KØBENHAVNS UNIVERSITET
INSTITUT FOR MATEMATISKE FAG

Master's Thesis in Actuarial Mathematics

Rasmus Frigaard Lemvig

Machine learning methods for survival and multi-state models

Date: 20 December 2024

Supervisor: Martin Rainer Bladt
Co-supervisor: Niklas Andreas Pfister

This thesis has been submitted in fulfilment of the requirements for the degree of MSc's in Actuarial Mathematics at the Department of Mathematical Sciences, University of Copenhagen.

Abstract

In this thesis, we present and explore the random survival forest procedure, a machine learning algorithm for predicting the cumulative hazard from right-censored survival data. We explore the algorithm from a practical perspective by implementing the method from scratch in Julia and analysing several real datasets as well as a simulated dataset from life insurance. We furthermore investigate the theoretical properties of the algorithm by stating and proving consistency. The final part of the project concerns an extension of the random survival forest to multi-state models. We describe estimation in such a model and establish a new result on consistency for a single tree under the Markov assumption.

Resumé

I denne afhandling præsenterer og udforsker vi random survival forest-proceduren, en maskinlæringsalgoritme til prædiktion af den kumulerede dødelighed fra højre-censureret overlevelsesdata. Vi udforsker algoritmen fra et praktisk perspektiv ved at implementere metoden fra bunden i Julia og ved at analysere adskillige virkelige datasæt samt et simuleret datasæt fra livsforsikring. Vi kortlægger ydermere de teoretiske egenskaber for algoritmen ved at formulere og bevise konsistens. Den sidste del af projektet omhandler en udvidelse af random survival forest til flertilstandsmodeller. Vi beskriver estimation i sådan en model og etablerer et nyt resultat om konsistens for et enkelt træ under Markovantagelsen.

P. C. Skovgaard, *Bøgeskov i maj. Motiv fra Iselingen, 1857.*

"For man also knoweth not his time: as the fishes that are taken in an evil net, and as the birds that are caught in the snare; so are the sons of men snared in an evil time, when it falleth suddenly upon them."
- Ecclesiastes 9:12

Contents

1 Introduction **1**
- 1.1 Background and motivation . . . 1
- 1.2 Structure of the project . . . 1
- 1.3 Acknowledgements . . . 2

2 Preliminaries from machine learning **4**
- 2.1 Tree-based learning methods . . . 4
  - 2.1.1 Introduction and terminology . . . 4
  - 2.1.2 Growing a tree . . . 5
- 2.2 Bagging and random forests . . . 7
  - 2.2.1 Out-of-bag error estimates . . . 8
  - 2.2.2 Empirical studies . . . 9
  - 2.2.3 Using one tree versus many . . . 15

3 Mathematical preliminaries **16**
- 3.1 The product integral . . . 16
- 3.2 Results from probability theory . . . 24

4 Survival analysis **29**
- 4.1 Setup and notation . . . 29
- 4.2 Nonparametric estimation - The Nelson–Aalen estimator and its properties . . . 30
- 4.3 Parametric estimation . . . 34
  - 4.3.1 The case without covariates . . . 34
  - 4.3.2 Including covariates - Poisson regression . . . 38
- 4.4 More on the Gompertz–Makeham distribution . . . 40
- 4.5 Evaluating a survival model - Harrell's C-index . . . 44

5 Random Survival Forests **47**
- 5.1 Notation and terminology . . . 47
- 5.2 The RSF algorithm . . . 47
- 5.3 Prediction . . . 48
- 5.4 Conservation of events . . . 49
- 5.5 Splitting rules . . . 50
  - 5.5.1 Log-rank splitting . . . 51
  - 5.5.2 Conservation of events splitting . . . 51
  - 5.5.3 Log-rank score splitting . . . 52
  - 5.5.4 Approximate log-rank splitting . . . 53
  - 5.5.5 C-index splitting . . . 54
- 5.6 Prediction error . . . 55
- 5.7 Data examples . . . 55
  - 5.7.1 Comparison of splitting rules . . . 55
  - 5.7.2 A case study: The `peakVO2` dataset . . . 57
  - 5.7.3 Comments on performance . . . 62

5.8 Simulated example from life insurance . . . 62
5.8.1 Simulating the data . . . 62
5.8.2 Analysing the data . . . 63
5.8.3 Further comments . . . 65
5.9 Consistency . . . 66

6 Multi-state models **73**
6.1 Setup and estimation . . . 73
6.2 The (smooth) Markov model . . . 76
6.2.1 Definitions and basic properties . . . 76
6.2.2 The Markov process as a marked point process . . . 79
6.2.3 The Kolmogorov equations . . . 80
6.2.4 Counting processes and compensators for a Markov process . . . 82
6.2.5 The case without intensities . . . 84
6.2.6 Advantages and disadvantages of the Markov assumption . . . 84

7 Extending RSF to multi-state models **86**
7.1 The Aalen–Johansen forest . . . 86
7.1.1 The Aalen–Johansen forest procedure . . . 86
7.1.2 Prediction . . . 86
7.2 Consistency for a single Markov AaJo tree . . . 89

8 Discussion and further work **95**
8.1 Theoretical work . . . 95
8.2 Practical work . . . 95

References **101**

A The `JuliaExtendableTrees` library **102**
A.1 Overview of the library . . . 102
A.2 Using the library . . . 103
A.3 Essential functions and their implementations . . . 107
A.3.1 Determining the best split (survival) . . . 107
A.3.2 Some criteria functions . . . 110
A.3.3 Growing a tree . . . 112
A.3.4 Growing a forest . . . 114
A.3.5 Predicting values . . . 115
A.3.6 Computing the error . . . 118
A.4 Overall comments on the code . . . 121

B `JuliaExtendableTrees` in the project **122**
B.1 Overview of test files . . . 122
B.2 Overview of figure files . . . 122
B.3 The regression and classification analyses in Subsection 2.2 . . . 122
B.4 The survival analyses in Subsection 5.7 . . . 124

C More on the simulated data **127**
C.1 Exploratory plots 127
C.2 The R code used for simulating the data 128
C.3 The R code used for analysing the data 129

D More on conditional independence and Markov processes **134**

1 Introduction

"The man who moves a mountain begins by carrying away small stones."
- Confucius

1.1 Background and motivation

A classical problem in statistics and insurance is modelling and predicting mortality. In biostatistics, it is often the case that patients in a study can drop out of the study before the time of death is observed. This phenomenon is referred to as *(right-)censoring*. The landmark achievement in incorporating censoring is to shift the perspective from the survival function of the data to the *cumulative hazard*, the most famous estimator being the *Nelson–Aalen estimator*. This nonparametric estimator has been extensively studied and applied for decades. Another challenge lies in incorporating covariates. A popular model in this regard is the *Cox proportional hazards model*. This model is semiparametric in nature, leaving the baseline hazard unspecified while assuming that the covariate effects are proportional. An alternative is a parametric approach where the hazard (and thus the density) is fully specified. A classical example is the Gompertz–Makeham model, to be discussed further below. In recent years, nonparametric approaches from the area of machine learning have gained traction. In 2008, Ishwaran, Kogalur, Blackstone and Lauer introduced *random survival forests*, Ishwaran et al. [32], an ensemble learning method utilising the popular random forest procedure as introduced by Breiman [10] for analysing right-censored survival data. A significant advantage of machine learning methods is that, in contrast to parametric and semiparametic methods, complex relations between covariates need not be modelled explicitly. This allows for the algorithm to find complex patterns in data, enabling more precise prediction.

In insurance and certain areas of biostatistics, the survival model is not adequate to describe the behaviour of the policyholder/patient, and a more general multi-state model is therefore needed. In life insurance, the famous *seven state model* (see e.g. chapter VII in Asmussen and Steffensen [5]) has become state of the art in the industry. Often the transition rates are estimated by parametric means such as piecewise constant intensities. When the goal is accurate prediction, such models pose unnecessarily strict assumptions. Furthermore, many estimation methods that incorporate covariates (e.g. kernel methods) suffer from the *curse of dimensionality*. It is therefore of great value to develop machine learning methods for multi-state models that are accurate and robust, even when many covariates are present.

1.2 Structure of the project

The project is structured as follows. Sections 2 and 3 concern preliminaries on machine learning and mathematics used throughout the project. We go through the basics of decision trees and random forests for regression and classification. As for mathematics, particular emphasis is on the product integral and its properties. These

properties will be put to use throughout the project in different contexts. We also recall some results from probability theory related to survival analysis such as counting processes, the Doob–Meyer decomposition and Lenglart's inequality. In section 4, we discuss survival analysis in both a nonparametric and a parametric framework. The main object of interest is the Nelson–Aalen estimator and its asymptotic properties. Parametric estimation is discussed since we later wish to compare machine learning methods to classical parametric methods. Section 5 is about random survival forests (RSF) and is the heart of the thesis. We present the RSF algorithm in detail and describe five different splitting rules. We present a wide variety of analyses carried out using RSF on both real data and simulated insurance data. This constitutes the practical work of the thesis. The rest of section 5 is dedicated to a proof of consistency for RSF based on the one in Ishwaran and Kogalur [31] but under different assumptions that are more generalisable to the multi-state setup. This closes the discussion of survival models in the project. Section 6 provides the background on multi-state models. We discuss estimation in a general context based on the exposition in Bladt and Furrer [6] while more theoretical emphasis is put on the Markov model with intensities. Section 7 concerns the work done on extending the RSF algorithm to multi-state models, a procedure we have chosen to call the *Aalen–Johansen forest*. We consider some theoretical aspects of such an algorithm with particular emphasis on a new result, namely consistency for a single Aalen–Johansen tree under the Markov assumption. Section 8 concludes the thesis with a discussion on future work.

As part of the thesis, a Julia library, `JuliaExtendableTrees`, has been written. The code is available via the link

https://github.com/RasmusFL/JuliaExtendableTrees

Most of the analyses in the project have been carried out using this library. The code for running the analyses is available via

https://github.com/RasmusFL/ThesisAnalyses

A thorough but concise introduction to the library can be found in appendix A. Most of the code for doing the analyses is in appendix B, while all code not written in Julia can be found in appendix C. Appendix D contains a brief discussion of conditional independence as well as a proof of a theorem in section 6.

1.3 Acknowledgements

I would like to thank my supervisor Martin Rainer Bladt for his helpful advice, encouragement and overall excellent supervision. I also want to thank my co-supervisor Niklas Pfister for sharing his insight into machine learning and the practical aspects of the project. I furthermore wish to express my gratitude to Christian Furrer for helpful discussions concerning both the project and my plans for the future.

I would like to thank my friends and fellow students who made studying and writing the thesis so enjoyable. This (far from complete) list includes Andreas Studsgaard, Erik Gimsing, Josef-Nam Nguyen, Laus Wullum (thanks for all the help with Julia!), Jinyang Liu (thanks for the help with Python!), Adam Hoffmann, Anna Mai Østergaard, Marie Kaltoft, Josefine Sjöström, Valdemar Skou, Sophus Willumsgaard, Magnus Hansen, Emil Wieser and Marius Kjærsgaard. Without you all, studying mathematics would have been a lot less fruitful, and most importantly, not nearly as fun. Last but not least, I am grateful to my parents for their love and support.

2 Preliminaries from machine learning

> "The real problem is not whether machines think but whether men do."
> - B. F. Skinner

The goal of this section is to provide a primer on the tree-based learning methods needed during the project. We start from the bottom with a single decision tree and continue with a discussion on bagging and random forests.

2.1 Tree-based learning methods

2.1.1 Introduction and terminology

Tree-based learning methods are machine learning methods built on *decision trees*. Our main reference for this section is the book Breiman et al. [11]. Assume we are given data on the form $(\mathbf{X}_1, Y_1), ..., (\mathbf{X}_n, Y_n)$ with each $\mathbf{X}_i$ a vector of covariates. It is typical in machine learning to refer to the Y_i as *labels* and the $\mathbf{X}_i$ as *features*. A decision tree is a binary tree constructed by performing splits in the form of yes/no questions on the variables in $\mathbf{X} = (X_1, ..., X_n)$. An example could be "is $X_3 \leq 4$". The data satisfying this constraint goes to the left node and the remaining data to the right node. It is important to note that each split separates the feature data $\mathbf{X}_1, ..., \mathbf{X}_n$ into two disjoint subsets. An illustration of a decision tree is below.

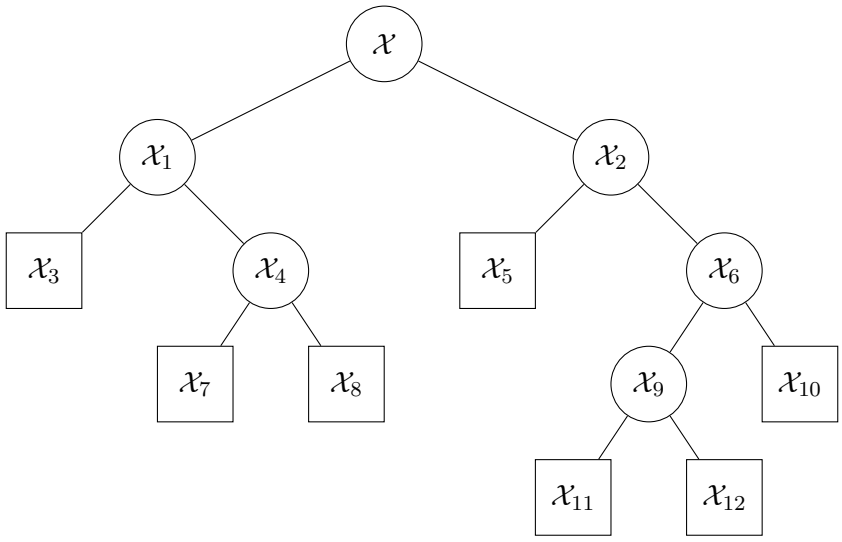

Figure 1: A simple decision tree. Here $\mathcal{X} = \{\mathbf{X}_1, ..., \mathbf{X}_n\}$ denotes all the available feature data. As an example, we have $\mathcal{X}_1 \cap \mathcal{X}_2 = \emptyset$ and $\mathcal{X}_1 \cup \mathcal{X}_2 = \mathcal{X}$.

The leaves in the tree are called *terminal nodes*. When we construct a decision tree from data (referred to as *growing* the tree), the terminal nodes decide the predicted values of the learner. In classical tree-based learning, one distinguishes between two types of labels. If the Y_i take a finite number of values $1, ..., J$, each j representing a class, we talk about *classification trees*. If the Y_i may take any value in an interval,

we talk about *regression trees*. A common terminology is *CART* (*Classification And Regression Trees*). We shall later see that the trees in random survival forests don't fall into either of these categories, but since all the methodology (and terminology) carry over directly to RSF, it makes sense to discuss the basics of CART.

2.1.2 Growing a tree

In growing a decision tree, the following need to be specified:

(i) Selecting the splits, in this project referred to as a *splitting rule*.

(ii) When to declare a node terminal.

(iii) The predicted value of a terminal node.

We start by adressing (iii). In the case of classification, a typical rule is to let the predicted class be the most common class in the terminal node (majority rule). In regression, one typically lets the predicted value be the average of the $Y_i's$ belonging to the $\mathbf{X}_i$ that fall into the terminal node. As for (i), we start by considering the problem of classification. The key idea is to choose a split which maximises the difference in proportions of the different classes. The following figure illustrates this idea.

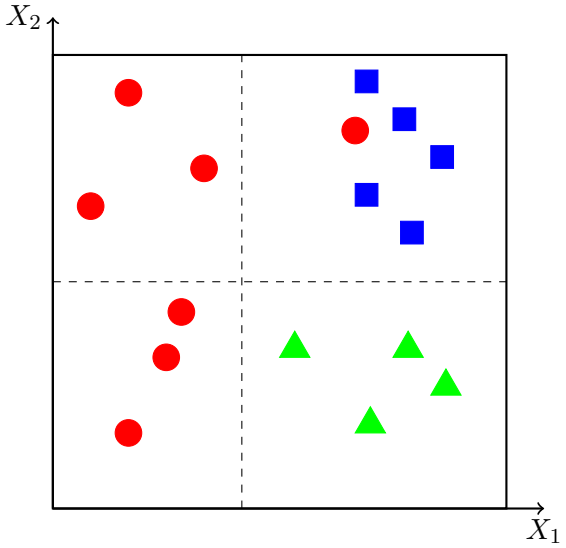

Figure 2: A simple illustration of splitting. We have three classes. The vertical dashed line indicates the first split in the data, while the horisontal line indicates the second split.

One formalisation of this idea is the following. Let h denote a node in the tree and $p_j(h)$ the proportion of data points with label j in node h. We then consider an *impurity function* i which, for a node h, gives a measure of the "impurity" of the

data in h. The impurity function i should be largest when the proportions of the different labels are the same and smallest when all the data belongs to the same class. Let S denote the set of possible splits at h. With the above figure in mind, such a split could be "is $X_1 \geq 3$?" This separates the data into two nodes, h_R and h_L. For a given split s, the difference in impurity is

$$\Delta i(s,h) := i(h) - p_L i(h_L) - p_R i(h_R)$$

with p_L and p_R denoting the proportions of the data in the left and right node, respectively. Now go through all possible splits s and choose the split that maximises $\Delta i(s,t)$. Note that we may intuitively interpret this as the split that minimises the tree "impurity". Popular choices for the impurity function i include the *entropy* defined by

$$E(h) = -\sum_j p_j(h) \log_2 p_j(h).$$

and the *Gini coefficient*

$$G(h) = 1 - \sum_j p_j(h)^2.$$

We can now discuss the point (ii) on when to declare a node terminal. A simple solution is to set a threshold for the gain in $\Delta i(s,t)$ and simply say that a node is terminal if for every split s, $\Delta i(s,t) < \beta$ for some chosen threshold $\beta > 0$. The problem with this approach is that we may lose good splits by setting β too high, while setting β too low results in a very complex tree that may severely overfit the data. Another approach is therefore to start by growing a full tree and thereafter *prune* the tree by removing more and more branches by gradually increasing the penalty for complexity (many terminal nodes). This yields a series of simpler and simpler subtrees of the full tree, and the final tree is chosen among these according to some criterion. Chapters 3 and 10 in Breiman et al. [11] discuss pruning and present a method called minimal cost-complexity pruning. Since this is not relevant for random forests, we omit further discussion of pruning here. Typical criteria for when splitting should stop include setting a minimum size for a terminal node or restricting the depth a tree can reach. We later investigate the effect of changing these two criteria on a real dataset.

The story for regression trees is very similar. A typical approach is *least squares regression* where the goal is to minimise the least squared error. Let $\mathcal{T}$ denote the tree and $\mathcal{N}(\mathcal{T})$ the set of terminal nodes. For a given node h, let n_h be the number of data points in h and let

$$\bar{Y}(h) = \frac{1}{n_h} \sum_{\mathbf{X}_i \in h} Y_i$$

denote the empirical average of the Y_i in h. We then define the estimator of the least squared error of the tree $\mathcal{T}$ by

$$R(\mathcal{T}) = \frac{1}{n} \sum_{h \in \mathcal{N}(\mathcal{T})} \sum_{\mathbf{X}_i \in h} (Y_i - \bar{Y}(h))^2.$$

Defining

$$R(h) = \frac{1}{n} \sum_{\mathbf{X}_i \in h} (Y_i - \bar{Y}(h))^2,$$

we can also write $R(\mathcal{T}) = \sum_{h \in \mathcal{N}(\mathcal{T})} R(h)$. Now assume we are in the process of constructing a tree, and that h is currently a terminal node. Consider a split s which separates the data in h into left and right nodes h_L and h_R. Defining

$$\Delta R(s, h) = R(h) - p_L R(h_L) - p_R R(h_R),$$

we choose the split s such that $\Delta R(s, h)$ is maximised. An alternative to least squares regression is to replace R with the absolute difference

$$R(h) = \frac{1}{n_h} \sum_{\mathbf{X}_i \in h} |Y_i - \bar{Y}(h)|.$$

In any case, the problem of when to declare a node terminal needs to be adressed. For a single tree, it is preferred to grow the tree to full size and then prune it. The story is similar to classification trees and is adressed in chapter 8 in Breiman et al. [11].

2.2 Bagging and random forests

Typically when using machine learning, the goal is to obtain a learner which is as accurate as possible. In that regard, using a single tree is far from optimal. A straightforward method to obtain more precise estimates is to use *bootstrap aggregation*, more commonly referred to as *bagging*. The idea of bootstrapping goes back to Efron [18] and is typically used to assess different properties of estimators such as standard deviations or approximating the distribution of an estimator or estimand. As for tree-based learners, in the article Breiman [8], Breiman argues that one can obtain substantial improvements in accuracy by aggregating the predictions from many individual decision trees. To be more precise, assume we are given a learning set $\mathcal{L} = \{(\mathbf{X}_i, Y_i) : i = 1, ..., n\}$ as above. Then draw B datasets with replacement from $\mathcal{L}$ and fit the learner on each of these. In the case of regression, simply take the empirical average to get the final prediction. For classification, the predicted class is simply chosen by majority rule.

How much does bagging improve accuracy? As explained in slightly more detail in section 4 of Breiman [8] for the case of regression, two factors are at play:

(i) The stability of the learner, that is, how much predictions vary from changes in the training set.

(ii) The loss of accuracy from sampling with replacement.

Let us elaborate on these two factors. If the learner is unstable, so that we see large variations in the predicted values when the training set varies, the aggregation

can substantially improve accuracy. On the other hand, if the learner is stable, the predictions vary very little and bagging has little effect. At the same time, whenever sampling with replacement is done, we fit a slightly less accurate learner compared to using the entire data set. So for a very stable learner, we can actually experience a loss of accuracy from bagging. An example of this phenomenon is using a linear model on data with strong linear tendencies.

Further improvements for tree-based learners can be obtained by injecting randomness into the process of growing the trees when using bagging. In the article Breiman [10] from 2001, Breiman explores randomly selecting the features used for growing each tree. The number of features used in fitting a tree becomes a new hyperparameter in the algorithm. Breiman called this procedure *Forest-RI*. While different variations of random forests exist, today, the term *random forest* typically refers to Breiman's method. Random forests have since become immensely popular since the method compares favorably to many state of the art methods such as boosting while remaining simple to use. Further advantages are relatively fast fitting times and robustness to outliers and noise.

Section 2 in Breiman [10] provides some mathematical arguments for why random forests work. The key observation is that the generalisation error can be minimised by increasing the strength of the individual trees and reducing the correlation between the trees. It turns out that randomly selecting the available features in each split reduces the correlation between the trees while preserving much of the predictive strength. Another possible way of decreasing correlation is to change the bootstrap sampling scheme. While most of the theory for random forests (including survival forests, as we shall see later) is formulated in terms of ordinary bootstrapping, where a bootstrap sample set is of the same size as the original, it is worth considering a different approach. An alternative is to subsample (with or without replacement) a dataset whose size is a certain fraction of the original, say 0.7. An obvious downside is a decrease in accuracy due to less data being used to grow each tree. For large datasets however, this downside may be negligible while the method may substantially decrease fitting times.

2.2.1 Out-of-bag error estimates

We want to assess the prediction error of our learning method as accurately as possible. The most naive method is to simply apply the learner on the training set and compare the result from the learner with the correct labels using some loss function. The problem with this approach is that it can severely underestimate the error since the learning algorithm is designed to minimise the loss on its training set. A more appropriate method that is often used in practice is to set aside a predetermined amount of data (for example 30%) to be used for testing purposes and train the learner on the remaining data. This provides more accurate estimates of the error, although some strength is lost due to less data being used for training.

When using bagging, an alternative is to compute the *out-of-bag* (OOB) error. Loosely speaking, OOB error works by only aggregating predictions over the learners fitted on a bootstrap set not containing the input feature. In the case of regression or classification, the method can be stated more precisely as follows.

1. From the learning set $\mathcal{L}$, draw B bootstrap samples $\mathcal{L}_1, ..., \mathcal{L}_B$ and fit the learner on each of these.

2. For each $(\mathbf{X}_i, Y_i)$ in the learning set $\mathcal{L}$, aggregate only over those learners for which the corresponding bootstrap training set does not contain $(\mathbf{X}_i, Y_i)$. Formally, we define $I_{i,b}$ by

$$I_{i,b} = \begin{cases} 1, & \text{if } (\mathbf{X}_i, Y_i) \in \mathcal{L}_b \\ 0, & \text{otherwise} \end{cases}.$$

   In the least squares regression paradigm, the OOB error in a node h is then estimated by

$$\frac{1}{n_h} \sum_{i:\mathbf{X}_i \in h} \frac{\sum_{b=1}^{B} I_{i,b}(Y_i - \bar{Y}(h))^2}{\sum_{b=1}^{B} I_{i,b}}.$$

   The approach is similar for classification. Simply replace the error with

$$\frac{1}{n_h} \sum_{i:\mathbf{X}_i \in h} \frac{\sum_{b=1}^{B} I_{i,b} \mathbb{1}_{\{Y_i \neq \widehat{Y}(h)\}}}{\sum_{b=1}^{B} I_{i,b}}$$

   with $\widehat{Y}(h)$ the prediction in the node h given by majority rule.

We leave out further discussion here and instead refer to the article Breiman [9] for more background. In contrast to the naive approach to estimate the error using all bootstrap sets, the OOB error has a tendency to overestimate the error. This is a consequence of the sampling process. Approximately 37% of the original data is left out in each bootstrap sample, and since the error decreases with larger data samples, we get an estimate which is biased upward. The extent of this bias is discussed in the article Janitza and Hornung [36]. While this bias is a deficiency, the OOB error has several advantages. One clear advantage is computational efficiency. In contrast to other popular methods such as cross-validation, OOB error estimation requires no additional computations, and it takes little effort to save the indices $I_{i,b}$ during the fitting process. Furthermore, there is empirical evidence that the OOB estimate is as accurate as using a test set of the same size as the training set. Hence OOB may remove the need to set aside a test set in the first place, see Breiman [9].

2.2.2 Empirical studies

We now present some applications of random forests in classification and regression. All analyses were made using the `JuliaExtendableTrees` library, and the primary purpose of this subsection is to illustrate the basic utilities of the package.

Some of the code used to fit the forests can be found in appendix B. All figures were made using the Algebra of Graphics library for Julia, [1]. The hyperparameters are the following:

- `L`: The function used to evaluate the value of the split. Possible choices for classification are `L_Gini_coeffcient` and `L_Entropy`. For regression, the choices are `L_squared_error` and `L_abs_error`.
- `max_depth`: The maximal depth of a tree. `max_depth = 0` corresponds to no maximal depth. The default value is 10 for classification and regression.
- `min_node_size`: The minimal size of a terminal node e.g. the minimal number of observations in a terminal node. The default value is 5 for classification and 10 for regression.
- `n_features`: Number of features that may be used in every split. By default, we use the square root of the total number of features rounded to the nearest integer.
- `n_split`: Maximum number of threshold values chosen from a feature when a node is to be split. By default, `n_split = 10`.
- `n_trees`: Number of trees in the forest.
- `sfrac`: Fraction of data being extracted in each bootstrap sample. Must be a number in $(0, 1]$. The default value is 0.7.
- `swr`: `true` means that sampling with replacement is applied, false (default) means without replacement.

We will investigate the effect of most of these hyperparameters in the following, some on a classification dataset and others on a regression set. We start by considering a dataset for classification. We use the *White Wine Quality Data* (`wine`) dataset, which can be found in the `randomForestSRC` package, Ishwaran et al. [33]. The labels have 11 levels, namely the integers 0 to 10, denoting the quality of the wine as assessed by wine experts. 0 is very bad and 10 is excellent. The dataset has 11 features, including measures of acidicity, pH, alcohol percentage and sulphates. The dataset contains 4898 observations and no missing data.

The first hyperparameters to be investigated are the split value function and the subsampling scheme. As mentioned earlier, using a smaller fraction of the data to fit each tree can reduce the number of computations substantially, so if we are able to get away with using a smaller fraction of the data without reducing the accuracy by too much, it is preferred to decrease `sfrac`. We test the values $0.1, 0.2, ..., 0.9, 1.0$ of `sfrac` where we use `swr = false` for all values except `sfrac = 1.0` (otherwise we would fit all trees on the same data). We set `n_trees = 1000`, `max_depth = 10`, `n_split = 10` and `n_features` is set to the square root of the number of features in total (rounded to the nearest integer). The following plot shows the

change in generalisation error (misclassification rate) when varying the fraction of the data used as just described.

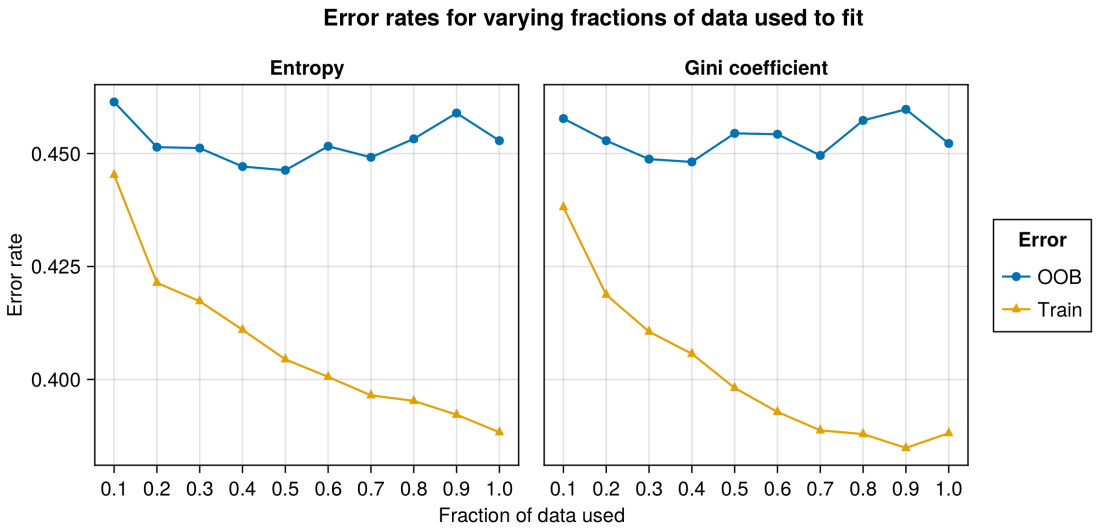

Figure 3: The generalisation error of a random forest fitted on the `wine` dataset for varying values of `sfrac`. The data is subsampled without replacement except in the case of `sfrac = 1.0`. The yellow lines are the training error estimates and the blue lines the OOB error estimates.

There are some interesting observations from these figures. The first is that the training error (yellow lines) decreases as a function of the fraction, which is not surprising. The more of the data actually used for growing a tree, the more the forest is optimised in terms of minimising the training error. As for the OOB error (blue lines), it hardly changes. One explanation is that the forest is poorly fitted for the small values of `sfrac` while too little of the data is OOB for larger values of `sfrac` which gives the OOB error an upward bias. Another explanation is that the dataset is very noisy. Running the same analysis in the `randomForestSRC` package confirms this hypothesis. It seems impossible to get an OOB error below 0.40. As a final comment, it seems that using the entropy as the splitting rule very marginally improves the accuracy over the Gini coefficient. It is not clear how one should choose `sfrac` from these figures, but 0.7 seems like a good guess.

Next we investigate varying the minimal size a node can be. Increasing the minimal size of a node is a way of regularising the trees as to prevent overfitting. But increasing the minimal size too much also means that the tree is not able to capture more complex patterns in the data. The plot below shows the generalisation error when varying the minimal node size. We have chosen `sfrac = 0.7` and `swr = false` with all other hyperparameters like before.

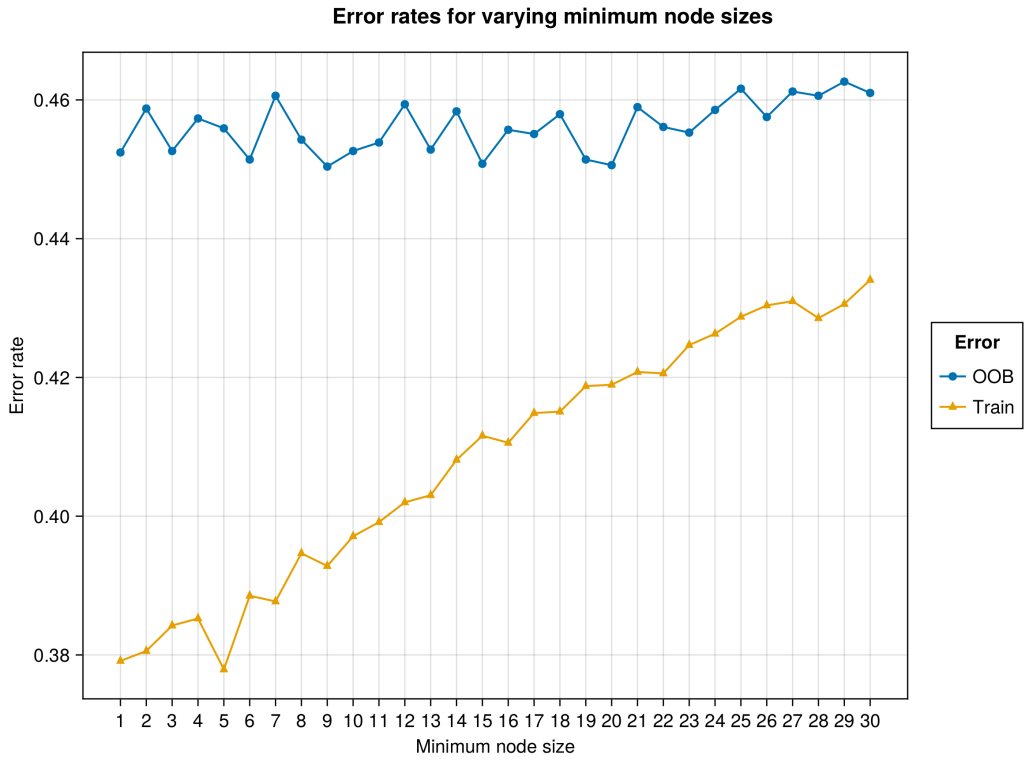

Figure 4: The generalisation error of a random forest fitted on the `wine` dataset for varying values of `min_node_size`. The yellow line is the training error estimate and the blue line the OOB error estimate.

This figure confirms the hypothesis that the data is quite noisy. The training error increases substantially for larger terminal nodes, while the OOB error remains the same. It is also likely that overfitting is taking place for small values of `min_node_size`.

The final hyperparameter to be investigated on the `wine` dataset is the maximum node depth, `max_depth`. The maximum depth of a node sets a direct restriction on how deep/complex a tree can be and therefore acts as a regularisation parameter. The plot below shows the generalisation error when varying the maximum node depth. The other hyperparameters are the same as before, where we choose `min_node_size` to be 10.

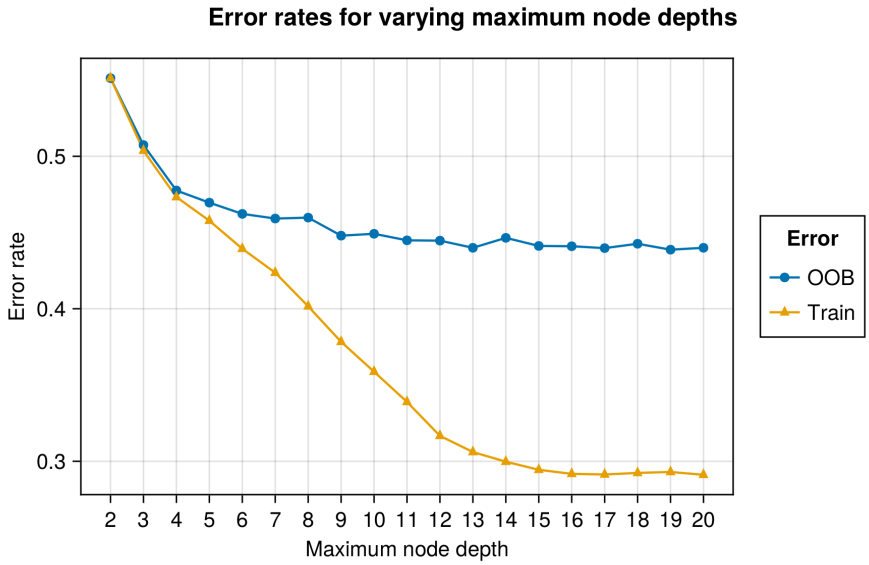

Figure 5: The generalisation error of a random forest fitted on the `wine` dataset for varying values of `max_depth`. The yellow line is the training error estimate and the blue line the OOB error estimate.

This concludes our study of the wine dataset, as we also want to illustrate the uses of `JuliaExtendableTrees` on regression data. For regression data, we use the Boston Housing dataset (`BostonHousing`, available through the R package `mlbench`, Leisch et al. [43]). The data was originally published in the article Harrison and Rubinfeld [26] and consists of 506 observations with no missing values. The label is the median value in $1000 of owner-occupied homes. There are 13 features in total. We first consider the effect of varying the number of features available in each split, `n_features`. We use the squared error as splitting function as well as `max_depth = 5`, `min_node_size = 5`, `n_split = 10`, `n_trees = 1000`, `sfrac = 0.7` and `swr = false`.

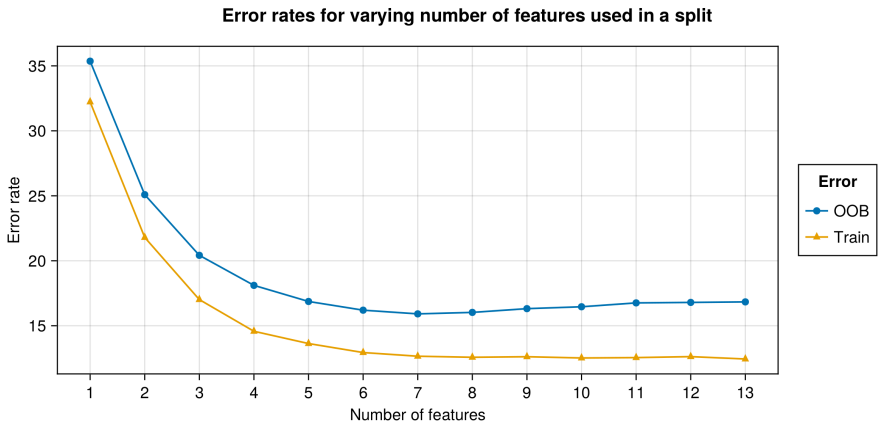

Figure 6: The mean squared error as a function of the number of features available in each split of a random forest fitted on the `BostonHousing` dataset. The yellow line is the training error and the blue line the OOB error. The case `n_features` = 13 corresponds to simply using bagging since the randomness in the chosen feature is omitted.

The figure illustrates the rationale for using random forests quite well. Not surprisingly, the error is high when few features are available. The error decreases until about half of the features are used, and then it starts climbing again. A plausible explanation is that some of the features are not strong predictors of the median value, but using more of them still introduces additional correlation between trees. Recall that we can expect predictive strength to increase when the correlation between trees is reduced and the predictive strength of each tree is increased. For this particular dataset, it seems that `n_features = 7` strikes a good balance between reducing the correlation between trees while preserving predictive strength. We thus choose `n_features = 7` for the final part of the analysis.

As a final demonstration of the library for regression, we consider the effect of varying the number of trees in the forest. When playing around with the data and hyperparameters, it was very clear that `max_depth` was essential in minimising the mean squared error. After some experimentation, `max_depth = 10` turned out to be a good choice. The following plot shows the error when varying the number of trees.

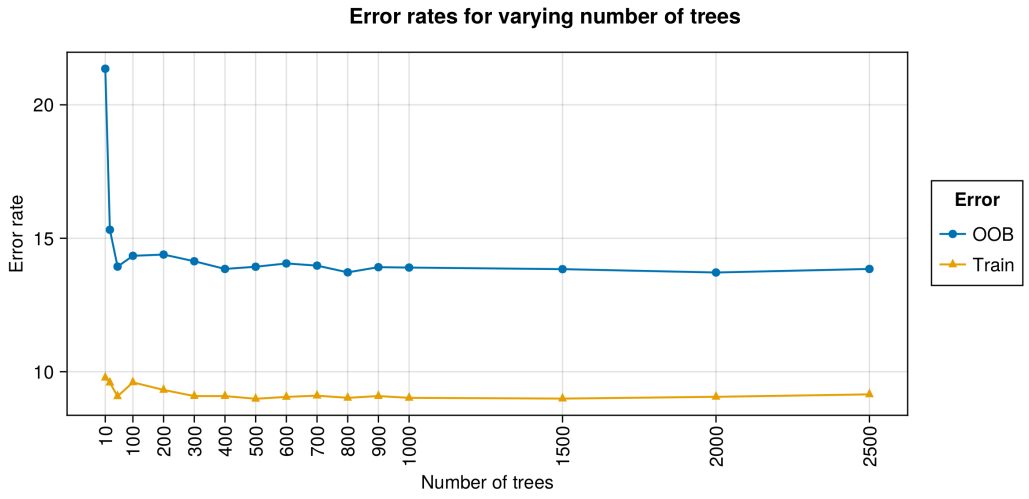

Figure 7: The mean squared error as a function of the number of trees in the forest. The yellow line is the training error, while the blue line is the OOB error. The number of trees vary over the values 10, 25, 50, 100, 200, ..., 900, 1000, 1500, 2000 and 2500.

Judging by the figure, it seems that for this particular dataset, the gain from using more than 1000 trees is negligible.

2.2.3 Using one tree versus many

Let us conclude this section with a brief discussion of using a single tree versus using many. An obvious advantage of using a single tree is that the algorithm is easily interpretable and explainable. This explainability is lost when we instead fit, say, a thousand trees and aggregate the results over all of them. However, for most purposes, the increase in accuracy when using forests more than compensates for the lack of interpretability. As discussed in section 9.2.4 of Hastie et al. [27], a single decision tree has the disadvantage of high variance. Slight alterations in the training data can change the final tree substantially. A major reason for this is the process of growing the tree itself. An early split affects every subsequent split and hence the error effect propagates all the way down the tree. Bagging remedies this by "averaging away" some of this variance. Similarly, individual trees have a tendency to overfit unless strict regularisation is applied, for example, by punishing trees with many terminal nodes. In bagging, this is rarely an issue, since the trees often overfit "in different directions", and bagging again averages this error away.

3 Mathematical preliminaries

> "If in other sciences we should arrive at certainty without doubt and truth without error, it behooves us to place the foundations of knowledge in mathematics..."
> - Roger Bacon

This section covers the necessary mathematical preliminaries. Particular emphasis is placed on the tools required for later proofs of consistency and the setup of multi-state modelling.

3.1 The product integral

We start by covering the product integral. All results are from the article Gill and Johansen [22], in which most of the proofs can also be found. The product integral allows for compact and elegant formulations of many formulas and results from statistics and probability theory. The product integral has been rediscovered several times in varying degrees of generality since its introduction by Volterra in 1887, Volterra [53]. As a consequence, several equivalent definitions of the product integral have emerged. Our starting point will be the product limit definition. Consider an interval $(0, T]$, $T = \infty$ being a possibility, equipped with the sigma-algebra of Borel sets $\mathcal{B}((0, T])$. Let X denote a $p \times p$ matrix of finite signed measures on $]0, T]$. We will also denote the distribution function of X by X, $X(t) = X((0, t])$. We let I denote the identity matrix.

Definition 3.1. Given X as above, we define the *product integral*

$$Y(t) = \prod_{(0,t]} (I + X(ds))$$

as

$$Y(t) = \lim_{\max |t_i - t_{i-1}| \to 0} \prod_i (I + X((t_i, t_{i-1}))$$

where the limit is taken over every finite partition $0 = t_0 < t_1 < \cdots < t_n = t$ of $(0, t]$, and the matrix product is taken in its natural order.

Remark 3.2. The above definition only defines the product integral over sets of the form $(0, t]$. To define the product integral over a general Borel set B, we do as follows. Let $dX_B = \mathbb{1}_B dX$ and let Y_B be the product integral of X_B. We then define the product integral Y over B by

$$Y(B) = \prod_B (I + X(ds)) = Y_B(T).$$

∘

Remark 3.3. One should of course show that the product integral exists. This is done in Gill and Johansen [22] via the concept of domination, first in the simple case where $p = 1$ and later for general p. For the sake of brevity, we will not further pursue proofs of existence (or uniqueness). ∘

Remark 3.4. We choose to follow the exposition in Gill and Johansen [22] where the product integral is defined for measures. The theory can just as well be stated for functions. Let X be a càdlàg function of locally bounded variation, that is, for every $0 \leq s < t$, the *variation* over $(s, t]$ given by

$$|X|(s,t) = \sup_{s=t_0<t_1<\cdots t_n=t} \sum_i |X(t_i) - X(t_{i-1})|$$

is finite. Like the limit before, the supremum is taken over every finite partition of $(s, t]$. In this case, the product integral of X is defined by

$$Y(t) = \prod_{(0,t]} (I + X(ds)) = \lim_{\max |t_i - t_{i-1}| \to 0} \prod_i (I + X(t_i) - X(t_{i-1})).$$

In the special case of X being a pure jump function (with countably many jumps), we see that the product integral is given by

$$Y(t) = \prod_{(0,t]} (I + \Delta X(s)).$$

∘

From the product limit definition of the product integral just given, the following proposition is a straightforward consequence.

Proposition 3.5. *For any $s < u < t$, we have*

$$Y((s,t]) = Y((s,u])Y((u,t]).$$

We now present another definition of the product integral, namely the Péano series.

Definition 3.6. The product (matrix) measure $X^{(n)}$ is defined as the measure satisfying

$$X^{(n)}(B_1 \times \cdots B_n) = X(B_1) \cdots X(B_n)$$

for Borel subsets $B_1, ..., B_n$. For an interval D, define the set $U(D; n)$ given by

$$U(D;n) = \{(u_1, ..., u_n) \in D^n : u_1 < \cdots < u_n\}.$$

The *Péano series* is defined by

$$P(D;X) = I + \sum_{n=1}^{\infty} X^{(n)}(U(D;n)) = I + \sum_{n=1}^{\infty} \int_{u_1<\cdots<u_n;u_i\in D} \cdots \int X(du_1) \cdots X(du_n).$$

Theorem 3.7. *The product integral may be defined in terms of the Péano series as follows,*

$$P((s,t];X) = \prod_{(s,t]} (I + X(ds)).$$

Proof. See section 2.4 in Gill and Johansen [22]. ■

Using the Péano series definition of the product integral, an equivalent and highly useful definition can be given, namely as the solution to certain forward and backward integral equations. To formulate the theorem, consider the matrix-valued measure Y as a function $Y(s,t) = Y((s,t])$. We will refer to such a function as an *interval function*. If α denotes an interval function, we call α *additive* if $\alpha(s,t) = \alpha(s,u) + \alpha(u,t)$ for any $s \leq u \leq t$. We call α *multiplicative* if $\alpha(s,t) = \alpha(s,u)\alpha(u,t)$ for any $s \leq u \leq t$. With this terminology in place, we see that a matrix valued finite measure X can be considered as an additive interval function and that the corresponding product integral is a multiplicative interval function.

Theorem 3.8. *The product integral Y satisfies the two equations*

$$Y(s,t) = I + \int_s^t Y(s,u-)X(du),$$
$$Y(s,t) = I + \int_s^t X(du)Y(u,t).$$

Conversely, if Y is a function $Y(s,t)$ which is càdlàg in both variables and if Y satisfies either of the two integral equations above, Y is equal to the product integral $\prod_{(s,t]} (I + X(du))$.

Proof. Let Y denote the product integral $Y(s,t) = \prod_{(s,t]} (I + X(du))$. Consider the $(n+1)$'th term of the Péano series and apply Fubini's theorem to obtain

$$\begin{aligned} X^{(n+1)}(U((s,t];n+1)) &= \int_s^t X^{(n)}(\{(u_1,...,u_n) : (u_1,...,u_n,u) \in U((s,t];n+1)\})X(du) \\ &= \int_s^t X^{(n)}(U((s,u);n))X(du). \end{aligned}$$

We can now verify that Y satisfies the forward equation. Indeed,

$$\begin{aligned} Y(s,t) - I &= \sum_{n=1}^{\infty} X^{(n)}(U((s,t];n)) \\ &= \int_s^t I + \sum_{n=1}^{\infty} X^{(n)}(U((s,u-);n))X(du) = \int_s^t Y(s,u-)X(du). \end{aligned}$$

The backward equation is proven in a similar fashion. The proof of the converse statement can be found in the proof of Theorem 5 in Gill and Johansen [22]. ■

We also need the following extension of this result. Gill and Johansen [22] refer to this result as the *inhomogeneous equation*. A proof is not given, so we provide one here for completeness.

Theorem 3.9. *Let Z and W be $k \times p$ càdlàg matrix functions and X a $p \times p$ matrix measure of bounded variation. Then Z satisfies the equation*

$$Z(t) = W(t) + \int_0^t Z(s-)X(ds)$$

if and only if

$$\begin{aligned} Z(t) &= W(t) + \int_0^t W(s-)X(ds) \prod_{(s,t]} (I + X(du)) \\ &= W(0) \prod_{(0,t]} (I + X(ds)) + \int_0^t W(ds) \prod_{(s,t]} (I + X(du)). \end{aligned}$$

Proof. We start by establishing that

$$K(t) := \int_0^t W(s-)X(ds) \prod_{(s,t]} (I+X(du)) = \sum_{n=1}^{\infty} \int \cdots \int_{0<u_1<\cdots<u_n\leq t} W(u_1-)X(du_1)\cdots X(du_n).$$

Using the Péano series, we get

$$\begin{aligned} K(t) &= \int_0^t W(s-)X(ds) \left(I + \sum_{n=1}^{\infty} \int \cdots \int_{s<u_1<\cdots<u_n\leq t} X(du_1)\cdots X(du_n) \right) \\ &= \int_0^t W(s-)X(ds) + \sum_{n=1}^{\infty} \int_0^t \int \cdots \int_{s<u_1<\cdots<u_n\leq t} W(s-)X(ds)X(du_1)\cdots X(du_n) \\ &= \int_0^t W(s-)X(ds) + \sum_{n=2}^{\infty} \int \cdots \int_{0<u_1<\cdots<u_n\leq t} W(u_1-)X(du_1)\cdots X(du_n) \\ &= \sum_{n=1}^{\infty} \int \cdots \int_{0<u_1<\cdots<u_n\leq t} W(u_1-)X(du_1)\cdots X(du_n) \end{aligned}$$

as claimed. Now suppose Z satisfies the equation

$$Z(t) = W(t) + \int_0^t Z(s-)X(ds).$$

Then in particular, $Z(0) = W(0)$. Letting $Y(t) = Z(t) - W(t)$, we then get $Y(0,t) = Z(t) - W(t)$ and

$$Y(0,t) = \int_0^t Z(s-)X(ds) = \int_0^t W(s-)X(ds) + \int_0^t Y(0,s-)X(ds).$$

Applying this identity iteratively, we obtain

$$\begin{aligned}
Y(0,t) &= \int_0^t \left(W(s_1-) + \int_0^{s_1-} (W(s_2-) + Y(0,s_2-))X(ds_2) \right) X(ds_1) \\
&= \int_0^t W(s_1-)X(ds_1) + \int_0^t \int_0^{s_1-} W(s_2-) + \int_0^{s_2-} (W(s_3-) + Y(0,s_3-))X(ds_3)X(ds_2)X(ds_1) \\
&= \cdots \\
&= \sum_{n=1}^{\infty} \int \cdots \int_{0<u_1<\cdots<u_n\leq t} W(u_1-)X(du_1)\cdots X(du_n) = \int_0^t W(s-)X(ds) \prod_{(s,t]} (I + X(du)).
\end{aligned}$$

We conclude that

$$Z(t) = W(t) + \int_0^t W(s-)X(ds) \prod_{(s,t]} (I + X(du)).$$

Conversely, suppose this identity holds. Then

$$Z(t) = W(t) + \sum_{n=1}^{\infty} \int \cdots \int_{0<u_1<\cdots<u_n\leq t} W(u_1-)X(du_1)\cdots X(du_n).$$

We can now apply a technique very similar to the one in the proof of Theorem 3.8. We have for $n \geq 1$ that

$$\int \cdots \int_{0<u_1<\cdots<u_{n+1}\leq t} W(u_1-)X(du_1)\cdots X(du_{n+1}) = \int_0^t \int \cdots \int_{0<u_1<\cdots<u_n<s} W(u_1-)X(du_1)\cdots X(du_n)X(ds)$$

and so

$$\begin{aligned}
Z(t) &= W(t) + \int_0^t W(s-) + \left(\sum_{n=1}^{\infty} \int \cdots \int_{0<u_1<\cdots<u_n<s} W(u_1-)X(du_1)\cdots X(du_n) \right) X(ds) \\
&= W(t) + \int_0^t Z(s-)X(ds)
\end{aligned}$$

as desired. It only remains to verify the alternate expression for Z, namely

$$Z(t) = W(0) \prod_{(0,t]} (I + X(ds)) + \int_0^t W(ds) \prod_{(s,t]} (I + X(du)).$$

Defining

$$Y(s,t) := \prod_{(s,t]} (I + X(du)),$$

the backward equation yields the dynamics $Y(ds,t) = -X(ds)Y(s,t)$. Hence, applying the backward equation and integration by parts, we have

$$\begin{aligned} Z(t) &= W(t) + \int_0^t W(s-)X(ds) \prod_{(s,t]} (I + X(du)) = W(t) - \int_0^t W(s-)Y(ds,t) \\ &= W(0) \prod_{(0,t]} (I + X(ds)) + \int_0^t W(ds) \prod_{(s,t]} (I + X(du)), \end{aligned}$$

which serves to complete the proof. ■

The following result is essential in proving that the product integral is continuous in the supremum norm.

Theorem 3.10 (**Duhamel's equation**). *For matrix valued measures X_1 and X_2, the following identity holds:*

$$\prod_{(s,t]} (I+X_1(du)) - \prod_{(s,t]} (I+X_2(dx)) = \int_s^t \prod_{(s,u]} (I+X_1(du))(X_1(du)-X_2(du)) \prod_{(u,t]} (I+X_2(dx)).$$

Proof. Consider the measure $X_{1,2}^{(n,m)}$ on $\mathbb{R}_+^{n+m}$ defined according to

$$X_{1,2}^{(n,m)}(A_1 \times \cdots \times A_n \times B_1 \times \cdots \times B_m) = X_1(A_1) \cdots X_1(A_n) X_2(B_1) \cdots X_2(B_m)$$

for Borel sets $A_1, ..., A_n, B_1, ..., B_m$. Now apply Fubini's theorem to obtain

$$\begin{aligned} X_{1,2}^{(n,m)}(U((s,t];n+m)) &= \int_s^t X_1^{(n-1)}(U((s,u);n-1)) X_1(du) X_2^{(m)}(U((u,t];m)) \\ &= \int_s^t X_1^{(n)}(U((s,u);n)) X_2(du) X_2^{(m-1)}(U((u,t];m-1)) \end{aligned}$$

Summing over $n \geq 1$ and $m \geq 1$, we have

$$\begin{aligned} &\int_s^t \prod_{(s,u)} (I + X_1(dx)) X_1(du) \left(\prod_{(u,t]} (I + X_2(dx)) - I \right) \\ &\quad = \int_s^t \left(\prod_{(s,u)} (I + X_1(dx)) - I \right) X_2(du) \prod_{(u,t]} (I + X_2(dx)). \end{aligned}$$

Rearranging, this identity becomes

$$\begin{aligned} &\int_s^t \prod\nolimits_{(s,u)} (I + X_1(du))(X_1(du) - X_2(du)) \prod_{(s,t]} (I + X_2(dx)) \\ &\quad = \int_s^t \prod_{(s,u)} (I + X_1(dx)) X_1(du) - \int_s^t X_2(du) \prod_{(u,t]} (I + X_2(dx)). \end{aligned}$$

Applying both the forward and backward equations, the desired result follows. ■

So far we have worked on an interval $(0, T]$ with $T = \infty$ a possibility. We now restrict the setting to a bounded interval, $T < \infty$. Recall that for a matrix A, we define the norm $|A|$ by $|A| = \max_k \sum_j |a_{kj}|$.

Definition 3.11. Let α denote an interval function. We define the *variation norm* $\|\alpha\|_v$ of α to be the variation of α over $(0, T]$, that is,

$$\|\alpha\|_v = |\alpha|(0, T].$$

The *supremum norm* $\|\alpha\|_\infty$ is defined by

$$\sup_{0 \le s \le t \le T} |\alpha(s, t)|.$$

Let H and K denote càdlàg matrix functions of bounded variation. Recall the integration by parts formula

$$H(t)K(t) - H(s)K(s) = \int_s^t H_-(dK) + \int_s^t (dH)K.$$

If one of H or K is not of bounded variation, we use this formula to define integration with respect to this function. For example, if H is not of bounded variation, we define

$$\int_s^t (dH)K = H(t)K(t) - H(s)K(s) - \int_s^t H_-(dK). \tag{1}$$

Note that we may consider H as an interval function by letting $H(s, t) = H(t) - H(s)$. We then obtain the following helpful result (Lemma 5 in Gill and Johansen [22]).

Lemma 3.12. *Let H, K and U be càdlàg matrix functions with K and U of bounded variation. H need not have bounded variation. Then*

$$\left\| \int (dH)K \right\|_\infty \le 2\|H\|_\infty \|K\|_v \tag{2}$$

$$\left\| \int U(dH) \right\|_\infty \le 2\|H\|_\infty \|U\|_v \tag{3}$$

$$\left\| \int U(dH)K \right\|_\infty \le 4\|H\|_\infty \|U\|_v \|K\|_v. \tag{4}$$

Proof. From equation (1), we get

$$\left| \int (dH)K \right| \le \|HK\|_\infty + \|H\|_\infty \|K\|_v \le 2\|H\|_\infty \|K\|_v$$

by using that the variation norm is larger than the supremum norm. This proves the first assertion. The second assertion follows by applying an analogous argument to

$$\int U(dH) = (UH)(0, T) - \int (dU)H_-.$$

As for the final result, use integration by parts, $d(HK) = (dH)K + H_-(dK)$, to get

$$\int U(dH)K = \int U(d(HK)) - \int UH_-(dK).$$

Now an application of the first two inequalities finishes the proof. ■

Remark 3.13. It is not difficult to show a sharpening of the inequality

$$\left\| \int U(dH)K \right\|_\infty \leq 4\|H\|_\infty \|U\|_v \|K\|_v.$$

Indeed, applying integration by parts twice instead of only once, we get

$$\int U(dH)K = (UHK)(0,T) - \int (dU)(HK)_- - \int UH_-(dK),$$

and all integrators are of finite variation, so we get

$$\left\| \int U(dH)K \right\| \leq \|UHK\|_\infty + \|U\|_v \|HK\|_\infty + \|UH\|_\infty \|K\|_v \leq 3\|H\|_\infty \|U\|_v \|K\|_v.$$

∘

We now have the tools to prove continuity of the product integral in the supremum norm.

Theorem 3.14. *Let $\{\alpha^{(n)}\}_n$ be a sequence of additive interval functions on $(0,T]$ satisfying*

$$\|\alpha^{(n)} - \alpha\|_\infty \to 0 \quad \textit{as} \quad n \to \infty \quad \textit{and} \quad \limsup_{n\to\infty} \|\alpha^{(n)}\|_v < \infty$$

for some interval function α. Then α is also additive and of bounded variation, and we have

$$\left\| \prod (I + \alpha^{(n)}(dx)) - \prod (I + \alpha(dx)) \right\|_\infty \to 0 \quad \textit{as} \quad n \to \infty.$$

Proof. Introduce the notation $\mu = \prod(I + \alpha(dx))$, $\mu^{(n)} = \prod(I + \alpha^{(n)}(dx))$. From Duhamel's equation and the previous lemma, we get

$$\begin{aligned} \|\mu^{(n)} - \mu\|_\infty &= \left\| \int_0^T \prod_{(0,u)} (I + \alpha^{(n)}(dx))(\alpha^{(n)}(du) - \alpha(du)) \prod_{(u,T)} (I + \alpha(dx)) \right\|_\infty \\ &\leq 4\|\alpha^{(n)} - \alpha\|_\infty \|\mu^{(n)}\|_v \|\mu\|_v. \end{aligned}$$

Now note that for any $a, b \geq 0$, we have

$$1 + a + b \leq (1 + a)(1 + b) \leq \exp(a + b).$$

From these inequalities, one deduces that for any non-negative interval function α_0, it holds that

$$1 + \alpha_0(s,t) \leq \prod_{(s,t]} (1 + \alpha_0(dx)) \leq \exp(\alpha_0(s,t)). \tag{5}$$

Using these inequalities and the forward equation, we get

$$\|\mu^{(n)}\|_v \leq \|\mu^{(n)}\|_\infty \|\alpha^{(n)}\|_v \leq \exp(\|\alpha^{(n)}\|_\infty) \|\alpha^{(n)}\|_v$$

which is bounded by assumption. We have thus shown that $\|\mu^{(n)} - \mu\|_\infty \to 0$ as $n \to \infty$ as desired.

■

Remark 3.15. Already from the elementary exponential inequality (5) and Duhamel's equation, it follows that the product integral is continuous in the variation norm considered as a functional from the space of additive interval functions on a bounded interval to the space of multiplicative interval functions. However, it turns out that with respect to the supremum norm, the product integral is not only continuous, but differentiable on compact intervals in a certain sense. We will not need this latter result, so we refrain from providing further details. We instead refer to the discussion following Theorem 7 in Gill and Johansen [22].

∘

3.2 Results from probability theory

In this subsection, we briefly recall the fundamental notions of counting processes, martingales and compensators. A thorough and concise reference is Cohen and Elliott [14]. A more brief recap can be found in Andersen et al. [4] chapter II.2 to II.4. Fix a time interval $I = [0, T]$ or $I = [0, T)$ with $0 < T \leq \infty$ and a filtered probability space $(\Omega, \mathcal{F}, \{\mathcal{F}_t\}_{t\in I}, \mathbb{P})$. We assume that the usual conditions (les conditions habituelles) hold, namely that the filtration is *right-continuous*

$$\mathcal{F}_t = \bigcap_{s>t} \mathcal{F}_s,$$

and that the filtration is *complete*, namely that for every $A \subseteq B \in \mathcal{F}$ with $\mathbb{P}(B) = 0$, we have $A \in \mathcal{F}_0$. Recall that a random variable $\tau : \Omega \to I$ is called a *stopping time* if for every t, $\{\tau \leq t\} \in \mathcal{F}_t$. A stochastic process $X = \{X_t\}_{t\in I}$ is simply a collection of random variables indexed by I. X is called *adapted* to the filtration $\{\mathcal{F}_t\}_{t\in I}$ if X_t is $\mathcal{F}_t$-measurable for all t. One can think of X as a function in two variables, namely $\omega \in \Omega$ and $t \in I$. For fixed ω, we refer to $t \mapsto X_t(\omega)$ as a *sample path* of X. We call X càdlàg if every sample path of X is càdlàg. We define the *predictable sigma-algebra* by

$$\mathrm{Pre}(\mathcal{F}) = \sigma(\{\{0\} \times A : A \in \mathcal{F}_0\} \cup \{(t, s] \times A : A \in \mathcal{F}_t, 0 \leq t < s < \infty\}),$$

and X is called *predictable* if the map $(t, \omega) \mapsto X(\omega, t)$ is measurable with respect to $\mathrm{Pre}(\mathcal{F})$. We think of a predictable process as a process whose value at time t is known immediately before t. Examples of predictable processes include all processes with left-continuous sample paths.

Definition 3.16. A càdlàg stochastic process $M = \{M_t\}_{t\in I}$ is a *martingale* if

(i) M is adapted to $\{\mathcal{F}_t\}_{t\in I}$,

(ii) $\mathbb{E}[|M_t|] < \infty$ for all $t \in I$ and

(iii) $\mathbb{E}[M_t \mid \mathcal{F}_s] = M_s$ a.s. for $s \leq t$.

If the equality in (iii) is replaced by $\geq$, we call M a *submartingale*. If we replace it with $\leq$, we call M a *supermartingale*. A martingale M is called *square-integrable* if

$$\sup_{t \in I} \mathbb{E}[M_t^2] < \infty.$$

A deep result in probability theory is the *Doob–Meyer decomposition theorem*. More than one version exists. For our purposes, the following version suffices. A stochastic process X is a *local* martingale (submartingale, supermartingale) if there exists a sequence $\{T_n\}_{n \in \mathbb{N}}$ of stopping times satisfying $\lim_{n \to \infty} T_n = \infty$ a.s. such that the stopped processes X^{T_n} are martingales (submartingales, supermartingales).

Theorem 3.17 (**Doob–Meyer decomposition**). *Let X be a right-continuous submartingale. Then there exists a non-decreasing predictable process Λ with $\Lambda_0 = 0$ a.s. such that the process $M = X - \Lambda$ is a local martingale. We call M the* Doob–Meyer decomposition *of X, and we refer to Λ as the* predictable compensator *of X.*

Proof. See section 9.2 in Cohen and Elliott [14]. ■

In the following, we will not worry about the "local" part of the above result. In most applications, one is allowed to assume that M is a true martingale.

Definition 3.18. Let X be a submartingale with Doob–Meyer decomposition $M = X - \Lambda$.

(i) The process Λ is called the *(predictable) compensator* of X.

(ii) If we can write

$$\Lambda_t = \int_0^t \lambda_s ds$$

for all t where λ is a predictable process, λ is called the *intensity process* of X.

Example 3.19. Consider a homogeneous Poisson process $N = \{N_t\}_{t \geq 0}$ with intensity λ with the natural filtration on the background space. Using that for $s \leq t$, the increment $N_t - N_s$ is Poisson distributed with parameter $\lambda(t-s)$ and that N has independent increments, trivial calculations show that $\Lambda_t = \lambda t$ is the compensator of N. In particular, the intensity process is constant and given by λ. ∘

Now consider a square integrable martingale M. One can then show that M^2 is a submartingale which satisfies conditions necessary to have a Doob–Meyer decomposition.

Definition 3.20. For a square integrable martingale M, we let $\langle M \rangle$ denote the predictable compensator of M^2 and call $\langle M \rangle$ the *predictable variation process* of M.

Example 3.21. Consider a standard Brownian motion $W = \{W_t\}_{t \geq 0}$ and let the filtration on the background space be the natural filtration generated by W. Using that W has independent increments with $W_t - W_s \sim \mathcal{N}(0, t-s)$ for $s \leq t$, it is easy to verify that $\langle W \rangle_t = t$. ∘

Say we have a finite variation square integrable martingale M and a predictable process H. Let Λ be the predictable compensator of M. Under certain integrability conditions, it follows that the predictable compensator of

$$\int H dN$$

is $\int H d\Lambda$. One can also show that the predictable variation process is given by

$$\left\langle \int H dM \right\rangle = \int H^2 d\langle M \rangle.$$

We now briefly recall the notion of a counting process.

Definition 3.22. (i) A *univariate counting process* N is a stochastic process which is zero at time zero, càdlàg, non-decreasing, takes values in $\mathbb{N}_0 \cup \{\infty\}$ and with $\Delta N_t \in \{0, 1\}$ for all $t \geq 0$.

(ii) A *multivariate counting process* $\mathbf{N} = (N^1, ..., N^m)$ is a vector of univariate counting processes with no two components jumping at the same time. That is, $\Delta N_t^j \Delta N_t^k = 0$ on $\{N_t^j < \infty, N_t^k < \infty\}$.

Consider a multivariate counting process $\mathbf{N} = (N^1, ..., N^m)$ with each N^j having intensity λ^j. One can then show (see Proposition II.4.1 in Andersen et al. [4]) that for $M^j = N^j - \int \lambda$, we have

$$\langle M^j \rangle_t = \int_0^t \lambda^j(s) ds.$$

Example 3.23. Let X be a non-negative random variable with density f with respect to Lebesgue measure and distribution function F. Let $S = 1 - F$ and define the hazard rate α by $\alpha = f/S$. Consider the *one-jump process* N given by $N_t = \mathbb{1}_{\{X \leq t\}}$ and the natural filtration $\mathcal{F}_t = \sigma(N_s : s \leq t)$. One can show, see example II.4.1 in Andersen et al. [4], that the predictable compensator of N is

$$\Lambda_t = \int_0^t Y_s \alpha(s) ds$$

where $Y_s = \mathbb{1}_{\{X \geq s\}}$. In particular, the intensity process is $Y_s \alpha(s)$. We will generalise this example, when we turn to the topic of survival analysis in the next section. ∘

The last tool from probability theory that we will need is Lenglart's inequality. We begin by stating and proving the inequality as presented in the paper Lenglart [44]. Afterwards we specialise to the case we will need. We adopt the convention $X_{0-} = 0$ for any stochastic process X in the following.

Definition 3.24. An adapted non-negative right-continuous process X is said to be *dominated* by a predictable, non-decreasing, right-continuous process A with $A_0 = 0$ if for every finite stopping time τ, we have $\mathbb{E}[X_\tau] \leq \mathbb{E}[A_\tau]$.

For a stochastic process X, let X^* denote the process given by

$$X_t^* = \sup_{s\in[0,t]} |X_s|.$$

Lemma 3.25. *Let X be dominated by A. Then for any stopping time τ and every $\eta > 0$, we have*

$$\mathbb{P}(X_\tau^* \geq \eta) \leq \frac{1}{\eta}\mathbb{E}[A_\tau].$$

Proof. Let $\sigma = \inf\{s \leq \tau \wedge n : X_s \geq \eta\}$ with the convention that $\sigma = \tau \wedge n$ if $\{s \leq \tau \wedge n : X_s \geq \eta\} = \emptyset$. Note that σ is a stopping time satisfying $\sigma \leq \tau \wedge n$. In particular, σ is finite, and we obtain the inequalities

$$\mathbb{E}[A_\tau] \geq \mathbb{E}[A_\sigma] \geq \mathbb{E}[X_\sigma] \geq \int_{\{X^*_{\tau\wedge n}>\eta\}} X_\sigma d\mathbb{P} \geq \eta\mathbb{P}(X^*_{\tau\wedge n} \geq \eta).$$

The first inequality is due to A being increasing, the second follows because A dominates X, and the final inequality is a consequence of the definition of σ. We have thus shown that $\mathbb{P}(X^*_{\tau\wedge n} \geq \eta) \leq \frac{1}{\eta}\mathbb{E}[A_\tau]$ for every $n \geq 1$. It follows that $\mathbb{P}(X_\tau^* \geq \eta) \leq \frac{1}{\eta}\mathbb{E}[A_\tau]$ and the proof is complete. ■

Theorem 3.26 (**Lenglart's inequality**). *Let X be a process dominated by A. For every stopping time X and every $\eta, \delta > 0$, we have*

$$\mathbb{P}(X_\tau^* \geq \eta) \leq \frac{1}{\eta}\mathbb{E}[A_\tau \wedge \delta] + \mathbb{P}(A_\tau \geq \delta).$$

Proof. We show that for every predictable stopping time $\tau > 0$ and for every $\eta, \delta > 0$, it holds that

$$\mathbb{P}(X^*_{\tau-} \geq \eta) \leq \frac{1}{\eta}\mathbb{E}[A_{\tau-} \wedge \delta] + \mathbb{P}(A_{\tau-} \geq \delta).$$

We first note that

$$\begin{aligned}\mathbb{P}(X^*_{\tau-} \geq \eta) &= \mathbb{P}(A_{\tau-} < \delta, X^*_{\tau-} \geq \eta) + \mathbb{P}(A_{\tau-} \geq \delta, X^*_{\tau-} \geq \eta)\\ &\leq \mathbb{P}(\mathbb{1}_{\{A_{\tau-}<\delta\}}X^*_{\tau-} \geq \eta) + \mathbb{P}(A_{\tau-} \geq \delta).\end{aligned}$$

Now let $\sigma = \inf\{t : A_t \geq \delta\}$. We claim that σ is a predictable stopping time. Indeed, we recognise σ as the debut of the predictable set $\{A_t \geq \delta\}$. The claim then follows from Lemma 7.3.7 in Cohen and Elliott [14][1]. Note that $\sigma > 0$ and hence $\sigma \wedge \tau > 0$ as well. Furthermore, $\sigma \wedge \tau$ is predictable. We claim that

$$\mathbb{1}_{\{A_{\tau-}<\delta\}}X^*_{\tau-} \leq X^*_{(\tau\wedge\sigma)-}.$$

Let ω satisfy $A_{\tau-}(\omega) < \delta$. Then $\tau(\omega) \leq \sigma(\omega)$ and we obtain an equality. If ω satisfies $A_{\tau-}(\omega) \geq \delta$, the left hand side is zero, and the right hand side is always at least zero by the convention $X_{0-} = 0$. Consequently,

$$\mathbb{P}(X^*_{\tau-} \geq \eta) \leq \mathbb{P}(X^*_{(\tau\wedge\sigma)-} \geq \eta) + \mathbb{P}(A_{\tau-} \geq \delta). \tag{6}$$

[1] We also refer to Cohen and Elliott [14] for general background on debuts of sets.

Now choose a sequence of stopping times $\{\sigma_n\}_{n\geq 1}$ such that $\sigma_n \uparrow \tau \wedge \sigma$ and so that $\sigma_n < \sigma \wedge \tau$ a.s. for all n. Note that for every $\varepsilon \in (0, \eta)$, we have

$$\{X^*_{(\tau\wedge\sigma)-} \geq \eta\} \subseteq \bigcap_{n=1}^{\infty} \{X^*_{\sigma_n} \geq \eta - \epsilon\}.$$

Using continuity from above, the previous lemma and monotone convergence yields

$$\begin{aligned}\mathbb{P}(X^*_{(\tau\wedge\sigma)-} \geq \eta) &\leq \mathbb{P}\left(\bigcap_{n=1}^{\infty} \{X^*_{\sigma_n} \geq \eta - \epsilon\}\right) = \lim_{n\to\infty} \mathbb{P}(X^*_{\sigma_n} \geq \eta - \varepsilon) \\ &\leq \lim_{n\to\infty} \frac{1}{\eta - \varepsilon}\mathbb{E}[A_{\sigma_n}] \leq \frac{1}{\eta - \varepsilon}\mathbb{E}[A_{(\tau\wedge\sigma)-}].\end{aligned}$$

As $\varepsilon \in (0, \eta)$ was arbitrary, $\mathbb{P}(X^*_{(\tau\wedge\sigma)-} \geq \eta) \leq \frac{1}{\eta}\mathbb{E}[A_{(\tau\wedge\sigma)-}]$. σ is the first time A passes δ, so $A_{(\tau\wedge\sigma)-} \leq A_{\tau-} \wedge \delta$. From (6), we obtain

$$\mathbb{P}(X^*_{\tau-} \geq \eta) \leq \frac{1}{\eta}\mathbb{E}[A_{\tau-} \wedge \delta] + \mathbb{P}(A_{\tau-} \geq \delta).$$

It only remains to use this fact to prove the theorem. Simply apply the inequality with X and A replaced by the stopped processes X^τ and A^τ and replace τ by the constant (and hence predictable) stopping time ∞. ■

The following corollary is the version that we will use.

Corollary 3.27. *Let M be a square integrable martingale with predictable variation process $\langle M\rangle$. For any $\eta > 0$ and $\delta > 0$, it holds that*

$$\mathbb{P}(\sup_{0\leq s\leq t} |M_s| > \eta) \leq \frac{\delta}{\eta^2} + \mathbb{P}(\langle M\rangle_t > \delta).$$

Proof. Simply note that

$$\mathbb{P}(\sup_{0\leq s\leq t} |M_s| > \eta) = \mathbb{P}((M^2)^*_t > \eta^2)$$

and that $\langle M\rangle$ satisfies all the conditions for dominating M^2. Now apply Lenglart's inequality as stated in the theorem above. ■

4 Survival analysis

"On a long enough timeline, the survival rate for everyone drops to zero."
- Narrator, *Fight Club*

4.1 Setup and notation

Let T° denote a non-negative random variable with distribution function F. Let $T_1^\circ, ..., T_n^\circ$ be iid copies of T°. The goal of survival analysis is to estimate F or, equivalently, the survival function $S = 1 - F$ in the presence of *censoring*. In this project, we consider right-censoring. To this end, let $R_1, ..., R_n$ be iid censoring variables with values in $(0, \infty]$. The observed data consists of $T = T^\circ \wedge R$ and $\delta = \mathbb{1}_{\{T=T^\circ\}} = \mathbb{1}_{\{T^\circ \leq R\}}$. The goal can then be formulated as estimating F based on the sample $(T_1, \delta_1), ..., (T_n, \delta_n)$. Classical estimators of F such as the empirical distribution function

$$\widehat{F}_n(t) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{\{T_i^\circ \leq t\}}$$

are no longer useful since we do not observe the T_i°. Replacing T_i° with T does not work either, since this would be a consistent estimator (by the strong law of large numbers) of $\mathbb{P}(T^\circ \wedge R \leq t)$ and not $\mathbb{P}(T^\circ \leq t)$ (unless censoring is not present). The key to estimating F in the presence of censoring is to work with (accumulated) hazard rates.

Definition 4.1. For a distribution function F, we define the *accumulated hazard/intensity* A of F as

$$A(t) = \int_0^t \frac{1}{S(s-)} F(ds).$$

Note that in the case where F admits a density f, the accumulated hazard is

$$A(t) = \int_0^t \frac{f(s)}{S(s)} ds,$$

and we refer to $\alpha(t) = f(t)/S(t)$ as the *hazard rate*. Note that we can recover F from A by

$$F(t) = 1 - \exp(-A(t)) \quad \text{or equivalently,} \quad S(t) = \exp(-A(t)).$$

Using the change of variable formula for càdlàg functions of finite variation, we see that

$$S(t) = \prod_{(0,t]} (1 + A(ds)),$$

and in the case with $A(t) = \int_0^t \alpha(s) ds$, we may write

$$S(t) = \prod_{(0,t]} (1 + \alpha(s) ds).$$

In order to do estimation, we need further assumptions on our data. The following assumption, which we refer to as *entirely random right-censoring*, suffices for our purposes.

Assumption 4.2. We assume that T° and R are independent.

Before stating an estimator of A, we discuss the example of a censored survival time. Assume T° has hazard rate α. Let N° be the counting process $N_t^\circ = \mathbb{1}_{\{T^\circ \leq t\}}$ counting the number of actual deaths with intensity process λ° given by $\lambda_t^\circ = \mathbb{1}_{\{T^\circ \geq t\}}\alpha(t)$. We do not observe this counting process however, but rather the process N given by

$$N_t = \mathbb{1}_{\{T \leq t\}}\mathbb{1}_{\{\delta = 1\}} = \int_0^t \mathbb{1}_{\{R \geq s\}} N^\circ(ds)$$

Under the assumption of entirely random right-censoring, one can show (see example III.2.5 in Andersen et al. [4]) that the intensity process λ of N is given by

$$\lambda_t = \mathbb{1}_{\{R \geq t\}}\lambda_t^\circ = \mathbb{1}_{\{R \geq t\}}\mathbb{1}_{\{T^\circ \geq t\}}\alpha(t) = \mathbb{1}_{\{T \geq t\}}\alpha(t).$$

In particular, the intensity process is of the same form as without censoring. This is one reason why hazard rate modelling is so mathematically appealing. Another reason is that the form of the intensity is preserved under aggregation. Indeed, from the iid sample $(T_1, \delta_1), ..., (T_n, \delta_n)$ with corresponding counting processes $N^1, ..., N^n$, define the aggregated counting process

$$N_t^{(n)} = \sum_{i=1}^n N_t^i.$$

Then $N^{(n)}$ has intensity process given by $\lambda_t^{(n)} = \alpha(t)Y_t^{(n)}$ where

$$Y_t^{(n)} = \sum_{i=1}^n \mathbb{1}_{\{T_i \geq t\}}$$

can be interpreted as the number of individuals at risk at time t. The form of the intensity process shows up often enough to deserve a name.

Definition 4.3. Consider a multivariate counting process $\mathbf{N} = (N^1, ..., N^m)$. We say that $\mathbf{N}$ satisfies the *multiplicative intensity model* if each N^k has intensity process λ^k of the form

$$\lambda_t^k = \alpha^k(t)Y_t^k,$$

where α^i is a non-negative deterministic function and Y^k is a non-negative predictable process with $\mathbb{1}_{\{Y_t^k > 0\}}/Y_t^k$ locally bounded.

4.2 Nonparametric estimation - The Nelson–Aalen estimator and its properties

In this subsection, we discuss nonparametric estimation methods in survival analysis with a particular emphasis on the Nelson–Aalen and Kaplan–Meyer estimators.

Definition 4.4. Consider a counting process N satisfying the multiplicative intensity model. Define $J_t = \mathbb{1}_{\{Y_t>0\}}$.

(i) We define the *Nelson–Aalen estimator* of A as

$$\widehat{A}_t = \int_0^t \frac{J_s}{Y_s} N(ds).$$

(ii) In the survival setup, we define the *Kaplan–Meyer estimator* of the survival function S as

$$\widehat{S}(t) = \prod_{(0,t]} (1 - \widehat{A}(ds)).$$

Note that the Nelson–Aalen estimator is a step function. At the j'th jump time τ_j of N, the jump is of size $1/Y_{\tau_j}$. Hence we can write

$$\widehat{S}(t) = \prod_{s\in(0,t]} \left(1 - \frac{\Delta N_s}{Y_s}\right)$$

by remark 3.4. The Nelson–Aalen estimator can heuristically be derived as follows. Consider the martingale

$$M_t = N_t - \int_0^t \alpha(s) Y_s ds.$$

Written using dynamics, this equation reads

$$M(dt) = N(dt) - \alpha(t) Y_t dt.$$

Dividing by Y_t, we get

$$\alpha(t)dt = \frac{1}{Y_t} N(dt) - \frac{1}{Y_t} M(dt).$$

The martingale term on the right hand side can be considered as noise, so that we informally have

$$\int_0^t \alpha(s)ds \approx \int_0^t \frac{1}{Y_s} N(ds).$$

It is possible that Y_s is zero, so we introduce J as in the definition and obtain the estimator

$$\widehat{A}_t = \int_0^t \frac{J_s}{Y_s} N(ds)$$

with the convention that $0/0 = 0$. It is convenient to introduce the process

$$A_t^* = \int_0^t \alpha(s) J_s ds$$

which we claim is the predictable compensator of the Nelson–Aalen estimator. Since Y is a càglàg process, A^* is seen to be predictable. Furthermore,

$$\begin{aligned}\widehat{A}_t - A_t^* &= \int_0^t \frac{J_s}{Y_s} N(ds) - \int_0^t \frac{J_s}{Y_s}\alpha(s)Y_s ds \\ &= \int_0^t \frac{J_s}{Y_s}(N(ds) - \alpha(s)Y_s ds) = \int_0^t \frac{J_s}{Y_s} M(ds)\end{aligned}$$

which is a martingale with mean zero. Thus A^* is indeed the compensator. Hence

$$\begin{aligned}\mathbb{E}[\widehat{A}_t] = \mathbb{E}[A_t^*] &= \int_0^t \alpha(s)\mathbb{E}[J_s]ds = \int_0^t \alpha(s)\mathbb{P}(Y_s > 0)ds \\ &= \int_0^t \alpha(s)(1 - \mathbb{P}(Y_s = 0))ds = A_t - \int_0^t \alpha(s)\mathbb{P}(Y_s = 0)ds.\end{aligned}$$

This shows that the Nelson–Aalen estimator is biased downward. However, this bias is negligible if $\mathbb{P}(Y_s = 0)$ is close to zero, which is often the case for studies with many observations. See section IV.1.1 in Andersen et al. [4] for more basic properties of the Nelson–Aalen estimator. We conclude this section by proving that the Nelson–Aalen and Kaplan–Meyer estimators are uniformly consistent on compact intervals under some mild assumptions.

Theorem 4.5. *Let a sequence of counting processes $\{N^n\}_{n\geq 1}$ be given, each N^n satisfying the multiplicative intensity model with $\lambda_t^n = \alpha(t)Y_t^n$. Let $J_t^n = \mathbb{1}_{\{Y_t^n > 0\}}$ for $n \geq 1$. Assume that for $n \to \infty$,*

(i)

$$\int_0^t \frac{J_s^n}{Y_s^n}\alpha(s)ds \xrightarrow{\mathbb{P}} 0 \quad \textit{and}$$

(ii)

$$\int_0^t (1 - J_s^n)\alpha(s)ds \xrightarrow{\mathbb{P}} 0.$$

Then as $n \to \infty$,

$$\sup_{s\in[0,t]} |\widehat{A}_s^n - A_s| \xrightarrow{\mathbb{P}} 0.$$

Proof. From the discussion earlier, the predictable variation process of the martingale $M_t^n = N_t^n - \int_0^t \alpha(s)Y_s^n ds$ is

$$\langle M^n\rangle_t = \int_0^t \alpha(s)Y_s^n ds.$$

Hence the predictable variation process of $\widehat{A} - A^*$ is given by

$$\langle \widehat{A}^n - A^{*n}\rangle_t = \int_0^t \frac{(J_s^n)^2}{(Y_s^n)^2}\alpha(s)Y_s^n ds = \int_0^t \frac{J_s^n}{Y_s^n}\alpha(s)ds.$$

By the form of Lenglart's inequality in Corollary 3.27, we have for every $\eta > 0$ and $\delta > 0$ that

$$\begin{aligned}\mathbb{P}\left(\sup_{s\in[0,t]} |\widehat{A}^n_s - A^{*n}_s| > \eta\right) &\leq \frac{\delta}{\eta^2} + \mathbb{P}(\langle \widehat{A}^n - A^*\rangle_t > \delta) \\ &= \frac{\delta}{\eta^2} + \mathbb{P}\left(\int_0^t \alpha(s)\frac{J^n_s}{Y^n_s} ds > \delta\right).\end{aligned}$$

It follows that

$$\sup_{s\in[0,t]} |\widehat{A}^n_s - A^{*n}_s| \xrightarrow{\mathbb{P}} 0$$

using assumption (i). Now apply the triangle inequality to get

$$\begin{aligned}\sup_{s\in[0,t]} |\widehat{A}^n_s - A_s| &\leq \sup_{s\in[0,t]} |\widehat{A}^n_s - A^{*n}_s| + \sup_{s\in[0,t]} |A^{*n}_s - A_s| \\ &\leq \sup_{s\in[0,t]} |\widehat{A}^n_s - A^{*n}_s| + \int_0^t \alpha(s)(1 - J^n_s) ds\end{aligned}$$

and from the above limit result and assumption (ii), the proof is complete. ■

We can now apply the product integral machinery developed in the beginning of this section to give a quick and elegant proof of uniform consistency on compact intervals for the Kaplan–Meyer estimator.

Corollary 4.6. *Let $\{N^n\}_{n\geq 1}$ be a sequence of counting processes in the survival setting. Let $t > 0$ satisfy $S(t) > 0$. Under the assumptions (i) and (ii) in the above theorem, we have*

$$\sup_{s\in[0,t]} |\widehat{S}^n(s) - S(s)| \xrightarrow{\mathbb{P}} 0$$

for $n \to \infty$.

Proof. Recall that

$$\widehat{S}^n(t) = \prod_{(0,t]} (1 - \widehat{A}^n(ds)) \quad \text{and} \quad S(t) = \prod_{(0,t]} (1 - A(ds)).$$

From the previous theorem, $\widehat{A}^n$ converges in probability to A in the supremum norm. As $\widehat{A}^n$ is non-decreasing,

$$\|\widehat{A}^n\|_v = \widehat{A}^n_t - \widehat{A}^n_0 \xrightarrow{\mathbb{P}} A_t - A_0 < \infty,$$

so by continuity of the product integral in the supremum norm, Theorem 3.14, and the continuous mapping theorem, the result follows. ■

4.3 Parametric estimation

4.3.1 The case without covariates

We now present a parametric approach to survival analysis. While the nonparametric approach presented above is very popular in biostatistics, parametric methods are often used in insurance for multistate modelling, and we later compare the performance of parametric methods with random survival forests in a simple life insurance simulation study. We follow the exposition in the book Aalen et al. [3]. For the remainder of this section, we assume that we are given counting processes $N^1, N^2, ..., N^n$ with intensity processes

$$\lambda_t^i = \alpha_i(t; \boldsymbol{\theta}) Y_t^i$$

where $\boldsymbol{\theta} = (\theta_1, ..., \theta_q)$ is a q-dimensional parameter. Later we assume that α also depends on a vector of covariates, but for now, we consider the situation without. Some classical examples of survival time distributions include the following.

Example 4.7. The *exponential* distribution. Here T° has density $f(t;\beta) = \beta e^{-\beta t}$ for $\beta > 0$ and constant intensity $\alpha(t;\beta) = \beta$. We recall that the interpretation of the hazard rate is that for a very small dt, $\alpha(t;\beta)dt$ approximately equals the probability of dying in $(t, t+dt]$ given that the individual is alive at time t. Hence a constant hazard rate implies that the probability of dying in the near future for a 20 year old is the same as for an 80 year old. This makes the exponential distribution unfit for many applications of survival analysis. Nevertheless, it constitutes a mathematically tractable example. ∘

Example 4.8. The *Weibull* distribution. Here the density of T° is $f(t;\beta,\gamma) = \beta t^{\gamma-1} e^{-\beta t^\gamma/\gamma}$ with intensity $\alpha(t;\beta,\gamma) = \beta t^{\gamma-1}$, $\beta, \gamma > 0$. Here the behaviour of the hazard rate differs depending on the value of γ. For $\gamma > 1$, the hazard is increasing while it is decreasing for $\gamma < 1$. The case $\gamma = 1$ is simply the exponential distribution. ∘

Example 4.9. The Gompertz–Makeham distribution has hazard rate $\alpha(t;\beta,\gamma,\rho) = \beta + \gamma e^{\rho t}$ for parameters $\beta, \gamma, \rho > 0$. The survival function is computed to be

$$S(t;\beta,\gamma,\rho) = \exp\left(-\int_0^t \alpha(s;\beta,\gamma,\rho)ds\right) = \exp\left(-\beta t - \frac{\gamma}{\rho}(e^{\rho t} - 1)\right)$$

which yields the density

$$f(t;\beta,\gamma,\rho) = \left(\beta + \gamma e^{\rho t}\right) \exp\left(-\beta t - \frac{\gamma}{\rho}(e^{\rho t} - 1)\right).$$

∘

In order to do estimation in the parametric setup, we need a likelihood. We present the (somewhat informal) derivation shown in section 3.2 of the book Kalbfleisch and Prentice [38]. We start by recalling that right-censoring is entirely random, $R \perp\!\!\!\perp T^\circ$. Given independent data $(T_1, \delta_1), ..., (T_n, \delta_n)$, assume T_i has distribution function F_i and density f_i, while R_i has distribution function G_i with density g_i. Then

$$\mathbb{P}(T_i \in (t, t+dt], \delta_i = 1; \boldsymbol{\theta}) = \mathbb{P}(T_i^\circ \in (t, t+dt], R_i > t; \boldsymbol{\theta}) = \overline{G}_i(t; \boldsymbol{\theta}) f_i(t; \boldsymbol{\theta}) dt$$

and

$$\mathbb{P}(T_i \in (t, t+dt], \delta_i = 0; \boldsymbol{\theta}) = \mathbb{P}(R_i \in (t, t+dt], T_i^\circ > t; \boldsymbol{\theta}) = \overline{F}_i(t; \boldsymbol{\theta}) g_i(t; \boldsymbol{\theta}) dt.$$

If censoring is *noninformative*, meaning that the distribution of the censoring variables R_i do not depend on $\boldsymbol{\theta}$, and using that the pairs (T_i, δ_i) are independent, the likelihood becomes

$$L(\boldsymbol{\theta}) \propto \prod_{i=1}^n f_i(T_i; \boldsymbol{\theta})^{\delta_i} \overline{F}_i(T_i; \boldsymbol{\theta})^{1-\delta_i}.$$

In the case where censoring is informative, $L(\boldsymbol{\theta})$ is called a *partial likelihood*. We assume noninformative censoring and therefore consider the above expression as a full likelihood. The likelihood above has the following interpretation. If $\delta_i = 1$ so that the i'th subject is known to have died at T_i, the subject contributes $f_i(T_i; \boldsymbol{\theta})$ to the likelihood. If $\delta_i = 0$, we only know that the subject was alive at time T_i, so the contribution is $\overline{F}_i(T_i; \boldsymbol{\theta})$. In order to do estimation, we rewrite the likelihood in terms of the hazard rates λ^i and counting processes N^i. As before, let

$$N_t^{(n)} = \sum_{i=1}^n N_t^i \quad \text{and} \quad \lambda_t^{(n)} = \sum_{i=1}^n \lambda_t^i,$$

and note that

$$f_i(T_i; \boldsymbol{\theta}) = \alpha_i(T_i; \boldsymbol{\theta}) \exp\left(- \int_0^{T_i} \alpha_i(t; \boldsymbol{\theta}) dt \right), \quad \overline{F}_i(T_i; \boldsymbol{\theta}) = \exp\left(- \int_0^{T_i} \alpha_i(t; \boldsymbol{\theta}) dt \right).$$

Assuming that all data is observed over the time horizon $[0, \tau]$, we have

$$\begin{aligned} L(\boldsymbol{\theta}) &= \prod_{i=1}^n \alpha_i(T_i; \boldsymbol{\theta})^{\delta_i} \exp\left(- \int_0^{T_i} \alpha_i(t; \boldsymbol{\theta}) dt \right) \\ &= \prod_{i=1}^n \prod_{0 < t \leq \tau} (\alpha_i(t; \boldsymbol{\theta} Y_t^i)^{\Delta N_t^i} \exp\left(- \int_0^\tau \alpha_i(t; \boldsymbol{\theta}) Y_t^i dt \right) \\ &= \left(\prod_{i=1}^n \prod_{0 < t \leq \tau} \lambda_t^i(\boldsymbol{\theta})^{\Delta N_t^i} \right) \exp\left(- \int_0^\tau \lambda_t^{(n)}(\boldsymbol{\theta}) dt \right) \end{aligned}$$

In practice, one has to furthermore assume that the $T_1^\circ, ..., T_n^\circ$ are iid. This simplifies the likelihood significantly in concrete examples. The following definition summarises the important quantities in estimation.

Definition 4.10. Given survival data $(T_1, \delta_1), ..., (T_n, \delta_n)$ where the true survival times T_i have intensity processes of the form $\lambda_t^i = \alpha_i(t;\theta)Y_t^i$, we define the *likelihood* by

$$L(\boldsymbol{\theta}) = \left(\prod_{i=1}^{n} \prod_{0<t\leq\tau} \lambda_t^i(\boldsymbol{\theta})^{\Delta N_t^i} \right) \exp\left(- \int_0^\tau \lambda_t^{(n)}(\boldsymbol{\theta}) dt \right).$$

The *log-likelihood* is given by

$$\ell(\boldsymbol{\theta}) = \log L(\boldsymbol{\theta}) = \sum_{i=1}^{n} \int_0^\tau \log \lambda_t^i(\boldsymbol{\theta}) N^i(dt) - \int_0^\tau \lambda_t^{(n)}(\boldsymbol{\theta}) dt.$$

The vector of *score functions*, $U(\boldsymbol{\theta}) = (U_1(\boldsymbol{\theta}), ..., U_q(\boldsymbol{\theta}))$, is defined by (assuming that we are allowed to interchange integration and differentiation)

$$U_j(\boldsymbol{\theta}) = \frac{\partial}{\partial\theta_j}\ell(\boldsymbol{\theta}) = \sum_{i=1}^{n} \int_0^\tau \frac{\partial}{\partial\theta_j} \log \lambda_t^i(\boldsymbol{\theta}) N^i(dt) - \int_0^\tau \frac{\partial}{\partial\theta_j}\lambda_t^{(n)}(\boldsymbol{\theta}) dt.$$

The *observed information matrix* $I(\boldsymbol{\theta})$ is the $q \times q$-matrix with entries

$$I(\boldsymbol{\theta})_{ij} = -\frac{\partial^2}{\partial\theta_i \partial\theta_j}\ell(\boldsymbol{\theta}),$$

and the *expected information matrix* $J(\boldsymbol{\theta})$ is the $q \times q$-matrix with entries

$$J(\boldsymbol{\theta})_{jk} = \sum_{i=1}^{n} \int_0^\tau \left(\frac{\partial}{\partial\theta_j} \log \lambda_t^i(\boldsymbol{\theta}) \right) \left(\frac{\partial}{\partial\theta_k} \log \lambda_t^i(\boldsymbol{\theta}) \right) N^i(dt).$$

We say that $\widehat{\boldsymbol{\theta}}$ is a *maximal likelihood estimator* if $\widehat{\boldsymbol{\theta}}$ is a (local) maximum of the log-likelihood $\ell(\boldsymbol{\theta})$.

Example 4.11 (**Exponential distribution**). If the $T_1^\circ, ..., T_n^\circ$ are exponentially distributed with parameter $\beta > 0$, the log-likelihood becomes

$$\begin{aligned} \ell(\beta) &= \sum_{i=1}^{n} \int_0^\tau \log Y_t^i N^i(dt) + \log\beta \sum_{i=1}^{n} \int_0^\tau N^i(dt) - \beta \int_0^\tau Y_t^{(n)} dt \\ &= \sum_{i=1}^{n} \int_0^\tau \log Y_t^i N^i(dt) + (\log\beta) N_\tau^{(n)} - \beta R(\tau) \end{aligned}$$

where

$$R(\tau) = \int_0^\tau Y_t^{(n)} dt = \sum_{i=1}^{n} \int_0^{T_i} dt = \sum_{i=1}^{n} T_i \quad \text{and} \quad N_\tau^{(n)} = \sum_{i=1}^{n} N_\tau^i$$

are the total *exposure* and *occurence* over the course of the experiment, respectively. The score function is

$$U(\beta) = \frac{N_\tau^{(n)}}{\beta} - R(\tau),$$

and the observed information is

$$I(\beta) = \frac{N_\tau^{(n)}}{\beta^2}$$

which is strictly positive. It follows that the MLE is unique and equal to

$$\widehat{\beta} = \frac{N_\tau^{(n)}}{R(\tau)}.$$

Note that the MLE is the total number of occurences divided by the total exposure. $\widehat{\beta}$ thus has the form of an *occurence/exposure rate*. We will later see more examples of O/E rates in estimation for parametric models. We also remark that the observed and expected information coincide in this example. ∘

Example 4.12 (**Gompertz–Makeham distribution**)**.** As a more complicated example in the parametric framework, consider the Gompertz–Makeham model. The log-likelihood is

$$\begin{aligned}\ell(\beta) = &\sum_{i=1}^n \int_0^\tau \log Y_t^i N^i(dt) + \sum_{i=1}^n \int_0^\tau \log(\beta + \gamma e^{\rho t}) N^i(dt) \\ &- \beta \int_0^\tau Y_t^{(n)} dt - \gamma \int_0^\tau e^{\rho t} Y_t^{(n)} dt,\end{aligned}$$

from which we can compute the score vector $U(\beta, \gamma, \rho) = (U_1(\beta, \gamma, \rho), U_2(\beta, \gamma, \rho), U_3(\beta, \gamma, \rho))$ to be

$$\begin{aligned}U_1(\beta, \gamma, \rho) &= \int_0^\tau \frac{1}{\beta + \gamma e^{\rho t}} N^{(n)}(dt) - \int_0^\tau Y_t^{(n)} dt \\ &= \sum_{i=1}^n \left(\frac{\delta_i}{\beta + \gamma e^{\rho T_i}} - T_i \right) \\ U_2(\beta, \gamma, \rho) &= \int_0^\tau \frac{e^{\rho t}}{\beta + \gamma e^{\rho t}} N^{(n)}(dt) - \int_0^\tau e^{\rho t} Y_t^{(n)} dt \\ &= \sum_{i=1}^n \left(\frac{\delta_i e^{\rho T_i}}{\beta + \gamma e^{\rho T_i}} - \frac{e^{\rho T_i} - 1}{\rho} \right) \\ U_3(\beta, \gamma, \rho) &= \int_0^\tau \frac{\gamma t e^{\rho t}}{\beta + \gamma e^{\rho t}} N^{(n)}(dt) - \gamma \int_0^\tau t e^{\rho t} Y_t^{(n)} dt \\ &= \sum_{i=1}^n \left(\frac{\gamma \delta_i T_i e^{\rho T_i}}{\beta + \gamma e^{\rho T_i}} - \gamma \left(\frac{T_i e^{\rho T_i}}{\rho} - \frac{e^{\rho T_i} - 1}{\rho^2} \right) \right).\end{aligned}$$

In this case, one has to solve numerically for an MLE. ∘

Example 4.13 (**Piecewise constant hazards**). If one does not want to pose strong parametric assumptions on α, a possible approach is to assume that α is piecewise constant. Consider a partition $0 = t_0 < t_1 < \cdots < t_K = \tau$, then we assume

$$\alpha(t; \boldsymbol{\theta}) = \sum_{k=1}^{K} \theta_k I_k(t)$$

with $\boldsymbol{\theta} = (\theta_1, ..., \theta_K)$ and $I_k(t) = \mathbb{1}_{(t_{k-1}, t_k]}(t)$. We can compute the score functions directly,

$$\begin{aligned} U_j(\boldsymbol{\theta}) &= \sum_{i=1}^{n} \int_0^\tau \frac{\partial}{\partial \theta_j} \left(\log Y_t^i + \log \left(\sum_{k=1}^{K} \theta_k I_k(t) \right) \right) N^i(dt) - \sum_{i=1}^{n} \int_0^\tau \frac{\partial}{\partial \theta_j} Y_t^i \sum_{k=1}^{K} \theta_k I_k(t) dt \\ &= \sum_{i=1}^{n} \int_0^\tau \frac{I_j(t)}{\sum_{k=1}^{K} \theta_k I_k(t)} N^i(dt) - \sum_{i=1}^{n} \int_0^\tau I_j(t) Y_t^i dt \\ &= \frac{1}{\theta_j} \int_0^\tau I_j(t) N^{(n)}(dt) - \int_0^\tau I_j(t) Y_t^{(n)} dt. \end{aligned}$$

Defining the quantities

$$O_j = \int_0^\tau I_j(t) N^{(n)}(dt) \quad \text{and} \quad R_j = \int_0^\tau I_j(t) Y_t^{(n)} dt,$$

we see that $U_j(\boldsymbol{\theta}) = O_j/\theta_j - R_j$ where O_j is the total number of events in the interval $(t_{j-1}, t_j]$, and R_j is the total exposure in the interval. The observed information matrix is

$$I(\boldsymbol{\theta}) = \begin{pmatrix} \frac{O_1}{\theta_1^2} & & & \\ & \frac{O_2}{\theta_2^2} & & \\ & & \ddots & \\ & & & \frac{O_K}{\theta_K^2} \end{pmatrix}$$

which is clearly positive definite. Hence the MLE for θ_j is again an occurence/exposure rate

$$\widehat{\theta}_j = \frac{O_j}{R_j}.$$

∘

4.3.2 Including covariates - Poisson regression

We now consider the case where covariates are included. Hence we assume that the intensities λ_t are of the form

$$\lambda_t^i = \alpha(t \mid \mathbf{X}_i) Y_t^i$$

where $\mathbf{X}_i$ as usual denotes the vector of covariates for the i'th observation. We will specialise further and consider intensities of the form

$$\lambda_t^i(\boldsymbol{\theta}, \boldsymbol{\beta}) = \alpha_0(t; \boldsymbol{\theta}) Y_t^i r(\boldsymbol{\beta}, \mathbf{X}_i).$$

We interpret α_0 as a baseline hazard function common for all individuals in the study while r describes the effect of the covariates. We refer to r as a *relative risk function*. A popular choice of r is

$$r(\boldsymbol{\beta}, \mathbf{X}) = \exp(\boldsymbol{\beta}^T \mathbf{X})$$

where it is assumed that the dimensions of $\mathbf{X}$ and $\boldsymbol{\beta}$ are compatible. With this choice of relative risk function, and if one assumes no further structure on α_0, one obtains the famous Cox proportional hazards model, see the original paper by Cox, Cox [16]. The tremendous impact of this model is summarised in the article Kalbfleisch and Schaubel [39]. In our exposition, we will to start with make no assumption on the form of r, and instead of leaving α_0 unspecified as in the Cox model, we will assume that α_0 is piecewise constant,

$$\alpha_0(t; \boldsymbol{\theta}) = \sum_{k=1}^{K} \theta_k I_k(t)$$

with I_k defined as in the example on piecewise constant hazards above. Now consider the total number of events for individual i in the k'th subinterval,

$$O_{ik} = \int_0^\tau I_k(t) N^i(dt)$$

as well as the total exposure for individual i in the subinterval,

$$R_{ik} = \int_0^\tau I_k(t) Y_t^i dt.$$

Using these quantities, we can rewrite the likelihood

$$L(\boldsymbol{\theta}, \boldsymbol{\beta}) = \prod_{i=1}^{n} \prod_{0<t\leq\tau} \lambda_t^i(\boldsymbol{\theta}, \boldsymbol{\beta})^{\Delta N_t^i} \exp\left(- \int_0^\tau \lambda_t^{(n)}(\boldsymbol{\theta}, \boldsymbol{\beta}) dt \right)$$

as

$$\begin{aligned} L(\boldsymbol{\theta}, \boldsymbol{\beta}) &= \prod_{i=1}^{n} \prod_{k=1}^{K} (\theta_k r(\boldsymbol{\beta}, \mathbf{X}_i))^{O_{ik}} \exp\left(- \sum_{i=1}^{n} \sum_{k=1}^{K} \theta_k r(\boldsymbol{\beta}, \mathbf{X}_i) R_{ik} \right) \\ &= \prod_{k=1}^{K} \prod_{i=1}^{n} \left((\theta_k r(\boldsymbol{\beta}, \mathbf{X}_i))^{O_{ik}} \exp(-\theta_k r(\boldsymbol{\beta}, \mathbf{X}_i) R_{ik}) \right) \end{aligned}$$

by simply regrouping the terms in the inner product over t according to which time interval the jump occurs in. We see that this likelihood is proportional to the likelihood one would obtain by assuming that the O_{ik} are independent and Poisson distributed with parameters

$$\mu_{ik} = \theta_k r(\boldsymbol{\beta}, \mathbf{X}_i) R_{ik}.$$

This observation is the reason why this model is called a *Poisson regression model*, even though O_{ik} is not assumed to be Poisson, and R_{ik} is not a fixed quantity.

Nevertheless, this observation is extremely useful for computational purposes since it allows us to use efficient software for estimation in generalised linear models when $r(\boldsymbol{\beta}, \mathbf{X}) = \exp(\boldsymbol{\beta}^T \mathbf{X})$. To be explicit, we would then have

$$\mu_{ik} = \exp(\log \theta_k + \boldsymbol{\beta}^T \mathbf{X}_i + \log R_{ik}).$$

Hence we can simply, for each $k = 1, ..., K$, fit a Poisson GLM to the data O_{ik}, $i = 1, ..., n$ with $\log R_{ik}$ as offsets and $\log \theta_k$ as the intercept.

We wrap up our discussion on parametric estimation by mentioning that several asymptotic results exist for the maximum likelihood estimator $\widehat{\boldsymbol{\theta}}$. Under some quite technical regularity assumptions (see Condition VI.1.1 in Andersen et al. [4]), one can show that with probability tending to one, the score equation $U(\boldsymbol{\theta}) = 0$ has a solution $\widehat{\boldsymbol{\theta}}$ with $\widehat{\boldsymbol{\theta}} \xrightarrow{\mathbb{P}} \boldsymbol{\theta}_0$. Under the same assumptions, asymptotic normality can also be established. A thourough discussion on large sample properties in the parametric regime may be found in chapter IV.1.2 of Andersen et al. [4].

4.4 More on the Gompertz–Makeham distribution

When we later do a life insurance related simulation study, we simulate from a Gompertz–Makeham distribution. This particular distribution was chosen since it strikes a good balance between complexity and tractability. We recall from earlier that the intensity is given by

$$\alpha(t; \beta, \gamma, \rho) = \beta + \gamma e^{\rho t}$$

which yields the survival function

$$S(t; \beta, \gamma, \rho) = \exp\left(-\beta t - \frac{\gamma}{\rho}(e^{\rho t} - 1)\right).$$

The Gompertz–Makeham distribution is important for historical reasons. The mortality table for males in 1982 used by Danish companies (known as G82M) is obtained by using a Gompertz–Makeham mortality model where

$$\beta = 5 \cdot 10^{-4}, \quad \gamma = 7.5858 \cdot 10^{-5} \quad \text{and} \quad \rho = \log(1.09144),$$

see the discussion in section 2.1 in Furrer [19].

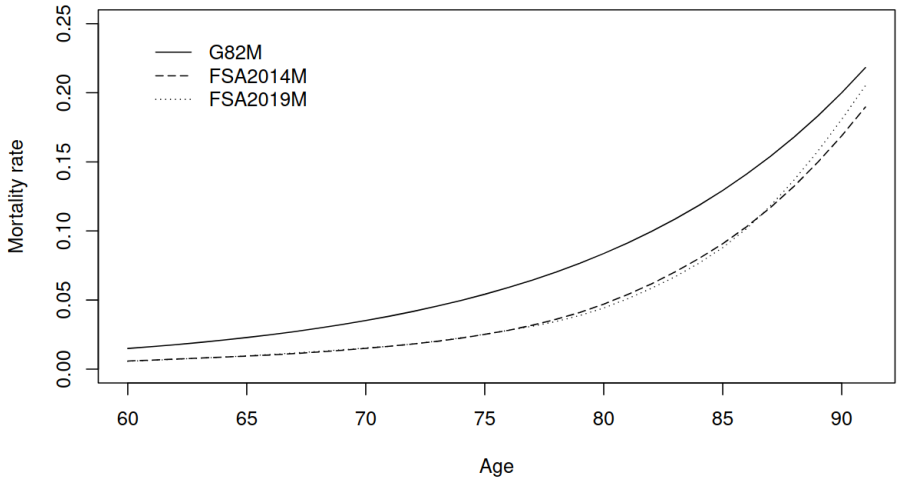

Figure 8: Three mortality curves for males for ages 60 to 90 taken from Furrer [19]. Since 2011, Danish life insurance companies have had to use the so-called *Levetidsmodel* from the Danish Financial Supervisory Authority (FSA). The FSA mortality curves for males from 2014 and 2019 are plotted along with the G82M model. Lifetimes have clearly improved since 1982.

Looking again at the intensity $\alpha(t;\beta,\gamma,\rho) = \beta + \gamma e^{\rho t}$, we see that in this model, mortality is modelled as an exponential function of age. More precisely, β is an age-independent component which attempts to capture all effects on mortality not related to age. γ is a baseline hazard at age zero, while ρ determines how fast mortality increases with age. An exponential model for the hazard rate was first proposed by Benjamin Gompertz, see Gompertz [23], which is why $\gamma e^{\rho t}$ is sometimes referred to as the *Gompertz term*. William M. Makeham later proposed adding the age-independent term β, see Makeham [47, 48, 49]. For this reason, β is sometimes referred to as the *Makeham term*. For a modern summary of results and properties of the Gompertz–Makeham distribution, we refer to Castellares et al. [13].

For the rest of this subsection, we supply the details needed in order to do efficient simulation from the Gompertz–Makeham distribution. We are interested in being able to simulate a portfolio of policyholders. These individuals will of course have different ages when the contract is initiated. Let T be the lifetime of a policyholder, assumed to follow a Gompertz–Makeham distribution, and let x be the age at time zero. We are then interested in the *residual lifetime* $T_x := T - x \mid T > x$. The survival function S_x of T_x is

$$S_x(t) = \mathbb{P}(T - x > t \mid T > x) = \frac{\mathbb{P}(T > t + x)}{\mathbb{P}(T > x)} = \frac{S(t+x)}{S(x)}$$

with S the survival function of T. Hence

$$S_x(t) = \frac{\exp\left(-\int_0^{t+x} \alpha(s;\beta,\gamma,\rho)ds\right)}{\exp\left(-\int_0^{x} \alpha(s;\beta,\gamma,\rho)ds\right)} = \exp\left(-\int_x^{t+x} \alpha(s;\beta,\gamma,\rho)ds\right),$$

and an elementary computation yields

$$S_x(t) = \exp\left(-\beta t - \frac{\gamma}{\rho}(e^{\rho(t+x)} - e^{\rho x})\right).$$

The goal is to simulate from this function using the inverse transform method. Recall that if we want to simulate from the distribution with distribution function F, we simulate $U \sim U(0,1)$ and apply the generalised inverse (or quantile function) q of F on U. The random variable $q(U)$ will then have distribution F. When F is absolutely continuous, q is simply the ordinary inverse. In order to use this method, we determine the quantile function of the Gompertz–Makeham distribution explicitly as done in Jodrá [37]. The expression for the quantile function involves (the principal branch of) the Lambert W function, and we therefore recall the definition.

Definition 4.14. The Lambert W function is the solution to the equation

$$W(z)e^{W(z)} = z, \quad z \in \mathbb{C}.$$

The Lambert W function is a multivalued complex function with two real branches. In both of these branches, z is restricted to the interval $[-1/e, \infty)$. The real branch taking values in $(-\infty, -1]$ is called the *negative branch* and is denoted by W_{-1}, while the real branch with values in $[-1, \infty)$ is called the *principal branch* and is denoted by W_0. We only use the principal branch in the following. Note that $W_0(-1/e) = -1$ and $W_0(0) = 0$. The history of the Lambert W function as well as several applications of the function can be found in Corless et al. [15].

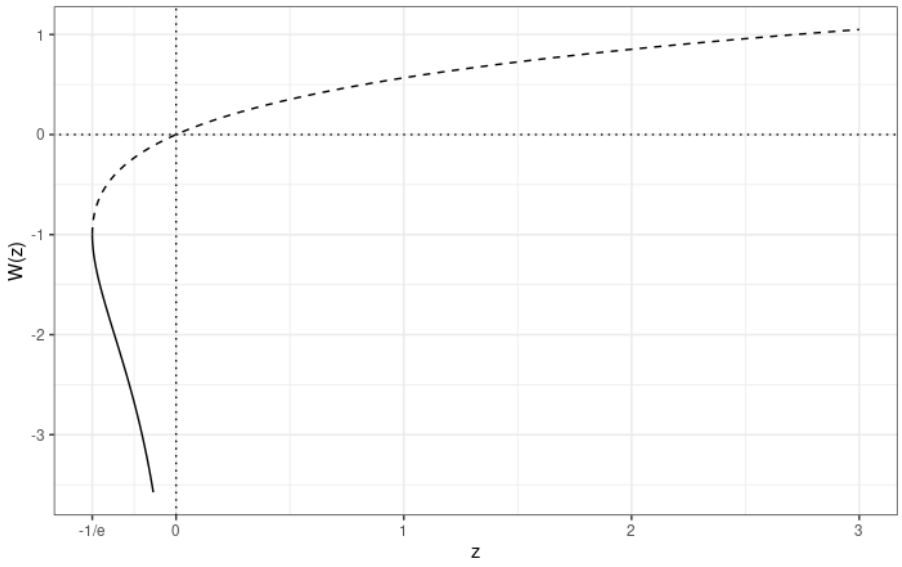

Figure 9: The principal branch W_0 (dashed line) and the negative branch W_{-1} (solid line) of the Lambert W function. The Lambert W function is computed using the `lambertW` function in the R package `emdBook`, Bolker et al. [7].

The goal is to derive the quantile function q_x of F_x. We proceed in two steps. We first compute the quantile function q of F and afterwards, we use a general result applied to q to compute q_x. The following lemma is essential.

Lemma 4.15. *Let $a, b, c > 0$ and $z \in \mathbb{R}$. The solution to the equation*

$$z + ae^{bz} = c$$

in terms of z is given by

$$z = c - \frac{1}{b} W_0(abe^{bc}).$$

Proof. The proof is a simple matter of rewriting the equation $z + ae^{bz} = c$. We get

$$\begin{aligned} z + ae^{bz} = c \quad &\Leftrightarrow \quad c - z = ae^{bc} \\ &\Leftrightarrow \quad (c-z)e^{-bc} = a \\ &\Leftrightarrow \quad b(c-z)e^{b(c-z)} = abe^{bc} \end{aligned}$$

which by definition of the Lambert W function means that $W_0(abe^{bc}) = b(c-z)$. We have to use the principal branch since the right hand side is positive. Now simply isolate z to get the desired result. ■

Theorem 4.16. *The quantile function q of the Gompertz–Makeham distribution with parameters β, γ, ρ is given by*

$$q(p) = \frac{\gamma}{\beta\rho} - \frac{1}{\beta}\log(1-p) - \frac{1}{\rho} W_0\left(\frac{\gamma}{\beta}\exp\left(\frac{\gamma}{\beta}\right)(1-p)^{-\rho/\beta}\right), \quad p \in (0,1).$$

Proof. Given $p \in (0,1)$, we have to solve the equation $S(t) = 1 - p$ in terms of t. We recall that

$$S(t) = \exp\left(-\beta t - \frac{\gamma}{\rho}(e^{\rho t} - 1)\right).$$

We solve the equation $S(t) = 1 - p$:

$$\begin{aligned} \exp\left(-\beta t - \frac{\gamma}{\rho}(e^{\rho t} - 1)\right) = 1 - p \quad &\Leftrightarrow \quad \exp\left(-\beta t - \frac{\gamma}{\rho}(e^{\rho t} - 1) - \log(1-p)\right) = 1 \\ &\Leftrightarrow \quad \beta t + \frac{\gamma}{\rho}(e^{\rho t} - 1) + \log(1-p) = 0 \\ &\Leftrightarrow \quad t + \frac{\gamma}{\beta\rho} e^{\rho t} = \frac{\gamma}{\beta\rho} - \frac{1}{\beta}\log(1-p). \end{aligned}$$

Note that the right hand side is strictly positive and that $\gamma/\beta\rho > 0$ and $t > 0$. Using Lemma 4.15 with

$$a = \frac{\gamma}{\beta\rho}, \quad b = \rho \quad \text{and} \quad c = \frac{\gamma}{\beta\rho} - \frac{1}{\beta}\log(1-p),$$

we obtain

$$\begin{aligned} t &= \frac{\gamma}{\beta\rho} - \frac{1}{\beta}\log(1-p) - \frac{1}{\rho}W_0\left(\frac{\gamma}{\beta}\exp\left(\frac{\gamma}{\beta} - \frac{\rho}{\beta}\log(1-p)\right)\right) \\ &= \frac{\gamma}{\beta\rho} - \frac{1}{\beta}\log(1-p) - \frac{1}{\rho}W_0\left(\frac{\gamma}{\beta}\exp\left(\frac{\gamma}{\beta}\right)(1-p)^{-\rho/\beta}\right), \end{aligned}$$

which completes the proof. ■

The final step in computing q_x follows from a more general result provided in the following lemma.

Lemma 4.17. *Let F be a continuous distribution function for the non-negative random variable T with quantile function q. For $x > 0$, let F_x denote the distribution function of $T - x \mid T > x$. The quantile function q_x of F_x is given by*

$$q_x(p) = q(1 - (1-p)S(x)) - x, \quad p \in (0,1)$$

where as usual, $S = 1 - F$ denotes the survival function of F.

Proof. We wish to solve $F_x(t) = p$ in terms of t. If $S_x = 1 - F_x$, this amounts to solving $S_x(t) = 1 - p$. Using that $S_x(t) = S(t+x)/S(x)$, this is equivalent to $S(x+t) = (1-p)S(x)$ so that $F(x+t) = 1 - (1-p)S(x)$. Applying the quantile function to both sides yields $x + t = q(1 - (1-p)S(x))$, and the desired expression for q_x follows immediately. ■

Corollary 4.18. *If F is the distribution function for the Gompertz–Makeham distribution, F_x has quantile function q_x given by*

$$q_x(p) = \frac{\gamma}{\beta\rho} - \frac{1}{\beta}\log\left((1-p)S(x)\right) - \frac{1}{\rho}W_0\left(\frac{\gamma}{\beta}\exp\left(\frac{\gamma}{\beta}\right)\left((1-p)S(x)\right)^{-\rho/\beta}\right) - x$$

Proof. Combine the lemma just presented with Theorem 4.16. ■

This corollary is the key result that we will use to simulate the portfolio of insured in the next section. To compute the principal branch W_0, we use the `lambertW` function in the R package `emdBook`.

4.5 Evaluating a survival model - Harrell's C-index

Evaluating a model in regression or classification is straightforward, but the situation is not as clear for survival models. The evaluation metric should take both the observations and the censoring into account and relate these quantities to the output of the model in a way that is easy to interpret. One such measure is *Harrell's concordance index (henceforth referred to as Harrell's C-index or simply the C-index)* as presented in Harrell et al. [25], see in particular the paragraph below the headline "Statistical Calculations". More discussion on the C-index including the estimator presented below can be found in Longato et al. [46]. Harrell's C-index takes censoring into account and is easy to compute and interpret, which makes it a popular

measure for survival settings. We here present the idea for a general survival setting and later specialise to the case of random survival forests. The idea is to form all possible pairs of T_i and T_j and determine if the smaller of the two also has a "worse" predicted outcome, in which case we say that (T_i, T_j) (or simply (i, j)) is *concordant*. If it is not possible to compare T_i and T_j in a meaningful way, we say that (i, j) is not a *permissible* pair. Harrell's C-index is then simply the ratio of the number of concordant and permissible pairs.

To be more precise, assume that every observation (T_i, δ_i) has an associated "risk measure" M_i which is a quantity related to the likelihood of experiencing an event (death) for subject i. This risk measure is computed from the predictions of the model in some way. Harrell's C-index C is then defined by

$$C = \mathbb{P}(M_i > M_j \mid T_i^\circ < T_j^\circ),$$

that is, the probability of the model predicting a worse outcome for i given that i actually experiences the event first for a randomly selected pair of observations (i, j). Note that $C = 1$ means that the model exhibits perfect concordance. In the case of no ties in the data or in the risk measures, Harrell's estimator is given by

$$\widehat{C} = \frac{\sum_{i=1}^n \delta_i \sum_{j=i+1}^n \mathbb{1}_{\{T_i < T_j\}} \mathbb{1}_{\{M_i > M_j\}}}{\sum_{i=1}^n \delta_i \sum_{j=i+1}^n \mathbb{1}_{\{T_i < T_j\}}}.$$

This estimator is readily extended to the case where ties ($T_i = T_j$ or $M_i = M_j$) are possible. The form of the general estimator becomes clear when we present the algorithm to compute it below. If ties are possible, the general estimator becomes

$$\widehat{C} = \frac{\sum_{i=1}^n \delta_i \sum_{j=i+1}^n (\mathbb{1}_{\{T_i < T_j\}} + (1 - \delta_j)\mathbb{1}_{\{T_i = T_j\}})(\mathbb{1}_{\{M_i > M_j\}} + \frac{1}{2}\mathbb{1}_{\{M_i = M_j\}})}{\sum_{i=1}^n \delta_i \sum_{j=i+1}^n (\mathbb{1}_{\{T_i < T_j\}} + (1 - \delta_j)\mathbb{1}_{\{T_i = T_j\}})}.$$

We use a straightforward algorithm to compute $\widehat{C}$. It is given as follows.

1. Initialise a numerator, `Concordance`, and a denominator, `Permissible`, both equal to zero.
2. Do the following for every $i = 1, ..., n$ and every $j = i + 1, ..., n$:
   (a) If $T_i < T_j$ and $\delta_i = 0$ or $T_j < T_i$ and $\delta_j = 0$, the pair (i, j) is not comparable, so skip this iteration of the j loop. Do the same if $T_i = T_j$ and $\delta_i = \delta_j$.
   (b) Add 1 to `Permissible`. Then do the following.
      i. If $T_i < T_j$ and $M_i > M_j$, the model correctly estimates that i has a higher risk than j. Add 1 to `Concordance`.
      ii. Repeat the previous step with i and j reversed.
      iii. If $M_i = M_j$, the model is inconclusive. Add 0.5 to `Concordance`.
3. Return Harrell's C-index given by `Concordance/Permissible`.

The large sample properties of $\widehat{C}$ are briefly discussed in Longato et al. [46]. One can show that

$$\widehat{C} \to \mathbb{P}(M_i > M_j \mid T_i^\circ < T_j^\circ, \delta_i = 1, T_i^\circ < \tau) \quad \text{for} \quad n \to \infty$$

where τ is the largest observed event time (the type of convergence is not further specified). Longato et al. [46] discusses the implication that the presence of τ has on interpretation. We shall not worry about this aspect and simply interpret $\widehat{C}$ as an estimate of C. In model evaluation, it is custom to consider the error rate E given by

$$E = 1 - C.$$

Recall that we can interpret C as the probability that given two random individuals, the individual with the worse predicted outcome is outlived by the other. Values of C (or E) around 0.5 indicate that the model is no better than random guessing while $E = 0$ indicates perfect accuracy. As a final remark, note that a very bad model (with E close to 1) can be turned into a "good" model when using this metric, since we can simply reverse the interpretation of the risk measure used. Therefore, a model is only truly bad if E is close to 0.5.

5 Random Survival Forests

"You can't see the wood for the trees."
- Proverb[2]

The random survival forest (RSF) is a relatively new method for estimating mortality rates in survival analysis. The method works analogously to the classification and regression setups with the key difference that the predicted value of a tree is a function, namely the Nelson–Aalen estimator for the data in the relevant node, and not a single number. In particular, the labels and the predicted values are not of the same type as in the case of classification and regression trees. We start by briefly presenting the relevant notation and then quickly move on to the algorithm itself. The details of the algorithm are then discussed in subsequent subsections. After the algorithm is presented and explained, we prove consistency of RSF and demonstrate the cababilities of RSF with a practical example. We closely follow the outline of the article Ishwaran et al. [32]. The technical details concerning the splitting rules are presented in the article Ishwaran and Kogalur [30]. The proof of consistency is more or less the same as in Ishwaran and Kogalur [31] but under a different set of assumptions.

5.1 Notation and terminology

In this section,we are given survival data of the form $(\mathbf{X}, T, \delta)$ with T the observed survival time, δ the censoring indicator and $\mathbf{X}$ a d-dimensional vector of features. We consider an iid sample $(\mathbf{X}_1, T_1, \delta_1), ..., (\mathbf{X}_n, T_n, \delta_n)$. We use the following terminology for a given data point $(\mathbf{X}_i, T_i, \delta_i)$. If $\delta_i = 0$, the i'th individual has been (right-) censored. If $\delta_i = 1$, we call T_i an event time or a time of death. Hence the term "death" refers to a true/observed death. A tree will be denoted by $\mathcal{T}$ and $\mathcal{N}(\mathcal{T})$ will denote the set of terminal nodes.

5.2 The RSF algorithm

RSF largely follows the procedure for general random forests. An outline of the algorithm is as follows:

1. Draw B bootstrap samples with replacement from the original data. The data not included in a particular bootstrap sample is referred to as *out-of-bag data* (OOB data).

2. Grow a survival tree on each bootstrap sample. Whenever we make a split, randomly select p candidate features from $\mathbf{X}$. Only these features will be used as candidates for the split. We use a splitting rule that maximises survival difference in the two daughter nodes. Different splitting rules are discussed in a later subsection. The tree is grown to full size under the condition that a terminal node should have at least $d_0 > 0$ unique deaths.

[2] Attributed to Sir Thomas More's *Confutacion of Tyndals Answere*, 1533.

3. Calculate the Nelson–Aalen estimator in each node.

4. Compute error estimates in the form of Harrell's C-index.

The algorithm as stated above requires three hyperparameters. These are

- The number of trees, B.
- The number of features selected in a split, p.
- The minimum number of unique deaths in a terminal node, d_0. In practice, this is implemented as simply the minimum number of observations (censored or not) in a node just like for classicication and regression.

The first two points above work in exactly the same way as an ordinary random forest. We just need to specify what splitting rule to apply. Before doing so, we need to establish conservation of events, a topic we will discuss soon. Let us first discuss prediction for a survival tree.

5.3 Prediction

Say we have fitted a survival tree $\mathcal{T}$ and consider a terminal node $h \in \mathcal{N}(\mathcal{T})$ with n_h data points. We let $(T_{1,h}, \delta_{1,h}), ..., (T_{n_h,h}, \delta_{n_h,h})$ denote the survival times and censoring indicators in h. Let $t_{1,h} < t_{2,h} < \cdots < t_{N_h,h}$ denote the N_h unique event times. Define the number of individuals at risk at time t,

$$Y_t^h = \sum_{i=1}^{n_h} \mathbb{1}_{\{T_{i,h} \geq t\}}$$

and the number of events that have occured by time t,

$$N_t^h = \sum_{i=1}^{n_h} \mathbb{1}_{\{T_{i,h} \leq t\}} \delta_{i,h}.$$

Furthermore, let $J_t^h = \mathbb{1}_{\{Y_t^h > 0\}}$. The predicted value for h is then the Nelson–Aalen estimator

$$\widehat{H}_h(t) = \int_0^t \frac{J_s^h}{Y_s^h} N^h(ds)$$

which we write in the more convenient form

$$\widehat{H}_h(t) = \sum_{l : t_{l,h} \leq t} \frac{d_{l,h}}{Y_{l,h}} \tag{7}$$

with $Y_{l,h} = Y_{t_{l,h}}$ and $d_{l,h}$ the number of deaths at time $t_{l,h}$, $d_{l,h} = \#\{k : T_{k,h} = t_{l,h}, \delta_{k,h} = 1\}$. All data points within h have the same predicted cumulative hazard function (CHF). To predict using our fitted tree $\mathcal{T}$, we simply drop the vector of covariates down the tree. Let $\mathbf{x}$ be a feature vector. $\mathbf{x}$ will fall into a unique

terminal node due to the way a survival tree is constructed. This allows us to define the Nelson–Aalen estimator for the conditional CHF $H(t \mid \mathbf{x})$ by

$$\widehat{H}(t \mid \mathbf{x}) = \widehat{H}_h(t), \quad \text{if } \mathbf{x} \in h. \tag{8}$$

When we grow a forest of survival trees, there are two ways of computing an ensemble CHF. Let $\widehat{H}^*_b(t \mid \mathbf{x})$ denote the estimated CHF (8) for the b'th bootstrap sample. We define $I_{i,b}$ by

$$I_{i,b} = \begin{cases} 1, & \text{if } i \text{ is an OOB case for } b \\ 0, & \text{otherwise} \end{cases}.$$

We then define the *OOB ensemble CHF* for i by

$$\widehat{H}^{**}_e(t \mid \mathbf{x}_i) = \frac{\sum_{b=1}^B I_{i,b} \widehat{H}^*_b(t \mid \mathbf{x}_i)}{\sum_{b=1}^B I_{i,b}}.$$

This corresponds to taking the average over the bootstrap samples where i is OOB. Another choice is the *bootstrap ensemble CHF* for i given by

$$\widehat{H}^*_e(t \mid \mathbf{x}_i) = \frac{1}{B} \sum_{b=1}^B \widehat{H}^*_b(t \mid \mathbf{x}_i).$$

This is just the usual aggregated prediction over all bootstrap samples.

5.4 Conservation of events

Conservation of events is a useful result that we will apply in defining one of the possible splitting rules for survival trees. Informally, conservation of events states that the total number of deaths in a node (or a tree) can be recovered from the Nelson–Aalen estimator. More precisely, the sum of the Nelson–Aalen estimator evaluated at every observed time (censored and uncensored) equals the total number of deaths. This principle was introduced in unpublished notes by Naftel, Blackstone and Turner in 1985.

Lemma 5.1 (**Conservation of events**). *For every terminal node h in a survival tree $\mathcal{T}$, we have*

$$\sum_{i=1}^{n_h} \widehat{H}_h(T_{i,h}) = \sum_{i=1}^{n_h} \delta_{i,h}.$$

Proof. By reindexing the data if necessary, we may assume without loss of generality that $T_{1,h} \leq T_{2,h} \leq \cdots \leq T_{n_h,h}$. To prove the result, note that

$$\sum_{i=1}^{n_h} \widehat{H}_h(T_{i,h})$$

is a double-sum where the terms can be regrouped in the following way. Consider the first i such that $\delta_{i,h} = 1$ (if such an i does not exist, both sums are zero and the

result is trivially true). Denote this i by i_1. Now note that the term

$$\frac{d_{i_1,h}}{Y_{i_1,h}}$$

will appear in the sum $\sum_{i=1}^{n_h} \widehat{H}_h(T_{i,h})$ exactly $Y_{i_1,h}$ times. Hence the total contribution to the sum is $d_{i_1,h}$. Jumping to the next $\delta_{i,h}$ equal to one (with index i_2), we can apply the same logic to see that the term $\frac{d_{i_2,h}}{Y_{i_2,h}}$ appears $Y_{i_2,h}$ times for a total contribution of $d_{2,h}$. Continuing this line of argument, we have

$$\sum_{i=1}^{n_h} \widehat{H}_h(T_{i,h}) = d_{1,h} + d_{2,h} + \cdots d_{N_h,h} = \sum_{i=1}^{n_h} \delta_{i,h}$$

as desired. ■

The following corollary extends conservation of events to a whole tree. The interpretation is similar, namely that the total number of deaths in the data can be recovered from all survival times and the Nelson–Aalen estimator.

Corollary 5.2. *For any survival tree $\mathcal{T}$, we have*

$$\sum_{i=1}^{n} \widehat{H}(T_i \mid \mathbf{x}_i) = \sum_{i=1}^{n} \delta_i.$$

Proof. Using the previous lemma, we have

$$\sum_{i=1}^{n} \widehat{H}(T_i \mid \mathbf{x}_i) = \sum_{h \in \mathcal{N}(\mathcal{T})} \sum_{i=1}^{n_h} \widehat{H}_h(T_{i,h}) = \sum_{h \in \mathcal{N}(\mathcal{T})} \sum_{i=1}^{n_h} \delta_{i,h} = \sum_{i=1}^{n} \delta_i.$$

■

5.5 Splitting rules

We now discuss the most technical aspect of growing a survival tree, namely the splitting rules. The setup is the following. We are in a node h and seek to split h into two daughters. The split itself works like the classification and regression case, namely by selecting a certain feature, which we will simply denote by x, and a threshold c. All cases $(\mathbf{x}, T, \delta)$ with $x \leq c$ go the left node and the remaining cases go to the right node. A split is then identified with the pair (x, c). A splitting rule $L(x, c)$ evaluates how good a split is. In the classification case, a good split creates a large difference in proportions between the daughter nodes, while in regression, a good split creates a large difference in the means of the daughters. Analogously, a good split in a survival tree maximises the difference in survival. Before presenting five choices of splitting function L, we fix some notation.

Let $(T_1, \delta_1), ..., (T_n, \delta_n)$ denote the data in h (for the sake of readability, we suppress the dependence on h in the notation). We let $t_1 < t_2 < \cdots < t_N$ denote the distinct

event times in the parent node h and let $d_{i,j}$ and $Y_{i,j}$ denote the number of deaths and number of individuals at risk at time t_i in the daughter nodes $j = 1, 2$. Explicitly,

$$\begin{aligned} Y_{i,1} &= \#\{l : T_l \geq t_i, x_l \leq c\}, \\ d_{i,1} &= \#\{l : T_l = t_i, \delta_l = 1, x_l \leq c\}, \\ Y_{i,2} &= \#\{l : T_l \geq t_i, x_l > c\}, \\ d_{i,2} &= \#\{l : T_l = t_i, \delta_l = 1, x_l > c\}, \end{aligned}$$

with x_l denoting the value of feature x for individual l. Finally, we let $Y_i = Y_{i,1} + Y_{i,2}, d_i = d_{i,1} + d_{i,2}$ and denote by n_j the total number of observations in daughter j, so that $n = n_1 + n_2$ with $n_1 = \#\{l : x_l \leq c\}$ and $n_2 = \#\{l : x_l > c\}$.

5.5.1 Log-rank splitting

The log-rank test for a split (x, c) is given by

$$L_{LR}(x, c) = \frac{\sum_{i=1}^{N} \left(d_{i,1} - Y_{i,1} \frac{d_i}{Y_i} \right)}{\sqrt{\sum_{i=1}^{N} \frac{Y_{i,1}}{Y_i} \left(1 - \frac{Y_{i,1}}{Y_i} \right) \left(\frac{Y_i - d_i}{Y_i - 1} \right) d_i}}.$$

The larger the value $|L_{LR}(x, c)|$, the greater the survival difference between the two nodes. Hence the best split is found by determining the pair (x^*, c^*) such that $|L_{LR}(x, c)|$ is maximised,

$$(x^*, c^*) = \underset{(x,c)}{\operatorname{argmax}} |L_{LR}(x, c)|.$$

As discussed and demonstrated in the article LeBlanc and Crowley [42], the log-rank test is robust in both proportional and non-proportional hazard settings, and it has become a popular splitting rule in the survival tree litterature. Nevertheless, it has been criticised for having an *end-cut-preference*, a term originating from Breiman et al. [11], which means that the test prefers splits that send a large majority of the observations to one node. The log-rank test belongs to the class of TW statistics, see the article Segal [51] for more details.

5.5.2 Conservation of events splitting

While the log-rank test is often a good choice, it is of interest to present an alternative which can help tackle the potential problem of end-cut-preference. One such splitting rule is conservation of events splitting. In daughter node j, the conservation of events principle established earlier yields

$$\sum_{l=1}^{n_j} \widehat{H}_j(T_{l,j}) = \sum_{l=1}^{n_j} \delta_{l,j}.$$

To define the conservation of events splitting rule, first order the time points within each daughter,

$$T_{(1),j} \leq T_{(2),j} \leq \cdots \leq T_{(n_j),j}$$

with $\delta_{(l),j}$ denoting the censoring indicator belonging to $T_{(l),j}$. Define the quantities

$$\mathcal{M}_{k,j} = \sum_{l=1}^{k} \widehat{H}_j(T_{(l),j}) - \sum_{l=1}^{k} \delta_{(l),j}, k = 1, ..., n_j.$$

One can interpreet $\mathcal{M}_{k,j}$ as a "residual" that measures how preserved conservation of events is within each group. Note that $\mathcal{M}_{n_j,j} = 0$. The *measure of conservation* $C(x, c)$ is then defined by

$$C(x,c) = \frac{1}{Y_1}\left(Y_{1,1}\sum_{k=1}^{n_1-1}|\mathcal{M}_{k,1}| + Y_{1,2}\sum_{k=1}^{n_2-1}|\mathcal{M}_{k,2}|\right)$$

which is a sum of the $|\mathcal{M}_{k,1}|$ and $|\mathcal{M}_{k,2}|$ weighted by the number of individuals in each node. A low value of $C(x, c)$ indicates that the nodes are well separated. In most implementations of decision trees, the splitting function is maximised to find the optimal split, so we instead consider the transformed value

$$L_C(x,c) = \frac{1}{1 + C(x,c)}.$$

The best split is then found by determining (x, c) such that $L_C(x, c)$ just defined is maximal. The expression $C(x, c)$ provided above is expensive to compute. As described in Ishwaran and Kogalur [30], $C(x, c)$ is equivalent to

$$C(x,c) = \frac{1}{Y_1}\sum_{j=1}^{2} Y_{1,j}\sum_{k=1}^{N-1}\left(N_{k,j}Y_{k+1,j}\sum_{l=1}^{k}\frac{d_{l,j}}{Y_{l,j}}\right),$$

which is easier to compute. This is therefore the expression we will use in an implementation.

5.5.3 Log-rank score splitting

The log-rank score test works as follows. We assume that the feature of interest x has been ordered for the n observations in the parent node, $x_1 \leq x_2 \leq \cdots \leq x_n$. Now compute the *ranks* for each survival time T_l,

$$a_l = \delta_l - \sum_{k=1}^{\Gamma_l}\frac{\delta_k}{n - \Gamma_k + 1}$$

where $\Gamma_k = \#\{m : T_m \leq T_k\}$ is the number of censored or uncensored observations less than or equal to T_k. The log-rank score test is then given by

$$L_S(x,c) = \frac{\sum_{l:x_l \leq c} a_l - n_1\overline{a}}{\sqrt{n_1\left(1 - \frac{n_1}{n}\right)s_a^2}}$$

where $\overline{a}$ is the sample mean of the ranks $a_1, a_2, ..., a_n$ and s_a^2 the sample variance,

$$\overline{a} = \frac{1}{n}\sum_{i=1}^{n} a_i, \quad \text{and} \quad s_a^2 = \frac{1}{n-1}\sum_{i=1}^{n}(a_i - \overline{a})^2.$$

The absolute value of the log-rank score test $|L_S(x,c)|$ indicates the measure of node separation. Hence the best split is found by maximising $|L_S(x,c)|$ over (x,c). Background on the log-rank score test can be found in the article Hothorn and Lausen [28], see section 5 in particular.

5.5.4 Approximate log-rank splitting

If one needs to reduce computations, an alternative to log-rank splitting is to use an approximation which we will call the approximate log-rank test. It is based on approximating the numerator and the denominator in the expression for the log-rank test. Considering the numerator, we claim that

$$\sum_{i=1}^{N}\left(d_{i,1} - Y_{i,1}\frac{d_i}{Y_i}\right) = D_1 - \sum_{i=1}^{n}\mathbb{1}_{\{x_l \le c\}}\widehat{H}(T_l)$$

where $D_j = \sum_{i=1}^{N} d_{i,j}$. To see why, we start by noting that it suffices to show that

$$\sum_{i=1}^{N} Y_{i,1}\frac{d_i}{Y_i} = \sum_{i=1}^{n}\mathbb{1}_{\{x_l \le c\}}\widehat{H}(T_l).$$

We can use reasoning similar to the proof of Lemma 5.1. Consider the right hand side,

$$\sum_{i=1}^{n}\mathbb{1}_{\{x_l \le c\}}\widehat{H}(T_l) = \sum_{i=1}^{n}\mathbb{1}_{\{x_l \le c\}}\sum_{m:t_m \le T_l}\frac{d_m}{Y_m}.$$

Regrouping the terms in the double sum, we see that d_1/Y_1 shows up all the times that the corresponding feature x is less than or equal to c. This is by definition $Y_{1,1}$ times. The same reasoning applies to all other terms of the form d_m/Y_m. Hence the sum equals

$$\sum_{i=1}^{N} Y_{i,1}\frac{d_i}{Y_i}$$

as desired. As for the sum in the denominator,

$$\sum_{i=1}^{N}\frac{Y_{i,1}}{Y_i}\left(1 - \frac{Y_{i,1}}{Y_i}\right)\left(\frac{Y_i - d_i}{Y_i - 1}\right)d_i,$$

we use the approximation outlined on page 105 in the book Cox and Oakes [17]. This yields the final approximation to the log-rank test:

$$\tilde{L}_{LR}(x,c) = \frac{\sqrt{D}\left(D_1 - \sum_{l=1}^{n}\mathbb{1}_{\{x_l \le c\}}\widehat{H}(T_l)\right)}{\sqrt{\left(\sum_{l=1}^{n}\mathbb{1}_{\{x_l \le c\}}\widehat{H}(T_l)\right)\left(D - \sum_{l=1}^{n}\mathbb{1}_{\{x_l \le c\}}\widehat{H}(T_l)\right)}}$$

where $D = D_1 + D_2 = \sum_{i=1}^{N} d_i$.

5.5.5 C-index splitting

When evaluating the predictive performance of a random survival forest, we use the C-index. It is therefore natural to suggest using the C-index as a splitting rule. This is explored in Schmid et al. [50], where they claim that this outperforms log-rank splitting in small datasets and if the censoring rate is high. To use the C-index as a splitting rule, we need some measure of risk (previously denoted M_i). As in the article, we choose the risk measure $\gamma_i := \mathbb{1}_{\{x_i > c\}}$, that is, the indicator that the data goes to the right node. The goal is then to maximise the concordance probability

$$\mathbb{P}(\gamma_i < \gamma_j \mid T_i > T_j) = \mathbb{P}(x_i \leq c, x_j > c \mid T_i > T_j).$$

The estimator is presented in Schmid et al. [50] in the case of no ties of the survival times. We present the general estimator

$$L_{HC}(x, c) = \frac{\sum_{i=1}^{n} \delta_i \sum_{j=i+1}^{n} (\mathbb{1}_{\{T_i < T_j\}} + (1 - \delta_j)\mathbb{1}_{\{T_i = T_j\}})(\gamma_i(1 - \gamma_i) + \frac{1}{2}\mathbb{1}_{\{\gamma_i = \gamma_j\}})}{\sum_{i=1}^{n} \delta_i \sum_{j=i+1}^{n} (\mathbb{1}_{\{T_i < T_j\}} + (1 - \delta_j)\mathbb{1}_{\{T_i = T_j\}})}.$$

Note that $\gamma_i(1 - \gamma_j) = \mathbb{1}_{\{\gamma_i > \gamma_j\}} = \mathbb{1}_{\{x_i > c, x_j \leq c\}}$ and that $\mathbb{1}_{\{\gamma_i = \gamma_j\}} = 1$ if and only if observations i and j land in the same node. Computing L_{HC} is simply a matter of adapting the algorithm presented in Subsection 4.5. The best split is then found by maximising $L_{HC}(x, c)$ over (x, c).

Both Harrell's C-index and the log-rank statistic are related to the *Gehan statistic* for testing the survival difference in two groups, see Gehan [21]. Adapted to the setting of two nodes in a survival tree, the statistic is (in the case of no ties between survival times) given by

$$U = \sum_{x_i > c, x_j \leq c} (\mathbb{1}_{\{T_i < T_j\}}\delta_i - \mathbb{1}_{\{T_j < T_i\}}\delta_j)$$

so that U assigns the value $+1$ to pairs with $T_i < T_j$ and -1 to pairs with $T_j < T_i$ as long as the shorter survival time is uncensored. Hence the Gehan statistic is a measure of whether the survival time in the right node is systematically larger than the survival time in the left node. Squaring and standardising the Gehan statistic yields the Gehan–Wilcoxon statistic given by

$$\frac{\left(\sum_{i=1}^{N} Y_i \left(d_{i,1} - Y_{i,1}\frac{d_i}{Y_i}\right)\right)^2}{\sum_{i=1}^{N} Y_i^2 Y_{i,1} Y_{i,2} \frac{d_i(Y_i - d_i)}{Y_i^2(Y_i - 1)}}$$

which is closely related to the log-rank statistic L_{LR} above. Indeed, the difference is that the summands are weighted by the number of individuals at risk, and that we have chosen to take the square root when computing L_{LR}. The above quantity is equal to U^2 divided by the variance of U conditional on the null hypothesis that the survival time is the same in both nodes and conditional on the pattern of observations. More details are provided in Gehan [21], see in particular the example just

before section 5. There is also a strong relationship between L_{HC} and the Gehan statistic. We may rewrite the Gehan statistic as

$$U = 2 \sum_{x_i > c, x_j \leq c} (\mathbb{1}_{\{T_i < T_j\}} + (1 - \delta_j)\mathbb{1}_{\{T_i = T_j\}})\delta_j - N$$

with N the number of possible comparisons between observations in the right and left node. On the other hand, the numerator of L_{HC} can be written as

$$\sum_{x_i > c, x_j \leq c} (\mathbb{1}_{\{T_i < T_j\}} + (1 - \delta_j)\mathbb{1}_{\{T_i = T_j\}})\delta_j + 0.5N_1 + 0.5N_2$$

where N_1 and N_2 are the number of permissible pairs in the left and right node, respectively. Hence the C-index is linearly related to the Gehan statistic.

5.6 Prediction error

For prediction error, we use the error $E = 1 - C$ where C is Harrell's C-index as presented earlier in the section on survival analysis. The setup is completely transferable. The only thing we need to do is define the risk measure M_i associated to an observation (T_i, δ_i). For this, we use the OOB ensemble estimate defined above. Let $t_1, ..., t_N$ denote all the unique event times in the data. We then define

$$M_i = \sum_{k=1}^{N} \widehat{H}_e^{**}(t_k \mid \mathbf{x}_i),$$

and we say that individual i has a *worse outcome* than individual j if

$$\sum_{k=1}^{N} \widehat{H}_e^{**}(t_k \mid \mathbf{x}_i) > \sum_{k=1}^{N} \widehat{H}_e^{**}(t_k \mid \mathbf{x}_j).$$

5.7 Data examples

5.7.1 Comparison of splitting rules

To illustrate the RSF method from the `JuliaExtendableTrees` package and comparing the different splitting rules above, we have analysed four datasets. All figures were made using the Algebra of Graphics Julia library. The datasets analysed are as follows.

- `pbc`: Primary Biliary Cirrhosis data from the Mayo Clinic trial. Data was collected between 1974 and 1984 and has a total of 418 observations. We remove all observations with missing data, which leaves 278 complete observations (33.49% incomplete observations).
- `veteran`: 137 observations without any missing data of a randomised trial of two treatments for lung cancer. The data is from Kalbfleisch and Prentice [38].

- `cancer`: Survival data from patients with advanced lung cancer from the North Central Cancer Treatment Group. A total of 228 observations with 167 complete observations (26.75% incomplete observations). We only use the complete data. The data is from the R package `survival`, Therneau et al. [52].
- `retinopathy`: Treatment data of diabetic retinopathy in the eye. Here an event is loss of vision in the eye. The data has 394 observations and no missing data. The data is from the R package `survival`.

We compare the performance of the different splitting rules applied to these datasets. For every dataset, we do the following: Sample 100 bootstrap datasets and fit a random forest for each splitting rule. Then compute the C-index. Each forest has 1000 trees and a minimum node size of 15. We use classic bootstrap. The result is plotted below.

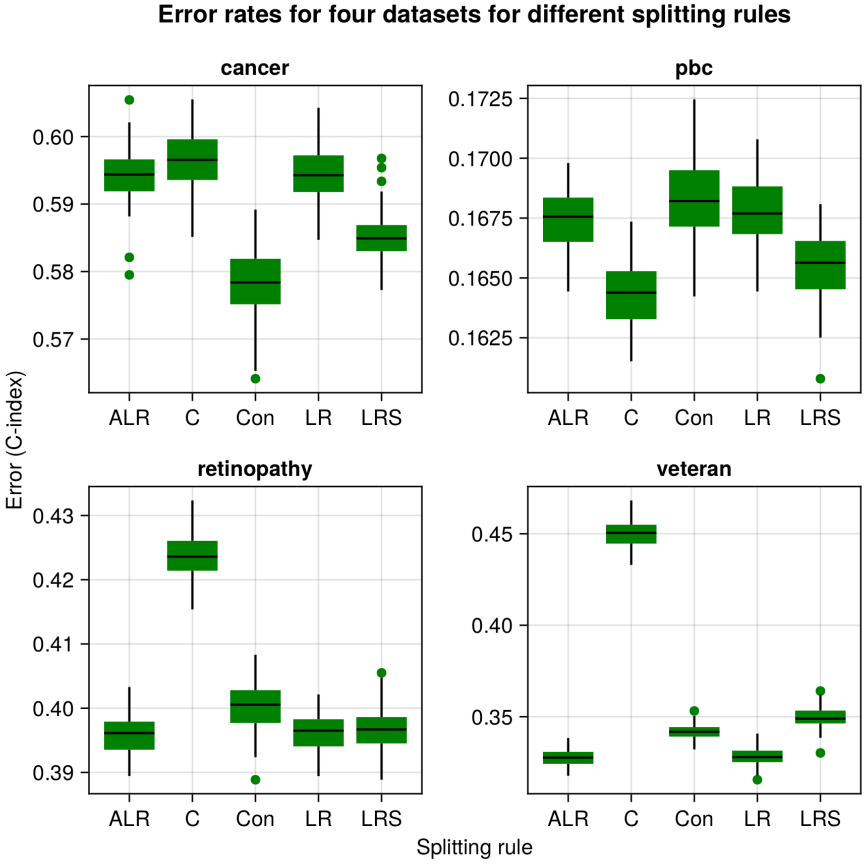

Figure 10: Error rates for the datasets `cancer`, `pbc`, `retinopathy` and `veteran` for each available splitting rule in `JuliaExtendableTrees`. The box-plots are computed based on 100 bootstrap samples of the original data.

From the figure, we note the following. The log-rank and approximate log-rank splitting rules yield almost identical results. In fact, approximate log-rank performs marginally better than log-rank. This is a nice observation since the approximate log-rank statistic is less computationally intensive. Interestingly, the best splitting rule varies quite a bit depending on the dataset.

It is worth noting the behaviour of the C-index splitting rule. For the datasets with no missing observations, the C-index splitting rule is noticably worse. It is noticably better for the `pbc` dataset however. According to Schmid et al. [50], the C-index outperforms log-rank splitting when the censoring rate is high. For `veteran`, the censoring rate is 6.57% while it is 59.78% for `pbc`. On the other hand, it is around 60% for `retinopathy` as well, so this only partially explains the observed behaviour.

As a final comment, we note that the error rates for the log-rank, conservation of events, log-rank score and approximate log-rank splitting rules concur with those produced by the `randomForestSRC` package, Ishwaran et al. [33].

5.7.2 A case study: The `peakVO2` dataset

We now do a study of the dataset `peakVO2` from the package `randomForestSRC`. The data is thoroughly discussed and analysed using RSF in the article Hsich et al. [29]. The data consists of 2231 observations from patients with systolic heart failure undergoing cardiopulmonary stress testing at the Cleveland Clinic. The dataset has 39 covariates involving both demographic variables, stress testing variables and medical information. It is worth noting that some of the observations in the dataset are imputed. As explained in Hsich et al. [29], laboratoy tests from before October 1999 were systematically missing, and as a consequence, 10% of the serum glucose, serum urea nitrogen, creatine and sodium values and 15% of hemoglobin values are imputed using informed imputation.

The goal is to tune the hyperparameters to obtain as small an (OOB) error as possible, and in the following, we only report the OOB error rate. We set no restriction on the maximum depth, and we use ordinary bootstrap to start with. We grow 1000 trees. We now establish the best splitting rule as well as the best minimal size of a node. We check the values 5, 10, 15, 20, 30, 40 and 50 for each of the five splitting rules and obtain the following plot.

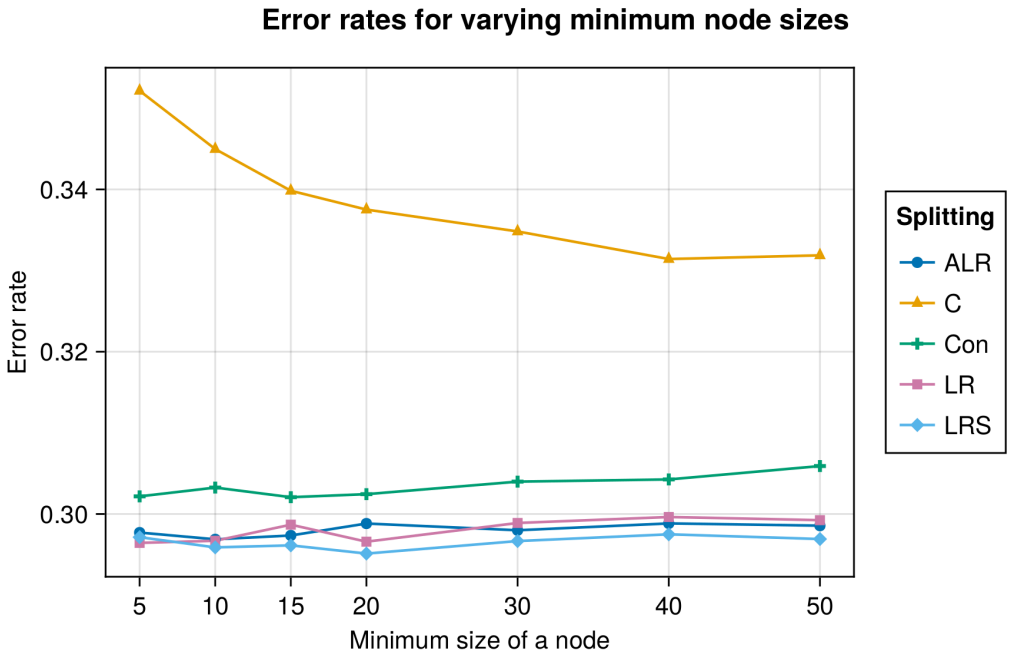

Figure 11: The error rate as a function of minimal node size for the `peakVO2` dataset. All five splitting rules were considered. ALR = approximate log-rank splitting, C = C-index splitting, Con = conservation of events splitting, LR = log-rank splitting, LRS = log-rank score splitting.

From the plot, it seems that a minimum node size of 20 with log-rank score splitting is ideal. Performance is quite similar for four of the splitting rules with only C-index splitting being a clear outlier. We choose to continue working with `min_node_size` = 20 and the log-rank score splitting rule. Next we investigate the sampling scheme. It is certainly interesting to see the effect of varying the fraction of data used, especially if choosing a low value does not increase the error too much. We vary the fraction of data used, `sfrac`, over 0.1, 0.2, ..., 0.9, 1.0 and sample with replacement for `sfrac` = 1.0 and without otherwise. This yields the following plot.

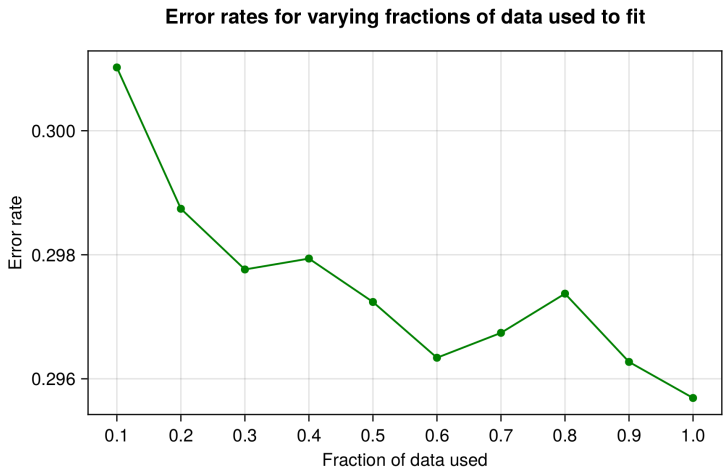

Figure 12: The error rate as a function of the fraction of data used to fit a tree in the forest for the `peakVO2` dataset. We have `swr = false` except for the case `sfrac = 1.0`.

The variation in the error is quite small. This is likely because the dataset contains many observations. From the figure, it seems that classic bootstrap yields the lowest error. It also seems that `sfrac` $= 0.6$ and `swr = false` is a good choice. In the final forest, we choose `sfrac` $= 1.0$ and `swr = true`, but for the next test, we work with `sfrac` $= 0.6$ and `swr = false` to save computation time. Another interesting hyperparameter is the number of split points used in a split, `n_split`. The default value in `JuliaExtendableTrees` is 10. We now investigate the effect for nine different values of `n_split`, namely 1, 2, 3, 4, 5, 10, 25, 50 and 100.

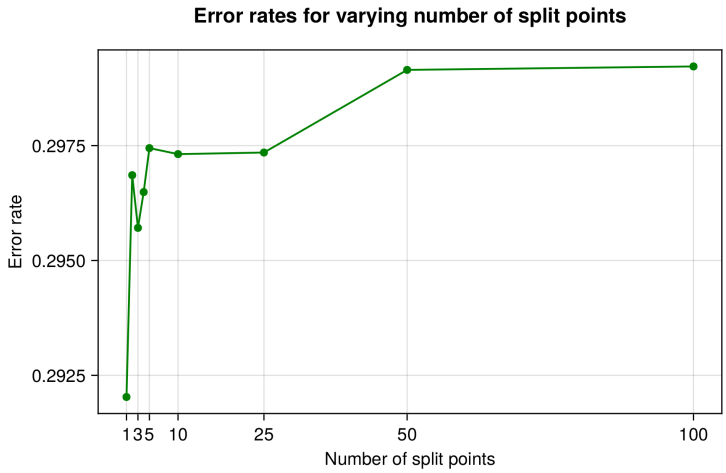

Figure 13: The error rate as a function of the number of split points used whenever a split occurs for the `peakVO2` dataset.

Surprisingly, only using a single split point yields the lowest error, followed by using three. One would expect a larger number of split points to give more precise estimates since a low number of split points prevents good splits. It is possible that a low number of split points decreases the correlation between trees to a degree which more than makes up for the loss in missing good splits. So far, we have used the default number of candidate features for a split, namely 6 (the integer closest to $\sqrt{39}$), but it is possible that using a different number of features is better. The following figure illustrates our findings.

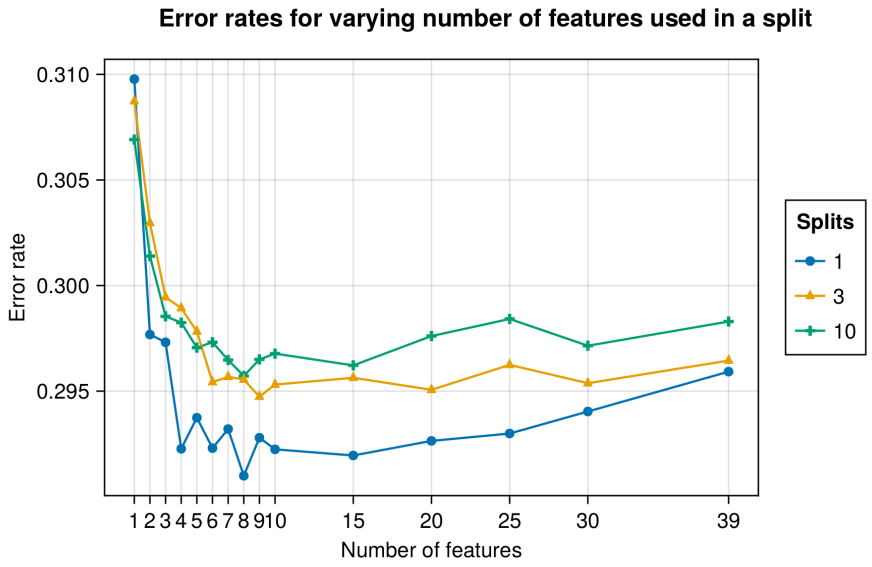

Figure 14: The error rate when varying the number of candidate features in a split for the `peakVO2` dataset. The case `n_features` $= 39$ corresponds to bagging. The error rate is plotted for three different values of `n_split`, namely 1, 3 and 10.

As shown in the figure, we choose to plot the error rate for three different values of `n_split` to confirm that it was not simply a coincidence that `n_split` $= 1$ was best. From the plot, it is evident that choosing `n_features` $= 8$ is a good idea (even if the best choice is 9 for `n_split` $= 3$). It also seems that `n_split` $= 1$ is still optimal. Lastly, we test the effect of varying the number of trees. Here we use ordinary bootstrapping.

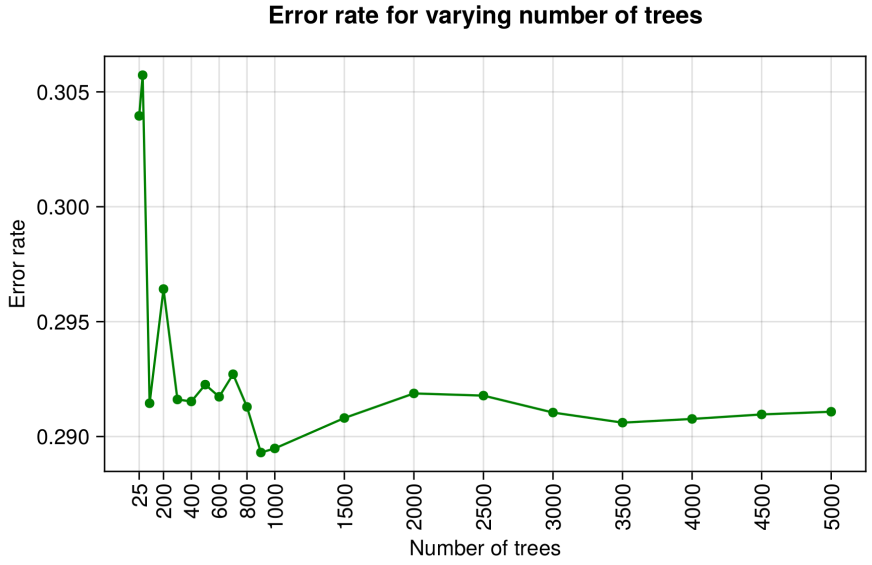

Figure 15: The error rate when varying the number of trees in the forest for the `peakVO2` dataset.

Interestingly, the lowest error occurs for 900 and 1000 trees. Since the absolute difference in error is so small, it is likely just due to randomness. Fixing the random seed (2024) and trying out a few values of `n_trees`, choosing `n_trees` = 2500 led to the error 0.2905 which seems to be the lowest value, we are able to get at this point. Let us summarise.

Hyperparameter	Value
`n_features`	8
`n_trees`	2500
`L`	`L_log_rank_score`
`n_split`	1
`min_node_size`	20
`sfrac`	1.0
`swr`	`true`

Figure 16: Table of the optimal parameters found for a random survival forest fitted on the dataset `peakVO2`.

In the article Hsich et al. [29], they report a C-index of 0.705, corresponding to an error of 0.2950. This was for a forest with 2000 trees, 3 split points, the log-rank splitting rule, 7 features and a minimal node size choosen such that a terminal node has no fewer than 3 observed deaths. We conclude that our choice of hyperparameters

seems to lead to a marginally better precision. The article further reports that a Cox proportional hazards model yields a C-index of 0.698, a precision lower but still very close to the one obtained for a random survival forest.

5.7.3 Comments on performance

A walkthrough of some of the code used for doing the above analyses can be found in appendix B. While running the code, it became clear that while the precision of the `JuliaExtendableTrees` library is comparable to state of the art implementations, the speed is not. The runtime scales quite poorly with the number of observations. As an illustration, running the 35 survival forests needed to make the plot of the error for varying minimum node sizes took about 108 minutes with an Intel Core i5-8250 CPU with 1.60 GHz (using 4 threads). For comparison, fitting the 2000 random forests used for the boxplots took less than 50 minutes. The slow runtime is not surprising considering the way the library is structured. The library is a lot more "functional" in nature compared to other implementations, which exhibit a more object-oriented structure, where essential quantities are computed as few times as possible. One can also choose to save results of previous computations during fitting which saves time both when fitting and computing the error (both the packages `randomForestSRC` and `ranger`, Wright and Ziegler [54], do the fitting and error computations at the same time by default). The price for such an optimisation is an increase in memory usage, but this price is often worth it. If one wants to extend the library to include more general multi-state models, such optimisations are necessary for the library to be useful.

5.8 Simulated example from life insurance

In this subsection, we simulate simple life insurance data from a survival model and analyse the data using different methods.

5.8.1 Simulating the data

We briefly describe how the data is simulated. We consider the covariates

- $X_1 \sim \text{Bern}(1/2)$,
- X_2 discrete uniform on the values $(0, 1/2, 4/5, 1)$ with probabilities $(5/12, 1/4, 1/6, 1/6)$,
- $X_3 \sim \text{Bern}(7/10)$,
- $X_4 \sim U(2.5 \cdot 10^5, 7.5 \cdot 10^5)$ (rounded to the nearest integer),
- $X_5 \sim \mathcal{N}(25, 8)$.

X_5 is the age of the policyholder when initiating the contract. A possible interpretation of the remaining variables is the following. X_1 is sex (1 = male, 0 = female), X_2 could be an indicator of type of residence (city, village etc.), X_3 could be an

indicator of marriage (1 = married, 0 = not married), and X_4 is the annual wage. We simulate from a Gompertz–Makeham model with the hazard

$$\alpha(t; X_1, X_2, X_3, X_4, X_5) = \beta(X_1) + \gamma(X_2)\exp(\rho(X_1, X_3, X_4)(t + X_5))$$

where

$$\begin{aligned}\beta(X_1) &= 0.0004 + X_1 \cdot 0.0001,\\ \gamma(X_2) &= 0.000072 + X_2 \cdot 0.000006,\\ \rho(X_1, X_3, X_4) &= 0.0785 + X_1 \cdot 0.005 - X_3 \cdot 0.008 + \frac{1000}{X_4}.\end{aligned}$$

This model fits into the framework described earlier. Indeed, if F is the Gompertz–Makeham distribution with β, γ and ρ specified by the above functions, we simulate from the model F_x, where the starting age is stochastic and given by X_5. This allows us to use Corollary 4.18 to simulate the data (see the appendix for the code and some exploratory plots of this data). We choose to simulate 10.000 observations. When applying the principal branch of the Lambert W function using the `emdBook` package, 210 NaN values where produced. We simply chose to ignore these observations and simply continue working with a dataset with 9.780 observations. We impose no censoring on the data.

5.8.2 Analysing the data

We analyse the data using three methods. A nonparametric approach using RSF, a semiparametric method in the form of the Cox proportional hazards model and a parametric method, namely Poisson regression. To estimate the baseline hazard in the Poisson regression model, we use piecewise constant hazards as described in the section on survival analysis with split points 0, 5, 20, 40, 60, 70, 80, 85, 90 and 100. All analyses are made in R. To fit the Cox model, we use the `survival` package, Therneau et al. [52]. For the Poisson regression model, we use the package `eha`, Broström [12]. For the RSF, we use `randomForestSRC`. The reason for using R is twofold. First and foremost, too few survival analysis tools are available in Julia, and the RSF implementation in `JuliaExtendableTrees` is not yet fast enough to handle datasets of this size. Playing around with the hyperparameters in the RSF, we found that a minimum node size of 5 and using the log-rank score splitting rule was best in terms of minimising the C-index error. The number of trees had little effect on the error, so we choose 500 trees. All other parameters are the default for the `rfsrc` function. We bootstrapped 100 datasets, fitted each model and computed the C-index error. This gave the following boxplot (made in R using the Tidyverse family of packages, [2]).

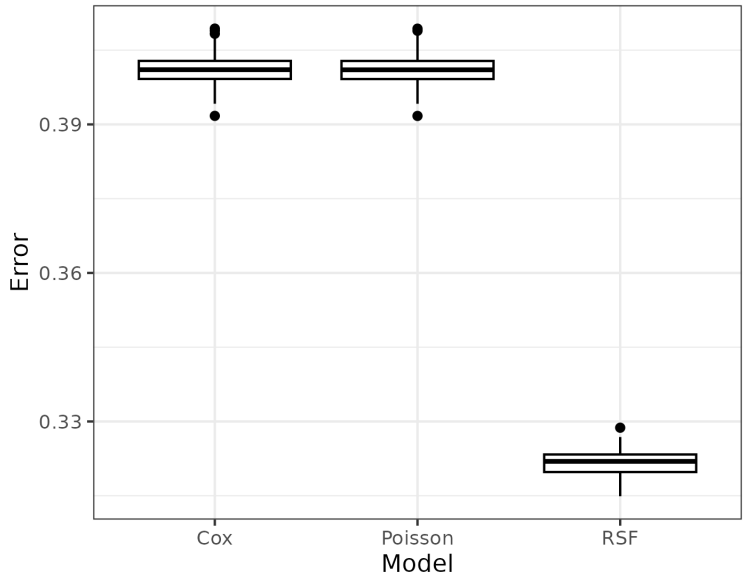

Figure 17: Boxplots of the C-index error when using three different survival models on the simulated insurance data.

From the figure, it is clear that the RSF model is best in terms of the C-index error. We also note that the variation in error for each model is very small, and that the two proportional hazard models yield very similar errors. When fitting these models, it became clear that the RSF grown on the full (non-bootstrapped) dataset had a much higher error (around 0.4126) than the error from the bootstrap datasets. This is quite surprising. A possible explanation is that resampling occurs twice. First a bootstrap dataset is created which introduces duplicates. Afterwards a subsample is extracted so that OOB observations may coincide with in-bag samples. This artificially reduces the error. A complete explanation requires extensive knowledge of the implementation of `rfsrc` in `randomForestSRC`.

Since we know the true hazard function, it makes sense to compare the fitted Nelson–Aalen estimator for the three models. In an insurance context, accurate prediction of hazards is key when estimating quantities like the reserve. In the following figure, the Nelson–Aalen estimators for eight different combinations of features plucked from the data are presented. From top left to bottom right, these combinations are

1. $X_1 = 1$, $X_2 = 1$, $X_3 = 1$, $X_4 = 307983$, $X_5 = 25.88$,
2. $X_1 = 1$, $X_2 = 1$, $X_3 = 1$, $X_4 = 601354$, $X_5 = 25.64$,
3. $X_1 = 1$, $X_2 = 1$, $X_3 = 0$, $X_4 = 301600$, $X_5 = 24.94$,
4. $X_1 = 1$, $X_2 = 1$, $X_3 = 0$, $X_4 = 604542$, $X_5 = 24.56$,
5. $X_1 = 0$, $X_2 = 1$, $X_3 = 1$, $X_4 = 299737$, $X_5 = 25.27$,

6. $X_1 = 0$, $X_2 = 1$, $X_3 = 1$, $X_4 = 623863$, $X_5 = 25.27$,
7. $X_1 = 0$, $X_2 = 1$, $X_3 = 0$, $X_4 = 302075$, $X_5 = 24.63$,
8. $X_1 = 0$, $X_2 = 1$, $X_3 = 0$, $X_4 = 636900$, $X_5 = 25.02$.

This yields the following plot.

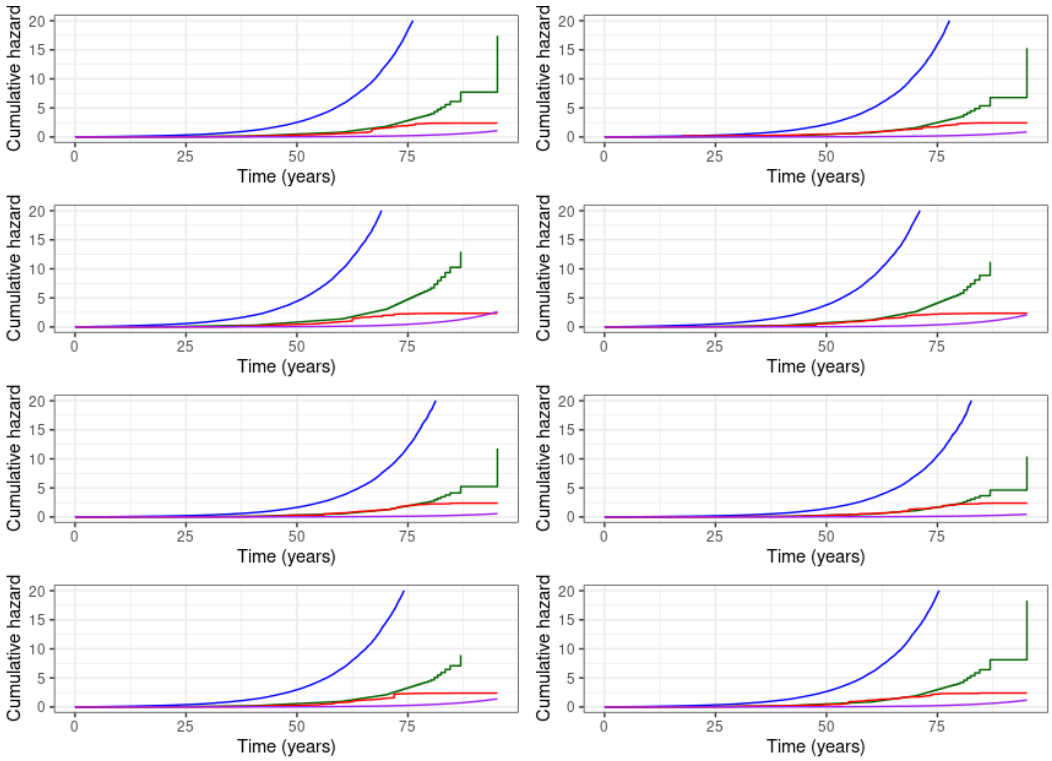

Figure 18: The Nelson–Aalen estimator for eight different combinations of features. Blue: The Cox model, green: The Poisson model, red: RSF, purple: the true hazard.

Several tendencies are clear. First and foremost, the Cox model is way off. The Poisson model starts as a good fit and then explodes in a similar manner as for the Cox model. It seems that the proportional hazard assumption is far from satisfied. Furthermore, the proportional hazard models have significant difficulty in incorporating the effect of varying starting ages. Compared to these two models, the RSF is a much better fit, although it seems to systematically overestimate the hazard.

5.8.3 Further comments

An important lesson from the above study is the importance of choosing the right evaluation metric. Harrell's C-index can be useful from a biostatistical perspective to evaluate if the model correctly chooses the individuals most likely to live longer, but it seems to be less suited for predictive models in life insurance. In life insurance, precise mortality estimates are extremely important since large deviations from the

true hazard will propagate into the model for valuation. This suggests that if RSF is to be used for prediction in an insurance context, a different evaluation criterion needs to be developed. It is however unclear at the moment how such a criterion should be defined.

5.9 Consistency

In this subsection, we prove that, under certain regularity assumptions, RSF is consistent in the sense of uniform convergence of the Kaplan–Meyer estimator on bounded intervals. Throughout this subsection, we assume that the survival function of T° is given by

$$S(t \mid \mathbf{X}) = \mathbb{P}(T^\circ > t \mid \mathbf{X}) = \sum_{\mathbf{x} \in \mathcal{X}} \mathbb{1}_{\{\mathbf{X}=\mathbf{x}\}} \exp\left(-\int_0^t \alpha(s \mid \mathbf{x}) ds\right)$$

so that $\alpha(\cdot \mid \mathbf{x})$ is the hazard rate for the subpopulation $\mathbf{X} = \mathbf{x}$. To set the stage, recall that for a survival tree $\mathcal{T}$ with a terminal node $h \in \mathcal{N}(\mathcal{T})$, the predicted value for the observations in h is the Nelson–Aalen estimator

$$\widehat{H}_h(t) = \int_0^t \frac{J_s^h}{Y_s^h} N^h(ds).$$

We can then form the Kaplan–Meyer estimator $\widehat{S}(t)$ for the observations in h by

$$\widehat{S}_h(t) = \prod_{s \in (0,t]} (1 - \widehat{H}_h(ds)) = \prod_{s \in (0,t]} \left(1 - \frac{\Delta N_s^h}{Y_s^h}\right).$$

Just like with the Nelson–Aalen estimator, to compute the predicted survival function $\widehat{S}(t \mid \mathbf{x})$ for a feature vector $\mathbf{x}$, simply drop $\mathbf{x}$ down the tree to obtain

$$\widehat{S}(t \mid \mathbf{x}) = \widehat{S}_h(t), \quad \text{if } \mathbf{x} \in h.$$

We now prove consistency of random survival forests. The proofs rely heavily on the choice of growing the trees to full size under the restriction that a terminal node must have at least $d_0 > 0$ events (true deaths). Before embarking on the proofs, we make the following assumptions:

Assumption 5.3. We assume the following.

(i) Right censoring is entirely random given covariates, $T^\circ \perp\!\!\!\perp R \mid \mathbf{X}$.

(ii) For every $A \neq \emptyset$, we have $\mathbb{P}(\mathbf{X} \in A) > 0$.

(iii) The feature space $\mathcal{X}$ is finite and discrete.

(iv) If $\tau_R := \sup\{t \geq 0 : F_R(t) < 1\}$ denotes the upper endpoint of the distribution of R, we have $\mathbb{P}(R \in (u, v)) > 0$ for every $0 \leq u < v < \tau_R$. We furthermore assume $\tau_R > 0$ so that R is not the one-point measure in zero.

The first result shows uniform consistency of the Kaplan–Meyer estimator on bounded intervals for a single tree.

Theorem 5.4. *Let $t \in (0, \tau(\mathbf{x}) \wedge \tau_R)$ where $\tau(\mathbf{x}) = \sup\{t : \int_0^t \alpha(s \mid \mathbf{x})ds < \infty\}$. If $\alpha(\cdot \mid \mathbf{x})$ is strictly positive over $[0,t]$ for every $\mathbf{x} \in \mathcal{X}$, then*

$$\sup_{s\in[0,t]} |\widehat{S}(s \mid \mathbf{x}) - S(s \mid \mathbf{x})| \xrightarrow{\mathbb{P}} 0 \quad as \quad n \to \infty.$$

If furthermore, $t \in (0, \tau \wedge \tau_R)$ with $\tau = \min\{\tau(\mathbf{x}) : \mathbf{x} \in \mathcal{X}\}$, we have the aggregate result

$$\sup_{s\in[0,t]} \int_{\mathcal{X}} |\widehat{S}(s \mid \mathbf{x}) - S(s \mid \mathbf{x})| \mathbb{P}(\mathbf{X} \in d\mathbf{x}) \xrightarrow{\mathbb{P}} 0 \quad as \quad n \to \infty.$$

Proof. We claim that $\mathbb{P}(\mathbf{X} \in A, \delta = 1) > 0$. Indeed, we have that

$$\mathbb{P}(\mathbf{X} \in A, \delta = 1) = \sum_{\mathbf{x}\in A} \mathbb{P}(\mathbf{X} = \mathbf{x}, \delta = 1) = \sum_{\mathbf{x}\in A} \mathbb{P}(\delta = 1 \mid \mathbf{X} = \mathbf{x})\mathbb{P}(\mathbf{X} = \mathbf{x}).$$

Using assumption (ii), $\mathbb{P}(\mathbf{X} = \mathbf{x}) > 0$ for all $\mathbf{x} \in \mathcal{X}$. Hence we have to show that $\mathbb{P}(\delta = 1 \mid \mathbf{X} = \mathbf{x}) > 0$. Using the tower property and the assumption of conditional entirely random right censoring, we have

$$\begin{aligned}\mathbb{P}(\delta = 1 \mid \mathbf{X} = \mathbf{x}) = \mathbb{P}(T^\circ \le R \mid \mathbf{X} = \mathbf{x}) &= \int_0^\infty \mathbb{P}(T^\circ \le r \mid \mathbf{X} = \mathbf{x}, R = r)\mathbb{P}(R \in dr) \\ &= \int_0^\infty \mathbb{P}(T^\circ \le r \mid \mathbf{X} = \mathbf{x})\mathbb{P}(R \in dr),\end{aligned}$$

and this quantity is strictly positive by the assumptions on α and R. It now follows from the law of large numbers that

$$\frac{1}{n}\sum_{i=1}^n \mathbb{1}_{\{\mathbf{X}_i\in A,\delta_i=1\}} \xrightarrow{\mathbb{P}\text{-a.s.}} \mathbb{P}(\mathbf{X} \in A, \delta = 1) > 0,$$

from which it follows that

$$\sum_{i=1}^n \mathbb{1}_{\{\mathbf{X}_i\in A,\delta_i=1\}} \xrightarrow{\mathbb{P}\text{-a.s.}} \infty$$

implying that for every $k \in \mathbb{N}$,

$$\mathbb{1}_{\{\sum_{i=1}^n \mathbb{1}_{\{\mathbf{X}_i\in A,\delta_i=1\}} \ge k\}} \xrightarrow{\mathbb{P}\text{-a.s.}} 1. \tag{9}$$

We now utilise that the tree is grown to full size with the restriction that each terminal node should have at least $d_0 > 0$ observed deaths. Assume that for some n, we have a terminal node with two distinct values of $\mathbf{x}$. Using (9), as n grows larger, we see that a.s. we will have sufficiently many observations in a node to obtain a split which ensures that both daughters have at least d_0 distinct observed events. Hence a split will happen with probability one as n growns. Since $\mathcal{X}$ is finite, this

shows that in the limit $n \to \infty$, each terminal node h contains a unique value of $\mathbf{x}$. We conclude that for large enough n, any terminal node h may be identified with exactly one $\mathbf{x}$-value. We will abuse notation and write $h = \mathbf{x}$ when h is identified with $\mathbf{x}$. Now fix $t \in (0, \tau(\mathbf{x}) \wedge \tau_R)$ and let $s \in [0, t]$. It now follows that

$$\widehat{S}(s \mid \mathbf{X}) = \sum_{\mathbf{x} \in \mathcal{X}} \mathbb{1}_{\{\mathbf{X}=\mathbf{x}=h\}} \widehat{S}_h(s) + o_{\mathbb{P}}(1) \tag{10}$$

uniformly in s since $\mathbb{1}_{\{\mathbf{X}=\mathbf{x}=h\}}$ and $\mathbb{1}_{\{\mathbf{X}\in h\}}$ are eventually equal in the limit $n \to \infty$ for all $\mathbf{x}$ and some $h \in \mathcal{N}(\mathcal{T})$, and since $\widehat{S}_h(s)$ is bounded for all h. We now wish to apply Corollary 4.6 to show that

$$\sup_{s \in [0,t]} |\widehat{S}_h(s) - S(s \mid \mathbf{x})| \xrightarrow{\mathbb{P}} 0 \tag{11}$$

as $n \to \infty$ for each $h = \mathbf{x}$. Recall that $Y^{(n)}(s \mid \mathbf{x}) = \sum_{i=1}^n \mathbb{1}_{\{T_i \geq s, \mathbf{X}_i = \mathbf{x}\}}$ denotes the number of individuals at risk with feature $\mathbf{x}$ and that the setup of RSF fits into the multiplicative intensity framework with intensity $\alpha(s \mid \mathbf{x}) Y^{(n)}(s \mid \mathbf{x})$. To apply the corollary, we need to verify the two assumptions

$$\int_0^t \frac{J^{(n)}(s \mid \mathbf{x})}{Y^{(n)}(s \mid \mathbf{x})} \alpha(s \mid \mathbf{x}) ds \xrightarrow{\mathbb{P}} 0 \tag{i}$$

and

$$\int_0^t (1 - J^{(n)}(s \mid \mathbf{x})) \alpha(s \mid \mathbf{x}) ds \xrightarrow{\mathbb{P}} 0 \tag{ii}$$

as $n \to \infty$ where $J^{(n)}(s \mid \mathbf{x}) = \mathbb{1}_{\{Y^{(n)}(s|\mathbf{x})>0\}}$. By the definition of τ, we have $K := \sup_{s \in [0,t]} \alpha(s \mid \mathbf{x}) < \infty$. We claim that $\inf_{s \in [0,t]} Y^{(n)}(s \mid \mathbf{x}) \xrightarrow{\mathbb{P}\text{-a.s.}} \infty$. Indeed, for all $s \in [0, t]$, we have

$$\frac{1}{n} Y^{(n)}(s \mid \mathbf{x}) \geq \frac{1}{n} \sum_{i=1}^n \mathbb{1}_{\{T_i \geq t, \delta_i = 1, \mathbf{X}_i = \mathbf{x}\}} \xrightarrow{\mathbb{P}\text{-a.s.}} \mathbb{P}(T \geq s, \delta = 1, \mathbf{X} = \mathbf{x}).$$

We show that $\mathbb{P}(T \geq s, \delta = 1, \mathbf{X} = \mathbf{x}) > 0$. Indeed,

$$\mathbb{P}(T \geq s, \delta = 1, \mathbf{X} = \mathbf{x}) = \mathbb{P}(T^\circ \geq s, \delta = 1 \mid \mathbf{X} = \mathbf{x}) \mathbb{P}(\mathbf{X} = \mathbf{x}),$$

and $\mathbb{P}(\mathbf{X} = \mathbf{x}) > 0$ so it suffices to show that $\mathbb{P}(T^\circ \geq s, \delta = 1 \mid \mathbf{X} = \mathbf{x}) > 0$. Here the tower property is again applied to get

$$\begin{aligned} \mathbb{P}(T^\circ \geq s, \delta = 1 \mid \mathbf{X} = \mathbf{x}) &= \int_0^\infty \mathbb{P}(s \leq T^\circ \leq r \mid \mathbf{X} = \mathbf{x}, R = r) \mathbb{P}(R \in dr) \\ &= \int_s^\infty \mathbb{P}(s \leq T^\circ \leq r \mid \mathbf{X} = \mathbf{x}) \mathbb{P}(R \in dr) \end{aligned}$$

which is strictly positive by a similar argument as before (note that $s \in [0, t]$). It follows that $\inf_{s \in [0,t]} Y^{(n)}(s \mid \mathbf{x}) \xrightarrow{\mathbb{P}\text{-a.s.}} \infty$. It is now easy to verify (i) and (ii) above. For (i),

$$\int_0^t \frac{J^{(n)}(s \mid \mathbf{x})}{Y^{(n)}(s \mid \mathbf{x})} \alpha(s \mid \mathbf{x}) ds \leq K \int_0^t \frac{1}{Y^{(n)}(s \mid \mathbf{x})} ds \leq \frac{Kt}{\inf_{s \in [0,t]} Y^{(n)}(s \mid \mathbf{x})} \xrightarrow{\mathbb{P}} 0$$

while for (ii),

$$\int_0^t (1 - J^{(n)}(s \mid \mathbf{x}))\alpha(s \mid \mathbf{x})ds \leq K \int_0^t 1 - J^{(n)}(s \mid \mathbf{x})ds \leq K \int_0^t 1 - \mathbb{1}_{\{\inf_{s\in[0,t]} Y^{(n)}(s|\mathbf{x})>0\}} ds$$
$$= Kt(1 - \mathbb{1}_{\{\inf_{s\in[0,t]} Y^{(n)}(s|\mathbf{x})>0\}}) \xrightarrow{\mathbb{P}} 0.$$

Hence the corollary applies and the first part of the theorem is proved. We furthermore conclude that by (10), whenever $t \in (0, \tau \wedge \tau_R)$,

$$\sup_{s\in[0,t]} \int_{\mathcal{X}} |\widehat{S}(s \mid \mathbf{x}) - S(s \mid \mathbf{x})| \mathbb{P}(\mathbf{X} \in d\mathbf{x}) \leq \int_{\mathcal{X}} \sup_{s\in[0,t]} |\widehat{S}(s \mid \mathbf{x}) - S(s \mid \mathbf{x})| \mathbb{P}(\mathbf{X} \in d\mathbf{x})$$
$$= \sum_{h=\mathbf{x}} \mathbb{P}(\mathbf{X} = \mathbf{x}) \left(\sup_{s\in[0,t]} |\widehat{S}_h(s) - S(s \mid \mathbf{x})| \right)$$
$$+ o_{\mathbb{P}}(1).$$

Since the sum is finite and each term converges to zero in probability by (11), the proof is complete. ■

Remark 5.5. In the article Ishwaran and Kogalur [31], the above theorem is proven by assuming

$$\mathbf{X} \perp\!\!\!\perp \delta \quad \text{and} \quad T^\circ \perp\!\!\!\perp R$$

instead of $T^\circ \perp\!\!\!\perp R \mid \mathbf{X}$. We have chosen to work with the assumption $T^\circ \perp\!\!\!\perp R \mid \mathbf{X}$ instead for two reasons. One reason is generalisability. This assumption carries over directly to the multistate setup. Another reason is the causal interpretation. It makes sense to consider $\mathbf{X}$ as a confounder for both the true survival time and the censoring mechanism. The interpretation of $T^\circ \perp\!\!\!\perp R \mid \mathbf{X}$ is then that all the dependence between T° and R is encapsulated in $\mathbf{X}$ which is a far more realistic assumption than assuming marginal independence between T° and R and between $\mathbf{X}$ and δ. ○

The theorem just proven shows that under our assumptions of a finite feature space and the form of the true survival function, a single survival tree is consistent. In reality, we fit several trees on bootstrap samples of the data and aggregate the result. We briefly recall the notation for this setup. We have a learning sample $\mathcal{L} = \{(\mathbf{X}_i, T_i, \delta_i) : i = 1, ..., n\}$. We sample B bootstrap samples with replacement from this set. Denote a bootstrap sample by $\mathcal{L}^* = \{(\mathbf{X}_i^*, T_i^*, \delta_i^*) : i = 1, ..., n\}$ and let $\mathcal{T}^*$ denote the survival tree grown on $\mathcal{L}^*$ with $\widehat{S}^*(t \mid \mathbf{x})$ the Kaplan–Meyer estimator of the bootstrap tree,

$$\widehat{S}^*(t \mid \mathbf{x}) = \sum_{h\in\mathcal{N}(\mathcal{T})} \mathbb{1}_{\{\mathbf{x}\in h\}} \widehat{S}_h^*(t).$$

The ensemble survival function for the forest of B trees is

$$\widehat{S}_e^*(t \mid \mathbf{x}) = \frac{1}{B} \sum_{b=1}^{B} \widehat{S}_b^*(t \mid \mathbf{x})$$

with $\widehat{S}_b^*(t \mid \mathbf{x})$ the Kaplan–Meyer estimator of the b'th bootstrap tree. We start by proving consistency of the b'th bootstrap survival tree. Consistency of the ensemble estimator is an immediate consequence provided in the corollary after the following theorem. Let F denote the distribution function of $T = T^\circ \wedge R$ and let $o_{\mathbb{P}}^*(1)$ mean $o(1)$ in bootstrap probability for almost all $\mathcal{L}$-sample sequences.

Theorem 5.6. *Let $\tau^*(\mathbf{x}) = \tau(\mathbf{x}) \wedge \sup(F)$ with $\tau(\mathbf{x})$ defined as in the previous theorem and $\sup(F) = \sup\{s \geq 0 : F(s) < 1\}$ the right endpoint of the support of F. For each $t \in (0, \tau^*(\mathbf{x}))$, we have*

$$\sup_{s \in [0,t]} |\widehat{S}^*(s \mid \mathbf{x}) - S(s \mid \mathbf{x})| = o_{\mathbb{P}}^*(1) + o_{\mathbb{P}}(1).$$

Furthermore, for $t \in (0, \tau^*)$ with $\tau^* = \tau \wedge \sup(F)$,

$$\sup_{s \in [0,t]} \int_{\mathcal{X}} |\widehat{S}^*(s \mid \mathbf{x}) - S(s \mid \mathbf{x})| \mathbb{P}(\mathbf{X} \in d\mathbf{x}) = o_{\mathbb{P}}^*(1) + o_{\mathbb{P}}(1).$$

Proof. Let $\mathbf{M}^* = (M_{n,1}^*, ..., M_{n,n}^*)$ be a multinomial random vector from n trials with each cell having probability $1/n$ of occuring. By simply regrouping the bootstrap samples, we have for every nonempty $A \subseteq \mathcal{X}$ that

$$\begin{aligned}
\frac{1}{n} \sum_{i=1}^n \mathbb{1}_{\{\mathbf{X}_i^* \in A, \delta_i^* = 1\}} &\overset{\mathrm{d}^*}{=} \frac{1}{n} \sum_{i=1}^n \mathbb{1}_{\{\mathbf{X}_i \in A, \delta_i = 1\}} M_{n,i}^* \\
&= \frac{1}{n} \sum_{i=1}^n \mathbb{1}_{\{\mathbf{X}_i \in A, \delta_i = 1\}} + \frac{1}{n} \sum_{i=1}^n \mathbb{1}_{\{\mathbf{X}_i \in A, \delta_i = 1\}} (M_{n,i}^* - 1)
\end{aligned}$$

where $\overset{\mathrm{d}^*}{=}$ means equality in bootstrap distribution. The first term converges to $\mathbb{P}(\mathbf{X} \in A, \delta = 1)$ in probability. We claim that the second term converges to zero in bootstrap probability. Let $\varepsilon > 0$. Using the Markov inequality, one obtains

$$\begin{aligned}
&\mathbb{P}^* \left(\left| \frac{1}{n} \sum_{i=1}^n \mathbb{1}_{\{\mathbf{X}_i \in A, \delta_i = 1\}} (M_{n,i}^* - 1) \right| > \varepsilon \right) \leq \frac{1}{\varepsilon^2 n^2} \mathbb{E}^* \left[\left(\sum_{i=1}^n \mathbb{1}_{\{\mathbf{X}_i \in A, \delta_i = 1\}} (M_{n,i}^* - 1) \right)^2 \right] \\
&= \frac{1}{\varepsilon^2 n^2} \mathbb{E}^* \left[\sum_{i=1}^n \mathbb{1}_{\{\mathbf{X}_i \in A, \delta_i = 1\}} (M_{n,i}^* - 1)^2 + \sum_{i \neq j} (M_{n,i}^* - 1)(M_{n,j}^* - 1) \mathbb{1}_{\{\mathbf{X}_i \in A, \delta_i = 1\}} \mathbb{1}_{\{\mathbf{X}_j \in A, \delta_j = 1\}} \right] \\
&\leq \frac{1}{\varepsilon^2 n^2} \sum_{i=1}^n \mathrm{Var}^*[M_{n,i}^*] + \frac{1}{\varepsilon^2 n^2} \sum_{i \neq j} |\mathrm{Cov}^*(M_{n,i}^*, M_{n,j}^*)|.
\end{aligned}$$

In the final inequality, we simply used that all the indicators are at most one and that $\mathbb{E}^*[M_{n,i}^*] = n\frac{1}{n} = 1$. Here we recall that for a multinomial random vector $\mathbf{M}$ with n trials and probability vector $\mathbf{p}$, $\mathbb{E}[\mathbf{M}] = n\mathbf{p}$ and $\mathrm{Var}[\mathbf{M}] = n(\mathrm{diag}(\mathbf{p}) - \mathbf{p}\mathbf{p}^T)$. In the case for $\mathbf{M}^*$, $\mathbf{p} = (1/n, ..., 1/n)$, which shows that the above expression is

$O(1/n)$ in bootstrap probability. Using an argument analogous to the one provided in the proof of the previous theorem, we have for $t \in (0, \tau^*(\mathbf{x}))$ and $s \in [0, t]$ that

$$\widehat{S}^*(s \mid \mathbf{X}) = \sum_{\mathbf{x} \in \mathcal{X}} \mathbb{1}_{\{\mathbf{X}=\mathbf{x}=h\}} \widehat{S}_h^*(s) + o_{\mathbb{P}}^*(1).$$

Again the boundedness of $\widehat{S}_h^*$ ensures that $o_{\mathbb{P}}^*(1)$ is uniform in s. Now apply the triangle inequality to obtain

$$|\widehat{S}^*(s \mid \mathbf{x}) - S(s \mid \mathbf{x})| \leq |\widehat{S}^*(s \mid \mathbf{x}) - \widehat{S}(s \mid \mathbf{x})| + |\widehat{S}(s \mid \mathbf{x}) - S(s \mid \mathbf{x})|.$$

The second term is $o_{\mathbb{P}}(1)$ uniformly in s by Theorem 5.4. Concerning the first term, recall from the proof of Theorem 5.4 that

$$\widehat{S}(s \mid \mathbf{X}) = \sum_{\mathbf{x} \in \mathcal{X}} \mathbb{1}_{\{\mathbf{X}=\mathbf{x}=h\}} \widehat{S}_h(s) + o_{\mathbb{P}}(1)$$

so

$$|\widehat{S}^*(s \mid \mathbf{x}) - \widehat{S}(s \mid \mathbf{x})| = |\widehat{S}_h^*(s) - \widehat{S}_h(s)| + o_{\mathbb{P}}^*(1) + o_{\mathbb{P}}(1)$$

uniformly in s for $h = \mathbf{x}$. Note that an equivalent procedure to obtaining a bootstrap sample from $\mathcal{L}$ is the following:

1. Draw a multinomial vector $(n_h^*)_h$ from n trials with $\#\mathcal{X}$ cells where each cell $h = \mathbf{x}$ has probability $\widehat{p}_h := \frac{1}{n} \sum_{i=1}^n \mathbb{1}_{\{\mathbf{X}_i=\mathbf{x}=h\}}$.
2. For each h, draw n_h^* samples from $\mathcal{L}_h := \{(\mathbf{X}_i, T_i, \delta_i) : \mathbf{X}_i = \mathbf{x} = h\}$.

We claim that $n_h^*/n_h \xrightarrow{\mathbb{P}^*} 1$ where $n_h = \#\mathcal{L}_h$. Note that $\widehat{p}_h = \frac{1}{n} n_h$. We have

$$\mathbb{E}^* \left[\frac{n_h^*}{n_h} \right] = \frac{1}{n_h} \mathbb{E}^*[n_h^*] = \frac{1}{\widehat{p}_h} \mathbb{E}^* \left[\frac{1}{n_h} n_h^* \right] = \frac{\widehat{p}_h}{\widehat{p}_h} = 1$$

and

$$\mathrm{Var}^* \left[\frac{n_h^*}{n_h} \right] = \frac{1}{n_h^2} n_h \widehat{p}_h (1 - \widehat{p}_h) = \frac{1}{n_h} \frac{1 - \widehat{p}_h}{\widehat{p}_h} \to 0 \quad \text{as} \quad n \to \infty,$$

so $n_h^*/n_h \xrightarrow{\mathbb{P}^*} 1$ follows from an application of Chebyshev's inequality. We still need to show uniform convergence of $\widehat{S}_h^*(s) - \widehat{S}_h(s)$ for each $h = \mathbf{x}$. Recall that $\widehat{S}_h(s)$ is constructed from a sample of size n_h from $\mathcal{L}_h$, and $\widehat{S}_h^*(s)$ is constructed from a bootstrap sample of size n_h^* from $\mathcal{L}_h$. Since $n_h^*/n_h \xrightarrow{\mathbb{P}^*} 1$, we may assume that both $\widehat{S}_h^*(s)$ and $\widehat{S}_h(s)$ are constructed from samples of size n_h. We may now apply Lemma 3 in the article Lo and Singh [45] to see that

$$\sup_{s \in [0,t]} |\widehat{S}_h^*(s) - \widehat{S}_h(s)| = O_{\mathbb{P}}^*(n_h^{-1/2} (\log n_h)^{1/2}) = o_{\mathbb{P}}^*(1).$$

The lemma is applicable because of entirely random right censoring and the posed assumptions on t. It follows that

$$\sup_{s \in [0,t]} |\widehat{S}^*(s \mid \mathbf{x}) - S(s \mid \mathbf{x})| = o_{\mathbb{P}}(1) + o_{\mathbb{P}}^*(1).$$

This proves the first assertion of the theorem. If $t \in (0, \tau^*)$, it immediately follows from the finiteness of $\mathcal{X}$ that

$$\sup_{s \in [0,t]} \int_{\mathcal{X}} |\widehat{S}^*(s \mid \mathbf{x}) - \widehat{S}(s \mid \mathbf{x})| \mathbb{P}(\mathbf{X} \in d\mathbf{x}) = o^*_{\mathbb{P}}(1) + o_{\mathbb{P}}(1)$$

and the theorem is proved. ■

Corollary 5.7. *For the ensemble predictor $\widehat{S}^*_e(t \mid \mathbf{x})$, we have the consistency result*

$$\sup_{s \in [0,t]} |\widehat{S}^*_e(s \mid \mathbf{x}) - S(s \mid \mathbf{x})| = o^*_{\mathbb{P}}(1) + o_{\mathbb{P}}(1).$$

for $t \in (0, \tau^*(\mathbf{x})$ and

$$\sup_{s \in [0,t]} \int_{\mathcal{X}} |\widehat{S}^*_e(s \mid \mathbf{x}) - S(s \mid \mathbf{x})| \mathbb{P}(\mathbf{X} \in d\mathbf{x}) = o^*_{\mathbb{P}}(1) + o_{\mathbb{P}}(1).$$

whenever $t \in (0, \tau^*)$, where we use the same notation as in the above theorem.

Proof. From Theorem 5.6, we have for $t \in (0, \tau^*(\mathbf{x}))$ that

$$\sup_{s \in [0,t]} |\widehat{S}^*_e(s \mid \mathbf{x}) - S(s \mid \mathbf{x})| \leq \frac{1}{B} \sum_{b=1}^{B} \sup_{s \in [0,t]} |\widehat{S}^*_b(s \mid \mathbf{x}) - S(s \mid \mathbf{x})| = o^*_{\mathbb{P}}(1) + o_{\mathbb{P}}(1).$$

The aggregate result is just as easy to prove. ■

6 Multi-state models

"All models are wrong, but some are useful."
- George Box

In this section, we provide all necessary background on multi-state modelling needed for extending the random survival forest. We first touch upon general multi-state models and later consider the Markov model with intensities in more depth. We wrap up the section with a discussion about the pros and cons of working under the Markov assumption.

6.1 Setup and estimation

We start by providing the multi-state model setup as presented in Bladt and Furrer [6]. Consider a non-explosive pure jump process $Z = (Z_t)_{t\geq 0}$ on a finite state space $\mathcal{Z} \subseteq \mathbb{R}$ and let τ_Z denote the (possibly infinite) absorption time of Z. The covariates $\mathbf{X}$ are assumed to be an $\mathbb{R}^d$-valued random variable. We define the multivariate counting process $N = (N_{jk}(t))_{t\geq 0}$ by

$$N_{jk}(t) := \#\{s \in (0,t] : Z_{s-} = j, Z_s = k\}, \quad t \geq 0, \quad j,k \in \mathcal{Z}, j \neq k,$$

that is, the total number of jumps from state j to k in the time interval $(0,t]$. The following quantities are of utmost importance in estimation.

Definition 6.1. Let the setup be as above.

(i) We define *conditional occupation probabilities* p_j by $p_j(t|\mathbf{x}) = \mathbb{E}[\mathbb{1}_{\{Z_t=j\}} \mid \mathbf{X} = \mathbf{x}]$ and let $p(t \mid \mathbf{x}) = (p_j(t \mid \mathbf{x}))_{j\in\mathcal{Z}}$ denote the corresponding row vector.

(ii) Let $p_{jk}(t|\mathbf{x}) = \mathbb{E}[N_{jk}(t) \mid \mathbf{X} = \mathbf{x}]$.

(iii) We define the *cumulative transition rates* Λ_{jk} according to

$$\Lambda_{jk}(t|\mathbf{x}) = \int_0^t \frac{1}{p_j(s-|\mathbf{x})} p_{jk}(ds|\mathbf{x}) \quad \text{for} \quad j \neq k \quad \text{and} \quad \Lambda_{jj}(t|\mathbf{x}) = -\sum_{k:k\neq j} \Lambda_{jk}(t|\mathbf{x}).$$

The main quantity of interest in a general multi-state model is the vector p of conditional occupation probabilities. To this end, define

$$q(t|\mathbf{x}) = \prod_{(0,t]} (I + \Lambda(ds|\mathbf{x})).$$

Whenever we write $q(t \mid \mathbf{x})$, it is implicitly assumed that $\Lambda_{jk}(t \mid \mathbf{x}) < \infty$ so that $q(t \mid \mathbf{x})$ is well-defined. The relation between q and p is given in the following proposition.

Proposition 6.2. *It holds that*

$$p(t|\mathbf{x}) = p(0|\mathbf{x})q(t|\mathbf{x}) = p(0|\mathbf{x}) \prod_{(0,t]} (I + \Lambda(ds|\mathbf{x})).$$

The proposition is more or less a direct consequence of the following lemma. Before stating it, we recall the useful *inflow/outflow formula*

$$\mathbb{1}_{\{Z_t=j\}} = \mathbb{1}_{\{Z_0=j\}} + \sum_{\substack{k\in\mathcal{Z}\\k\neq j}} (N_{kj}(t) - N_{jk}(t)). \tag{12}$$

Lemma 6.3. *It holds that*

$$p(t|\mathbf{x}) = p(0|\mathbf{x}) + \int_0^t p(s-|\mathbf{x})\Lambda(ds|\mathbf{x}) \tag{i}$$

and

$$p(0|\mathbf{x})q(t|\mathbf{x}) = p(0|\mathbf{x})q(0|\mathbf{x}) + \int_0^t p(0|\mathbf{x})q(s-|\mathbf{x})\Lambda(ds|\mathbf{x}). \tag{ii}$$

Proof. We first consider (i). The j'th component of $p(s-|\mathbf{x})\Lambda(ds|\mathbf{x})$ is given by

$$\begin{aligned}
(p(s-|\mathbf{x})\Lambda(ds|\mathbf{x}))_j &= \sum_{k\in\mathcal{Z}} p_k(s-|\mathbf{x})\Lambda_{kj}(ds|\mathbf{x}) \\
&= \sum_{\substack{k\in\mathcal{Z}\\k\neq j}} p_k(s-|\mathbf{x})\Lambda_{kj}(ds|\mathbf{x}) + p_j(s-|\mathbf{x})\Lambda_{jj}(ds|\mathbf{x}) \\
&= \sum_{\substack{k\in\mathcal{Z}\\k\neq j}} (p_{kj}(ds|\mathbf{x}) - p_j(s-|\mathbf{x})\Lambda_{jk}(ds|\mathbf{x})) \\
&= \sum_{\substack{k\in\mathcal{Z}\\k\neq j}} (p_{kj}(ds|\mathbf{x}) - p_{jk}(ds|\mathbf{x})).
\end{aligned}$$

Integrating and applying (12) backwards, we obtain

$$\begin{aligned}
\int_0^t (p(s-|\mathbf{x})\Lambda(ds|\mathbf{x}))_j &= \sum_{\substack{k\in\mathcal{Z}\\k\neq j}} (p_{kj}(t|\mathbf{x}) - p_{jk}(t|\mathbf{x})) = \mathbb{E}\Big[\sum_{\substack{k\in\mathcal{Z}\\k\neq j}} (N_{kj}(t) - N_{jk}(t)) \mid \mathbf{X} = \mathbf{x}\Big] \\
&= \mathbb{E}[\mathbb{1}_{\{Z_t=j\}} - \mathbb{1}_{\{Z_0=j\}} \mid \mathbf{X} = \mathbf{x}] = p_j(t|\mathbf{x}) - p_j(0|\mathbf{x}).
\end{aligned}$$

Rearranging and rewriting to vector form, we obtain (i). As for (ii), we know from Theorem 3.8 that $q(t|\mathbf{x})$ satisfies the equation

$$q(t|\mathbf{x}) = I + \int_0^t q(s-|\mathbf{x})\Lambda(ds|\mathbf{x})$$

and that $q(0|\mathbf{x}) = I$. Now multiply by $p(0|\mathbf{x})$, and we obtain (ii). ■

Proof of Proposition 6.2. The result follows immediately by applying Theorem 3.9 with $Z(t) = p(t|\mathbf{x})$, $W(t) = p(0|\mathbf{x})$ and $X(t) = \Lambda(t|\mathbf{x})$ (note that each coordinate of $\Lambda(\cdot|\mathbf{x})$ is increasing and hence of finite variation) to equation (i) in the above lemma. ■

Proposition 6.2 gives an obvious approach to estimation. If we can find an estimator for the cumulative transition rates $\Lambda(\cdot|\mathbf{x})$ and the initial distribution $p(0|\mathbf{x})$ (in some cases, this is given without the need for estimation), we can simply plug this estimator into the product integral to obtain an estimator for p. We discuss estimators of $\Lambda(\cdot|\mathbf{x})$ in the presence of right-censoring. We assume that we are given data of the form

$$(\mathbf{X}, (Z_t)_{0\leq t\leq R}, \tau_Z \wedge R)$$

where R is a censoring variable. In order to do estimation in a general multi-state model, the following assumption is crucial.

Assumption 6.4. We assume *entirely random right-censoring*, that is, $Z \perp\!\!\!\perp R \mid \mathbf{X}$.

Definition 6.5. We define

(i) $p^c_j(t|\mathbf{x}) = \mathbb{E}[\mathbb{1}_{\{Z_t=j\}}\mathbb{1}_{\{t<R\}} \mid \mathbf{X} = \mathbf{x}]$ and

(ii) $p^c_{jk}(t|\mathbf{x}) = \mathbb{E}[N_{jk}(t \wedge R) \mid \mathbf{X} = \mathbf{x}]$.

The following lemma is key to providing a general estimation strategy when right-censoring is present.

Lemma 6.6. *Under the assumption $Z \perp\!\!\!\perp R \mid \mathbf{X}$, it holds that*

$$\Lambda_{jk}(t|\mathbf{x}) = \int_0^t \frac{1}{p^c_j(s-|\mathbf{x})} p^c_{jk}(ds|\mathbf{x}).$$

Proof. Using $Z \perp\!\!\!\perp R \mid \mathbf{X}$, we compute

$$p^c_j(t|\mathbf{x}) = \mathbb{E}[\mathbb{1}_{\{Z_t=j\}} \mid \mathbf{X} = \mathbf{x}]\mathbb{E}[\mathbb{1}_{\{t<R\}} \mid \mathbf{X} = \mathbf{x}] = p^c_j(t|\mathbf{x})\mathbb{P}(t < R|\mathbf{X} = \mathbf{x})$$

and

$$\begin{aligned} p^c_{jk}(t|\mathbf{x}) &= \mathbb{E}\left[\int_0^t \mathbb{1}_{\{s\leq R\}} N_{jk}(ds) \mid \mathbf{X} = \mathbf{x}\right] = \int_0^t \mathbb{E}[\mathbb{1}_{\{s\leq R\}} \mid \mathbf{X} = \mathbf{x}] d\mathbb{E}[N_{jk}(ds) \mid \mathbf{X} = \mathbf{x}] \\ &= \int_0^t \mathbb{P}(s \leq R \mid \mathbf{X} = \mathbf{x}) p_{jk}(ds|\mathbf{x}). \end{aligned}$$

Combining these results, we get

$$\int_0^t \frac{1}{p^c_j(s-|\mathbf{x})} p^c_{jk}(ds|\mathbf{x}) = \int_0^t \frac{\mathbb{P}(s \leq R \mid \mathbf{X} = \mathbf{x})}{p_j(s-|\mathbf{x})\mathbb{P}(s \leq R|\mathbf{X} = \mathbf{x})} p_{jk}(ds|\mathbf{x}) = \Lambda_{jk}(t|\mathbf{x})$$

as desired. ■

Remark 6.7. As pointed out in Remark 2.1 of Bladt and Furrer [6], if Z is Markov (see the following subsection), one can replace the assumption $Z \perp\!\!\!\perp R \mid \mathbf{X}$ with the weaker condition that the compensator of the multivariate counting process of Z is unchanged when adding censoring information, see chapter III.2.2 in Andersen et al. [4]. Indeed, imposing this assumption, we get

$$\begin{aligned}\int_0^t \frac{1}{p_j^c(s- \mid \mathbf{x})} p_{jk}^c(ds \mid \mathbf{x}) &= \int_0^t \frac{1}{p_j^c(s- \mid \mathbf{x})} d\mathbb{E}[\mathbb{1}_{\{s\leq R\}}\mathbb{1}_{\{Z_{s-}=j\}}\Lambda_{jk}(s \mid \mathbf{x}) \mid \mathbf{X}=\mathbf{x}] \\ &= \int_0^t \frac{p_j^c(s- \mid \mathbf{x})}{p_j^c(s- \mid \mathbf{x})} \Lambda_{jk}(ds \mid \mathbf{x}) = \Lambda_{jk}(t \mid \mathbf{x}).\end{aligned}$$

∘

6.2 The (smooth) Markov model

When we later consider consistency for Aalen–Johansen forests, the Markov assumption plays an essential role in certain calculations. We therefore provide a recap of the definition along with some essential properties. Along the way, we highlight the advantages of working with Markov models in several aspects. All results in regards to (cumulative) transition intensities and transition probabilities are stated without covariates, but the results are easily extended to include covariates. We fix a pure jump process $Z = (Z_t)_{t\geq 0}$ on a finite state space $\mathcal{Z}$ (although most results in this subsection may as well be stated for countable state spaces).

6.2.1 Definitions and basic properties

Definition 6.8. We let $\mathcal{F}^Z = (\mathcal{F}_t^Z)_{t\geq 0}$ denote the natural filtration generated by Z, that is, $\mathcal{F}^Z = \sigma(Z_u : u \leq t)$. We say that Z is a *Markov process* (or for brevity, that Z is Markov) if for every $s < t$,

$$Z_t \perp\!\!\!\perp \mathcal{F}_s^Z \mid Z_s.$$

This definition encapsulates how one should think of a Markov process intuitively, namely that the future is independent of the past given the present. In other words, the present contains all information necessary to infer about the future behaviour of the process. The following lemma is essential in computations and is often provided as the definition of a Markov process.

Lemma 6.9. *Z is Markov if and only if for every choice of $s_1 < \cdots < s_n < t$, we have*

$$Z_t \perp\!\!\!\perp (Z_{s_1}, ..., Z_{s_n}) \mid Z_{s_n},$$

that is, if and only if

$$\mathbb{P}(Z_t = k \mid Z_{s_n} = j_n, ..., Z_{s_1} = j_1) = \mathbb{P}(Z_t = k \mid Z_{s_n} = j_n)$$

for every $s_1 < \cdots < s_n$ and $k, j_1, ..., j_n \in \mathcal{Z}$.

Proof. Assume that for every $s_1 < \cdots < s_n < t$, we have $Z_t \perp\!\!\!\perp (Z_{s_1}, ..., Z_{s_n}) \mid Z_{s_n}$ and choose $s < t$. If $s_1 < \cdots < s_n \leq s < t$, we get

$$Z_t \perp\!\!\!\perp (Z_{s_1}, ..., Z_{s_n}, Z_s) \mid Z_s$$

so that in particular,

$$Z_t \perp\!\!\!\perp (Z_{s_1}, ..., Z_{s_n}) \mid Z_s.$$

As

$$\bigcup_{\substack{I \subseteq [0,s] \\ \#I < \infty}} \sigma(Z_u : u \in I)$$

is an $\cap$-stable generator for $\mathcal{F}_s^Z$, basic measure-theoretical arguments imply that $Z_t \perp\!\!\!\perp \mathcal{F}_s^Z \mid Z_s$. The converse statement is trivial. ■

In fact, the Markov property may be extended even more as the following result shows. The theorem solidifies our intuition of Markov processes that future information is independent of the past given the present.

Theorem 6.10. *If Z is Markov, then $\sigma(Z_u : u \geq t) \perp\!\!\!\perp \mathcal{F}_s^Z \mid Z_s$ for every $t > s$.*

See the appendix for an elementary proof of this result as well as some additional properties for conditional independence, following the arguments in Hansen [24]. Alternatively, consult Lemma 11.1 in Kallenberg [40]. This result has far-reaching consequences and is a major reason for the use of Markov models in life insurance. The lemma implies that $\mathbb{P}(A \mid \mathcal{F}_s^Z) = \mathbb{P}(A \mid Z_s)$ for every $A \in \sigma(Z_u : u \geq t)$ and $s < t$. Elementary measure theory implies that for every bounded measurable function $f : \mathcal{Z} \to \mathbb{R}$, we have

$$\mathbb{E}[f(Z_t) \mid \mathcal{F}_s^Z] = \mathbb{E}[f(Z_t) \mid Z_s],$$

and it is often the case that the right hand side is much easier to compute compared to the left hand side.

Example 6.11. An important quantity in life insurance is the *prospective reserve*

$$V(t) = \mathbb{E}\left[\int_t^T e^{-\int_t^s r(u)du} B(ds) \mid \mathcal{F}_t^Z\right]$$

where r is a deterministic interest rate and B is the payment stream consisting of contractual benefits minus premiums satisfying sufficient regularity conditions to ensure that the integral as well as the expected value is well-defined. T is a maximal contract time (120 years, say). If the payments only depend on the future position of Z and Z is Markov, we may write

$$V(t) = V^{Z(t)}(t) := \mathbb{E}\left[\int_t^T e^{-\int_t^s r(u)du} B(ds) \mid Z(t)\right].$$

This expression is easily computable, see e.g. Proposition 3.1 in Asmussen and Steffensen [5]. We also refer to chapter V of the same book for more background on Markov models in life insurance. ○

For any Markov jump process on $\mathcal{Z}$, we may define the *transition probabilities* p_{jk} by

$$p_{jk}(s,t) = \mathbb{P}(Z_t = k \mid Z_s = j), \quad s \leq t, \quad j,k \in \mathcal{Z},$$

and we bundle these probabilites into a matrix $p = (p_{jk})_{j,k\in\mathcal{Z}}$. As for a general multi-state model, the best approach for estimation is not to estimate p directly but via some auxiliary quantity. In practical applications it is often assumed that the Markov jump process is *smooth* which is a somewhat informal way of saying that the process has *intensities*.

Definition 6.12. If the limits

$$\alpha_{jk}(s) = \lim_{h\downarrow 0} \frac{p_{jk}(s,s+h) - \delta_{jk}}{h}$$

exist for all $j,k \in \mathcal{Z}$ and $s \geq 0$, we refer to these as the *transition intensities* for the process Z and to $\alpha = (\alpha_{jk})_{j,k\in\mathcal{Z}}$ as the matrix of transition intensities.

Note that for $j \neq k$, we have

$$p_{jk}(s,s+h) = \alpha_{jk}(s)h + o(h)$$

so that for small h, we may interpret $\alpha_{jk}(s)h$ as being approximately the probability of making a jump to state k in the interval $[s,s+h]$ conditional of being in state j at time s. It is easily seen that $\alpha_{jk}(s) \geq 0$ for all $s \geq 0, j \neq k$ and that $\alpha(s)\cdot\mathbf{1} = 0$ where $\mathbf{1} = (1,...,1)$ which implies that

$$\alpha_{jj}(s) = -\sum_{k:k\neq j} \alpha_{jk}(s).$$

In the following, we will use the notation

$$\alpha_{j\bullet}(s) = \sum_{k:k\neq j} \alpha_{jk}(s)$$

to denote the total intensity out of state j. With this notation, $\alpha_{jj}(s) = -\alpha_{j\bullet}(s)$. We see that

$$\begin{aligned} \alpha_{jk}(s)ds &= \text{probability of a jump from } j \text{ to } k \text{ in } [s,s+ds] \\ 1-\alpha_{j\bullet}(s)ds &= \text{probability of no jumps in } [s,s+ds] \text{ when in state } j \text{ at time } s, \end{aligned}$$

and combining these, we get that the probability of making a jump from state j to k in $[s,s+ds)$ conditional on a jump occuring is

$$\frac{\alpha_{jk}(s)}{\alpha_{j\bullet}(s)}. \tag{13}$$

Now define $T(s) := \inf\{u \geq 0 : Z(s+u) \neq Z(s)\}$ so that $T(s)$ is the time from s that Z jumps out of the state $Z(s)$. Given $Z(s) = j$, $T(s)$ is the time from s until

Z leaves j. Letting $S_j(t) := \mathbb{P}(T(s) > t \mid Z(s) = j)$, then since $-S_j'$ is the density of $T(s)$ given $Z(s) = j$, we get

$$\alpha_{j\bullet}(u+s) = -\frac{S_j'(u)}{S_j(u)} = -\frac{d}{ds}\log S_j(u)$$

from which it follows that

$$S_j(t) = \mathbb{P}(T(s) > t \mid Z(s) = j) = \exp\left(-\int_s^{s+t} \alpha_{j\bullet}(u)du\right). \tag{14}$$

6.2.2 The Markov process as a marked point process

An alternative to describing a Markov jump process via Z is via its *event times* and *marks*. To be precise, Z may equivalently be described as a *marked point process* in the sense of chapter 2 of Jacobsen [35]. For convenience, we recall the relevant definitions. Let E be a set equipped with a sigma-algebra $\mathcal{E}$. To denote an event which never occurs, we introduce an element ∇ referred to as the *irrelevant mark*. Let $\overline{E} = E \cup \{\nabla\}$ and $\overline{\mathcal{E}} = \sigma(\mathcal{E}, \{\nabla\})$ and fix a background probability space $(\Omega, \mathcal{F}, \mathbb{P})$.

Definition 6.13. A *simple point process* (SPP for short) $\mathcal{T}$ is a sequence $\mathcal{T} = (T_m)_{m\geq 1}$ of $[0,\infty]$-valued random variables defined on $(\Omega, \mathcal{F}, \mathbb{P})$ satisfying the following properties:

(i) $\mathbb{P}(0 < T_1 \leq T_2 \leq ...) = 1$,

(ii) $\mathbb{P}(T_m < T_{m+1}, T_m < \infty) = \mathbb{P}(T_m < \infty)$ for $m \geq 1$ and

(iii) $\mathbb{P}(\lim_{m\to\infty} T_m = \infty) = 1$.

Definition 6.14. A *marked point process* (MPP for short) with mark space E is a double sequence $(\mathcal{T}, \mathcal{Y}) = ((T_m)_{m\geq 1}, (Y_m)_{m\geq 1})$ with T_m $[0,\infty]$-valued variables and Y_m $\overline{E}$-valued variables both defined on $(\Omega, \mathcal{F}, \mathbb{P})$ satisfying that

(i) $\mathcal{T}$ is an SPP,

(ii) $\mathbb{P}(Y_m \in E, T_m < \infty) = \mathbb{P}(T_m < \infty)$ and

(iii) $\mathbb{P}(Y_m = \nabla, T_m = \infty) = \mathbb{P}(T_m = \infty)$ for $m \geq 1$.

Given a Markov jump process Z, we let $(\mathcal{Z}, \mathcal{P}(\mathcal{Z}))$ be the corresponding mark space. We may then define the jump times $\mathcal{T} = (T_m)_{m\geq 1}$ by

$$T_1 := \inf\{t \geq 0 : Z_t \neq Z_0\}, \quad T_m := \inf\{t \geq T_{m-1} : Z_t \neq Z_{T_{m-1}}\}, \quad m \geq 2.$$

We may then define the mark sequence $\mathcal{Y} = (Y_m)_{m\geq 1}$ by $Y_m = Z_{T_m}$ for $m \geq 1$. In later considerations, we define $T_0 := 0$ for convenience. Y_0 is then the starting state of Z. Note that we can easily recover Z from the double sequence $(\mathcal{T}, \mathcal{Y})$ by

$$Z_t = \sum_{m=0}^{\infty} Y_m \mathbb{1}_{[T_m, T_{m+1})}(t).$$

Using this correspondence between Z and $(\mathcal{T}, \mathcal{Y})$, it immediately follows that $\sigma(Z) = \sigma((T_m, Y_m) : m \geq 0)$, that is, the two processes Z and $(\mathcal{T}, \mathcal{Y})$ generate the same information. In particular, all independence statements concerning Z translate one-to-one to independence statements about $(\mathcal{T}, \mathcal{Y})$. A particularly useful example is the assumption $Z \perp\!\!\!\perp R \mid \mathbf{X}$ which then holds if and only if $(T_m, Y_m)_{m\geq 0} \perp\!\!\!\perp R \mid \mathbf{X}$. Before moving on, we translate equation (14) into the context of MPP's.

Lemma 6.15. *Let Z be a Markov pure jump process with intensities α_{jk} and corresponding MPP $(\mathcal{T}, \mathcal{Y}) = ((T_m)_{m\geq 1}, (Y_m)_{m\geq 1})$. We then have*

$$\mathbb{P}(T_{m+1} > t \mid Y_m = j, T_m = s) = \exp\left(-\int_s^t \alpha_{j\bullet}(u)du\right), \quad s \leq t$$

and $\mathbb{P}(T_{m+1} > t \mid Y_m = j, T_m = s) = 1$ if $s > t$.

Proof. Let $s \leq t$ and note that, in this case, $T(s) \overset{\mathrm{d}}{=} T_{m+1} - s \mid T_m = s$. Hence

$$\begin{aligned}\mathbb{P}(T_{m+1} > t \mid Y_m = j, T_m = s) &= \mathbb{P}(T_{m+1} - s > t - s \mid Y_m = j, T_m = s) \\ &= \mathbb{P}(T(s) > t - s \mid Z_s = j, T_m = s).\end{aligned}$$

Note that $\{T(s) > t - s\} \in \sigma(Z_u : u \geq t)$ and $\sigma(Z_u : u \geq t) \perp\!\!\!\perp \mathbb{F}_s^Z \mid Z_s$ by Theorem 6.10. Since $\{T_m = s\} \in \mathbb{F}_s^Z$, we have

$$\mathbb{P}(T_{m+1} > t \mid Y_m = j, T_m = s) = \mathbb{P}(T(s) > t - s \mid Z_s = j) = \exp\left(-\int_s^t \alpha_{j\bullet}(u)du\right)$$

using equation (14). The case $s > t$ is trivial since $T_{m+1} \geq T_m$ a.s. This completes the proof. ■

6.2.3 The Kolmogorov equations

In this subsection, we prove an essential result on the transition probabilities $p(s,t)$ for a Markov jump process. This result is crucial in both estimation and computations in general.

Theorem 6.16 (**The Kolmogorov equations**). *When Z is smooth with intensity matrix α, it holds that*

$$p(s,t) = \prod_{(s,t]} (I + \alpha(u)du), \quad s \leq t.$$

Proof. Fix $s \leq t$ and let $p_{jk}^{(n)}(s,t)$ be the probability of going from state j at time s to state k at time t in at most n jumps. We note that

$$p_{jk}^{(0)}(s,t) = \delta_{jk} \exp\left(-\int_s^t \alpha_{j\bullet}(u)du\right)$$

using equation (14) and the fact that one can only go from state j to k in zero jumps if $j = k$. As for $p_{jk}^{(n)}(s,t)$ with $n \geq 1$, we have two cases. If no jump occurs, we

get $p_{jk}^{(0)}(s,t)$. If a jump occurs at time $u \in (s,t)$ to, say, state l, it happens with probability

$$\mathbb{P}(T(s) \in du \mid Z(s) = j)\frac{\alpha_{jl}(u)}{\alpha_{j\bullet}(u)}$$

by the discussion above. After the jump, we have $n-1$ jumps remaining to get from l to k in the interval (u,t). Since

$$\mathbb{P}(T(s) \in du \mid Z(s) = j) = \alpha_{j\bullet}(u)\exp\left(-\int_s^u \alpha_{j\bullet}(v)dv\right)du$$

and as the jump can occur at any time in (s,t) to any state $l \neq j$, we get

$$p_{jk}^{(n)}(s,t) = p_{jk}^{(0)}(s,t) + \sum_{l:l\neq j}\int_s^t \exp\left(-\int_s^u \alpha_{j\bullet}(v)dv\right)\alpha_{jl}(u)p_{lk}^{(n-1)}(u,t)du.$$

From this identity, it follows that $p_{jk}^{(n)}(s,t)$ is differentiable in both s and t. Using Leibniz' rule, we obtain

$$\begin{aligned}
\frac{\partial}{\partial s}p_{jk}^{(n)}(s,t) &= \frac{\partial}{\partial s}p_{jk}^{(0)}(s,t) + \sum_{l:l\neq j}\int_s^t \alpha_{j\bullet}(s)\exp\left(-\int_s^u \alpha_{j\bullet}(v)dv\right)\alpha_{jl}(u)p_{lk}^{(n-1)}(u,t)du \\
&\quad - \sum_{l:l\neq j}\alpha_{jl}(s)p_{lk}^{(n-1)}(s,t) \\
&= \delta_{jk}\alpha_{j\bullet}(s)\exp\left(-\int_s^u \alpha_{j\bullet}(v)dv\right) - \sum_{l:l\neq j}\alpha_{jl}(s)p_{lk}^{(n-1)}(s,t) \\
&\quad + \sum_{l:l\neq j}\int_s^t \alpha_{j\bullet}(s)\exp\left(-\int_s^u \alpha_{j\bullet}(v)dv\right)\alpha_{jl}(u)p_{lk}^{(n-1)}(u,t)du \\
&= \alpha_{j\bullet}(s)p_{jk}^{(n)}(s,t) - \sum_{l:l\neq j}\alpha_{jl}(s)p_{lk}^{(n-1)}(s,t).
\end{aligned}$$

Note that the $p_{jk}^{(n)}(s,t)$ are bounded with $p_{jk}^{(n)}(s,t) \uparrow p_{jk}(s,t)$ for $n \to \infty$, so taking limits in the above expression, we get

$$\frac{\partial}{\partial s}p_{jk}(s,t) = -\alpha_{jj}(s)p_{jk}(s,t) - \sum_{l:l\neq j}\alpha_{jl}(s)p_{lk}(s,t) = -\sum_{l\in\mathcal{Z}}\alpha_{jl}(s)p_{lk}(s,t)$$

which in matrix form means

$$\frac{\partial}{\partial s}p(s,t) = -\alpha(s)p(s,t).$$

Since $p(t,t) = I$, we have shown that p satisfies the integral equation

$$p(s,t) = I + \int_s^t \alpha(u)p(u,t)du.$$

The desired result now follows by applying Theorem 3.8. ■

The equation

$$\frac{\partial}{\partial s}p(s,t) = -\alpha(s)p(s,t)$$

which we proved directly above is called the *Kolmogorov backward equation*. Using Theorem 3.8, we obtain the *Kolmogorov forward equation*

$$\frac{\partial}{\partial t}p(s,t) = p(s,t)\alpha(t).$$

Corollary 6.17 (The Chapman–Kolmogorov equations). *For any $s \leq u \leq t$, it holds that*

$$p(s,t) = p(s,u)p(u,t).$$

Proof. Combine the result of the previous theorem with Proposition 3.5. ■

6.2.4 Counting processes and compensators for a Markov process

In the survival model, there was a strong relationship between the intensity and the compensator for the counting process, see example 3.23. A very similar relation holds for a Markov jump process. The following result is theorem II.6.8 in Andersen et al. [4]. A proof may be found in Jacobsen [34].

Theorem 6.18. *Let Z be a Markov jump process with intensity matrix α. Consider the natural filtration $\mathcal{F}^Z$. Define*

$$Y_j(t) = \mathbb{1}_{\{Z_{t-}=j\}}, \quad N_{jk}(t) = \#\{s \in (0,t] : Z_{s-} = j, Z_s = k\}, \quad j \neq k.$$

Then $N = (N_{jk})_{j\neq k}$ is a multivariate counting process, and the predictable compensators are given by

$$\int_0^t Y_j(s)\alpha_{jk}(s)ds.$$

Equivalently, the processes M_{jk} defined by

$$M_{jk}(s) = N_{jk}(s) - \int_0^t Y_j(s)\alpha_{jk}(s)ds$$

are martingales.

Right-censoring is incorporated in the same fashion as in the survival setup but for completeness, we describe it in detail. Say we are given data

$$((Z_t)_{0\leq t\leq R^i}^i, \tau_{Z^i} \wedge R^i)_{i=1}^n.$$

We then form the aggregate (censored) counting processes

$$N_{jk}^{(n)}(t) = \sum_{i=1}^n N_{jk}^i(t \wedge R^i)$$

where N^i_{jk} is the counting process for observation i. Since $N^i_{jk}(\cdot \wedge R^i)$ has compensator

$$\int_0^t Y_j(s)\alpha_{jk}(s)ds \quad \text{where} \quad Y_j(s) = \mathbb{1}_{\{Z_{s-}=j\}}\mathbb{1}_{\{s<R^i\}},$$

the aggregate process $N^{(n)}_{jk}$ has compensator given by

$$\int_0^t Y^{(n)}_j(s)\alpha_{jk}(s)ds \quad \text{where} \quad Y^{(n)}_j(s) = \sum_{i=1}^n \mathbb{1}_{\{Z_{s-}=j\}}\mathbb{1}_{\{s<R^i\}}.$$

Define

$$J^{(n)}_j(s) = \mathbb{1}_{\{Y^{(n)}_j(s)>0\}},$$

then we may form Nelson–Aalen estimators of the cumulative transition rates by

$$\widehat{\Lambda}^{(n)}_{jk}(t) = \int_0^t \frac{J^{(n)}_j(s)}{Y^{(n)}_j(s)} N^{(n)}_{jk}(ds).$$

Given the form of the compensators and estimators just described, the following result is hardly surprising.

Theorem 6.19. *Consider a Markov jump process with intensity matrix $\alpha = (\alpha_{jk})_{j,k\in\mathcal{Z}}$. Let $\tau = \sup\{u \geq 0 : \int_0^u \alpha_{jk}(v)dv < \infty, j \neq k\}$. Let $s, t \in [0, \tau)$ with $s < t$. Assume that for $n \to \infty$, we have for every $j \neq k$ that*

$$\int_s^t \frac{J^{(n)}_j(u)}{Y^{(n)}_j(u)}\alpha_{jk}(u)du \xrightarrow{\mathbb{P}} 0 \tag{i}$$

and

$$\int_s^t (1 - J^{(n)}_j(u))\alpha_{jk}(u)du \xrightarrow{\mathbb{P}} 0. \tag{ii}$$

Then as $n \to \infty$

$$\sup_{u\in[s,t]} |\widehat{p}(s,u) - p(s,u)| \xrightarrow{\mathbb{P}} 0$$

where

$$\widehat{p}(s,t) = \prod_{(s,t]} (I + \widehat{\Lambda}^{(n)}(du)).$$

Proof. Simply copy the proof of Theorem 4.5 and Corollary 4.6. ■

Remark 6.20. Since the proof of the above result is identical to the one for 4.5, it relies heavily on the martingale machinery presented in the preliminaries section. It is important to be aware that such techniques simply do not apply when the Markov assumption is omitted. Thus consistency results like the one above will have to be established by other means, for example via the assumption of independent right-censoring. ∘

6.2.5 The case without intensities

Here is a good place to briefly discuss the case without intensities. In some applications, it can be an obstacle to assume intensities, for example in applications with non-zero probability of jumps at specific timepoints. A relevant example from life insurance which has attracted some attention in recent years is the problem of stochastic retirement as described in Gad and Nielsen [20].

In this more general setting, we simply work with the cumulative transition rates Λ as given in the beginning of this section. One can then verify that Theorem 6.16 still holds,

$$p(s,t) = \prod_{(s,t]} (I + \Lambda(du)),$$

and Theorem 6.18 also generalises as one would expect. The compensators become

$$\int_0^t Y_j(s)\Lambda_{jk}(ds).$$

Furthermore, equation (13) becomes

$$\frac{\Lambda_{jk}(ds)}{\Lambda_{j\bullet}(ds)}$$

where $\Lambda_{j\bullet}(s) = \sum_{k:k\neq j} \Lambda_{jk}(s)$, while equation (14) reads

$$S_j(t) = \mathbb{P}(T(s) > t \mid Z_s = j) = \exp\left(\Lambda_{j\bullet}(s+t) - \Lambda_{j\bullet}(s)\right),$$

and as a direct consequence, Lemma 6.15 states in this case that

$$\mathbb{P}(T_{m+1} > t \mid Y_m = j, T_m = s) = \exp\left(\Lambda_{j\bullet}(t) - \Lambda_{j\bullet}(s)\right)$$

whenever $s < t$.

6.2.6 Advantages and disadvantages of the Markov assumption

Let us close this subsection by considering the implications of working with the Markov assumption. There are several **advantages** of working with the Markov assumption, some of which we already touched upon:

- The model is mathematically tractable and allows the use of martingale techniques to prove large sample properties. The Markov assumption also eases certain computations of quantities based on the estimates of the transition rates and transition probabilities such as prospective reserves in life insurance.
- In a general multi-state model, the goal is to estimate the occupation probabilities
$$q(t \mid \mathbf{x}) = \prod_{(0,t]} (I + \Lambda(du \mid \mathbf{x})).$$

The estimator of q is the same in both models. In a life insurance context though, the quantity of interest is the matrix of transition probabilities $p(s,t \mid \mathbf{x})$. In a Markov model, this quantity comes for free from q via the Kolmogorov equations and alternatively via the relation

$$p(s,t \mid \mathbf{x}) = q(s \mid \mathbf{x})^{-1} q(t \mid \mathbf{x}).$$

In a non-Markov model, this relationsship is not guaranteed to hold. Instead, one has to condition on the internal covariate Z_s by conditioning on $(\mathbf{X}, Z_s)$ instead of just $\mathbf{X}$. This introduces subsampling, which increases the variance of the estimators, decreasing the robustness of the results.

That being said, there are serious **disadvantages** of assuming the Markov property. An obvious problem is that the model may be misspecified. If the underlying data-generating process is not truly Markov, it does not make sense to use $\widehat{p}(s,t)$ as an estimate of $p(s,t)$ for $0 < s \leq t$. In many cases, we can say with near certainty that the Markov assumption fails. As an example, consider the disability model from life insurance.

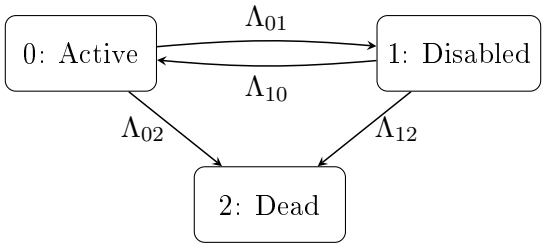

Figure 19: The disability model.

It is realistic to assume that Λ_{10} and Λ_{12} depend on the duration in state 1, and a Markov model is incapable of capturing this phenomenon. In general, any model with duration dependent intensities cannot satisfy the Markov assumption. Also, for a sufficiently complicated model, it may not be reasonable to assume that the behaviour of the process in the future does not depend on the past history, even when given the current state of the insured.

In conclusion, while the Markov property guarantees nice mathematical properties and more robust results, in many cases, it simply does not make sense to assume the Markov property. It is therefore essential to have tools that work for both. The advantage of our setup is that estimating q is the same, regardless of whether the Markov assumption holds or not. There is however still the question of consistency for a general multi-state model. Such results will need to be established via more general techniques than the martingale approach presented above.

7 Extending RSF to multi-state models

> "If I have seen further, it is by standing on the shoulders of giants."
> - Isaac Newton

This section touches upon all the work related to extending random survival forests to multi-state models. As with RSF, there is both a practical aspect (splitting rules, evaluation, implementation details etc.) and a theoretical aspect in the question of consistency. We choose to focus on the latter in this project.

7.1 The Aalen–Johansen forest

The goal of this subsection is to describe the Aalen–Johansen forest, an extension of random survival forests to a multi-state setup. We first describe the algorithm very briefly and then we touch upon the Nelson–Aalen estimator in this setup.

7.1.1 The Aalen–Johansen forest procedure

The Aalen–Johansen forest algorithm is very reminiscent of the RSF procedure and works as follows. We will in the following use the abbreviation *AaJo* instead of Aalen–Johansen.

1. Draw B bootstrap samples from the data.
2. Grow an AaJo tree on each bootstrap sample. In each split, we randomly choose p candidate features from $\mathbf{X}$. We use a splitting rule that in some way maximises the difference in some quantity of interest between the two nodes. The tree is grown to full size under the condition that a terminal node should have at least $d_{0j} > 0$ distinct observed jumps out of state j for every non-absorbing state j ($d_{0j} = 0$ for an absorbing state).
3. In each tree, calculate the matrix of Nelson–Aalen estimators (see equation (16) below) in every terminal node.
4. Compute some error estimate using OOB data to check the quality of the fit.

We see that the procedure is very reminiscent of the one for survival trees. A key difference is the number of hyperparameters. The AaJo forest requires a whole vector of d_{0j}. In practice, one could pass a vector $(d_{0j})_{j\in\mathcal{Z}}$, where $d_{0j} = 0$ indicates that the node is absorbing and $d_{0j} > 0$ that the node is non-absorbing. Alternatively, one could just continue working with a minimum number of observations in a node. In this case, it may be necessary to specify somehow which nodes are absorbing.

7.1.2 Prediction

We begin by relating the multi-state terminology to an example, we have already extensively studied, namely the survival model. Here we only have a single transition as illustrated in the following figure.

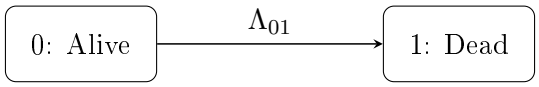

Figure 20: The survival model.

Hence the cumulative transition rate matrix becomes the classical two by two matrix

$$\Lambda(t|\mathbf{x}) = \begin{pmatrix} -\Lambda_{01}(t|\mathbf{x}) & \Lambda_{01}(t|\mathbf{x}) \\ 0 & 0 \end{pmatrix}.$$

If T° denotes the true survival time and $T = T^\circ \wedge R$ as before, we see that

$$p_0^c(t|\mathbf{x}) = \mathbb{P}(s \leq T^\circ, s \leq R \mid \mathbf{X} = \mathbf{x}) = \mathbb{P}(T \geq s \mid \mathbf{X} = \mathbf{x}),$$

and if N_{01}° is the true jump process and N_{01} the censored process, we get

$$\Lambda_{01}(t|\mathbf{x}) = \int_0^t \frac{1}{\mathbb{P}(T \geq s \mid \mathbf{X} = \mathbf{x})} d\mathbb{E}[N_{01}(ds|\mathbf{x})].$$

Say we have grown a tree and that $\mathbf{x}$ lands in the terminal node h with the data $(T_{i,h}, \delta_{i,h})_{i=1}^{n_h}$. Then $\mathbb{P}(T \geq s \mid \mathbf{X} = \mathbf{x})$ is estimated by

$$\frac{1}{n_h} Y_s^h = \frac{1}{n_h} \sum_{i=1}^{n_h} \mathbb{1}_{\{T_{i,h} \geq s\}},$$

and $\mathbb{E}[N_{01}(s) \mid \mathbf{X} = \mathbf{x}]$ is estimated by

$$\frac{1}{n_h} N_s^h = \frac{1}{n_h} \sum_{i=1}^{n_h} N_{01}^i(s) = \frac{1}{n_h} \sum_{i=1}^{n_h} \mathbb{1}_{\{T_{i,h} \leq s\}} \delta_{i,h}$$

and so we get

$$\widehat{\Lambda}_{01}^h(t|\mathbf{x}) = \int_0^t \frac{J_s^h}{Y_s^h} N^h(ds)$$

which coincides with $\widehat{H}_h(t)$ from earlier. We see that the main objective is to generalise the estimators of $p_j^c(s - |\mathbf{x})$ and $p_{jk}^c(s|\mathbf{x})$.

Say we have a tree and we consider a particular terminal node h with data

$$(\mathbf{X}^{i,h}, (Z_t^{i,h})_{0 \leq t \leq R^{i,h}}, \tau_{Z^{i,h}} \wedge R^{i,h})_{i=1}^{n_h}. \tag{15}$$

We see that an estimator of $p_j^c(s - |\mathbf{x})$ is

$$\widehat{p}_j^{c,h}(s - |\mathbf{x}) = \frac{1}{n_h} \sum_{i=1}^{n_h} \mathbb{1}_{\{Z_{s-}^{i,h} = j\}} \mathbb{1}_{\{s < R^{i,h}\}}$$

while $p_{jk}^c(s|\mathbf{x})$ is estimated by

$$\widehat{p}_{jk}^{c,h}(s|\mathbf{x}) = \frac{1}{n_h} \sum_{i=1}^{n_h} N_{jk}(s \wedge R^{i,h}).$$

Combining these leads to the estimator

$$\widehat{\Lambda}_{jk}^{h}(t) = \int_0^t \frac{1}{\widehat{p}_j^{c,h}(s - |\mathbf{x})} \widehat{p}_{jk}^{c,h}(ds|\mathbf{x}).$$

For computational purposes, a different representation is practical. Given the data (15) in h, we may compute the following quantities.

(i) The sequence of event times across all jump processes N_{jk},

$$0 < t_{1,h} < \cdots < t_{N_h,h}.$$

(ii) The number of individuals in each state of $\mathcal{Z}$ just before the jump at each event time,

$$(Y_1^{j,h}, ..., Y_{N_h}^{j,h})_{j \in \mathcal{Z}}.$$

To be precise, $Y_i^{j,h}$ is the number of individuals residing in state j immediately before time $t_{i,h}$.

(iii) The number of observed/true jumps from j to k at each event time

$$(d_1^{j,k,h}, ..., d_{N_h}^{j,k,h})_{j,k \in \mathcal{Z}, j \neq k}.$$

In terms of these quantities, we may write

$$\widehat{\Lambda}_{jk}^{h}(t) = \sum_{l : t_{l,h} \leq t} \frac{d_l^{j,k,h}}{Y_l^{j,h}} \tag{16}$$

which is of the exact same form as the Nelson–Aalen estimator in the survival model. Bundling these together in a matrix, we obtain the estimator $\widehat{\Lambda}^h$ for the cumulative transition probabilities in node h. Given a feature vector $\mathbf{x}$, we compute the predicted value $\widehat{\Lambda}(t \mid \mathbf{x})$ by dropping $\mathbf{x}$ down the tree,

$$\widehat{\Lambda}(t \mid \mathbf{x}) = \widehat{\Lambda}^h(t), \quad \text{if } \mathbf{x} \in h.$$

When we grow a forest of AaJo trees, there are two ways of computing an ensemble cumulative transition rate matrix, the *OOB ensemble* and the *bootstrap ensemble*. As with the survival forest, let $I_{i,b} = 1$ when observation i is an OOB case for b and 0 otherwise. Let $\widehat{\Lambda}_b^*$ denote the estimate from the tree grown on the b'th bootstrap sample. The OOB ensemble is then the matrix $\widehat{\Lambda}_e^{**}$ with entries

$$\widehat{\Lambda}_{e,jk}^{**}(t \mid \mathbf{x}_i) = \frac{\sum_{b=1}^{B} I_{i,b} \widehat{\Lambda}_{b,jk}^{*}(t \mid \mathbf{x}_i)}{\sum_{b=1}^{B} I_{i,b}},$$

that is, we take the average only over the cases where i is OOB. The bootstrap ensemble for observation i is given by

$$\widehat{\Lambda}_{e,jk}^{*}(t \mid \mathbf{x}_i) = \frac{1}{B} \sum_{b=1}^{B} \widehat{\Lambda}_b^{*}(t \mid \mathbf{x}_i).$$

7.2 Consistency for a single Markov AaJo tree

In this subsection, we present the work done in this project on consistency of the Markov AaJo forest. The goal is to prove consistency of the Aalen–Johansen estimator. We wish to estimate the matrix of transition probabilities given covariates

$$p(s,t \mid \mathbf{x}) = \prod_{(s,t]} (I + \Lambda(du \mid \mathbf{x})).$$

Prediction works as with the Nelson–Aalen estimator, namely by dropping $\mathbf{x}$ down the tree. $\mathbf{x}$ lands in a unique terminal node h, and so the predicted value $\widehat{p}(s,t \mid \mathbf{x})$ is given by

$$\widehat{p}(s,t \mid \mathbf{x}) = \widehat{p}_h(s,t), \quad \mathbf{x} \in h$$

where

$$\widehat{p}_h(s,t) = \prod_{(s,t]} (I + \widehat{\Lambda}^h(du)).$$

As with the survival setup, the first step in proving consistency is to focus on a single tree. Let $\tau(\mathbf{x}) = \sup\{t \geq 0 : \int_0^t \alpha_{jk}(s \mid \mathbf{x})ds < \infty, j \neq k\}$ and $\tau = \min\{\tau(\mathbf{x}) : \mathbf{x} \in \mathcal{X}\}$. We make the following assumptions. To make the following proofs easier to follow, we also recall earlier assumptions.

Assumption 7.1. We assume the following.

- Z has intensities,
$$\Lambda_{jk}(t \mid \mathbf{x}) = \int_0^t \alpha_{jk}(s \mid \mathbf{x})ds,$$
and $\#\mathcal{Z} > 1$ so that the model is non-trivial.
- The model is connected considered as an undirected graph. In other words, there is at least one non-zero intensity to or from any node. This implies in particular that there are no isolated/unreachable nodes.
- Right censoring is entirely random given covariates, $Z \perp\!\!\!\perp R \mid \mathbf{X}$.
- If $\tau_R := \sup\{t \geq 0 : F_R(t) < 1\}$ denotes the upper endpoint of the distribution of R, we have $\mathbb{P}(R \in (u,v)) > 0$ for every $0 \leq u < v < \tau_R$. We furthermore assume $\tau_R > 0$ so that R is not the one-point measure in zero.
- For every $A \neq \emptyset$, we have $\mathbb{P}(\mathbf{X} \in A) > 0$.
- The feature space $\mathcal{X}$ is discrete and has finite cardinality.

Theorem 7.2. *Assume that $\alpha_j(\cdot \mid \mathbf{x})$ is strictly positive on $[0,t]$ for every non-absorbing state j and every $\mathbf{x} \in \mathcal{X}$. Let $t \in (0, \tau(\mathbf{x}) \wedge \tau_R)$. Then*

$$\sup_{u \in [s,t]} |\widehat{p}(s,u \mid \mathbf{x}) - p(s,u \mid \mathbf{x})| \xrightarrow{\mathbb{P}} 0 \quad \textit{as} \quad n \to \infty.$$

For $t \in (0, \tau \wedge \tau_R)$, we furthermore have

$$\sup_{u \in [s,t]} \int_{\mathcal{X}} |\widehat{p}(s,u \mid \mathbf{x}) - p(s,u \mid \mathbf{x})| \mathbb{P}(\mathbf{X} \in d\mathbf{x}) \xrightarrow{\mathbb{P}} 0 \quad as \quad n \to \infty.$$

The proof is very reminiscent of the one for Theorem 5.4, but the computations are slightly more complicated and are therefore treated separately.

Lemma 7.3. *Under assumptions 7.1 and the assumptions in Theorem 7.2, we have for any non-absorbing state j and any $\mathbf{x} \in \mathcal{X}$ that*

$$\mathbb{P}(Y_m = j, T_{m+1} \leq R \mid \mathbf{X} = \mathbf{x}) > 0$$

for at least one $m \in \mathbb{N}_0$, where $((T_m)_{m \geq 1}, (Y_m)_{m \geq 1})$ is the MPP representation of Z.

Proof. Using that $Z \perp\!\!\!\perp R \mid \mathbf{X}$ and the tower property, we have

$$\mathbb{P}(Y_m = j, T_{m+1} \leq R \mid \mathbf{X} = \mathbf{x}) = \int_0^\infty \mathbb{P}(Y_m = j, T_{m+1} \leq r \mid \mathbf{X} = \mathbf{x}) \mathbb{P}(R \in dr).$$

By the assumptions on R, we only need to show that the integrand is non-negative for at least one $m \in \mathbb{N}_0$ and some $0 < r \leq t$. Since

$$\mathbb{P}(Y_m = j, T_{m+1} \leq r \mid \mathbf{X} = \mathbf{x}) = \mathbb{P}(T_{m+1} \leq r \mid Y_m = j, \mathbf{X} = \mathbf{x}) \mathbb{P}(Y_m = j \mid \mathbf{X} = \mathbf{x}),$$

it suffices to show

1. $\mathbb{P}(T_{m+1} \leq r \mid Y_m = j, \mathbf{X} = \mathbf{x}) > 0$ and
2. $\mathbb{P}(Y_m = j \mid \mathbf{X} = \mathbf{x}) > 0$

for at least one $m \in \mathbb{N}_0$ (the same m for both 1 and 2) and all $r < t$. As for 1, the tower property yields

$$\mathbb{P}(T_{m+1} \leq r \mid Y_m = j, \mathbf{X} = \mathbf{x}) = 1 - \mathbb{E}[\mathbb{P}(T_{m+1} > r \mid Y_m = j, \mathbf{X} = \mathbf{x}, T_m)]$$

so we have to determine the conditional probability $\mathbb{P}(T_{m+1} > r \mid Y_m = j, \mathbf{X} = \mathbf{x}, T_m)$. Consider the event $T_m = s$. If $s < r$, then

$$\mathbb{P}(T_{m+1} > r \mid Y_m = j, \mathbf{X} = \mathbf{x}, T_m = s) = \exp\left(- \int_s^r \alpha_{j\bullet}(u \mid \mathbf{x}) du \right)$$

by Lemma 6.15, and if $s \geq r$, $\mathbb{P}(T_{m+1} > r \mid Y_m = j, \mathbf{X} = \mathbf{x}, T_m) = 1$. Thus,

$$\mathbb{P}(T_{m+1} > r \mid Y_m = j, \mathbf{X} = \mathbf{x}) = \mathbb{E}\left[\exp\left(- \int_{T_m}^r \alpha_{j\bullet}(u \mid \mathbf{x}) du \right) \mathbb{1}_{\{T_m < r\}} + \mathbb{1}_{\{T_m \geq r\}} \right],$$

from which we get

$$\begin{aligned} \mathbb{P}(T_{m+1} \leq r \mid Y_m = j, \mathbf{X} = \mathbf{x}) &= \mathbb{E}\left[\mathbb{1}_{\{T_m < r\}} - \exp\left(- \int_{T_m}^r \alpha_{j\bullet}(u \mid \mathbf{x}) du \right) \mathbb{1}_{\{T_m < r\}} \right] \\ &= \mathbb{E}\left[\mathbb{1}_{\{T_m < r\}} \left(1 - \exp\left(- \int_{T_m}^r \alpha_{j\bullet}(u \mid \mathbf{x}) du \right) \right) \right]. \end{aligned}$$

By the assumption that $\alpha_{j\bullet}(\cdot \mid \mathbf{x})$ is strictly positive on $[0,t]$, the above quantity is strictly positive as long as $\mathbb{P}(T_m < r) > 0$ or equivalently, that $\mathbb{P}(T_m \leq r) > 0$. Using the convention $T_0 = 0$, $\mathbb{P}(T_0 \leq r) = 1$ while for $m \in \mathbb{N}$, we have by similar calculations as before that

$$\begin{aligned}\mathbb{P}(T_m \leq r) &= \mathbb{E}[\mathbb{P}(T_m \leq r \mid Y_{m-1}, T_{m-1}, \mathbf{X})] \\ &= \mathbb{E}\left[\mathbb{1}_{\{T_{m-1} \leq r\}}\left(1 - \exp\left(-\int_{T_m}^{r} \alpha_{Y_{m-1}\bullet}(u \mid \mathbf{X})du\right)\right)\right].\end{aligned}$$

This implies that $\mathbb{P}(T_m \leq r) > 0$ if and only if Y_{m-1} is non-absorbing and $T_{m-1} < r$ with positive probability. This observation implies that we have two cases. Either there is some $M \in \mathbb{N}$ such that Y_m is absorbing a.s. for $m \geq M$ which implies a hiearchical structure of the model without cycles. Otherwise, no Y_m is absorbing a.s. In the first case, choose the smallest M such that Y_m is a.s. absorbing for $m \geq M$ and in the other, simply set $M = \infty$. Here we should note that $M \geq 1$ since we assume a non-trivial model. In any case, for $m = 0, 1, ..., M-1$, we have $\mathbb{P}(T_m \leq r) > 0$. It follows that

$$\mathbb{P}(T_{m+1} \leq r \mid Y_m = j, \mathbf{X} = \mathbf{x}) > 0$$

for every $r < t$ and $m < M$ whenever j is non-absorbing. We now turn our attention to 2. Assume for the sake of contradiction that

$$\mathbb{P}(Y_m = j \mid \mathbf{X} = \mathbf{x}) = 0$$

for every $m < M$. Since Y_m is absorbing a.s. for $m \geq M$, we must then have $\mathbb{P}(Y_m = j \mid \mathbf{X} = \mathbf{x}) = 0$ for every $m \in \mathbb{N}_0$ (since j is not absorbing). But this implies that j is an unreachable state, which is impossible by assumption. We conclude that there is at least one $m < M$ such that

$$\mathbb{P}(Y_m = j \mid \mathbf{X} = \mathbf{x}) > 0.$$

We conclude that we have found at least one $m \in \mathbb{N}_0$ such that both 1 and 2 hold. This completes the proof of the lemma. ■

Lemma 7.4. *Under assumptions 7.1 and the assumptions in Theorem 7.2, we have for any non-absorbing state j that*

$$\mathbb{P}(Z_{u-} = j \mid \mathbf{X} = \mathbf{x}) > 0$$

for every $\mathbf{x} \in \mathcal{X}$ and any $u \in [0,t]$.

Proof. Note that $\{Z_{u-} = j\} = \{Y_m = j, T_{m+1} \leq u \text{ for some } m \in \mathbb{N}_0\}$. Hence it suffices to show that

$$\mathbb{P}(Y_m = j, T_{m+1} \geq u \mid \mathbf{X} = \mathbf{x}) > 0$$

for some $m \in \mathbb{N}_0$. By the tower property and the assumption $Z \perp\!\!\!\perp R \mid \mathbf{X}$,

$$\begin{aligned}\mathbb{P}(Y_m = j, T_{m+1} \geq u \mid \mathbf{X} = \mathbf{x}) &\geq \mathbb{P}(Y_m = j, u \leq T_{m+1} \leq R \mid \mathbf{X} = \mathbf{x}) \\ &= \int_u^\infty \mathbb{P}(Y_m = j, u \leq T_{m+1} \leq r \mid \mathbf{X} = \mathbf{x}) \mathbb{P}(R \in dr)\end{aligned}$$

which is equal to

$$\mathbb{P}(Y_m = j \mid \mathbf{X} = \mathbf{x}) \int_u^\infty \mathbb{P}(u \leq T_{m+1} \leq r \mid \mathbf{X} = \mathbf{x}, Y_m = j) \mathbb{P}(R \in dr).$$

The first factor is non-zero for at least one m by the proof of the previous lemma. Like earlier, it suffices to show that the integrand is non-zero for this particular m and $r \leq t$. Recall the result

$$\mathbb{P}(T_{m+1} \leq r \mid Y_m = j, \mathbf{X} = \mathbf{x}) = \mathbb{E}\left[\mathbb{1}_{\{T_m < r\}} \left(1 - \exp\left(-\int_{T_m}^r \alpha_{j\bullet}(u \mid \mathbf{x}) du\right)\right)\right] =: P_{m,j,\mathbf{x}}(r)$$

which was established during the proof of the previous lemma. We now compute $\mathbb{P}(u \leq T_{m+1} \leq r \mid \mathbf{X} = \mathbf{x}, Y_m = j)$ as follows. Start by observing that

$$\mathbb{P}(u \leq T_{m+1} \leq r \mid \mathbf{X} = \mathbf{x}, Y_m = j) = P_{m,j,\mathbf{x}}(r) - P_{m,j,\mathbf{x}}(u).$$

Written out, the right hand side is

$$\begin{aligned}&\mathbb{E}\left[\mathbb{1}_{\{T_m < r\}} \left(1 - \exp\left(-\int_{T_m}^r \alpha_{j\bullet}(v \mid \mathbf{x}) dv\right)\right) - \mathbb{1}_{\{T_m < u\}} \left(1 - \exp\left(-\int_{T_m}^u \alpha_{j\bullet}(v \mid \mathbf{x}) dv\right)\right)\right] = \\ &\mathbb{E}\left[\mathbb{1}_{\{T_m < r\}} \left(1 - \mathbb{1}_{\{T_m < u\}} + \exp\left(-\int_{T_m}^u \alpha_{j\bullet}(v \mid \mathbf{x}) dv\right) \left(\mathbb{1}_{\{T_m < u\}} - \exp\left(-\int_u^r \alpha_{j\bullet}(v \mid \mathbf{x}) dv\right)\right)\right)\right].\end{aligned}$$

Multiplying the inner expression by $1 = \mathbb{1}_{\{T_m < u\}} + \mathbb{1}_{\{T_m \geq u\}}$ and using that $u < r$, we get that the above equals

$$\begin{aligned}&\mathbb{E}\left[\mathbb{1}_{\{T_m < u\}} \exp\left(-\int_{T_m}^u \alpha_{j\bullet}(v \mid \mathbf{x}) dv\right) \left(1 - \exp\left(-\int_u^r \alpha_{j\bullet}(v \mid \mathbf{x}) dv\right)\right)\right] \\ &+\mathbb{E}\left[\mathbb{1}_{\{u \leq T_m < r\}} \left(1 - \exp\left(-\int_{T_m}^r \alpha_{j\bullet}(v \mid \mathbf{x}) dv\right)\right)\right].\end{aligned}$$

The second term is non-negative and the first term is strictly positive for all $m < M$ and $u < t$ (with M defined as in the proof of the previous lemma) since $\mathbb{P}(T_m < u) > 0$ as shown earlier and the fact that $\alpha_{j\bullet}(\cdot \mid \mathbf{x})$ is strictly positive on $[0, t]$. We have thus shown that for at least one $m \in \mathbb{N}_0$,

$$\mathbb{P}(u \leq T_{m+1} \leq r \mid \mathbf{X} = \mathbf{x}, Y_m = j) > 0$$

for $r < t$. This proves the result. ■

With more or less all the tedious calculations out of the way, the proof of Theorem 7.2 proceeds similarly as in the survival setup.

Proof of Theorem 7.2. Consider the iid data $(\mathbf{X}^i, (Z_t)_{0\le t\le R^i}, \tau_{Z^i} \wedge R^i)_{i=1}^n$ and introduce the indicators

$$\delta_j^i = \begin{cases} 1, & \text{if a jump from } j \text{ occurs for } Z^i \text{ in } [0, R^i] \\ 0, & \text{otherwise} \end{cases}.$$

As usual, we let δ_j denote a generic δ_j^i. We claim that for any $A \neq \emptyset$ and j non-absorbing, $\mathbb{P}(\mathbf{X} \in A, \delta_j = 1) > 0$. To see why, note that

$$\mathbb{P}(\mathbf{X} \in A, \delta_j = 1) = \sum_{\mathbf{x}\in A} \mathbb{P}(\delta_j = 1 \mid \mathbf{X} = \mathbf{x})\mathbb{P}(\mathbf{X} = \mathbf{x}), \tag{17}$$

and that $\mathbb{P}(\mathbf{X} = \mathbf{x}) > 0$ for every $\mathbf{x} \in \mathcal{X}$. Let $((T_m)_{m\ge1}, (Y_m)_{m\ge1})$ be the MPP representation of Z. Using this representation, $\delta_j = 1$ if and only if $Y_m = j$ and $T_{m+1} \le R$ for some $m \in \mathbb{N}_0$. Thus,

$$\mathbb{P}(\delta_j = 1 \mid \mathbf{X} = \mathbf{x}) = \mathbb{E}\left[\mathbb{1}_{\bigcup_{n\in\mathbb{N}_0}\{Y_m=j, T_{m+1}\le R\}} \mid \mathbf{X} = \mathbf{x}\right].$$

From Lemma 7.3, $\mathbb{P}(Y_m = j, T_{m+1} \le R \mid \mathbf{X} = \mathbf{x}) > 0$ for at least one $m \in \mathbb{N}_0$, and so $\mathbb{P}(\mathbf{X} \in A, \delta_j = 1) > 0$ by the decomposition (17). Using the law of large numbers,

$$\frac{1}{n}\sum_{i=1}^n \mathbb{1}_{\{\mathbf{X}^i\in A, \delta_j^i=1\}} \xrightarrow{\mathbb{P}\text{-a.s.}} \mathbb{P}(\mathbf{X} \in A, \delta_j = 1) > 0 \quad \text{as} \quad n \to \infty$$

and so

$$\sum_{i=1}^n \mathbb{1}_{\{\mathbf{X}^i\in A, \delta_j^i=1\}} \xrightarrow{\mathbb{P}\text{-a.s.}} \infty \quad \text{as} \quad n \to \infty$$

from which we may conclude that for any $k \in \mathbb{N}$,

$$\mathbb{1}_{\{\sum_{i=1}^n \mathbb{1}_{\{\mathbf{X}^i\in A, \delta_j^i=1\}} \ge k\}} \xrightarrow{\mathbb{P}\text{-a.s.}} 1 \quad \text{as} \quad n \to \infty. \tag{18}$$

The tree is grown to full size with the restriction that a terminal node should have at least $d_{0j} > 0$ unique jumps from any non-absorbing state j. Using (18), we see that as n grows, we will with probability one have sufficiently many observations in a node to obtain a split such that both daughter nodes have at least d_{0j} unique jumps out of state j. We conclude that for large enough n, any terminal node h may be identified with exactly one $\mathbf{x}$-value, and like before, we write $h = \mathbf{x}$ when h is identified with $\mathbf{x}$. We have just argued that $\mathbb{1}_{\{\mathbf{X}=\mathbf{x}=h\}}$ and $\mathbb{1}_{\{\mathbf{x}\in h\}}$ are eventually equal as n grows to infinity, from which it follows that

$$\widehat{p}(s, u \mid \mathbf{X}) = \sum_{\mathbf{x}\in\mathcal{X}} \mathbb{1}_{\{\mathbf{X}=\mathbf{x}=h\}}\widehat{p}_h(s, u) + o_{\mathbb{P}}(1)$$

uniformly in $u \in [s, t]$ (note that $\widehat{p}$ is bounded in the matrix norm since each entry is bounded by one). We want to be able to apply Theorem 6.19 to show that for $h = \mathbf{x}$ and $t \in (0, \tau(\mathbf{x}) \wedge \tau_R)$,

$$\sup_{u\in[s,t]} |\widehat{p}_h(s, u) - p(s, u \mid \mathbf{x})| \xrightarrow{\mathbb{P}} 0 \quad \text{as} \quad n \to \infty.$$

Hence we have to show that

$$\int_s^t \frac{J_j^{(n)}(u \mid \mathbf{x})}{Y_j^{(n)}(u \mid \mathbf{x})} \alpha_{jk}(u \mid \mathbf{x}) du \xrightarrow{\mathbb{P}} 0 \tag{i}$$

and

$$\int_s^t (1 - J_j^{(n)}(u \mid \mathbf{x})) \alpha_{jk}(u \mid \mathbf{x}) du \xrightarrow{\mathbb{P}} 0 \tag{ii}$$

as $n \to \infty$ for any $\mathbf{x}$ and $j \neq k$. Note that if j is absorbing, both (i) and (ii) are trivially satisfied, so let j be non-absorbing. By copying the procedure in the proof of Theorem 5.4, it suffices to show that

$$\inf_{u \in [s,t]} Y_j^{(n)}(u \mid \mathbf{x}) \xrightarrow{\mathbb{P}\text{-a.s.}} \infty \quad \text{as} \quad n \to \infty \tag{19}$$

where $Y_j^{(n)}(u \mid \mathbf{x}) = \sum_{i=1}^n \mathbb{1}_{\{Z_{u-}^i = j, \mathbf{X}^i = \mathbf{x}\}}$ is the number of individuals at risk of making a jump from state j just before time u with feature $\mathbf{x}$. Using the strong law of large numbers,

$$\frac{1}{n} Y_j^{(n)}(u \mid \mathbf{x}) \xrightarrow{\mathbb{P}\text{-a.s.}} \mathbb{P}(Z_{u-} = j, \mathbf{X} = \mathbf{x}). \tag{20}$$

As $\mathbb{P}(Z_{u-} = j, \mathbf{X} = \mathbf{x}) = \mathbb{P}(Z_{u-} = j \mid \mathbf{X} = \mathbf{x}) \mathbb{P}(\mathbf{X} = \mathbf{x})$, $\mathbb{P}(\mathbf{X} = \mathbf{x}) > 0$ by assumption and $\mathbb{P}(Z_{u-} = j \mid \mathbf{X} = \mathbf{x}) > 0$ by Lemma 7.4, we have

$$Y_j^{(n)}(u \mid \mathbf{x}) \xrightarrow{\mathbb{P}\text{-a.s.}} \infty \quad \text{as} \quad n \to \infty$$

and hence also (20) is true. This establishes (19) and thus

$$\sup_{u \in [s,t]} |\widehat{p}(s, u \mid \mathbf{x}) - p(s, u \mid \mathbf{x})| \xrightarrow{\mathbb{P}} 0 \quad \text{as} \quad n \to \infty.$$

If $t \in (0, \tau \wedge \tau_R)$, we have

$$\begin{aligned}
\sup_{u \in [s,t]} \int_{\mathcal{X}} |\widehat{p}(s, t \mid \mathbf{x}) - p(s, t \mid \mathbf{x})| \mathbb{P}(\mathbf{X} \in d\mathbf{x}) &\leq \int_{\mathcal{X}} \sup_{u \in [s,t]} |\widehat{p}(s, t \mid \mathbf{x}) - p(s, t \mid \mathbf{x})| \mathbb{P}(\mathbf{X} \in d\mathbf{x}) \\
&= \sum_{h = \mathbf{x}} \mathbb{P}(\mathbf{X} = \mathbf{x}) \left(\sup_{u \in [s,t]} |\widehat{p}_h(s, u) - p(s, u \mid \mathbf{x})| \right) \\
&\quad + o_{\mathbb{P}}(1).
\end{aligned}$$

The sum is finite and each term converges to zero in probability, and so the proof is complete. ■

8 Discussion and further work

> "I'm happy to put this all behind us and get back to work. After all, we've got a lot to do, and only sixty more years to do it. More or less. I don't have the actuarial tables in front of me."
> - GLaDOS (*Portal 2*)

In this section, we discuss future aspects of the project. There is still a lot to be done of both a theoretical and practical nature.

8.1 Theoretical work

The theoretical endgoal is to establish consistency of the AaJo forest. We have already established consistency for a single tree when the underlying model is Markov. It is of great interest to extend the result to a model which is not necessarily Markov. This likely requires different regularity conditions on the behaviour of Z. We also need to extend the consistency from a single tree to a whole forest by proving consistency of a bootstrap tree. For the survival forest, the fundamental result is Lemma 3 from Lo and Singh [45]. All results in that article relies on a specific form of the Kaplan–Meyer estimator which is not directly transferable to the multi-state setup. If one wants to follow their approach, the results of the article need to be extended or modified. Since we are (for now) only interested in consistency and not a rate of convergence, it is plausible that a simpler approach exists which only yields convergence and not a specific rate. The role of the Markov assumption in the question of consistency for a bootstrap tree is still unclear and needs to be investigated.

All consistency results rely on the assumption of discrete feature spaces. It would be interesting to investigate consistency results based on less restrictive or different assumptions. We furthermore conjecture that all results hold without the intensity assumption. To verify this, we need to restate and possibly reprove some of the results in this thesis, possibly using different methods.

8.2 Practical work

The practical work for the AaJo forest lies in the following aspects.

1. **Splitting rules**: We need to develop splitting rules for the AaJo forest, preferably more than one. A possible solution may lie in simply extending those for the survival forest. A possibility is to use survival splitting rules for every possible transition and then weigh the split value for every transition with the total number of transitions or some user-specified weight. The goodness of the splitting rule will depend highly on the application.

2. **Evaluation metric**: It is at the moment unclear what evaluation metric one should use for the AaJo forest. A possibility is to extend Harrell's C-index to multi-models by considering each transition with proper weights. There is, however, an argument to be made that the C-index is not necessarily the right

choice in an insurance context as it does not take the absolute size of the fitted cumulative hazard into account. If the AaJo forest is to be adopted by the life insurance industry, more work needs to be done in this regard.

3. **Estimation of the initial distribution** $p(0 \mid \mathbf{x})$: While in some applications (such as life insurance), the initial distribution $p(0 \mid \mathbf{x})$ may be fully specified, we still need to consider how to estimate it when it is not. One possibility is to simply use a classification random forest, treating each state as a label.

4. **An optimised implementation**: When we have come up with sensible splitting rules, it is absolutely essential to write an optimised implementation of the AaJo forest. A possibility is to extend an existing library, but down the line it makes sense to implement an R package from scratch.

5. **Testing**: With an implementation at hand, thorough testing needs to be done. It makes sense to compare the predictive power of the AaJo forest with state of the art methods used in biostatistics and insurance. This will also help in determining which splitting rules work best in which contexts. Some splitting rules may perfom better with high or low degrees of censoring for example.

A final consideration (relevant from both a theoretical and practical perspective) is the question of incorporating left-truncation. Left-truncation is an essential problem in life insurance and is therefore a highly relevant problem to work on. It is an open question to what extend left-truncation affects both consistency results and the performance of the algorithm in practice.

References

[1] Algebra of graphics, 2024. URL https://aog.makie.org/stable/.

[2] Tidyverse, 2024. URL https://www.tidyverse.org/.

[3] O. O. Aalen, Ø. Borgan, and H. K. Gjessing. *Survival and Event History Analysis*. Statistics for Biology and Health. Springer New York, NY, 1 edition, 2008. ISBN 978-0-387-68560-1. doi: https://doi.org/10.1007/978-0-387-68560-1.

[4] P. K. Andersen, Ø. Borgan, R. D. Gill, and N. Keiding. *Statistical Models Based on Counting Processes*. Springer Series in Statistics. Springer-Verlag New York, 1993. ISBN 0-387-97872-0.

[5] S. Asmussen and M. Steffensen. *Risk and Insurance*. Probability Theory and Stochastic Modelling. Springer Cham, 1 edition, 2020. ISBN 978-3-030-35175-5. doi: https://doi.org/10.1007/978-3-030-35176-2.

[6] M. Bladt and C. Furrer. Conditional Aalen–Johansen Estimation. 2024. URL http://arxiv.org/pdf/2303.02119.

[7] B. Bolker, S. W. Park, J. Vonesh, J. Wilson, R. Schmitt, S. Holbrook, J. D. Thomson, and R. S. Duncan. emdbook: Support Functions and Data for "Ecological Models and Data", 2023. URL https://cran.r-project.org/package=emdbook.

[8] L. Breiman. Bagging Predictors. *Machine Learning*, 24:123 – 140, 1996. doi: 10.1023/A:1018054314350. URL https://link.springer.com/article/10.1023/A:1018054314350.

[9] L. Breiman. Out-of-bag Estimation. 1996. URL https://api.semanticscholar.org/CorpusID:17166335.

[10] L. Breiman. Random Forests. *Machine Learning*, 45:5 – 32, 2001. doi: 10.1023/A:1010933404324. URL https://link.springer.com/article/10.1023/A:1010933404324.

[11] L. Breiman, J. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Chapman and Hall/CRC, 1 edition, 1984. ISBN 9781315139470.

[12] G. Broström. *eha: Event History Analysis*, 2024. URL https://cran.r-project.org/package=eha. R package version 2.11.5.

[13] F. Castellares, S. C. Patricio, and A. Lemonte. On the Gompertz–Makeham law: A useful mortality model to deal with human mortality. *Brazilian Journal of Probability and Statistics*, 3(36):613–639, 2022.

[14] S. N. Cohen and R. J. Elliott. *Stochastic Calculus and Applications*. Probability and Its Applications. Birkhäuser New York, NY, 2 edition, 2015. ISBN 978-1-4939-2866-8. doi: https://doi.org/10.1007/978-1-4939-2867-5.

[15] R. M. Corless, G. H. Gonnet, D. E. G. Hare, D. J. Jeffrey, and D. E. Knuth. On the Lambert W Function. *Advances in Computational Mathematics*, 5: 329 – 359, 1996. doi: https://doi.org/10.1007/BF02124750. URL https://link.springer.com/article/10.1007/BF02124750.

[16] D. R. Cox. Regression Models and Life-Tables. *Journal of the Royal Statistical Society. Series B (Methodological)*, 34(2):187–220, 1972. URL http://www.jstor.org/stable/2985181.

[17] D. R. Cox and D. Oakes. *Analysis of Survival Data*. Chapman and Hall/CRC New York, 1 edition, 1984. ISBN 9781315137438.

[18] B. Efron. Bootstrap Methods: Another Look at the Jackknife. *The Annals of Statistics*, 7(1):1 – 26, 1979. doi: 10.1214/aos/1176344552. URL https://doi.org/10.1214/aos/1176344552.

[19] C. Furrer. *Introduction to actuarial mathematics*. Department of Mathematical Sciences University of Copenhagen, 2 edition, 2024. ISBN 978-87-7125-287-3. URL https://noter.math.ku.dk/matematik.htm.

[20] K. S. T. Gad and J. W. Nielsen. Reserves and cash flows under stochastic retirement. *Scandinavian Actuarial Journal*, 2016(10):876–904, 2016. doi: 10.1080/03461238.2015.1028432. URL https://doi.org/10.1080/03461238.2015.1028432.

[21] E. A. Gehan. A generalized Wilcoxon test for comparing arbitrarily singly-censored samples. *Biometrika*, 52:203–23, 1965. URL https://api.semanticscholar.org/CorpusID:8385578.

[22] R. D. Gill and S. Johansen. A Survey of Product-Integration with a View Toward Application in Survival Analysis. *The Annals of Statistics*, 18(4): 1501–1555, 1990. ISSN 00905364. URL http://www.jstor.org/stable/2241874.

[23] B. 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. *Philosophical Transactions of the Royal Society of London*, 115:513–583, 1825. URL http://www.jstor.org/stable/107756.

[24] E. Hansen. *Stochastic Processes*, volume 2. Institut for Matematiske Fag Københavns Universitet, 5 edition. ISBN 978-87-71253-00-9.

[25] F. E. Harrell, R. M. Califf, D. B. Pryor, K. L. Lee, and R. A. Rosati. Evaluating the yield of medical tests. *JAMA*, 247(18):2543–2546, 1982. doi: 10.1001/JAMA.1982.03320430047030.

[26] D. Harrison and D. L Rubinfeld. Hedonic housing prices and the demand for clean air. *Journal of Environmental Economics and Management*, 5(1):81–102, 1978. doi: https://doi.org/10.1016/0095-0696(78)

90006-2. URL https://www.sciencedirect.com/science/article/pii/0095069678900062.

[27] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York, NY, 2 edition, 2009. ISBN 978-0-387-84857-0. doi: https://doi.org/10.1007/978-0-387-84858-7.

[28] T. Hothorn and B. Lausen. On the exact distribution of maximally selected rank statistics. *Computational Statistics & Data Analysis*, 43(2):121–137, 2003. ISSN 0167-9473. doi: https://doi.org/10.1016/S0167-9473(02)00225-6. URL https://www.sciencedirect.com/science/article/pii/S0167947302002256.

[29] E. Hsich, E. Z. Gorodeski, E. H. Blackstone, H. Ishwaran, and M. S. Lauer. Identifying Important Risk Factors for Survival in Patients With Systolic Heart Failure Using Random Survival Forests. *Circulation: Cardiovascular Quality and Outcomes*, 4(1):39–45, 2011. doi: 10.1161/CIRCOUTCOMES.110.939371. URL https://www.ahajournals.org/doi/abs/10.1161/CIRCOUTCOMES.110.939371.

[30] H. Ishwaran and U. B. Kogalur. Random Survival Forests for R. *Rnews*, pages 25 – 31, 2007. URL https://ishwaran.org/papers/randomSurvivalForests.pdf.

[31] H. Ishwaran and U. B. Kogalur. Consistency of random survival forests. *Statistics & Probability Letters*, 80(13):1056–1064, 2010. ISSN 0167-7152. doi: 10.1016/j.spl.2010.02.020. URL https://www.sciencedirect.com/science/article/pii/S0167715210000672.

[32] H. Ishwaran, U. B. Kogalur, E. H. Blackstone, and M. S. Lauer. Random survival forests. *The Annals of Applied Statistics*, 2(3):841 – 860, 2008. doi: 10.1214/08-AOAS169. URL https://doi.org/10.1214/08-AOAS169.

[33] H. Ishwaran, M. Lu, and U. B. Kogalur. Fast Unified Random Forests with randomForestSRC. 2023. URL https://www.randomforestsrc.org/articles/getstarted.html.

[34] M. Jacobsen. *Statistical Analysis of Counting Processes*. Lecture Notes in Statistics. Springer New York, NY, 1 edition, 1982. ISBN 978-0-387-90769-7. doi: https://doi.org/10.1007/978-1-4684-6275-3.

[35] M. Jacobsen. *Point Process Theory and Applications*. Probability and Its Applications. Birkhäuser Boston, MA, 1 edition, 2005. ISBN 978-0-8176-4215-0. doi: https://doi.org/10.1007/0-8176-4463-6.

[36] S. Janitza and R. Hornung. On the overestimation of random forest's out-of-bag error. 2018. doi: 10.1371/journal.pone.0201904. URL https://pubmed.ncbi.nlm.nih.gov/30080866/.

[37] P. Jodrá. A closed-form expression for the quantile function of the Gompertz–Makeham distribution. *Mathematics and Computers in Simulation*, 79(10):3069–3075, 2009. doi: https://doi.org/10.1016/j.matcom.2009.02.002. URL https://www.sciencedirect.com/science/article/pii/S0378475409000445.

[38] J. D. Kalbfleisch and R. L. Prentice. *The Statistical Analysis of Failure Time Data*. Wiley Series in Probability and Statistics. John Wiley & Sons, 2002. ISBN 9780471363576. doi: 10.1002/9781118032985.

[39] J. D. Kalbfleisch and D. E. Schaubel. Fifty years of the cox model. *Annual Review of Statistics and Its Application*, 10(Volume 10, 2023):1–23, 2023. doi: https://doi.org/10.1146/annurev-statistics-033021-014043. URL https://www.annualreviews.org/content/journals/10.1146/annurev-statistics-033021-014043.

[40] O. Kallenberg. *Foundations of Modern Probability*. Probability Theory and Stochastic Modelling. Springer Cham, 3 edition, 2021. ISBN 978-3-030-61871-1. doi: https://link.springer.com/book/10.1007/978-3-030-61871-1.

[41] A. Kassambara. ggpubr: 'ggplot2' Based Publication Ready Plots, 2023. URL https://cran.r-project.org/web/packages/ggpubr/index.html.

[42] M. LeBlanc and J. Crowley. Survival Trees by Goodness of Split. *Journal of the American Statistical Association*, 88(422):457–467, 1993. ISSN 01621459, 1537274X. URL http://www.jstor.org/stable/2290325.

[43] F. Leisch, E. Dimitriadou, and K. Hornik. mlbench: Machine Learning Benchmark Problems, 2024. URL https://cran.r-project.org/web/packages/mlbench/.html.

[44] E. Lenglart. Relation de domination entre deux processus. *Annales de l'institut Henri Poincaré*, 13(2):171 – 179, 1977. URL http://www.numdam.org/item/AIHPB_1977__13_2_171_0/.

[45] S. Lo and K. Singh. The Product-Limit Estimator and the Bootstrap: Some Asymptotic Representations. *Probability Theory and Related Fields*, 71:455 – 465, 1986. doi: 10.1007/BF01000216. URL https://doi.org/10.1007/BF01000216.

[46] E. Longato, M. Vettoretti, and B. Di Camillo. A practical perspective on the concordance index for the evaluation and selection of prognostic time-to-event models. *Journal of Biomedical Informatics*, 108:103496, 2020. doi: https://doi.org/10.1016/j.jbi.2020.103496. URL https://www.sciencedirect.com/science/article/pii/S1532046420301246.

[47] W. M. Makeham. On the Law of Mortality and the Construction of Annuity Tables. *The Assurance Magazine, and Journal of the Institute of Actuaries*, 8 (6):301–310, 1860. URL http://www.jstor.org/stable/41134925.

[48] W. M. Makeham. On the Law of Mortality. *Journal of the Institute of Actuaries (1866-1867)*, 13(6):325–358, 1867. URL http://www.jstor.org/stable/41134517.

[49] W. M. Makeham. On the Further Development of Gompertz's Law (Concluded). *Journal of the Institute of Actuaries (1886-1994)*, 28(4):316–332, 1890. URL http://www.jstor.org/stable/41135946.

[50] M. Schmid, M. N. Wright, and A. Ziegler. On the use of Harrell's C for clinical risk prediction via random survival forests. *Expert Systems with Applications*, 63:450–459, 2016. doi: https://doi.org/10.1016/j.eswa.2016.07.018. URL https://www.sciencedirect.com/science/article/pii/S0957417416303633.

[51] M. R. Segal. Regression Trees for Censored Data. *Biometrics*, 44(1):35–47, 1988. ISSN 0006341X, 15410420. URL http://www.jstor.org/stable/2531894.

[52] T. M. Therneau, T. Lumley, A. Elizabeth, and C. Cynthia. survival: Survival Analysis, 2024. URL https://cran.r-project.org/web/packages/survival/index.html.

[53] V. Volterra. Sulle equazioni differenziali lineari. 1887. URL https://www.rcin.org.pl/impan/dlibra/publication/181806/edition/147683#info.

[54] M. N. Wright and A. Ziegler. ranger: A Fast Implementation of Random Forests for High Dimensional Data in C++ and R. *Journal of Statistical Software*, 77(1): 1 – 17, 2017. doi: 10.18637/jss.v077.i01. URL https://www.jstatsoft.org/index.php/jss/article/view/v077i01.

A The `JuliaExtendableTrees` library

In this section, we provide an overview of the `JuliaExtendableTrees` library, its structure and functionalities. We also highlight certain implementations.

A.1 Overview of the library

`JuliaExtendableTrees` is a library for performing prediction using decision trees and random forests in Julia. The library supports classification, regression and survival trees. The library is composed of eight .jl files with seven of them structured as in the following figure.

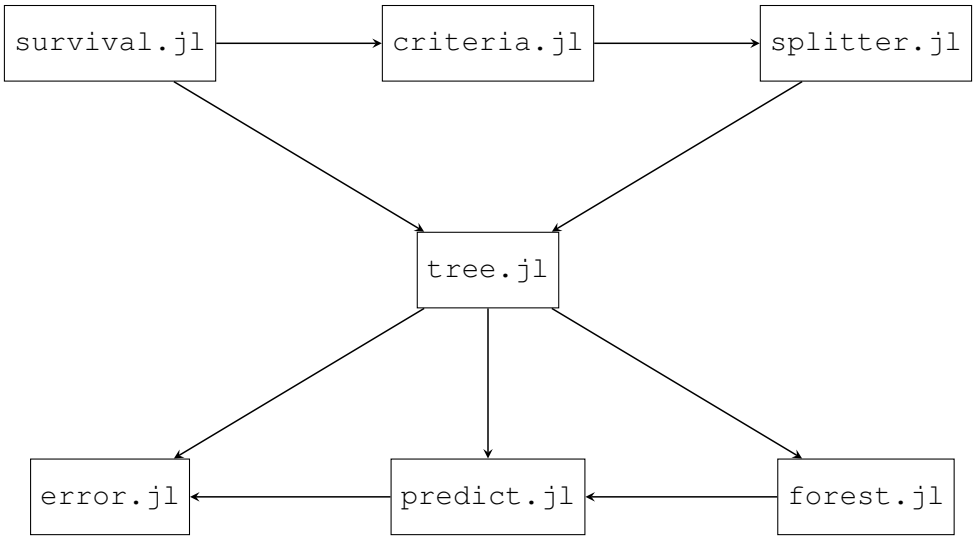

Figure 21: An illustration of the hierarchy of the files in the `JuliaExtendableTrees` library. An arrow $A \rightarrow B$ means that file B depends on file A.

The eighth file, `JuliaExtendableTrees.jl`, depends on all other files and is the only file one needs to include to use the library. It also contains wrapper functions for growing trees and forests as well as functions for printing useful output to the terminal. An overview of the contents of the remaining files are below.

- `criteria.jl`: All splitting rule functions. The naming convention is that any function to be used by `splitter.jl` must have a name starting with `L_` e.g. `L_log_rank` or `L_squared_error`. The other functions in the file are helper functions. The file currently contains two splitting rules for classification (Gini coefficient and entropy), two for regression (squared error, absolute error) and five for survival (log-rank, conservation of events splitting, log-rank score, approximate log-rank and C-index splitting).
- `splitter.jl`: Contains functions for finding best splits for each type of tree. The best split procedure is divided into two parts. The first is a function to

find the best split for a given feature (`best_feature_split` for classification and regression and `best_feature_split_Survival` for survival). The second is a function for finding the best overall split. These are called `best_split_<Type>`.

- `tree.jl`: Functions for growing each type of tree. Contains `structs` for nodes in classification, regression and survival as well as a tree struct.
- `forest.jl`: Contains a forest struct and a function for bootstrapping data that also saves OOB indices. Also contains functions for growing each type of forest.
- `predict.jl`: Contains functions for predicting from trees and all types of forests. A general function for computing which indices are OOB for a given observation is provided. Functions to compute all OOB predictions are also provided.
- `error.jl`: Contains functions to compute errors for all types of trees and forests. For classification, misclassification error and R^2 error is provided. For regression, mean squared error and R^2 error is provided, while for survival, the error is given by $1 - C$ with C denoting Harrell's C-index. All error functions contain the option to only use OOB observations in the computation, while for survival, a list of predicted values (Nelson–Aalen estimators) may also be provided. If such a list is not provided, predictions are computed manually. All types of forests have functions of the form `OOB_error_<type>` which computes an approximation to the OOB error using all OOB observations which does not take ties between features into account at the benefit of substantial improvements in speed.
- `survival.jl`: Helper functions for everything survival related. Contains functions for computing the Nelson–Aalen and Kaplan–Meyer estimators as well as a wide range of helper functions to compute quantities used by survival splitting rules.

A.2 Using the library

Here we showcase some basic usage of the library. We start simple by growing a survival decision tree on the `pbc` dataset. The first step is to load the library, read in the data and extracting the response and the features.

```
include("JuliaExtendableTrees.jl")
df = CSV.read("pbc.csv", DataFrame)
y = Matrix{Float64}(df[:, [:days, :status]])
X = Matrix{Float64}(df[:, Not([:days, :status])])
```

The following code grows a single decision tree using the `grow_tree` function. This function requires four arguments and has several optional arguments. One needs to

supply the features, the response, the type and the splitting rule. We grow a survival tree with the log-rank splitting rule as follows.

```
tree = grow_tree(X, y, "Survival", L_log_rank)
```

For a tree, one can further specify the minimum node size, the maximum depth of the tree, the number of split points used in a split and the number of features used at each split. If none of these are set manually, default values are used. Below is code for growing a tree with each hyperparameter specified manually (in this case to the default values).

```
tree = grow_tree(X, y, "Survival", L_log_rank; min_node_size = 15, max_depth = 0,
                 n_split = 0, n_features = Int(round(sqrt(size(X, 2)))))
```

One can print information about the fitted tree to the terminal using the `print_tree` function. This function returns a vector of strings with information about each node plus some information about the tree itself. The code

```
print_tree(tree)
```

yields the following output.

```
33-element Vector{String}:
 "Type of tree: Survival"
 "List of nodes:"
 "Depth: 1 Feature: 8 Threshold: 6.45"
 "Depth: 2 Feature: 11 Threshold: 109.0"
 "Depth: 3 Feature: 10 Threshold: 3.3049999999999997"
 ⋮
 "Depth: 3 Value: [326.0 0.058823" … 351 bytes … "0758] #Observations: 34 (Leaf)"
 "Depth: 3 Value: [41.0 0.05; 51." … 404 bytes … "3485] #Observations: 40 (Leaf)"
 "Number of nodes: 29"
 "Number of terminal nodes: 15"
```

For every node, the depth is printed. If the node is not terminal, the feature index and the threshold value is provided. If the node is terminal, the value (in this case the Nelson–Aalen estimator) is printed along with the number of observations. We now turn to prediction. The following code predicts the Nelson–Aalen estimator based on the first observation in the training data.

```
predict(tree, X[1, :])
```

This yields the output

```
17×2 Matrix{Float64}:
   41.0  0.05
   51.0  0.102632
   71.0  0.158187
   77.0  0.217011
  131.0  0.279511
    ⋮
  890.0  1.14774
  930.0  1.31441
  943.0  1.51441
 2540.0  2.51441
```

that is, a matrix with the first column the jump times and the second column the corresponding value of the cumulative hazard function. Computing the predicted value of a whole dataset is easy. To illustrate, if one wants to compute all predictions of the training data, one simply writes

```
predict(tree, X)
```

Computing an error estimate (the C-index error) is done in the following way. The `error_Survival` function needs a tree or forest as its first input, a test dataset as its second input and a test response as its third output. Running the code

```
error_Survival(tree, X, y)
```

yields the number 0.19019219751028704 for this particular run. Variations should be expected when running the code multiple times since single decision trees have high variance. We now turn to forests. Growing a forest is very similar to a tree, although more hyperparameters are available. If one wants to use default parameters and the log-rank splitting rule, one can simply write the following.

```
forest = grow_forest(X, y, "Survival", L_log_rank)
```

To obtain useful information on this forest, use the `print_forest` function.

```
print_forest(forest)
```

For this particular run, we get the following output to the terminal.

```
Type of forest: Survival
Number of observations: 276
Number of features available: 17
Number of trees: 500
Maximum depth of a tree: No maximum depth
Minimum size of a node: 5
Number of splitting points used: 10
Number of features used in a split: 4
Average number of terminal nodes: 26.866
```

The average number of terminal nodes is a nice bit of information for the model which says something about the tree complexity. For a forest, there are two ways to predict. Consider the first feature in the dataset. The classical ensemble is computed with

```
predict(forest, X[1, :])
```

which yields

```
109×2 Matrix{Float64}:
   41.0  0.0556207
   51.0  0.118365
   71.0  0.144286
   77.0  0.180587
  110.0  0.199799
    ⋮
 3839.0  2.27065
 3853.0  2.27065
 4079.0  2.27065
 4191.0  2.27065
```

An alternative is to only average over the predictions where the first feature is OOB. This is done as follows.

```
predict(forest, X[1, :]; OOB = true)
```

yielding the slightly different result

```
109×2 Matrix{Float64}:
   41.0  0.0599531
   51.0  0.127831
   71.0  0.160415
   77.0  0.202532
  110.0  0.22694
    ⋮
```

```
3839.0  2.24601
3853.0  2.24601
4079.0  2.24601
4191.0  2.24601
```

The exact same syntax is used for predicting for more values. Simply replace `X[1, :]` with a dataset. One should be aware of the options that exist for computing the error for a forest. One choice is to simply use the `error` function for that type, in this case `error_Survival`. Say we want to compute the training error by supplying all the training data as test data. This is done by

```
error_Survival(forest, X, y)
```

which yields 0.0902130319287463. If we want the OOB error, we simply do

```
error_Survival(forest, X, y; OOB = true)
```

which gives 0.17089431741236527. Since we are supplying the whole dataset, an alternative is to use the (often much faster) function `OOB_error_Survival`

```
OOB_error_Survival(forest)
```

which yields the exact same result since there are no ties in the feature data.

We stress that the library currently has the limitation that it cannot handle categorical data. The library converts the matrix of features into a matrix of floats. This is not an issue for 0/1 data, but it means that the library cannot handle features that are not representable as floats, such as strings. For a feature with two levels, "yes" or "no", say, this is not an issue, since such data can simply be converted to 0/1 data. But for categorical data with more than two levels, the library does not group the data when determining the best split and thus some optimal splits may be lost.

A.3 Essential functions and their implementations

A.3.1 Determining the best split (survival)

The splitter functions are very similar for the different types of trees. We choose to present the code for survival trees. The following helper function splits a feature matrix into two sets of indices with `left_ind` being the indices for the data in the left node and analogously for `right_ind`.

```
function split_dataset(X::Matrix{Float64}, feature::Int, threshold::Float64)
    left_ind = X[:, feature] .<= threshold
    right_ind = .!left_ind
    return left_ind, right_ind
end
```

The following helper function determines the best split given a feature in the form of an index (integer) and a sorted list of threshold values with no duplicates.

```
function best_feature_split_Survival(X::Matrix{Float64}, y::Matrix{Float64},
L::Function, feature::Int, min_node_size::Int, thresholds::Array{Float64})
    best_threshold = nothing
    best_split_val = -1
    best_left_ind = nothing
    best_right_ind = nothing

    n = length(thresholds)
    mid = Int(ceil(n/2))

    # check the first half of the splits
    for j in 1:(mid - 1)

        left_ind, right_ind = split_dataset(X, feature, thresholds[mid - j])

        # if the split creates a node with too few data points, skip the split
        # entirely and all following splits (works because thresholds is sorted)
        if sum(left_ind) < min_node_size || sum(right_ind) < min_node_size
            break
        end

        l = L(X, y, feature, thresholds[mid - j])
        if l > best_split_val
            best_threshold = (thresholds[mid - j] + thresholds[mid - j + 1])/2
            best_split_val = l
            best_left_ind = left_ind
            best_right_ind = right_ind
        end
    end

    # check the second half of the splits
    for j in mid:n

        left_ind, right_ind = split_dataset(X, feature, thresholds[j])

        # if the split creates a node with too few data points, skip the split
        # entirely and all following splits
        if sum(left_ind) < min_node_size || sum(right_ind) < min_node_size
            break
        end

        l = L(X, y, feature, thresholds[j])
        if l > best_split_val
```

```
            if j < n
                best_threshold = (thresholds[j] + thresholds[j + 1])/2
            else
                best_threshold = thresholds[j]
            end
            best_split_val = l
            best_left_ind = left_ind
            best_right_ind = right_ind
        end
    end
    # return the best split for this particular feature
    return best_left_ind, best_right_ind, best_threshold, best_split_val
end
```

Some comments are in order. Since this function is the workhorse of the library, several attempts have been made to save computations. Since the threshold values `thresholds` are sorted and contain no duplicates, one can start iterating from the middle. If an illegal split occurs (the number of observations in one of the nodes is too small), all subsequent splits away from the middle in that direction will also be illegal and can thus be skipped. When the best threshold is saved, we choose the average of the current best threshold and the next. This is to improve stability when predicting using new data. This function is applied to each feature available in a split using the following function.

```
function best_split_Survival(X::Matrix{Float64}, y::Matrix{Float64},
L::Function, min_node_size::Int, n_features::Int, n_split::Int)
    best_feature = nothing
    best_threshold = nothing
    best_split_val = -1
    best_left_ind = nothing
    best_right_ind = nothing

    # randomly select n_features features to split on
    features = sample(1:size(X, 2), n_features, replace = false)
    for i in features
        # select the unique values for the given feature
        thresholds = unique(X[:, i])
        # only use n_split different threshold values (all are used if n_split = 0
        # or n_split is larger than the number of unique values)
        if n_split != 0 && n_split < length(thresholds)
            thresholds = sort(sample(thresholds, n_split, replace = false))
        else
            thresholds = sort(thresholds)
        end
        best_left_ind_i, best_right_ind_i, best_threshold_i, best_split_val_i =
        best_feature_split_Survival(X, y, L, i, min_node_size, thresholds)

        if best_split_val_i > best_split_val
            best_feature = i
            best_threshold = best_threshold_i
            best_split_val = best_split_val_i
```

```
            best_left_ind = best_left_ind_i
            best_right_ind = best_right_ind_i
        end
    end

    # return the overall best split
    return best_left_ind, best_right_ind, best_feature, best_threshold,
    best_split_val
end
```

A.3.2 Some criteria functions

When computing the Gini coefficient or the entropy, it is necessary to compute proportions in the labels. This is done using dictionaries in Julia as follows.

```
function proportions(y::Array{Int})::Array{Float64}
    counts = Dict{eltype(y), Int}()
    for elem in y
        counts[elem] = get(counts, elem, 0) + 1
    end
    collect(values(counts))/length(y)
end
```

This allows us to compute the Gini coefficient easily.

```
function Gini_coefficient(y::Array{Int})::Float64
    prop = proportions(y)
    1 - sum(prop .^2)
end

function L_Gini_coefficient(X::Matrix{Float64}, y::Array{Int},
left_ind::BitArray{1}, right_ind::BitArray{1})::Float64
    y_left = y[left_ind]
    y_right = y[right_ind]
    Gini_coefficient(y) -(length(y_left) * Gini_coefficient(y_left)
    + length(y_right) * Gini_coefficient(y_right))/length(y)
end
```

The function `L_Gini_coefficient` is one of two possible choices for splitting rules for classification. As for regression, the most popular splitting rule is the squared error (called `L_squared_error`), which is computed as follows.

```
function squared_error(y::Array{Float64})::Float64
    sum(y .^2)/length(y) - (sum(y)/length(y))^2
end

function L_squared_error(X::Matrix{Float64}, y::Array{Float64},
```

```
left_ind::BitArray{1}, right_ind::BitArray{1})::Float64
    y_left = y[left_ind]
    y_right = y[right_ind]
    squared_error(y) - (length(y_left) * squared_error(y_left)
    + length(y_right) * squared_error(y_right))/length(y)
end
```

We also provide an example of a splitting rule in the survival setup. The most popular choice is the log-rank splitting rule. The split value is computed in the function `L_log_rank` given by

```
function L_log_rank(X::Matrix{Float64}, y::Matrix{Float64}, feature::Int,
threshold::Float64)::Float64
    t, Y1, d1, Y2, d2 = survival_criteria_helper(X, y, feature, threshold)
    Y = Y1 + Y2
    d = d1 + d2

    sum_num = 0
    sum_den = 0
    for i in 1:length(t)
        if Y[i] < 2 || Y1[i] < 1
            break
        end
        if d[i] > 0
            sum_num += d1[i] - Y1[i] * d[i] / Y[i]
            sum_den += d[i] * (Y1[i] / Y[i]) * (1 - Y1[i]/Y[i])
                     * (Y[i] - d[i]) / (Y[i] - 1)
        end
    end

    if sum_den != 0
        return(abs(sum_num/sqrt(sum_den)))
    else
        return(-1)
    end
end
```

The input `y` is a matrix with the first column containing the observed survival times and the second containing the censoring indicators. `survival_criteria_helper` is a helper function for computing the vectors of unique event times in the parent node t as well as the vectors of individuals at risk and deaths at the times in t in both nodes Y_1, d_1, Y_2, d_2 (see the section on random survival forests above). The function is found in the `survival.jl` file and is given by

```
function survival_criteria_helper(X::Matrix{Float64}, y::Matrix{Float64},
feature::Int, threshold::Float64)
    # select the (unique) true survival times from y
    t = sort(unique(y[y[:, 2] .== 1, :][:, 1]))
    N = length(t)
```

```
    n = size(y, 1)

    # compute the vectors of individuals at risk, Y, and the
      number of deaths, d for each node
    Y1 = Array{Int}(undef, N)
    Y2 = Array{Int}(undef, N)
    d1 = Array{Int}(undef, N)
    d2 = Array{Int}(undef, N)
    for i in 1:N
        Y1_res::Int = 0
        Y2_res::Int = 0
        d1_res::Int = 0
        d2_res::Int = 0

        for l in 1:n
            if y[l, 1] >= t[i]
                if X[l, feature] <= threshold
                    Y1_res += 1
                else
                    Y2_res += 1
                end
            end
            if y[l, 1] == t[i] && y[l, 2] == 1
                if X[l, feature] <= threshold
                    d1_res += 1
                else
                    d2_res += 1
                end
            end
        end
        Y1[i] = Y1_res
        Y2[i] = Y2_res
        d1[i] = d1_res
        d2[i] = d2_res
    end

    return(t, Y1, d1, Y2, d2)
end
```

This helper function is used in every criteria function for survival except for C-index splitting.

A.3.3 Growing a tree

Growing a tree works pretty much in the same way for all types of trees. We demonstrate for survival trees. We define structs `SurvivalNode` and `Tree` given as follows.

```
struct SurvivalNode
    feature::Union{Int, Nothing}          # index of the feature split upon
    threshold::Union{Float64, Nothing}    # the threshold for the split
    left::Union{SurvivalNode, Nothing}    # left daughter
```

```
    right::Union{SurvivalNode, Nothing}   # right daughter
    depth::Int                            # depth of the node
    num::Union{Int, Nothing}              # number of observed events in the node
    val::Union{Matrix{Float64}, Nothing}  # predicted value of the node
end

struct Tree
    root::Union{ClassificationNode, RegressionNode, SurvivalNode}   # root node
    type::String                                                    # type
    n_terminal_nodes::Int                # number of terminal nodes in the tree
end
```

The following function builds a tree recursively. It checks the two stopping conditions (maximum depth reached or the minimum node size is violated) and becomes a terminal node if any of them are fulfilled. In that case, we don't need to save a feature, threshold or any daughter nodes. We furthermore compute a value. Since we are growing a survival tree, `value_Survival` computes the Nelson–Aalen estimator based on the survival data `y` passed to the node. If the stopping criteria are not met, we compute the best split. If such a split is not found (corresponding to the best split value being -1), we again declare the node terminal. If not, we call the function again for the daughters and remember to update the depth reached `depth`. Node that we are also returning a number in addition to a `SurvivalNode`. This number is used to recursively compute the number of terminal nodes in the tree, which is why we in the case of a split return the sum of the number of terminal nodes for both daughters.

```
function tree_builder_Survival(X::Matrix{Float64}, y::Matrix{Float64}, depth::Int,
L::Function, max_depth::Int, min_node_size::Int, n_features::Int, n_split::Int)
    # a node should be declared terminal if the max_depth is reached or if the
    number of true deaths falls below min_node_size
    if depth == max_depth || size(X, 1) < 2 * min_node_size
        return(SurvivalNode(nothing, nothing, nothing, nothing, depth, length(y),
        value_Survival(y)), 1)
    end

    left_ind, right_ind, feature, threshold, split_val =
    best_split_Survival(X, y, L, min_node_size, n_features, n_split)

    # if the best split is no split, the node is also declared terminal
    if split_val < 0
        return(SurvivalNode(nothing, nothing, nothing, nothing, depth,
        length(y), value_Survival(y)), 1)
    end

    # make daughters
    left_node, n_terminal_nodes_left = tree_builder_Survival(X[left_ind, :],
    y[left_ind, :], depth + 1, L, max_depth, min_node_size, n_features, n_split)
    right_node, n_terminal_nodes_right = tree_builder_Survival(X[right_ind, :],
    y[right_ind, :], depth + 1, L, max_depth, min_node_size, n_features, n_split)
    return(SurvivalNode(feature, threshold, left_node, right_node, depth, nothing,
```

```
    nothing), n_terminal_nodes_left + n_terminal_nodes_right)
end
```

With the recursive tree-builder in place, the following function actually grows the tree.

```
function grow_tree_Survival(X::Matrix{Float64}, y::Matrix{Float64}, L::Function,
max_depth::Int, min_node_size::Int, n_features::Int, n_split::Int)
    root, n_terminal_nodes = tree_builder_Survival(X, y, 1, L, max_depth,
    min_node_size, n_features, n_split)
    return(Tree(root, "Survival", n_terminal_nodes))
end
```

A.3.4 Growing a forest

A forest is given as the following struct.

```
mutable struct Forest
    X::Matrix{Float64}                      # training data features
    y                                       # training data labels
    trees::AbstractArray{Tree}              # collection of trees
    type::String                            # type
    max_depth::Int                          # maximum depth of a tree
    min_node_size::Int                      # maximum size of a node
    n_features::Int                         # number of features used in each split
    n_split::Int                            # maximum number of thresholds
                                            # considered in each split
    bootstrap_indices::BitMatrix            # a matrix where each column is a
                                            # bitvector indicating which datapoints
                                            # are in the sample
    avr_number_terminal_nodes::Float64      # average number of terminal nodes
                                            # in a tree
end
```

All `grow_forest` functions use the following helper function for the bootstrapping procedure.

```
function bootstrap_data(n::Int, n_trees::Int, sfrac::Float64, swr::Bool)
    bootstrap_indices = falses(n, n_trees)
    sample_indices = reduce(hcat, [sample(1:n, Int(round(n * sfrac)),
    replace = swr) for _ in 1:n_trees])
    for i in 1:n_trees
        bootstrap_indices[sample_indices[:, i], i] .= 1
    end

    # return the indices in both bitvector format and int format
    # (the latter is used to optimise performance)
```

```
    return bootstrap_indices, sample_indices
end
```

The function makes a bootstrap sample `n_trees` times. It also computes a matrix where each column is a bitvector where a one in the i'th row indicates that the observation is part of that bootstrap sample. This is used to effectively approximate the OOB error later on. The following function grows a survival forest (the `grow_forest_Classification` and `grow_forest_Regression` functions are more or less identical).

```
function grow_forest_Survival(X::Matrix{Float64}, y::Matrix{Float64},
L::Function, max_depth::Int = 0, min_node_size::Int = 10,
n_features::Int = Int(round(sqrt(size(X, 2)))), n_split::Int = 10,
n_trees::Int = 500, sfrac::Float64 = 0.7, swr::Bool = false)

    # choosing n_features larger than the number of features
    # simply means no randomness
    if n_features > size(X, 2)
        n_features = size(X, 2)
    end

    trees = Array{Tree}(undef, n_trees)
    n = size(X, 1)
    bootstrap_indices, sample_indices =
    bootstrap_data(n, n_trees, sfrac, swr)
    total_terminal_nodes = 0

    # grow the trees from the bootstrap data
    @threads for i in 1:n_trees
        trees[i] = grow_tree_Survival(X[sample_indices[:, i], :],
        y[sample_indices[:, i], :], L, max_depth, min_node_size,
        n_features, n_split)
        total_terminal_nodes += trees[i].n_terminal_nodes
    end

    return(Forest(X, y, trees, "Survival", max_depth, min_node_size,
    n_features, n_split, bootstrap_indices, total_terminal_nodes/n_trees))
end
```

The `@threads` in the for loop means that Julia uses multithreading. Depending on the setup, it may require some tweaking to ensure that multithreading is actually applied. In Visual Studio Code for example, the number of threads should be set in the `settings.json` file.

A.3.5 Predicting values

Predicting the value of a tree works the same way for all types. Simply drop the observation down the tree. In code, this is done as follows.

```
function predict(tree::Tree, x::Array{Float64})
    current_node = tree.root
    while current_node.val === nothing   # not yet in a terminal node
        if x[current_node.feature] <= current_node.threshold
            current_node = current_node.left
        else
            current_node = current_node.right
        end
    end
    return current_node.val
end
```

It is then easy to extend to the case where `x` is a whole matrix of observations. When computing the OOB error for some observation in the dataset, it is necessary to determine the indices of the trees where the observation is OOB. This is done as follows.

```
function OOB_indices(forest::Forest, x::Array{Float64})
    B = length(forest.trees)

    # by default, x is not OOB
    OOB = falses(B)

    # find the indices of all rows equal to x in the original data
    x_ind = findall(row -> row == x, eachrow(forest.X))

    @threads for i in 1:B
        # if none of the indices in a bootstrap set is one, x is OOB
        if sum(forest.bootstrap_indices[:, i][x_ind]) == 0
            OOB[i] = true
        end
    end
    OOB
end
```

The following helper function computes the predicted value of `x` for every tree in the forest.

```
function predicted_labels(forest::Forest, x::Array{Float64}, OOB::Bool = false)
    if OOB
        oob = OOB_indices(forest, x)
        trees = forest.trees[oob]
    else
        trees = forest.trees
    end

    n = length(trees)
    res = Array{Any}(undef, n)
    @threads for i in 1:n
```

```
        res[i] = predict(trees[i], x)
    end

    res
end
```

With these helper functions in place, it is easy to write functions for computing the aggregated prediction over the forest in the case of classification or regression. Simply compute the median or the mean of the predicted labels. For survival, the situation is more complicated since each predicted label is a Nelson–Aalen estimator. Such a value is given as a matrix with two columns consisting of the jump times and the cumulative hazard. These need to be aggregated into one jump function over all event times in the data.

```
function predict_Survival(forest::Forest, x::Array{Float64}, t::Array{Float64},
OOB::Bool = false)
    ensemble_NA(predicted_labels(forest, x, OOB), t)
end
```

The function `ensemble_NA` computes the ensemble Nelson–Aalen estimator.

```
function ensemble_NA(predictions, t::Array{Float64})
    N = length(t)
    B = length(predictions)
    jump_values = zeros(Float64, N, B)

    # compute the matrix of jump values in terms of the event times
    for j in 1:B
        val = predictions[j]
        last_value = 0
        for i in 1:N
            k = 1
            while k <= size(val, 1) && val[k, 1] <= t[i]
                if val[k, 1] == t[i]
                    last_value = val[k, 2]
                end
                k += 1
            end
            # if no val[k, 1] match the t[i], no jumps have happened yet
            # and so the jump_value is zero
            if k != 1
                jump_values[i, j] = last_value
            end
        end
    end

    average_values = mean(jump_values, dims = 2)
    hcat(t, average_values)
end
```

All types of forests have a corresponding `OOB_predict` function which computes the OOB predictions when the whole training data is supplied. The function actually computes an approximation based on the OOB indices from the function `bootstrap_data`. The approximation lies in the fact that this function does not take ties between feature vectors into account. If no ties are present, the computed predictions are exact. Since the OOB indices are supplied instead of manually computed for every observation in the training data, the predictions are computed much faster. The code is very similar for the three types of forest. For survival forests, the code is given below.

```
function OOB_predict_Survival(forest::Forest)
    n = size(forest.X, 1)
    predictions = Array{Matrix{Float64}}(undef, n)
    OOB_ind = transpose(.!forest.bootstrap_indices)

    # save the sorted unique event times for later
    t = sort(unique(forest.y[forest.y[:, 2] .== 1, :][:, 1]))

    @threads for i in 1:n
        # choose the trees where X[i, :] is OOB
        trees = forest.trees[OOB_ind[:, i]]
        m = length(trees)

        # compute the predictions over all the trees where X[i, :] is OOB
        temp_predictions = Array{Matrix{Float64}}(undef, m)
        for j in 1:m
            temp_predictions[j] = predict(trees[j], X[i, :])
        end

        # compute the final prediction for X[i, :] as the mode
        # if X[i, :] is not OOB for any dataset, set the prediction
        # to be the mode over the whole dataset
        if isempty(temp_predictions) == false
            predictions[i] = ensemble_NA(temp_predictions, t)
        else
            predictions[i] = ensemble_NA(forest.y, t)
        end
    end
    predictions
end
```

A.3.6 Computing the error

For classification, we use the misclassification rate as error. The library contains functions for both the misclassification rate and the R^2 error. For forests, the code looks as follows.

```
function error_Classification(forest::Forest, X_test::Matrix{Float64},
y_test::Array{Int64}, OOB::Bool = false)::Float64
```

```
    1 - mean(predict(forest, X_test, OOB) .== y_test)
end
```

```
function R2_Classification(forest::Forest, X_test::Matrix{Float64},
y_test::Array{Int64}, OOB::Bool = false)::Float64
    predicted = predict(forest, X_test, OOB)
    1 - mean(predicted .== y_test)/mean()
end
```

Computing the mean squared error and the corresponding R^2 estimate for regression forests is very similar. When one wishes to compute the OOB error based on the whole dataset, one can compute an approximation based on the `OOB_predict_Classification` function as follows.

```
function OOB_error_Classification(forest::Forest)::Float64
    predicted_values = OOB_predict_Classification(forest)
    1 - mean(predicted_values .== forest.y)
end
```

```
function OOB_R2_Classification(forest::Forest)::Float64
    predicted_values = OOB_predict_Classification(forest)
    1 - (1 - mean(predicted_values .== forest.y))/
    (1 - mean(mode(forest.y) .== forest.y))
end
```

Similar OOB error functions are provided for regression and survival forests. For survival, the error is given by one minus Harrell's C-index. The function for computing the C-index based on a vector of outcomes is based directly on the description above and is given as follows.

```
function Harrell_C(outcomes::Vector{Float64}, y::Matrix{Float64})
    # initialise numerator and denominator
    Concordance = 0
    Permissible = 0
    n = length(outcomes)

    for i in 1:n
        for j in (i + 1):n
            # if T_i < T_j and delta_i = 0, the pair is not comparable
            if y[i, 1] < y[j, 1] && y[i, 2] == 0
                continue
            end
            # similarly with i and j reversed
            if y[j, 1] < y[i, 1] && y[j, 2] == 0
```

```
                continue
            end
            # if T_i = T_j and delta_i = delta_j, the pair is also
            # considered incomparable
            if y[i, 1] == y[j, 1] && y[i, 2] == y[j, 2]
                continue
            end

            # if T_i < T_j and outcomes[i] > outcomes[j], the model
            # predicts correctly (similarly for i and j reversed), so
            # add 1 to Concordance
            # if outcomes[i] = outcomes[j], the model is indecisive,
            # so add 0.5 to Concordance
            if y[i, 1] < y[j, 1] && outcomes[i] > outcomes[j]
                Concordance += 1
            elseif y[j, 1] < y[i, 1] && outcomes[j] > outcomes[i]
                Concordance += 1
            elseif outcomes[i] == outcomes[j]
                Concordance += 0.5
            end

            Permissible += 1
        end
    end

    return Concordance/Permissible
end
```

To compute the C-index for predictions on a fitted forest, one uses the following function.

```
function Harrell_C(forest::Forest, X_test::Matrix{Float64},
y_test::Matrix{Float64}; OOB::Bool = false, predicted = nothing)::Float64
    # if predictions are not supplied, compute them from scratch
    if predicted === nothing
        predicted = predict(forest, X_test, OOB)
    end

    Harrell_C(predicted, y_test)
end
```

It is then a trivial matter to compute the error for the forest. In contrast to regression and classification, it is furthermore possible to supply a vector of predicted values (i.e. a vector of Nelson–Aalen estimators) if these are already computed.

```
function error_Survival(forest::Forest, X_test::Matrix{Float64},
y_test::Matrix{Float64}; OOB::Bool = false, predicted = nothing)::Float64
    1 - Harrell_C(forest, X_test, y_test; OOB = OOB, predicted = predicted)
end
```

As for regression and classification, it is also possible to directly compute an OOB error approximation with the `OOB_predict_Survival` function.

A.4 Overall comments on the code

Before closing this section, we provide some of the considerations that went into designing this library. First and foremost, the code is very functional in nature. An alternative would be to write the library in an object-oriented fashion. The main motivation for this design choice is simplicity and for making the library easier to extend. For example, adding additional splitting rules is as easy as writing a single function in the `criteria.jl` file.

When writing a function, the type of each parameter (`::Forest`, `::Float64` etc.) as well as the output type of the function is specified in advance whenever possible, even though Julia is not a statically typed language. The primary reason for this is philosophical. Specifying the type makes it easier to understand how to apply a function, and it makes error handling easier. In some cases, specifying types also improves performance, which is the reason there are three node objects (one for each type of tree). If an object in a struct is not specified, the type has to be determined at runtime which slows performance, since the CPU handles different types via different instructions. Performance is especially critical for the node structs since these are used thousands of times when fitting a forests.

B `JuliaExtendableTrees` in the project

In this section, we provide an overview of all the testing done with the `JuliaExtendableTrees` library. We start by providing an overview of all the .jl files used, and afterwards we present some of the code used in the analyses of real data throughout the project.

B.1 Overview of test files

File name	Type	Description
`classification_nodedepth.jl`	C	Varying maximum node depth on `wine`.
`classification_nodesize.jl`	C	Varying minimum node size on `wine`.
`classification_sampling.jl`	C	Varying fraction of data sampled on `wine`.
`classification_splitter.jl`	C	Varying splitting rule on `iris`. Not included.
`regression_nfeatures.jl`	R	Varying number of features on `BostonHousing`
`regression_n_trees.jl`	R	Varying number of trees on `BostonHousing`
`rsf_testing.jl`	S	Bootstrap estimates of the C-index for four datasets
`peakVO2_tests.jl`	S	Hyperparameter tuning on `peakVO2`.

Figure 22: Table of all .jl files used for analysing data throughout the project. C: Classification, R: Regression, S: Survival.

B.2 Overview of figure files

File name	Type	Description
`mlpre_figures.jl`	C/R	All figures for section 2.
`rsf_figures.jl`	S	All figures related to `rsf_testing.jl` and `peakVO2_tests.jl`.

Figure 23: Table of all .jl files used for making figures using AlgebraOfGraphics throughout the project. C: Classification, R: Regression, S: Survival.

B.3 The regression and classification analyses in Subsection 2.2

To analyse the effect of changing `sfrac` on the `wine` dataset, the following code was run two times. One where the Gini coefficient was chosen as splitting rule (as in the code below) and one with the entropy splitting rule (simply replace `Gini_coefficient` with `Entropy` everywhere).

```
include("JuliaExtendableTrees.jl")

Random.seed!(2024)

wine = CSV.read("wine.csv", DataFrame)
y = wine[:, :quality]
X = Matrix(wine[:, Not(:quality)])
oob_errors = Array{Float64}(undef, 10)
train_errors = Array{Float64}(undef, 10)

@time begin
    for b in 1:9
        wine_forest = grow_forest(X, y, "Classification", L_Gini_coefficient;
        max_depth = 10, min_node_size = 10, sfrac = 0.1 + 0.1*(b - 1), swr = false)
        oob_errors[b] = error_Classification(wine_forest, X, y, true)
        train_errors[b] = error_Classification(wine_forest, X, y)
        println(b)
    end
    # full bootstrap
    wine_forest = grow_forest(X, y, "Classification", L_Gini_coefficient;
    max_depth = 10, min_node_size = 10, sfrac = 1.0, swr = true)
    oob_errors[10] = error_Classification(wine_forest, X, y, true)
    train_errors[10] = error_Classification(wine_forest, X, y)
end

CSV.write("wine_sampling_Gini.csv", Tables.table(hcat(train_errors, oob_errors)),
writeheader = true)
```

To generate the corresponding figure, the following code was used.

```
using CSV, DataFrames, AlgebraOfGraphics, CairoMakie

df = (x = [0.1:0.1:1; 0.1:0.1:1; 0.1:0.1:1; 0.1:0.1:1],
      y = [sampling_Gini.Training_error; sampling_Gini.OOB_error;
      sampling_Entropy.Training_error; sampling_Entropy.OOB_error],
      Error = [fill("Train", 10); fill("OOB", 10); fill("Train", 10);
      fill("OOB", 10)], l = [fill("Gini coefficient", 20); fill("Entropy", 20)])

layers = visual(Lines) + visual(Scatter) * mapping(marker = :Error)
plt = data(df) * layers * mapping(:x, :y, color = :Error, layout = :l)
fg = draw(plt, axis = (xlabel = "Fraction of data used", xticks = 0.1:0.1:1,
                       ylabel = "Error rate"),
                       figure = (; size = (800, 400),
                       title = "Error rates for varying fractions of data
                       used to fit",
                       titlealign = :center))

save("Figures/sampling_plot.png", fg, px_per_unit = 3)
```

The following code was used to analyse the effect of varying the number of features `n_features` on the `BostonHousing` dataset.

```
include("JuliaExtendableTrees.jl")

Random.seed!(2024)

BostonHousing = CSV.read("BostonHousing.csv", DataFrame)
y = BostonHousing[:, :medv]
X = Matrix(BostonHousing[:, Not(:medv)])
oob_errors = Array{Float64}(undef, 13)
train_errors = Array{Float64}(undef, 13)

@time begin
    for b in 1:13
        BostonHousing_forest = grow_forest(X, y, "Regression", L_squared_error;
        max_depth = 5, min_node_size = 5, n_features = b, sfrac = 0.7, swr = false)
        oob_errors[b] = error_Regression(BostonHousing_forest, X, y, true)
        train_errors[b] = error_Regression(BostonHousing_forest, X, y)
        println(b)

        # clear memory before fitting the next forest
        BostonHousing_forest = nothing
    end
end

CSV.write("BostonHousing_nfeatures.csv",
Tables.table(hcat(1:13 ,train_errors, oob_errors)), writeheader = true)
```

To generate the corresponding figure, the following code was used.

```
nfeatures = CSV.read("BostonHousing_nfeatures.csv", DataFrame)
df = (; x = [nfeatures.n_features; nfeatures.n_features],
        y = [nfeatures.Training_error; nfeatures.OOB_error],
        Error = [fill("Train", 13); fill("OOB", 13)])
layers = visual(Lines) + visual(Scatter) * mapping(marker = :Error)
plt = data(df) * layers * mapping(:x, :y, color = :Error)
fg_nf = draw(plt, axis = (xlabel = "Number of features",
                xticks = nfeatures.n_features,
                ylabel = "Error rate"),
                figure = (; size = (800, 400),
                title = "Error rates for varying number of features
                used in a split",
                titlealign = :center))

save("Figures/n_features_plot.png", fg_nf, px_per_unit = 3)
```

B.4 The survival analyses in Subsection 5.7

The following code was used to test the effect of varying the minimum node size as well as the splitting rule on the `peakVO2` dataset.

```
include("JuliaExtendableTrees.jl")

# read in the data
df = CSV.read("peakVO2.csv", DataFrame)
y = Matrix(df[:, [:ttodead, :died]])
X = Matrix(df[:, Not([:ttodead, :died])])

Random.seed!(2024)

# min_node_sizes to check
sizes = [5, 10, 15, 20, 30, 40, 50]

# save errors and average number of terminal nodes
error_mns_lr = zeros(Float64, 7)
antn_mns_lr = zeros(Float64, 7)
error_mns_con = zeros(Float64, 7)
antn_mns_con = zeros(Float64, 7)
error_mns_lrs = zeros(Float64, 7)
antn_mns_lrs = zeros(Float64, 7)
error_mns_alr = zeros(Float64, 7)
antn_mns_alr = zeros(Float64, 7)
error_mns_C = zeros(Float64, 7)
antn_mns_C = zeros(Float64, 7)

@time begin
    for i in 1:length(sizes)
        # log-rank
        forest = grow_forest(X, y, "Survival", L_log_rank;
        min_node_size = sizes[i], n_trees = 1000, sfrac = 1.0, swr = true)
        error_mns_lr[i] = OOB_error(forest)
        antn_mns_lr[i] = forest.avr_number_terminal_nodes
        forest = nothing    # free memory

        # conserve
        forest = grow_forest(X, y, "Survival", L_conserve;
        min_node_size = sizes[i], n_trees = 1000, sfrac = 1.0, swr = true)
        error_mns_con[i] = OOB_error(forest)
        antn_mns_con[i] = forest.avr_number_terminal_nodes
        forest = nothing    # free memory

        # log-rank-score
        forest = grow_forest(X, y, "Survival", L_log_rank_score;
        min_node_size = sizes[i], n_trees = 1000, sfrac = 1.0, swr = true)
        error_mns_lrs[i] = OOB_error(forest)
        antn_mns_lrs[i] = forest.avr_number_terminal_nodes
        forest = nothing    # free memory

        # approximate log-rank
        forest = grow_forest(X, y, "Survival", L_approx_log_rank;
        min_node_size = sizes[i], n_trees = 1000, sfrac = 1.0, swr = true)
        error_mns_alr[i] = OOB_error(forest)
        antn_mns_alr[i] = forest.avr_number_terminal_nodes
        forest = nothing    # free memory

        # C-index
```

```
        forest = grow_forest(X, y, "Survival", L_C;
        min_node_size = sizes[i], n_trees = 1000, sfrac = 1.0, swr = true)
        error_mns_C[i] = OOB_error(forest)
        antn_mns_C[i] = forest.avr_number_terminal_nodes
        forest = nothing    # free memory

        println("Iteration: ", i, "/7")
    end
end

# total time: 6489.334204 seconds (about 108 minutes)

CSV.write("peakVO2_min_node_size_error.csv",
          Tables.table(hcat(sizes, error_mns_lr, error_mns_con, error_mns_lrs,
          error_mns_alr, error_mns_C)), writeheader = true)

CSV.write("peakVO2_min_node_size_antn.csv",
          Tables.table(hcat(sizes, antn_mns_lr, antn_mns_con, antn_mns_lrs,
          antn_mns_alr, antn_mns_C)), writeheader = true)

# conclusion: log_rank_score with min_node_size = 20 is best
```

The other parts of the hyperparameter tuning was not as complicated. Below is the code used to test the effect of the number of split points.

```
Random.seed!(2024)

splits = [1, 2, 3, 4, 5, 10, 25, 50, 100]
error_split = zeros(Float64, 9)
antn_split = zeros(Float64, 9)

for i in 1:9
    forest = grow_forest(X, y, "Survival", L_log_rank_score; min_node_size = 20,
    n_trees = 1000, sfrac = 0.6, swr = false, n_split = splits[i])
    error_split[i] = OOB_error(forest)
    antn_split[i] = forest.avr_number_terminal_nodes
    forest = nothing    # free memory
    println("Iteration: ", i)
end

CSV.write("peakVO2_split.csv",
          Tables.table(hcat(splits, error_split, antn_split)), writeheader = true)

# best result is n_split = 1 (hmm, strange). n_split = 3 is also quite good
```

The code for varying the sampling scheme, the number of features and the number of trees is very similar and is thus omitted here. All tests may be found in the `peakVO2_tests.jl` file. The code for producing the figures is very similar to the classification and regression case and is hence also omitted. See the file `rsf_figures.jl`.

C More on the simulated data

C.1 Exploratory plots

We start by providing a few plots of the data as well as some basic statistics. When we talk about lifetimes, we consider the total lifetimes, that is, the observed survival time plus the age variable (denoted X_5 above). For males, the average lifetime is 79.85 years, and the median is 82.12 years. For females, the corresponding statistics are 83.63 and 85.96. A plot of the distribution of lifetimes for males and females are below.

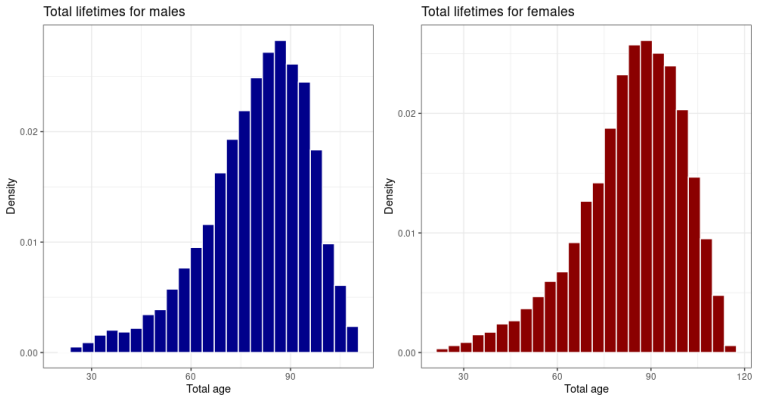

Figure 24: Histograms of lifetimes for males and females in the simulated data.

We see that it is a lot more likely for women to reach very high ages (100 years or above) in this model. We also plot the lifetimes for two wage groups, those earning less than 500.000 a year and those who earn above this figure.

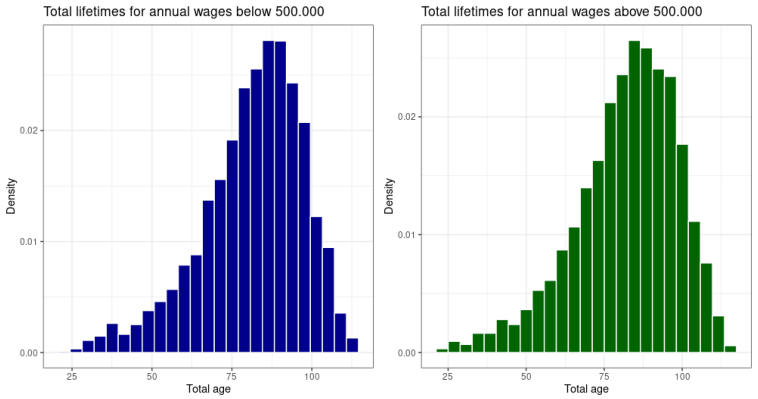

Figure 25: Histograms of lifetimes for two wage groups in the simulated data.

The difference in the two groups is not that large, since other covariates play a more dominant role in determining the mortality. Nevertheless, some difference is visible. A larger part of the high wage population lives longer. Finally, we plot the distribution for the married and unmarried individuals.

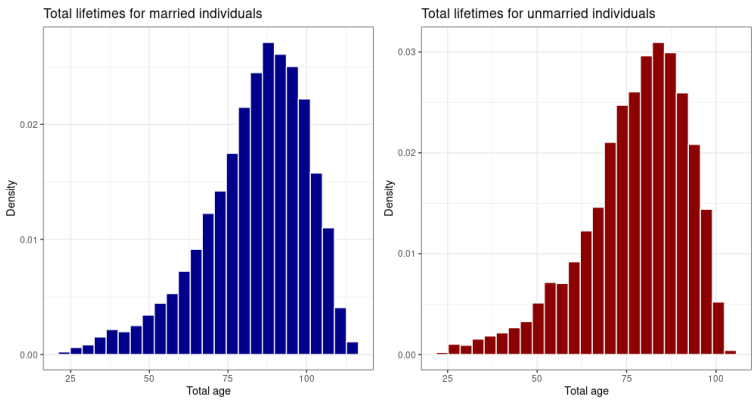

Figure 26: Histograms of lifetimes for married and unmarried individuals in the simulated data.

Here the tendency is clear. Married individuals live longer than unmarried individuals. In fact, the average lifetime for married individuals is 83.57, and the median is 86.19. For unmarried individuals, the corresponding statistics are 77.42 and 79.72.

C.2 The R code used for simulating the data

The following code defines the quantile function for the residual lifetime distribution F_x, when F is the Gompertz–Makeham distribution as well as the parameter functions of the covariates.

```
library(emdbook)

# quantile function for the Gompertz-Makeham distribution
Q <- function(u, b, g, r) {
  g/(b * r) - 1/b * log(1 - u) - 1/r * lambertW(g/b * exp(g/b) * (1 - u)^(-r/b))
}

# quantile function for the Gompertz-Makeham distribution (with starting age)
Q_x <- function(u, x, b, g, r) {
  Q(1 - (1 - u)*S_GM(x, 0, b, g, r), b, g, r) - x
}

beta <- function(x) {
  0.0004 + x * 0.0001
}

gamma <- function(x) {
  0.000072 + x * 0.000006
}

rho <- function(x, y, z) {
  0.0785 - x * 0.008 + 1000/y + z * 0.005
}
```

With these functions in place, the data is simulated as follows.

```
n <- 10^4

set.seed(2024)
# sex, male = 1, female = 0
X1 <- rbinom(n, 1, 0.5)
# place of residence, 0 = city, 0.5 = suburbia, 0.8 = village, 1 = open lands
X2 <- sample(x = c(0, 0.5, 0.8, 1), size = n, replace = TRUE,
prob = c(5/12, 1/4, 1/6, 1/6))
# married, yes = 1, no = 0
X3 <- rbinom(n, 1, 0.7)
# annual wage
X4 <- round(runif(n, 2.5*10^5, 7.5*10^5))
# age at time 0
X5 <- rnorm(n, 25, sqrt(8))

# now simulate the data
U <- runif(n)
data <- tibble(Time = rep(0, n), Sex = X1, Residence = X2, Married = X3,
Annual_Wage = X4, Age = X5, Total_Age = rep(0, n))
for (i in 1:n) {
  data[i, 1] <- Q_x(U[i], X5[i], beta(X1[i]), gamma(X2[i]), rho(X3[i], X4[i], X1[i]))
  if (i %% 100 == 0) {
    print(i)
  }
}
data$Total_Age <- data$Time + data$Age
sum(is.na(data$Time)) # 210 NaN values
data <- na.omit(data) # just remove the NaN values

View(data)
write.csv(data, "sim_data.csv")
```

C.3 The R code used for analysing the data

In order to analyse the data, we start by importing the packages needed and adding "censoring". Since we impose no censoring, we simply add a vector of ones to the dataframe.

```
library("randomForestSRC")
library(tidyverse)
library(survival)
library(eha)

# add the censoring indicators (a vector of ones in this case)
sim_data <- sim_data %>% mutate(Status = rep(1, nrow(sim_data)))
```

Predicting the cumulative hazard from a Poisson model is not implemented in the `eha` package. Hence we need to do this manually[3]. In the following, `t` denotes a vector of sorted unique event times, and `fit_obj` is an object of type `pchreg` from the `eha` package.

```
# compute the cumulative hazard function given constant hazards haz between
# the jump points cuts
cumulative_hazard <- function(haz, cuts) {
  num_cuts <- length(cuts)
  res <- rep(0, num_cuts)
  if (num_cuts > 1) {
    for (i in 2:num_cuts) {
      res[i] <- res[i - 1] + (cuts[i] - cuts[i - 1]) * haz[i - 1]
    }
  }
  res
}

# compute the predicted chf from an object of type pchreg for a single covariate
# vector x
pred_pcph_onedim <- function(fit_obj, x, t, cum_haz) {
  haz <- as.numeric(fit_obj$hazards)
  cuts <- as.numeric(fit_obj$cuts)
  coef <- as.numeric(fit_obj$coefficients)

  N <- length(t)
  res <- rep(0, N)

  for (i in 1:N) {
    # find the interval for the event time
    start <- 2
    baseline = 0
    for (j in start:length(cuts)) {
      if (t[i] <= cuts[j]) {
        # compute the baseline hazard in the point t[i]
        baseline <- cum_haz[j - 1] + (t[i] - cuts[j - 1]) * haz[j - 1]

        # the next event time is larger, so start next i iteration in current j
        start <- j
        break
      }
```

[3] To do this properly, one should vectorize the `pred_pcph_onedim` function instead of defining `pred_pcph` as done here.

```
    }
    res[i] <- baseline * exp(t(coef) %*% x)
    print(res[i])
  }
  res
}

# compute the predicted chf from an object of type pchreg for a matrix of
# covariates X
pred_pcph <- function(fit_obj, X, t) {
  n <- nrow(X)
  N <- length(t)
  res <- matrix(, nrow = n, ncol = N)
  cum_haz <- cumulative_hazard(fit_obj$hazards, fit_obj$cuts)

  for (i in 1:n) {
    res[i, ] <- pred_pcph_onedim(fit_obj, as.numeric(X[i, ]), t, cum_haz)
  }

  return(res)
}
```

It was also necessary to implement a function for computing the C-index in R. This was done the same way as in Julia and we hence omit the code here. With all these functions in place, the analysis is run using the following code.

```
B <- 100
n <- nrow(sim_data)
error_RSF <- error_Cox <- error_Poisson <- rep(0, B)
#set.seed(2024)

start.time <- Sys.time()
for (b in 1:B) {
  # generate a bootstrap sample
  bootstrap_sample <- sim_data[sample(1:n, n, replace = TRUE), ]

  # fit the RSF and save the error
  rf <- rfsrc(Surv(Time, Status) ~ ., bootstrap_sample,
              ntree = 500, save.memory = TRUE,
              splitrule = "logrankscore", nodesize = 5)
  error_RSF[b] <- last(rf$err.rate)

  # fit the Cox PH model and save the error
  error_Cox[b] <- 1 - concordance(coxph(Surv(Time, Status) ~ .,
  bootstrap_sample), newdata = bootstrap_sample)$concordance

  # fit the Poisson regression model
  fit <- pchreg(Surv(Time, Status) ~ ., data = bootstrap_sample,
  cuts = c(0, 5, 20, 40, 60, 70, 80, 85, 90, 100))
  # compute predictions for the Poisson regression model. rf$xvar are the features
  # from the data, and rf$time.interest are the event times used in the RSF model
  predictions_pcph <- pred_pcph(fit, rf$xvar, rf$time.interest)
  # save the error
```

```
  error_Poisson[b] <- 1 - C_index(rowSums(predictions_pcph),
  bootstrap_sample$Time, bootstrap_sample$Status)

  # count iteration and print results
  print("Iteration:")
  print(b)

  results <- tibble(RSF = error_RSF, Cox = error_Cox, Poisson = error_Poisson)
  write.csv(results, file = "sim_results.csv", row.names = FALSE)
}
end.time <- Sys.time()
end.time - start.time
```

On my laptop, the above code took a little under nine hours to run in total. I was even forced to split up the computations in two chunks with the first chunk containing the first 90 iterations. A .csv file is saved for every iteration to make sure no results were lost. Using this file with the errors, the box plot from the RSF section was made with the following code.

```
sim_results <- tibble(read.csv("sim_results.csv"))

sim_results_plot <- tibble(Error = c(sim_results$RSF, sim_results$Cox,
                                     sim_results$Poisson),
                           Model = c(rep("RSF", 100), rep("Cox", 100),
                                     rep("Poisson", 100)))

# boxplot of the above analysis
box <- ggplot(data = sim_results_plot, mapping = aes(Model, Error)) +
  geom_boxplot(colour = "black")

ggsave("sim_box.png", box)
```

In order to compute and plot the eight Nelson–Aalen estimators, we start by fitting the models and defining the different features.

```
# fit models
rf <- rfsrc(Surv(Time, Status) ~ ., sim_data, ntree = 500, save.memory = TRUE,
            splitrule = "logrankscore", nodesize = 5)
cox <- coxph(Surv(Time, Status) ~ ., sim_data)
pcph <- pchreg(Surv(Time, Status) ~ ., data = sim_data,
               cuts = c(0, 5, 20, 40, 60, 70, 80, 85, 90, 100))

event_times <- rf$time.interest

# choose eight different combinations of features
feature1 <- filter(rf$xvar, Residence == 1, Sex == 1,
                   Married == 1, Annual_Wage <= 310000)[3,]
feature2 <- filter(rf$xvar, Residence == 1, Sex == 1,
                   Married == 1, Annual_Wage >= 600000)[20,]
feature3 <- filter(rf$xvar, Residence == 1, Sex == 1,
```

```
                  Married == 0, Annual_Wage <= 310000)[1,]
feature4 <- filter(rf$xvar, Residence == 1, Sex == 1,
                  Married == 0, Annual_Wage >= 590000)[57,]
feature5 <- filter(rf$xvar, Residence == 1, Sex == 0,
                  Married == 1, Annual_Wage <= 310000)[6,]
feature6 <- filter(rf$xvar, Residence == 1, Sex == 0,
                  Married == 1, Annual_Wage >= 590000)[10, ]
feature7 <- filter(rf$xvar, Residence == 1, Sex == 0,
                  Married == 0, Annual_Wage <= 310000)[2, ]
feature8 <- filter(rf$xvar, Residence == 1, Sex == 0,
                  Married == 0, Annual_Wage >= 600000)[18, ]
```

We now make the eight plots manually by changing the value of `feature` below and the name of `plot`. The combined plot is then made using `ggarrange` from the `ggpubr` package, Kassambara [41].

```
# change feature
feature <- feature8

# predict for the RSF model
pred_rf <- predict.rfsrc(rf, feature)
data_rf <- tibble(time = event_times, hazard = as.numeric(pred_rf$chf))

# predict for the CoxPH model
pred_cox <- tibble(basehaz(fit = cox))
pred_cox[, 1] <- pred_cox[, 1] *
exp(t(as.numeric(cox$coefficients)) %*% as.numeric(feature))

# predict for the Poisson model
data_pcph <- tibble(time = event_times,
hazard = as.numeric(pred_pcph(pcph, feature, event_times)))

# the true cumulative hazard curve
mu <- function(t, x) {
  beta(x[1]) + gamma(x[2]) * exp(rho(x[3], x[4], x[1]) * (t + x[5]))
}

# this needs to be done for plot1, plot2, ..., plot8
plot8 <- ggplot() +
  geom_step(data = pred_cox, mapping = aes(x = time, y = hazard),
            color = "blue") +
  geom_step(data = data_pcph, mapping = aes(x = time, y = hazard),
            color = "darkgreen") +
  geom_step(data = data_rf, mapping = aes(x = time, y = hazard),
            color = "red") +
  geom_function(fun = mu, args = list(x = as.numeric(feature)),
            color = "purple") +
  xlab("Time (years)") + ylab("Cumulative hazard") + ylim(0, 20)

ggarrange(plot1, plot2, plot3, plot4, plot5, plot6, plot7, plot8,
          nrow = 4, ncol = 2)
```

D More on conditional independence and Markov processes

In this final section of the appendix, we briefly go through the setup of conditional independence and Markov processes as presented in Hansen [24]. The exposition highlights the strengths of applying a purely measure-theoretical approach to Markov processes and is therefore worth presenting. Fix a probability space $(\Omega, \mathcal{F}, \mathbb{P})$. All collections of events (subsets) are implicitly assumed to be contained in $\mathcal{F}$.

Definition D.1. Let $\mathcal{A}$ and $\mathcal{B}$ be two collections of events. $\mathcal{A}$ and $\mathcal{B}$ are *conditionally independent* given a sigma-algebra $\mathcal{H}$ if

$$\mathbb{P}(A \cap B \mid \mathcal{H}) = \mathbb{P}(A \mid \mathcal{H})\mathbb{P}(B \mid \mathcal{H}) \quad \text{a.s.} \quad \text{for all } A \in \mathcal{A}, B \in \mathcal{B}.$$

For the sake of brevity, we will from now on simply say that $\mathcal{A}$ and $\mathcal{B}$ are independent given $\mathcal{H}$. The following result is more or less immediate.

Lemma D.2. *Let $\mathcal{A}$ and $\mathcal{B}$ be collections of events.*

(i) If $\mathcal{H}$ is trivial, in the sense that $\mathbb{P}(H) \in \{0, 1\}$ for all $H \in \mathcal{H}$, then $\mathcal{A} \perp\!\!\!\perp \mathcal{B} \mid \mathcal{H}$ if and only if $\mathcal{A} \perp\!\!\!\perp \mathcal{B}$.

(ii) If $\mathcal{A}$ and $\mathcal{B}$ are $\cap$-stable, then

$$\mathcal{A} \perp\!\!\!\perp \mathcal{B} \mid \mathcal{H} \quad \Rightarrow \quad \sigma(\mathcal{A}) \perp\!\!\!\perp \sigma(\mathcal{B}) \mid \mathcal{H}.$$

(iii) If $\mathcal{C} \subseteq \mathcal{A}$, then $\mathcal{A} \perp\!\!\!\perp \mathcal{B} \mid \mathcal{H}$ implies that $\mathcal{C} \perp\!\!\!\perp \mathcal{B} \mid \mathcal{H}$.

(iv) Assume $\mathcal{A}$ and $\mathcal{B}$ are sigma-algebras with $\mathcal{A} \perp\!\!\!\perp \mathcal{B} \mid \mathcal{H}$. Then it holds for any bounded real-valued random variables X and Y with X $\mathcal{A}$-measurable and Y $\mathcal{B}$-measurable that

$$\mathbb{E}[XY \mid \mathcal{H}] = \mathbb{E}[X \mid \mathcal{H}]\mathbb{E}[Y \mid \mathcal{H}].$$

Proof. (i) follows immediately from the fact that $\mathbb{P}(A \mid \mathcal{H}) = \mathbb{P}(A)$ for every $A \in \mathcal{A}$ under the assumption on $\mathcal{H}$. As for (ii), let $A \in \mathcal{A}$ and define

$$\mathcal{G}_A = \{B \in \sigma(\mathcal{B}) : \mathbb{P}(A \cap B \mid \mathcal{H}) = \mathbb{P}(A \mid \mathcal{H})\mathbb{P}(B \mid \mathcal{H})\}.$$

Since

$$\mathcal{G}_A = \sigma(\mathcal{B}) \cap \{B \in \mathcal{F} : \mathbb{P}(A \cap B \mid \mathcal{H}) = \mathbb{P}(A \mid \mathcal{H})\mathbb{P}(B \mid \mathcal{H})\}$$

is an intersection of two Dynkin systems, $\mathcal{G}_A$ is itself a Dynkin system. By assumption, $\mathcal{B} \subseteq \mathcal{G}_A$, and by construction, $\mathcal{G}_A \subseteq \sigma(\mathcal{B})$, so the stability under finite intersections of $\mathcal{B}$ gives by Dynkin's lemma that $\mathcal{G}_A = \sigma(\mathcal{B})$. As this holds for every A, we have $\mathcal{A} \perp\!\!\!\perp \sigma(\mathcal{B}) \mid \mathcal{H}$. Now repeat the argument to obtain $\sigma(\mathcal{A}) \perp\!\!\!\perp \sigma(\mathcal{B}) \mid \mathcal{H}$. (iii) is a trivial observation. (iv) follows by usual measure-theoretical arguments. By assumption, the claim is true for indicator functions, and linearity of expectations extends the result to simple functions. Now approximate X and Y by simple functions and apply proper limit theorems. ■

The following result provides a useful reformulation of the concept of conditional independence which is particularly nice for certain computations. It states that $\mathcal{A} \perp\!\!\!\perp \mathcal{B} \mid \mathcal{H}$ means that the information in $\mathcal{B}$ tells us nothing about the behaviour of $\mathcal{A}$-measurable variables if the information in $\mathcal{H}$ is already provided.

Proposition D.3. *Let $\mathcal{A}$ and $\mathcal{B}$ be sigma-algebras. We have $\mathcal{A} \perp\!\!\!\perp \mathcal{B} \mid \mathcal{H}$ if and only if*

$$\mathbb{P}(A \mid \mathcal{B} \vee \mathcal{H}) = \mathbb{P}(A \mid \mathcal{H}) \quad \text{a.s.} \quad \text{for every } A \in \mathcal{A}$$

where $\mathcal{B} \vee \mathcal{H} = \sigma(\mathcal{B} \cup \mathcal{H})$.

Proof. First note that for any $A \in \mathcal{A}, B \in \mathcal{B}$ and $H \in \mathcal{H}$, we have that

$$\begin{aligned}\int_{B\cap H} \mathbb{P}(A \mid \mathcal{H}) d\mathbb{P} &= \int_H \mathbb{1}_B \mathbb{P}(A \mid \mathcal{H}) d\mathbb{P} = \int_H \mathbb{E}[\mathbb{1}_B \mathbb{P}(A \mid \mathcal{H}) \mid \mathcal{H}] d\mathbb{P} \\ &= \int_H \mathbb{P}(A \mid \mathcal{H}) \mathbb{P}(B \mid H) d\mathbb{P}.\end{aligned}$$

Suppose $\mathcal{A} \perp\!\!\!\perp \mathcal{B} \mid \mathcal{H}$. In this case, we can continue the computation and obtain

$$\int_{B\cap H} \mathbb{P}(A \mid \mathcal{H}) d\mathbb{P} = \int_H \mathbb{P}(A \cap B \mid \mathcal{H}) d\mathbb{P} = \mathbb{P}(A \cap B \cap H) = \int_{B\cap H} \mathbb{1}_A d\mathbb{P}.$$

Sets of the form $B \cap H$ form a $\cap$-stable generator for $\mathcal{B} \vee \mathcal{H}$. Hence we see that $\mathbb{P}(A \mid \mathcal{H})$ satisfies all assumptions necessary to be a conditional expectation of $\mathbb{1}_A$ given $\mathcal{B} \vee \mathcal{H}$, that is, $\mathbb{P}(A \mid \mathcal{H}) = \mathbb{P}(A \mid \mathcal{B} \vee \mathcal{H})$ a.s. Conversely, suppose $\mathbb{P}(A \mid \mathcal{B} \vee \mathcal{H}) = \mathbb{P}(A \mid \mathcal{H})$ a.s. for every $A \in \mathcal{A}$. We can use the relation above to see that

$$\int_H \mathbb{P}(A \mid \mathcal{H}) \mathbb{P}(B \mid \mathcal{H}) d\mathbb{P} = \int_{B\cap H} \mathbb{P}(A \mid \mathcal{B} \vee \mathcal{H}) d\mathbb{P} = \int_{B\cap H} \mathbb{1}_A d\mathbb{P} = \int_H \mathbb{1}_{A\cap B} d\mathbb{P}.$$

Hence $\mathbb{P}(A \mid \mathcal{H})\mathbb{P}(B \mid \mathcal{H})$ satisfies all requirements of being the conditional expectation of $\mathbb{1}_{A\cap B}$ given $\mathcal{H}$, and the statement is proved. ■

Using standard techniques, it immediately follows that for any bounded $\mathcal{A}$-measurable variable X, it holds that

$$\mathcal{A} \perp\!\!\!\perp \mathcal{B} \mid \mathcal{H} \quad \Rightarrow \quad \mathbb{E}[X \mid \mathcal{B} \vee \mathcal{H}] = \mathbb{E}[X \mid \mathcal{H}] \quad \text{a.s.}$$

The following results are essential tools in strengthening the Markov property.

Proposition D.4. *Let $\mathcal{A}$ and $\mathcal{B}$ be sigma-algebras. Then*

$$\mathcal{A} \perp\!\!\!\perp \mathcal{B} \mid \mathcal{H} \quad \Rightarrow \quad \mathcal{A} \perp\!\!\!\perp (\mathcal{B} \vee \mathcal{H}) \mid \mathcal{H}.$$

Proof. Suppose $\mathcal{A} \perp\!\!\!\perp \mathcal{B} \mid \mathcal{H}$ and let $A \in \mathcal{A}, B \in \mathcal{B}$ and $H \in \mathcal{H}$. Then

$$\begin{aligned}\mathbb{P}(A \cap B \cap H \mid \mathcal{H}) &= \mathbb{1}_H \mathbb{P}(A \cap B \mid \mathcal{H}) = \mathbb{1}_H \mathbb{P}(A \mid \mathcal{H}) \mathbb{P}(B \mid \mathcal{H}) \\ &= \mathbb{P}(A \mid \mathcal{H}) \mathbb{P}(B \cap H \mid \mathcal{H}),\end{aligned}$$

and since events of the form $B \cap H$ is a $\cap$-stable generator of $\mathcal{B} \vee \mathcal{H}$, the result now follows from part (ii) of the lemma above. ■

Proposition D.5. *Let $\mathcal{A}$ and $\mathcal{B}$ be sigma-algebras. Then*

$$\mathcal{A} \perp\!\!\!\perp \mathcal{B} \mid \mathcal{H} \quad \textit{and} \quad \mathcal{H} \subseteq \mathcal{G} \subseteq \mathcal{B} \vee \mathcal{H} \quad \Rightarrow \quad \mathcal{A} \perp\!\!\!\perp \mathcal{B} \mid \mathcal{G}.$$

Proof. Let $A \in \mathcal{A}$. Using the extended tower property as well as Proposition D.3, we compute

$$\begin{aligned}\mathbb{P}(A \mid \mathcal{B} \vee \mathcal{G}) &= \mathbb{E}[\mathbb{P}(A \mid \mathcal{B} \vee \mathcal{G} \vee \mathcal{H}) \mid \mathcal{B} \vee \mathcal{G}] = \mathbb{E}[\mathbb{P}(A \mid \mathcal{B} \vee \mathcal{H}) \mid \mathcal{B} \vee \mathcal{G}] \\ &= \mathbb{E}[\mathbb{P}(A \mid \mathcal{H}) \mid \mathcal{B} \vee \mathcal{G}] = \mathbb{P}(A \mid \mathcal{H}) \quad \text{a.s.}\end{aligned}$$

By the same argument,

$$\mathbb{P}(A \mid \mathcal{G}) = \mathbb{E}[\mathbb{P}(A \mid \mathcal{B} \vee \mathcal{H}) \mid \mathcal{G}] = \mathbb{E}[\mathbb{P}(A \mid \mathcal{H}) \mid \mathcal{G}] = \mathbb{P}(A \mid \mathcal{H}).$$

Combining these two calculations, we have $\mathbb{P}(A \mid \mathcal{B} \vee \mathcal{G}) = \mathbb{P}(A \mid \mathcal{G})$ a.s. for every $A \in \mathcal{A}$ which by Proposition D.3 implies that $\mathcal{A} \perp\!\!\!\perp \mathcal{B} \mid \mathcal{G}$ as desired. ■

Proposition D.6. *Let $\mathcal{A}, \mathcal{B}$ and $\mathcal{G}$ be sigma-algebras. Then*

$$\mathcal{A} \perp\!\!\!\perp \mathcal{B} \mid \mathcal{H} \quad \textit{and} \quad \mathcal{A} \perp\!\!\!\perp \mathcal{G} \mid \mathcal{B} \vee \mathcal{H} \quad \Rightarrow \quad \mathcal{A} \perp\!\!\!\perp (\mathcal{B} \vee \mathcal{G}) \mid \mathcal{H}.$$

Proof. Follows immediately from Proposition D.3 and the calculation

$$\mathbb{P}(A \mid \mathcal{B} \vee \mathcal{G} \vee \mathcal{H}) = \mathbb{P}(A \mid \mathcal{B} \vee \mathcal{H}) = \mathbb{P}(A \mid \mathcal{H}) \quad \text{a.s.}$$

for every $A \in \mathcal{A}$. ■

We can now prove the desired strengthening of the Markov property.

Proof of Theorem 6.10. We aim to show that

$$\sigma(Z_u : u \geq t) \perp\!\!\!\perp \mathcal{F}_s^Z \mid Z_s \quad \text{for } s < t.$$

The collection

$$\bigcup_{\substack{I \subseteq [t,\infty) \\ \#I < \infty}} \sigma(Z_u : u \in I)$$

is a $\cap$-stable generator of $\sigma(Z_u : u \geq t)$. Hence it suffices to show that for every $s < t_1 < \cdots < t_n$, we have

$$(Z_{t_1}, ..., Z_{t_n}) \perp\!\!\!\perp \mathcal{F}_s^Z \mid Z_s.$$

We prove this statement using induction on n, the case $n = 1$ being simply the definition of the Markov property. Now assume the statement is proven for some n. Consider $s < t_1 < \cdots < t_n < t_{n+1}$. We know that

$$Z_{t_{n+1}} \perp\!\!\!\perp \mathcal{F}_{t_n}^Z \mid Z_{t_n}.$$

Since $\sigma(Z_{t_n}) \subseteq \sigma(Z_s, Z_{t_1}, ..., Z_{t_n}) \subseteq \mathcal{F}_{t_n}^Z$, it follows from Proposition D.5 that

$$Z_{t_{n+1}} \perp\!\!\!\perp \mathcal{F}_{t_n}^Z \mid (Z_s, Z_{t_1}, ..., Z_{t_n})$$

which implies

$$Z_{t_{n+1}} \perp\!\!\!\perp \mathcal{F}_s^Z \mid (Z_s, Z_{t_1}, ..., Z_{t_n}).$$

By the induction hypothesis, we have

$$(Z_{t_1}, ..., Z_{t_n}) \perp\!\!\!\perp \mathcal{F}_s^Z \mid Z_s.$$

Now apply Proposition D.6 with $\mathcal{A} = \mathcal{F}_s^Z, \mathcal{H} = \sigma(Z_s), \mathcal{B} = (Z_{t_1}, ..., Z_{t_n})$ and $\mathcal{G} = \sigma(Z_{t_{n+1}})$ to obtain

$$(Z_{t_1}, ..., Z_{t_{n+1}}) \perp\!\!\!\perp \mathcal{F}_s^Z \mid Z_s$$

which finishes the induction step and thus the proof of the theorem. ■