[1] Object-oriented programming (OOP) requires users to define classes with fields and methods that mutate them. These classes may relate to each other in some way (e.g. composition, inheritance) and may change depending on the other's state. Functional programming (FP) treats every line (or group) of code as independent variables/functions with inputs and outputs. Listings 1 and 2 demonstrate programming two 'animals' to 'shout' in Python and R respectively. In the object-oriented Python code two classes are created, `Duck` and `Dog`, which could even inherit from a common `Animal` class. In the functional R code, there is no concept of a class or object, instead there is a `shout` function that is independent of the other defined variables; variables are then assigned values and passed to this function.

Listing 1 Python code for constructing two classes with one method and calling these methods. First two classes are defined, then a method with the same name within these, finally the method is called on the constructed objects.

```
>>> class Duck:
...   def shout(self):
...     print("QUACK!")
...
>>> class Dog:
...   def shout(self):
...     print("BARK!")
...
>>> Duck().shout()
>>> Dog().shout()
```

Listing 2 R code using 'base' functionality (no OOP) for making two 'animals' 'shout'. Without a concept of an object or class, a 'method' is assigned directly to a variable and then passed to a function.

```
> duck <- "QUACK!"
> dog <- "BARK!"
> shout <- function(x) cat(x)
> shout(duck)
> shout(dog)
```

In this small example, there may not be a clear advantage of OOP over FP, so instead take the example of something more complex like a probability distribution. Probability distributions can have methods, such as density and cumulative distribution functions, but they also have fields, such as parameter sets and their support. Fully representing a probability distribution via functions would require

[1] https://www.tiobe.com/tiobe-index/

one function for every field and method of every distribution. For 30 distributions with 10 fields and methods, that would mean 300 functions, each with different names and attributes; here lies the clear OOP advantage. Listings 3 and 4 contrast evaluating the pdf, mean, and variance of a Binomial distribution in **distr6** and 'base' package **stats** [245] respectively. The OOP approach of **distr6** involves constructing an object of a given class and then calling the respective method; the functional approach of **stats** requires first defining the function and then passing variables into this.

Listing 3 Evaluating the pdf, mean, and variance of a Binom(5, 0.42) distribution in **distr6**.

```
> size = 5; prob = 0.42;
> x <- Binomial$new(size = size, prob = prob)
> x$pdf(1:2)
> x$mean()
> x$variance()
```

Listing 4 Evaluating the pdf, mean, and variance of a Binom(5, 0.42) distribution in **stats**.

```
> size = 5; prob = 0.42;
> b_mean <- function(prob, size) prob * size
> b_var <- function(prob, size) size * prob * (1-prob)
> dbinom(1:2, prob = prob, size = size)
> b_mean(prob, size)
> b_var(prob, size)
```

Even if pre-defined functions existed for the mean and variance of the Binomial distribution in **stats**, these would still require unique names for all these functions, instead of a single method name and unified approach taken in a object-oriented setting.

Why R6? Once OOP has been established as the way forward with these packages, there is a secondary question about why R and R6 are chosen. The first of these questions is simply due to personal preference and availability of statistical software in R. To answer the second question, R ships with three object-oriented paradigms: S3, S4, and R6 and a full technical comparison of these is provided in section 6.3.5.1. For now this can be summarised by stating that R6 is the first class-object-oriented-programming paradigm (section 6.1.3.1) in R and has many advantages over S3 and S4 including mutable classes, storage efficiency, and a clear OOP structure.

6.1.3. Package Ecosystem

This subsection provides more technical detail about the two 'ecosystems' that are required for the packages discussed in this chapter. Section 6.1.3.1 discusses the

xoop family of packages, which encompasses **R62S3**, **set6**, and **distr6**, and section 6.1.3.2 introduces the **mlr3verse**, in which **mlr3proba** lives. Section 6.1.3.3 demonstrates the dependencies between these package 'universes'.

6.1.3.1. FOOP, COOP, and Now xoop

A relatively new OOP taxonomy has been introduced as a result of functional languages, such as R and Julia, increasing their OO capability. This taxonomy separates Functional Object-Oriented Programming (FOOP) and Class Object-Oriented Programming (COOP) [41]. COOP is also sometimes referred to as encapsulated OOP. This is discussed in more detail as part of the **distr6** paper in section 6.3.5.1, in summary:

i) FOOP is a modern adaptation to COOP that is utilised by functional programming languages. COOP is the more classical setting that is found in OO languages such as Java and Python.

ii) The S3 and S4 paradigms are implicitly using FOOP whereas R6 uses COOP.

iii) R6/COOP is preferred for large-scale interfaces that are required to model complex objects, such as probability distributions or machine learning algorithms.

The xoop Universe Just as programmers will have their preference in programming languages, there will also be those with a preference for different OOP paradigms. R6 is relatively new in R and utilises a novel interface for calling methods and fields. Whilst in its relative infancy, support for R6 in the R community is relatively small. Therefore the **xoop** universe of packages was created to make R6 more accessible and to help further OOP in R. The name **xoop** was chosen to represent a cross between FOOP and COOP. The packages in the 'universe' can be split into: i) software engineering packages that contribute to R6; and ii) toolboxes for handling mathematical objects in R6. **R62S3** is an example of one of the packages that contributes to R6 by enabling coders familiar with FOOP to interface the paradigm without a steep learning curve. Listing 5 demonstrates the use of **R62S3** to convert an R6 class method into an S3 dispatch function. If a package automatically utilises **R62S3** on loading, then from a user perspective only lines 4 and 6 are required.

R62S3 is not discussed in further detail in this paper nor are more abstract contributions to R6. These contributions include implementation of OOP concepts such as abstract classes and decorators, as well as R6-specific helper functions. The next two sections will discuss the **xoop** packages **set6** and **distr6**, but first the '**mlr3verse**' is introduced.

Listing 5 Demonstrating the use of **R62S3** for S3 dispatch on R6 objects. First an R6 class, `printMachine`, is created and then **R62S3** is utilised to create S3 dispatch methods. An object is then constructed, the method `print` is called in the 'usual' R6 way and then with the newly created S3 dispatch method.

```
> printMachine <- R6::R6Class("printMachine",
+  public = list(printer = function(...) cat(...)))
> R62S3(printMachine, assignEnvir = .GlobalEnv)
> x <- printMachine$new()
> x$printer("Hello World!")
> printer(x, "Hello World!")
```

6.1.3.2. The mlr3verse

mlr3proba depends on all the packages in **xoop** but lives in the **mlr3** family of packages, which is termed the '**mlr3verse**'. **mlr3** [182] is the official upgrade to **mlr** [22], written by the same developers. It makes use of R6 and thus COOP programming principles. Whereas **mlr** contained all functionality within one package, **mlr3** modularises the code into small, distinct packages, with clear individual use-cases. Table 11 gives a short description of the packages in the **mlr3verse** that **mlr3proba** depends on.

6.1.3.3. Dependencies and Relationships

Figure 34 visualises the dependency structure of all the packages described above. The structure of the **xoop** family is relatively simple and the **mlr3** family is more complex with each package depending on each other. **mlr3proba** links the two families together with a one-way relationship, so that **mlr3proba** depends on **xoop** but this does not hold in reverse. The next sections will attempt to make a strong case for why the **xoop** packages can be a strong dependency for not just the **mlr3verse** but any R6 interface.

6.2. set6: An Object-Oriented Mathematical Sets Interface in R

6.2.1. Introduction

[1]**set6** makes use of the R6 object-oriented paradigm to introduce classes for important mathematical objects, including sets, tuples, and intervals (finite and infinite). Until now, the R programming language has traditionally supported mathematical sets in one of two ways: 1. via the five set operation functions: `union`, `intersect`, `setdiff`, `setequal`, `is.element`; and 2. via the **sets** [216] package; a full comparison to these solutions is given in section 6.2.2. **set6** uses R6

[1]Parts of this section have been published as part of a paper in the *Journal of Open Source Software* [278].

Table 11: mlr3proba dependencies in the **mlr3verse**. Descriptions copied from package descriptions.

Package	Description	Use in mlr3proba
mlr3 [182]	Implements core ML features including tasks, measures, and resampling methods.	All core features.
mlr3pipelines [21]	Dataflow programming toolbox that implements pipelining operators such as data preprocessing and ensembling.	Composition and reduction pipelines.
mlr3tuning [180]	Implements methods for hyperparameter tuning including grid search and random search with various termination criteria.	Model tuning.
paradox [181]	Defines parameter spaces and constraints to program on such spaces.	Parameter sets for models, tuning and pipelines.
mlr3misc [179]	Helper functions for use in the **mlr3verse**.	Time and computational efficiency.
mlr3learners [183] **mlr3extralearners** [280]	Additional learners are stored in these packages.	Constructing survival, density, and probabilistic regression learners.

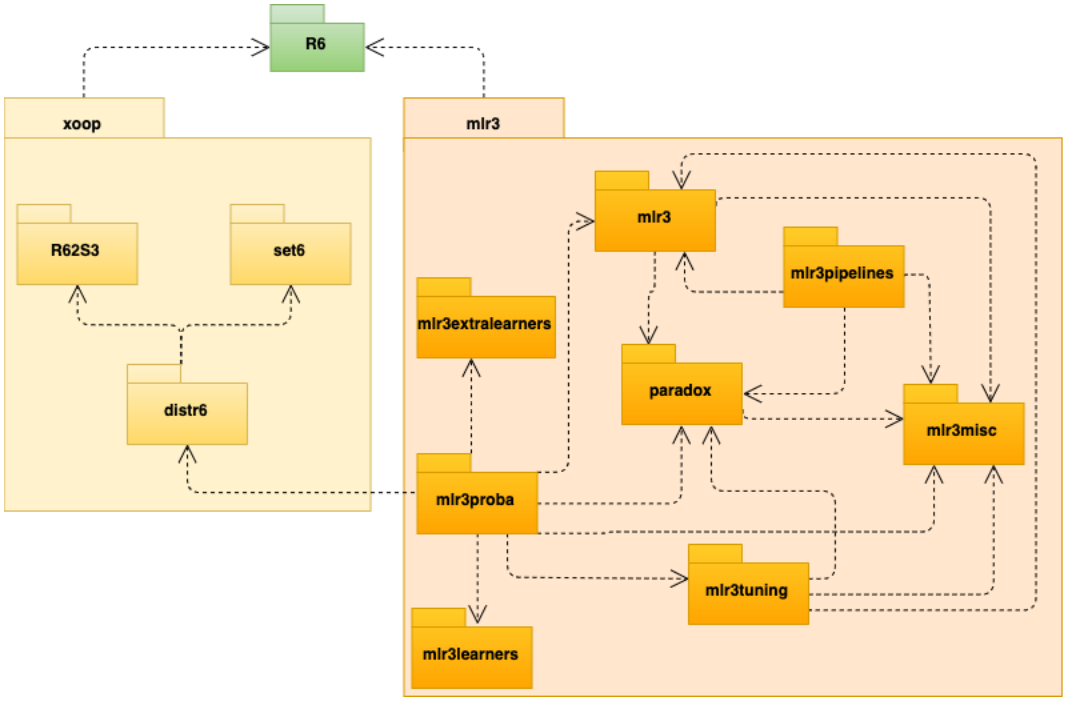


Figure 34: Dependency structure of the **xoop** and **mlr3** family of packages.

and has a clear class interface with minimal dependencies, which makes it a robust and flexible dependency for any package that requires mathematical data types as R6 objects. Making use of design patterns [91], such as wrappers and compositors, **set6** allows for symbolic representation of sets to ensure maximum efficiency, and to provide neat and clear print methods. The package utilises **Rcpp** [74] for improved speed, R6 for efficiency and scalability, and design patterns for a clean and intuitive interface. An emphasis on symbolic representation and lazy evaluation allows the package to handle infinite and very large sets with minimal computational overhead.

The example in listing 6 demonstrates construction of a set and interval, comparisons of these, the set complement operator, and printing of the final result.

Listing 6 Example code for constructing, printing, and subtracting sets.

```
> a <- Set$new(1, 2, 3)
> a$print()
{1, 2, 3}
> a$contains(list(1, "a"))
[1] TRUE FALSE
> b <- Interval$new(1, Inf, class = "numeric")
> b$isSubset(a)
[1] TRUE
> c <- b - a
> c$print()
(1,2) ∪ (2,3) ∪ (3,+∞]
```

Overview to set6 Features A detailed overview of the **set6** API is given in section 6.2.5, below is a list of the most important features.

- Construction of finite mathematical sets

```
> Set$new("a", 2, 3i)
{a, 2, 3i}
```

- Construction of finite and infinite intervals

```
> Interval$new(1, 10, class = "integer")
{1,...,10}
> Interval$new()
[−∞, +∞]
```

- Construction of multi-dimensional sets

```
> Set$new(Tuple$new(1,1), Tuple$new(1,2), Tuple$new(2,1))
{(1, 1), (1, 2), (2, 1)}
```

- Explicit set operations

```
> Set$new(1,2,3) - Set$new(3,4,5)
{1, 2}
> Set$new("a") + Set$new("b")
{a, b}
```

- Symbolic set operations and lazy evaluation

```
> Interval$new(1,5) * Interval$new(1,10)
[1,5] × [1,10]
> powerset(Interval$new())
℘([−∞, +∞])
```

- Set comparisons

```
# subsets
> Set$new(1,2) < Set$new(1,2,3)
# supersets
> Set$new(1,2) > Set$new(1)
# equality
> Set$new(1,2,3) == Set$new(1,2,3)
> Set$new(1,2,3) != Set$new(1,2)
```

- Membership/containedness checks

```
> Set$new(1,2,3)$contains(2)
[1] TRUE
> Set$new(1) %inset% Set$new(2,3,Set$new(1))
[1] TRUE
```

6.2.2. Comparison to Other Software

Only **sets** [216] provides a full sets interface in R. Base R (**base**) [245] supports a few set operations, but does not contain any concrete set objects. Below are comparisons between **set6**, **sets**, and **base**, with respect to representation, speed, and efficiency.

Representation of Sets **base** does not contain an object for sets but uses `lists` and set operation functions. For example to combine two lists in R via a set union,

```
> x = list("Apples")
> y = 1:2
> union(x, y)
[[1]]
[1] "Apples"
[[2]]
```

```
[1] 1
[[3]]
[1] 2
```

Elements are printed individually, which is problematic for large lists. Lists are finite only and vectors in R can only hold around 10^8 elements before crashing, which makes them unsuitable for infinite or very large intervals.

sets supports sets as objects and has set operations that can be called on these, however as the example below demonstrates, there is no way to identify which intervals are contained in the product.

```
> set_cartesian(reals(), reals(), reals())
> print(z)
{(<<interval>>, <<interval>>, <<interval>>)}
```

Sets in **set6** are symbolically represented and use lazy-evaluation to prevent the system crashing.

```
> Reals$new() * Reals$new()
ℝ^2
> Interval$new(1,100) - Set$new(1,10)
(1,10) ∪ (10,100]
```

Speed and Performance The three packages are now compared with respect to speed. Three experiments are conducted, firstly to compare how quickly each can construct a set, secondly to compare how quickly each can perform set union and thirdly to compare set complement. The full results are given in table 12. As expected **base** is much quicker than the other two packages. **set6** outperforms **sets** in construction but is currently slower in set operations, this is due to known bottlenecks that are currently being resolved.

Table 12: Benchmark experiment comparing **set6**, **sets**, and **base** speed.

Package	Operation	Mean (s)	cld[1]
base	Construction	3.3e-5	a
sets	Construction	3.6e-3	c
set6	Construction	1.5e-3	b
base	Complement	5.3e-3	a
sets	Complement	5.8e-2	b
set6	Complement	1.3e-1	c
base	Union	1.5e-4	a
sets	Union	2.8e-2	b
set6	Union	2.3e-1	c

1. Significance test for run-time of each method where 'a' is fastest and 'c' is slowest.
The experiment is conducted on R version 3.6.1; Platform: x86_64-apple-darwin15.6.0 (64-bit); Running under: macOS Mojave 10.14.4 with **microbenchmark** v1.4-7 [215], **sets** v1.0.18, and **set6** v0.1.1.

Table 13 gives a high-level overview to advantages and disadvantages of each of the packages. In summary **base** is the most efficient for performing simple set operations on lists, however lacks functionality for sets as classes, infinite sets, or anything more complex than a list to hold elements. **sets** uses S3 to define classes and has support for infinite sets, as well as some symbolic representation; however it lacks a concise class-infrastructure and is poorly documented. **set6** makes use of the relatively new R6 paradigm, which enables a precise object-oriented interface. It includes finite and infinite sets, as well as clear user-options for symbolic representation of sets; however **set6** may be overly-complicated as a dependency if only list manipulation is required.

Table 13: Comparing advantages and disadvantages of set operations in **base**, **sets**, and **set6**.

Package	Advantages	Disadvantages
base	Very fast operations on finite sets. Very efficient for `lists`.	Union, complement, and intersection only. No set object. No symbolic representation. No support for infinite (or very large) sets.
sets	Classes for sets, tuples, and intervals. Some symbolic representation. Support for infinite sets.	Slow object construction. Inconsistent symbolic representation. Several bugs in functionality. Limited support for composing classes.
set6	R6 classes for sets, tuples and intervals. Flexibility in representation and evaluation. Supports infinite sets.	Slower than **base** for atomic classes. Operations relatively slow.

6.2.3. Use-Cases and Requirements

6.2.3.1. Use-Cases

set6 allows users to clearly navigate the following key use-cases:

U1) **Constructing and querying mathematical sets** Many mathematical Set-like objects can be constructed including sets, tuples, intervals, and fuzzy variants. Sets and tuples can contain objects of any R type (atomic or otherwise).

```
> Set$new("a", 2, TRUE)
{a, 2, TRUE}
> Set$new(Set$new(), Set$new(1), Set$new(1,2))
{{1}, {1, 2}}
```

U2) **Containedness checks** Public methods allow all objects inheriting from `Set` to check if elements are contained within them. This provides a powerful mechanism for use with parameter or distribution supports for other packages as it can be viewed as a 'type check', i.e. checks if a value fits within a specified mathematical type. A C++ implementation of these checks in **Rcpp** [74] means that the computations are incredibly quick for sets and intervals of any size.

```
> Set$new(1, 2, 3)$contains(c(1, 5))
[1]  TRUE FALSE
# Tests if all elements in the list are in the special set of
# the Reals.
>  Reals$new()$contains(list(1, "a", 2.5), all = TRUE)
[1] FALSE
```

U3) **Representation of infinite sets** Symbolic representation and lazy evaluation allows infinite (or very large) sets and intervals to be constructed. This also allows operations such as powerset to be used without crashing the system.

```
> Interval$new()
[−∞, +∞]
> Reals$new()
ℝ
```

U4) **Comparison of, possibly infinite, sets** Two `Set` objects can be compared to check if they are equal or (proper) sub/supersets. Infix operators allow quick and neat comparison.

```
> x <- Set$new(1, 2, 3)
# subset
> x <=  Integers$new()
[1] TRUE
# equality
> x == x
[1] TRUE
# proper superset
> x > x
[1] FALSE
```

U5) **Creation of composite sets from simpler classes** Common set operations, such as unions and complements are implemented, as well as products and exponents. These make use of S3 dispatch to allow quick calculation of composite sets. Lazy evaluation with symbolic representation allow for composite sets to be created, inspected, and printed, without ever needing to be evaluated themselves.

```
> Set$new(Tuple$new(1,1), Tuple$new(1,2))
{(1, 1), (1, 2)}
> Set$new(1,2) * Set$new(3,4)
{1, 2} × {3, 4}
> Reals$new() - Set$new(6)
(−∞, 6) ∪ (6, +∞)
```

6.2.3.2. Requirements

set6 fulfils the following requirements:

R1) Minimal number of dependencies required in order to serve as a base dependency for any OOP R interface.

R2) User control over symbolic representation, which allows neat printing and a choice in lazy evaluation.

```
> setunion(Set$new(1,2), Set$new(3,4), simplify = TRUE)
{1, 2, 3, 4}
> setunion(Set$new(1,2), Set$new(3,4), simplify = FALSE)
{1, 2} ∪ {3, 4}
```

R3) Wrappers for composite sets for symbolic representation.

```
> p <- Set$new(1,2) * Set$new(3,4)
{1, 2} × {3, 4}
> class(p)
[1] "ProductSet" "SetWrapper" "Set" "R6"
```

R4) Minimal number of classes with a clear purpose for each one.

```
# sets for unique objects
> Set$new(1, 2, "a", 2)
{1, 2, a}
# Tuples when ordering matters and duplicates allowed
> Tuple$new(1, 1)
(1, 1)
# Intervals for infinite sets
> Interval$new(1, Inf)
[1, +∞]
# ConditionalSets for complex set-builder defined sets
> ConditionalSet$new(function(x, y) x + y == 2 & x == 0)
{x + y == 2 & x == 0 : x ∈ ℝ, y ∈ ℝ}
```

6.2.4. Design Principles

The design principles of **set6** are directly built upon the use-cases and requirements identified in section 6.2.3, these are:

D1) Maximum user-control over set operations, including choice of associativity, lazy evaluation, and unicode printing. It is vital that users are allowed full control over how complicated sets are built out of simpler ones. Where possible, users can decide:

   (a) Whether associativity and commutativity properties of sets should be respected or not (section 6.2.5.1).

   (b) If set operations should be evaluated immediately or if a wrapper for lazy-evaluation should be used.

   (c) If unicode should be used for printing.

D2) Minimal dependencies to allow sets to be a good base class in any R6 package. This is given in (R1) and re-iterated here for emphasis. A dependency in **set6** should only be added with a very good and clear argument. **set6** is intended to serve as the base class to any object-oriented interface, therefore it is imperative that it cannot run into problems due to an over-reliance on other dependencies. Currently it only depends on, **utils** [245], **R6**, **Rcpp** [74], **checkmate** [178]; the first two of which are developed by the R core team and the last two by developers who are close to the **set6** project.

D3) A clear use-case over `lists`. This design principle is important in order to ensure that every **set6** class is created with a purpose. In general, a parsimonious solution is always preferred to a complex one. In this case, it should be clear why a `list` cannot be used in place of a `Set`, `Interval`, or other **set6** object. Section 6.2.5 provides clear reasoning for each implemented object.

D4) Inspectability and reactive user interface. **set6** prioritises symbolic representation and lazy evaluation to allow for the package to be scalable and to fit into any other package. However it is ensured that this does not detract from a clear user interface, i.e. at all times it should be clear what an object contains both externally (how it prints) and internally (inspection methods). **set6** allows sets to be queried in many different ways, including calling the elements in the set (if finite), finding the bounds of the set (if numeric), and listing properties and traits.

```
> I <- Interval$new(10, Inf, type = "[)")
> print(I)
[10, ∞)
> I$contains(9:11)
[1] FALSE TRUE TRUE
```

D5) Combination of lazy and greedy evaluation. By default, 'multiplying' operations such as products and powersets are evaluated lazily, whereas 'adding' operations such as unions and differences are evaluated greedily. These prevent system crashes from trying to evaluate sets of very large cardinality. In all cases, the user can override defaults. Symbolic representation is used with lazy evaluation so that large sets can be printed neatly without the individual elements ever being evaluated.

```
# Lazy evaluation allows very large sets to be queried without
# explicit evaluation
> p <- powerset(powerset(powerset(Set$new(1, 2, 3))))
> print(p)
℘(℘(℘({1, 2, 3})))
> p$properties$cardinality
[1] 1.157921e+77
> p$contains(Set$new(Set$new(Set$new(1), Set$new(1, 2, 3))))
[1] TRUE
```

6.2.5. Overview to Functionality and API

The UML diagram in fig. 35 visualises the key class structure of **set6**; all classes written in R6. The key components are:

- `Set` parent class
- `SetWrapper` for composite sets built from operations
- `FuzzySet` for fuzzy logic
- `Interval` for continuous sets

This section will focus firstly on the `Set` class as this is the 'parent' to all other classes in the package. Then the use-cases of the mid-level classes (rows 2-3 of fig. 35) are compared. Finally implementation of set operations and their respective classes (rows 4-5 of fig. 35) is discussed.

6.2.5.1. Sets

Figure 36 details the public fields and methods in the `Set` class. Sets can be initialized by one of:

```
> Set$new(..., universe = UniversalSet$new(), class = NULL)
> Set$new(elements, universe = UniversalSet$new(), class = NULL)
```

The first constructor allows (theoretically) infinite elements to passed to the first `...` argument, whereas the second requires a `list` of elements to be passed to the `elements` argument. In most use-cases the first constructor will be used, however internally they represent an important difference as use of the second

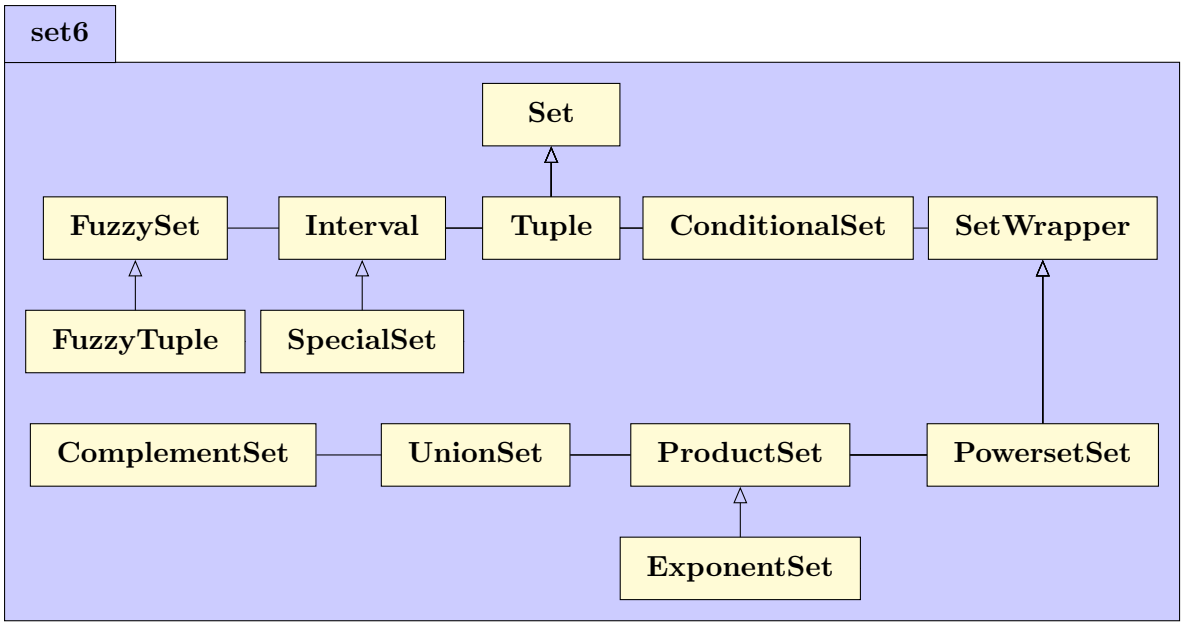


Figure 35: Simplified **set6** class diagram.

bypasses some list-validation checks and therefore slightly improves efficiency for larger sets. The `universe` argument specifies the universe in which the set lives, i.e. the range of possible values that could be added to the set. By default this is the Universal set, which is the set of all possible elements. Finally the `class` argument can create 'typed' sets. A *typed* set is one in which all elements must be of the same type. A 'type' refers to an R class, such as 'numeric', or 'integer', though non-atomic classes are also allowed. For example specifying `class = Set` would coerce every element using `as.Set` on construction (it is assumed a coercion method exists).

Returning to the `Set` class diagram, the public methods are classified into representation methods and membership methods.

Representation Methods The first group of methods refers to the `print` and `summary` methods that all objects in R should include. The `print` method has one argument, `n`, which allows the user to control how many elements in the set to display. The `useUnicode` function controls if unicode should be used when printing sets.

```
> s <- Set$new(1,2,3,4,5,6)
# default, n = 2
> print(s)
{1, 2,...,5, 6}
> s$print(n = 6)
{1, 2, 3, 4, 5, 6}
> i <- Interval$new()
# useUnicode is TRUE by default
> print(i)
```

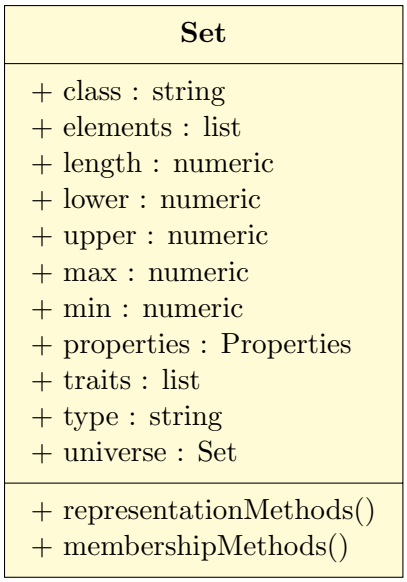


Figure 36: High-level overview of `Set` class.

```
[−∞, +∞]
> useUnicode(FALSE)
> print(i)
[-Inf, Inf]
```

The `summary` method additionally lists the properties and traits of the set.

```
> summary(Set$new(1,2,3,4,5,6))
Set
        {1, 2,...,5, 6}
Traits:
        Crisp
Properties:
        Cardinality = 6  -  countably finite
        Closed
> summary(Reals$new())
Reals
        ℝ
Traits:
        Crisp
Properties:
        Cardinality = Beth1  -  uncountable
        Open
```

Breaking this down there is: i) the name of the class; ii) the symbolic representation of the class; iii) list of class traits; and iv) list of class properties. In object-oriented programming, properties and traits describe objects and classes respectively. In this case, the only trait is whether a class is fuzzy or crisp. The

properties are stored in a `Properties` class, which efficiently calculates all the set properties in construction, these include the set cardinality, its countability, and its closure. The set cardinality and countability are primarily informative, whilst the set closure is used in the various membership methods.

Membership Methods Whilst the representation methods are responsible for making sure that **set6** looks good, the membership methods are responsible for making sure the classes function as expected. This group is comprised by three methods: `contains`, `isSubset`, and `equals`. The design decision was made for these to be public methods, and not S3 functions as is the case with set operations. This is due to the fact that `contains` and `isSubset` are not associative and both require a clear point of reference. Whilst `equals` is associative, it was cleaner to implement this in the same group as the other two. Infix operators simplify the calling of these functions:

```
> Set$new(1,2,3)$equals(Set$new(1,2,3))
> Set$new(1,2,3) == Set$new(1,2,3) # equal to
> Set$new(1,2,3) != Set$new(1,2) # not equal to
> Set$new(1,2,3)$isSubset(Set$new(1,2,3), proper = FALSE)
> Set$new(1,2) < Set$new(1,2,3) # proper subset
> Set$new(1,2,3,4) >= Set$new(1,2,3) # superset
> Set$new(1,2,3)$contains(1)
> 1 %inset% Set$new(1,2,3) # contains
```

For testing set equality, there were three methods considered for implementing `equals`: computational equality, mathematical equality, and membership equality. The latter was chosen in implementation. The difference is illustrated by considering if `Set$new(1,2,3)` and `Tuple$new(1,2,3)` are equal. Computational equality would check to see if all objects are of the same class and have the same methods and fields, this would be `FALSE` in this example as both inherit from a different class. Mathematical equality compares the mathematical object, in this example a tuple contains more information than a set as it also encompasses information about ordering and uniqueness of elements, hence these objects are not mathematically equal. Finally membership equality tests if each object contains the same elements, this is the case here and hence both are equal in **set6**.

Checking if an element is a member of a finite (crisp) set is straightforward to implement, as an element is either in a set, or not. For open intervals, there is the option to decide whether elements on the bounds of the interval should be included. By definition these should not strictly be included, however there are several cases where this may aid in quick construction of important sets. An example of this is given below by making the set of positive Reals act as the set of non-negative Reals, using the `bound` argument.

```
# Positive reals
> PosReals$new()$contains(0, bound = FALSE)
[1] FALSE
```

```
# Non-negative reals
> PosReals$new()$contains(0, bound = TRUE)
[1] TRUE
```

Finally the `isSubset` method determines if one set is a subset of another. In this package, a set `A` is considered a subset of a set `B` if all the elements in `A` are contained in `B`. However, additional properties are required if `B` is a tuple:

```
# This Tuple is a subset of the Set as it is fully contained in the Set.
# Additional properties of the Tuple (e.g. ordering) are ignored.
> Set$new(1,2,3)$isSubset(Tuple$new(2,1))
[1] TRUE
# But, this Set is not a subset of the Tuple as the ordering is different.
> Tuple$new(1,2,3)$isSubset(Set$new(2,1))
FALSE
```

The reference set, i.e. the set on which the method is being called, determines the properties that are required by the method. So a `FuzzySet` requires the subset to be fuzzy, and a `Tuple` requires ordering.

6.2.5.2. Intervals, Tuples, and Conditionals

Now returning to fig. 35 and looking at the second and third rows of the diagram. Fuzzy sets are not discussed here as they are less common, and were added to the package for completeness. Instead a more detailed look is taken at intervals, tuples, and 'conditional sets'. A formal introduction to each with clear use-cases for each class is given below, in keeping with (D3) and (R4).

Tuples Tuples are similar to sets except that they are ordered, always finite, and allow duplicated elements. Hence the `Tuple` class is almost identical to its parent class (`Set`) and only the `equals` and `isSubset` methods are overloaded. In printing, parentheses are used to distinguish a tuple from a set.

```
> Tuple$new(1:5)
(1, 2,...,4, 5)
# tuples preserve ordering
> Tuple$new(1,2,3) == Tuple$new(2,3,1)
[1] FALSE
# sets ignore ordering
> Set$new(1,2,3) == Set$new(2,3,1)
[1] TRUE
# tuples preserve duplicated elements
> Tuple$new(1,2,2,3) == Tuple$new(1,2,3)
[1] FALSE
# sets remove duplicates
> Set$new(1,2,2,3) == Set$new(1,2,3)
[1] TRUE
```

There is an argument that the `Tuple` class is very similar to `lists` in base R. There are two main reasons for including a separate class for tuples, and not relying on lists. Firstly, to inherit from the `Set` parent-class, thereby copying all methods and fields. Secondly, so they can be incorporated neatly into other **set6** objects including other `Sets` and wrappers. The code below gives an example of a common use-case: multi-dimensional sets.

```
> s <- Set$new(Tuple$new(1,2), Tuple$new(2,1), Tuple$new(1,1),
+  Tuple$new(1,3))
> print(s)
{(1, 2), (2, 1), (1, 1), (1, 3)}
> s$contains(list(Tuple$new(1,1), Tuple$new(3,1)))
[1] TRUE FALSE
```

Intervals Intervals have been discussed a few times earlier and the concept is likely familiar to most. In general, an interval is a set on a given number line between two bounds and without breaks. A 'number line' usually refers to the set of Reals, but could be any mathematical set. Intervals are initialized in **set6** with the constructor

```
> Interval$new(lower = -Inf, upper = Inf,
+  type = c("[]","(]","[)","()"), class = "numeric",
+  universe = Reals$new())
```

The first two arguments, `lower` and `upper`, specify the bounds of the interval, i.e. the range of values that are included. The third argument, `type`, determines the interval closure. The *closure* of an interval defines the boundary types. The options correspond to closed, left-open, right-open, and open respectively, the default is 'closed'. The difference can be demonstrated with the `contains` method.

```
> Interval$new(lower = 1, upper = 10, type = "[]")$contains(c(1,10))
[1] TRUE TRUE
> Interval$new(lower = 1, upper = 10, type = "(]")$contains(c(1,10))
[1] FALSE TRUE
> Interval$new(lower = 1, upper = 10, type = "[)")$contains(c(1,10))
[1] TRUE FALSE
> Interval$new(lower = 1, upper = 10, type = "()")$contains(c(1,10))
[1] FALSE FALSE
```

The `class` argument specifies if the interval should be constructed on the Reals, `class = "numeric"`, or Integers, `class = "integer"`. Intervals are constructed by default on the Reals as this appears to be the more common use-case.

```
> Interval$new(class = "numeric")
[−∞, +∞]
> Interval$new(class = "integer")
{−∞, ..., +∞}
```

Intervals on the Reals are denoted by brackets indicating the closure of the interval and the interval bounds, whereas intervals on the Integers are denoted by the bounds, separated by an ellipsis, between curly brackets. Unfortunately this results in a clash of notation between open intervals and tuples, which is a known problem in the literature. **set6** resolves this by using spacing when printing sets and tuples, but no spacing in intervals; this is not an ideal solution but it stays true to common notation whilst distinguishing the two objects.

```
> Tuple$new(1, 2)
(1, 2)
> Interval$new(1, 2, type = "()")
(1,2)
```

Intervals include an extra public method for determining if one interval is a 'subinterval' of another. The difference between a subset and a subinterval can be demonstrated by example

```
> Reals$new()$isSubset(Integers$new())
[1] TRUE
> Reals$new()$isSubinterval(Integers$new())
[1] FALSE
```

The first method demonstrates that the set of Integers is a subset of the set of Reals. This is intuitive as all elements in the Integers can also be found in the Reals. However, the second shows that the Integers are not a subinterval of the Reals. Formally an interval $\mathcal{J} = [c, d]$ (or $\mathcal{J} = \{c, ..., d\}$) is a subinterval of interval $\mathcal{I} = [a, b]$ (or $\mathcal{I} = \{a, ..., b\}$) if $a \leq c \cap b \geq d$. $\mathcal{J}$ is a proper subinterval if the inequality is strict. If $\mathcal{I}$ is a subinterval of $\mathcal{J}$ then it must also be a subset of $\mathcal{J}$, but the reverse does not generally hold. Returning to the above example, the Integers are not a subinterval of the Reals as whilst the boundary condition is satisfied, the Integers is a set of discrete numbers whilst the Reals are continuous. Similarly the set $\{1, 3\}$ is a subset of the Integers but not a subinterval as '2' is missing. $\{1, 2, 3\}$ is both a subset and subinterval of the Integers.

The clearest use-case for intervals in R is to symbolically represent and query infinite sets without causing a system error. For example any interval defined by `Interval$new` is a relatively small object in R that contains very basic information including the bounds and closure type of the interval. The methods of the object allow querying to determine which elements live in the interval. This is vastly more efficient than trying to create a very long (possibly-infinite) vector and then using the `%in%` method. Another advantage of **set6** is the quick construction of 'special' mathematical sets, which are of class `SpecialSet` inheriting from `Interval`. These are essentially intervals but with efficient construction. These special constructors exist for the set of Naturals, Integers, and Reals, as well as the positive and negative variants of each (where sensible). Where appropriate these have an argument to determine if zero should be included in the interval. For example: the set of positive Reals is defined by `PosReals$new()`, the set of negative Reals by `NegReals$new()`, the non-negative Reals by `PosReals$new(zero = TRUE)`, and analogously for the non-positive Reals.

Conditional Sets Finally, the `ConditionalSet` class can be constructed for finite or infinite sets using set-builder notation for one or more logical conditions.

```
> ConditionalSet$new(condition, argclass = NULL)
```

The constructor takes a boolean-valued function, `condition`, that defines the `ConditionalSet`, and an optional list, `argclass`, giving the `Set` that the formal arguments of the `condition` live in.

By combining the two arguments in the constructor, any possible set can be built. For example the simplest `ConditionalSet` is the Universal set, which contains all elements. By default `argclass` is the Universal set. The Universal set is denoted here with '$\mathcal{V}$' to prevent confusion with union (denoted by '$\cup$').

```
> universal <- ConditionalSet$new()
> print(universal)
{x ∈ V}
> universal$contains(list("apple", 5i, FALSE))
[1] TRUE TRUE TRUE
```

An element is a member of a `ConditionalSet` if it: a) satisfies the condition given by the function passed to the constructor; and b) is a member of the given `argclass`. Therefore, the set of Integers can be constructed by specifying the `argclass`.

```
> integers <- ConditionalSet$new(argclass = list(x = Integers$new()))
> integers$print()
{x ∈ ℤ}
> integers$contains(c(1, 1.5))
[1] TRUE FALSE
```

Or the Integers within certain bounds:

```
> integers <- ConditionalSet$new(function(x) x > 2 & x < 4,
+  argclass = list(x = Integers$new()))
> integers$print()
{x ∈ ℤ : x > 2 & x < 4}
> integers$contains(2:4)
[1] FALSE TRUE FALSE
```

These are overly-simple examples to demonstrate that this class can construct any set but the real power of this class lies in the R6 reference semantics. This is best demonstrated by example

```
> yellow_set <- Set$new("sun", "lemons", "yellowpages")
> fruit_set <- Set$new("tomato", "apple", "lemons", "pineapple")
> yellow_fruit <- ConditionalSet$new(function(x)
```

```
+  yellow_set$contains(x) & fruit_set$contains(x))
> yellow_fruit$contains("pineapple")
[1] FALSE
> yellow_set$add("pineapple")
> yellow_fruit$contains("pineapple")
[1] TRUE
```

This example demonstrates how R6 objects reference other R6 objects, and do not copy them. This means that the `contains` method of a `ConditionalSet` retrieves the internal elements in their *current* state and not their state when the conditional set was created. This powerful feature means that complex assertions and checks can be utilised with set-builder notation for any number of R6 (or other paradigm) objects. Cloning can be specified if reference semantics are not required in conditional sets.[1]

6.2.5.3. Operations and Wrappers

The final part of the API discussed here are the set operators and their corresponding wrappers. Table 14 gives the seven set operations, their infix operator, corresponding wrapper, and a brief description of their function. A *wrapper* is a class that contains (or wraps) another one. In **set6**, each wrapper corresponds to an operation in order to provide a symbolic representation of the set. Every wrapper inherits from the `SetWrapper` class, which in turn inherits from `Set` (fig. 35). The `SetWrapper` class adds the additional field `wrappedSets`, which allows internally wrapped sets to be accessed. The operation/wrapper interface is exemplified below.

```
> P = setproduct(Set$new(1,2), Set$new(3,4))
> class(P)
[1] "ProductSet" "SetWrapper" "Set" "R6"
> print(P)
{1, 2} × {3, 4}
> P$wrappedSets
[[1]]
{1, 2}
[[2]]
{3, 4}
```

This example demonstrates how: a) two constructed sets are combined via a set operation; b) a wrapper is created to provide a symbolic representation of the Cartesian product; and c) the original sets are accessible via `wrappedSets`.

The power of the wrappers lies in their symbolic representation of sets, which has the aesthetic benefit of pretty printing, and the important computational benefit of preventing the system crashing via lazy evaluation. For example, powersets can 'blow-up' very quickly even with small sets. In the example below, the

[1] For more details see (e.g.) Hadley's R6 tutorials https://adv-r.hadley.nz/r6.html#r6-semantics.

Table 14: Set operations and corresponding wrappers in **set6**.

Operation	Infix	Wrapper	Description
`powerset`	None	`PowersetSet`	Powerset of a given set.
`setunion`	+ or |	`UnionSet`	Union of two or more sets.
`setcomplement`	-	`ComplementSet`	Relative or absolute complement.
`setsymdiff`	%-%	-	Symmetric difference of two sets.
`setproduct`	*	`ProductSet`	(n-ary) Cartesian product of two or more sets.
`setpower`	^	`ExponentSet`	(n-ary) Cartesian product of a set with itself.
`setintersect`	&	-	Intersection of two sets.

`simplify` argument tells the `powerset` function to return the result as a `Set` and not a `SetWrapper`, overriding the default.

```
# Crashes - Do not try!
> powerset(powerset(powerset(Set$new(1,2,3),
+  simplify = TRUE), TRUE), TRUE)

# Works very well
> ps <- powerset(powerset(powerset(Set$new(1,2,3))))
> print(ps)
℘(℘(℘({1, 2, 3})))
> ps$properties$cardinality
[1] 1.157921e+77
```

To minimise the risk of crashing, the `simplify` argument is set to `FALSE` by default for all 'product' methods (`setproduct`, `powerset`, `setpower`), which can result in sets of large cardinality. The default for 'adding' methods (`setunion`, `setcomplement`, `setsymdiff`, `setintersect`) is `TRUE` as a simpler object is always preferred where possible.

The connection between symbolic representation and lazy evaluation is demonstrated in the code below.

```
# A single Set is created by explicit calculation of the Set
# elements at run-time
> a <- setunion(Set$new(1), Set$new(2), simplify = TRUE)
> class(a)
[1] "Set" "R6"
> a$print()
{1, 2}
# Lazy evaluation and symbolic representation via wrappers
> b <- setunion(Set$new(1), Set$new(2), simplify = FALSE)
```

```
> class(b)
[1] "UnionSet" "SetWrapper" "Set" "R6"
> b$print()
{1} ∪ {2}
# The individual elements are only calculated if called
> b$elements
[1] 1 2
```

Cartesian Product and Associativity There appears to be confusion in implementation surrounding associativity in set operations, particularly with regards to the Cartesian product. An operation, $(\cdot)$, is said to be associative if $(A \cdot B) \cdot C = A \cdot (B \cdot C)$, for any objects A, B, C. In computing, addition and multiplication are both associative, e.g. `(1+2)+3 = 1+(2+3)`, but exponentiation is not, e.g. `(2^3)^4 = 4096` but `2^(3^4) = 2.4e+24`. The R language favours left- or right- associativity, depending on the operator. Right-associativity is used for exponents, i.e. `x^y^z = x^(y^z)`, but for most other operators, left-associativity is the default, e.g. `x+y+z = (x+y)+z`.

The Cartesian product of two sets is not associative, despite often being coded this way. A commonly found example of this is in the representation of coordinate (or matrix) spaces, e.g. the three-dimensional Reals $\mathbb{R}^3 = \{(1,1,1), (1,1,2), (1,2,1),...,(2,1,1),(2,1,2),(2,2,2),...\}$. This is the *n-ary* Cartesian product, which is often implied by (but not a property of) the exponent of a set. This difference is often overlooked, despite being two different operations. This is highlighted by looking at the definitions for three sets, $\mathcal{A}, \mathcal{B}, \mathcal{C}$.

The Cartesian product is a binary operator and is therefore defined in two parts:

$$\begin{aligned}\mathcal{A} \times \mathcal{B} \times \mathcal{C} &= (\mathcal{A} \times \mathcal{B}) \times \mathcal{C} \\ &= \{(a,b) : a \in \mathcal{A}, b \in \mathcal{B}\} \times \mathcal{C} \\ &= \{((a,b),c) : a \in \mathcal{A}, b \in \mathcal{B}, c \in \mathcal{C}\}\end{aligned}$$

where left-associativity is assumed. It can be seen that the Cartesian product is not associative as $\mathcal{A} \times (\mathcal{B} \times \mathcal{C}) = \{(a,(b,c)) : a \in \mathcal{A}, b \in \mathcal{B}, c \in \mathcal{C}\}$. In contrast the n-ary Cartesian product is associative as

$$\begin{aligned}\mathcal{A} \times \mathcal{B} \times \mathcal{C} = \mathcal{A} \times (\mathcal{B} \times \mathcal{C}) &= (\mathcal{A} \times \mathcal{B}) \times \mathcal{C} \\ &= \{(a,b,c) : a \in \mathcal{A}, b \in \mathcal{B}, c \in \mathcal{C}\}\end{aligned}$$

In order to maximise user-flexibility but minimise the number of functions, the difference highlighted above is controlled with the `nest` argument in `setproduct` and `setpower`. It is called `nest` as the (non-n-ary) Cartesian product can be thought of as nesting sets within each other. The default is `nest = FALSE` as this appears to be the more common use-case in practice. A few examples are given

below to conclude this section.

```
# nest = FALSE - Associative n-ary Cartesian product
> setproduct(Set$new(1,2), Set$new(1,3), Set$new(1,4),
+  simplify = TRUE, nest = FALSE)
{(1, 1, 1), (2, 1, 1),...,(1, 3, 4), (2, 3, 4)}
# nest = TRUE - Non-associative Cartesian product
> setproduct(Set$new(1,2), Set$new(1,3), Set$new(1,4),
+  simplify = TRUE, nest = TRUE)
{((1, 1), 1), ((2, 1), 1),...,((1, 3), 4), ((2, 3), 4)}
# nest = FALSE - Associative n-ary Exponentiation
> setpower(Set$new(1,2), power = 3, simplify = TRUE, nest = FALSE)
{(1, 1, 1), (2, 1, 1),...,(1, 2, 2), (2, 2, 2)}
# nest = TRUE - Non-associative Exponentation
> setpower(Set$new(1,2), power = 3, simplify = TRUE, nest = TRUE)
{((1, 1), 1), ((2, 1), 1),...,((1, 2), 2), ((2, 2), 2)}
```

6.2.6. Conclusion and Availability

set6 introduces the first R6 object-oriented mathematical sets interface in R, and the first interface that allows full user-control over symbolic representation and evaluation of sets. The goal of **set6** is not to be a large package with a lot of functionality, but instead to be an efficient base that any R6 interface can depend on. Future development plans are primarily focused on maintenance and optimisation with an emphasis on improving construction and operation speeds.

set6 is released under an MIT licence on GitHub and CRAN. Extended documentation, tutorials, and examples are available on the project website[1]. Code quality is monitored and maintained by an extensive suite of unit tests with GitHub Actions on multiple operating systems.

6.3. distr6: An Object-Oriented Probability Distributions Interface in R

6.3.1. Introduction

[2]Probability distributions are an essential part of data science, underpinning models, simulations, and inference. Hence, they are central to computational data science. With the advent of modern machine learning and AI, it has become increasingly common to adopt a conceptual model where distributions are considered objects in their own right, as opposed to primarily represented through distribution defining functions (e.g., cdf, pdf), or random samples. **distr6** is

[1] https://xoopR.github.io/set6/

[2] This section has been accepted for publication in *The R Journal* [277].

an object oriented implementation of these conceptual models, and allows manipulation, combination, and inspection of distributions as objects with defined properties and methods.

To appreciate the object oriented conceptual model for distributions in more detail, it is important to conceptualize and distinguish some mathematical concepts which are similar and thus often conflated:

- A random variable, distributed according to a certain distribution, e.g., $X \sim \mathcal{N}(0, 1)$.
- The cdf of that random variable X, usually denoted by F_X, a function $F_X : \mathbb{R} \to [0, 1]$.
- The pdf of that random variable X, often denoted by f_X, a function $f_X : \mathbb{R} \to [0, \infty)$.
- The distribution, d, according to which X is distributed – often called 'the law of' X. This can be represented by multiple mathematical objects, such as the cdf F_X or the pdf of X. Note that d is not identical to either these representation functions.

A full mathematical definition of the conceptual model is given in the next section.

Critically, random variables and distributions are neither identical objects nor concepts. A random variable X *has* distribution d, and multiple random variables may be distributed according to d. Further, random variables are sampled from, while the distribution is only a description of probabilities for X. Thus, X and d are not identical objects. Figure 37 visually summarizes these differences.

As a possible logical consequence of the above, the conceptual model is adopted where a distribution is an abstract object, which:

i) Has multiple defining representations, for example through cdf and possibly through pdf, but is not identical with any of these representations.

ii) Possesses traits, such as being absolutely continuous over the Reals, and properties, such as skewness and symmetry.

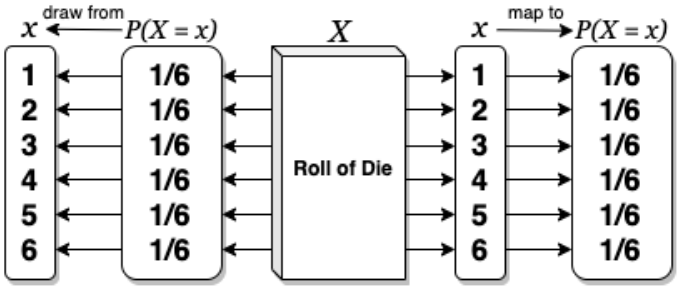
iii) Can be used to define sampling laws of random variables, but is not conceptually identical with a random variable.

Abstracting distributions as objects from multiple, non-identical, representations (random variables), introduces major consequences for the conceptual model:

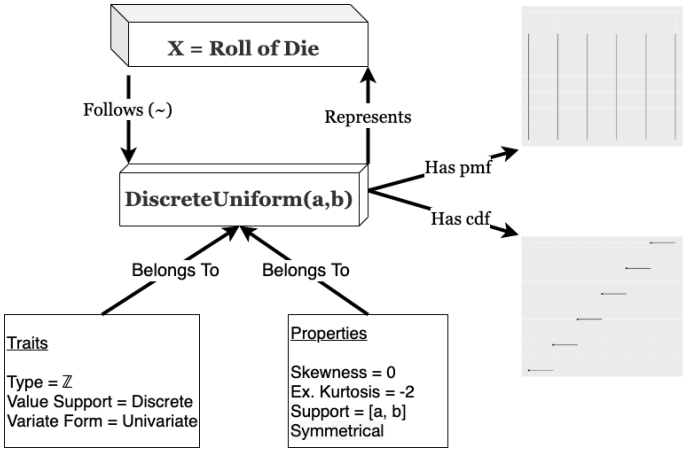
i) It lends itself naturally to a class-object representation, in the computer scientific sense of object oriented programming. Abstract distributions become classes, concrete distributions are objects, and distribution defining functions are methods of these classes. Random variables are a separate type of object.

ii) It strongly suggests adoption of mathematical conceptualization and notation which cleanly separates distributions from random variables and distribution defining functions – in contrast to common convention where random variables or random sampling takes conceptual primacy above all.

iii) It allows clean formulation of algorithmic manipulations involving distributions, especially higher-order constructs (truncation, huberization, etc.), as well as clean mathematical definitions.



(a) Discrete Uniform Random Variable



(b) Discrete Uniform Probability Distribution

Figure 37: (a) A random variable following a Discrete Uniform distribution. (b) A Discrete Uniform distribution representing a random variable.

6.3.1.1. Distributions as Software and Mathematical Objects

In **distr6**, distributions are first-class objects subject to an object oriented class-object representation. For example, a discrete uniform distribution (fig. 37b) is a 'class' with traits such as type (Naturals), and variate form (univariate). With a given parametrisation, this becomes an 'object' with properties including symmetry and support. An alternative definition to the conceptual model of distributions is now provided.

On the mathematical level, distributions are considered as objects in their own right, not being identical with a cdf, pdf, or measure, but instead 'having' these as properties.

For a set $\mathcal{Y}$ (endowed with suitable topology), define Distr($\mathcal{Y}$) as a set containing formal objects d which are in bijection to (but not identical with) probability

measures over $\mathcal{Y}$. Elements of Distr($\mathcal{Y}$) are called distributions over $\mathcal{Y}$. Further define formal symbols which, in case of existence, denote 'aspects' that such elements have, in the following way: the symbol $d.F$, for example, denotes the cdf of d, which is to be read as the 'F' of d, with F in this case to be read as a modifier to a standard symbol d, rather than a fixed, bound, or variable symbol. In this way, define:

i) $d.F$ for the cdf of d. This typically exists if $\mathcal{Y} \subseteq \mathbb{R}^n$ for some n, in which case $d.F$ is a function of type $d.F : \mathbb{R}^n \to [0, 1]$.

ii) $d.f$ for the pdf of d. This exists if $\mathcal{Y} \subseteq \mathbb{R}^n$, and the distribution d is absolutely continuous over $\mathcal{Y}$. In this case, $d.f$ is a function of type $d.f : \mathbb{R}^n \to [0, \infty)$.

iii) $d.P$ for the probability measure that is in bijection with d. This is a function $d.P : \mathcal{F} \to [0, 1]$ where $\mathcal{F}$ is the set of measurable sub-sets of $\mathcal{Y}$.

The above is indeed a full formal mathematical definition of this notion of distribution. While distributions, defined this way, are not identical with any of the conventional mathematical objects that define them (cdf, pdf, measures), they are conceptually, formally, and notationally well-defined. Similarly, the aspects ($d.F$, $d.f$, etc) are also well-defined, since they refer to one of the conventional mathematical objects which are well-specified in dependence of the distribution (in case of existence).

This notation provides a more natural and clearer separation of distribution and random variables and allows us to talk about and denote concepts such as 'the cdf of any random variable following the distribution d' with ease ($d.F$), unlike classical notation that would see one define $X \sim d$ and then write F_X. This notation more clearly follows the software implementation of distributions.

For example, in **distr6**, the code counterpart to defining a distribution d which is Normal with mean 1 and variance 2 is

```
> d <- Normal$new(1, 2)
```

The pdf and cdf of this Normal distribution evaluated at 2 are obtained in code as

```
> d$pdf(2)
> d$cdf(2)
```

which evaluates to 'numerics' that represent the real numbers $d.f(2)$ and $d.F(2)$.

The consideration of distributions as objects, and their separation from random variables as objects, is notably distinct from R **stats**, which implements both distribution and random variable methods by the '`dpqr`' functions. Whilst this may allow very fast generation of probabilities and values, there is no support for querying and inspection of distributions as objects. By instead treating the `dpqr` functions as methods that belong to a distribution object, **distr6** encapsulates all the information in R **stats** as well as distribution properties, traits, and other important mathematical methods. The object orientation principle that defines the architecture of **distr6** is further discussed throughout this manuscript.

Treating distributions as objects is not unique to this package. Possibly the first instance of the object oriented conceptualization is the **distr** [258] family of packages, of which **distr6** is the 'official' upgrade. **distr6** is the first such package to use the full object orientation paradigm R6, with other distribution related packages using S3 or S4. The choice of R6 over S3 and S4 is discussed in detail in section 6.3.5.1. This choice allows **distr6** to fully leverage the conceptual model, and make use of core R6 functionality. As well as introducing fundamental object-oriented programming (OOP) principles such as abstract classes, and tried and tested design patterns [91] including decorators, wrappers, and compositors (see section 6.3.5.3).

Besides an overview to **distr6**'s novel approach to probability distributions in R, this paper also presents a formal comparison of the different OOP paradigms, while detailing the use of relevant design patterns.

6.3.1.2. Motivating Example: Higher-Order Distribution Constructs

The strength of the object oriented approach, both on the algorithmic and mathematical side, lies in its ability to efficiently express higher-order constructs and operations: actions between distributions, resulting in new distributions. One such example is mixture distributions. In the **distr6** software interface, a `MixtureDistribution` is a higher-order distribution depending on two or more other distributions. For example take a uniform mixture of two distributions `distr1` and `distr2`:

```
> my_mixt <- MixtureDistribution$new(list(distr1, distr2))
```

Internally, the dependency of the constructs on the components is remembered so that `my_mixt` is not only evaluable for `cdf` (and other methods), but also carries a symbolic representation of its construction and definition history in terms of `distr1` and `distr2`.

On the mathematical side, the object oriented formalism allows clean definitions of otherwise more obscure concepts, for example the mixture distribution is now defined by:

For distributions $d_1, \ldots, d_m$ over $\mathbb{R}^n$ and weights $w_1, \ldots, w_m$, define the mixture of $d_1, \ldots, d_m$ with weights $w_1, \ldots, w_m$ to be the unique distribution $\tilde{d}$ such that $\tilde{d}.F(x) = \sum_{i=1}^{m} w_i \cdot d_i.F(x)$ for any $x \in \mathbb{R}^n$. Note the added clarity by defining the mixture on the distribution d_i, i.e., a first-order concept in terms of distributions.

6.3.2. Related software

This section provides a review to other related software that implement probability distributions, this is focused on, but not limited to, software in R.

R stats, actuar, and extraDistr The core R programming language consists of packages for basic coding and maths as well as the **stats** package for statistical functions. **stats** contains 17 common probability distributions and four

lesser-known distributions. Each distribution consists of (at most) four functions: `dX, pX, qX, rX` where `X` represents the distribution name. These correspond to the probability density/mass, cumulative distribution, quantile (inverse cumulative distribution) and simulation functions respectively. Each is implemented as a separate function, written in C, with both inputs and outputs as numerics. The strength of these functions lies in their speed and efficiency, there is no quicker way to find, say, the pdf of a Normal distribution than to run the `dnorm` function from **stats**. However, this is the limit of the package in terms of probability distributions. As there is no physical distribution object, there is no way to query results from the distributions outside of the '`dpqr`' functions.

Several R packages implement `dpqr` functions for extra probability distributions. Of particular note are the **extraDistr** [323] and **actuar** [73] packages that add over 60 distributions between them. Both of these packages are limited to `dpqr` functions and therefore have the same limits as R **stats**.

distr **distr** was the first package in R to implement an object-oriented interface for distributions, using the S4 object-oriented paradigm. **distr** tackles the two fundamental problems of **stats** by introducing distributions as objects that can be stored and queried. These objects include important statistical results, for example the expectation, variance and moment generating functions of a distribution. The **distr** family of packages includes a total of five packages for object-oriented distributions in R. **distr** has two major weaknesses caused by using the S4 paradigm, these are related to inheritance and object size and are given full consideration in section 6.3.5.1.

distributions3 **distributions3** [119] defines distributions as objects using the S3 paradigm. However, whilst **distributions3** treats probability distributions as S3 objects, it does not add any properties, traits, or methods and instead uses the objects solely for `dpqr` dispatch. In terms of comparison to **distr**, **distributions3** removes features and 'downgrades' the paradigm from S4 to S3.

mistr **mistr** [259] is another recent distributions package, which is also influenced by **distr**. The sole focus of **mistr** is to add a comprehensive and flexible framework for composite models and mixed distributions. Similarly to **distributions3**, the package uses an S3 framework and also implements distributions as objects, an overlap in the packages.

Distributions.jl Despite not being a package written in R, the Julia **Distributions.jl** [198] package provided inspiration for **distr6**. **Distributions.jl** implements distributions as objects with statistical properties including expectation, variance, moment generating and characteristic functions, and many more. This package uses multiple inheritance for 'valueSupport' (discrete/continuous) and 'variateForm' (univariate/multivariate/matrixvariate). Every distribution inherits from both of these, e.g. a distribution can be 'discrete-univariate', 'continuous-multivariate', 'continuous-matrixvariate', etc. The package provides a unified and user-friendly OOP interface.

6.3.3. Design Principles

distr6 was designed and built around the following principles.

D1) **Unified interface** The package is designed such that all distributions, no matter how complex, have an identical user-facing interface. This helps make the package easy to navigate and the documentation simple to read. Moreover it minimises any confusion resulting from using multiple distributions. A clear inheritance structure also allows wrappers and decorators to have the same methods as distributions, which means even complex composite distributions should be intuitive to use. Whether a user constructs a simple Uniform distribution, or a mixture of 100 Normal distributions, the same methods and fields are seen in both objects.

D2) **Separation of core/exotic and numerical/analytic** Via abstraction and encapsulation, core statistical results (common methods such as mean and variance) are separated from 'exotic' ones (less common methods such as anti-derivatives and p-norms). Similarly, implemented distributions only contain analytic results; users can impute numerical results using decorators. This separation has two benefits: 1) a less-technical user can guarantee precision of results as they are unlikely to use numerical decorators; 2) a user has access to the most important distribution methods immediately after construction but is not overwhelmed by many 'exotic' methods that they may never want to use. Use of decorators and wrappers allow the user to manually expand the interface at any time. For example a user can choose between an undecorated Binomial distribution, with common methods such as mean and variance, or they can decorate the distribution to additionally gain access to survival and hazard functions.

D3) **Inheritance without over-inheritance** The class structure stems from a series of a few abstract classes with concrete child classes, which allows for a sensible, but not over-complicated, inheritance structure. For example all implement distributions inherit from a single parent class (`Distribution`) in order that common methods can be unified and only coded once. By allowing extension of classes by decorators and wrappers, and not solely inheritance, the interface is highly scalable and extensible. All decorators and wrappers in **distr6** stem from abstract classes, which in turn inherit from the `Distribution` super-class. In doing so, any method of expanding an object's interface in **distr6** (i.e. via decorators, wrappers or inheritance) will automatically lead to an interface that inherits from the top-level class, maintaining the principle of a unified interface (D1).

D4) **Inspection and manipulation of multiple parameterisations** The design process identified that use of distributions in R **stats** is inflexible as in the majority of cases, only one parameterisation of each distribution is allowed. This can lead to isolating users who may only be familiar with one parameterisation. For example the use of the *precision* parameter in the Normal distribution is typically more common in Bayesian statistics whereas using the *variance* or *standard deviation* parameters are more common in

frequentist statistics. **distr6** allows the user to choose from multiple parameterisations for all distributions (where more than one parameterisation is possible/known). Furthermore, querying and updating of any parameter in the distribution is allowed, even if it was not specified in construction (section 6.3.4). This allows for a flexible parameter interface that can be fully queried and modified at any time.

D5) **Flexible interfacing for technical and non-technical users** Throughout the design process, it was required that **distr6** be accessible to all R users. This was a challenge as R6 is a very different paradigm from S3 and S4. To reduce the learning curve, the interface is designed to be as user-friendly and flexible as possible. This includes: 1) a 'sensible default principle' such that all distributions have justified default values; 2) an 'inspection principle' with functions to list all distributions, wrappers, and decorators. As discussed in (D2), abstraction and encapsulation allow technical users to expand any distribution's interface to be as arbitrarily complex as they like, whilst maintaining a minimal representation by default. Where possible defaults are 'standard' distributions, i.e. with location 0 and scale 1, otherwise sensible defaults are identified as realistic scenarios, for example `Binomial(n = 10, p = 0.5)`.

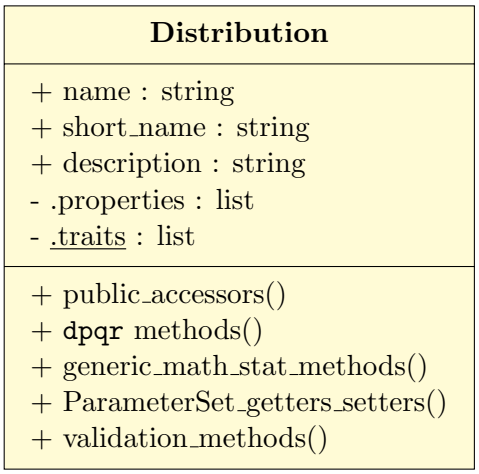
D6) **Flexible OO paradigms** Following from (D5), R6 is still relatively new in R with only 314 out of 16,050 packages depending on it (as of July 2020). Therefore this was acknowledged and taken into account when building the package. R6 is also the first paradigm in R with the dollar-sign notation (though S4 uses '@' notation) and with a proper construction method. Whilst new users are advised to learn the basics of R6, S3 compatibility is available for all common methods via **R62S3**. Users can therefore decide on calling a method via dollar-sign notation or dispatch, the example below demonstrates 'piping' and S3. As the core package is built on R6, the thin-wrappers provided by **R62S3** do not compromise the above design principles.

```
> library(magrittr)
> N <- Normal$new(mean = 2)
> N %>%
+  setParameterValue(mean = 1) %>%
+  getParameterValue("mean")
[1] 1
> pdf(N, 1:4)
[1] 0.398942280 0.241970725 0.053990967 0.004431848
```

6.3.4. Overview to Functionality and API

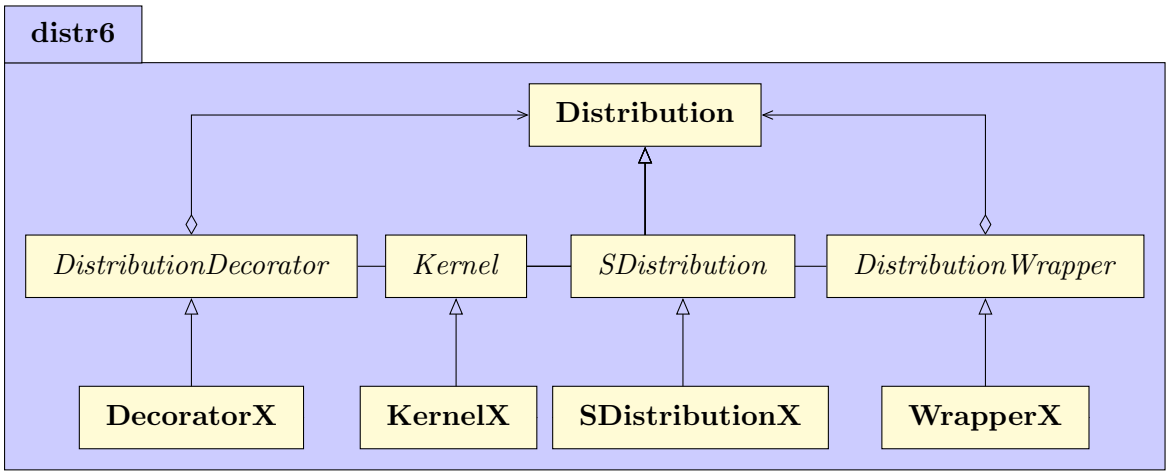
distr6 1.4.3 implements 56 probability distributions, including 11 probability kernels. Individual distributions are modelled via classes that inherit from a common interface, implemented in the `Distribution` parent class. The `Distribution` class specifies the distribution interface for parameter access, properties, traits,

and methods, such as a distribution's pdf or cdf. The most important interface points are described in Section 6.3.4.1



Concrete distributions, kernels, and wrappers are the grandchildren of `Distribution`, and children of one of the mid-layer abstract classes:

- `SDistribution`, which models abstract, generic distributions. Concrete distributions, such as `Normal` which models the normal distribution, inherit from `SDistribution`.
- `Kernel`, which models probability kernels, such as `Triangular` and `Epanechnikov`. Probability kernels are absolutely continuous distributions over the Reals, with assumed mean 0 and variance 1.
- `DistributionWrapper`, which is an abstract parent for higher-order operations on distributions, including compositions, that is, operations that create distributions from other distributions, such as truncation or mixture.
- `DistributionDecorator`, whose purpose is supplementing methods to distributions in the form of a decorator design pattern, this includes methods such as integrated cdf or squared integrals of distribution defining functions.



The UML diagram above visualises the key class structure of **distr6** including the concrete `Distribution` parent class, from which all other classes in the package

inherit from (with the exception of the `ParameterSet`). These abstract classes allow simple extensibility for concrete sub-classes.

6.3.4.1. The Distribution Interface

The base, or top-level, class in **distr6** is the `Distribution` class. Its primary function is to act as a parent class for the implemented probability distributions and higher-order compositions, it is also utilised for creation of custom distributions. By design, any distribution already implemented in **distr6** will have the same interface as a user-specified custom distribution, ensuring (D1) is upheld. A table of the most important methods for a distribution are in table 15 alongside their meaning and definitions (mathematical if possible). The two use-cases for the `Distribution` class are discussed separately.

Table 15: Common methods available to all classes inheriting from `Distribution`. Horizontal lines separate mathematical, property, parameter, and representation methods.

Method	Description/Definition[1]
`pdf/cdf/quantile/rand`	`dpqr` functions.
`mean`	$d.\mu = \mathbb{E}[X]$ *
`variance`	$d.\sigma^2 = \mathbb{E}[(X - d.\mu)^2]$ *
`traits`	List including value support (discrete/continuous/mixed); variate form (uni-/multi-/matrixvariate); type (mathematical domain).
`properties`	List including skewness ($\mathbb{E}[((X - d.\mu)/d.\sigma)^3]$) and symmetry (boolean).
`get/setParameterValue`	Getters and setters for parameter values.
`parameters`	Returns the internal parameterisation set.
`print/summary`	Representation functions, summary includes distribution properties and traits.

1. Mathematical definition (if available) or description of method.
* – $d.\mu$ and $d.\sigma$ are the mean and standard deviation associated with distribution d and $d.\sigma^2 = (d.\sigma)^2$. X is a random variable following distribution d.

Distribution for Inheritance It is anticipated that the majority of **distr6** users will be using the package for the implemented distributions and kernels. With this in mind, the `Distribution` class defines all variables and methods common to all child classes. The most important of these are the common analytical expressions and the `dpqr` public methods. Every concrete implemented distribution/kernel has identical public `dpqr` methods that internally call private `dpqr` methods. This accounts for inconsistencies occurring from packages returning functions in different formats and handling errors differently; a problem most prominent in multivariate distributions. Another example is handling of non-integer values for discrete distributions, in some packages this returns 0, in others the value is rounded down, and in others an error is returned. The `dpqr` functions for all distributions have unified validation checks and return types (`numeric` or

`data.table`). In line with base R and other distribution packages, **distr6** implements a single `pdf` function to cover both probability mass and probability density functions.

```
> Normal$new()$pdf(1:2)
[1] 0.24197072 0.05399097
> Binomial$new()$cdf(1:2, lower.tail = FALSE, log.p = TRUE,
+  simplify = FALSE)
          Binom
1: -0.01080030
2: -0.05623972
```

A key design principle in the package is separation of analytical and numerical results (D2), which is ensured by only including analytical results in implemented distributions. Missing methods in a distribution therefore signify that no closed-form expression for the method is available, however all can be numerically estimated with the `CoreStatistics` decorator (see section 6.3.4.2). Ideally, all distributions will include analytical methods for the following: probability density/mass function (`pdf`), cumulative distribution function (`cdf`), inverse cumulative distribution function/quantile function (`quantile`), simulation function (`rand`), mean, variance, skewness, (excess) kurtosis, and entropy of the distribution (`mean, variance, skewness, kurtosis, entropy`), as well as the moment generating function (`mgf`), characteristic function (`cf`), and probability generating function (`pgf`). Speed is currently a limitation in **distr6** but the use of **Rcpp** [74] in all `dpqr` functions helps mitigate against this.

The fourth design principle of **distr6** ensures that multiple parameterisations of a given distribution can be both provided and inspected at all times. For example the Normal distribution can be parametrised in terms of variance, standard deviation, or precision. Its constructor takes this into account:

```
> Normal$new(mean = 0, var = 1, sd = NULL, prec = NULL,
+  decorators = NULL)
```

To avoid conflicting parameterisations, all distributions have a 'right-to-left' priority when multiple parameterisations are possible. This is best demonstrated by example:

```
# 'prec' has priority
> Normal$new(var = 1, sd = 2, prec = 1/3)$getParameterValue("var")
[1] 3
# 'prec' is not supplied so 'sd' has priority
> Normal$new(var = 1, sd = 2)$getParameterValue("var")
[1] 4
# only 'var' supplied
> Normal$new(var = 1)$getParameterValue("var")
[1] 1
```

The same principle is used for parameter setting with `setParameterValue`. Possible parameterisations and their prioritisation are carefully documented and also can be seen from `parameters`. The example above utilised the `getParameterValue` and `setParameterValue` methods for getting and setting parameter values respectively. The former takes a single argument, the parameter name, and the second a named list of arguments corresponding to the parameter name and the value to set. The example below demonstrates this for a Gamma distribution. Here the distribution is constructed, the shape parameter is queried, both shape and rate parameters are updated and the latter queried, finally the scale parameter is set which auto-updates the rate parameter.

```
> G <- Gamma$new(shape = 1, rate = 1)
> G$getParameterValue("shape")
[1] 1
> G$setParameterValue(shape = 2, rate = 2)
> G$getParameterValue("rate")
[1] 2
> G$setParameterValue(scale = 2)
> G$getParameterValue("rate")
[1] 0.5
```

Distribution and parameter domains and types are represented by mathematical sets, implemented in **set6**. This allows for clear representation of infinite sets and most importantly for internal containedness checks. For example all public `dpqr` methods first call the `contains` method in their respective `type` and return an error if any points are outside the distribution's domain. As **set6** uses **Rcpp** for this method, these come at minimal cost to speed.

```
> B <- Binomial$new()
> B$pdf(-1)
Error in B$pdf(-1) :
  Not all points in {-1} lie in the distribution domain (N0).
```

These domains and types are returned along with other important properties and traits in a call to `properties` and `traits` respectively, this is demonstrated below for the Arcsine distribution.

```
> A <- Arcsine$new()
> A$properties
$support
[0,1]


$symmetry
[1] "symmetric"

```

```
> A$traits
$valueSupport
[1] "continuous"

$variateForm
[1] "univariate"

$type
R
```

Extending distr6 with Custom Distributions Users of **distr6** can create temporary custom distributions using the constructor of the `Distribution` class directly. Permanent extensions, e.g., as part of an R package, should create a new concrete distribution as a child of the `SDistribution` class.

The `Distribution` constructor is given by

```
> Distribution$new(name = NULL, short_name = NULL, type = NULL,
+   support = NULL, symmetric = FALSE, pdf = NULL, cdf = NULL,
+   quantile = NULL, rand = NULL, parameters = NULL,
+   decorators = NULL, valueSupport = NULL, variateForm = NULL,
+   description = NULL)
```

The `name` and `short_name` arguments are identification for the custom distribution used for printing. `type` is a trait corresponding to scientific type (e.g. Reals, Integers,...) and `support` is the property of the distribution support. Distribution parameters are passed as a `ParameterSet` object, this defines each parameter in the distribution including the parameter default value and support. The `pdf/cdf/quantile/rand` arguments define the corresponding methods and are passed to the private `.pdf/.cdf/.quantile/.rand` methods, as above the public methods are already defined and ensure consistency in each function. At a minimum users have to supply the distribution `name`, `type` and either `pdf` or `cdf`, all other information can be numerically estimated with decorators (see section 6.3.4.2).

```
> d <- Distribution$new(name = "Custom Distribution",
+   type = Integers$new(), support = Set$new(1:10),
+   pdf = function(x) rep(1/10, length(x)))
> d$pdf(1:3)
[1] 0.1 0.1 0.1
```

6.3.4.2. DistributionDecorator

Decorators add functionality to classes in object-oriented programming. These are not natively implemented in R6 and this novel implementation is therefore discussed further in section 6.3.5.3. Decorators in **distr6** are only 'allowed' if they have have at least three methods and cover a clear use-case, this prevents

too many decorators bloating the interface. However by their nature, they are lightweight classes that will only increase the methods in a distribution if explicitly requested by a user. Decorators can be applied to a distribution in one of three ways:

In construction:

```
> N <- Normal$new(decorators = c("CoreStatistics",
+  "ExoticStatistics"))
```

Using the `decorate()` function:

```
> N <- Normal$new()
> decorate(N, c("CoreStatistics", "ExoticStatistics"))
```

Using the `decorate` method inherited from the `DistributionDecorator` superclass:

```
> N <- Normal$new()
> ExoticStatistics$new()$decorate(N)
```

The first option is the quickest if decorators are required immediately. The second is the most efficient once a distribution is already constructed. The third is the closest method to true OOP but does not allow adding multiple decorators simultaneously.

Three decorators are currently implemented in **distr6**, these are briefly described.

CoreStatistics This decorator imputes numerical functions for common statistical results that could be considered core to a distribution, e.g. the mean or variance. The decorator additionally adds generalised expectation (`genExp`) and moments (`kthmoment`) functions, which allow numerical results for functions of the form $\mathbb{E}[f(X)]$ and for crude/raw/central K moments. The example below demonstrates how the `decorate` function exposes methods from the `CoreStatistics` decorator to the Normal distribution object.

```
> n <- Normal$new(mean = 2, var = 4)
> n$kthmoment(3, type = "raw")
Error: attempt to apply non-function
> decorate(n, CoreStatistics)
> n$kthmoment(3, type = "raw")
[1] 32
```

ExoticStatistics This decorator adds more 'exotic' methods to distributions, i.e. those that are unlikely to be called by the majority of users. For example this includes methods for the p-norm of survival and cdf functions, as well as anti-derivatives for these functions. Where possible, analytic results are exploited. For example, this decorator can implement the survival function in one of two ways: either as: i) 1 minus the distribution cdf, if an analytic expression for the cdf is available; or ii) via numerical integration of the distribution.

FunctionImputation This decorator imputes numerical expressions for the `dpqr` methods. This is most useful for custom distributions in which only the `pdf` or `cdf` is provided. Numerical imputation is implemented via **Rcpp**.

6.3.4.3. Composite Distributions

Composite distributions – that is, distributions created from other distributions – are common in advanced usage. Examples for composites are truncation, mixture, or transformation of domain. In **distr6**, a number of such composites are supported. Implementation-wise, this uses the wrapper OOP pattern, which is not native to R6 but part of the extensions to R6 discussed in section 6.3.5.3.

As discussed above, wrapped distributions inherit from `Distribution` thus have an identical interface to any child of `SDistribution`, with the following minor differences:

- The `wrappedModels` method provides a unified interface to access any component distribution.
- Parameters are still accessed via the same method but stored in a `ParameterSetCollection` object instead of a `ParameterSet`, thus allowing efficient representation of composite and nested parameter sets.

Composition can be iterated and nested any number of times, consider the following example where a mixture distribution is created from two distributions that are in turn composites – a truncated Student T, and a huberized Exponential – note too the parameter inspection and automatic prefixing of distribution 'short names' to the parameters for identification:

```
> M <- MixtureDistribution$new(list(
+  truncate(StudentT$new(), lower = -1, upper = 1),
+  huberize(Exponential$new(), upper = 4)
+ ))
> M$parameters()
                 id   value           support
1:        mix_T_df       1                R+
2: mix_trunc_lower      -1      R U {-Inf, +Inf}
3: mix_trunc_upper       1       R U {-Inf +Inf}
4:    mix_Exp_rate       1                R+
5:   mix_Exp_scale       1                R+
6:   mix_hub_lower       0      R U {-Inf, +Inf}
7:   mix_hub_upper       4      R U {-Inf, +Inf}
8:     mix_weights uniform     {uniform} U [0,1]
```

Implemented Compositors Tables 16 and 17 summarise some important implemented compositors in order to illustrate the way composition is handled and implemented.

Table 16: Examples of common compositors implemented in **distr6** with parameters and type.

Class	Parameters[1]	Type[2]	Components[3]
`TruncatedDistribution`	$a, b \in \mathbb{R}$	$\mathbb{R}$	d', type $\mathbb{R}$
`HuberizedDistribution`	$a, b \in \mathbb{R}$	$\mathbb{R}$, mixed	d', type $\mathbb{R}$
`MixtureDistribution`	$w_i \in \mathbb{R}, \sum_{i=1}^{n} w_i = 1$	$\mathbb{R}^n$	d'_i, type $\mathbb{R}^n$
`ProductDistribution`	-	$\mathbb{R}^N, N = \sum_{i=1}^{n} n_i$	d'_i type $\mathbb{R}^{n_i}$

1. Parameters that the composite has.
2. Type of the resultant distribution d that is created when the class is constructed; this states the formal domain.
3. Number, names, and assumptions on the components, if any.

Table 17: Common compositors implemented in **distr6** with mathematical definitions.

Class	$d.F(x)$[1]	$d.f(x)$[2]
`TruncatedDistribution`[3]	$\frac{d'.F(x)-d'.F(a)}{d'.F(b)-d'.F(a)}$	$\frac{d'.f(x)}{d'.P([a,b])}$
`HuberizedDistribution`	$d'.F(x) + \mathbb{1}[x = b] \cdot d'.P(b)$	-*
`MixtureDistribution`	$\sum_{i=1}^{N} w_i \cdot d'_i.F(x)$	$\sum_{i=1}^{N} w_i \cdot d'_i.f(x)$ **
`ProductDistribution`	$\prod_{i=1}^{N} d'_i.F(x)$	$\prod_{i=1}^{N} d'_i.f(x)$ **

1. The resultant cdf, $d.F$, in terms of the component cdf, as implemented in the `cdf` method of the compositor in the same row.
2. The resultant pdf, $d.f$, in terms of the component pdf, as implemented in the `pdf` method of the compositor in the same row.
3. Truncation is currently only implemented for the left-open interval $(a, b]$.

$*$ – After Huberization, the resultant distribution is in general not absolutely continuous and hence the pdf does not exist.

$**$ – If exists.

d' is defined in table 16.

Example code to obtain a truncated or huberized distribution is below. First a truncated Normal distribution is constructed with truncation parameters -1 and 1, and a huberized Binomial with bounding parameters 2 and 5.

```
> TN <- truncate(Normal$new(), lower = -1, upper = 1)
> TN$cdf(-2:2)
[1] 0.0 0.0 0.5 1.0 1.0
> class(TN)
[1] "TruncatedDistribution" "DistributionWrapper" "Distribution"
[4] "R6"

> HB <- huberize(Binomial$new(), lower = 2, upper = 5)
> HB$cdf(1:6)
[1] 0.0000000 0.0546875 0.1718750 0.3769531 1.0000000 1.0000000
> HB$median()
[1] 5
```

Vectorization of Distributions A special feature of **distr6** is that it allows vectorization of distributions, i.e. vectorized representation of multiple distributions in an array-like structure. This is primarily done for computational efficiency with general best R practice of vectorisation. Vectorisation of **distr6** distributions is implemented via the `VectorDistribution` which is logically treated as a compositor.

Mathematically, a `VectorDistribution` is simply a vector of component distributions $d_1, \ldots, d_N$ that allows vectorized evaluation. Two kinds of vectorized evaluation are supported – paired and product vectorization – which are illustrated below in the case of cdfs.

- Paired vectorized evaluation of the cdfs $d_1.F, \ldots, d_N.F$ at numbers $x_1, \ldots, x_N$, yields a real vector $(d_1.F(x_1), \ldots, d_N.F(x_N))$ via the `cdf` method.
- Product vectorized evaluation of the cdfs $d_1.F, \ldots, d_N.F$ at numbers $x_1, \ldots, x_M$, yields a real $(N \times M)$ matrix, with (i, j)-th entry $d_i.F(x_j)$.

`VectorDistribution` allows for efficient vectorisation across *both* the distributions *and* points to evaluate, which is a feature unique to **distr6** among distribution frameworks in R.

Example code for vectorization via `VectorDistribution` is below. The first `pdf` call shows how to create a vector of two Normal distributions evaluated at different points (product mode), and the second demonstrates evaluation at the same points (paired mode).

```
> V <- VectorDistribution$new(distribution = "Normal",
+  params = data.frame(mean = 1:2))
> V$pdf(1:2, 3:4)
        Norm1      Norm2
1: 0.3989423 0.24197072
2: 0.2419707 0.05399097

> V$pdf(5:6)
            Norm1        Norm2
1: 1.338302e-04 0.0044318484
2: 1.486720e-06 0.0001338302
```

Further, common composites such as `ProductDistribution` and `MixtureDistribution` inherit from `VectorDistribution`, allowing for efficient vector dispatch of pdf and cdf methods. Inheriting from `VectorDistribution` results in identical constructor and methods. Thus a minor caveat is that users could evaluate a product or mixture at different points for each distribution, which is not a usual use-case in practice.

Two different choices of constructors are provided, the first '`distlist`' constructor passes distribution *objects* into the constructor, whereas the second passes a reference to the distribution *class* along with the parameterisations. Therefore the first allows different types of distributions but is vastly slowly as the various methods have to be calculated individually, whereas the second only

allows a single class of distribution at a time, but is much quicker in evaluation. In the example below, the mixture uses the second constructor and the product uses the first.

```
> M <- MixtureDistribution$new(distribution = "Degenerate",
+  params = data.frame(mean = 1:10))
> M$cdf(1:5)
[1] 0.1 0.2 0.3 0.4 0.5
> class(M)
[1] "MixtureDistribution" "VectorDistribution"
[3] "DistributionWrapper" "Distribution" "R6"

> P <- ProductDistribution$new(list(Normal$new(),
+  Exponential$new(), Gamma$new()))
> P$cdf(1:5)
[1] 0.3361815 0.7306360 0.9016858 0.9636737 0.9865692
```

6.3.5. Design Patterns and Object-Oriented Programming

This paper has so far discussed the API and functionality in **distr6**. This section discusses object-oriented programming (OOP), firstly a brief introduction to OOP and OOP in R and then the package's contributions to the field.

6.3.5.1. S3, S4, and R6

R has four major paradigms for object-oriented programming: S3, S4, reference classes (R5), and most recently, R6. S3 and S4 are known as functional object-oriented programming (FOOP) paradigms whereas R5 and R6 move towards class object-oriented programming (COOP) paradigms (R6) [41]. One of the main differences (from a user-perspective) is that methods in COOP are associated with a class whereas in FOOP, methods are associated with generic functions. In the first case methods are called by first specifying the object and in the second, a dispatch registry is utilised to find the correct method to associate with a given object.

S3 introduces objects as typed lists in R, which can hold functions or variables. The functions are called via the dispatch system and every function comprises both a generic and a method for each object. Without a formal definition of a class, S3 does not have a clear concept of object construction. S3 is embedded deep in the infrastructure of R and single dispatch is behind a vast majority of the base functionality and it is part of the main reason why R is easily readable. However, S3 is not a formal OOP language[1] and lacks the concept of classes, constructors and thereby inheritance (although this is possible it isn't well formalised).

[1] http://adv-r.had.co.nz/OO-essentials.html

S4 formalises S3 by introducing the basics of object-oriented programming including the distinction between classes and objects as well as multiple inheritance. S4 formalises class-object separation by constructor functions that exist independently from the class definition. S4 has more syntax for the user to learn and a few more steps in class and method definitions. S4 syntax is quite clunky and not overly user-friendly. In practice, S3 is used vastly more than S4 [41].

There is a big jump from S3 and S4 to R6 as they transition from functional- to class-object-oriented programming. This means new notation, semantics, syntax, and conventions. The key changes are: 1) introducing methods and fields that are associated with classes not functions; 2) mutable objects with copy-on-modify semantics; and 3) new dollar-sign notation. In the first case this means that when a class is defined, all the methods are defined as existing within the class, and these can be accessed at any time after construction. Methods are further split into *public* and *private*, as well as *active bindings*; which incorporates the abstraction part of OOP. The mutability of objects and change to copy-on-modify means that to create an independent copy of an object, the new method `clone(deep = TRUE)` has to be used, which would be familiar to users who know more classical OOP but very different to most R users. Finally methods are accessed via the dollar-sign, and not by calling a function on an object.

Contrasting the three paradigms with a toy example to create a 'duck' class with a method 'quack':

S3

```
> quack <- function(x) UseMethod("quack", x)
> duck <- function(name) return(structure(list(name = name),
+  class = "duck"))
> quack.duck <- function(x) cat(x$name, "QUACK!")
> quack(duck("Arthur"))
Arthur QUACK!
```

S4

```
> setClass("duck", slots = c(name = "character"))
> setGeneric("quack", function(x) {
+  standardGeneric("quack")
+ })
> setGeneric("duck", function(name) {
+  standardGeneric("duck")
+ })
> setMethod("duck", signature(name = "character"),
+  definition = function(name){
+  new("duck", name = name)
+ })
```

```
> setMethod("quack",
+  definition = function(x) {
+  cat(x@name, "QUACK!")
+ })
> quack(duck("Ford"))
Ford QUACK!
```

R6

```
> duck <- R6::R6Class("duck", public = list(
+  initialize = function(name) private$.name = name,
+  quack = function() cat(private$.name, "QUACK!")),
+  private = list(.name = character(0)))
> duck$new("Zaphod")$quack()
Zaphod QUACK!
```

The example clearly highlights the extra code introduced by S4 and the difference between the S3 dispatch and R6 method system.

Comparing the Paradigms There is no doubt that R6 is the furthest paradigm from conventional R usage and as such there is a steep learning curve for the majority of R users. However R6 will be most natural for users coming to R from more traditional OOP languages. In contrast, S3 is a natural FOOP paradigm that will be familiar to all R users (even if they are not aware that S3 is being used). S4 is an unfortunate midpoint between the two, which whilst being very useful, is not particularly user-friendly in terms of programming classes and objects. **distr** was developed soon after S4 was released and is arguably one of the best case-studies for how well S4 performs. Whilst S4 formalises S3 to allow for a fully OO interface to be developed, its dependence on inheritance forces design decisions that quickly become problematic. This is seen in the large inheritance trees in **distr** in which one implemented distribution can be nested five child classes deep. This is compounded by the fact that S4 does not use pointer objects but instead nests objects internally. Therefore **distr** has problems with composite distributions in that they quickly become very large in size, for example a mixture of two distributions can easily be around 0.5Mb, which is relatively large. In contrast, R6 introduces pointers, which means that a wrapped object simply points to its wrapped component and does not copy it needlessly. Whilst a fully object-oriented interface can be developed in S3 and S4, they do not have the flexibility of R6, which means that in the long run, extensibility and scalability can be problematic. R6 forces R users to learn a paradigm that they may not be familiar with but packages like **R62S3** allow users to become acquainted with R6 on a slightly shallower learning curve. Speed differences for the three paradigms are formally compared on the example above using **microbenchmark** [215], the results are in table 18. The R6 example is compared both including construction of the class, `duck$new("Zaphod")$quack()`, and without construction, `d$quack()`, where `d` is the object constructed before comparison. A significant 'bottleneck' is noted when construction is included in the comparison but despite this S4 is still significantly the slowest.

Table 18: Comparing S3, S4, and R6 in calling a method. R6 is tested both including object construction (R6) and without (R6*).

Paradigm	**mean** (μs)	**cld**[1]
S3	73.44	a
S4	276.17	c
R6	187.70	b
R6*	38.32	a

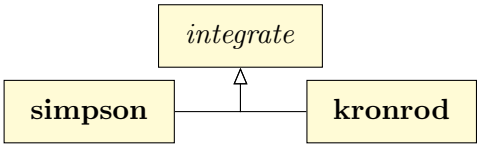
1. Significance test for run-time of each method where 'a' is fastest and 'c' is slowest. The experiment is conducted on R v4.0.2 (2020-06-22); Platform: x86_64-apple-darwin17.0 (64-bit); Running under: macOS Catalina 10.15.3 with R6 v2.4.1 and **microbenchmark** v1.4.7.

6.3.5.2. Design Patterns

In the simplest definition, 'design patterns' are abstract solutions to common coding problems. They are probably most widely known due to the book 'Design Patterns Elements of Reusable Object-Oriented Software' (*Design Patterns*) [91]. **distr6** primarily makes use of the following design patterns

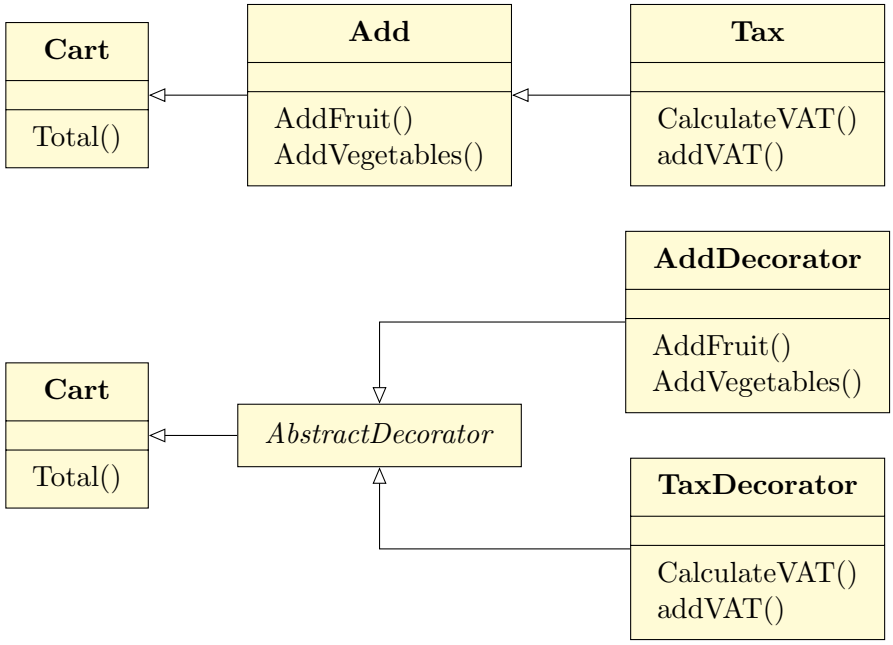
- Abstract Factory
- Decorator
- Composite
- Strategy

Strategy The strategy pattern is common in modelling toolboxes, in which multiple algorithms can be used to solve a problem. This pattern defines an abstract class for a given problem and concrete classes that each implement different strategies, or algorithms, to solve the problem. For example in the context of mathematical integration (a common problem in R), one could use Simpson's rule, Kronrod's, or many others. These can be specified by an `integrate` abstract class with concrete sub-classes `simpson` and `kronrod`.



Composite The composite pattern defines a collection of classes that have an identical interface when treated independently or when composed into a single class with constituent parts. To the user, this means that only one interface needs to be learnt in order to interact with composite or individual classes. A well-built composite pattern allows users to construct complex classes with several layers of composition, and yet still be able to make use of a single interface. By inheriting from a parent class, each class and composite share a common interface.

Decorator Decorators add additional responsibilities to an object without making any other changes to the interface. An object that has been decorated will be identical to its un-decorated counter-part except with additional methods. This provides a useful alternative to inheritance. Whereas inheritance can lead to large tree structures in which each sub-class inherits from the previous and contains all previous methods, decorators allow the user to pick and choose with responsibilities to add. The figure below demonstrates how this is useful in a shopping cart example. The top of the figure demonstrates using inheritance, in which each sub-class adds methods to the `Cart` parent class. By the `Tax` child class there are a total of five methods in the interface. In the bottom of the figure, the decorator pattern demonstrates how the functionality for adding items and tax is separated and can be added separately.



6.3.5.3. Contributions to R6

In order to implement **distr6**, several contributions were made to the R6 paradigm, to extend its abilities and to implement the design patterns discussed above.

Abstract Classes R6 did not have a concept of abstract classes, which meant that patterns such as adapters, composites, and decorators, could not be directly implemented without problems. This is produced in **distr6** with the `abstract` function, which is placed in the first line of all abstract classes. In the example below, `obj` expects the `self` argument from R6 classes, and `class` is the name of the class, `getR6Class` is a custom function for returning the name of the class of the given object.

```
1  > abstract <- function(obj, class) {
2  +  if (getR6Class(obj) == class) {
3  +   stop(sprintf("%s is an abstract class that can't be initialized.",
4  +   class))
```

```
+  }
+ }
```

For example in decorators the following line is placed at the top of the `initialize` function:

```
> abstract(self, "DistributionDecorator")
```

Decorators The typical implementation of decorators is to have an abstract decorator class with concrete decorators inheriting from this, each with their own added responsibilities. In **distr6** this is made possible by defining the `DistributionDecorator` abstract class (see above) with a public `decorate` method. Concrete decorators are simply R6 classes where the public methods are the ones to 'copy' to the decorated object.

```
> DistributionDecorator
<DistributionDecorator> object generator
  Public:
    packages: NULL
    initialize: function ()
    decorate: function (distribution, ...)
    clone: function (deep = FALSE)

> CoreStatistics
<CoreStatistics> object generator
  Inherits from: <DistributionDecorator>
  Public:
    mgf: function (t)
    cf: function (t)
    pgf: function (z)
```

When the `decorate` method from a constructed decorator object is called, the methods are simply copied from the decorator environment to the object environment. The `decorate()` function simplifies this for the user.

Composite and Wrappers The composite pattern is made use of in what **distr6** calls 'wrappers'. Again this is implemented via an abstract class (`DistributionWrapper`) with concrete sub-classes.

```
> DistributionWrapper
<DistributionWrapper> object generator
  Inherits from: <Distribution>
  Public:
    initialize: function (distlist = NULL, name, short_name,
    wrappedModels: function (model = NULL)
```

```
    setParameterValue: function (..., lst = NULL, error = "warn")
  Private:
    .wrappedModels: list

> TruncatedDistribution
<TruncatedDistribution> object generator
  Inherits from: <DistributionWrapper>
  Public:
    initialize: function (distribution, lower = NULL, upper = NULL)
    setParameterValue: function (..., lst = NULL, error = "warn")
  Private:
    .pdf: function (x, log = FALSE)
    .cdf: function (x, lower.tail = TRUE, log.p = FALSE)
    .quantile: function (p, lower.tail = TRUE, log.p = FALSE)
    .rand: function (n)
```

Wrappers in **distr6** alter objects by modifying either their public or private methods. Therefore an 'unwrapped' distribution looks identical to a 'wrapped' one, despite inheriting from different classes. This is possible via two key implementation strategies: 1) on construction of a wrapper, parameters are prefixed with a unique ID, meaning that all parameters can be accessed at any time; 2) the `wrappedModels` public field allows access to the original wrapped distributions. These two factors allow any new method to be called either by reference to `wrappedModels` or by using `getParameterValue` with the newly prefixed parameter ID. This is demonstrated in the `.pdf` private method of the `TruncatedDistribution` wrapper (slightly abridged):

```
> .pdf = function(x, log = FALSE) {
+  dist <- self$wrappedModels()[[1]]
+  lower <- self$getParameterValue("trunc_lower")
+  upper <- self$getParameterValue("trunc_upper")
+
+  pdf <- numeric(length(x))
+  pdf[x > lower & x <= upper] <- dist$pdf(x[x > lower & x <= upper]) /
+   (dist$cdf(upper) - dist$cdf(lower))
+
+  return(pdf)
+ }
```

As the public `pdf` is the same for all distributions, and this is inherited by wrappers, only the private `.pdf` method needs to be altered.

6.3.6. Examples

This final section looks at concrete short examples for four key use-cases. The output figures from the examples are in appendix D.

6.3.6.1. Constructing and Querying Distributions

The primary use-case for the majority of users will be in constructing distributions in order to query their results and visualise their shape.

Below, a Binomial distribution is constructed and queried for its distribution-specific traits and parameterisation-specific properties.

```
> b <- Binomial$new(prob = 0.1, size = 5)
> b$setParameterValue(size = 6)
> b$getParameterValue("size")
> b$parameters()
> b$properties
> b$traits
```

Specific methods from the distribution are queried as well.

```
> b$mean()
> b$entropy()
> b$skewness()
> b$kurtosis()
> b$cdf(1:5)
```

The distribution is visualised by plotting its density, distribution, inverse distribution, hazard, cumulative hazard, and survival function.

```
> plot(b, fun = "all")
```

6.3.6.2. Analysis of Empirical Data

distr6 can also serve as a toolbox for analysis of empirical data by making use of the three 'empirical' distributions: `Empirical`, `EmpricalMV`, and `WeightedDiscrete`.

First an empirical distribution is constructed with samples from a standard exponential distribution.

```
> E <- Empirical$new(samples = rexp(10000))
```

The `summary` function is used to quickly obtain key information about the empirical distribution.

```
> summary(E)

Empirical Probability Distribution.

  Quick Statistics
        Mean:           0.9983612
```

```
          Variance:       1.031437
          Skewness:       2.066763
          Ex. Kurtosis:   6.236536

 Support: (0.00,...,9.18)          Scientific Type: ℝ

 Traits: discrete; univariate
 Properties: asymmetric; leptokurtic; positive skew
```

The distribution is compared to a (standard) Normal distribution and then (standard) Exponential distribution.

```
> qqplot(E, Normal$new(), xlab = "Empirical", ylab = "Normal")
> qqplot(E, Exponential$new(), xlab = "Empirical", ylab = "Exponential")
```

The CDF of a bivariate empirical distribution is visualised.

```
> plot(EmpiricalMV$new(data.frame(rnorm(100, mean = 3), rnorm(100))),
+  fun = "cdf")
```

6.3.6.3. Learning from Custom Distributions

Whilst empirical distributions are useful when data samples have been generated, custom distributions can be used to build an entirely new probability distribution – though this example uses a simple discrete uniform distribution. This example highlights the power of decorators to estimate distribution results without manual computation of every possible method. The output demonstrates the precision and accuracy of these results.

Below, a custom distribution is created and, by including the `decorators` argument, all further methods are imputed numerically. The distribution is summarised for properties, traits and common results (this is possible with the 'CoreStatistics' decorator). The summary is identical to the analytic `DiscreteUniform` distribution.

```
> U <- Distribution$new(
+  name = "Discrete Uniform",
+  type = set6::Integers$new(), support = set6::Set$new(1:10),
+  pdf = function(x) ifelse(x < 1 | x > 10, 0, rep(1/10,length(x))),
+  decorators = c("CoreStatistics",
+   "ExoticStatistics", "FunctionImputation"))
> summary(U)

Discrete Uniform

  Quick Statistics
```

```
        Mean:           5.5
        Variance:       8.25
        Skewness:       0
        Ex. Kurtosis:   -1.224242

  Support: {1,...,10}         Type: ℤ

  Traits: discrete; univariate
  Properties: asymmetric; platykurtic; no skew

  Decorated with:  CoreStatistics, ExoticStatistics, FunctionImputation
```

The CDF and simulation functions are called (numerically imputed with the `FunctionImputation` decorator), the hazard function from the `ExoticStatistics` decorator, and the `kthmoment` function from the `CoreStatistics` decorator.

```
> U$cdf(1:10)
[1] 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
> U$rand(10)
[1]  8 10  5  8  5 10  6  7  1  4
> U$hazard(2)
[1] 0.125
> U$kthmoment(2)
[1] 8.25
```

6.3.6.4. Composite Distribution Modelling

Composite distributions are an essential part of any distribution software, the following example demonstrates two types of composites: composition via distribution transformation (truncation), and composition via mixtures and vectors.

First, a Binomial distribution is constructed and truncated between 1 and 5, the CDF of the new distribution is queried.

```
> TB <- truncate(
+  Binomial$new(size = 20, prob = 0.5),
+  lower = 1,
+  upper = 5
+ )
> round(TB$cdf(0:6), 4)
[1] 0.0000 0.0000 0.0088 0.0613 0.2848 1.0000 1.0000
```

Next, a vector distribution is constructed of two Normal distributions, with respective means 1 and 2 and unit standard deviation. The parameters are queried (some columns suppressed).

```
> V <- VectorDistribution$new(distribution = "Normal",
+  params = data.frame(mean = 1:2))
> V$parameters()
            id value support
1: Norm1_mean     1       R
2:  Norm1_var     1      R+
3:   Norm1_sd     1      R+
4: Norm1_prec     1      R+
5: Norm2_mean     2       R
6:  Norm2_var     1      R+
7:   Norm2_sd     1      R+
8: Norm2_prec     1      R+
```

Vectorisation is possible across distributions, samples, and both. In the example below, the first call to `pdf` evaluates both distributions at (1, 2), the second call evaluates the first at (1) and the second at (2), and the third call evaluates the first at (1, 2) and the second at (3, 4).

```
> V$pdf(1:2)
       Norm1     Norm2
1: 0.3989423 0.2419707
2: 0.2419707 0.3989423
> V$pdf(1, 2)
       Norm1     Norm2
1: 0.3989423 0.3989423
> V$pdf(1:2, 3:4)
       Norm1      Norm2
1: 0.3989423 0.24197072
2: 0.2419707 0.05399097
```

Finally a mixture distribution with uniform weights is constructed from a $\mathcal{N}(2, 1)$ and Exp(1).

```
> MD <- MixtureDistribution$new(
+  list(Normal$new(mean = 2, sd = 1), Exponential$new(rate = 1))
+ )
> MD$pdf(1:5)
[1] 0.304925083 0.267138782 0.145878896 0.036153303 0.005584898
> MD$cdf(1:5)
[1] 0.3953879 0.6823324 0.8957788 0.9794671 0.9959561
> MD$rand(5)
[1] 3.6664473 0.1055126 0.6092939 0.8880799 3.4517465
```

6.3.7. Conclusion and Availability

distr6 introduces a robust and scalable object-oriented interface for probability distributions to R. It officially upgrades the **distr** family of packages and aims to be the first-stop for object-oriented probability distributions in R. By making use of R6, every implemented distribution is clearly defined with properties, traits, and analytic results. Whilst R **stats** is limited to very basic `dpqr` functions for representing evaluated distributions, **distr6** ensures that probability distributions are treated as complex mathematical objects.

Future updates of the package will include adding further numerical approximation strategies in the decorators to allow users to choose different methods (instead of being forced to use one). Additionally, the extensions to R6 could be abstracted into an independent package in order to better benefit the R community.

distr6 is released under an MIT licence on GitHub and CRAN. Extended documentation, tutorials, and examples are available on the project website[1]. Code quality is monitored and maintained by an extensive suite of unit tests with GitHub Actions on multiple operating systems.

6.4. mlr3proba: Machine Learning Survival Analysis in R

6.4.1. Introduction

[2]**mlr3proba** is part of the **mlr3** family of packages, the '**mlr3verse**'. **mlr3** [182] is the official upgrade to **mlr** [22], both developed by the mlr-org core team[3]. Whilst **mlr** included an interface for classification and regression, with many learners and measures, there was less support for survival analysis. **mlr3** has been designed with modularisation in mind, this means that instead of all functionality being grouped into one large package, there are instead many smaller packages. **mlr3proba**'s place in this universe (fig. 38) is to add probabilistic supervised learning, which includes survival analysis, density estimation, and probabilistic regression.

To-date work on **mlr3proba** has primarily focused on survival analysis though support is also available for density estimation and probabilistic regression. For time and scope of this thesis, only the survival analysis aspects will be discussed here.

Motivating Example Listing 7 gives an example of how to benchmark three survival models, set hyper-parameter values, and make use of the distribution compositor. Line 1: Essential packages are loaded, **mlr3proba** always requires

[1] https://alan-turing-institute.github.io/distr6/
[2] Parts of this section have been published as part of a paper in *Bioinformatics* [281].
[3] https://github.com/mlr-org

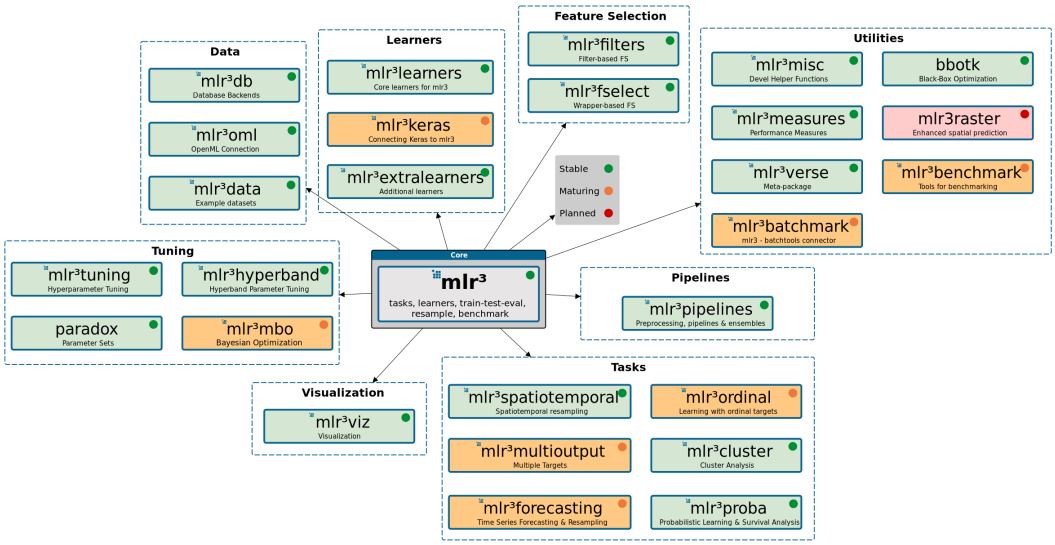


Figure 38: The **mlr3verse**. **mlr3proba** sits in the bottom-right with Tasks. Image copied from https://mlr3.mlr-org.com.

mlr3. Line 2: Extra packages are loaded, **mlr3extralearners** is required for the GBM learner and **mlr3pipelines** is required for the distribution composition. Lines 3-4: Kaplan-Meier and Cox PH learners are constructed with default parameters. Lines 5-6: The GBM learner is wrapped in the `distrcompositor` pipeline to transform its ranking prediction to a probabilistic prediction. Line 7: Learners are combined into a list for use in the benchmark function. Line 8: The pre-specified `rats` task is selected for the experiment. Line 9: Three-fold cross-validation is specified. Line 10: The infrastructure for the experiment is automatically determined by supplying the task(s), learners, and resampling method. Line 11: Learners are resampled according to the chosen scheme and benchmarked. Line 12: Predictions are aggregated over all folds and scored with the IGS to provide a final comparison.

Listing 7 Example code for constructing, benchmarking, and evaluating survival models.

```
1  > library(mlr3); library(mlr3proba)
2  > library(mlr3extralearners); library(mlr3pipelines)
3  > kaplan = lrn("surv.kaplan")
4  > cox = lrn("surv.coxph")
5  > gbm = ppl("distrcompositor", learner = lrn("surv.gbm"),
6  +   estimator = "kaplan", form = "ph")
7  > learns = list(cox, kaplan, gbm)
8  > task = tsk("rats")
9  > resample = rsmp("cv", folds = 3)
10 > design = benchmark_grid(task, learns, resample)
11 > bm = benchmark(design)
12 > bm$aggregate(msr("surv.graf"))
```

6.4.2. Related Software

There are an increasing number of machine learning packages across programming languages, including **caret** [170], **mlr** [22], **tidymodels** [171], and **scikit-learn** [236]. However, functionality for survival analysis has been mostly limited to 'classical' statistical models with relatively few packages supporting a machine learning framework.

R ships with the package **survival** [291], which supports left-,
interval-,and right-censoring, competing risks, time-dependent models, stratification, and model evaluation. However the package is limited to classical statistical models, with no support for machine learning and limited support for formal comparison or non-linear models.

pec [217] implements no models itself but instead interfaces with many different survival packages to create survival probability predictions. The package's main focus is on model evaluation via prediction error curves ('pec's) with little support for model building/training and predicting.

skpro [111] is a probabilistic supervised learning interface in Python. **skpro** extends the **scikit-learn** interface to probabilistic models and appears to be the only package (in any language) dedicated to domain-agnostic probabilistic supervised learning. The interface provides an infrastructure for machine learning based survival analysis with design choices influencing **mlr3proba**, but **skpro** does not currently support survival models.

pysurvival [83] is another Python package, which implements classical and machine learning survival analysis models. The package has the advantage of being able to natively leverage neural network survival models, which are almost exclusively implemented in Python. Whilst not directly interfacing the **scikit-learn** interface, the package introduces unified functions for model fitting, predicting, and evaluation.

scikit-survival [235] builds directly on **scikit-learn** to implement a few survival models and measures in a machine learning framework. Unlike **pysurvival**, no neural networks are included, thus the two packages complement each other well.

6.4.3. Use-Cases and Requirements

The use-cases, requirements, and design principles of this package were primarily dictated by **mlr3**, however there were a few that are important to **mlr3proba** in its own right.

Use-Cases **mlr3proba** serves the following use-cases:

U1) **Fitting and predicting survival models** This is the first and foremost use-case of the survival framework in the package. By making use of the **mlr3** train/predict methods, users can treat classical survival models just like machine learning ones. In the example below: a Cox model and random forest are constructed (Lines 1-2), a pre-specified task is requested (Line 3), and the two models are trained and tested on different observations (Lines 4-7).

```
> cox = lrn("surv.coxph")
> ranfor = lrn("surv.rfsrc")
> task = tsk("rats")
> train = sample(300, 150)
> test = setdiff(seq(300), train)
> cox$train(task, row_ids = train)$predict(task, row_ids = test)
> ranfor$train(task, row_ids = train)$predict(task, row_ids = test)
```

U2) **Inspection of fitted survival models** Whilst many machine learning models are a 'black-box' and cannot be inspected, there are several interpretable classical survival models, such as GLMs; inspecting these is informative for exploration and prediction. Different packages in R have different ways of inspecting fitted models, this is unified by the `model` field, which is exemplified below.

```
> lrn("surv.coxph")$train(tsk("rats"))$model
```

U3) **Tuning of machine leaning models** Users can make use of **mlr3tuning** [180] with any of the implemented survival models in order to tune and improve the available ML models.

U4) **Evaluation of survival models with transparent measures** Whilst several packages exist in R for the evaluation of survival models (section 6.4.2), each has a different API that must be learnt and interfaced. This is streamlined with **mlr3**'s measure objects.

Requirements **mlr3proba** fulfils the following requirements:

R1) **Part of the mlr3verse** The primary requirement of **mlr3proba** is to fit into the **mlr3** ecosystem. This means that all classes should inherit from classes in **mlr3** where possible, documentation should be written in the same manner, and **paradox** [181] should be used to define learner parameter sets.

R2) **Transparency in measures** Derived from (U4), **mlr3proba** requires that all models should be evaluated only by measures that have been tried and tested. Slightly worse approximations are preferred if they are more transparent in their functionality (and on the assumption they will be improved). Transparency requires that all implemented measures are well-documented with clear maths to define them.

R3) **Transparency in compositions** Several toolboxes implement methods to compose distributions from predictions of linear predictors, however there is a lack of consistency in how these compositions are performed and little-to-no user control or documentation. **mlr3proba** abstracts composition to separated functions in order to ensure that the user is always clear what each package returns and how this can be transformed between prediction types.

R4) **Transparency in return types** This relates to (R3). Most packages implement models with a `predict` function that returns a survival prediction, however poor documentation means that it is often unclear what exactly is predicted. Some return a linear predictor (e.g. `survival::coxph`), some will return a full distribution (e.g. `ranger::ranger`), and some will return a continuous ranking (e.g. `survivalsvm::survivalsvm`). However the documentation for each of these does not make clear: i) which is returned; ii) how one can be derived from the other; and iii) how they should be interpreted. **mlr3proba** requires that this distinction is made clear in documentation, as well as how to derive each return type from one another, and how to interpret them.

6.4.4. Overview to Functionality and API

Many of the methods and fields are implemented via the **mlr3** package and thus to avoid any overlap in writing, core methods and objects will not be discussed in detail here, instead see the various documentation that are provided for **mlr3**[1]. Figure 39 provides a reference of the main classes in **mlr3proba**, with their **mlr3** dependencies. The classes in the top row of the diagram all live in **mlr3**, all others in **mlr3proba**. The classes 'LearnerSurvX' and 'MeasureSurvX' indicate the implemented learners (table 19) and measures (table 20). Below is an overview to the API specific to **mlr3proba**.

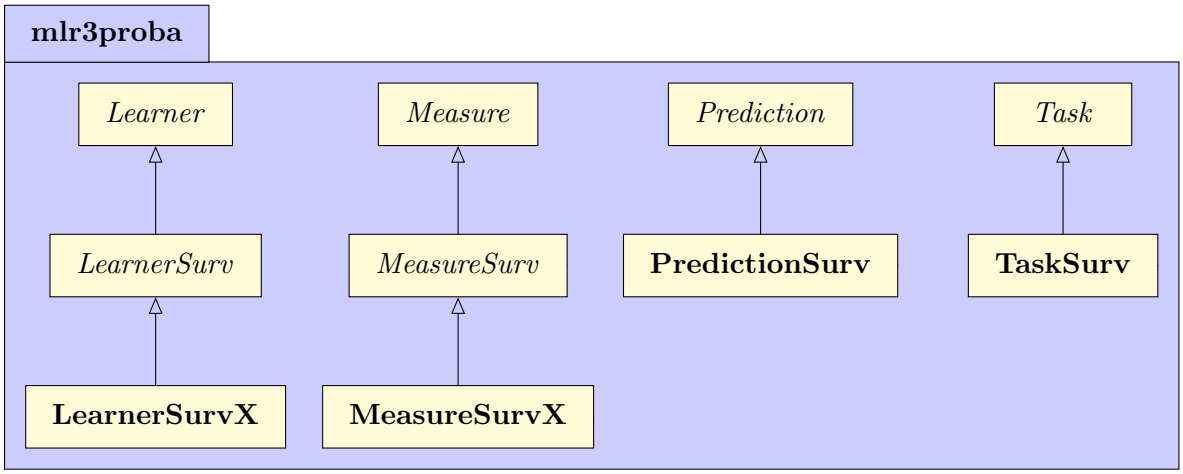


Figure 39: Simplified **mlr3proba** class diagram. The classes in the top row all live in **mlr3**, the others live in **mlr3proba**. Abstract classes are in italics, concrete class are in bold.

6.4.4.1. Task

In all **mlr3** packages, the `Task` object is the central point of reference for all measures, learners, and other classes. `TaskSurv` is constructed to initialize a survival task.

[1]The mlr3book is a good place to start https://mlr3book.mlr-org.com/.

```
> TaskSurv$new(id, backend, time, event, time2,
+  type = c("right","left","counting","interval","interval2","mstate"))
```

The survival task has three arguments that correspond to the 'target'. These are `time`, `time2`, and `event`, which are passed directly to a `Surv` object from **survival**. `time` is a string pointing to the columns in the data, `backend`, which identify the survival times, `time2` is additionally used for interval censoring. `event` is a string pointing to the column in the data with the censoring indicator. The `event` argument corresponds to the type of censoring, the default is `right` censoring. As of **mlr3proba** v0.2.2, no models are implemented that can handle left- or interval-censoring, this is a general problem with survival models implemented in R.

Tasks can either be constructed using the `backend` argument to supply the data, constructed from pre-specified tasks in the **mlr3** registry, or simulated; each of the three is demonstrated below.

```
# Constructing a new task
> data("rats", package = "survival")
> TaskSurv$new(id = "rats", backend = rats, time = "time",
+  event = "status", type = "right")
+
# Using 'tsk' for a pre-specified task
> tsk("lung")
+
# Using 'tgen' to simulate a task from the package `simsurv`
> tgen("simsurv")$generate(200)
```

Meta-analysis is automatically performed on tasks to identify useful properties.

```
>  tsk("rats")
<TaskSurv:rats> (300 x 5)
* Target: time, status
* Properties: -
* Features (3):
  - int (1): litter
  - dbl (1): rx
  - chr (1): sex
```

6.4.4.2. Learners

LearnerSurv All fields and methods of the learners in the package inherit from the `LearnerSurv` class, which in turn inherits from `Learner` in **mlr3**. Survival learners have their '`task_type`' set to '`surv`'. Their '`predict_type`' must be one or more of: `crank`, `lp`, `distr`, `response` (see below). The code below demonstrates three methods of initializing the same learner.

```
> lrn("surv.coxph")
> LearnerSurvCoxPH$new()
> mlr_learners$get("surv.coxph")
```

Fit, Predict, Inspect Learners are trained, predicted, and inspected using inherited methods from **mlr3**.

```
> task = tsk("rats")
> learn = lrn("surv.coxph")
# Train a model
> learn$train(task)
# In a real example, models would not be trained
# and tested on the same data.
> learn$predict(task)
# inspection is available for the fitted model and more meta-data
> learn$model
> learn$state
```

Return Types A key advantage of **mlr3proba**, is a clear distinction between model prediction types. Survival models can be used to produce a variety of different prediction types but implementation has historically not reflected this. In other supervised learning fields this is not a problem as regression always predicts a continuous value for the outcome, and classification either predicts a category or a probability (the two can immediately be seen to be different). However in survival analysis there are several different possible predictions that could be made and without clear documentation, these can look very similar. For example, if a user wanted to compare predictions from Cox PH and survival tree models without **mlr3proba**, this would require the following steps: i) Train and predict from the models using two separate packages; ii) identify that the model predictions are not directly comparable; iii) use a third package to transform the predictions into a compatible form (for example combining a relative risk prediction with a baseline hazard estimation); iv) identify which measures can be used to evaluate this form; and v) find the package that includes these measures and potentially write functions to interface with the package.

In **mlr3proba**, this distinction is made clear by defining four distinct prediction types: i) `response`, which returns the predicted survival time (expected time until the event occurs); ii) `lp`, which returns a prediction for the linear predictor of a linear model; iii) `crank`, a continuous ranking for comparing the relative risk between observations in the test sample; iv) `distr`, a survival distribution implemented via **distr6**, which includes functionality for evaluating the survival and hazard function.

The `crank`, 'continuous rank', return type was introduced as part of this package and is non-standard beyond this, however it neatly ties together predictions from multiple models. `crank` represents a relative risk, therefore a higher predicted value indicates that the individual is more likely to experience the event than someone with a lower predicted value. All models return a relative risk

either implicitly or explicitly. If a model predicts an abstract ranking (such as `surv.svm`) then `crank` is this ranking, if the model predicts a linear predictor then `crank = lp`, if a model returns a `response` then `crank = response`, or if the model only returns `distr` then `crank = -distr$mean()`, the negative mean of the predicted distribution. As a consequence, the `crank` is always returned and at a minimum all models can be compared with discrimination measures. The choice of imputation of `crank` by `-distr$mean()` is a natural default for a relative ranking as it has a natural interpretation as a survival time prediction, note that users can also override this with the `crankcompositor` pipeline (section 6.4.4.4).

One of the advantages of the `crank` return type is to provide a unified interpretation of predicted linear predictors. By convention, fitted linear predictors are usually coded such that higher values signify higher risk for PH models but lower risk for all other model forms; this is seen in packages **survival** [291] and **mboost** [132] amongst others. The `crank` return type in **mlr3proba** guarantees the 'higher value higher risk' interpretation and as such users will know exactly how to interpret these values, even if unfamiliar with the above convention.

This problem of return types is important as different types are not comparable and historically this has not been successfully managed, leading to impractical benchmark experiments. By defining these prediction types as different objects in **mlr3proba**, incompatible benchmark experiments are avoided by internal validation checks. By making use of **mlr3pipelines** [21], transformations between prediction types can be made, with clear documentation and parameters that ensure a simple interface for the user (section 6.4.4.4).

Implemented Learners 30+ learners are interfaced in **mlr3proba** 0.2.2, these are shown, with their respective packages, in table 19. A few of these learners are implemented directly in **mlr3extralearners** or **survivalmodels** [275] either to improve performance or because no other implementation could be found in R.

Table 19: Learners implemented in **mlr3proba** along with **mlr3proba** key and unique ID from table 5.

mlr3proba key[1]	**ID/Name**[2]	**Package**[3]
surv.akritas^E	Akritas Estimator	survivalmodels [275]
surv.blackboost^E	GBM-COX/AFT/GEH/UNO	mboost [132]
surv.cforest^E	RSCIFF	partykit [129]
surv.coxboost^E	CoxBoost	CoxBoost [20]
surv.coxph^P	Cox PH	survival [291]
surv.coxtime^E	Cox-Time	survivalmodels
surv.ctree^E	SDCIFT	partykit
surv.cv_coxboost^E	Cross-Validated CoxBoost	CoxBoost
surv.cv_glmnet^L	Cross-Validated Regularized GLM	glmnet [87]
surv.deephit^E	DeepHit	survivalmodels
surv.deepsurv^E	DeepSurv	survivalmodels
surv.dnn^E	DNNSurv	survivalmodels
surv.flexible^E	Flexible Parametric Splines	flexsurv [141]
surv.gamboost^E	GBM-COX/AFT/GEH/UNO	mboost
surv.gbm^E	GBM-COX	gbm [110]
surv.glmboost^E	GBM-COX/AFT/GEH/UNO	mboost
surv.glmnet^L	Regularized GLM	glmnet
surv.kaplan^P	Kaplan-Meier Estimator	survival

Continued on next page...

Table 19: (continued)

mlr3proba key[1]	**ID/Name**[2]	**Package**[3]
surv.loghaz^E	Nnet-Survival	survivalmodels
surv.mboost^E	GBM-COX/AFT/GEH/UNO	mboost
surv.nelson^E	Nelson-Aalen Estimator	survival
surv.obliqueRSF^E	Oblique Random Survival Forest	obliqueRSF [143]
surv.parametric^E	Fully Parametric Survival Models	survival
surv.pchazard^E	PC-Hazard	survivalmodels
surv.penalized^E	L1 and L2 Penalized GLMs	penalized [101]
surv.ranger^L	RSDF-STAT	ranger [325]
surv.rfsrc^E	RSDF-STAT	randomForestSRC [139]
surv.rpart^P	RRT	rpart [292]
surv.svm^E	SVCR/RANKSVMC/SSVMVB1/SSVMVB2	survivalsvm [84]
surv.xgboost^L	GBM-COX	xgboost [45]

1. Key passed to `lrn` in order to construct the learner. Learners marked with 'E' are implemented in **mlr3extralearners** [280], with 'L' are implemented in **mlr3learners** [183], and with 'P' in **mlr3proba**.
2. Either ID from table 5 (if included) or name of the model. Slashes indicate that multiple models can be run with this learner.
3. Package in which the algorithm is implemented.

6.4.4.3. Measures

MeasureSurv All fields and methods of the measures in the package inherit from the `MeasureSurv` class, which in turn inherits from `Measure` in **mlr3**. Survival measures have their '`task_type`' set to '`surv`'. Measures can evaluate one of: `crank`, `lp`, `distr`, and `response`. The code below demonstrates three methods of initializing the same measure.

```
> msr("surv.logloss")
> MeasureSurvLogloss$new()
> mlr_measures$get("surv.logloss")
```

As all learners return `crank`, the default measure is Harrell's Concordance index [116].

```
> task = tsk("rats")
> pred =  lrn("surv.coxph")$train(task)$predict(task)
> pred$score()
surv.harrellC
    0.7780967
```

Alternatively, measures can be specified by passing the constructed measure to one of the scoring methods.

```
> pred$score(msr("surv.intlogloss"))
surv.intlogloss
     0.03045892
```

Some measures will have parameters that can be set in construction. For example the time-points for which the integrated Graf score should be integrated over can be provided via the `times` argument.

```
> # integrated over times 0-60
> meas1 = msr("surv.graf", times = 0:60, id = "Graf60")
> # integrated over all times
> meas2 = msr("surv.graf", id = "GrafAll")
> pred$score(c(meas1, meas2))
     Graf60     GrafAll
0.008550876 0.045674017
```

Implemented Measures Over 20 measures are implemented in **mlr3proba** 0.2.2, either via **survAUC** [240] or implemented directly in **mlr3proba**, these are listed in table 20.

To maximise user-control over how measures are calculated, all integrated scores implemented in **mlr3proba** have a `method` argument to determine how the approximation to integration should be calculated. Currently two methods are implemented, these are to either approximate integration by taking the sample

Table 20: Measures implemented in **mlr3proba**.

ID[1]	Measure	Package[2]
surv.calib_alpha*	van Houwelingen's Alpha	**mlr3proba**
surv.calib_beta*	van Houwelingen's Beta	**mlr3proba**
surv.chambless_auc	Chambless and Diao's AUC	**survAUC**
surv.cindex†	Concordance Indices	**mlr3proba**
surv.graf*	Integrated Graf Score	**mlr3proba**
surv.hungAUC	Hung and Chiang's AUC	**survAUC**
surv.intlogloss*	Integrated Log Loss	**mlr3proba**
surv.logloss*	Log Loss	**mlr3proba**
surv.mae*	Mean Absolute Error	**mlr3proba**
surv.mse*	Mean Square Error	**mlr3proba**
surv.nagelk_r2	Nagelkerke's R2	**survAUC**
surv.oquigley_r2	O'Quigley, Xu, and Stare's R2	**survAUC**
surv.rmse*	Root Mean Square Error	**mlr3proba**
surv.schmid	Schmid Absolute Score	**mlr3proba**
surv.song_auc	Song and Zhou's AUC	**survAUC**
surv.song_tnr	Song and Zhou's TNR	**survAUC**
surv.song_tpr	Song and Zhou's TPR	**survAUC**
surv.uno_auc	Uno's AUC	**survAUC**
surv.uno_tnr	Uno's TNR	**survAUC**
surv.uno_tpr	Uno's TPR	**survAUC**
surv.xu_r2	Xu and O'Quigley's R2	**survAUC**

1. Key passed to `msr` to construct the measure.
2. Package in which the measure is implemented.
∗ – These measures have an additional `se` argument that can be used to request the standard error is returned.
† – This is six different concordance measures, which can be specified with the `weight_meth` argument.

mean over all time-points, or to take the mean weighted by the difference in time-points. Mathematically, let N be the number of observations and M be the number of time-points, then let $\mathbf{L}$ be a matrix of losses, $\mathbf{L} = L_{i,j}, i = 1, ..., N, j = 1, ..., M$. Both methods first take the sample mean over all N observations

$$L_j = \frac{1}{N} \sum_{i=1}^{N} L_{i,j}, \quad j = 1, ..., M \tag{6.4.1}$$

Method 1 calculates the integrated score via

$$\frac{1}{M} \sum_{j=1}^{M} L_j \tag{6.4.2}$$

Whereas for Method 2,

$$\sum_{j=2}^{M} \frac{(T_j - T_{j-1})(L_j + L_{j-1})}{2(T_M - T_1)} \tag{6.4.3}$$

where $T_1, ..., T_M$ are the unique time-points.

Neither of these is necessarily 'better' than the other and so the `method` argument provides the freedom to choose. The second method is the default to be consistent with other packages.

```
> meas1 = msr("surv.graf", method = 1, id = "Graf.M1")
> meas2 = msr("surv.graf",method = 2, id = "Graf.M2")
> learn$score(c(meas1, meas2))
   Graf.M1    Graf.M2
0.06495095 0.04567402
```

Standard Errors For the measures directly implemented in the package, standard errors can also be computed. For non-integrated measures, or measures that are only requested for a single time-point, the standard error can be approximated with

$$se(L) = \frac{sd(L)}{\sqrt{N}} \tag{6.4.4}$$

where $L = L_1, ..., L_N$ is the loss-vector over all N observations and $sd(L)$ is the sample standard deviation of L.

However for measures that are integrated over time, this approximation cannot be used as the loss between time-points is not-independent and the above estimator would therefore be 'over-confident'. Instead the standard error is approximated with

$$\sqrt{\frac{\sum_{i=1}^{M}\sum_{j=1}^{M}\Sigma_{i,j}}{NM^2}} \tag{6.4.5}$$

where $\Sigma_{i,j}, i = 1, ..., M, j = 1, ..., M$ is the sample covariance matrix over the M time-points.

Note that this approximation is suitable for losses generated by discrete survival models, however it may be less accurate for continuous models. Future updates will see this improved.

Standard errors of measures can be requested with the `se` argument:

```
> m = msr("surv.graf")
> se = msr("surv.graf", se = TRUE)
> learn$score(c(m, se))
   surv.graf surv.graf_se
  0.04567402   0.01024113
```

Future updates will simplify this process by returning both the score and its standard error together when `se = TRUE`, and not requiring two separate objects.

6.4.4.4. Compositors and Pipelines

Almost all (not (R4)) the compositors and reduction workflows discussed in chapter 5 are implemented in **mlr3proba** with `PipeOp` and `Graph` objects from **mlr3pipelines**. Full details and tutorials for **mlr3pipelines** are provided in the package website[1].

Of particular note are the `distrcompositor` (section 5.4.1) and the `crankcompositor` (section 5.4.3). These compositors are implemented as `PipeOp` objects from **mlr3pipelines** however pipeline implementations are also provided for simpler construction. The example below demonstrates manually creating a composition pipeline, and then making use of the `ppl` pipeline constructor. Whilst the first method is more coding-heavy, it is required for manual creation of complex graphs.

```
> library(mlr3); library(mlr3learners); library(mlr3pipelines)
+
# manual creation
> Graph$new()$
+  add_pipeop(po("compose_distr", form = "ph"))$
+  add_pipeop(po("learner", lrn("surv.kaplan")))$
+  add_pipeop(po("learner", lrn("surv.glmnet")))$
+  add_edge("surv.kaplan", "compose_distr", dst_channel = "base")$
+  add_edge("surv.glmnet", "compose_distr", dst_channel = "pred")
# ppl pipeline
> ppl("distrcompositor", lrn("surv.glmnet"), form = "ph",
+  estimator = "kaplan")
```

With either method, the resultant learner is now capable of predicting a distribution.

```
> task = tgen("simsurv")$generate(20)
> learn = ppl("distrcompositor", lrn("surv.glmnet"), form = "ph",
+  estimator = "kaplan")
> learn$train(task)
> learn$predict(task)[[1]]$distr
WeightDisc1 WeightDisc2 ... WeightDisc19 WeightDisc20
```

The example below demonstrates the `crankcompositor` which allows composition from a `distr` prediction to a `crank` or `response` prediction, which is particularly important if predicting the deterministic time until event.

[1] https://mlr3pipelines.mlr-org.com/articles/introduction.html

```
> learn = ppl("crankcompositor", lrn("surv.kaplan"), response = TRUE)
> learn$train(tgen("simsurv")$generate(1000))
> p = learn$predict(tgen("simsurv")$generate(100))[[1]]
> p$score(c(msr("surv.rmse"), msr("surv.rmse", se = TRUE)))
   surv.rmse surv.rmse_se
  1.65823788   0.08313664
```

6.4.5. Conclusion and Availability

mlr3proba extends the **mlr3** family of packages to survival analysis, with a long-term goal of being a complete probabilistic supervised learning toolbox that includes density estimation and probabilistic regression. Future updates will therefore be primarily focused on finalising the designs of the density and probabilistic regression tasks, before implementing respective learners and measures.

mlr3proba is released under an LGPL-3 licence on GitHub and CRAN. Extended documentation, tutorials, and examples are available on the project website[1]. Extra survival learners are available from **mlr3learners** and **mlr3extralearners**. Code quality is monitored and maintained by an extensive suite of unit tests with GitHub Actions on multiple operating systems.

6.5. Conclusions

This chapter presented R packages that pull together all the work in this thesis in order to improve accessibility, transparency, and performance in survival modelling.

mlr3proba implements many of the models and measures discussed in chapters 3 and 4, and almost all the compositions and reductions described in chapter 5. The survival analysis framework in **mlr3proba** is stable and has many implemented learners, measures, and pipelines. The density estimation framework is still maturing and whilst many models have been implemented, future work will add further measures as well as updating the prediction objects. Probabilistic regression is currently in development with the current framework limited to interfacing deterministic regression models and transforming them to probabilistic predictions via compositors. Future work will include adding functionality for interfacing Bayesian simulation packages (e.g. JAGS and Stan), implementing analytical Bayesian models (e.g. Bayesian linear regression), and extending the task to interval regression. Designs for this will first look at the work of **skpro** [111].

Software development is never complete and all three packages will require constant maintenance and optimisation. **distr6** optimisation will focus on improving numerical results as well as run-time performance. Its current strengths lie in a C++ implementation of many distributions, though this does not extend to composite distributions and future updates will prioritise these shortcomings. **set6**

[1] https://mlr3proba.mlr-org.com/

is an important dependency of **distr6** and **mlr3proba** and as such will continue to be a vital part of future developments. The various benchmark experiments in section 6.2 demonstrate that whilst **set6** outperforms **sets** in construction, it is slower in operations, and is significantly slower than **base** by several orders of magnitude. **set6** will be optimized to prevent 'bottlenecks' in downstream dependencies.

The power and performance of these three packages is demonstrated in the next chapter, in which **mlr3proba** is utilised to run a large-scale benchmark experiment that had previously not been possible (or at least too difficult with smaller packages).

Chapter 7

A Benchmark Experiment of Survival Models

This thesis culminates in two benchmark experiments comparing the performance of a large number of classical and machine learning survival models (chapter 3) across a comprehensive range of measures (chapter 4) using **mlr3proba** (chapter 6). Whilst benchmark experiments for survival analysis have been undertaken before, there has been a well-documented lack of large-scale experiments covering a large range of survival models (section 7.1.2) on many datasets. Here 'large-scale' refers to a large number of models, datasets, and measures. This chapter includes two experiments. The first studies survival models on 30 real-world datasets in order to make generalisations about performance of models on real-world data in the right-censoring setting. The second investigates survival model performance on 36 simulated datasets in order to make generalisations about model performance given different outcome conditions. The design of the experiment is first described in section 7.2 and then results are presented (section 7.3) and discussed (section 7.4).

7.1. Introduction

This chapter introduces what is believed to be (section 7.1.2) the first large-scale benchmark study for right-censored, non-competing risks survival analysis. The term 'large-scale' refers to the comprehensive nature of the experiment, which includes 30 real-world datasets and 36 simulated datasets, thus allowing more robust conclusions about model performance than previous studies (section 7.1.2). This chapter will use the term 'large-scale' to refer to experiments that include at least 15 datasets, 5 models, and 2 measures. The primary motivation to determine if a study is 'large-scale' is in ensuring that conclusions can be generalised to datasets that are representative of those included in the study [123].

As this is the first experiment of its kind, not only does it provide a comprehensive overview of the performance of different survival models, but it also provides researchers with easy access to survival datasets for future studies and sets a performance baseline for future experiments.

Notation and Terminology Relevant notation introduced in chapter 2 is recapped for use in this chapter: the generative template for the survival setting is given by (X, T, Δ, Y, C) t.v.i. $\mathcal{X} \times \mathcal{T} \times \{0, 1\} \times \mathcal{T} \times \mathcal{T}$ where $\mathcal{X} \subseteq \mathbb{R}^p$ and $\mathcal{T} \subseteq \mathbb{R}_{\geq 0}$, where C, Y are unobservable, $T := \min\{Y, C\}$, and $\Delta = \mathbb{I}(Y = T)$. Random survival data is given by $(X_i, T_i, \Delta_i, Y_i, C_i) \overset{i.i.d.}{\sim} (X, T, \Delta, Y, C)$.

7.1.1. Research Questions

This study has two primary research questions:

RQ1) What survival model or models perform best on right-censored time-to-event datasets without competing risks?

RQ2) How does performance of survival models vary with different simulated time-to-event dataset conditions including censoring proportion, underlying survival time distributions, and type of censoring?

Determining if one model is 'better' than another for (RQ1) is answered by generalising model performance over 30 representative real-world datasets. Similarly for (RQ2) model performance is generalised over 36 simulated datasets covering a range of values in the conditions of interest. The conditions of interest for (RQ2) were determined from reviewing the literature presented throughout this thesis, in particular papers that included simulation experiments, and identifying which criteria were most frequently discussed with respect to model performance.

As well as answering these questions, this study provides future researchers with a public repository that contains the code for selecting and tuning models,[1] as well as loading the real-world datasets and simulating the others. Finally, successfully answering these questions additionally demonstrates that large-scale survival experiments can be run off-shelf with open-source software, which was not previously possible.

7.1.2. Literature Review

The experiments described in this paper provide a neutral comparison (section 7.2.1) of both classical and machine learning survival models. This experiment is called 'large-scale' due to the number and range of datasets (30 real and 36 simulated), models (24) and measures (7); it is believed this is the first experiment of its kind that includes a comprehensive range of these components and the literature review below supports this claim.

Historically, surveys, reviews, and analytical comparisons of survival models can be grouped into: i) comparison of models with limited scope (section 7.1.2.1); and ii) qualitative surveys without benchmark experiments (section 7.1.2.2).

7.1.2.1. Comparison of Survival Models

Papers that compare survival models are further separated into comparisons: i) of multiple ML and classical model classes; ii) on high-dimensional data; iii) of

[1] https://github.com/RaphaelS1/thesis_supplementary

'classical' models only; and iv) of one novel model (or class) to a single CPH or AFT baseline. Whilst no specific number of features can be provided to determine if a dataset is high-dimensional, this review conservatively terms a dataset 'high-dimensional' if the number of features exceeds the number of observations.

Comparisons of ML and Classical Models The experiments run in this chapter fall into this category. Only one prior experiment could be found that benchmarked more than one ML model class on low-dimensional data. Though the scope is still limited to a few model classes and only three datasets.

Kattan (2003) [154] benchmarked tree-based models, ANNs and Cox models with Harrell's C-index across three datasets with varying censoring proportions. The models are compared for significant differences by repeating the experiments many (up to 50) times with different seeds thus allowing for different hyper-parameter configurations and folds in cross-validation. Boxplots across all replications indicate no machine learning model outperformed the Cox PH.

Comparisons on High-Dimensional Data Herrmann *et al.* (2020) [123] performed a large-scale benchmark experiment of survival models on multi-omics high-dimensional (60,000+ variables) data. Models fall into the following groups: penalized regression; boosting; and random forests. Comparisons are made with Uno's C and the IGS. The IGS for all models overlapped with the Kaplan-Meier baseline though all C indices were significantly higher than the baseline – however it is not stated how the standard errors for the confidence intervals were derived and nor is it stated in the paper if multiple testing correction is applied.

Spooner *et al.* (2020) [283] also compared machine learning models on high-dimensional data. In this study GBMs, RSFs, Cox PH, and some extensions to the Cox PH were compared. Models were evaluated by Harrell's C only. The results indicated that all models outperformed the Cox PH when no additional feature selection was used but that there were no significant differences when feature selection was applied to the Cox model. There were few significant statistical differences between models.

Comparisons of Classical Models Moghimi-dehkordi *et al.* (2008) [218] compare Cox PH to parametric survival models with various distributions. Out-of-sample measures for comparison are not provided though the AIC produced by the Cox PH is far higher (and therefore inferior) than those of the parametric models. Model inspection demonstrated that all models provided similar (non-significantly different) confidence intervals for hazard ratios.

Georgousopoulou *et al.* (2015) [97] compared the Cox PH to a Weibull and Exponential parametric model. Again no out-of-sample measures were utilised, models were compared by the Cox-Snell residuals and the BIC. Similarly to Moghimi-dehkordi *et al.*, hazard ratios produced from all three models were nearly identical. The authors claim the Cox PH is inferior to the parametric models though only graphical comparisons are included.

Zare *et al.* (2015) [330] provide another comparison of the Cox PH to AFT models, utilising Cox-Snell residuals and AIC as their measures of comparison. Similarly to the previous studies the Cox model has the highest AIC though

the plotted Cox-Snell residuals are very similar. The authors acknowledge no significant differences between the model classes and instead conclude that AFT is a useful and more interpretable alternative. No significant differences were found between the different AFT parameterisations.

Dirick *et al.* (2017) [70] make use of a financial setting to compare the Cox PH, AFT, flexible Cox models with splines modelling the hazard, and mixture cure models. The authors compare the models using a time-dependent AUC, the MSE, and the MAE. Survival times are generated from the Cox PH with a deterministic composition using quantiles chosen to minimise the MSE and MAE, it is not clear if this is performed in an unbiased nested resampling manner or after predictions are made. By averaging the ranking of model performance the authors conclude that Cox PH with penalized splines outperformed the other models with respect to the chosen metrics.

Habibi *et al.* (2018) [112] performed another experiment on PH and AFT models. Models were again compared exclusively by the AIC with the PH having the highest result and log-normal AFT the lowest; differences between AFT models were non-significant. Confidence intervals for hazard ratios were similar (non-significantly different) for all models.

Comparisons of a Novel Model Class Luxhoj and Shyur (1997) [204] benchmarked neural networks against the Cox PH in the engineering field of reliability analysis. The baseline hazard of the Cox model is modelled by splines with a single knot. The models are compared using the mean squared error on a validation set of sample size 40, of these 40 there are only 9 unique failure times that are used for model testing. Insufficient information is provided to determine the architecture or training procedure of the neural networks compared. This review suffers from (RM2) (section 5.3.3) which here refers to the Cox model being used for time-to-event predictions without indicating how these predictions were composed. Moreover the MSE difference between the Cox and ANN was 0.003; on a utilised test set of only nine observations, this is highly unlikely to be a significant difference.

Ohno-Machado (1997) [233] also compared the Cox PH to ANNs. Several Cox models were fit with automated variable selection by backwards elimination. For each, survival curves were predicted and for a given patient they were considered dead at a particular time point if the predicted survival curve at the time is less than the 'arbitrary' [233] probability of 0.5. The Cox models were compared to a single hidden layer neural network, fit with backpropagation. This model made probabilistic predictions of death in four time-intervals that were within the predicted time of the Cox models. The probabilistic predictions from both models were compared with the AUC and its corresponding ROC. No significant differences in performance were found between the two models.

Puddu and Menotti (2012) [241] again compared the Cox PH to ANNs however in this setting the outcome was 45-year all-cause mortality, which is a classification outcome. Therefore this also suffers from (RM2) (section 5.3.3) as a classification neural network is being compared to a survival model on a classification task. Despite this, the authors found no statistical difference between the compared models.

Goli *et al.* (2016) [102] provide a comprehensive comparison of support vec-

tor machine models with the Cox PH as a reference class (Kaplan-Meier is not included). Models are compared against the C-index and log-rank test, though it is unstated which C-index is utilised. No model outperformed the Cox PH with respect to the chosen C-index.

7.1.2.2. Surveys of Survival Models

The final class of papers do not perform analytical benchmark experiments but instead survey/review available survival models (including ML). These are therefore only discussed very briefly.

Ohno-Machado (2001) [231] provide an overview to models available for survival analysis from non-parametric estimators and classical models to neural networks. The review highlights useful applications of the models and their respective limitations. In particular their Table 1 clearly states advantages and disadvantages of Cox models versus ANNs.

Patel *et al.* (2006) [234] compare proportional hazards and accelerated failure time models. This comparison is primarily theoretical and based on model properties, no analytical comparison with measures is provided though comparisons of predicted median survival times are compared to those from a Kaplan-Meier estimator. The authors conclude that AFT models should be considered more often due to simpler interpretation.

Wang *et al.* (2017) [317] provide a review of survival analysis models and measures that is strongly recommended here as a precise and comprehensive introduction to the field of machine learning in survival analysis. The authors provide strong arguments for comparing classical models against one another, though this is not extended to the machine learning setting. More mathematical detail is provided to the classical setting however a clear and detailed overview is still provided for machine learning models. Some attention is also given to the more complex cases of competing risks and multiple events.

Lee and Lim (2019) [193] provide a short but concise overview to survival analysis models with an emphasis on genetic data and implementation in R. Their review covers classical models, penalization, and many machine learning models. They provide a clear, practical illustration (but not full benchmark experiment) comparing the models on a real dataset against Harrell's C. No model outperforms the Cox PH.

7.2. Benchmark Experiments

7.2.1. Study Design

The experiments in this study are designed to assess the status quo of survival models, both machine learning and classical (chapter 3). In order to achieve this objective, this study aims to be a 'neutral comparison study' [28]. Following the guidelines put forward by Boulesteix *et al.*:

Focus on model comparison The focus of this study is on model comparison and not examining a novel model. Therefore the novel models suggested in this

thesis (chapter 3), as well as the reductions and compositions (chapter 5) are not included in these experiments.

The author is neutral The author has no involvement in the development of any of the compared models, with the exception of providing an R interface for the ANNs. This interface does not affect neutrality as the underlying implementation is independent of the author and there is no personal gain in these models being found superior (no papers published for the **survivalmodels** [275] package in which these models are implemented).

Measures, methods and data are chosen in a rational way Details of chosen methods, models, and evaluation measures are provided in full detail in the next three sub-sections. The study is designed to assess the status quo, which does not include novel models, methods, or measures. Therefore, common measures of evaluation are utilised, despite the flaws identified in chapter 4; this allows comparison to previous studies. Given these flaws, extra care is taken in drawing conclusions and a full range of evaluation measures is provided. Limitations in the experiments (section 7.4.3), which are primarily due to resource limitations, mean that methods such as tuning, are constrained by computational power. This still results in a 'rational' selection of methods as the study mimics real-world settings that have similar constraints on computing power. The inclusion criteria for real datasets were: datasets in R that include at least two features, 100 observations, a right-censoring indicator, and a survival time. The first 30 datasets to select this criteria, that were found by online searches, were selected for analysis. No quota was specified to obtain a certain number of datasets with different censoring proportions, instead the random search was assumed to yield enough variation in proportion of censoring and this was confirmed once all datasets were selected. Careful selection of these datasets allows conclusions to be drawn on (RQ1) (section 7.1.1), namely generalisation of model performance to right-censored time-to-event datasets. For the 36 simulated datasets, a group of conditions was identified by reviewing previous studies and real-world datasets. The range of simulations allows conclusions to be drawn on how models may perform on different right-censored time-to-event datasets.

The present limitations (section 7.4.3) mean that this study serves best as a pilot study. Conclusions are limited to understanding how these models compare in a real-world setting limited by computational resources; therefore models that do not require extensive tuning may be expected to perform better.

7.2.2. Implementation and Accessibility

Platform All experiments were conducted on UCL's 'Myriad' high-performance cluster [221] with multicore parallelisation via **future** v1.20.1 [14].

Reproducibility and Accessibility For generation of simulated data and running of experiments, L'Ecuyer's random number generator [190] was utilised to ensure reproducibility of parallelised code. All code required to run the exper-

iments, as well as the results, are freely available in a public GitHub repository[1]. Software, packages, and version numbers that were utilised to run the code are listed below.

Packages All code is written and implemented in R v4.0.2 [245]. Models and measures are implemented in **mlr3proba** v0.2.4.9000 [281], tuning is implemented in **mlr3tuning** v0.2.0 [180], benchmark functionality is implemented in **mlr3** v0.8.0.9000 [182], and compositions are implemented through **mlr3pipelines** v0.2.1.9000 [21]. Helper functions are made use of from **mlr3misc** v0.5.0 [179], **paradox** v0.5.0.9000 [181], **checkmate** v2.0.0 [178], and **survival** v3.2.3 [291]. Python datasets and models are interfaced via **reticulate** v1.16 [301]. Simulations utilise **distr6** v1.4.7 [277]. The packages and versions for the tested models are given in table 21, all measures are directly implemented in **mlr3proba**.

7.2.3. Methods

Subsampling In order for the models to be successfully trained, very large datasets had to be subsampled. After prior exploration, any dataset with over 4000 observations was sampled uniformly at random without replacement to a maximum of 4000 observations. The cluster required all jobs to run within 48 hours, in order to achieve this jobs were first split by dataset and then if these were not able to complete in time, then by model class. When experiments by model class took longer than 48 hours then the dataset was reduced by 500 observations at a time until the job could finish within the allotted period. Subsampling the dataset is considered to be part of a model's pipeline (and therefore related to model performance) as the requirement to subsample is a disadvantage of some of the more complex models (classical models never required further subsampling).

Resampling For the real datasets, K-fold cross-validation (section 2.4.1) is performed with five folds except when this leads to folds of less than 30 observations, in this case the number of folds is decreased until a minimum number of 30 observations is reached for each fold. For simulated datasets, models are trained on 1,000 draws from the chosen data generating process and subsequently tested on another 1,000 draws from the same process.

Tuning For learners with 60 or less possible configurations, tuning (section 2.4.1) is performed over all possible combinations by grid search; for learners with more than 60 configurations, random search is employed with 60 iterations [16]. In both experiments, three fold nested cross-validation is performed for model tuning.

Prediction Types **mlr3proba** compositors are utilised in order for models to be compared on their predictive ability for deterministic, ranking, and probabilistic prediction types. Where models only predict a probabilistic prediction, the `response` and `crank` prediction types are estimated as the mean and negative mean of the predicted distribution (section 5.4.3) respectively. When models predict only `crank` or `lp` then the prediction distribution is composed with a

[1] https://github.com/RaphaelS1/thesis_supplementary

Kaplan-Meier baseline and a PH model form, chosen as this is a common model assumption (section 5.4.1).

Pre-Processing Pre-processing is applied only when required by models. This may include standardization of covariates to unit variance and zero mean, and/or treatment encoding via the `model.matrix` function. Pre-processing data is considered to be part of the model's pipeline (and therefore related to model performance). Pre-processing is performed with **mlr3pipelines**. Appendix G lists the pre-processing requirements of all models.

***p*-values** To avoid '*p*-hacking' and reporting of erroneous significant results, multiple-testing correction is applied twice to all test results. In the first instance, Benjamini-Hochberg correction is applied to all post-hoc methods that do not already correct for multiple-testing. Then after running and reporting the results of all tests, a conservative Bonferonni correction is applied by multiplying all reported *p*-values by the total number of reported tests (i.e. tests relating directly to the results of interest). After correction, a significant result is taken to be one in which (adjusted) $p \leq 0.05$.

7.2.4. Models and Configurations

The models compared in this experiment were chosen as a result of the literature review and survey in chapter 3. Table 21 lists all the compared models, along with their software, version numbers, and prediction types. The horizontal lines separate the models into different groups (chapter 3): classical, random survival forests (RSFs), gradient boosting machines (GBMs), support vector machines (SVMs), and artificial neural networks (ANNs).

The Kaplan-Meier and Nelson-Aalen models are treated as unconditional baselines (section 3.1) and the Akritas estimator as a conditional baseline. Note that in general the Kaplan-Meier and Nelson-Aalen will give very similar (if not identical) results and the Akritas estimator may also perform similarly (section 3.1). Appendix G lists the models' hyper-parameter and pre-processing configurations.

7.2.5. Performance Evaluation

Evaluation Measures The chosen range of measures are those suggested in section 4.7, which are summarised in table 22. Models are optimised with respect to Harrell's C. Houwelingen's α, which can be arbitrarily large, is upper-truncated at 10 (no primary conclusions are drawn from this measure).

Evaluation of Real Data Experiments Following Demšar [67], Friedman rank sum tests are initially performed for all measures, where the 'groups' are the models and the 'blocks' are the (independent) datasets. The corresponding post-hoc (multiple-testing corrected) Friedman-Nemenyi tests are conducted as critical difference diagrams if the global tests are significant. Post-hoc tests are restricted only to Uno's C and IGS; other results are reported to support findings but are not

Table 21: Models for testing with packages and versions.

Model Name[1]	**Learner**[2]	**Package**[3]	**distr**[4]	**crank**[5]	**lp**[6]	**response**[7]
Kaplan Meier (KM)	kaplan^P	survival v3.2.3	✓	mean	×	mean
Nelson Aalen (Nel)	nelson	survival v3.2.3	✓	mean	×	mean
Akritas Estimator (AE)	akritas	survivalmodels v0.1.0	✓	mean	×	mean
Cox PH (CPH)	coxph^P	survival v3.2.3	✓	η	✓	mean
CV Regularized Cox PH (GLM)	cv_glmnet^L	glmnet v4.0.2	PH	η	✓	mean
Penalized (Pen)	penalized	penalized v0.9.51	✓	mean	×	mean
Parametric (Par)	parametric	survival v3.2.3	✓	η	✓	mean
Flexible Splines (Flex)	flexible	flexsurv v1.1.1	✓	η	✓	mean
RSDF-STAT (RFB/RFL)**	rfsrc	randomForestSRC v2.9.3	✓	mean	×	mean
RSDF-STAT (RFC)	ranger^L	ranger v0.12.1	✓	mean	×	mean
RSCIFF (RFCIF)	cforest	partykit v1.2.9	✓	mean	×	mean
RRT*	rpart^P	rpart v4.1.15	✓	mean	×	mean
GBM*** (GBC/GBG/GBU)	mboost	mboost v.2.9.3	✓	η	✓	mean
CoxBoost (COXB)	cv_coxboost	CoxBoost v1.4	✓	η	✓	mean
SSVM-Hybrid (SVM)	svm	survivalsvm v0.0.5	PH	✓	×	✓
Cox-Time (CoxT)	coxtime	survivalmodels v0.1.0	✓	mean	×	mean
DeepHit (DH)	deephit	survivalmodels v0.1.0	-✓	mean	×	mean
DeepSurv (DS)	deepsurv	survivalmodels v0.1.0	✓	mean	×	mean

Continued on next page...

Table 21: (continued)

Model Name[1]	Learner[2]	Package[3]	distr[4]	crank[5]	lp[6]	response[7]
Nnet-Survival (LH)	loghaz	survivalmodels v0.1.0	✓	mean	×	mean
PC-Hazard (PCH)	pchazard	survivalmodels v0.1.0	✓	mean	×	mean
DNNSurv (DNN)	dnn	survivalmodels v0.1.0	✓	mean	×	mean

1. Name of algorithm given in chapter 3. Horizontal lines separate model classes: classical, RSFs, GBMs, SSVMs, ANNs. Model abbreviations in parentheses are used in results. Abbreviations in parantheses are used in plots but the unique model ID from chapter 3 is used in text.

2. Learner ID in **mlr3**. All learners are implemented in **mlr3extralearners** v0.1.1.9000 unless indicated with a P for **mlr3proba** v0.2.4.9000 or an L for **mlr3learners** v0.4.2.9000.

3. Package in which the learner is implemented with version used in experiment.

4. `distr` predict type in **mlr3proba** is the probabilistic prediction. A check (✓) represents the distribution being predicted directly by the package. 'PH' represents distribution composition with PH form and Kaplan-Meier baseline.

5. `crank` predict type in **mlr3proba** is the continuous ranking prediction. A check (✓) represents the ranking being predicted directly by the package. 'η' represents the ranking being identical to the predicted linear predictor. Finally 'mean' represents probabilistic to deterministic composition with the distribution mean.

6. `lp` predict type in **mlr3proba** is the linear predictor prediction. A check (✓) represents the linear predictor being predicted directly by the package whereas a cross (×) means the prediction is not available (and cannot be composed).

7. `response` predict type in **mlr3proba** is the deterministic survival time prediction. A check (✓) represents the survival time being predicted directly by the package. 'mean' represents probabilistic to deterministic composition with the distribution mean.

$*$ – Ideally this would be bagged to Relative Risk Forests but current implementation does not allow for this.

$**$ – The RSDF-STAT is included twice once for a Brier splitting rule and one for log-rank, Both utilise the `rfsrc` learner (with different hyper-parameters).

$***$ – The GBM model is included three times separately as GBM-GEH (GBG), GBM-UNO (GBU), and GBM-COX (GBC).

used to form conclusions. All analysis is conducted with **mlr3benchmark** [279] v0.1.0.

Evaluation of Simulated Data Experiments As datasets are non-independent, analysis is performed with ANOVA tests where the dependent variables are the measures and the blocks are the conditions of interest. The initial analysis fits two ANOVA models, one for Uno's C and one for IGS, to test the different conditions across all models and measures. Post-hoc Tukey HSD's are run on significant effects and interactions. A secondary analysis is conducted regressing one of the two measures on the model type (not model group), this is more exploratory and does not inform the final results (the real experiments are for drawing conclusions about overall model performance). Results for other measures are reported in the appendices.

Table 22: Measures for evaluating models in benchmark experiments.

Measure	Type[1]	Evaluates[2]
van Houwelingen's α	Calibration	Ranking
Harrell's C	Discrimination	Ranking
Uno's C	Discrimination	Ranking
Integrated Graf Score	Scoring Rule	Distribution
Integrated Log Loss	Scoring Rule	Distribution
MAE	Distance	Survival Time
RMSE	Distance	Survival Time

1. Type of measure.
2. Which survival prediction type the measure evaluates.

7.2.6. Datasets

In total 66 datasets are used in these experiments, 30 real datasets and 36 simulated ones. These numbers are chosen in order to provide generalisations to future datasets via asymptotic approximations. The real datasets reflect real-life scenarios and allow conclusions to be drawn about the models compared to one another, for example to determine which models perform well or poorly on right-censored datasets. The simulated datasets allow the models to be tested against different conditions that they may experience in reality, including different censoring proportions and types, and different survival distributions. As this thesis is focused on right-censoring, all datasets are either simulated, collected, or modified to reflect this. Similarly, this paper is not concerned with a model's abilities to handle or impute missing covariate data, hence any missing data is removed from real-world datasets. This could potentially introduce bias if missingness is dependent on the feature or outcome, however as the end-goal is model comparison and not interpretability, this should not affect the conclusions.

7.2.6.1. Real Datasets

Real datasets are taken from packages in R and Python. All datasets model the time until an event (not necessarily guaranteed) takes place with possible right-

censoring. Datasets were chosen so that each has a minimum of 100 observations and two covariates. High-level summaries of the datasets in terms of number of observations and covariates (after any modification) are given in table 23 along with citations detailing the complete dataset details. Minor changes are made to variable names, recoding of factor levels, and deletion of non-informative covariates (e.g. id)[1]. Complete case analysis is performed to remove observations with any missing data. Observations are deleted if their event time is equal to zero (this was very rare). A few datasets had their outcome altered, these include:

- `diabetic` – Original: Visual loss in both eyes. Modified: Visual loss in the left eye only. This prevents non i.i.d. observations in the dataset.
- `pbc` – Original: Death, transplant, or censored. Modified: Death or censored (including informative censoring due to transplant).
- `transplant` – Original: Death, transplant, censored, withdrawn. Modified: Death or censored (including informative censoring due to transplant or withdrawal).
- `melanoma` – Original: Death from melanoma, death from other causes, censored. Modified: Death by melanoma or censored (including informative censoring due to death by other causes).
- `prostateSurvival` – Original: Death from prostate cancer, death from other causes, or censored. Modified: Death by prostate cancer or censored (including informative censoring due to death by other causes).

7.2.6.2. Simulated Datasets

Simulated datasets follow closely the recommendations of Burton *et al.* [40]. Datasets are defined in order to draw meaningful conclusions from models by directly comparing the conditions that define the simulated datasets. 36 different generating procedures are defined (appendix E), one for each dataset, from all combinations of

i) Survival distribution: Cox-Weibull, Weibull, Gompertz, Log-normal.

ii) Censoring type: Type I, informative right, uninformative.

iii) Censoring proportion: 20%, 50%, 80%.

The covariates for each dataset are drawn from the same generating process, the only differences are in the outcome data. The datasets are primarily split by survival distribution, each block of datasets with the same survival distribution have identical (in value) covariates and survival times, Y, only the censoring mechanism, C is different, and therefore by extension (T, Δ). Datasets with different survival distributions generate new covariates and survival times. For each dataset, 2,000 observations are drawn and split for training and testing (50/50). Details for generating features, X, and survival outcomes (T, Δ) are provided below. Each simulated dataset is generated only once and not repeated (section 7.4.3).

[1] https://github.com/RaphaelS1/thesis_supplementary/blob/main/code/c7_bench/real_jobs/create_survival_data.R

Table 23: Real-world datasets used in benchmark experiment.

Dataset[1]	**Cens %**[2]	n_C^3	n_D^4	**n**[5]	**p**[6]	**Package**[7]
Aids2 [222]	38	1	3	2814	4	**MASS** [222]
ALL [133]	63	0	4	2279	4	**dynpred** [242]
bmt [164]	41	4	13	137	17	**KMsurv** [165]
channing [164]	62	1	1	458	2	**KMsurv**
diabetic [290]	65	1	3	197	4	**survival** [291]
flchain [71]	72	4	3	7871	7	**survival**
gbsg [156]	43	3	4	2232	7	**pycox** [172]
grace [126]	68	4	2	1000	6	**mlr3proba** [281]
hepatoCellular [194]	55	30	12	101	42	**asaur** [77]
kidtran [164]	84	1	3	863	4	**KMsurv**
lung [200]	28	5	3	167	8	**survival**
melanoma [7]	72	2	3	205	5	**MASS**
metabric [156]	42	5	4	1903	9	**pycox**
mgus [175]	6	6	1	176	7	**survival**
nafld1 [6]	92	4	1	12588	5	**survival**
nki [307]	73	5	5	295	10	**dynpred**
nwtco [33]	86	1	2	4028	3	**survival**
ova [310]	26	1	4	358	5	**dynpred**
patient [12]	79	2	5	1985	7	**pammtools** [11]
pbc [294]	60	10	7	276	17	**survival**
pharmacoSmoking [287]	29	4	5	113	9	**asaur**
prostateSurvival [201]	94	0	3	14065	3	**asaur**
rats [207]	86	0	3	300	3	**survival**
support [156]	32	10	4	8873	14	**pycox**
transplant [160]	92	2	2	793	6	**survival**
tumor [11]	52	1	6	776	7	**pammtools**
udca1 [199]	57	2	2	169	4	**survival**
veteran [151]	7	3	3	137	6	**survival**
wbc1 [289]	43	2	0	190	4	**dynpred**
whas [126]	48	3	6	481	9	**mlr3proba**

1. Dataset ID and citation.
2. Proportion of censoring in the (modified) dataset.
3-4. Number of continuous and discrete features respectively before recoding.
5-6. Total number of observations and features respectively after alterations described above but before sub-sampling.

Features All simulated datasets use the same feature generation process and furthermore datasets with the same survival distribution use identical feature values, i.e. only one draw is made for each of the nine datasets with the same survival distribution for a total of four draws. This ensures that conclusions can be drawn about the outcome data as the feature data is held constant.

To mimic real-world data, the simulated data includes three variables with a real-world interpretation that are jointly i.i.d., and 10 variables with random dependence on each other; the 'real' features are mutually independent of the other 10 features. The variables include two discrete features, X_{sex}, X_{trt}, one continuous feature, X_{age}, and 10 features drawn from a multivariate normal distribution, $X_{\mathbf{x}}$; these are generated as follows

$$\begin{aligned} X_{age} &\overset{i.i.d.}{\sim} \text{DiscreteUniform}[20, 50] \\ X_{sex} &\overset{i.i.d.}{\sim} \text{Bern}(0.5) \\ X_{trt} &\overset{i.i.d.}{\sim} \text{Bern}(0.7) \\ X_{\mathbf{x}} &\overset{i.i.d.}{\sim} \text{MultivariateNormal}(\mu, \Sigma) \end{aligned}$$

The distribution parameters were chosen arbitrarily. For the $X_{\mathbf{x}}$ covariates, μ is a vector of 0s and Σ is chosen such that all variables have zero correlation except, $\rho_{1,2} = 0.5, \rho_{1,3} = 0.8, \rho_{2,3} = 0.8, \rho_{8,9} = 0.4, \rho_{8,10} = 0.4, \rho_{9,10} = 0.2$, where $\rho_{i,j}$ is the sample correlation between the ith and jth variables in $X_{\mathbf{x}}$. The demographic variables are independent of each other and the $X_{\mathbf{x}}$ variables. The covariates are visualised in appendix F for a single seed.

Survival Times Four distributions are modelled for survival times: Cox-Weibull, Weibull, Gompertz, and Log-normal. The distribution choices are informed by prior experiments [13, 40] and their reflection of real-world data. The Cox-Weibull [13] is a Cox PH with a Weibull baseline hazard. Inclusion of Cox-Weibull allows conclusions about how models handle PH data, Weibull is a standard distribution included in all survival experiments, Gompertz models human mortality, and Log-normal allows a non-monotonic hazard. The distribution parameterisations were found by prior exploration in order to create different hazard shapes of interest. Depending on the survival distribution condition of the dataset of interest (appendix E), the true survival times, Y, are drawn from one of

$$\begin{aligned} Y &\overset{i.i.d.}{\sim} \text{Weibull}(k = 2(X_{trt} + 1) + 1.5(X_{sex} + 1)) + 2X_{age}, \lambda = 1 + \frac{X_{\mathbf{x}}\beta}{100}) \\ Y &\overset{i.i.d.}{\sim} \text{Gompertz}(k = 2X_{age} + \frac{X_{\mathbf{x}}\beta}{100}, \lambda = 2(X_{trt} + 1) + 1.5(X_{sex} + 1)) \\ Y &\overset{i.i.d.}{\sim} \text{Lognormal}(\mu = \log(2X_{age} + X_{\mathbf{x}}\beta), \sigma = (X_{trt} + X_{sex})/2) \end{aligned} \tag{7.2.1}$$

where k is the shape parameter, λ is the scale parameter, μ is the log-mean parameter and σ is log-standard deviation. Or if the survival distribution condition

is Cox-Weibull [13], then

$$Y = \Big(- \frac{\log(U)}{5 \exp(3(X_{trt} + 3) + 2(X_{sex} + 2) + 2X_{age} + \frac{X_{\mathbf{x}}\beta}{100})} \Big)^{1/20} \tag{7.2.2}$$

where β is a vector of coefficients randomly drawn from $\mathcal{U}(-1, 1)$, and U is a draw from $\mathcal{U}(0, 1)$.

All values are mean/variance scaled after simulation with the `scale` function and then shifted by +4 and lower-truncated at 1, which allows meaningful comparison between experiments whilst still preserving distribution shapes (appendix F). These parameterisations result in survival times that are primarily dependent on the three 'real-world' covariates with random dependence on the other 10 variables.

As no clear concensus exists in the literature on how to generate survival time data, the 'usual' approach is to iterate in prior exploration to find parameters that produce the hazard and survival curves of interest and that ensure the survival time can be predicted by features [40, 50]; this is the approach taken above.

Censoring Outcome Three separate methods are chosen for generating the censoring distribution: Type I, informative right [40, 50], and random uninformative. Each censoring outcome type is repeated three times for each survival distribution so that there are datasets with $p = 20\%, 50\%$ and 80% censoring [225].

Type I censoring times follow the degenerate Delta distribution $C \sim \delta(\alpha)$ which has total mass at point α. Intuitively this means an observation experiences the event if $T \leq \alpha$ or are censored at $C = \alpha$ if $T > \alpha$. The value of α is given by $\alpha := F_Y^{-1}(p)$, where $p \in \{0.2, 0.5, 0.8\}$ and F_Y^{-1} is the inverse empirical cdf (quantile) of the survival time distribution Y.

Informative right-censoring times are identically and independently drawn six times from the same distribution of the corresponding survival distribution and the minimum value is taken for each draw. This process preserves the conditional censoring time distribution up to a linear shift. The desired proportion of censoring is then achieved by 'thinning' the resulting simulations by multiplying C by

$$B := \begin{cases} \infty, & B_p = 1 \\ 1, & \text{otherwise} \end{cases} \tag{7.2.3}$$

where B_p is an independent draw from $\text{Bern}(1 - p)$ and $p \in \{0.2, 0.5, 0.8\}$.

Uninformative censoring times are simulated by taking the minimum of six independent draws from $\text{Weibull}(10, 5)$ and then multiplying C by B (eq. (7.2.3)).

For each outcome Y is not independent of X. C is independent of X (and Y) for uninformative censoring and C is dependent on X (and conditionally independent of Y given X) in the other cases.

The outcomes for all three types are visualised in appendix F for a single seed.

7.3. Results

The results of the experiments are now presented, first for the real data experiments and then the simulated data experiments. Discussion about the results is provided in the next section (section 7.4).

7.3.1. Real Data Experiments

The full table of raw results is freely available online[1]. Global Friedman tests were significant ($p < 0.01$) for all measures.

Critical difference (CD) plots for discrimination (fig. 40) indicate that all models outperform the baselines except for DNNSurv, DeepHit, CoxTime and RRTs. The GBMs are the top-performing models however they do not significantly outperform the Cox PH. The ANNs are outperformed by almost all RSFs (except RSFCIF), GBMs (except GBM-UNO), and SSVM. These results are supported by the CD plot for Harrell's C (appendix H).

CD plots for overall performance (fig. 41) indicate that only half the models significantly outperform the unconditional baselines however only CoxBoost outperforms the conditional Akritas baseline. The Cox PH is significantly better performing than several ML methods, including most of the neural networks, the SSVM, and the RRTs. Only CoxBoost outperforms the Cox PH but again non-significantly. The non-ML penalized and flexible splines methods are also amongst the best performing.

Further visualisations for all measures are in appendix H. The best calibrated (non-baseline) models were flexible splines, RSFs with C-index split, CoxBoost, GBM-UNO, and DeepSurv.

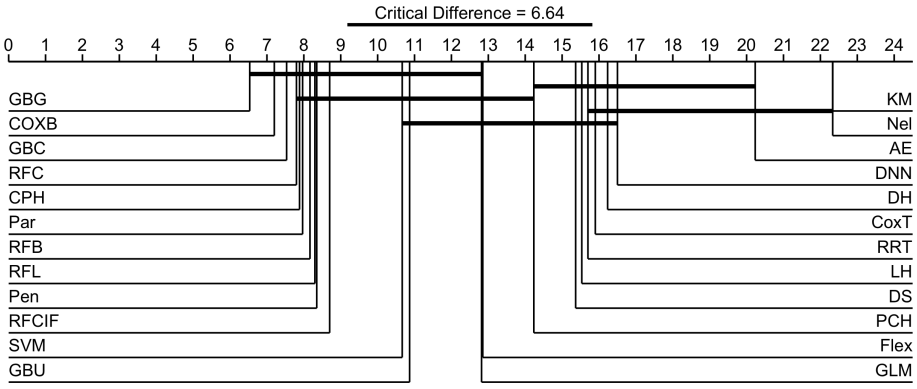


Figure 40: Critical difference plots comparing models on Uno's C. Superior plots (lower rank) are on the left with decreasing performance (higher rank) moving right. Models connected by a thick horizontal black line are not significantly different in performance.

[1] https://github.com/RaphaelS1/thesis_supplementary/blob/main/results/real_results.csv

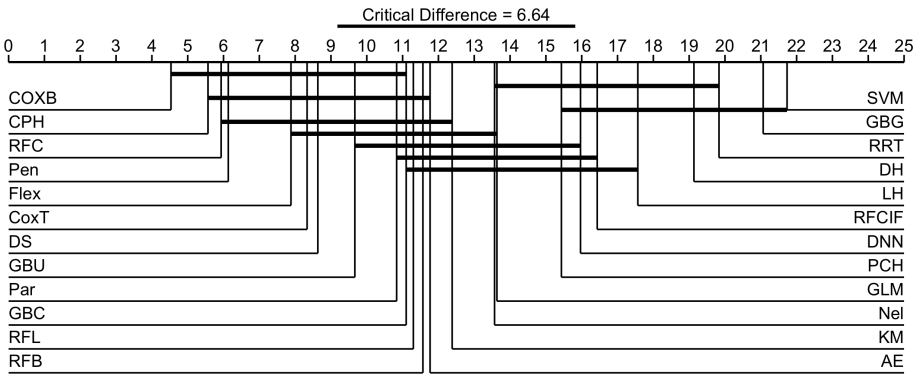


Figure 41: Critical difference plots comparing models on IGS. Superior plots (lower rank) are on the left with decreasing performance (higher rank) moving right. Models connected by a thick horizontal black line are not significantly different in performance.

7.3.2. Simulated Data Experiments

The full table of raw results is freely available online[1].

Global ANOVAs Initial ANOVAs over all datasets for Uno's C found significant effects for model group, survival distribution, and the interaction of the two; no significant effect of censoring proportion or type (all $p < 0.01$). For IGS, the same conditions were significant and additionally the censoring type (all $p \leq 0.01$).

For measuring effects of censoring type with IGS, initial analysis found outcomes with Type I censoring were 'harder' to predict than right (table 24). Once broken down by model group, significant differences between censoring types for IGS were only observed in RSFs between independent and Type I ($p < 0.01$) and in ANNs where Type I was 'harder' to predict than both other types ($p < 0.01$).

Table 24: Tukey HSD computed on different censoring types for IGS. First column is censoring types being compared, second column is their estimated difference, third and fourth are lower and upper confidence intervals for the estimate, fifth is adjusted p-value.

groups	diff	lwr	upr	p
Right − Type I	−0.05	−0.07	−0.02	< 0.01
Independent − Type I	−0.07	−0.09	−0.04	< 0.01
Independent − Right	−0.02	−0.05	0.00	0.19

Post-hoc tests on the model group for Uno's C (fig. 42 (I)) reveal all ML models significantly outperform classical models ($p < 0.04$) and GBMs significantly outperform ANNs ($p < 0.01$), all other results are non-significant. For IGS (fig. 42 (II)), SVMs are significantly worse than all other groups ($p < 0.01$) as are ANNs (with the exception of being 'better' than SVMs) ($p < 0.01$), all other results non-significant. The effects of survival distribution type are again slightly

[1] https://github.com/RaphaelS1/thesis_supplementary/blob/main/results/sim_results.csv

contradictory between Uno's C and IGS. For Uno's C (fig. 42 (III)), Gompertz and Log-normal are both significantly harder to predict than Cox-Weibull and Weibull ($p < 0.01$), and Log-normal is significantly easier to predict than Gompertz ($p < 0.01$). For IGS (fig. 42 (IV)), Weibull and Log-normal are significantly easier to predict than Cox-Weibull ($p < 0.01$), Gompertz is significantly harder than Weibull ($p < 0.01$), and Gompertz is significantly easier than Log-normal ($p < 0.01$). These are visualised as boxplots in fig. 43 combining all models.

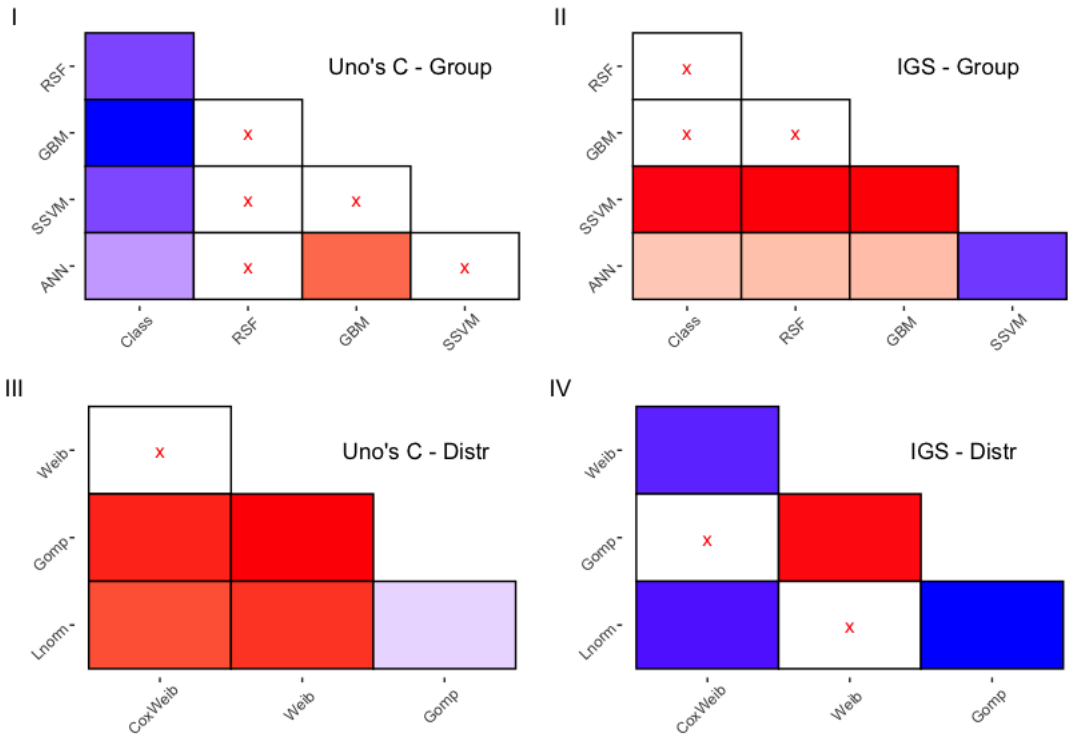


Figure 42: Tukey HSD for model group and survival distribution. Left column are results for Uno's C, right column is IGS. Top row are post-hoc tests on model group, bottom row is survival distributions. Blue squares indicate that the model on the y-axis outperforms the one on the x-axis, and the reverse for red squares. Red 'x's indicate no significant difference.

Comparison Between Survival Distributions The model group had a significant effect within all four survival distributions, and in addition the type of censoring had a significant effect within the Log-normal survival distribution datasets. All $p < 0.01$ except the effect size of the Group block within the Cox-Weibull distribution for IGS, with $p = 0.02$, and the Group block within Weibull for Uno's C, with $p = 0.01$. Post-hoc tests on the censoring type for the Log-normal distribution yield conflicting results with datasets with Type I censoring being 'easier' to predict for discrimination than right and independent censoring (both $p < 0.01$) however with respect to IGS, Type I is 'harder' to predict than the other types ($p < 0.01$). Post-hoc tests examining the group effects again reveal different results for Uno's C and IGS. For Uno's C (fig. 44, left column), all models outperform the classical models for Cox-Weibull, and GBMs outperform the classical models in all cases. In addition, ANNs outperform classical mod-

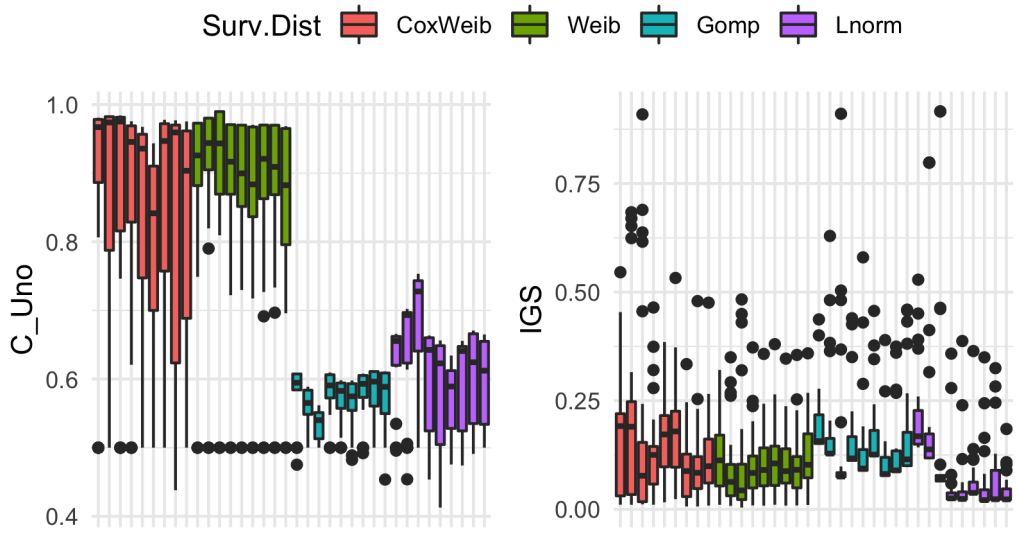


Figure 43: Survival distribution effects across all models. x-axis is the dataset (appendix E) and y-axis is boxplots of Uno's C (left) and IGS (right) for all models. Colours of boxes highlights the type of survival distribution simulated for the dataset.

els for Weibull distribution and RSFs outperform for Log-normal. The ANNs perform particularly badly for the Log-normal distribution, inferior to all other classes. For IGS (fig. 44, right column), only RSF outperforms the classical models and this is for Cox-Weibull only. In almost all cases, the SVM is inferior to all other models and in several cases the ANNs are inferior to most other models.

Comparison of Models ANOVAs regressing the model type on the measure found significant results for Uno's C and IGS ($p < 0.01$). Tukey HSD's on Uno's C found all sophisticated (non-baseline) models were significantly better than the baselines (KM, Nel, AE) however no sophisticated model was significantly better than another across all datasets, this is clearly visualised in fig. 45. More nuanced results are seen for the IGS (fig. 46), in which GLM, SVM, and LH all perform significantly worse than the majority of models. In fact, most models outperform the ANNs.

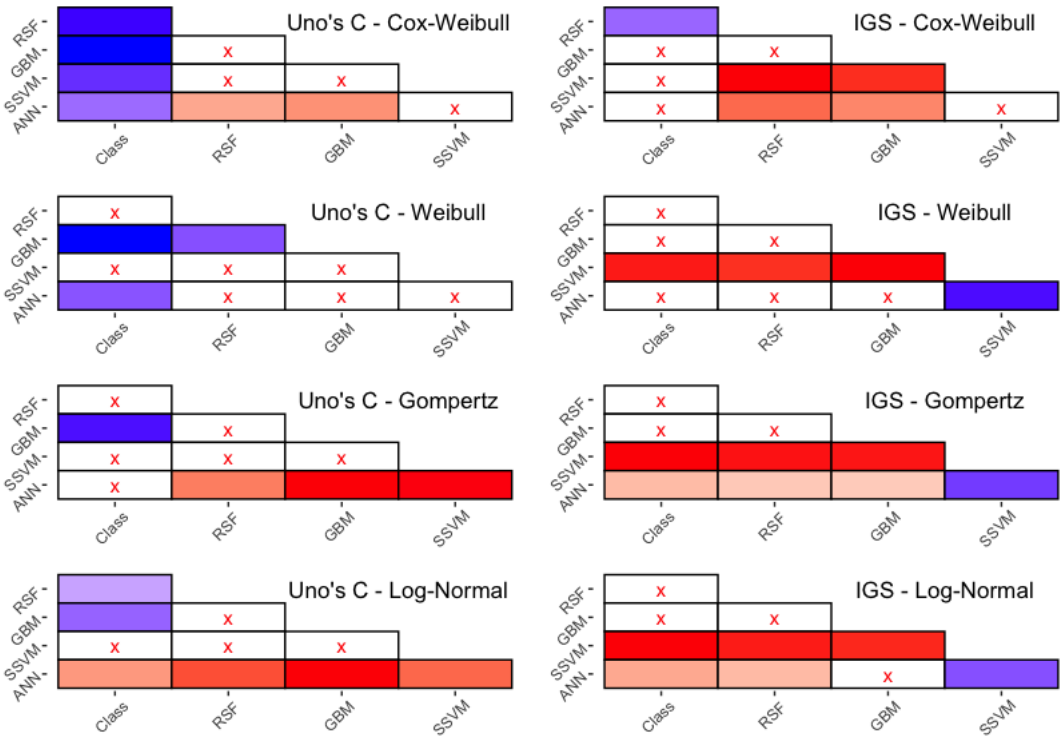


Figure 44: Tukey's HSD for within group effects by distribution. Blue squares indicate that the model on the y-axis outperforms the one on the x-axis, and the reverse for red squares. Red 'x's indicate no significant difference.

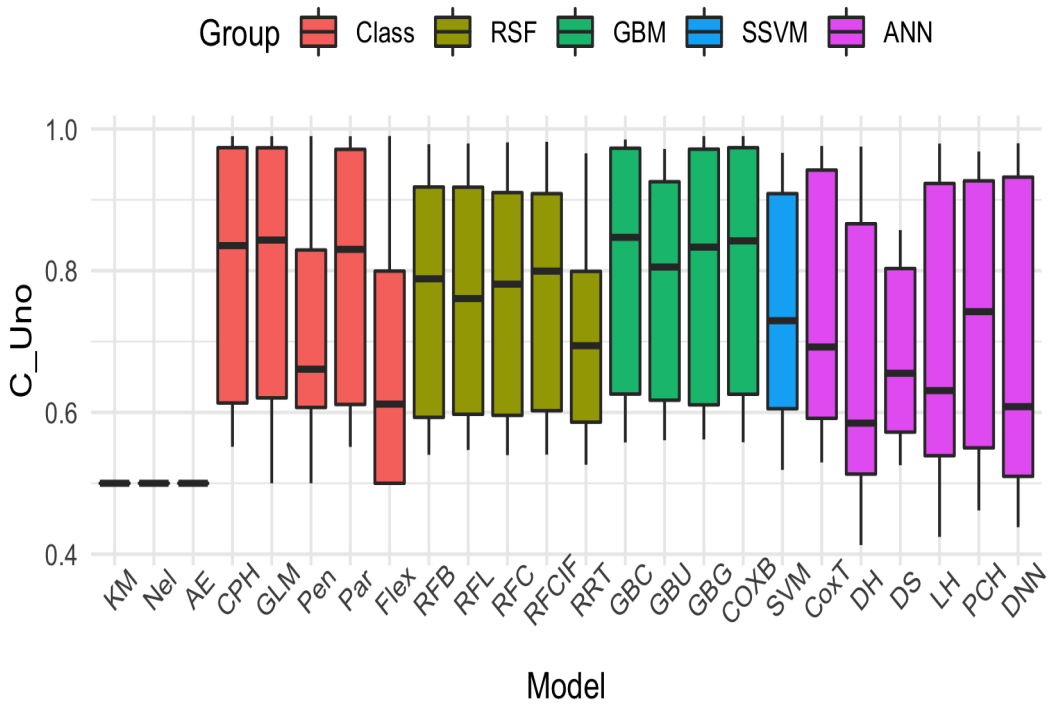


Figure 45: Boxplots of Uno's C across all simulated datasets for all models, coloured by model group. x-axis are the models and y-axis is Uno's C; boxplots constructed from calculation of Uno's C across all datasets.

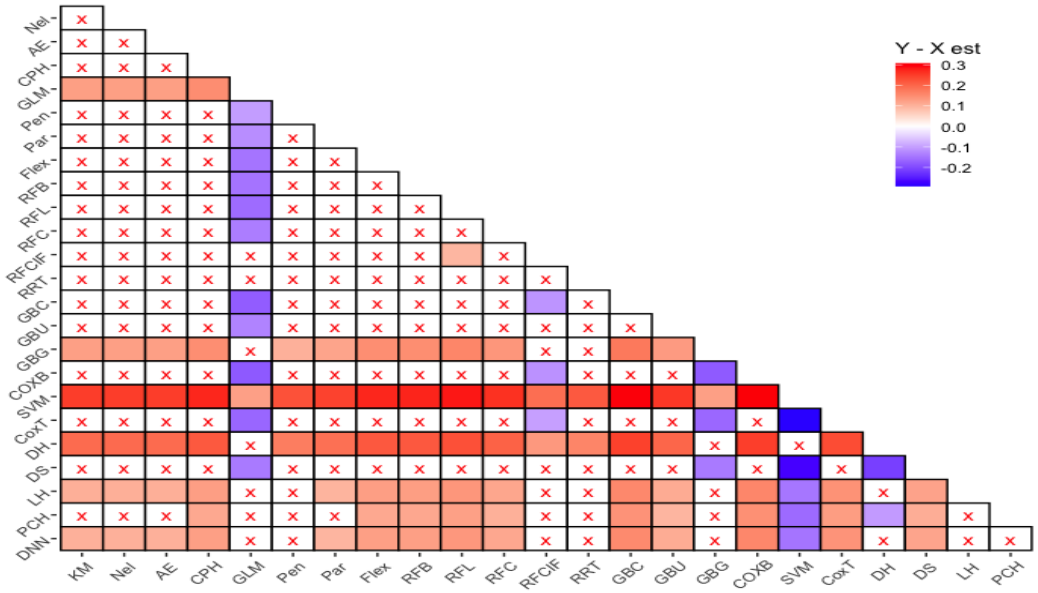


Figure 46: Tukey HSD on IGS for all simulated datasets. Blue squares indicate the model on the y-axis is significantly better than the model on the x-axis, and the reverse for red squares. Red 'x's indicate no significant difference.

7.4. Discussion

Similarly to how results were presented, first the real data experiments are discussed and then the simulated data experiments. Limitations of the experiments are then discussed in section 7.4.3 and conclusions are finally presented in section 7.4.4.

7.4.1. Real Data Experiments

The experiments on the real datasets are designed to determine which survival models are the best-performing on right-censored datasets with variable amounts of censoring. Results on individual datasets are not discussed as these cannot answer the research questions (section 7.1.1). The findings are limited to discrimination and overall performance, though calibration and distance is very briefly discussed.

Discrimination The critical difference plots indicate that GBMs and RSFs were among the best performing methods, however so too were non-ML models including Cox PH, parametric AFTs, and penalized Cox. Poor performance from all the neural networks was likely due to insufficient tuning, which is a problem in general with neural networks. The strong performance from the boosting models should be highlighted as they required more sub-sampling of the data than other models, yet despite this they were among the best performing. This may indicate that with less time-constraints, the boosted models could significantly outperform other methods. Non-significant differences between GBM models indicates that it is worth including all GBM 'types' (i.e. different underlying

losses) in benchmark experiments when these can be simply tuned as hyper-parameters (such as with **mboost**). Similarly the concordance-optimised RSF did not significantly outperform the other RSFs, again highlighting the benefit of including the splitting rule choice as a hyper-parameter.

Overall Performance Overall performance is judged by the IGS and should capture both a model's discrimination and calibration (though properness of the loss is in question). The Cox PH was the second best ranked model and significantly outperformed many neural networks and all the baselines. The slightly better performance from CoxBoost meant that in addition to the models outperformed by Cox PH, CoxBoost also outperformed the log-rank and Brier score splitting rule RSFs, as well as GBM-UNO. The poor performance from the SVM, the lowest ranked model, is likely due to the composition method used to transform the ranking prediction to a distribution. In fact, almost all the best performing models were those that directly predicted distributions and did not require compositions. Some models improved significantly in rankings compared to their evaluation on discrimination only, these include DeepSurv, Cox-Time, and flexible splines. This is likely due to calibration and is expanded on below.

Calibration and Distance For Houwelingen's α, the baselines, flexible splines, RSF-C, GBM-Uno, CoxBoost, and DeepSurv were all consistently well-calibrated across all datasets. This supports the hypothesis that RSF-C, GBM-Uno, and DeepSurv were all consistently better ranked with respect to IGS than Uno's C, due to their calibration. In stark contrast to previous results, RRTs performed the best with respect to distance measures, significantly outperforming some neural networks, all baselines, and the SSVM. It should be noted that only the SSVM directly predicted the survival time, all other models were via composition, despite this the SSVM did not significantly outperform the baselines.

7.4.2. Simulated Data Experiments

The simulation experiments are designed to test how classical and machine learning survival models differ in performance against different conditions: proportion of censoring, type of censoring, and survival time distribution.

Censoring Proportion In all tests with IPCW measures (i.e. the primary measures of IGS and Uno's C), the proportion of censoring had no significant impact upon model performance. This is a slightly surprising result as it could reasonably be expected that datasets with 80% censoring should be harder to make predictions for than datasets with 20% censoring. This finding is unlikely to be due to the chosen measures as both account for censoring with IPCW. Results were affected by the proportion of censoring when considering the distance measures, however this is likely an artefact of the measures removing censored observations in the test-set for evaluation without further weighting.

Censoring Type Determining the effect of censoring type on model performance is a more complex problem as the results changed significantly within

different model groups and underlying survival distributions. It is possible that these differences within models were due to the underlying assumptions on the censoring mechanism made by the models, but this cannot be definitively concluded without further study and model-specific investigation. It is worth noting that the typical usage of Type I censoring in simulation experiments (e.g. [50]) and packages for survival simulation (e.g. [35]), is not reflective of the real world. In the real world, Type I censoring does not occur in isolation of other censoring types. For example, if a study is conducted on patients within a set time period then Type I censoring occurs at the study end. However, right-informative censoring will also occur due to drop-out during the study period. As the 'pure' Type I censoring found in simulation studies only incorporates a single censoring mechanism, it may not even be representative of real-world data. Therefore, it is questionable if any results comparing censoring types with this pure Type I can be extrapolated to real settings.

Survival Distribution The most significant differences in performances were seen as a result of the survival time distribution (fig. 43), with models consistently performing worse on the Gompertz and Log-normal distributions. There is a strong possibility that this is a result of Cox-Weibull and Weibull having the PH property, whereas Gompertz and Log-normal do not.[1] Therefore it can be hypothesized that increased model performance was due to the majority of models assuming a PH form, especially as the majority of models in the experiment are derived from the Cox model. Conversely inferior performance would be due to model misspecification in the presence of non-PH data. It is possible that further analysis would indicate that models without an underlying PH assumption would see more equal performance spread across the four survival distribution types, however this is slightly tangential to this experiment design. For now we can tentatively conclude that survival models will in general perform better on datasets with the PH property. The post-hoc analysis within survival distributions closely follows the analysis with all distributions combined. For Uno's C, most ML models outperform the classical ones in all distributions whereas for IGS, the only strong conclusion is inferior performance by the SVMs and ANNs. Results between distributions for each measure are all concordant, e.g. model groups outperform each other in the 'same' direction (no reversal of which is superior/inferior).

Model Groups Comparing the model groups directly is not of primary interest in the simulation experiments as this is covered more extensively by the real experiments, however the results are still interesting and briefly discussed. The post-hoc analysis provides slightly contradictory results between Uno's C and IGS. This may highlight differences in calibration between the fitted models. This is hypothesised as the IGS can be decomposed into discrimination and calibration [109], thus any discrepancies between Uno's C and IGS may be due to model calibration. Post-hoc analysis on Uno's C indicates all models outperform the classical models, which is a promising finding that demonstrates machine learning has at least been successful in improving the discrimination of

[1] Gompertz distribution does have the PH property but post-hoc analysis of simulated data failed PH tests.

survival models. Conclusions should not be drawn on discrimination measures alone, as highlighted by the IGS results. For this measure the only strong conclusion is poor performance from the SVM and ANNs (as with real experiments), all other comparisons are non-significant. As above, perhaps this is due to poor model calibration, which could be identified given more widely utilised calibration measures. In this experiment only Houwelingen's α was computed; exploratory analysis identified no significant differences with respect to this measure.

7.4.3. Limitations

Primarily due to time and resource constraints, these experiments had several limitations. As the experiments were carried out on a high-performance cluster, the 'jobs' were limited to a maximum run-time of 48 hours, thus placing physical limitations on the size of the experiments. Many of these limitations accurately reflect real-world modelling and can thus be viewed not as limitations on the data but as a realistic comparison of models on real-world data. These are expanded on below.

In order to derive confidence intervals constructed from the standard error of the observed measures – which would enable model performance guarantees not just on the tested simulated datasets but on any similar dataset with the same conditions – the simulated experiments should have been repeated 30 times with varying seeds.

All models were originally intended to be tuned and optimised with 100 iteration random search, this was downgraded to 60. In some (though few) cases grid search with a small number of configurations was instead utilised, when this was the case hyper-parameter configurations were carefully chosen either by comparing default values from different packages or by researching previous experiment configurations.

Whilst some models were tuned using only a small grid of configurations, others utilised random search over a large range. This may be seen as an 'unfair advantage' to the models tuned with random search as there is more chance to find the optimal configuration for the given dataset [16].

Models were tuned with respect to Harrell's concordance index and therefore optimised for discrimination. This could explain why the results indicated a discrepancy between a model's calibration and discriminatory ability. A better choice would be to repeat the experiments in order to tune with respect to each measure separately.

Finally, some real datasets were sub-sampled considerably to ensure the models could finish within the set times. Again this can be viewed as a realistic limitation as resources are usually limited, however a full experiment should attempt to run all models with complete data where physically possible.

7.4.4. Conclusions and Future Work

Two research questions were asked in this study:

RQ1) What survival model or models perform best on right-censored time-to-event datasets without competing risks?

RQ2) How do performance of survival models vary with different simulated time-to-event dataset conditions including censoring proportion, underlying survival time distributions, and type of censoring?

(RQ1) is answered by considering measures of discrimination and overall performance (scoring rules) separately, as a secondary investigation measures of calibration and distance were briefly considered.

Results for the discrimination measure Uno's C, indicated that the top performing models came from a combination of all model groups, except ANNs. By the principle of parsimony, the Cox PH can still be considered an optimal model as it ranked amongst the top best performing and requires no model tuning. For overall performance, at least one model from all groups appeared amongst the best performing. Once again, the Cox PH is the most parsimonious model in the best performing group. The calibration measures further suggest that problems with model calibration may lead the discrepancy between model rankings between Uno's C and IGS. These results indicate that it is still worthwhile including a range of machine learning models in experiments on right-censored time-to-event datasets and most importantly that one should not be dismissive of implementing more parsimonious solutions such as the Cox PH.

(RQ2) is simpler to answer as the results of each condition can be broken down and addressed separately. The most important conditions were identified from a range of prior simulation studies, which leads one to naturally assume that large differences may be expected in predictive performance between these conditions. However the results indicate that there were no significant differences between the proportion of censoring between datasets and that any significant differences in the type of censoring can likely be explained by underlying model assumptions. The results further demonstrate that, on average, models are more likely to perform well on datasets that have the PH assumptions however this does not mean that this is true of all models, and it is worthwhile conducting similar experiments only on models that do not assume a PH form. In support of (RQ1), tests comparing model groups demonstrated inferior performance from ANNs and SVMs, though as noted above this may be due to study limitations.

Future Work These experiments can be considered a preliminary study and a larger version is already in progress that will address all the limitations above before being submitted for publication in the near-future.

As well as this experiment, three others are planned as part of a series of comprehensive experiments of survival models. The next experiment will look at the novel models proposed in chapter 3, which will be benchmarked against baselines (Kaplan-Meier) and classical models (Cox PH). The second experiment will look at the reductions and compositions proposed in chapter 5, in particular to assess the affects of tuning different hyper-parameters and also to practically establish sensible defaults and guidelines for future use. A final experiment will benchmark the most successful models from the prior three experiments.

Chapter 8

Conclusion and Future Research

Survival analysis is pervasive throughout the real-world with applications in public health, policy, engineering, and more. Machine learning is at the cutting-edge of Statistics with 'machine learning', 'artificial intelligence', and 'big data' being in the Zeitgeist. Correctly incorporating machine learning is a critical step in the evolution of survival analysis. This thesis surveys the efforts made to unify 'machine learning survival analysis', identifies how these could be improved, and addresses some of the open problems.

Contributions

This thesis attempted to achieve unification of machine learning survival analysis by presenting a theoretical and methodological framework. This framework consists of: i) the APT criteria for judging survival models and measures; ii) a unified nomenclature and taxonomy for survival analysis and machine learning; iii) models for survival predictions; iv) measures for evaluation; v) composition and reduction techniques for improved model performance and transparency; and vi) software for model accessibility. The title of this thesis claims to enable 'transparent and accessible predictive modelling' for machine learning survival analysis. This has been achieved by: i) reviewing and surveying existing models; ii) critically surveying measures to identify the pitfalls in current evaluation practices; and iii) implementing software packages in R to increase accessibility.

Chapter 2 introduced notation and terminology for use with survival analysis and machine learning, in particular clear definitions for the 'survival problem' or 'survival task'. This was especially important as it was noted in the survey of machine learning models and measures that conflicting terminology has often lead to repeated work, confusion in discussion between publications, or incompatible model and/or measure comparisons.

Chapter 3 surveyed and reviewed models for survival analysis, both in the classical and machine learning settings. The findings demonstrate that for model building, while there appears a large body of work in some areas, in others there is a distinct lack. For example, the survey of support vector machines reduced all proposals to a single 'hybrid' model. On the other hand whilst there have been many neural networks proposed for survival analysis, experiments in this paper identify that these do not outperform the classical Cox PH (chapter 7). There is

also evidence of a lack of transparency, and perhaps honesty, in model evaluation. Many proposed novel algorithms have either been presented: i) with measures that have been proven to lack robustness in the presence of censoring; ii) with non-proper scoring rules; iii) without comparison to simpler models or baselines; or iv) with unfair comparison to incomplete reductions.

As well as reviewing measures of discrimination and calibration, chapter 4 presented an in-depth look at survival scoring rules. The proofs in the chapter demonstrated that no commonly-utilised survival scoring rules are proper, which makes model comparison very difficult, if not impossible. A small class of measures were found to be strictly proper, however these require assumptions that are rarely realistic in real-world data.

Chapter 5 formalised the concepts of composition and reduction in survival analysis and presented workflows for both pre-existing and novel reduction pipelines. In doing so, non-survival (e.g. regression) models may be utilised to improve predictive performance in a survival setting. Formalisation allows more transparent implementation of these strategies.

In chapter 6, software packages and contributions to object-oriented programming in R were discussed. Most importantly, **mlr3proba** for machine learning in survival analysis was presented. This package has been well-received thus far and its accessibility will hopefully contribute to further interest in survival analysis within the machine learning community.

Finally, chapter 7 presented the first large-scale benchmark experiments for survival analysis. This was made possible by **mlr3proba** and the prior work in this thesis. The experiments highlight how much work remains in machine learning for survival analysis as there was significant evidence of machine learning models performing worse than classical counterparts. The experiments highlight the necessary research and further study required for machine learning survival analysis and should create engagement and interest in establishing more advanced techniques and technology in this area.

Future academic contributions resulting from this thesis will include: i) releasing chapters 3 and 4 as part of a (free) online book on machine learning survival analysis; ii) submitting chapter 5 for publication to collate and formalise composition and reduction workflows in survival; and iii) running and submitting a larger benchmark experiment using the code and setup provided in chapter 7.

Future Research

The research in this thesis will not finish here and years of future work remains. There are some sensible next steps that are immediately present, whilst others will emerge over time. Potential next steps include: i) studying the theoretical properties of the novel adaptations proposed in chapter 3 and if deemed sensible, implementing them in R and including them in future benchmark experiments; ii) studying the reduction strategies in chapter 5 with an emphasis on understanding when they perform well and also determining sensible defaults for their many hyper-parameters; iii) researching survival scoring rules further in order to establish best practice techniques for survival model evaluation; and iv) extending the reviews and surveys to Bayesian methods.

Concluding Thoughts

When I started writing this thesis, I was motivated by the lack of curiosity in machine learning survival analysis and frequent use of improper technique. In other fields of statistics, researchers (and data scientists in general) will often only train machine learning models without classical alternatives, the reverse is seen in survival analysis.

As I finish writing, two years later and in the middle of a global pandemic, I, alongside many others, am concerned and at times angry about how a lack of formalisation and rigour is leading to the development and promotion of poor survival models, even in top journals. At the time of writing, the BMJ 'living systematic review' [326] concluded that none of the 145 reviewed prediction models for COVID-19 could be recommended 'for use in current practice'.[1] One of their top reasons for dismissing a model was poor evaluation technique, a problem that is also highlighted throughout this thesis.

If I have achieved a single goal by writing this thesis, I hope that it will be an increased awareness of machine learning methods for survival analysis with proper techniques for model building and evaluation. Survival analysis and machine learning are two major fields in Statistics with a real-world benefit that cannot be understated. It is critical, now more than ever, that this is understood.

[1] It is unclear how many of these solve the survival task, at least 91 are classifiers.

Bibliography

[1] Odd Aalen. "Nonparametric Inference for a Family of Counting Processes". In: *The Annals of Statistics* 6.4 (1978), pp. 701–726.

[2] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. URL: https://www.tensorflow.org/.

[3] Erin L Abner, Richard J Charnigo, and Richard J Kryscio. "Markov chains and semi-Markov models in time-to-event analysis". eng. In: *Journal of biometrics & biostatistics* Suppl 1.e001 (2013), p. 19522. ISSN: 2155-6180. DOI: 10.4172/2155-6180.S1-e001. URL: https://www.ncbi.nlm.nih.gov/pubmed/24818062https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4013002/.

[4] Hirotugu Akaike. "A New Look at the Statistical Model Identification". In: *IEEE Transactions on Automatic Control* 19.6 (1974), pp. 716–723. ISSN: 17451337. DOI: 10.1093/ietfec/e90-a.12.2762. arXiv: arXiv:1011.1669v3.

[5] Michael G Akritas. "Nearest Neighbor Estimation of a Bivariate Distribution Under Random Censoring". en. In: *Ann. Statist.* 22.3 (1994), pp. 1299–1327. ISSN: 0090-5364. DOI: 10.1214/aos/1176325630. URL: https://projecteuclid.org:443/euclid.aos/1176325630.

[6] Alina M Allen et al. "Nonalcoholic fatty liver disease incidence and impact on metabolic burden and death: A 20 year-community study." eng. In: *Hepatology (Baltimore, Md.)* 67.5 (2018), pp. 1726–1736. ISSN: 1527-3350 (Electronic). DOI: 10.1002/hep.29546.

[7] Per K Andersen et al. *Statistical models based on counting processes*. Springer Science & Business Media, 2012. ISBN: 1461243483.

[8] Axel Andres et al. "A novel learning algorithm to predict individual survival after liver transplantation for primary sclerosing cholangitis". In: *PLOS ONE* 13.3 (2018), e0193523. URL: https://doi.org/10.1371/journal.pone.0193523.

[9] Bart Bakker et al. "Improving Cox survival analysis with a neural-Bayesian approach". In: *Statistics in Medicine* 23.19 (2004), pp. 2989–3012. ISSN: 0277-6715. DOI: 10.1002/sim.1904. URL: https://doi.org/10.1002/sim.1904.

[10] Ghalib A Bello et al. "Deep-learning cardiac motion analysis for human survival prediction". In: *Nature Machine Intelligence* 1.2 (2019), pp. 95–104. ISSN: 2522-5839. DOI: 10.1038/s42256-019-0019-2. URL: https://doi.org/10.1038/s42256-019-0019-2.

[11] Andreas Bender and Fabian Scheipl. "pammtools: Piece-wise exponential Additive Mixed Modeling tools". In: *arXiv:1806.01042 [stat]* (2018). URL: http://arxiv.org/abs/1806.01042.

[12] Andreas Bender et al. "Penalized estimation of complex, non-linear exposure-lag-response associations". In: *Biostatistics* 20.2 (2018), pp. 315–331. ISSN: 1465-4644. DOI: 10.1093/biostatistics/kxy003. URL: https://doi.org/10.1093/biostatistics/kxy003.

[13] Ralf Bender, Thomas Augustin, and Maria Blettner. "Generating survival times to simulate Cox proportional hazards models". In: *Statistics in Medicine* 24.11 (2005), pp. 1713–1723. ISSN: 0277-6715. DOI: 10.1002/sim.2059. URL: https://doi.org/10.1002/sim.2059.

[14] Henrik Bengtsson. *future: Unified Parallel and Distributed Processing in R for Everyone*. 2020. URL: https://cran.r-project.org/package=future.

[15] Steve Bennett. "Analysis of survival data by the proportional odds model". In: *Statistics in Medicine* 2.2 (1983), pp. 273–277. ISSN: 0277-6715. DOI: https://doi.org/10.1002/sim.4780020223. URL: https://doi.org/10.1002/sim.4780020223.

[16] James Bergstra and Yoshua Bengio. "Random search for hyper-parameter optimization". In: *The Journal of Machine Learning Research* 13.1 (2012), pp. 281–305. ISSN: 1532-4435.

[17] E M Biganzoli, F Ambrogi, and P Boracchi. "Partial logistic artificial neural networks (PLANN) for flexible modeling of censored survival data". In: *2009 International Joint Conference on Neural Networks*. 2009, pp. 340–346. ISBN: 2161-4407 VO -. DOI: 10.1109/IJCNN.2009.5178824.

[18] Elia Biganzoli et al. "Feed forward neural networks for the analysis of censored survival data: a partial logistic regression approach". In: *Statistics in Medicine* 17.10 (1998), pp. 1169–1186. ISSN: 0277-6715. DOI: 10.1002/(SICI)1097-0258(19980530)17:10<1169::AID-SIM796>3.0.CO;2-D.

URL: https://doi.org/10.1002/(SICI)1097-0258(19980530)17:10{\%}3C1169::AID-SIM796{\%}3E3.0.COhttp://2-d.

[19] Harald Binder and Martin Schumacher. "Allowing for mandatory covariates in boosting estimation of sparse high-dimensional survival models". In: *BMC Bioinformatics* 9.1 (2008), p. 14. ISSN: 1471-2105. DOI: 10.1186/1471-2105-9-14. URL: https://doi.org/10.1186/1471-2105-9-14.

[20] Harold Binder. *CoxBoost: Cox models by likelihood based boosting for a single survival endpoint or competing risks*. 2013.

[21] Martin Binder et al. *mlr3pipelines: Preprocessing Operators and Pipelines for 'mlr3'*. 2019. URL: https://cran.r-project.org/package=mlr3pipelines.

[22] Bernd Bischl et al. "mlr: Machine Learning in R". In: *Journal of Machine Learning Research* 17.170 (2016), pp. 1–5. URL: http://jmlr.org/papers/v17/15-066.htmlhttps://cran.r-project.org/package=mlr.

[23] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006. ISBN: 1493938436.

[24] Paul Blanche, Jean-François Dartigues, and Hélène Jacqmin-Gadda. "Review and comparison of ROC curve estimators for a time-dependent outcome with marker-dependent censoring". In: *Biometrical Journal* 55.5 (2013), pp. 687–704. ISSN: 0323-3847. DOI: 10.1002/bimj.201200045. URL: https://doi.org/10.1002/bimj.201200045.

[25] Paul Blanche, Aurélien Latouche, and Vivian Viallon. "Time-dependent AUC with right-censored data: a survey study". In: (2012). DOI: 10.1007/978-1-4614-8981-8_11.

[26] J Martin Bland and Douglas G. Altman. "The logrank test". eng. In: *BMJ (Clinical research ed.)* 328.7447 (2004), p. 1073. ISSN: 1756-1833. DOI: 10.1136/bmj.328.7447.1073. URL: https://pubmed.ncbi.nlm.nih.gov/15117797https://www.ncbi.nlm.nih.gov/pmc/articles/PMC403858/.

[27] Imad Bou-Hamad, Denis Larocque, and Hatem Ben-Ameur. "A review of survival trees". en. In: *Statist. Surv.* 5 (2011), pp. 44–71. ISSN: 1935-7516. DOI: 10.1214/09-SS047. URL: https://projecteuclid.org:443/euclid.ssu/1315833185.

[28] Anne-Laure Boulesteix, Sabine Lauer, and Manuel J A Eugster. "A Plea for Neutral Comparison Studies in Computational Sciences". In: *PLOS ONE* 8.4 (2013), e61562. URL: https://doi.org/10.1371/journal.pone.0061562.

[29] Hannah Bower et al. "Capturing simple and complex time-dependent effects using flexible parametric survival models: A simulation study". In: *Communications in Statistics - Simulation and Computation* (2019), pp. 1–17. ISSN: 0361-0918. DOI: 10.1080/03610918.2019.1634201. URL: https://doi.org/10.1080/03610918.2019.1634201.

[30] L Breiman et al. *Classification and Regression Trees*. The Wadsworth and Brooks-Cole statistics-probability series. Taylor & Francis, 1984. ISBN: 9780412048418. URL: https://books.google.co.uk/books?id=JwQx-WOmSyQC.

[31] Leo Breiman. "Bagging Predictors". In: *Machine Learning* 24.2 (1996), pp. 123–140. ISSN: 1573-0565. DOI: 10.1023/A:1018054314350. URL: http://statistics.berkeley.edu/sites/default/files/tech-reports/421.pdf.

[32] Leo Breiman and Philip Spector. "Submodel Selection and Evaluation in Regression. The X-Random Case". In: *International Statistical Review / Revue Internationale de Statistique* 60.3 (1992), pp. 291–319. ISSN: 03067734, 17515823. DOI: 10.2307/1403680. URL: http://www.jstor.org/stable/1403680.

[33] N E Breslow and N Chatterjee. "Design and analysis of two-phase studies with binary outcome applied to Wilms tumour prognosis". In: *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 48.4 (1999), pp. 457–468. ISSN: 0035-9254. DOI: 10.1111/1467-9876.00165. URL: https://doi.org/10.1111/1467-9876.00165.

[34] Glenn Brier. "Verification of forecasts expressed in terms of probability". In: *Monthly Weather Review* 78.1 (1950), pp. 1–3.

[35] Sam Brilleman. *simsurv: Simulate Survival Data*. 2019. URL: https://cran.r-project.org/package=simsurv.

[36] Jonathan Buckley and Ian James. "Linear Regression with Censored Data". In: *Biometrika* 66.3 (1979), pp. 429–436. ISSN: 00063444. DOI: 10.2307/2335161. URL: http://www.jstor.org/stable/2335161.

[37] Peter Buhlmann. "Boosting for high-dimensional linear models". en. In: *Ann. Statist.* 34.2 (2006), pp. 559–583. ISSN: 0090-5364. DOI: 10.1214/009053606000000092. URL: https://projecteuclid.org:443/euclid.aos/1151418234.

[38] Peter Buhlmann and Torsten Hothorn. "Boosting Algorithms: Regularization, Prediction and Model Fitting". en. In: *Statist. Sci.* 22.4 (2007), pp. 477–505. ISSN: 0883-4237. DOI: 10.1214/07-STS242. URL: https://projecteuclid.org:443/euclid.ss/1207580163.

[39] Peter Bühlmann and Bin Yu. "Boosting With the L2 Loss". In: *Journal of the American Statistical Association* 98.462 (2003), pp. 324–339. ISSN: 0162-1459. DOI: 10.1198/016214503000125. URL: https://doi.org/10.1198/016214503000125.

[40] Andrea Burton et al. "The design of simulation studies in medical statistics". In: *Statistics in Medicine* 25.24 (2006), pp. 4279–4292. ISSN: 02776715. DOI: 10.1002/sim.2673. arXiv: NIHMS150003. URL: http://doi.wiley.com/10.1002/sim.2673.

[41] John M Chambers. "Object-Oriented Programming, Functional Programming and R". en. In: *Statist. Sci.* 29.2 (2014), pp. 167–180. ISSN: 0883-4237. DOI: 10.1214/13-STS452. URL: https://projecteuclid.org:443/euclid.ss/1408368569.

[42] Lloyd E Chambless and Guoqing Diao. "Estimation of time-dependent area under the ROC curve for long-term risk prediction". In: *Statistics in Medicine* 25.20 (2006), pp. 3474–3486. ISSN: 0277-6715. DOI: 10.1002/sim.2299. URL: https://doi.org/10.1002/sim.2299.

[43] Winston Chang. *R6: Classes with Reference Semantics*. 2018. URL: https://cran.r-project.org/package=R6.

[44] Hung Chia Chen et al. "Assessment of performance of survival prediction models for cancer prognosis". In: *BMC Medical Research Methodology* 12 (2012). ISSN: 14712288. DOI: 10.1186/1471-2288-12-102.

[45] Tianqi Chen et al. *xgboost: Extreme Gradient Boosting*. 2020. URL: https://cran.r-project.org/package=xgboost.

[46] Yen-Chen Chen, Wan-Chi Ke, and Hung-Wen Chiu. "Risk classification of cancer survival using ANN with gene expression data from multiple laboratories". In: *Computers in Biology and Medicine* 48 (2014), pp. 1–7. ISSN: 0010-4825. DOI: https://doi.org/10.1016/j.compbiomed.2014.02.006. URL: http://www.sciencedirect.com/science/article/pii/S0010482514000377.

[47] Yifei Chen et al. "A Gradient Boosting Algorithm for Survival Analysis via Direct Optimization of Concordance Index". In: *Computational and Mathematical Methods in Medicine* 2013 (2013). Ed. by Lev Klebanov,

p. 873595. ISSN: 1748-670X. DOI: 10.1155/2013/873595. URL: https://doi.org/10.1155/2013/873595.

[48] Travers Ching. *cox-nnet*. 2015. URL: https://github.com/lanagarmire/cox-nnet.

[49] Travers Ching, Xun Zhu, and Lana X Garmire. "Cox-nnet: An artificial neural network method for prognosis prediction of high-throughput omics data". In: *PLOS Computational Biology* 14.4 (2018), e1006076. URL: https://doi.org/10.1371/journal.pcbi.1006076.

[50] Babak Choodari-Oskooei, Patrick Royston, and Mahesh K.B. Parmar. "A simulation study of predictive ability measures in a survival model I: Explained variation measures". In: *Statistics in Medicine* 31.23 (2012), pp. 2627–2643. ISSN: 02776715. DOI: 10.1002/sim.4242.

[51] Babak Choodari-Oskooei, Patrick Royston, and Mahesh K.B. Parmar. "A simulation study of predictive ability measures in a survival model II: explained randomness and predictive accuracy". In: *Statistics in Medicine* 31.23 (2012), pp. 2644–2659. ISSN: 02776715. DOI: 10.1002/sim.4242.

[52] Antonio Ciampi et al. "Stratification by stepwise regression, correspondence analysis and recursive partition: a comparison of three methods of analysis for survival data with covariates". In: *Computational Statistics and Data Analysis* 4.3 (1986), pp. 185–204. ISSN: 01679473. DOI: 10.1016/0167-9473(86)90033-2.

[53] Antonio Ciampi et al. "RECPAM: a computer program for recursive partition and amalgamation for censored survival data and other situations frequently occurring in biostatistics. I. Methods and program features". In: *Computer Methods and Programs in Biomedicine* 26.3 (1988), pp. 239–256. ISSN: 0169-2607. DOI: https://doi.org/10.1016/0169-2607(88)90004-1. URL: http://www.sciencedirect.com/science/article/pii/0169260788900041.

[54] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. "Fast and accurate deep network learning by exponential linear units (elus)". In: *arXiv preprint arXiv:1511.07289* (2015).

[55] David Collett. *Modelling Survival Data in Medical Research*. 3rd ed. CRC, 2014. ISBN: 978-1584883258.

[56] Gary S. Collins et al. "External validation of multivariable prediction models: A systematic review of methodological conduct and reporting". In: *BMC Medical Research Methodology* 14.1 (2014), pp. 1–11. ISSN: 14712288. DOI: 10.1186/1471-2288-14-40. URL: BMCMedicalResearchMethodology.

[57] Enrico Colosimo et al. "Empirical comparisons between Kaplan-Meier and Nelson-Aalen survival function estimators". In: *Journal of Statistical Computation and Simulation* 72.4 (2002), pp. 299–308. ISSN: 0094-9655. DOI: 10.1080/00949650212847. URL: https://doi.org/10.1080/00949650212847.

[58] Corinna Cortes and Vladimir Vapnik. "Support-Vector Networks". In: *Machine Learning* 20 (1995), pp. 273–297. ISSN: 08856125. DOI: 10.1007/BF00994018. arXiv: arXiv:1011.1669v3.

[59] D. R. Cox. "Regression Models and Life-Tables". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 34.2 (1972), pp. 187–220.

[60] D. R. Cox. "Partial Likelihood". In: *Biometrika* 62.2 (1975), pp. 269–276. ISSN: 03610918. DOI: 10.1080/03610910701884021.

[61] Lei Cui et al. "A deep learning-based framework for lung cancer survival analysis with biomarker interpretation". In: *BMC Bioinformatics* 21.1 (2020), p. 112. ISSN: 1471-2105. DOI: 10.1186/s12859-020-3431-z. URL: https://doi.org/10.1186/s12859-020-3431-z.

[62] Data Study Group Team. "Data Study Group Final Report: Great Ormond Street Hospital." 2020. URL: https://www.turing.ac.uk/research/publications/data-study-group-final-report-great-ormond-street-hospital.

[63] A P Dawid. "Present Position and Potential Developments: Some Personal Views: Statistical Theory: The Prequential Approach". In: *Journal of the Royal Statistical Society. Series A (General)* 147.2 (1984), pp. 278–292. ISSN: 00359238. DOI: 10.2307/2981683. URL: http://www.jstor.org/stable/2981683.

[64] A Philip Dawid. "Probability Forecasting". In: *Encyclopedia of Statistical Sciences* 7 (1986), pp. 210–218.

[65] A Philip Dawid and Monica Musio. "Theory and Applications of Proper Scoring Rules". In: *Metron* 72.2 (2014), pp. 169–183. arXiv: arXiv:1401.0398v1.

[66] Olga V Demler, Nina P Paynter, and Nancy R Cook. "Tests of calibration and goodness-of-fit in the survival setting". eng. In: *Statistics in medicine* 34.10 (2015), pp. 1659–1680. ISSN: 1097-0258. DOI: 10.1002/sim.6428. URL: https://pubmed.ncbi.nlm.nih.gov/25684707https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4555993/.

[67] Janez Demšar. "Statistical comparisons of classifiers over multiple data sets". In: *Journal of Machine learning research* 7.Jan (2006), pp. 1–30.

[68] John M Dennis et al. "Type 2 Diabetes and COVID-19–Related Mortality in the Critical Care Setting: A National Cohort Study in England, March–July 2020". In: *Diabetes Care* (2020), p. dc201444. DOI: 10.2337/dc20-1444. URL: http://care.diabetesjournals.org/content/early/2020/10/23/dc20-1444.abstract.

[69] Thomas G Dietterich. "Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms". In: *Neural Computation* 10.7 (1998), pp. 1895–1923. ISSN: 0899-7667. DOI: 10.1162/089976698300017197. URL: https://direct.mit.edu/neco/article/10/7/1895-1923/6224.

[70] Lore Dirick, Gerda Claeskens, and Bart Baesens. "Time to default in credit scoring using survival analysis: A benchmark study". In: *Journal of the Operational Research Society* 68.6 (2017), pp. 652–665. ISSN: 14769360. DOI: 10.1057/s41274-016-0128-9.

[71] Angela Dispenzieri et al. "Use of nonclonal serum immunoglobulin free light chains to predict overall survival in the general population". eng. In: *Mayo Clinic proceedings* 87.6 (2012), pp. 517–523. ISSN: 1942-5546. DOI: 10.1016/j.mayocp.2012.03.009. URL: https://www.ncbi.nlm.nih.gov/pubmed/22677072https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3538473/.

[72] Xian Du and Sumeet Dua. "Cancer prognosis using support vector regression in imaging modality". eng. In: *World journal of clinical oncology* 2.1 (2011), pp. 44–49. ISSN: 2218-4333. DOI: 10.5306/wjco.v2.i1.44. URL: https://pubmed.ncbi.nlm.nih.gov/21603313https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3095462/.

[73] Christophe Dutang, Vincent Goulet, and Mathieu Pigeon. "actuar: An R Package for Actuarial Science". In: *Journal of Statistical Software* 25.7 (2008), p. 38.

[74] Dirk Eddelbuettel and Romain Francois. "Rcpp: Seamless R and C++ Integration". In: *Journal of Statistical Software* 40.8 (2011), pp. 1–18. DOI: 10.18637/jss.v040.i08. URL: http://www.jstatsoft.org/v40/i08/.

[75] Bradley Efron. "Logistic Regression, Survival Analysis, and the Kaplan-Meier Curve". In: *Journal of the American Statistical Association* 83.402 (1988), pp. 414–425. ISSN: 01621459. DOI: 10.2307/2288857. URL: http://www.jstor.org/stable/2288857.

[76] Ludger Evers and Claudia-Martina Messow. "Sparse kernel methods for high-dimensional survival data". In: *Bioinformatics* 24.14 (2008), pp. 1632–1638. ISSN: 1460-2059.

[77] Dirk F. Moore. *asaur: Data Sets for "Applied Survival Analysis Using R"*. 2016. URL: https://cran.r-project.org/package=asaur.

[78] David Faraggi and Richard Simon. "A neural network model for survival data". In: *Statistics in Medicine* 14.1 (1995), pp. 73–82. ISSN: 10970258. DOI: 10.1002/sim.4780140108. arXiv: arXiv:1011.1669v3.

[79] David Faraggi et al. "Bayesian Neural Network Models for Censored Data". In: *Biometrical Journal* 39.5 (1997), pp. 519–532. ISSN: 0323-3847. DOI: 10.1002/bimj.4710390502. URL: https://doi.org/10.1002/bimj.4710390502.

[80] Tamara Fernández, Nicolas Nicolás Rivera, and Yee Whye Teh. "Gaussian Processes for Survival Analysis". In: *Neural Information Processing Systems* Nips (2016). ISSN: 10495258. arXiv: 1611.00817. URL: http://arxiv.org/abs/1611.00817.

[81] Thomas R Fleming et al. "Modified Kolmogorov-Smirnov Test Procedures with Application to Arbitrarily Right-Censored Data". In: *Biometrics* 36.4 (1980), pp. 607–625. ISSN: 0006341X, 15410420. DOI: 10.2307/2556114. URL: http://www.jstor.org/stable/2556114.

[82] Stephane Fotso. "Deep Neural Networks for Survival Analysis Based on a Multi-Task Framework". In: *arXiv preprint arXiv:1801.05512* (2018). arXiv: 1801.05512. URL: http://arxiv.org/abs/1801.05512.

[83] Stephane Fotso. *PySurvival: Open source package for Survival Analysis modeling*. 2019. URL: https://www.pysurvival.io/.

[84] Cesaire J K Fouodo et al. "Support vector machines for survival analysis with R". In: *The R Journal* 10.July (2018), pp. 412–423. ISSN: 20734859.

[85] Yoav Freund and Robert E Schapire. "Experiments with a new boosting algorithm". In: Citeseer, 1996.

[86] Jerome Friedman. "Stochastic Gradient Boosting". In: *Computational Statistics & Data Analysis* 38 (1999), pp. 367–378. DOI: 10.1016/S0167-9473(01)00065-2.

[87] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. "Regularization Paths for Generalized Linear Models via Coordinate Descent." In: *Journal of Statistical Software* 33.1 (2010), pp. 1–22. URL: http://www.jstatsoft.org/v33/i01/.

[88] Jerome H Friedman. "Greedy Function Approximation: A Gradient Boosting Machine". In: *The Annals of Statistics* 29.5 (2001), pp. 1189–1232. ISSN: 00905364. URL: http://www.jstor.org/stable/2699986.

[89] Michael Friedman. "Piecewise exponential models for survival data with covariates". In: *The Annals of Statistics* 10.1 (1982), pp. 101–113. ISSN: 0090-5364.

[90] Stefan Fritsch, Frauke Guenther, and Marvin N. Wright. *neuralnet: Training of Neural Networks*. 2019. URL: https://cran.r-project.org/package=neuralnet.

[91] E Gamma et al. *Design Patterns: Elements of Reusable Software*. 1996.

[92] Enora Gandon et al. "Assessing the influence of culture on craft skills: A quantitative study with expert Nepalese potters". In: *PLOS ONE* 15.10 (2020), e0239139. URL: https://doi.org/10.1371/journal.pone.0239139.

[93] Enora Gandon et al. "Cultural transmission and perception of vessel shapes among Hebron potters". In: *Journal of Anthropological Archaeology* (2021).

[94] Alan E Gelfand et al. "Proportional hazards models: a latent competing risk approach". In: *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 49.3 (2000), pp. 385–397. ISSN: 0035-9254. DOI: https://doi.org/10.1111/1467-9876.00199. URL: https://doi.org/10.1111/1467-9876.00199.

[95] Michael F. Gensheimer and Balasubramanian Narasimhan. "A Simple Discrete-Time Survival Model for Neural Networks". In: (2018), pp. 1–17. DOI: arXiv:1805.00917v3. arXiv: 1805.00917. URL: http://arxiv.org/abs/1805.00917.

[96] Michael F Gensheimer and Balasubramanian Narasimhan. "A scalable discrete-time survival model for neural networks". In: *PeerJ* 7 (2019), e6257. ISSN: 2167-8359.

[97] Ekavi N Georgousopoulou et al. "Comparisons between Survival Models in Predicting Cardiovascular Disease Events : Application in the ATTICA Study (2002-2012)." In: *Journal of Statistics Applications & Probability* 4.2 (2015), pp. 203–210.

[98] Thomas A Gerds and Martin Schumacher. "Consistent Estimation of the Expected Brier Score in General Survival Models with Right-Censored Event Times". In: *Biometrical Journal* 48.6 (2006), pp. 1029–1040. ISSN: 0323-3847. DOI: 10.1002/bimj.200610301. URL: https://doi.org/10.1002/bimj.200610301.

[99] Eleonora Giunchiglia, Anton Nemchenko, and Mihaela van der Schaar. "Rnn-surv: A deep recurrent model for survival analysis". In: *International Conference on Artificial Neural Networks*. Springer, 2018, pp. 23–32.

[100] Tilmann Gneiting and Adrian E Raftery. "Strictly Proper Scoring Rules, Prediction, and Estimation". In: *American Statistical Association* 102.477 (2007). DOI: 10.1198/016214506000001437.

[101] J. J. Goeman. "L1 penalized estimation in the Cox proportional hazards model". In: *Biometrical Journal* 52.1 (2010), pp. –14.

[102] Shahrbanoo Goli et al. "Performance Evaluation of Support Vector Regression Models for Survival Analysis: A Simulation Study". In: *International Journal of Advanced Computer Science and Applications* 7 (2016). DOI: 10.14569/IJACSA.2016.070650.

[103] Shahrbanoo Goli et al. "Survival Prediction and Feature Selection in Patients with Breast Cancer Using Support Vector Regression". In: *Computational and Mathematical Methods in Medicine* 2016 (2016). Ed. by Francesco Pappalardo, p. 2157984. ISSN: 1748-670X. DOI: 10.1155/2016/2157984. URL: https://doi.org/10.1155/2016/2157984.

[104] Benjamin Gompertz. "On the Nature of the Function Expressive of the Law of Human Mortality, and on a New Mode of Determining the Value of Life Contingencies". In: *Philosophical Transactions of the Royal Society of London* 115 (1825), pp. 513–583.

[105] Mithat Gönen and Glenn Heller. "Concordance Probability and Discriminatory Power in Proportional Hazards Regression". In: *Biometrika* 92.4 (2005), pp. 965–970.

[106] I J Good. "Rational Decisions". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 14.1 (1952), pp. 107–114. ISSN: 00359246. URL: http://www.jstor.org/stable/2984087.

[107] Louis Gordon and Richard A Olshen. "Tree-structured survival analysis." In: *Cancer treatment reports* 69.10 (1985), pp. 1065–1069. ISSN: 0361-5960.

[108] Erika Graf and Martin Schumacher. "An Investigation on Measures of Explained Variation in Survival Analysis". In: *Journal of the Royal Statistical Society. Series D (The Statistician)* 44.4 (1995), pp. 497–507. ISSN: 00390526, 14679884. DOI: 10.2307/2348898. URL: http://www.jstor.org/stable/2348898.

[109] Erika Graf et al. "Assessment and comparison of prognostic classification schemes for survival data". In: *Statistics in Medicine* 18.17-18 (1999), pp. 2529–2545. ISSN: 0277-6715. DOI: 10.1002/(SICI)1097-0258(19990915/30)18:17/18<2529::AID-SIM274>3.0.CO;2-5. URL: http://doi.wiley.com/10.1002/{\%}28SICI{\%}291097-0258{\%}2819990915/30{\%}2918{\%}3A17/18{\%}3C2529{\%}3A{\%}3AAID-SIM274{\%}3E3.0.CO{\%}3B2-5.

[110] Brandon Greenwell et al. *gbm: Generalized Boosted Regression Models*. 2019. URL: https://cran.r-project.org/package=gbm.

[111] Frithjof Gressmann et al. "Probabilistic supervised learning". 2018. URL: http://arxiv.org/abs/1801.00753.

[112] Danial Habibi et al. "Comparison of Survival Models for Analyzing Prognostic Factors in Gastric Cancer Patients". In: *Asian Pacific journal of cancer prevention : APJCP* 19.3 (2018), pp. 749–753. ISSN: 2476-762X. DOI: 10.22034/APJCP.2018.19.3.749. URL: http://www.ncbi.nlm.nih.gov/pubmed/29582630{\%}0Ahttp://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC5980851.

[113] Humza Haider et al. "Effective ways to build and evaluate individual survival distributions". In: *Journal of Machine Learning Research* 21.85 (2020), pp. 1–63. ISSN: 1533-7928.

[114] Ilkyu Han et al. "Deep learning approach for survival prediction for patients with synovial sarcoma". In: *Tumor Biology* 40.9 (2018), p. 1010428318799264. ISSN: 1010-4283. DOI: 10.1177/1010428318799264. URL: https://doi.org/10.1177/1010428318799264.

[115] F E Jr Harrell et al. "Regression modelling strategies for improved prognostic prediction." eng. In: *Statistics in medicine* 3.2 (1984), pp. 143–152. ISSN: 0277-6715 (Print). DOI: 10.1002/sim.4780030207.

[116] Frank E. Harrell, Robert M. Califf, and David B. Pryor. "Evaluating the yield of medical tests". In: *JAMA* 247.18 (1982), pp. 2543–2546. ISSN: 0098-7484. URL: http://dx.doi.org/10.1001/jama.1982.03320430047030.

[117] Frank E. Harrell, Kerry L. Lee, and Daniel B. Mark. "Multivariable Prognostic Models: Issues in Developing Models, Evaluating Assumptions and Adequacy, and Measuring and Reducing Errors". In: *Statistics in Medicine* 15 (1996), pp. 361–387. ISSN: 02776715. DOI: 10.1002/0470023678.ch2b(i).

[118] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer New York Inc., 2001.

[119] Alex Hayes and Ralph Moller-Trane. *distributions3: Probability Distributions as S3 Objects*. 2019. URL: https://cran.r-project.org/package=distributions3.

[120] Patrick J. Heagerty, Thomas Lumley, and Margaret S. Pepe. "Time-Dependent ROC Curves for Censored Survival Data and a Diagnostic Marker". In: *Biometrics* 56.2 (2000), pp. 337–344. ISSN: 0006-341X. DOI: 10.1111/j.0006-341X.2000.00337.x. URL: https://doi.org/10.1111/j.0006-341X.2000.00337.x.

[121] Patrick J Heagerty and Yingye Zheng. "Survival Model Predictive Accuracy and ROC Curves Patrick". In: *Biometrics* 61 (2005), pp. 92–105. ISSN: 00220930.

[122] Henderson and Velleman. "Building multiple regression models interactively". In: *Biometrics* 37 (1981), pp. 391–411.

[123] Moritz Herrmann et al. "Large-scale benchmark study of survival prediction methods using multi-omics data". In: *arXiv preprint arXiv:2003.03621* (2020).

[124] Thomas Hielscher et al. "On the Prognostic Value of Gene Expression Signatures for Censored Data BT - Advances in Data Analysis, Data Handling and Business Intelligence". In: ed. by Andreas Fink et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 663–673. ISBN: 978-3-642-01044-6.

[125] David W Hosmer and Stanley Lemeshow. "Goodness of fit tests for the multiple logistic regression model". In: *Communications in statistics-Theory and Methods* 9.10 (1980), pp. 1043–1069. ISSN: 0361-0926.

[126] David W Hosmer Jr, Stanley Lemeshow, and Susanne May. *Applied survival analysis: regression modeling of time-to-event data*. Vol. 618. John Wiley & Sons, 2011. ISBN: 1118211588.

[127] Torsten Hothorn, Kurt Hornik, and Achim Zeileis. "Unbiased Recursive Partitioning: A Conditional Inference Framework". In: *Journal of Computational and Graphical Statistics* 15.3 (2006), pp. 651–674.

[128] Torsten Hothorn and Berthold Lausen. "On the exact distribution of maximally selected rank statistics". In: *Computational Statistics & Data Analysis* 43.2 (2003), pp. 121–137. ISSN: 0167-9473. DOI: 10.1016/S0167-9473(02)00225-6. URL: https://www.sciencedirect.com/science/article/pii/S0167947302002256.

[129] Torsten Hothorn and Achim Zeileis. "partykit: A Modular Toolkit for Recursive Partytioning in R." In: *Journal of Machine Learning Research* 16 (2015), pp. 3905–3909. URL: http://jmlr.org/papers/v16/hothorn15a.html.

[130] Torsten Hothorn et al. "Bagging survival trees". In: *Statistics in Medicine* 23.1 (2004), pp. 77–91. ISSN: 0277-6715. DOI: 10.1002/sim.1593. URL: https://doi.org/10.1002/sim.1593.

[131] Torsten Hothorn et al. "Survival ensembles". In: *Biostatistics* 7.3 (2005), pp. 355–373. ISSN: 1465-4644. DOI: 10.1093/biostatistics/kxj011. URL: https://doi.org/10.1093/biostatistics/kxj011.

[132] Torsten Hothorn et al. *mboost: Model-Based Boosting*. 2020. URL: https://cran.r-project.org/package=mboost.

[133] Hans C van Houwelingen and Hein Putter. "Dynamic predicting by landmarking as an alternative for multi-state modeling: an application to acute lymphoid leukemia data". eng. In: *Lifetime data analysis* 14.4 (2008), pp. 447–463. ISSN: 1380-7870. DOI: 10.1007/s10985-008-9099-8. URL: https://pubmed.ncbi.nlm.nih.gov/18836831https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2798037/.

[134] Shigao Huang et al. "Artificial intelligence in cancer diagnosis and prognosis: Opportunities and challenges". In: *Cancer Letters* 471 (2020), pp. 61–71. ISSN: 0304-3835. DOI: https://doi.org/10.1016/j.canlet.2019.12.007. URL: http://www.sciencedirect.com/science/article/pii/S0304383519306135.

[135] Shigao Huang et al. "Artificial intelligence in cancer diagnosis and prognosis: Opportunities and challenges". In: *Cancer Letters* 471 (2020), pp. 61–71. ISSN: 0304-3835. DOI: https://doi.org/10.1016/j.canlet.2019.12.007. URL: http://www.sciencedirect.com/science/article/pii/S0304383519306135.

[136] Hung Hung and Chin-Tsang Chiang. "Estimation methods for time-dependent AUC models with survival data". In: *The Canadian Journal of Statistics / La Revue Canadienne de Statistique* 38.1 (2010), pp. 8–26. ISSN: 03195724. URL: http://www.jstor.org/stable/27805213.

[137] CLIFFORD M HURVICH and CHIH-LING TSAI. "Regression and time series model selection in small samples". In: *Biometrika* 76.2 (1989), pp. 297–307. ISSN: 0006-3444. DOI: 10.1093/biomet/76.2.297. URL: https://doi.org/10.1093/biomet/76.2.297.

[138] By Hemant Ishwaran et al. "Random survival forests". In: *The Annals of Statistics* 2.3 (2008), pp. 841–860. DOI: 10.1214/08-AOAS169. arXiv: arXiv:0811.1645v1.

[139] Hemant Ishwaran and Udaya B Kogalur. *randomForestSRC*. 2018. URL: https://cran.r-project.org/package=randomForestSRC.

[140] Hemant Ishwaran et al. "Relative Risk Forests for Exercise Heart Rate Recovery as a Predictor of Mortality". In: *Journal of the American Statistical Association* 99.467 (2004), pp. 591–600. ISSN: 0162-1459. DOI: 10.1198/016214504000000638. URL: https://doi.org/10.1198/016214504000000638.

[141] Christopher Jackson. "flexsurv: A Platform for Parametric Survival Modeling in R". In: *Journal of Statistical Software* 70.8 (2016), pp. 1–33.

[142] Dan Jackson et al. "Relaxing the independent censoring assumption in the Cox proportional hazards model using multiple imputation". In: *Statistics in Medicine* 33.27 (2014), pp. 4681–4694. ISSN: 10970258. DOI: 10.1002/sim.6274.

[143] Byron Jaeger. *obliqueRSF: Oblique Random Forests for Right-Censored Time-to-Event Data*. 2019. URL: https://cran.r-project.org/package=obliqueRSF.

[144] Kitty J Jager et al. "The analysis of survival data: the Kaplan–Meier method". In: *Kidney International* 74.5 (2008), pp. 560–565. ISSN: 0085-2538. DOI: https://doi.org/10.1038/ki.2008.217. URL: http://www.sciencedirect.com/science/article/pii/S0085253815533681.

[145] Gareth James et al. *An introduction to statistical learning*. Vol. 112. New York: Springer, 2013.

[146] Yasen Jiao and Pufeng Du. "Performance measures in evaluating machine learning based bioinformatics predictors for classifications". In: *Quantitative Biology* 4.4 (2016), pp. 320–330. ISSN: 2095-4689.

[147] Bingzhong Jing et al. *RankDeepSurv*. 2018. URL: https://github.com/sysucc-ailab/RankDeepSurv.

[148] Bingzhong Jing et al. "A deep survival analysis method based on ranking". In: *Artificial Intelligence in Medicine* 98 (2019), pp. 1–9. ISSN: 0933-3657. DOI: https://doi.org/10.1016/j.artmed.2019.06.001. URL: http://www.sciencedirect.com/science/article/pii/S0933365718305992.

[149] Brent A Johnson and Qi Long. "Survival ensembles by the sum of pairwise differences with application to lung cancer microarray studies". en. In: *Ann. Appl. Stat.* 5.2A (2011), pp. 1081–1101. ISSN: 1932-6157. DOI: 10.1214/10-AOAS426. URL: https://projecteuclid.org:443/euclid.aoas/1310562217.

[150] Eric J Johnson and Daniel Goldstein. "Do Defaults Save Lives?" In: *Science* 302.5649 (2003), 1338 LP –1339. DOI: 10.1126/science.1091721. URL: http://science.sciencemag.org/content/302/5649/1338.abstract.

[151] John D Kalbfleisch and Ross L Prentice. *The statistical analysis of failure time data*. Vol. 360. John Wiley & Sons, 2011. ISBN: 1118031237.

[152] Adina Najwa Kamarudin, Trevor Cox, and Ruwanthi Kolamunnage-Dona. "Time-dependent ROC curve analysis in medical research: Current methods and applications". In: *BMC Medical Research Methodology* 17.1 (2017), pp. 1–19. ISSN: 14712288. DOI: 10.1186/s12874-017-0332-6.

[153] E. L. Kaplan and Paul Meier. "Nonparametric Estimation from Incomplete Observations". In: *Journal of the American Statistical Association* 53.282 (1958), pp. 457–481. ISSN: 01621459. DOI: 10.2307/2281868.

[154] MICHAEL W KATTAN. "Comparison of Cox Regression With Other Methods for Determining Prediction Models and Nomograms". In: *Journal of Urology* 170.6S (2003), S6–S10. ISSN: 0022-5347. DOI: 10.1097/01.ju.0000094764.56269.2d. URL: http://www.sciencedirect.com/science/article/pii/S0022534701682508http://www.jurology.com/doi/10.1097/01.ju.0000094764.56269.2d.

[155] Jared Katzman et al. "Deep Survival: A Deep Cox Proportional Hazards Network". In: (2016).

[156] Jared L Katzman et al. "DeepSurv: personalized treatment recommender system using a Cox proportional hazards deep neural network". In: *BMC Medical Research Methodology* 18.1 (2018), p. 24. ISSN: 1471-2288. DOI: 10.1186/s12874-018-0482-1. URL: https://doi.org/10.1186/s12874-018-0482-1.

[157] John T. Kent and John O'Quigley. "Measures of dependence for censored survival data". In: *Biometrika* 75.3 (1988), pp. 525–534. ISSN: 00063444. DOI: 10.1093/biomet/75.3.525.

[158] Faisal M. Khan and Valentina Bayer Zubek. "Support vector regression for censored data (SVRc): A novel tool for survival analysis". In: *Proceedings - IEEE International Conference on Data Mining, ICDM* (2008), pp. 863–868. ISSN: 15504786. DOI: 10.1109/ICDM.2008.50.

[159] Minyoung Kim and Vladimir Pavlovic. "Variational Inference for Gaussian Process Models for Survival Analysis." In: *UAI* (2018), pp. 435–445.

[160] W Ray Kim et al. "Deaths on the liver transplant waiting list: an analysis of competing risks." eng. In: *Hepatology (Baltimore, Md.)* 43.2 (2006), pp. 345–351. ISSN: 0270-9139 (Print). DOI: 10.1002/hep.21025.

[161] Franz J Király, Bilal Mateen, and Raphael Sonabend. "NIPS - Not Even Wrong? A Systematic Review of Empirically Complete Demonstrations of Algorithmic Effectiveness in the Machine Learning and Artificial Intelligence Literature". In: *arXiv* (2018). arXiv: 1812.07519. URL: http://arxiv.org/abs/1812.07519.

[162] Franz J. Király et al. "Designing Machine Learning Toolboxes: Concepts, Principles and Patterns". In: *arXiv* (2021). arXiv: 2101.04938. URL: http://arxiv.org/abs/2101.04938.

[163] S N U A Kirmani and Ramesh C Gupta. "On the Proportional Odds Model in Survival Analysis". In: *Annals of the Institute of Statistical Mathematics* 53.2 (2001), pp. 203–216. ISSN: 1572-9052. DOI: 10.1023/A:1012458303498. URL: https://doi.org/10.1023/A:1012458303498.

[164] John P Klein and Melvin L Moeschberger. *Survival analysis: techniques for censored and truncated data*. 2nd ed. Springer Science & Business Media, 2003. ISBN: 0387216456.

[165] John P Klein and Melvin L Moeschberger. *KMsurv: Data sets from Klein and Moeschberger (1997), Survival Analysis*. 2012. URL: https://cran.r-project.org/package=KMsurv.

[166] Ron Kohavi. "A study of cross-validation and bootstrap for accuracy estimation and model selection". In: *Ijcai* 14.2 (1995), pp. 1137–1145.

[167] Edward L. Korn and Richard Simon. "Measures of explained variation for survival data". In: *Statistics in Medicine* 9.5 (1990), pp. 487–503. ISSN: 10970258. DOI: 10.1002/sim.4780090503.

[168] Edward L Korn and Richard Simon. "Explained Residual Variation, Explained Risk, and Goodness of Fit". In: *The American Statistician* 45.3 (1991), pp. 201–206. ISSN: 00031305. DOI: 10.2307/2684290. URL: http://www.jstor.org/stable/2684290.

[169] James A. Koziol and Zhenyu Jia. "The concordance index C and the Mann-Whitney parameter Pr(X>Y) with randomly censored data". In: *Biometrical Journal* 51.3 (2009), pp. 467–474. ISSN: 03233847. DOI: 10.1002/bimj.200800228.

[170] Max Kuhn. "Building Predictive Models in R Using the caret Package". In: *Journal of Statistical Software; Vol 1, Issue 5 (2008)* (2008). URL: https://www.jstatsoft.org/v028/i05http://dx.doi.org/10.18637/jss.v028.i05.

[171] Max Kuhn and Hadley Wickham. *tidymodels: Easily Install and Load the 'Tidymodels' Packages*. 2020. URL: https://cran.r-project.org/package=tidymodels.

[172] Håvard Kvamme. *pycox*. 2018. URL: https://pypi.org/project/pycox/.

[173] Håvard Kvamme and Ørnulf Borgan. "Continuous and discrete-time survival prediction with neural networks". In: *arXiv preprint arXiv:1910.06724* (2019).

[174] Håvard Kvamme, Ørnulf Borgan, and Ida Scheel. "Time-to-event prediction with neural networks and Cox regression". In: *Journal of Machine Learning Research* 20.129 (2019), pp. 1–30. ISSN: 1533-7928.

[175] R A Kyle. ""Benign" monoclonal gammopathy–after 20 to 35 years of follow-up." eng. In: *Mayo Clinic proceedings* 68.1 (1993), pp. 26–36. ISSN: 0025-6196 (Print). DOI: 10.1016/s0025-6196(12)60015-9.

[176] Mark K van der Laan, Eric C Polley, and Alan E Hubbard. "Super Learner". In: *Statistical Applications in Genetics and Molecular Biology* 6.1 (2007). DOI: 10.2202/1544-6115.1309. URL: https://www.degruyter.com/view/j/sagmb.2007.6.issue-1/sagmb.2007.6.1.1309/sagmb.2007.6.1.1309.xml.

[177] Walker H Land et al. "A new tool for survival analysis: evolutionary programming/evolutionary strategies (EP/ES) support vector regression hybrid using both censored / non-censored (event) data". In: *Procedia Computer Science* 6 (2011), pp. 267–272. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2011.08.050. URL: http://www.sciencedirect.com/science/article/pii/S1877050911005151.

[178] Michel Lang. "checkmate: Fast Argument Checks for Defensive R Programming". In: *The R Journal* 9.1 (2017), pp. 437–445.

[179] Michel Lang and Patrick Schratz. *mlr3misc: Helper Functions for 'mlr3'*. 2020. URL: https://cran.r-project.org/package=mlr3misc.

[180] Michel Lang et al. *mlr3tuning: Tuning for 'mlr3'*. 2019. URL: https://cran.r-project.org/package=mlr3tuning.

[181] Michel Lang et al. *paradox: Define and Work with Parameter Spaces for Complex Algorithms*. 2019. URL: https://cran.r-project.org/package=paradox.

[182] Michel Lang et al. "mlr3: A modern object-oriented machine learning framework in R". In: *Journal of Open Source Software* 4.44 (2019), p. 1903. DOI: 10.21105/joss.01903. URL: https://joss.theoj.org/papers/10.21105/joss.01903https://cran.r-project.org/package=mlr3.

[183] Michel Lang et al. *mlr3learners: Recommended Learners for 'mlr3'*. 2020. URL: https://cran.r-project.org/package=mlr3learners.

[184] John Langford et al. "Learning Reductions that Really Work". In: *Proceedings of the IEEE* 104.1 (2016).

[185] Jiangwei Lao et al. "A Deep Learning-Based Radiomics Model for Prediction of Survival in Glioblastoma Multiforme". In: *Scientific Reports* 7.1 (2017), p. 10353. ISSN: 2045-2322. DOI: 10.1038/s41598-017-10649-8. URL: https://doi.org/10.1038/s41598-017-10649-8.

[186] Jerald F Lawless and Yan Yuan. "Estimation of prediction error for survival models". In: *Statistics in Medicine* 29.2 (2010), pp. 262–274. ISSN: 0277-6715. DOI: 10.1002/sim.3758. URL: https://doi.org/10.1002/sim.3758.

[187] David Lazer et al. "The Parable of Google Flu: Traps in Big Data Analysis". In: *Science* 343.6176 (2014), 1203 LP –1205. DOI: 10.1126/science.1248506. URL: http://science.sciencemag.org/content/343/6176/1203.abstract.

[188] Michael LeBlanc and John Crowley. "Relative Risk Trees for Censored Survival Data". In: *Biometrics* 48.2 (1992), pp. 411–425. ISSN: 0006341X, 15410420. DOI: 10.2307/2532300. URL: http://www.jstor.org/stable/2532300.

[189] Michael LeBlanc and John Crowley. "Survival Trees by Goodness of Split". In: *Journal of the American Statistical Association* 88.422 (1993), pp. 457–467. ISSN: 01621459. DOI: 10.2307/2290325. URL: http://www.jstor.org/stable/2290325.

[190] Pierre L'Ecuyer. "Good Parameters and Implementations for Combined Multiple Recursive Random Number Generators". In: *Operations Research* 47.1 (1999), pp. 159–164. ISSN: 0030-364X. DOI: 10.1287/opre.47.1.159. URL: https://pubsonline.informs.org/doi/abs/10.1287/opre.47.1.159.

[191] Changhee Lee et al. "Deephit: A deep learning approach to survival analysis with competing risks". In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[192] Donald K K Lee, Ningyuan Chen, and Hemant Ishwaran. "Boosted nonparametric hazards with time-dependent covariates". 2019.

[193] Seungyeoun Lee and Heeju Lim. "Review of statistical methods for survival analysis using genomic data". eng. In: *Genomics & informatics* 17.4 (2019), e41–e41. ISSN: 1598-866X. DOI: 10.5808/GI.2019.17.4.e41. URL: https://pubmed.ncbi.nlm.nih.gov/31896241https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6944043/.

[194] Li Li et al. "CXCL17 Expression Predicts Poor Prognosis and Correlates with Adverse Immune Infiltration in Hepatocellular Carcinoma". In: *PLOS ONE* 9.10 (2014), e110064. URL: https://doi.org/10.1371/journal.pone.0110064.

[195] Liang Li, Tom Greene, and Bo Hu. "A simple method to estimate the time-dependent receiver operating characteristic curve and the area under the curve with right censored data". In: *Statistical Methods in Medical Research* 27.8 (2018), pp. 2264–2278. ISSN: 0962-2802. DOI: 10.1177/0962280216680239. URL: https://doi.org/10.1177/0962280216680239.

[196] Hua Liang and Guohua Zou. "Improved AIC Selection Strategy for Survival Analysis". eng. In: *Computational statistics & data analysis* 52.5 (2008), pp. 2538–2548. ISSN: 0167-9473. DOI: 10.1016/j.csda.2007.09.003. URL: https://www.ncbi.nlm.nih.gov/pubmed/19158943https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2344147/.

[197] Knut Liestol, Per Kragh Andersen, and Ulrich Andersen. "Survival analysis and neural nets". In: *Statistics in Medicine* 13.12 (1994), pp. 1189–1200. ISSN: 0277-6715. DOI: 10.1002/sim.4780131202. URL: https://doi.org/10.1002/sim.4780131202.

[198] Dahua Lin et al. *JuliaStats/Distributions.jl: a Julia package for probability distributions and associated functions*. 2019. DOI: 10.5281/zenodo.2647458.

[199] K D Lindor et al. "Ursodeoxycholic acid in the treatment of primary biliary cirrhosis." eng. In: *Gastroenterology* 106.5 (1994), pp. 1284–1290. ISSN: 0016-5085 (Print). DOI: 10.1016/0016-5085(94)90021-3.

[200] C L Loprinzi et al. "Prospective evaluation of prognostic variables from patient-completed questionnaires. North Central Cancer Treatment Group." eng. In: *Journal of clinical oncology : official journal of the American Society of Clinical Oncology* 12.3 (1994), pp. 601–607. ISSN: 0732-183X (Print). DOI: 10.1200/JCO.1994.12.3.601.

[201] Grace L Lu-Yao et al. "Outcomes of localized prostate cancer following conservative management". In: *Jama* 302.11 (2009), pp. 1202–1209. ISSN: 0098-7484.

[202] Scott M Lundberg and Su-In Lee. "A Unified Approach to Interpreting Model Predictions". In: *Advances in Neural Information Processing Systems* 30 (2017).

[203] M Lundin et al. "Artificial Neural Networks Applied to Survival Prediction in Breast Cancer". In: *Oncology* 57.4 (1999), pp. 281–286. ISSN: 0030-2414. DOI: 10.1159/000012061. URL: https://www.karger.com/DOI/10.1159/000012061.

[204] James T. Luxhoj and Huan Jyh Shyur. "Comparison of proportional hazards models and neural networks for reliability estimation". In: *Journal of Intelligent Manufacturing* 8.3 (1997), pp. 227–234. ISSN: 09565515. DOI: 10.1023/A:1018525308809.

[205] Shuangge Ma and Jian Huang. "Regularized ROC method for disease classification and biomarker selection with microarray data". In: *Bioinformatics (Oxford, England)* 21 (2006), pp. 4356–4362. DOI: 10.1093/bioinformatics/bti724.

[206] D R Mani et al. "Statistics and data mining techniques for lifetime value modeling". In: *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*. 1999, pp. 94–103.

[207] N. Mantel, N. R. Bohidar, and J. L. Ciminera. "Mantel-Haenszel analyses of litter-matched time to response data, with modifications for recovery of interlitter information." In: *Cancer Research* 37 (1977), pp. 3863–3868.

[208] L Mariani et al. "Prognostic factors for metachronous contralateral breast cancer: A comparison of the linear Cox regression model and its artificial neural network extension". In: *Breast Cancer Research and Treatment* 44.2 (1997), pp. 167–178. ISSN: 1573-7217. DOI: 10.1023/A:1005765403093. URL: https://doi.org/10.1023/A:1005765403093.

[209] Bilal Mateen and Raphael Sonabend. "All I want for Christmas is... Rigorous validation of predictive models to prevent hasty generalisations". In: *Significance* 16.6 (2019), pp. 20–24. ISSN: 1740-9705. DOI: 10.1111/j.1740-9713.2019.01336.x. URL: https://doi.org/10.1111/j.1740-9713.2019.01336.x.

[210] Bilal A Mateen et al. "The role of impulsivity in neurorehabilitation: A prospective cohort study of a potential cognitive biomarker for fall risk?" In: *Journal of Neuropsychology* n/a.n/a (2020). ISSN: 1748-6645. DOI: https://doi.org/10.1111/jnp.12239. URL: https://doi.org/10.1111/jnp.12239.

[211] Andreas Mayr, Benjamin Hofner, and Matthias Schmid. "Boosting the discriminatory power of sparse survival models via optimization of the concordance index and stability selection". In: *BMC Bioinformatics* 17.1 (2016), p. 288. ISSN: 1471-2105. DOI: 10.1186/s12859-016-1149-8. URL: https://doi.org/10.1186/s12859-016-1149-8.

[212] Andreas Mayr and Matthias Schmid. "Boosting the concordance index for survival data–a unified framework to derive and evaluate biomarker combinations". eng. In: *PloS one* 9.1 (2014), e84483–e84483. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0084483. URL: https://pubmed.ncbi.nlm.nih.gov/24400093https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3882229/.

[213] Scott Mayer McKinney et al. "International evaluation of an AI system for breast cancer screening". In: *Nature* 577.7788 (2020), pp. 89–94. ISSN: 1476-4687. DOI: 10.1038/s41586-019-1799-6. URL: https://doi.org/10.1038/s41586-019-1799-6.

[214] Nicolai Meinshausen and Peter Bühlmann. "Stability selection". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 72.4 (2010), pp. 417–473. ISSN: 1369-7412. DOI: 10.1111/j.1467-9868.2010.00740.x. URL: https://doi.org/10.1111/j.1467-9868.2010.00740.x.

[215] Olaf Mersmann. *microbenchmark: Accurate Timing Functions*. 2019. URL: https://cran.r-project.org/package=microbenchmark.

[216] David Meyer and Kurt Hornik. "Generalized and Customizable Sets in R". In: *Journal of Statistical Software* 31.2 (2009), pp. 1–27. DOI: 10.18637/jss.v031.i02.

[217] Ulla B Mogensen, Hemant Ishwaran, and Thomas A Gerds. *Evaluating Random Forests for Survival Analysis using Prediction Error Curves*. 2014.

[218] Bijan Moghimi-dehkordi et al. "Statistical Comparison of Survival Models for Analysis of Cancer Data". In: *Asian Pacific Journal of Cancer Prevention* 9 (2008), pp. 417–420. ISSN: 1513-7368.

[219] Christoph Molnar. *Interpretable Machine Learning*. 2019. URL: https://christophm.github.io/interpretable-ml-book/.

[220] Allan H Murphy. "A New Vector Partition of the Probability Score". English. In: *Journal of Applied Meteorology and Climatology* 12.4 (1973), pp. 595–600. DOI: 10.1175/1520-0450(1973)012<0595:ANVPOT>2.0.CO;2. URL: https://journals.ametsoc.org/view/journals/apme/12/4/1520-0450{_}1973{_}012{_}0595{_}anvpot{_}2{_}0{_}co{_}2.xml.

[221] *Myriad technical specs*. URL: http://wiki.rc.ucl.ac.uk/index.php/RC{_}Systems{\#}Myriad{_}technical{_}specs (visited on 03/07/2021).

[222] W N. Venables and B D. Ripley. *Modern Applied Statistics with S*. Springer, 2002. URL: http://www.stats.ox.ac.uk/pub/MASS4.

[223] Claude Nadeau and Yoshua Bengio. "Inference for the Generalization Error". In: *Machine Learning* 52.3 (2003), pp. 239–281. ISSN: 1573-0565. DOI: 10.1023/A:1024068626366. URL: https://doi.org/10.1023/A:1024068626366.

[224] Vinod Nair and Geoffrey E Hinton. "Rectified linear units improve restricted boltzmann machines". In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, pp. 807–814.

[225] Justine B Nasejje et al. "A comparison of the conditional inference survival forest model to random survival forests based on a simulation study as well as on two applications with time-to-event data". eng. In: *BMC medical research methodology* 17.1 (2017), p. 115. ISSN: 1471-2288. DOI: 10.1186/s12874-017-0383-8. URL: https://www.ncbi.nlm.nih.gov/pubmed/28754093https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5534080/.

[226] Wayne Nelson. "Theory and Applications of Hazard Plotting for Censored Failure Data". In: *Technometrics* 14.4 (1972), pp. 945–966.

[227] Roger B Newson. "Comparing the predictive power of survival models using Harrell's c or Somers' D". In: *The Stata Journal* ii (1983), pp. 1–19.

[228] Milad Zafar Nezhad et al. "A Deep Active Survival Analysis approach for precision treatment recommendations: Application of prostate cancer". In: *Expert Systems with Applications* 115 (2019), pp. 16–26. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2018.07.070. URL: http://www.sciencedirect.com/science/article/pii/S0957417418304949.

[229] Ryan Ng et al. "The current application of the Royston-Parmar model for prognostic modeling in health research: a scoping review". In: *Diagnostic and Prognostic Research* 2.1 (2018), p. 4. ISSN: 2397-7523. DOI: 10.1186/s41512-018-0026-5. URL: https://diagnprognres.biomedcentral.com/articles/10.1186/s41512-018-0026-5.

[230] Sung Eun Oh et al. "Prediction of Overall Survival and Novel Classification of Patients with Gastric Cancer Using the Survival Recurrent Network". In: *Annals of Surgical Oncology* 25.5 (2018), pp. 1153–1159. ISSN: 1534-4681. DOI: 10.1245/s10434-018-6343-7. URL: https://doi.org/10.1245/s10434-018-6343-7.

[231] L Ohno-Machado. "Modeling medical prognosis: survival analysis techniques." eng. In: *Journal of biomedical informatics* 34.6 (2001), pp. 428–439. ISSN: 1532-0464 (Print). DOI: 10.1006/jbin.2002.1038.

[232] Lucila Ohno-Machado. *Medical applications of artificial neural networks: connectionist models of survival*. 1996.

[233] Lucila Ohno-Machado. "A COMPARISON OF COX PROPORTIONAL HAZARDS AND ARTIFICIAL NEURAL NETWORK MODELS FOR MEDICAL PROGNOSIS The theoretical advantages and disadvantages of using different methods for predicting survival have seldom been tested in real data sets [1 , 2]. Althou". In: *Comput. Biol. Med* 27.1 (1997), pp. 55–65.

[234] Katie Patel, Richard Kay, and Lucy Rowell. "Comparing proportional hazards and accelerated failure time models: An application in influenza". In: *Pharmaceutical Statistics* 5.3 (2006), pp. 213–224. ISSN: 15391604. DOI: 10.1002/pst.213.

[235] Sebastian P\"olsterl. "scikit-survival: A Library for Time-to-Event Analysis Built on Top of scikit-learn". In: *Journal of Machine Learning Research* 21.212 (2020), pp. 1–6. URL: http://jmlr.org/papers/v21/20-729.html.

[236] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[237] Michael J. Pencina, Ralph B. D'Agostino, and Linye Song. "Quantifying discrimination of Framingham risk functions with different survival C statistics". In: *Statistics in Medicine* 31.15 (2012), pp. 1543–1553. ISSN: 02776715. DOI: 10.1002/sim.4508. arXiv: NIHMS150003.

[238] Andrea Peters and Torsten Hothorn. *ipred: Improved Predictors*. 2019. URL: https://cran.r-project.org/package=ipred.

[239] Eric C Polley and Mark J Van Der Laan. "Super learner in prediction". In: (2010).

[240] Sergej Potapov, Werner Adler, and Matthias Schmid. *survAUC: Estimators of prediction accuracy for time-to-event data.* 2012.

[241] Paolo Emilio Puddu and Alessandro Menotti. "Artificial neural networks versus proportional hazards Cox models to predict 45-year all-cause mortality in the Italian Rural Areas of the Seven Countries Study". In: *BMC Medical Research Methodology* 12.1 (2012), p. 100. ISSN: 1471-2288. DOI: 10.1186/1471-2288-12-100. URL: https://bmcmedresmethodol.biomedcentral.com/articles/10.1186/1471-2288-12-100.

[242] Hein Putter. *dynpred: Companion Package to "Dynamic Prediction in Clinical Survival Analysis"*. 2015. URL: https://cran.r-project.org/package=dynpred.

[243] Jiezhi Qi. "Comparison of Proportional Hazards and Accelerated Failure Time Models". PhD thesis. 2009.

[244] Cox R. and Snell J. "A General Definition of Residuals". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 30.2 (1968), pp. 248–275.

[245] R Core Team. *R: A Language and Environment for Statistical Computing*. Vienna, 2017.

[246] M. Shafiqur Rahman et al. "Review and evaluation of performance measures for survival prediction models in external validation settings". In: *BMC Medical Research Methodology* 17.1 (2017), pp. 1–15. ISSN: 14712288. DOI: 10.1186/s12874-017-0336-2.

[247] C. E. Rasmussen and C. K. I. Williams. *Gaussian processes for machine learning*. Vol. 14. 2. 2004, pp. 69–106. ISBN: 026218253X. DOI: 10.1142/S0129065704001899. arXiv: 026218253X.

[248] Nancy Reid. "A Conversation with Sir David Cox". In: *Statistical Science* 9.3 (1994), pp. 439–455. ISSN: 00905364. DOI: 10.1214/aos/1176348654. arXiv: arXiv:1011.1669v3. URL: http://www.jstor.org/stable/2238700{\%}5Cnhttp://projecteuclid.org/euclid.aoms/1177705148.

[249] Greg Ridgeway. "The state of boosting". In: *Computing Science and Statistics* 31 (1999), pp. 172–181.

[250] Carl Rietschel, Jinsung Yoon, and Mihaela van der Schaar. "Feature Selection for Survival Analysis with Competing Risks using Deep Learning". In: *arXiv preprint arXiv:1811.09317* (2018).

[251] Brian D Ripley and Ruth M Ripley. "Neural networks as statistical methods in survival analysis". In: *Clinical Applications of Artificial Neural Networks*. Ed. by Richard Dybowski and Vanya Gant. Cambridge: Cambridge University Press, 2001, pp. 237–255. ISBN: 9780521662710. DOI: DOI:10.1017/CBO9780511543494.011. URL: https://www.cambridge.org/core/books/clinical-applications-of-artificial-neural-networks/neural-networks-as-statistical-methods-in-survival-analysis/6AC01B644586FE2EF1D34B6A59CC183E.

[252] R M Ripley, A L Harris, and L Tarassenko. "Neural network models for breast cancer prognosis". In: *Neural Computing & Applications* 7.4 (1998), pp. 367–375. ISSN: 1433-3058. DOI: 10.1007/BF01428127. URL: https://doi.org/10.1007/BF01428127.

[253] P Royston. "The Lognormal Distribution as a Model for Survival Time in Cancer, With an Emphasis on Prognostic Factors". In: *Statistica Neerlandica* 55.1 (2001), pp. 89–104. ISSN: 0039-0402. DOI: 10.1111/1467-9574.00158. URL: https://doi.org/10.1111/1467-9574.00158.

[254] Patrick Royston and Douglas G. Altman. "External validation of a Cox prognostic model: Principles and methods". In: *BMC Medical Research Methodology* 13.1 (2013). ISSN: 14712288. DOI: 10.1186/1471-2288-13-33.

[255] Patrick Royston, Mahesh K B Parmar, and Douglas G Altman. "Visualizing Length of Survival in Time-to-Event Studies: A Complement to Kaplan–Meier Plots". In: *JNCI: Journal of the National Cancer Institute* 100.2 (2008), pp. 92–97. ISSN: 0027-8874. DOI: 10.1093/jnci/djm265. URL: https://doi.org/10.1093/jnci/djm265.

[256] Patrick Royston and Mahesh K.B. Parmar. "Flexible parametric proportional-hazards and proportional-odds models for censored survival data, with application to prognostic modelling and estimation of treatment effects". In: *Statistics in Medicine* 21.15 (2002), pp. 2175–2197. ISSN: 02776715. DOI: 10.1002/sim.1203.

[257] Patrick Royston and Willi Sauerbrei. "A new measure of prognostic separation in survival data". In: *Statistics in Medicine* 23.5 (2004), pp. 723–748. ISSN: 02776715. DOI: 10.1002/sim.1621.

[258] Peter Ruckdeschel et al. *S4 Classes for Distributions*. 2006. URL: https://cran.r-project.org/package=distr.

[259] Lukas Sablica and Kurt Hornik. "mistr: A Computational Framework for Mixture and Composite Distributions". In: *The R Journal* 12.1 (2020), p. 283. ISSN: 2073-4859. DOI: 10.32614/RJ-2020-003. URL: https://journal.r-project.org/archive/2020/RJ-2020-003/index.html.

[260] Andreas Sashegyi and David Ferry. "On the Interpretation of the Hazard Ratio and Communication of Survival Benefit". eng. In: *The oncologist* 22.4 (2017), pp. 484–486. ISSN: 1549-490X. DOI: 10.1634/theoncologist.2016-0198. URL: https://pubmed.ncbi.nlm.nih.gov/28314839https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5388384/.

[261] Alan D Saul. *Gaussian Process Based Approaches for Survival Analysis*. 2016.

[262] Michael Schemper and Robin Henderson. "Predictive Accuracy and Explained Variation in Cox Regression". In: *Biometrics* 56 (2000), pp. 249–255. ISSN: 02776715. DOI: 10.1002/sim.1486.

[263] Matthias Schmid and Torsten Hothorn. "Flexible boosting of accelerated failure time models". In: *BMC bioinformatics* 9 (2008), p. 269. DOI: 10.1186/1471-2105-9-269.

[264] Matthias Schmid and Torsten Hothorn. "Boosting additive models using component-wise P-splines". In: *Computational Statistics & Data Analysis* 53.2 (2008), pp. 298–311. ISSN: 0167-9473.

[265] Matthias Schmid and Sergej Potapov. "A comparison of estimators to evaluate the discriminatory power of time-to-event models". In: *Statistics in Medicine* 31.23 (2012), pp. 2588–2609. ISSN: 02776715. DOI: 10.1002/sim.5464.

[266] Matthias Schmid et al. "A Robust Alternative to the Schemper-Henderson Estimator of Prediction Error". In: *Biometrics* 67.2 (2011), pp. 524–535. ISSN: 0006341X. DOI: 10.1111/j.1541-0420.2010.01459.x.

[267] Gideon Schwarz. "Estimating the Dimension of a Model". In: *The Annals of Statistics* 6.2 (1978), pp. 461–464. ISSN: 0090-5364. DOI: 10.1214/aos/1176344136. arXiv: arXiv:1011.1669v3. URL: http://projecteuclid.org/euclid.aos/1176344136.

[268] Guido Schwarzer, Werner Vach, and Martin Schumacher. "On the misuses of artificial neural networks for prognostic and diagnostic classification in oncology". In: *Statistics in Medicine* 19.4 (2000), pp. 541–561. ISSN: 0277-6715. DOI: 10.1002/(SICI)1097-0258(20000229)19:4<541::AID-SIM355>3.0.CO;2-V. URL: https://pubmed.ncbi.nlm.nih.gov/10694735/.

[269] Mark Robert Segal. "Regression Trees for Censored Data". In: *Biometrics* 44.1 (1988), pp. 35–47.

[270] H Seker et al. "An artificial neural network based feature evaluation index for the assessment of clinical factors in breast cancer survival analysis". In: *IEEE CCECE2002. Canadian Conference on Electrical and Computer Engineering. Conference Proceedings (Cat. No.02CH37373)*. Vol. 2. 2002, 1211–1215 vol.2. ISBN: 0840-7789 VO - 2. DOI: 10.1109/CCECE.2002.1013121.

[271] Huseyin Seker et al. "Assessment of nodal involvement and survival analysis in breast cancer patients using image cytometric data: statistical, neural network and fuzzy approaches". eng. In: *Anticancer research* 22.1A (2002), pp. 433–438. ISSN: 0250-7005. URL: http://europepmc.org/abstract/MED/12017328.

[272] Han-Tai Shiao and Vladimir Cherkassky. "SVM-based approaches for predictive modeling of survival data". In: *Proceedings of the International Conference on Data Mining (DMIN)*. The Steering Committee of The World Congress in Computer Science, Computer ..., 2013, p. 1.

[273] Pannagadatta K. Shivaswamy, Wei Chu, and Martin Jansche. "A support vector approach to censored targets". In: *Proceedings - IEEE International Conference on Data Mining, ICDM*. 2007, pp. 655–660. ISBN: 0769530184. DOI: 10.1109/ICDM.2007.93.

[274] Raphael Sonabend. *R62S3: Automatic Method Generation from R6*. 2019. URL: https://cran.r-project.org/package=R62S3.

[275] Raphael Sonabend. *survivalmodels: Models for Survival Analysis*. 2020. URL: https://cran.r-project.org/package=survivalmodels.

[276] Raphael Sonabend. *param6: A Fast and Lightweight R6 Parameter Interface*. 2021. URL: https://cran.r-project.org/package=param6.

[277] Raphael Sonabend and Franz Kiraly. "distr6: The Complete R6 Probability Distributions Interface". In: *The R Journal* (2021). arXiv: 2009.02993. URL: https://cran.r-project.org/package=distr6.

[278] Raphael Sonabend and Franz J. Kiraly. "set6: R6 Mathematical Sets Interface". In: *Journal of Open Source Software* 5.55 (2020), p. 2598. ISSN: 2475-9066. DOI: 10.21105/joss.02598. URL: https://cran.r-project.org/package=set6.

[279] Raphael Sonabend and Florian Pfisterer. *mlr3benchmark: Benchmarking analysis for 'mlr3'*. 2020. URL: https://cran.r-project.org/package=mlr3benchmark.

[280] Raphael Sonabend and Patrick Schratz. *mlr3extralearners: Extra Learners For mlr3*. 2020. URL: https://github.com/mlr-org/mlr3extralearners.

[281] Raphael Sonabend et al. "mlr3proba: An R Package for Machine Learning in Survival Analysis". In: *Bioinformatics* (2021). ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btab039. URL: https://cran.r-project.org/package=mlr3proba.

[282] Xiao Song and Xiao-Hua Zhou. "A semiparametric approach for the covariate specific ROC curve with survival outcome". In: *Statistica Sinica* 18 (2008), pp. 947–965.

[283] Annette Spooner et al. "A comparison of machine learning methods for survival analysis of high-dimensional clinical data for dementia prediction". In: *Scientific Reports* 10.1 (2020), p. 20410. ISSN: 2045-2322. DOI: 10.1038/s41598-020-77220-w. URL: https://doi.org/10.1038/s41598-020-77220-w.

[284] Spotswood L Spruance et al. "Hazard ratio in clinical trials". eng. In: *Antimicrobial agents and chemotherapy* 48.8 (2004), pp. 2787–2792. ISSN: 0066-4804. DOI: 10.1128/AAC.48.8.2787-2792.2004. URL: https://pubmed.ncbi.nlm.nih.gov/15273082https://www.ncbi.nlm.nih.gov/pmc/articles/PMC478551/.

[285] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958. ISSN: 1532-4435.

[286] Mikis Stasinopoulos et al. *gamlss.add: Extra Additive Terms for Generalized Additive Models for Location Scale and Shape*. 2020. URL: https://cran.r-project.org/package=gamlss.add.

[287] Michael B Steinberg et al. "Triple-combination pharmacotherapy for medically ill smokers: a randomized trial". In: *Annals of internal medicine* 150.7 (2009), pp. 447–454. ISSN: 0003-4819.

[288] W Nick Street. "A Neural Network Model for Prognostic Prediction." In: *Proceedings of the Fifteenth International Conference on Machine Learning*. San Francisco, 1998.

[289] The Benelux C M L Study Group. "Randomized Study on Hydroxyurea Alone Versus Hydroxyurea Combined With Low-Dose Interferon-α2b for Chronic Myeloid Leukemia". In: *Blood* 91.8 (1998), pp. 2713–2721. ISSN: 1528-0020. DOI: 10.1182/blood.V91.8.2713.2713_2713_2721. URL: https://doi.org/10.1182/blood.V91.8.2713.2713{_}2713{_}2721https://ashpublications.org/blood/article/91/8/2713/107615/Randomized-Study-on-Hydroxyurea-Alone-Versus.

[290] The Diabetic Retinopathy Study Research Group. "Preliminary report on effects of photocoagulation therapy." eng. In: *American journal of ophthalmology* 81.4 (1976), pp. 383–396. ISSN: 0002-9394 (Print). DOI: 10.1016/0002-9394(76)90292-0.

[291] Terry M. Therneau. *A Package for Survival Analysis in S*. 2015. URL: https://cran.r-project.org/package=survival.

[292] Terry M. Therneau and Beth Atkinson. *rpart: Recursive Partitioning and Regression Trees*. 2019.

[293] Terry M. Therneau and Elizabeth Atkinson. *Concordance*. 2020. URL: https://cran.r-project.org/web/packages/survival/vignettes/concordance.pdf (visited on 06/02/2020).

[294] Terry M. Therneau and Patricia M. Grambsch. *Modeling Survival Data: Extending the Cox Model*. New York, 2000. ISBN: 0-387-98784-3.

[295] Terry M. Therneau, Patricia M. Grambsch, and Thomas R. Fleming. "Martingale-based residuals for survival models". In: *Biometrika* 77.1 (1990), pp. 147–160. ISSN: 00063444. DOI: 10.1093/biomet/77.1.147.

[296] Grigorios Tsoumakas and Ioannis Katakis. "Multi-Label Classification: An Overview". In: *International Journal of Data Warehousing and Mining* 3.3 (2007), pp. 1–13. ISSN: 1548-3924. DOI: 10.4018/jdwm.2007070101. URL: http://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/jdwm.2007070101.

[297] Gerhard Tutz and Harald Binder. "Boosting Ridge Regression". In: *Computational Statistics & Data Analysis* 51 (2007), pp. 6044–6059. DOI: 10.1016/j.csda.2006.11.041.

[298] Gerhard Tutz and Matthias Schmid. *Modeling Discrete Time-to-Event Data*. Springer Series in Statistics. Cham: Springer International Publishing, 2016. ISBN: 978-3-319-28156-8. DOI: 10.1007/978-3-319-28158-2. URL: http://link.springer.com/10.1007/978-3-319-28158-2.

[299] Hajime Uno et al. "Evaluating Prediction Rules for t-Year Survivors with Censored Regression Models". In: *Journal of the American Statistical Association* 102.478 (2007), pp. 527–537. ISSN: 01621459. URL: http://www.jstor.org/stable/27639883.

[300] Hajime Uno et al. "On the C-statistics for Evaluating Overall Adequacy of Risk Prediction Procedures with Censored Survival Data". In: *Statistics in Medicine* 30.10 (2011), pp. 1105–1117. ISSN: 02776715. DOI: 10.1002/sim.4154. arXiv: NIHMS150003.

[301] Kevin Ushey, J J Allaire, and Yuan Tang. *reticulate: Interface to 'Python'*. 2020. URL: https://cran.r-project.org/package=reticulate.

[302] V Van Belle et al. "Additive survival least-squares support vector machines". In: *Statistics in Medicine* 29.2 (2010), pp. 296–308. ISSN: 0277-6715. DOI: 10.1002/sim.3743. URL: https://doi.org/10.1002/sim.3743.

[303] Vanya Van Belle et al. "Support Vector Machines for Survival Analysis". In: *In Proceedings of the Third International Conference on Computational Intelligence in Medicine and Healthcare*. 1. 2007. DOI: 10.1016/j.microrel.2005.05.002.

[304] Vanya Van Belle et al. "Survival SVM: a practical scalable algorithm". In: *Proceedings of the 16th European Symposium on Artificial Neural Networks (ESANN)*. 2008, pp. 89–94.

[305] Vanya Van Belle et al. "Learning Transformation Models for Ranking and Survival Analysis". In: *Journal of Machine Learning Research* 12 (2011), pp. 819–862. ISSN: 15324435.

[306] Vanya Van Belle et al. "Support vector methods for survival analysis: A comparison between ranking and regression approaches". In: *Artificial Intelligence in Medicine* 53.2 (2011), pp. 107–118. ISSN: 09333657. DOI: 10.1016/j.artmed.2011.06.006. URL: http://dx.doi.org/10.1016/j.artmed.2011.06.006.

[307] Marc J Van De Vijver et al. "A gene-expression signature as a predictor of survival in breast cancer". In: *New England Journal of Medicine* 347.25 (2002), pp. 1999–2009. ISSN: 0028-4793.

[308] Hans C. Van Houwelingen. "Validation, calibration, revision and combination of prognostic survival models". In: *Statistics in Medicine* 19.24 (2000), pp. 3401–3415. ISSN: 02776715. DOI: 10.1002/1097-0258(20001230)19:24<3401::AID-SIM554>3.0.CO;2-2.

[309] Hans C. Van Houwelingen. "Dynamic prediction by landmarking in event history analysis". In: *Scandinavian Journal of Statistics* 34.1 (2007), pp. 70–85. ISSN: 03036898. DOI: 10.1111/j.1467-9469.2006.00529.x.

[310] J C Van Houwelingen et al. "Predictability of the survival of patients with advanced ovarian cancer." In: *Journal of Clinical Oncology* 7.6 (1989), pp. 769–773. ISSN: 0732-183X.

[311] Vladimir Vapnik. *The Nature of Statistical Learning Theory*. 1998. ISBN: 978-0-387-94559-0.

[312] Aki Vehtari and Heikki Joensuu. *A Gaussian processes model for survival analysis with time dependent covariates and interval censoring*. 2013. URL: https://users.aalto.fi/{~}ave/VehtariJoensuu{_}GIST{_}CT{_}timing{_}poster{_}2013.pdf (visited on 04/20/2020).

[313] Bhanukiran Vinzamuri, Yan Li, and Chandan K. Reddy. "Pre-processing censored survival data using inverse covariance matrix based calibration". In: *IEEE Transactions on Knowledge and Data Engineering* 29.10 (2017), pp. 2111–2124. ISSN: 10414347. DOI: 10.1109/TKDE.2017.2719028.

[314] David M Vock et al. "Adapting machine learning techniques to censored time-to-event health record data: A general-purpose approach using inverse probability of censoring weighting". In: *Journal of Biomedical Informatics* 61 (2016), pp. 119–131. ISSN: 1532-0464. DOI: https://doi.org/10.1016/j.jbi.2016.03.009. URL: http://www.sciencedirect.com/science/article/pii/S1532046416000496.

[315] Chris T Volinsky and Adrian E Raftery. "Bayesian Information Criterion for Censored Survival Models". In: *International Biometric Society* 56.1 (2000), pp. 256–262.

[316] Hong Wang and Gang Li. "A Selective Review on Random Survival Forests for High Dimensional Data". eng. In: *Quantitative bio-science* 36.2 (2017), pp. 85–96. ISSN: 2508-7185. DOI: 10.22283/qbs.2017.36.2.85. URL: https://pubmed.ncbi.nlm.nih.gov/30740388https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6364686/.

[317] Ping Wang, Yan Li, and Chandan K. Reddy. "Machine Learning for Survival Analysis: A Survey". In: *ACM Computing Surveys* 1.1 (2017). arXiv: arXiv:1708.04649v1.

[318] Zhu Wang. *bujar: Buckley-James Regression for Survival Data with High-Dimensional Covariates*. 2019. URL: https://cran.r-project.org/package=bujar.

[319] Zhu Wang and C Y Wang. "Buckley-James Boosting for Survival Analysis with High-Dimensional Biomarker Data". English. In: *Statistical Applications in Genetics and Molecular Biology* 9.1 (2010). DOI: https://doi.org/10.2202/1544-6115.1550. URL: https://www.degruyter.com/view/journals/sagmb/9/1/article-sagmb.2010.9.1.1550.xml.xml.

[320] L J Wei. "The Accelerated Failure Time Model: A Useful Alternative to the Cox Regression Model in Survival Analysis". In: *Statistics in Medicine* 11 (1992), pp. 1871–1879.

[321] Thomas Welchowski and Matthias Schmid. *discSurv: Discrete Time Survival Analysis*. 2019. URL: https://cran.r-project.org/package=discSurv.

[322] Hadley Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN: 978-3-319-24277-4. URL: https://ggplot2.tidyverse.org.

[323] Tymoteusz Wolodzko. *extraDistr: Additional Univariate and Multivariate Distributions*. 2019. URL: https://cran.r-project.org/package=extraDistr.

[324] David H Wolpert. "Stacked generalization". In: *Neural Networks* 5.2 (1992), pp. 241–259. ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(05)80023-1. URL: http://www.sciencedirect.com/science/article/pii/S0893608005800231.

[325] Marvin N. Wright and Andreas Ziegler. "ranger: A Fast Implementation of Random Forests for High Dimensional Data in C++ and R". In: *Journal of Statistical Software* 77.1 (2017), pp. 1–17.

[326] Laure Wynants et al. "Prediction models for diagnosis and prognosis of covid-19: systematic review and critical appraisal". In: *BMJ* 369 (2020), p. m1328. DOI: 10.1136/bmj.m1328. URL: http://www.bmj.com/content/369/bmj.m1328.abstract.

[327] Anny Xiang et al. "Comparison of the performance of neural network methods and Cox regression for censored survival data". In: *Computational Statistics & Data Analysis* 34.2 (2000), pp. 243–257. ISSN: 0167-9473. DOI: https://doi.org/10.1016/S0167-9473(99)00098-5. URL: http://www.sciencedirect.com/science/article/pii/S0167947399000985.

[328] Yanying Yang. "Neural Network Survival Analysis". PhD thesis. Universiteit Gent, 2010, p. 57. ISBN: 9781617796326.

[329] Angeline Yasodhara, Mamatha Bhat, and Anna Goldenberg. *Prediction of New Onset Diabetes after Liver Transplant*. 2018.

[330] Ali Zare et al. "A Comparison between Accelerated Failure-time and Cox Proportional Hazard Models in Analyzing the Survival of Gastric Cancer Patients." In: *Iranian journal of public health* 44.8 (2015), pp. 1095–102. ISSN: 03044556. DOI: 10.1007/s00606-006-0435-8. URL: http://www.ncbi.nlm.nih.gov/pubmed/26587473{\%}0Ahttp://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC4645729.

[331] Yucheng Zhang et al. "CNN-based survival model for pancreatic ductal adenocarcinoma in medical imaging". In: *BMC Medical Imaging* 20.1 (2020), p. 11. ISSN: 1471-2342. DOI: 10.1186/s12880-020-0418-1. URL: https://doi.org/10.1186/s12880-020-0418-1.

[332] Lili Zhao and Dai Feng. "DNNSurv: Deep Neural Networks for Survival Analysis Using Pseudo Values". In: (2020). arXiv: 1908.02337. URL: https://arxiv.org/abs/1908.02337.

[333] Zheng Zhou et al. "Survival Bias Associated with Time-to-Treatment Initiation in Drug Effectiveness Evaluation: A Comparison of Methods". In: *American Journal of Epidemiology* 162.10 (2005), pp. 1016–1023. ISSN: 0002-9262. DOI: 10.1093/aje/kwi307. URL: https://doi.org/10.1093/aje/kwi307.

[334] Wan Zhu et al. "The Application of Deep Learning in Cancer Prognosis Prediction". eng. In: *Cancers* 12.3 (2020), p. 603. ISSN: 2072-6694. DOI: 10.3390/cancers12030603. URL: https://pubmed.ncbi.nlm.nih.gov/32150991https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7139576/.

[335] X Zhu, J Yao, and J Huang. "Deep convolutional neural network for survival analysis with pathological images". In: *2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. 2016, pp. 544–547. ISBN: VO -. DOI: 10.1109/BIBM.2016.7822579.

Appendices

Appendix A

Chapter 5 Pseudo-code

Algorithm 7 Fitting the (C1) compositor (section 5.4.1).
Input Continuous ranking model, M_R. Training data, $\mathcal{D}_0$. Kaplan-Meier estimator, M_S. Model parameters, θ.
Output Fitted ranking model and estimated baseline survival function, $\{\hat{M}_R, S_0\}$.

$\hat{M}_R \leftarrow \texttt{fit}(M_R, \mathcal{D}_0, \theta)$
$S_0 \leftarrow \texttt{fit}(M_S, \mathcal{D}_0)$
return $\{\hat{M}_R, S_0\}$

Algorithm 8 Predicting with the (C1) compositor (section 5.4.1).
Input Fitted continuous ranking model, $\hat{M}_R$. Estimated baseline survival function, S_0. Testing data, $\mathcal{D}_1$. Model parameters, Θ. Model form, F.
Output Composed distribution, ζ.

$\hat{\eta} \leftarrow \texttt{predict}(\hat{M}_R, \mathcal{D}_1, \Theta)$
function $C(t)$
  switch F **do**
    case ph
      return $S_0(t)^{\wedge} \exp(\hat{\eta})$
    case aft
      return $S_0(t/\exp(\hat{\eta}))$
    case po
      return $S_0(t)/(\exp(-\hat{\eta}) + (1 - \exp(-\hat{\eta})) * S_0(t))$
end function
$\zeta.S \leftarrow C$
return ζ

Algorithm 9 Fitting the (C2) compositor (section 5.4.2).
Input Survival time model, M_T. Standard error model, M_S. Training data, $\mathcal{D}_0$. Model parameters, θ_T, θ_S.
Output Fitted survival time and standard error models, $\{\hat{M}_T, \hat{M}_S\}$.

$\hat{M}_T \leftarrow \texttt{fit}(M_T, \mathcal{D}_0, \theta_T)$
$\hat{M}_S \leftarrow \texttt{fit}(M_S, \mathcal{D}_0, \theta_S)$
return $\{\hat{M}_T, \hat{M}_S\}$

Algorithm 10 Predicting with the (C2) compositor (section 5.4.2).
Input Fitted survival time model, $\hat{M}_T$. Fitted standard error model, $\hat{M}_S$. Model parameters, Θ_T, Θ_S. Assumed distribution with respective location-scale parameters: $d(\mu, \sigma)$. Testing data, $\mathcal{D}_1$.
Output Composed distribution, ζ.

$\mu \leftarrow \texttt{predict}(\hat{M}_T, \mathcal{D}_1 | \Theta_T)$
$\sigma \leftarrow \texttt{predict}(\hat{M}_S, \mathcal{D}_1 | \Theta_S)$
$\zeta \leftarrow d(\mu, \sigma)$
return ζ

Algorithm 11 Fitting the (C3) compositor (section 5.4.3).
Input Probabilistic survival model: M. Training data, $\mathcal{D}_0$. Model parameters, θ.
Output Fitted probabilistic survival model, $\hat{M}$.

$\hat{M} \leftarrow \texttt{fit}(M, \mathcal{D}_0, \theta)$
return $\hat{M}$

Algorithm 12 Predicting with the (C3) compositor (section 5.4.3).
Input Fitted probabilistic survival model, $\hat{M}$. Summary method, ϕ. Testing data, $\mathcal{D}_1$. Model parameters, Θ.
Output Composed survival time prediction, $\hat{T}$.

$\zeta \leftarrow \texttt{predict}(\hat{M}, \mathcal{D}_1, \Theta)$
$\hat{T} \leftarrow \phi(\zeta)$
return $\hat{T}$

Algorithm 13 Fitting the (C4) compositor (section 5.4.4).
Input B survival models: $M = \{M_b\}_{b=1}^{B}$. Training data, $\mathcal{D}_0$. Model parameters, $\theta = \{\theta_b\}_{b=1}^{B}$.
Output B fitted survival models, $\hat{M}$.

for $b = 1, ..., B$ **do**
    $\hat{M}_b \leftarrow \texttt{fit}(M_b, \mathcal{D}_0, \theta_b)$
end for
$\hat{M} \leftarrow \{\hat{M}_b\}_{b=1}^{B}$
return $\hat{M}$

Algorithm 14 Predicting with the (C4) compositor (section 5.4.4).
Input B fitted survival models, $\hat{M} = \{\hat{M}_b\}_{b=1}^{B}$. Testing data, $\mathcal{D}_1$. Model parameters, $\Theta = \{\Theta_b\}_{b=1}^{B}$.
Output Averaged survival prediction, ϕ.

for $b = 1, ..., B$ **do**
  $\phi_b \leftarrow \texttt{predict}(\hat{M}_b, \mathcal{D}_1, \Theta_b)$
end for
$\phi \leftarrow \texttt{mean}(\phi_1, ..., \phi_B)$
return ϕ

Appendix B

Full Text of Section 5.5.7.4

This section discusses multi-label classification algorithms and includes a novel 'wrapper' for solving multi-label classification problems with any classifier.

Formally, let $\mathcal{X} \subseteq \mathbb{R}^p$ for p features in a training dataset and let K be the number of labels to predict, then the multi-label classification task is the problem of estimating the function $\hat{g}$,

$$\hat{g} : \mathcal{X} \to [0,1]^K \tag{B.0.1}$$

The models are trained on data $(X_1, Y_1), ..., (X_n, Y_n) \overset{iid}{\sim} (X, Y)$ t.v.i. $\mathcal{X} \times [0,1]^K$.

Multi-label classification models can be grouped into 'problem transformation' methods (PTs) and 'algorithm adaptation methods' (AAs) [296]. PTs solve the multi-class problem by transforming the data into a single-label problem and then utilising off-shelf classification models, they are therefore reductions themselves. AAs adapt the classifiers themselves and few of these algorithms exist off-shelf ($g_L(D_B|\theta)$ in fig. 30); hence PTs are the focus in this section (C_C and C_B in fig. 30). Common PTs are summarised below and then generalised into a single multi-label classification 'wrapper'. For the PTs below that include an example, each assumes the original dataset is the one in table 25 where X is the only feature and $Y1, Y2, Y3$ are three labels to predict.[1]

Table 25: Example multi-label classification dataset.

X	$Y1$	$Y2$	$Y3$
1	1	0	0
2	0	0	0
3	1	1	0

Exclusive Multi-Class **Method:** A new outcome variable, Y^* t.v.i. $\{1, ..., K\}$, is defined such that all K labels are treated as K classes. **Advantages:** Simple to implement. Majority of classifiers can handle multiple categories. **Disadvantages:** Assumes labels are mutually exclusive. Only for binary multi-label

[1] Table 25 is a generic multi-label classification dataset and not one resulting from the survival reduction.

classification. **Example data (Ex.) after transformation (trafo):** Multi-class classification dataset with one label consisting of three possible 'classes', corresponding to the original three labels:

X	Y
1	2
2	1
3	3

Label Power Set Method: A new outcome variable, Y^* t.v.i. $\{1, ..., 2^K\}$, is defined that takes as classes the powerset of the original K labels. **Advantages:** Simple to implement. Does not require original labels to be mutually exclusive. **Disadvantages:** Probabilistic predictions have no meaningful interpretation. Only works if original multi-label dataset has binary classes. **Ex. after trafo:** Multi-class classification dataset with 2^3 classes, corresponding to the labels in which an observation is alive:

X	Y
1	$\{1\}$
2	$\{\}$
3	$\{1 \wedge 2\}$

Binary Relevance Method: The original dataset is replicated into K datasets where the same features are used, but each includes only one of the original outcome labels. **Advantages:** Reduces the problem to multiple single-label binary-class problems. Can be handled by any classifier. **Disadvantages:** Additional run-time and storage requirements as K classifiers are required. Classifiers make independent predictions, which may be unrealistic when labels are correlated. **Ex. after trafo:** Three datasets, corresponding to the probability of event in Y1-3 independently:

X	$Y1$
1	1
2	0
3	1

X	$Y2$
1	0
2	0
3	1

X	$Y3$
1	0
2	0
3	0

Classifier Chains Method: Labels are ordered and for each ordered label to predict, classifiers are trained on features and the true, observed values for all previous labels. **Advantages:** Predictions are not independent as classifier makes use of previous labels. Sophisticated models should learn that if the event can occur only once then once it is observed in a previous label, it is guaranteed for future labels. **Disadvantages:** Each subsequent model makes use of previous labels and so the running time and required storage increases with every model. **Ex. after trafo:** Three datasets with the final column being the label to predict and the others being features and prior labels:

X	$Y1$
1	1
2	0
3	1

X	$Y1$	$Y2$
1	1	0
2	0	0
3	1	1

X	$Y1$	$Y2$	$Y3$
1	1	0	0
2	0	0	0
3	1	1	0

Nested Stacking Almost identical to classifier chains except that instead of using real values from previous labels, the predicted estimates are used instead. This makes it a stacking method as each subsequent model is built using elements of the previous. In terms of model size this is similar to classifier chains but may take longer to fit as the algorithm calls back to previous models.

Dependent Binary Relevance and Stacking The final two methods are similar to classifier chains and nested stacking except that the ordering of labels is not required. Instead every covariate and label – except the target – is included as a feature to train the model. These are not appropriate methods when a natural ordering is present in the data.

In the classification reduction, target labels are correlated, multi-class, and have a natural ordering. Hence exclusive multi-class, label power set, dependent binary relevance and stacking are not appropriate; only binary relevance, classifier chains and nested stacking are recommended for this reduction. Binary relevance cannot be applied to the discrete hazard representation (section 5.5.7.2) composition as doing so will estimate the pmf not hazard. This is the case as the hazard function assumes knowledge about 'the past', i.e. survival up until a given time. Therefore only an algorithm that makes sequential predictions can estimate this conditional quantity.

Classifier chains and nested stacking have the added advantage of taking into account information from previous labels, given some pre-specified ordering, but at the cost of increased time and storage requirements. Table 26 shows the time taken to fit/predict a logistic regression model, using each PT above, to a simulated dataset consisting of 66 observations, one binary feature, one numeric feature, and ten binary, mutually-exclusive, labels. The fastest methods for fit/predict were binary relevance and classifier chains. A fitted binary relevance model was also the smallest (in Mb). The stacking PT was both the longest to fit/predict and one of the largest stored models (double the size of binary relevance).

The LWrapper Each of the PT methods can be generalised into a single 'wrapper' for multi-label classification, which is termed the 'LWrapper'. This is a *wrapper* as any classifier can be 'wrapped' in the LWrapper to form a composite model capable of making multi-label classification predictions. Doing so increases flexibility in user-choice and potential hyper-parameters for tuning.

Let K be the number of labels in the multi-label classification dataset and let $k \in 1, ..., K$ denote a single label to be predicted.

The LWrapper has three hyper-parameters. The first is the 'L' parameter which is a value in $1, ..., k-1$, and corresponds to the number of labels to utilise as covariates. For example if X_i t.v.i. $\mathbb{R}^p$ are p features for observation i and

Table 26: Times to fit/predict logistic regression for each PT, as well as the size of the trained model. PTs incorporating other labels require double the space. Time taken to fit/predict significantly increases with method complexity.

Method	Mean (ms)	cld[1]	Size (Mb)
Binary Relevance	90	a	2.3
Classifier Chains	113	a	2.4
Dependent Binary Relevance	214	b	4.8
Nested Stacking	331	c	2.4
Stacking	437	d	4.8

1. Significance test for run-time of each method where 'a' is fastest and 'd' is slowest. The experiment is conducted on R version 3.6.1; Platform: x86_64-apple-darwin15.6.0 (64-bit); Running under: macOS Mojave 10.14.4 with **mlr** v2.16.0 [22], and **microbenchmark** v1.4-7 [215].

Table 27: Correspondence of Select-l wrapper to PT algorithms.

	Ordered		Not Ordered	
	Stacked	Not Stacked	Stacked	Not Stacked
$l = 0$[2]	BR[1]	BR	BR	BR
$l = k - 1$[3]	NS	CC	ST	DBR

1. Key: Binary Relevance (BR); Nested Stacking (NS); Classifier Chains (CC); Stacking PT (ST); Dependent Binary Relevance (DBR).
2. Select-0 is always equivalent to Binary Relevance (as is Past-0 and First-0).
3. If data is ordered then Select-$(k-1)$ is equivalent to classifier chains and Stacked-Select-$(k-1)$ is equivalent to nested stacking. If data is not ordered then Select-$(k-1)$ is equivalent to dependent binary relevance and Stacked-Select-$(k-1)$ is equivalent to stacking PT.

$L = 2$, then the covariates for prediction are the p features and two of the labels.[1] The second parameter, 'type', corresponds to which L of the labels are utilised as covariates, the options are 'Past', 'First', or 'Select', these are described below. The final parameter, 'stacking', indicates if stacking should be applied to the models.

For notational convenience, below 'Past-l' will refer to the wrapper with the 'past' type and with 'L' set to l. 'Stacked-Past-l' is the wrapper when stacking is additionally applied. Analogously for first and select.

The *Past-l* wrapper predicts the probability of event in each of the K labels individually using the 'previous' l labels for prediction, this assumes a given ordering for the labels. The *First-l* wrapper predicts the probability of event in each of the K labels individually using the 'first' l labels for prediction, again assuming an ordering. The *Select-l* wrapper incorporates variable selection to select l variables. If an ordering is specified then the model will only select from labels 'before' k. Each model can be adapted to *stacking* methods by using 'previous' predictions as features instead of the true observed labels. The PT methods are special cases of the Select-l wrapper (table 27).

[1]The LWrapper extends simply to the time-varying covariates setting by noting that any 'sliding window' approach, which fits and predicts models over time, can be set-up so covariates update over time with the sequentially fitted model. This is not discussed further due to being out of scope of this thesis.

Appendix C

Chapter 4 Proofs

Proof of lemma 4.5.1

Proof. Let $Y := S_X(X)$ and as X is a continuous random variable assume F_X is continuous and non-decreasing and so the quantile function F_X^{-1} exists. Note that F_X is continuous in [0,1]. By transformation of random variables,

$$\begin{aligned} F_Y(x) &= P(Y \leq x) \\ &= P(S_X(X) \leq x) \\ &= P(F_X(X) \geq 1 - x) \\ &= P(X \geq F_X^{-1}(1-x)) \\ &= 1 - P(X \leq F_X^{-1}(1-x)) \\ &= 1 - F_X(F_X^{-1}(1-x)) \\ &= 1 - 1 + x = x \end{aligned}$$

Further note that if $U \sim \mathcal{U}(0,1)$ then $F_U(u) = u$. As U and $Y := S_X(X)$ have the same distribution function, it follows $S_X(X) \sim \mathcal{U}(0,1)$. □

Proof of lemma 4.6.2

Proof.

Proof of (i). By definition L is outcome-independent proper if $\mathbb{E}[L(p_Y, T, \Delta)] \leq \mathbb{E}[L(p, T, \Delta)]$ but this is a contradiction to the statement, hence L is not outcome-independent proper, proving (i). ■

Proof of (ii). By definition L is outcome-independent strictly proper if L is outcome-independent proper and $\mathbb{E}[L(p_Y, T, \Delta)] = \mathbb{E}[L(p, T, \Delta)] \Leftrightarrow p = p_Y$, however by (i) L is not outcome-independent proper and therefore by definition cannot be outcome-independent strictly proper, proving (ii). ■

Proof of (iii). Proof is identical to (i). ■

Proof of (iv). Proof of (iv): By definition L is strictly proper if L is proper and $\mathbb{E}[L(p_Y, T, \Delta)] = \mathbb{E}[L(p, T, \Delta)] \Leftrightarrow p = p_Y$, however by (iii) L is improper and therefore by definition cannot be strictly proper, proving (iv). ■

□

Proof of lemma 4.6.3

Proof.

Proof of (i). By definition L is approximately proper if $\mathbb{E}[L(p_Y, T, \Delta|c)] \leq \mathbb{E}[L(p, T, \Delta|c)]$ but this is a contradiction to the statement, hence L is not approximately proper, proving (i). ■

Proof of (ii). By definition L is approximately strictly proper if L is approximately proper and $\mathbb{E}[L(p_Y, T, \Delta|c)] = \mathbb{E}[L(p, T, \Delta|c)] \Leftrightarrow p = p_Y$, however by (i) L is not approximately proper and therefore by definition cannot be approximately strictly proper, proving (ii). ■

□

Proof of lemma 4.6.4

Proof.

Proof of (i). Proof follows by definition of properness and substituting the expressions defined above. If L is proper then

$$\begin{aligned}
&\mathbb{E}[L(p_Y, T, \Delta)] \leq \mathbb{E}[L(p, T, \Delta)] \\
&\Rightarrow S_L(p_Y, p_Y) \leq S_L(p_Y, p) \\
&\Rightarrow H_L(p_Y) \leq S_L(p_Y, p) \\
&\Rightarrow 0 \leq S_L(p_Y, p) - H_L(p_Y) \\
&\Rightarrow D_L(p_Y, p) \geq 0
\end{aligned}$$

where the second inequality is substituting definition of S_L, the third is substituting definition of H_L, the third is subtracting H_L from both sides, and the final by substituting definition of D_L and reversing the inequality. ■

Proof of (ii). Proof follows similarly to (i) after replacing the inequalities by strict inequalities. ■

□

Proof of lemma 4.6.5

Proof. Proof follows by transformation of random variables via the joint cdf.

The joint cdf of (X, Y) is defined by,

$$F_{X,Y}(x, y) = \int_{-\infty}^{x} \int_{-\infty}^{y} f_{X,Y}(s, t)\ dt\ ds \tag{C.0.1}$$

By definition of indicator variables, $Z = 1$ iff $Y \geq X$ and 0 otherwise and so on substituting Z for Y,

$$\begin{aligned} F_{X,Z}(x, z) &= P(X \leq x, Z \leq z) \\ &= \begin{cases} P(X \leq x), & z = 1 \\ P(X \leq x, Z = 0) = P(X \leq x, Y < X), & z = 0 \end{cases} \end{aligned}$$

where the first case follows as $Z \in \{0, 1\}$ and hence $P(X \leq x, Z \leq 1) = P(X \leq x)$ as Z is marginalised out. The second case follows as $Z \in \{0, 1\}$ and so $P(Z \leq 0) = P(Z = 0)$, and by definition of indicator variables $Z = 0$ iff $Y < X$. Now focusing on the second case,

$$\begin{aligned} P(X \leq x, Y < X) &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f_{X,Y}(t, y) \mathbb{I}_x(t, y)\ dt\ dy \\ &= \int_{-\infty}^{x} \int_{-\infty}^{s} f_{X,Y}(s, y)\ dy\ ds \end{aligned}$$

where $\mathbb{I}_x = \mathbb{I}(a \leq x, b < a)$. The first line follows by definition of joint probabilities and the second by change of notation. Now,

$$F_{X,Z}(x, z) = \begin{cases} F_X(x), & z = 1 \\ \int_{-\infty}^{x} \int_{-\infty}^{s} f_{X,Y}(s, y)\ dy\ ds, & z = 0 \end{cases} \tag{C.0.2}$$

For mixed joint distributions, $F_{X,Y}(x, y) = \sum_{t \leq y} \int_{s=\infty}^{x} f_{X,Y}(s, t)\ ds$, hence the joint density is given by

$$\begin{aligned} f_{X,Z}(x, z) &= \begin{cases} \frac{\partial F_{X,Z}(x,1) - F_{X,Z}(x,0)}{\partial x}, & z = 1 \\ \frac{\partial F_{X,Z}(x,0)}{\partial x}, & z = 0 \end{cases} \\ &= \begin{cases} \frac{\partial}{\partial x} \int_{-\infty}^{x} \int_{-\infty}^{\infty} f_{X,Y}(s, y)\ dy\ ds - \int_{-\infty}^{x} \int_{-\infty}^{s} f_{X,Y}(s, y)\ dy\ ds, & z = 1 \\ \frac{\partial}{\partial x} \int_{-\infty}^{x} \int_{-\infty}^{s} f_{X,Y}(s, y)\ dy\ ds, & z = 0 \end{cases} \\ &= \begin{cases} \frac{\partial}{\partial x} \int_{-\infty}^{x} \int_{-\infty}^{\infty} f_{X,Y}(s, y)\ dy - \int_{-\infty}^{s} f_{X,Y}(s, y)\ dy\ ds, & z = 1 \\ \int_{-\infty}^{x} f_{X,Y}(x, y)\ dy, & z = 0 \end{cases} \end{aligned}$$

$$= \begin{cases} \frac{\partial}{\partial x} \int_{-\infty}^{x} \int_{s}^{\infty} f_{X,Y}(s,y)\ dy\ ds, & z = 1 \\ \int_{-\infty}^{x} f_{X,Y}(x,y)\ dy, & z = 0 \end{cases}$$
$$= \begin{cases} \int_{x}^{\infty} f_{X,Y}(x,y)\ dy, & z = 1 \\ \int_{-\infty}^{x} f_{X,Y}(x,y)\ dy, & z = 0 \end{cases}$$

where the first equality is the definition of the joint mixed pdf in terms of the cdf, the second equality follows by substituting eq. (C.0.2), the third by taking the partial derivative of the second case over x and by linearity of integration in the first case, the fourth by subtracting the inner integrals, and the fifth by taking the partial derivative of the first case over x. The proof is now complete. □

Proof of corollary 4.6.6

Proof. From lemma 4.6.5, for any jointly absolutely continuous random variables, X, Y supported on the Reals and $Z = \mathbb{I}(X \leq Y)$ it holds that,

$$f_{X,Z}(x,z) = \begin{cases} \int_{x}^{\infty} f_{X,Y}(x,y)\ dy, & z = 1 \\ \int_{-\infty}^{x} f_{X,Y}(x,y)\ dy, & z = 0 \end{cases} \tag{C.0.3}$$

If X, Y are independent then $f_{X,Y}(x,y) = f_X(x) f_Y(y)$ by definition of independence. Substituting this result into the above equation,

$$f_{X,Z}(x,z) = \begin{cases} \int_{x}^{\infty} f_X(x) f_Y(y)\ dy, & z = 1 \\ \int_{-\infty}^{x} f_X(x) f_Y(y)\ dy, & z = 0 \end{cases}$$
$$= \begin{cases} f_X(x) \int_{x}^{\infty} f_Y(y)\ dy, & z = 1 \\ f_X(x) \int_{-\infty}^{x} f_Y(y)\ dy, & z = 0 \end{cases}$$
$$= \begin{cases} f_X(x) S_Y(x), & z = 1 \\ f_X(x) F_Y(x), & z = 0 \end{cases}$$

where the first equality holds as X, Y independent, the second by properties of integration, and the third by definition of the cumulative distribution and survival functions. The proof is now complete. □

Proof of proposition 4.6.12

Proof.

Proof of (i). Let $\mathcal{P}$ be a family of absolutely continuous distributions over the positive Reals and let ζ, ξ be some distributions in $\mathcal{P}$. Let Y be some random variable distributed according to ξ and let C be an r.v. t.v.i. $\mathcal{T}$ and let Y and

C be independent. Let $T := \min\{Y, C\}$ and $\Delta := \mathbb{I}(T = Y)$. Let $\hat{Y}$ be some random variable distributed according to ζ, independent of Y, C, T, and Δ.

By lemma 4.6.4, the loss is improper if there exists some ζ, ξ such that $D_{SDLL}(\xi, \zeta) < 0$. Proof follows by demonstrating such ζ, ξ exist and therefore that the loss is improper. First calculating $S_{IGS}(\xi, \zeta)$,

$$
\begin{aligned}
S_{IGS}(\xi, \zeta) &= \mathbb{E}\{\int_0^{\tau^*} \frac{S^2_{\hat{Y}}(\tau)\mathbb{I}(T \leq \tau, \Delta = 1)}{\hat{G}_{KM}(T)} + \frac{F^2_{\hat{Y}}(\tau)\mathbb{I}(T > \tau)}{\hat{G}_{KM}(\tau)} \; d\tau\} \\
&= \mathbb{E}[\int_T^{\tau^*} \frac{S^2_{\hat{Y}}(\tau)\Delta}{\hat{G}_{KM}(T)} \; d\tau] + \mathbb{E}[\int_0^{T} \frac{F^2_{\hat{Y}}(\tau)}{\hat{G}_{KM}(\tau)} \; d\tau] \\
&= q\mathbb{E}[\int_T^{\tau^*} \frac{S^2_{\hat{Y}}(\tau)\Delta}{\hat{G}_{KM}(T)} \; d\tau | \Delta = 1] + q\mathbb{E}[\int_0^{T} \frac{F^2_{\hat{Y}}(\tau)}{\hat{G}_{KM}(\tau)} \; d\tau | \Delta = 1] + \\
&\quad (1-q)\mathbb{E}[\int_T^{\tau^*} \frac{S^2_{\hat{Y}}(\tau)\Delta}{\hat{G}_{KM}(T)} \; d\tau | \Delta = 0] + (1-q)\mathbb{E}[\int_0^{T} \frac{F^2_{\hat{Y}}(\tau)}{\hat{G}_{KM}(\tau)} \; d\tau | \Delta = 0] \\
&= q\mathbb{E}[\int_Y^{\tau^*} \frac{S^2_{\hat{Y}}(\tau)}{\hat{G}_{KM}(Y)} \; d\tau | \Delta = 1] + q\mathbb{E}[\int_0^{Y} \frac{F^2_{\hat{Y}}(\tau)}{\hat{G}_{KM}(\tau)} \; d\tau | \Delta = 1] + \\
&\quad (1-q)\mathbb{E}[\int_0^{C} \frac{F^2_{\hat{Y}}(\tau)}{\hat{G}_{KM}(\tau)} \; d\tau | \Delta = 0] \\
&= q \int f_{Y|\Delta}(y|1) \int_y^{\tau^*} \frac{S^2_{\hat{Y}}(\tau)}{\hat{G}_{KM}(y)} \; d\tau dy + q \int f_{Y|\Delta}(y|1) \int_0^{y} \frac{F^2_{\hat{Y}}(\tau)}{\hat{G}_{KM}(\tau)} \; d\tau dy + \\
&\quad (1-q) \int f_{C|\Delta}(c|0) \int_0^{c} \frac{F^2_{\hat{Y}}(\tau)}{\hat{G}_{KM}(\tau)} \; d\tau dc \\
&= \int f_{Y,\Delta}(y, 1) \int_t^{\tau^*} \frac{S^2_{\hat{Y}}(\tau)}{\hat{G}_{KM}(y)} \; d\tau dy + \int f_{Y,\Delta}(y, 1) \int_0^{y} \frac{F^2_{\hat{Y}}(\tau)}{\hat{G}_{KM}(\tau)} \; d\tau dy + \\
&\quad \int f_{C,\Delta}(c, 0) \int_0^{c} \frac{F^2_{\hat{Y}}(\tau)}{\hat{G}_{KM}(\tau)} \; d\tau dc \\
&= \int f_Y(y) S_C(y) \int_y^{\tau^*} \frac{S^2_{\hat{Y}}(\tau)}{\hat{G}_{KM}(y)} \; d\tau dy + \int f_Y(y) S_C(y) \int_0^{y} \frac{F^2_{\hat{Y}}(\tau)}{\hat{G}_{KM}(\tau)} \; d\tau dy + \\
&\quad \int f_C(c) S_Y(c) \int_0^{c} \frac{F^2_{\hat{Y}}(\tau)}{\hat{G}_{KM}(\tau)} \; d\tau dc
\end{aligned}
$$

where the second equality is linearity of integration and expectation, the third is law of total expectation and substituting $q := P(\Delta = 1)$, the fourth is substituting Y for $T|\Delta = 1$, C for $T|\Delta = 0$, the fifth is definition of conditional expectation, the sixth is definition of conditional probability, the seventh is corollary 4.6.6 and lemma 4.6.7. To prove L_{IGS} is not proper, proof continues in the asymptotic as $n \to \infty$ and $\hat{G}_{KM} \to S_C$ [153],

$$S_{IGS}(\xi,\zeta) = \int f_Y(y) \int_y^{\tau^*} S^2_{\hat{Y}}(\tau)\ d\tau dy + \int f_Y(y) S_C(y) \int_0^y \frac{F^2_{\hat{Y}}(\tau)}{S_C(\tau)}\ d\tau dy + \int f_C(c) S_Y(c) \int_0^c \frac{F^2_{\hat{Y}}(\tau)}{S_C(\tau)}\ d\tau dc$$

In addition for this counter-example let $C \sim \xi$ so that the above reduces to,

$$S_{IGS}(\xi,\zeta) = \int f_Y(y) \int_y^{\tau^*} S^2_{\hat{Y}}(\tau)\ d\tau dy + 2\int f_Y(y) S_Y(y) \int_0^y \frac{F^2_{\hat{Y}}(\tau)}{S_Y(\tau)}\ d\tau dy$$

The loss is improper if there is at least one $\zeta \neq \xi \in \mathcal{P}$ s.t. $S_{IGS}(\xi,\zeta) < H_{IGS}(\xi)$. To find such a counter-example let $\xi = \text{Exp}(a)$ and let $\zeta = \text{Exp}(b)$. If some $X \sim \text{Exp}(\lambda)$ then $f_X(x) = \lambda e^{-\lambda x}, S_X(x) = e^{-\lambda x}, S^2_X(x) = e^{-2\lambda x}, F_X(x) = 1 - e^{-\lambda x}, F^2_X(x) = 1 - 2e^{-\lambda x} + e^{-2\lambda x}$. Continuing by substituting these results,

$$\begin{aligned}
S_{IGS}(\xi,\zeta) &\Rightarrow \int (ae^{-ay}) \int_y^{\tau^*} (e^{-2b\tau})\ d\tau dy + 2\int (ae^{-2ay}) \int_0^y \frac{1 - 2e^{-b\tau} + e^{-2b\tau}}{(e^{-a\tau})}\ d\tau dy \\
&= \int (ae^{-ay}) \Big[-\frac{e^{-2b\tau}}{2b} \Big]_y^{\tau^*}\ dy + \\
&\qquad 2\int (ae^{-2ay}) \int_0^y e^{a\tau} - 2e^{\tau(a-b)} + e^{\tau(a-2b)}\ d\tau dy \\
&= \int (ae^{-ay}) \Big[-\frac{e^{-2b\tau}}{2b} \Big]_y^{\tau^*}\ dy + \\
&\qquad 2\int (ae^{-2ay}) \Big[\frac{e^{a\tau}}{a} - \frac{2e^{\tau(a-b)}}{a-b} + \frac{e^{\tau(a-2b)}}{a-2b} \Big]_0^y\ dy \\
&= \int (ae^{-ay}) \Big[-\frac{e^{-2b\tau^*}}{2b} + \frac{e^{-2by}}{2b} \Big]\ dy + \\
&\qquad 2\int (ae^{-2ay}) \Big[(\frac{e^{ay}}{a} - \frac{2e^{y(a-b)}}{a-b} + \frac{e^{y(a-2b)}}{a-2b}) - (\frac{1}{a} - \frac{2}{a-b} + \frac{1}{a-2b}) \Big]\ dy \\
&= \int (ae^{-ay}) \Big[-\frac{e^{-2b\tau^*}}{2b} + \frac{e^{-2by}}{2b} \Big]\ dy + \\
&\qquad 2\int (ae^{-2ay}) \Big[(\frac{e^{ay}}{a} - \frac{2e^{y(a-b)}}{a-b} + \frac{e^{y(a-2b)}}{a-2b}) - \Big(\frac{2b^2}{a(a-b)(a-2b)} \Big) \Big]\ dy \\
&= -\frac{ae^{-2b\tau^*}}{2b} \int e^{-ay} dy + \frac{a}{2b} \int e^{y(-a-2b)} dy - \\
&\qquad \Big(\frac{4b^2}{(a-b)(a-2b)} \Big) \int e^{-2ay} dy + \\
&\qquad 2\int e^{-ay}\ dy - \frac{4a}{a-b} \int (e^{y(-a-b)})\ dy + \frac{2a}{a-2b} \int e^{y(-a-2b)}\ dy
\end{aligned}$$

$$
\begin{aligned}
&= (-\frac{ae^{-2b\tau^*}}{2b} + 2)\int_0^\infty e^{-ay}dy + \frac{a(a+2b)}{2b(a-2b)}\int_0^\infty e^{y(-a-2b)}dy - \\
&\quad \Big(\frac{4b^2}{(a-b)(a-2b)}\Big)\int_0^\infty e^{-2ay}dy - \frac{4a}{a-b}\int_0^\infty (e^{y(-a-b)})\,dy \\
&= (-\frac{ae^{-2b\tau^*}}{2b} + 2)\Big[\frac{e^{-ay}}{-a}\Big]_0^\infty + \frac{a(a+2b)}{2b(a-2b)}\Big[\frac{e^{y(-a-2b)}}{-a-2b}\Big]_0^\infty - \\
&\quad \Big(\frac{4b^2}{(a-b)(a-2b)}\Big)\Big[\frac{e^{-2ay}}{-2a}\Big]_0^\infty - \frac{4a}{a-b}\Big[\frac{e^{y(-a-b)}}{-a-b}\Big]_0^\infty \\
&= (-\frac{ae^{-2b\tau^*}}{2b} + 2)\Big[\frac{1}{a}\Big] + \frac{a(a+2b)}{2b(a-2b)}\Big[\frac{1}{a+2b}\Big] - \Big(\frac{4b^2}{(a-b)(a-2b)}\Big)\Big[\frac{1}{2a}\Big] - \\
&\quad \frac{4a}{a-b}\Big[\frac{1}{a+b}\Big] \\
&= \frac{2}{a} - \frac{e^{-2b\tau^*}}{2b} + \frac{a}{2b(a-2b)} - \frac{2b^2}{a(a-b)(a-2b)} - \frac{4a}{(a+b)(a-b)}
\end{aligned}
$$

Where the second equality is integration of the first inner integral and expanding the second inner integral, the third equality is integration of the second inner integral, the fourth equality is substitution of limits, the fifth and sixth are simplification of the fractions, the seventh is further simplification and specifying the bounds of integration, the eighth is integration, the ninth is substitution of limits, and the tenth is expansion.

Proof that the loss is not strictly proper only requires a single counter-example, hence for convenience select $\tau^* = 1$, which simplifies the above as follows

$$
\begin{aligned}
S_{IGS}(\xi,\zeta) \Rightarrow \frac{2}{a} - \frac{e^{-2b}}{2b} + \frac{a}{2b(a-2b)} - \frac{2b^2}{a(a-b)(a-2b)} - \frac{4a}{(a+b)(a-b)} \\
= \frac{a^2 - ae^{-2b}(a+b) - ab + 2b^2}{2ab(a+b)}
\end{aligned}
$$

To demonstrate the loss is improper let $a = 2$ and first let $b = a = 2$, then

$$
\begin{aligned}
H_{IGS}(\xi) &= \frac{a^2 - ae^{-2b}(a+b) - ab + 2b^2}{2ab(a+b)} \\
&= \frac{1 - e^{-2a}}{2a} \\
&= \frac{1 - e^{-4}}{4}
\end{aligned}
$$

Now when $\zeta \neq \xi$, let $b = \frac{3}{2}$,

$$
\begin{aligned}
S_{IGS}(\xi,\zeta) &= \frac{2^2 - 2e^{-2\frac{3}{2}}(2+\frac{3}{2}) - 2\frac{3}{2} + 2\frac{3}{2}^2}{(2)(2)\frac{3}{2}(2+\frac{3}{2})} \\
&= \frac{11}{42} - \frac{e^{-3}}{3}
\end{aligned}
$$

Now computing the divergence of ζ from ξ,

$$\begin{aligned} D_{IGS}(\xi, \zeta) &= S_{IGS}(\xi, \zeta) - H_{IGS}(\xi) \\ &= \frac{11}{42} - \frac{e^{-3}}{3} - \frac{1 - e^{-4}}{4} \\ &= -0.000112 < 0 \end{aligned}$$

By lemma 4.6.4 as $D_{IGS}(\xi, \zeta) < 0$ and as $Y \perp\!\!\!\perp C$, it follows that L_{IGS} is not outcome-independent proper. ■

Proof of (ii)-(iv). Proofs follow from (i) and lemma 4.6.2. ■

□

Appendix D

Section 6.3.6 Figures

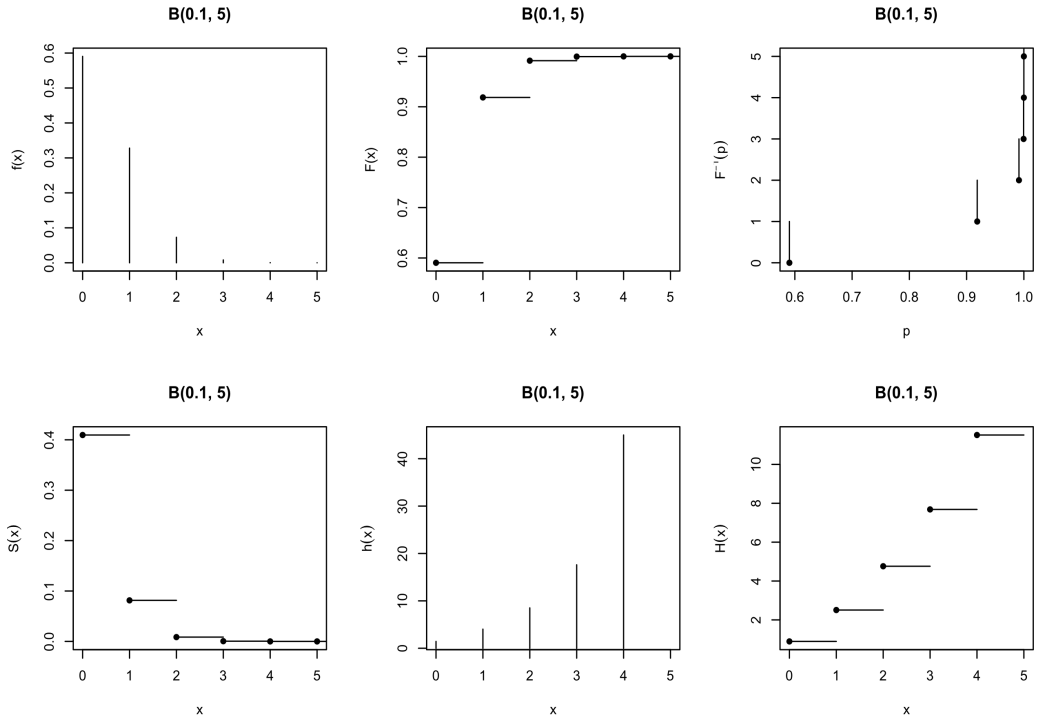


Figure 47: Visualising a Binom(0.1, 5) distribution, `B`, with `plot(B, fun = "all")`. Functions clockwise from top-left: probability mass, cumulative distribution, quantile (inverse cdf), cumulative hazard, hazard, and survival.

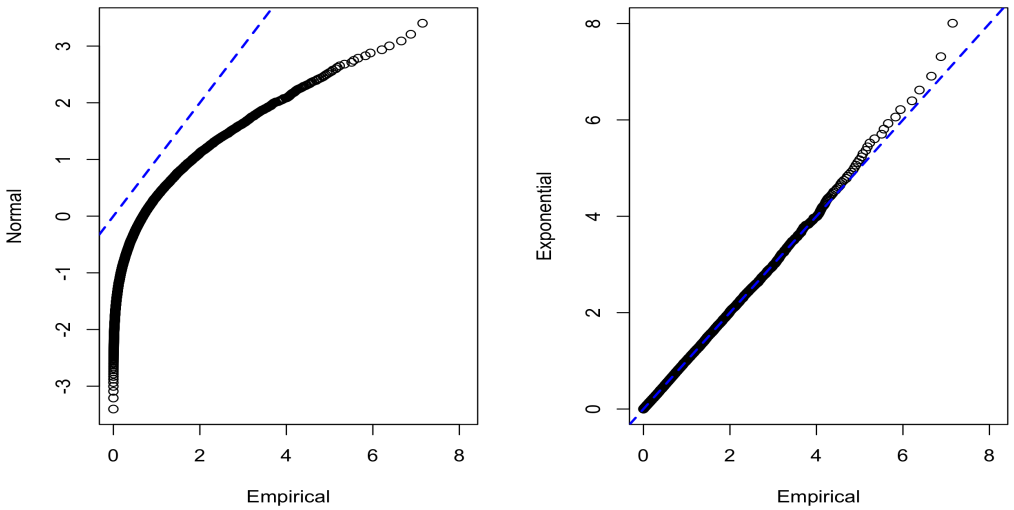


Figure 48: Q-Q plots comparing an unknown Empirical distribution, `E`, to theoretical Normal (left) and Exponential (right) distributions with `qqplot(E, Normal$new())` and `qqplot(E, Exponential$new())` respectively.

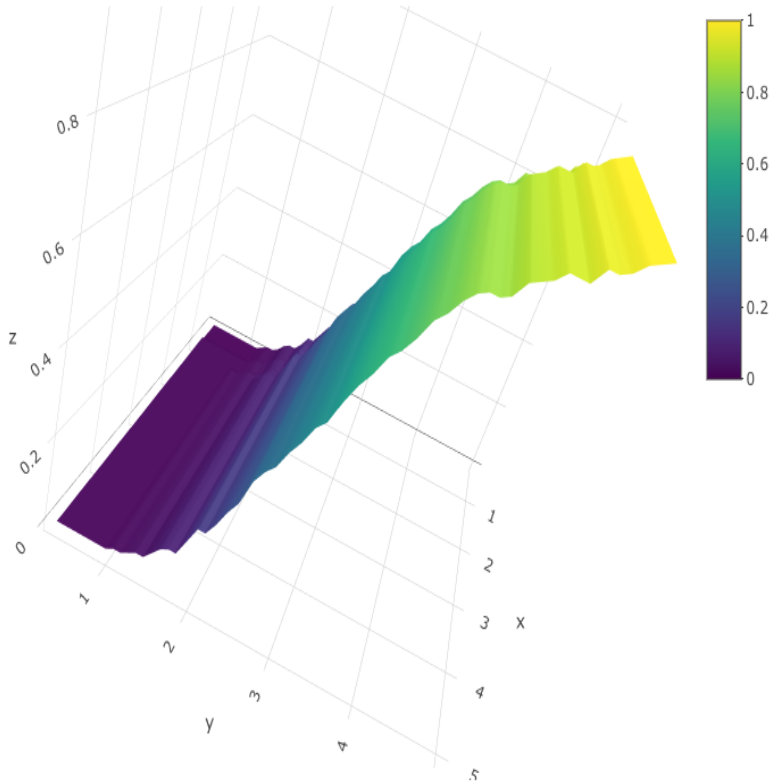


Figure 49: Visualising a bivariate empirical distribution, `E`, with `plot("E", fun = "cdf")`.

Appendix E

Chapter 7 Simulated Datasets

Table 28: Simulated datasets used in benchmark experiment.

Dataset[1]	Surv Dist[2]	Cens %[3]	Cens Type[4]
Sim1	Cox-Weibull	Type I	20
Sim2	Cox-Weibull	Type I	50
Sim3	Cox-Weibull	Type I	80
Sim4	Cox-Weibull	Right	20
Sim5	Cox-Weibull	Right	50
Sim6	Cox-Weibull	Right	80
Sim7	Cox-Weibull	Indep.	20
Sim8	Cox-Weibull	Indep.	50
Sim9	Cox-Weibull	Indep.	80
Sim10	Weibull	Type I	20
Sim11	Weibull	Type I	50
Sim12	Weibull	Type I	80
Sim13	Weibull	Right	20
Sim14	Weibull	Right	50
Sim15	Weibull	Right	80
Sim16	Weibull	Indep.	20
Sim17	Weibull	Indep.	50
Sim18	Weibull	Indep.	80
Sim19	Gompertz	Type I	20
Sim20	Gompertz	Type I	50
Sim21	Gompertz	Type I	80
Sim22	Gompertz	Right	20

Continued on next page...

Table 28: (continued)

Dataset[1]	**Surv Dist**[2]	**Cens %**[3]	**Cens Type**[4]
Sim23	Gompertz	Right	50
Sim24	Gompertz	Right	80
Sim25	Gompertz	Indep.	20
Sim26	Gompertz	Indep.	50
Sim27	Gompertz	Indep.	80
Sim28	Log-Normal	Type I	20
Sim29	Log-Normal	Type I	50
Sim30	Log-Normal	Type I	80
Sim31	Log-Normal	Right	20
Sim32	Log-Normal	Right	50
Sim33	Log-Normal	Right	80
Sim34	Log-Normal	Indep.	20
Sim35	Log-Normal	Indep.	50
Sim36	Log-Normal	Indep.	80

[1] Dataset identifier.

[2] Survival distribution.

[3] Type of censoring, either: Type I; Minimum of survival and censoring distribution (Right); uninformative/independent censoring (Indep.)

[4] Percentage of censoring in dataset.

Appendix F

Chapter 7 Simulated Data Plots

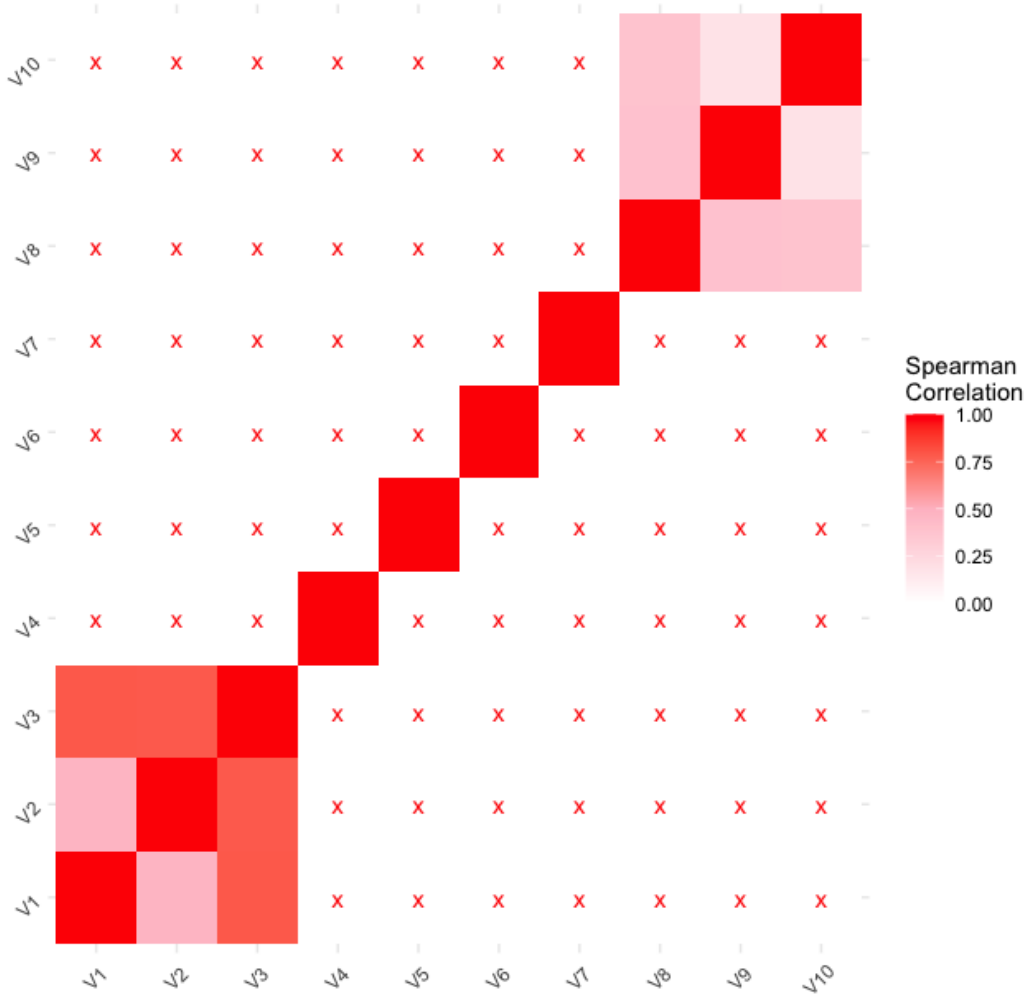


Figure 50: Correlation heatmap of the non-demographic $X_{\mathbf{x}}$ variables in the simulated datasets for one random simulation of 2,000 observations. A red 'x' indicates the correlation is non-significant, otherwise squares increase in darkness as correlation becomes more positive.

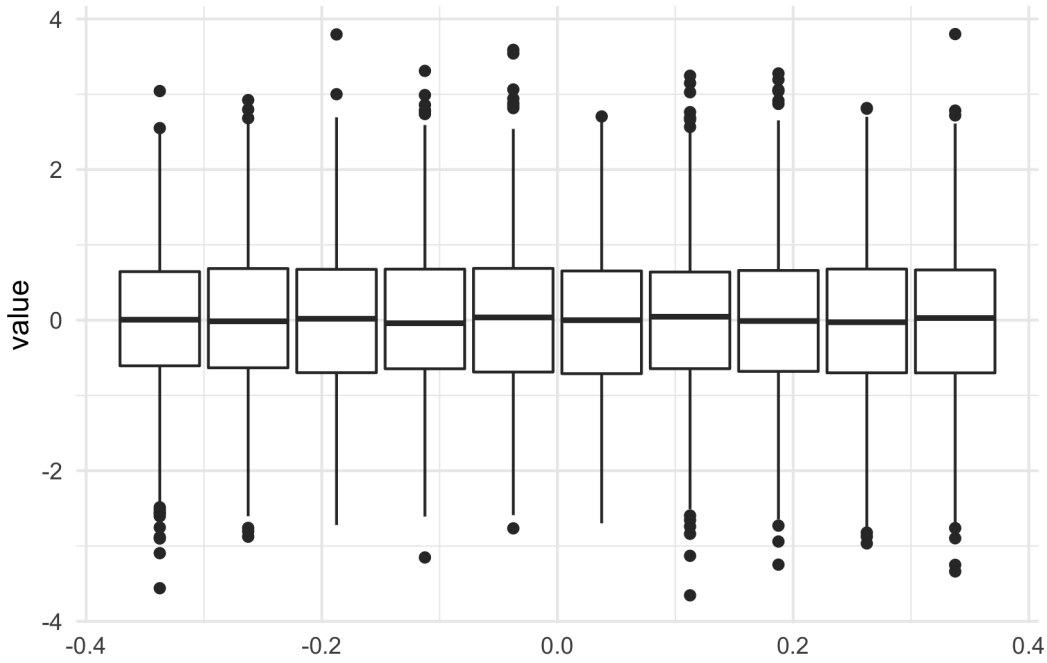


Figure 51: Boxplots demonstrating the variable data spread of the 'non-demographic' $X_{\mathbf{x}}$ variables in the simulated datasets for one random simulation of 2,000 observations. Each box is one of the variables and y-axis is the variable value.

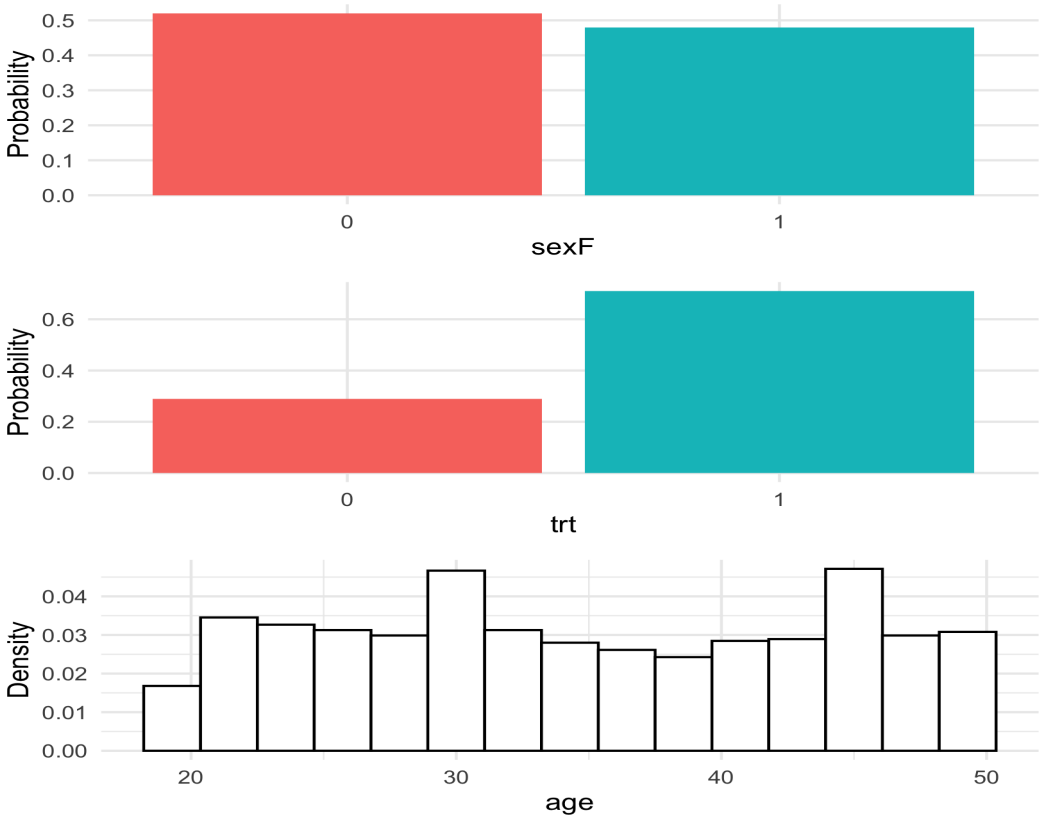


Figure 52: Plots demonstrating the variable data spread of the 'demographic' variables in the simulated datasets for one random simulation of 2,000 observations. Top plot is probability of each category in $X_{sex} \overset{i.i.d.}{\sim} \text{Bern}(0.5)$, middle is probability of each category in $X_{trt} \overset{i.i.d.}{\sim} \text{Bern}(0.7)$, bottom is density of $X_{age} \overset{i.i.d.}{\sim} \text{DiscreteUniform}[20, 50]$.

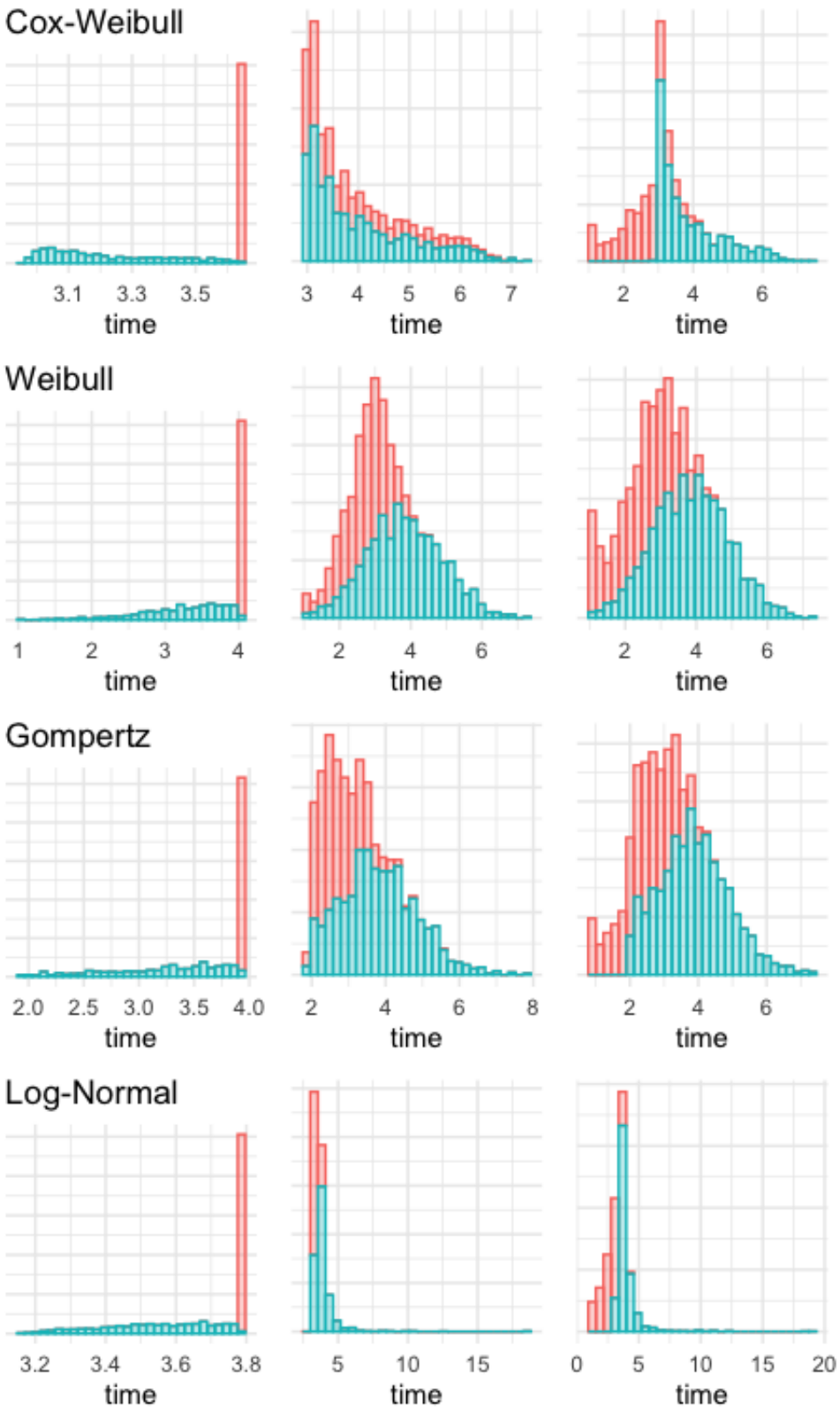


Figure 53: Density plots for simulated outcome time for a single seed with 50% censoring and 2,000 observations for Cox-Weibull (top row), Weibull (second row), Gompertz (third row), and Log-normal (fourth row) distributions and Type I (first column), right informative (second column), and uninformative (third column) censoring.

Appendix G

Chapter 7 Model Hyper-Parameter Ranges

Table 29: Hyper-parameters for tuned or non-default configurations for models in chapter 7.

Model	**Hyper-parameters**[1]	**Values**[2]	**Standardize**[3]	**Encode**[4]
Kaplan Meier	-	-	×	×
Nelson Aalen	-	-	×	×
Akritas Estimator	-	-	×	×
Cox PH	-	-	×	×
CV Regularized Cox PH	alpha	$[0, 1]$	×	✓
Penalized	lambda1	$[0, 10]$	×	×
	lambda2	$[0, 10]$		
Parametric	dist	{weibull, logistic, lognormal, loglogistic}	×	×
Flexible Splines	k	$\{1, ..., 7\}$	×	×
RSDF-STAT (rfsrc)	splitrule	logrank/bs.gradient**	×	×
	ntree	$\{250, ..., 2500\}$		
	mtry	$\{1, ..., 12\}$		
	nodesize	$\{1, ..., 20\}$		
RSDF-STAT (ranger)	splitrule	C	×	×
	num.trees	$\{250, ..., 2500\}$		
	mtry	$\{1, ..., 12\}$		
	min.node.size	$\{1, ..., 20\}$		

Continued on next page...

Table 29: (continued)

Model	Hyper-parameters[1]	Values[2]	Standardize[3]	Encode[4]
RSCIFF	ntree	$\{250, ..., 2500\}$	×	×
	mtry	$\{1, ..., 12\}$		
RRT	minbucket	$\{1, ..., 20\}$	×	×
	maxdepth	$\{2, ..., 30\}$		
GBM-GEH/UNO/COX*	family	gehan/cindex/coxph**	×	×
	mstop	$\{10, ..., 2500\}$		
	nu	$[0, 0.1]$		
	baselearner	{bols, btree}		
CoxBoost	penalty	optimCoxBoostPenalty	×	×
	maxstepno	1000		
SSVM	type	hybrid	✓	✓
	diff.meth	makediff3		
	gamma.mu	$([10^{-3}, 10^3], [10^{-3}, 10^3])$		
	kernel	{lin_kernel, rbf_kernel}		

Continued on next page...

Table 29: (continued)

Model	**Hyper-parameters**[1]	**Values**[2]	**Standardize**[3]	**Encode**[4]
Cox-Time	frac	0.3	✓	✓
	standardize_time	TRUE		
	num_nodes	$\{1^k,...,32^k\}$, $k = \{1, 2, 3, 4\}$		
	dropout	$[0, 1]$		
	weight_decay	$[0, 0.5]$		
	learning_rate	$[0, 1]$		
	epochs	100		
	early_stopping	TRUE		
DeepHit, DeepSurv, Nnet-Survival, PC-Hazard*	frac	0.3	✓	✓
	num_nodes	$\{1^k,...,32^k\}$, $k = \{1, 2, 3, 4\}$		
	dropout	$[0, 1]$		
	weight_decay	$[0, 0.5]$		
	learning_rate	$[0, 1]$		
	epochs	100		
	early_stopping	TRUE		

Continued on next page...

Table 29: (continued)

Model	Hyper-parameters[1]	Values[2]	Standardize[3]	Encode[4]
	validation_split	0.3		
	decay	$[0, 0.5]$		
DNNSurv	lr	$[0, 1]$	✓	✓
	epochs	100		
	early_stopping	TRUE		

[1] Hyper-parameters for tuning over. The choice of hyper-parameters are largely informed by recommendations from the model author and subsequent papers exploring optimisation. A '-' indicates no tuning is performed.

[2] Ranges for the respective hyper-parameters to tune over. Single values represent deviations from the defaults. Omitted parameters use the package defaults.

[3] Pre-processing of covariates by scaling to unit variance and centering to zero mean. A check (✓) indicates this step is performed before training the model, and a cross (✗) otherwise.

[4] Pre-processing of covariates by treatment encoding with `model.matrix`. A check (✓) indicates this step is performed before training the model, and a cross (✗) otherwise.

* The same tuning parameters are used for these models.

** These parameters are separated and implemented as separate models but condensed in this table for efficiency as all other hyper-parameters configurations in these rows are identical.

Appendix H

Chapter 7 Real Experiment Results Plots

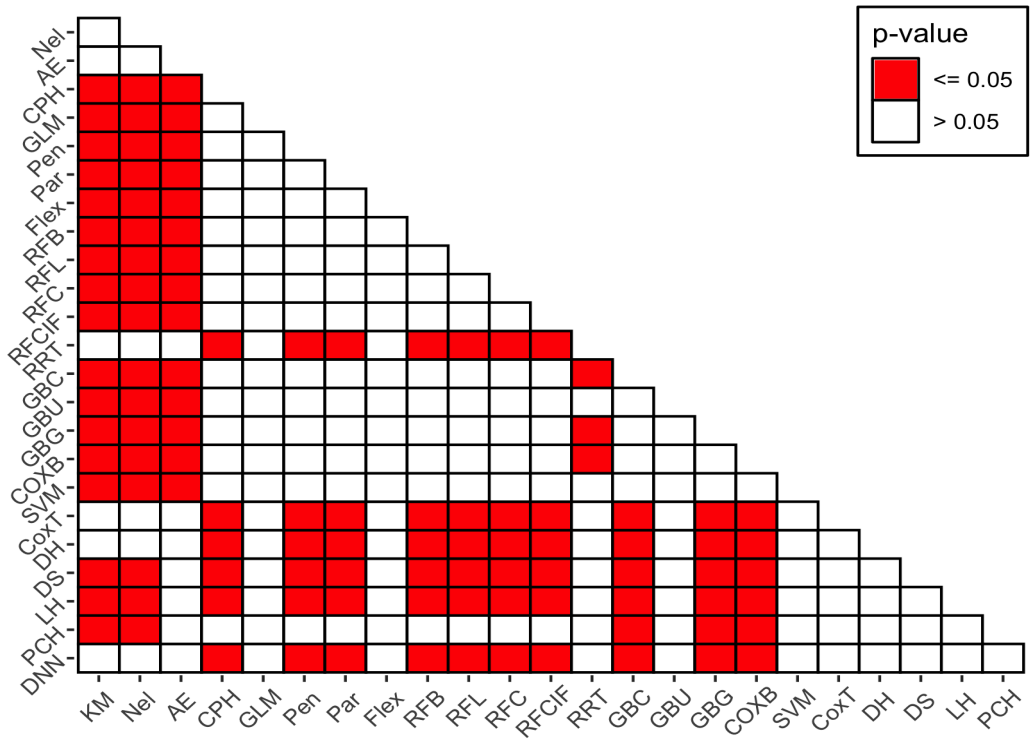


Figure 54: Post-hoc Friedman-Nemenyi tests comparing models on Uno's C. Red squares indicate significant differences ($p \leq 0.05$) between the pair of models on the x- and y-axis.

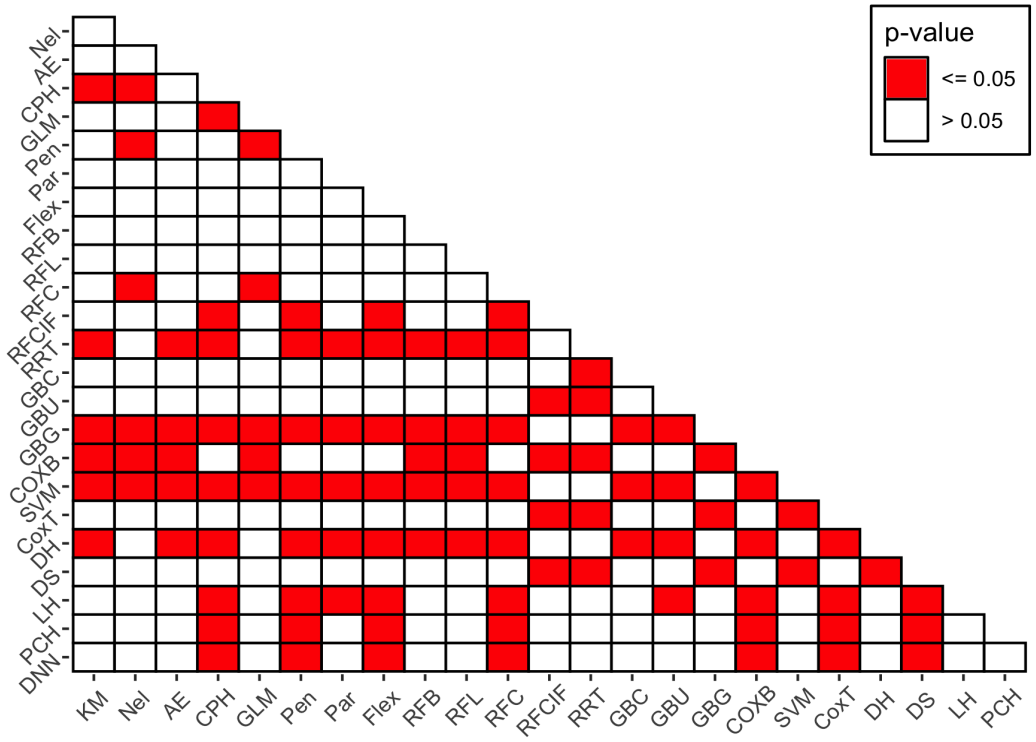


Figure 55: Post-hoc Friedman-Nemenyi tests comparing models on IGS. Red squares indicate significant differences ($p \leq 0.05$) between the pair of models on the x- and y-axis.

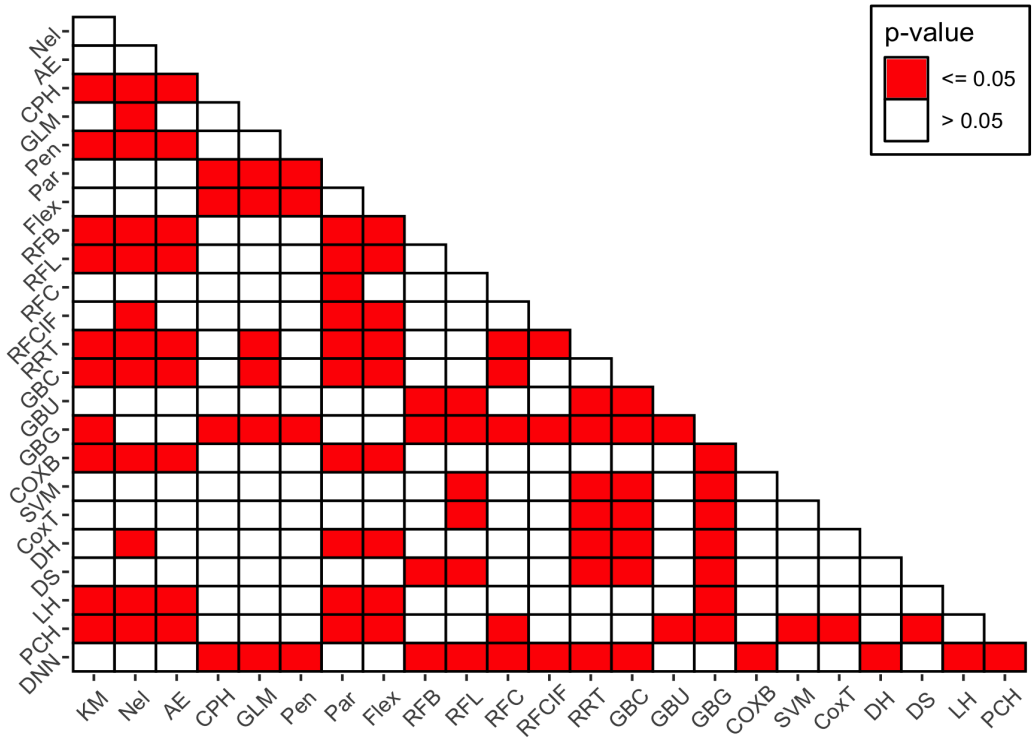


Figure 56: Post-hoc Friedman-Nemenyi tests comparing models on MAE. Red squares indicate significant differences ($p \leq 0.05$) between the pair of models on the x- and y-axis.

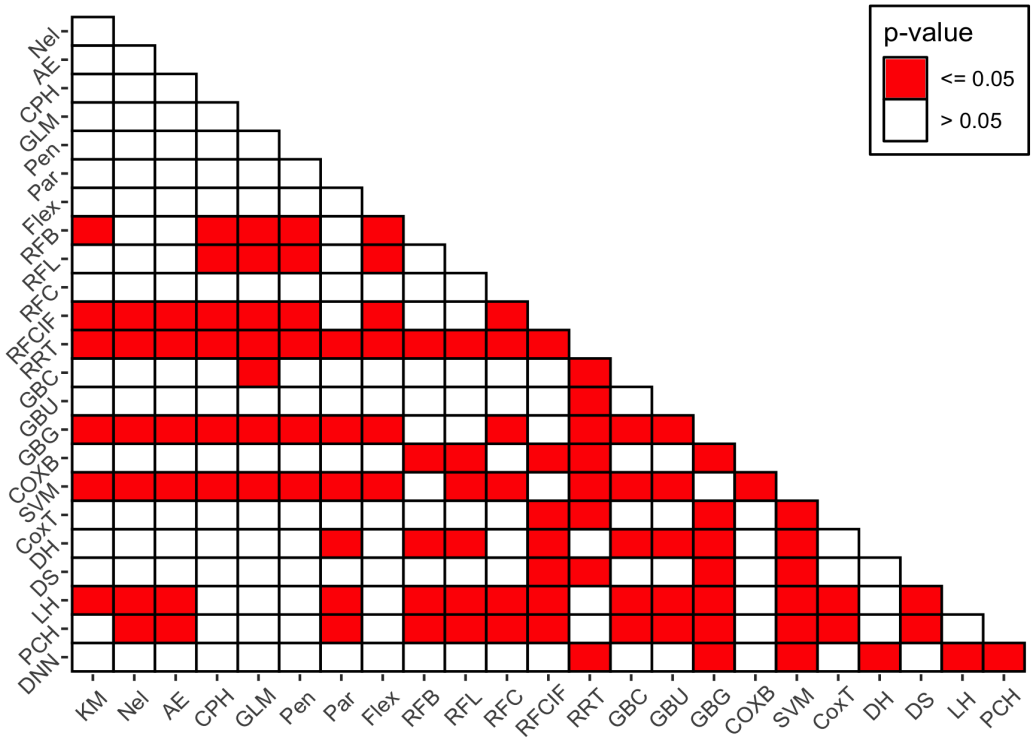


Figure 57: Post-hoc Friedman-Nemenyi tests comparing models on Houwelingen's α. Red squares indicate significant differences ($p \leq 0.05$) between the pair of models on the x- and y-axis.

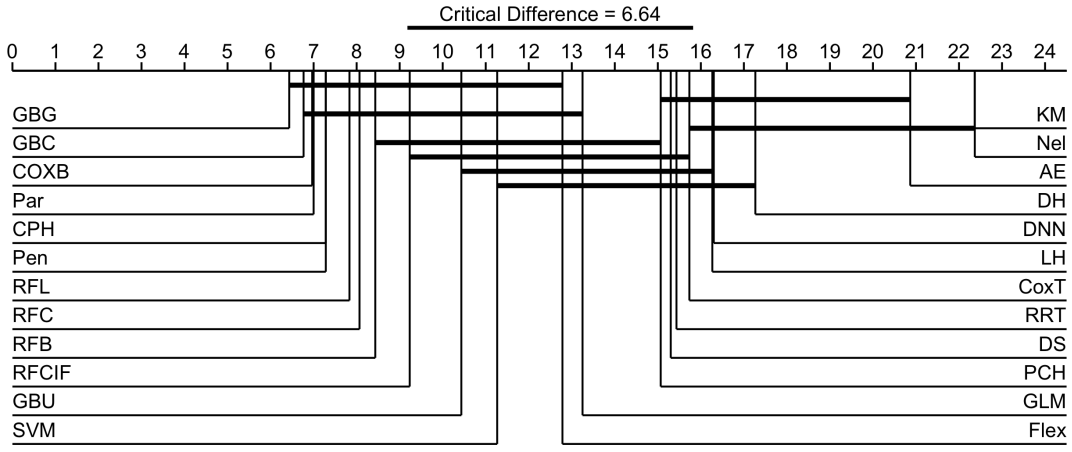


Figure 58: Critical difference plots comparing models on Harrell's C. Superior plots (lower rank) are on the left with decreasing performance (higher rank) moving right. Models connected by a thick horizontal black line are not significantly different in performance.

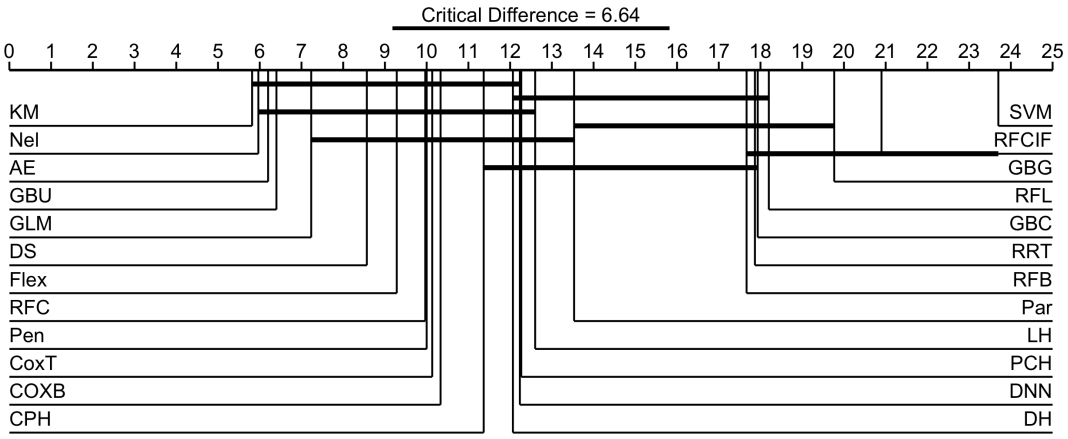


Figure 59: Critical difference plots comparing models on ISLL. Superior plots (lower rank) are on the left with decreasing performance (higher rank) moving right. Models connected by a thick horizontal black line are not significantly different in performance.

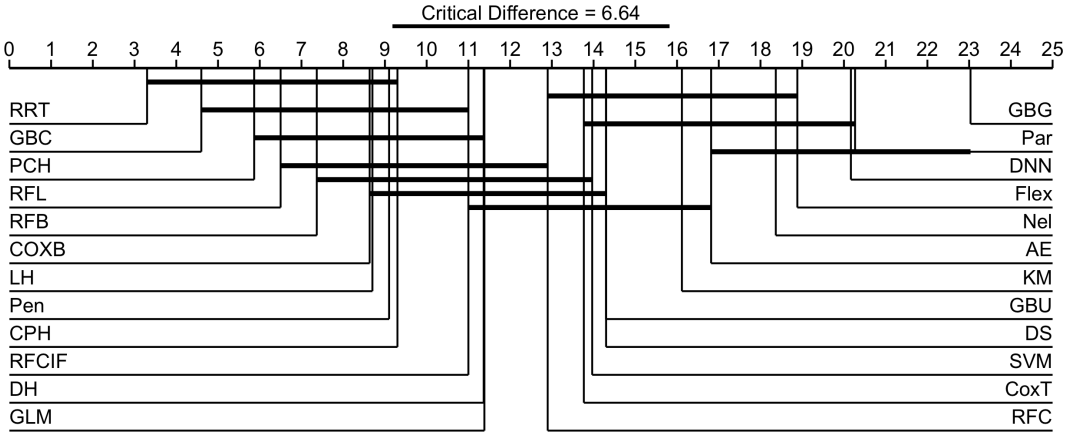


Figure 60: Critical difference plots comparing models on MAE. Superior plots (lower rank) are on the left with decreasing performance (higher rank) moving right. Models connected by a thick horizontal black line are not significantly different in performance.

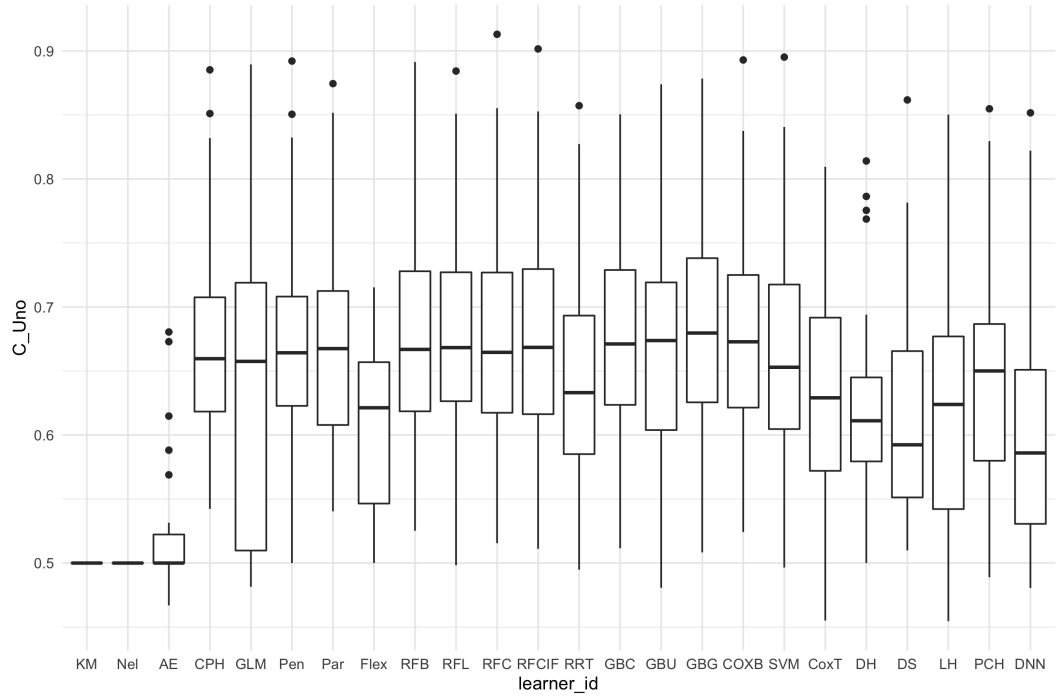


Figure 61: Boxplots of average model (x-axis) performance constructed across all datasets for Uno's C (y-axis).

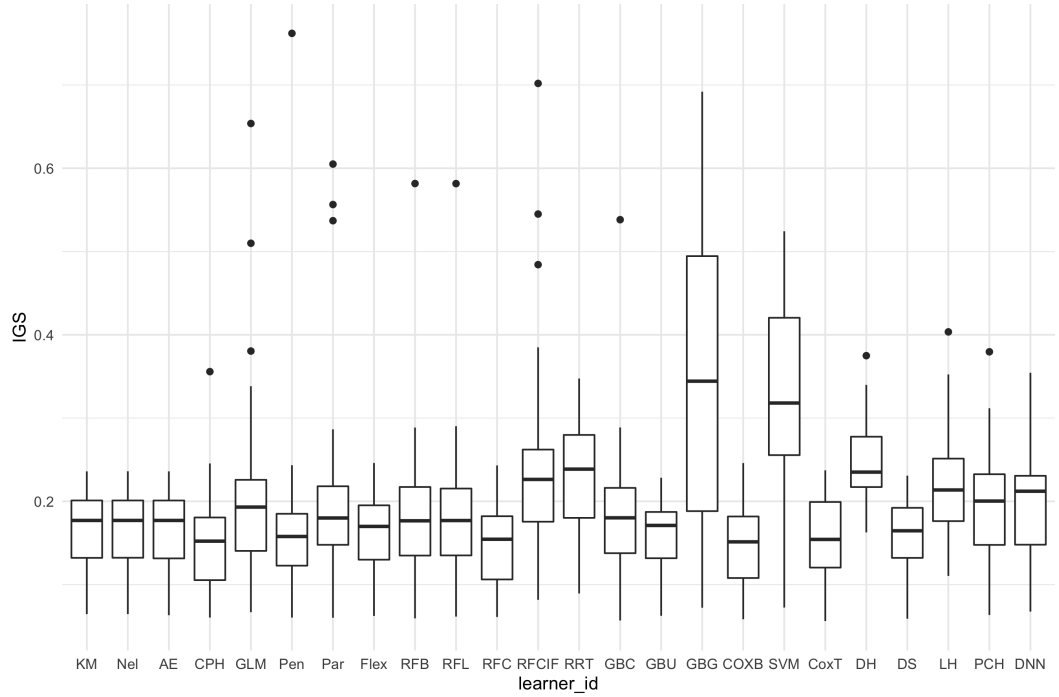


Figure 62: Boxplots of average model (x-axis) performance constructed across all datasets for IGS (y-axis).

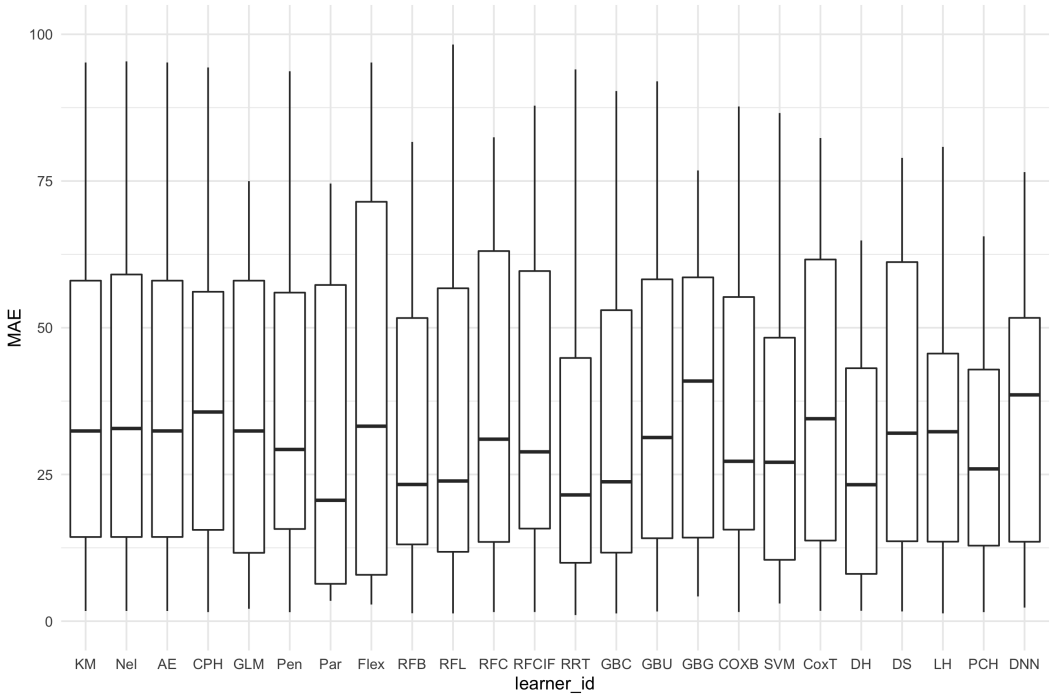


Figure 63: Boxplots of average model (x-axis) performance constructed across all datasets for MAE (y-axis).

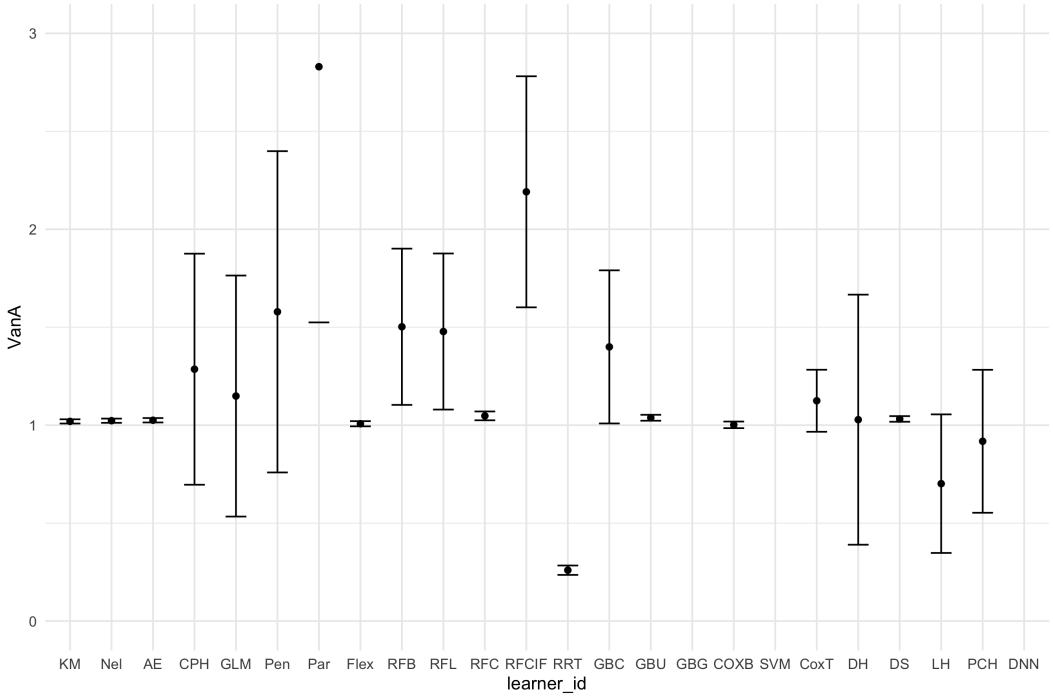


Figure 64: Mean and standard error of model (x-axis) performance constructed across all datasets for Houwelingen's α (y-axis). Values close to 1 indicate good calibration.

Appendix I

Chapter 7 Simulation Experiment Results Plots

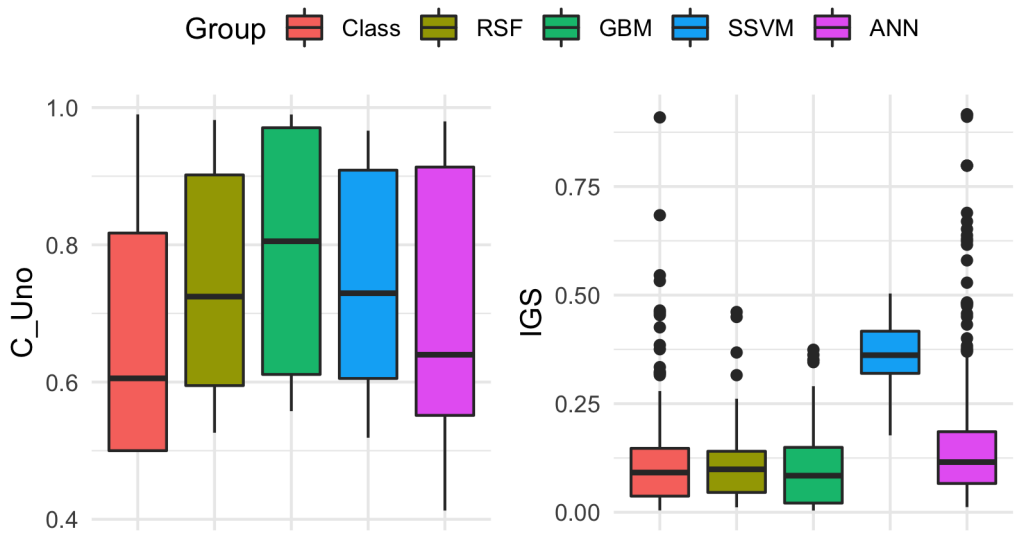


Figure 65: Model group performance across all datasets. x-axis are different model groups and y-axis is Uno's C (left) and IGS (right). Boxplots constructed across all datasets.

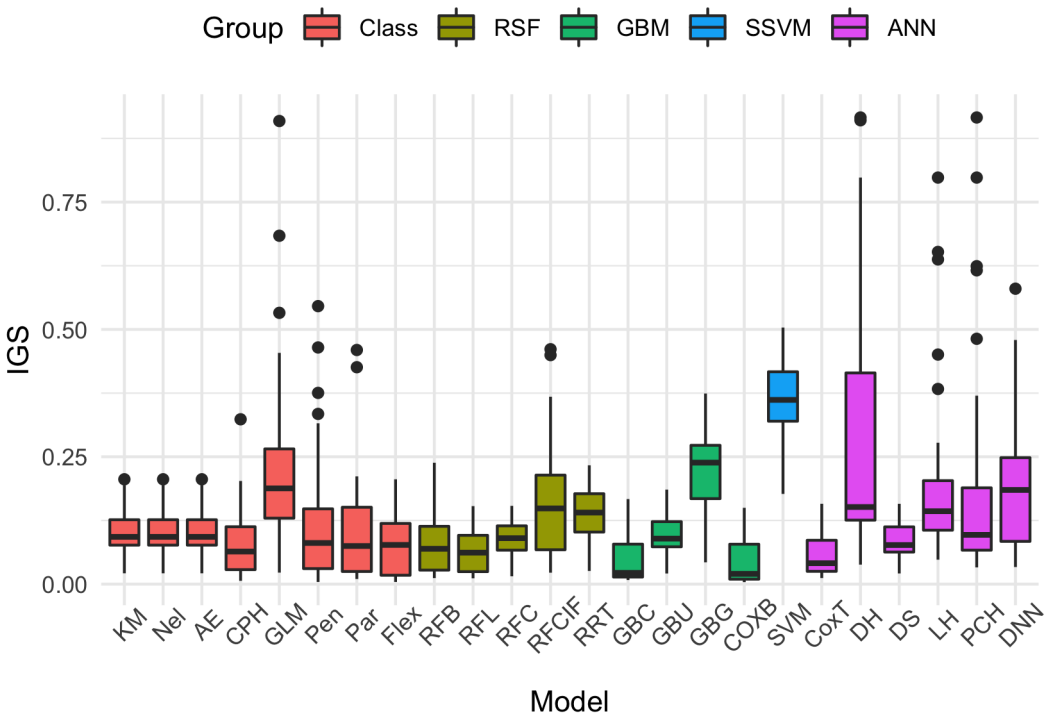


Figure 66: Boxplots of IGS (y-axis) constructed across all simulated datasets for all models (x-axis).