PREDICTION MODELS FOR LIVER TRANSPLANTATION

Comparisons between Cox proportional hazards regression models and machine learning techniques.

Georgios Kantidakis (s1839276)

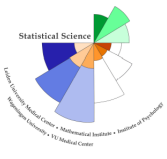
First supervisor: Dr. Marta Fiocco

Second supervisor: Dr. Carlo Lancia

MASTER THESIS

Defended on 29th October 2018

Specialization: Statistical Science



STATISTICAL SCIENCE FOR THE LIFE AND BEHAVIOURAL SCIENCES

Abstract

Liver transplantation -i.e. the replacement of a diseased liver with healthy liver of another person, is the most effective therapeutic strategy for patients with end-stage liver disease. Predicting survival of patients after liver transplantation is regarded as one of the most challenging areas in medicine. Hence, selecting the best prediction model is of paramount importance.

Machine learning - field of computer science where specific algorithms are used to learn and make predictions on data - has lately received increased attention in the medical field due to contribution in medical imaging, ability to diagnose diseases and its great potential for personalized treatment.

In survival analysis, machine learning implementation is difficult due to censored data. In this thesis, survival random forests and partial logistic artificial neural networks have been applied.

Cox model has been exclusively used due to its easy implementation and straightforward interpretation. The model is however restricted to the proportionality of hazards assumption whereas the machine learning techniques do not make any assumptions.

Nowadays, there is a strong discussion in the medical field about machine learning and if it has greater potential than Cox models when it comes to complex data. Criticism to machine learning is related to unsuitable performance measures and lack of interpretability which is important for the medical personnel.

The potential of machine learning is investigated for large data of 62294 patients in USA for 106 prognostic factors selected from over 600; 52 donor's characteristics and 54 patient's characteristics. A meticulous comparison is performed between 3 proportional hazards models and machine learning techniques. For the artificial neural network novel extensions are provided to its original specification using state-of-the-art R software.

A variety of measures is employed not only from survival field but also from simple classification setting. In this project, it is of particular interest the identification of potential risk factors post-operatively. Two survival outcomes are reported: overall survival (time to death since operation) and failure-free survival (minimum time between graft-failure and death since operation).

In this thesis, it is shown that machine learning techniques can be a useful tool for both prediction and interpretation. Random survival forest shows in general better predictive performance than Cox models. Neural networks can reach comparable performance with the Cox models and even perform better in some classification metrics. However, high instability is present due to the lack of a global performance evaluation measure in survival setting.

Keywords — Survival Analysis, Random Forests, Artificial Neural Networks

Acknowledgements

I would like to thank my first supervisor Dr. Marta Fiocco for her incredible support and thorough guidance during this project and for giving me the opportunity to work with her.

I would like to express my gratitude to my second supervisor Dr. Carlo Lancia for his support and statistical insights not only during this thesis but also during my internship.

I would also like to thank the Scientific Registry of Transplant Recipients for providing data about liver transplantation to Leiden University and the doctors from Leiden University Medical Center for their advice.

A special thanks to Dea Hazewinkel for her useful remarks and help with part of R code. It was an honor working with you.

I would like to express my gratitude to Max Hollmann, Efklidis Katsaros, Michail Tsiaousis and Georgios Vlassopoulos for their great suggestions in machine learning.

Few more names of friends that I would like to mention from this master are: Edouard Bonneville, Abdelhah Chahid, Daniela Gawehns, Martha Kartalidou, Niek Mereu, Shifai Rahamat, Ourania Tsampieri, Liu Yuping. Thank you for the fruitful statistical discussions and general conversations. I would also like to thank the other fellow students and professors from this master for those memorable 2 years.

I would like to express my deepest regards to Daniel Avila Ortega, Cynthia Flores Santana and Andrea Kyprou for hanging around with me in the beautiful city of Leiden and always supporting me.

Last but not least, I would like to thank the members of my family - my sister Angelikoula, my mother Ourania and my father Yiannis - for their extraordinary support and unconditional love.

Until we meet again, never let go of your dreams...

Contents

List of Figures

List of Tables

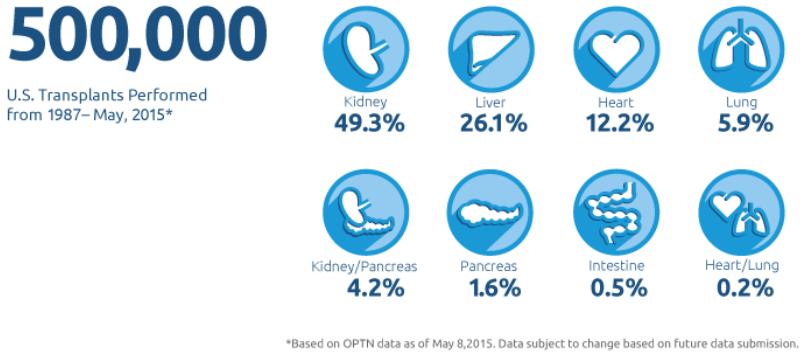
Chapter 1

Introduction

[1] Transplantations is one of the biggest chapters of medicine and of the revolution in science on the 20th century. The prelude to modern area of transplantations comes from World War 2, where the British pilots were replacing burnt areas in their skin from skin of healthy people in an effort to survive from airplane crushes (Barker and Markmann, 2013).

Liver transplantation - i.e. the replacement of a diseased liver with a healthy liver from another person (allograft) - is the second most common type of transplant surgery after kidney in the United States as illustrated in figure 1.1. Over the last decades, the success of liver transplants has improved survival outcome for a large number of patients suffering from chronic liver disease everywhere on earth (Merion et al., 2005). Availability of donor organs is a major limitation especially when compared with the growing demand of liver candidates due to the enlargement of age limits. Therefore, selecting the best prediction model is of paramount importance.

Figure 1.1: Percentages of transplantations per organ in the United States from 1987 to 2015.



One of the key discoveries that enabled transplantations development - together with improvement of surgery techniques - is that human antigens cause serious problems to the graft (the transplant of tissue or organ) on the recipients by rejecting or gradually destroying it. Therefore, even transplantations between people who are 100% compatible require suppression of their immune system to allow the graft successfully establish in their body.

[1] Navigation to sections, references, figures, tables, equations and algorithms is possible through the blue colored hyper-links (file should be in pdf format). To return to previous point use (Alt and ←).

On parallel with advances in medicine, the technological breakthroughs of this century brought a vast increase in computational power. This allowed the development of machine learning; field of computer science where particular algorithms are used to learn and make predictions on data. Lately, machine learning has received increased attention in the medical field due to its applicability in imaging data - a field with large variation and complexity - and its capability to diagnose diseases; an ability that requires years of medical training for doctors.

In the statistical field, there is an open discussion about the value of machine learning (ML) versus statistical modeling (SM) for medical application. SM is usually parametric/semi-parametric regression models such (generalized) linear models, longitudinal models or time-to-event models. For survival analysis data, Cox proportional hazards model (Cox, 1972) is the most frequently applied. This model has a straightforward interpretation, but is at the same time restricted to the proportional hazards assumption. ML techniques are assumption-free and data adaptive. This implies that they can be used for modeling complex data. ML methods are for example bagging, boosting, random forests, support vector machines, Bayesian networks, artificial neural networks and deep learning.

In this thesis, analysis is performed on a data set which includes 62294 patients. Information was collected by the United Network of Organ Sharing (UNOS); a non-profit, scientific organization in the United States which arranges organ donation and transplantation. After extensive pre-processing from a set of more than 600 covariates and imputation of missing values, 106 variables were included in the final dataset: 52 regarding characteristics of donors and 54 regarding characteristics of liver recipients. A detailed explanation of the variables meaning and their levels (if categorical) is given in appendix A tables A.1 and A.2 respectively. For the analysis, data will be split into two complementary parts. 2/3 will be used for training and 1/3 will be used for testing the predictive accuracy on new "unseen" observations. As UNOS data is large in both number of observations and covariates it is of interest to see how ML algorithms - which are able to capture naturally multi-way interactions between variables and can handle big data - will perform compared to the Cox models.

Predicting survival outcome in liver transplantations - especially postoperatively - is hard as it depends on many unpredictable factors and is associated with both donor and recipient. A first aim of this project is to identify potential risk factors for overall and failure-free survival (see section 2.2 for their definition). The relative importance of risk factors will be compared using different methods and focus will be on the identification of variables which are strong predictors.

The present manuscript aims to extend the discussion on ML by making a comprehensive comparison between Cox proportional hazards models and machine learning techniques based on the following points:

- general structure and assumption
- predictive performance
- interpretability
- ease of use
- potential for the medical field

More specifically, 3 Cox models will be used: 1) Cox model with all prognostic variables, 2) Cox model using backward elimination and 3) Cox model using variable selection with LASSO (Tibshirani, 1997). These models are compared with two extensions of machine learning for right censored survival data 1) random survival forest (RSF) Ishwaran et al. (2008a) and 2) partial logistic artificial neural networks (PLANN) (Biganzoli et al., 1998). For the second model, a novel approach has been applied to modeling in terms of structure.

RSF is an ensemble tree method for the analysis of right censored survival data. It is known that constructing ensembles from base learners such as decision trees can significantly improve prediction performance (Hastie et al., 2001). During growing each survival tree, a recursive application of binary splitting per node on a specific predictor is performed in such a way that survival difference across daughter nodes is maximized.

Artificial neural networks are based on a collection of connected units which are called *nodes* or *artificial neurons*, who transmit signals to one another. Connections between nodes are called *edges*. *Nodes* and *edges* have a weight which adjusts as learning proceeds increasing or decreasing the strength of the signal at connections. Typically, nodes are aggregated into layers. Different layers may perform different transformations on their inputs. Signals travel from the first layer (the input layer), to the last layer (the output layer). Layers in between are called hidden layers. A suitable loss (or cost) function is algorithmically minimized to assign proper values to the weights. PLANN extension for right censored data can be applied by transforming data to a long format and using time variable as part of the input features.

As machine learning models require a lot of training to achieve good performance, emphasis will be given in correct specification of tuning parameters for both RSF and neural networks. To compare the performance of SM with ML techniques measures are employed from 2 different areas: 1) survival field 2) machine learning field. The C-index, the time-dependent Brier score and a global version of it the integrated Brier score (IBS) are used as performance measure with survival data. From machine learning 5 metrics of a simple classification confusion matrix: accuracy, sensitivity, specificity, precision and f1score are used.

A major point of criticism for ML is the lack of suitable performance measures (Harrell Frank). Hyper-parameter selection and model evaluation can be misleading in case of poorly specified performance measures. In the field of computer science, the most widely used measure to evaluate model's performance is accuracy. Most of the time reaching the highest accuracy possible is the main goal at the cost of interpretability. One of the goals of this thesis is to get highly performing models which can be interpreted by using a proper measure for model tuning when possible, or a combination of many measures otherwise.

The outline of this thesis is as follows. In chapter 2, the data gathering process is discussed. Strategies for variable pre-selection and data pre-processing are identified and a suitable imputation method is used to reconstruct missing values. In chapter 3 an overview of all methods applied is presented. The focus is on theoretical background, model assumptions, technical details, software of implementation and extraction of interpretation. Chapter 4 provides a review of the performance measures that will be applied to compare models which consist of measures specialized for survival analysis but also of metrics used in simple classification setting. Results for each model are presented in chapter 5. Cox models are briefly discussed, whereas an extensive

emphasis is given to RSF and PLANN. All results are presented for overall and failure-free survival separately. In chapter 6 comparison of the methods is provided. A distinction is made between exact time points (all survival times in the data set) and discrete time points defined at 1, 2,...,12 years. Finally, chapter 7 offers an extensive discussion about ML in medical application and provides some guidelines about the possible choices between ML and SM. Future research and contribution of this thesis are presented.

Chapter 2

Data description

In this chapter, we discuss extensively relevant information about data originating from the United Network for Organ Sharing (UNOS). Data collection is discussed in section 2.1; in section 2.2 the pre-processing steps and decisions about variable pre-selection are presented. Section 2.3 provides some descriptive statistics. Finally, section 2.4 provides more details about the missing values and the imputation method used to reconstruct missing values.

2.1 Data gathering process

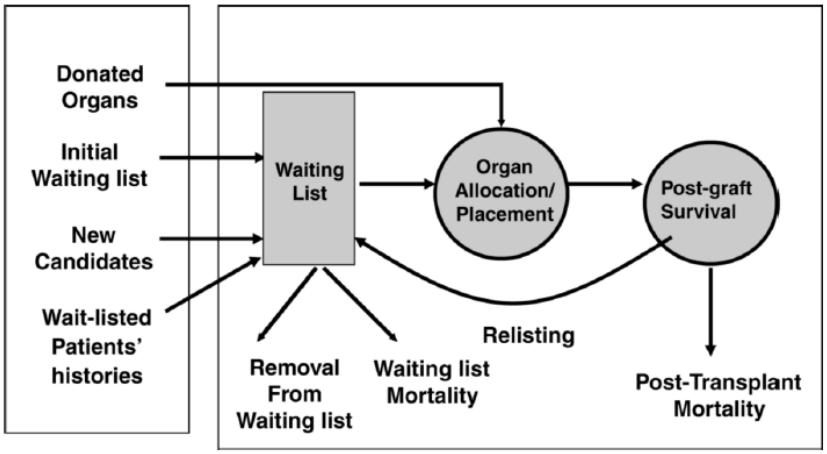
UNOS is a non-profit, educational and scientific organization based in the United States. It manages the Organ Procurement and Transplantation Network (OPTN) and together they operate by grouping states into several different regions. The aim of UNOS is to bring together professionals and volunteers under one purpose: saving lives through organ donation and transplantation.

UNOS and OPTN collect, organize and maintain data of statistical information regarding organ transplants in the Scientific Registry of Transplant Recipients (SRTR) database. SRTR collects data from hospitals, local Organ Procurement Organizations (OPO) and from OPTN (primary source of the data). It includes data from all transplantations performed in the United States. This information is used to set priorities and seek improvements in the organ donation process. Organs involved in the transplant programs are liver, kidney, pancreas, heart, lungs and intestines. SRTR seeks to provide information that is accurate, clear, and updated for use by the public, the department of health and human services, OPTN, UNOS, OPO, transplant programs, transplant candidates and recipients as well as living donors and their families.

In figure 2.1, the typical allocation model used by UNOS managing the OPTN and collaborating with the hospitals of the states is presented. Possible transplant candidates are placed in the waiting list and new candidates are added regularly. According to the candidates history, UNOS uses a system for prioritizing the allocation of the donated livers. This system uses statistical formulas - the Model for End-Stage Liver Disease (MELD) discussed in section 2.2 - to predict who needs a liver transplantation more urgently (Q-A for transplant candidates). After surgery, patients are monitored and their post-graft survival time is measured. In case of liver rejection, patients are placed back in the waiting list.

For this thesis, we requested Standard Analysis Files (SAF) about liver

Figure 2.1: UNOS allocation model.



transplantation from SRTR. SAF include most data fields collected by the OPTN for donors, candidates and recipients in the US from 1988 to present. However, due to the OPTN policy for timeliness of form submission, they contain information about liver transplantations up to 2015. The data have encrypted candidate - recipient and donor identifiers to maintain the privacy. SRTR offers online a SAF data dictionary, where format are readily available. [1]

The structure of the SAF datasets is illustrated in figure 2.2. The candidates table includes persons who are registered in the OPTN waiting list (to receive liver in our case). The information about the candidates comes from candidate registration and waiting list information collected by OPTN. One record is included per registration.

TX_{LI} (transplant information) data file contains one record per transplantation. It includes characteristics of each patient at transplant as well as the operation itself. Characteristics of the donor are added and donor-recipient interactions e.g. HLA mismatch scores, blood compatibilities, organ share.

Follow-up information (including characteristics) is provided in the table TXF_{LI} at 6 months, 1 year and annually later on post-transplant till recipient's re-transplantation, death or get lost to follow-up. This is again linked to the transplant records.

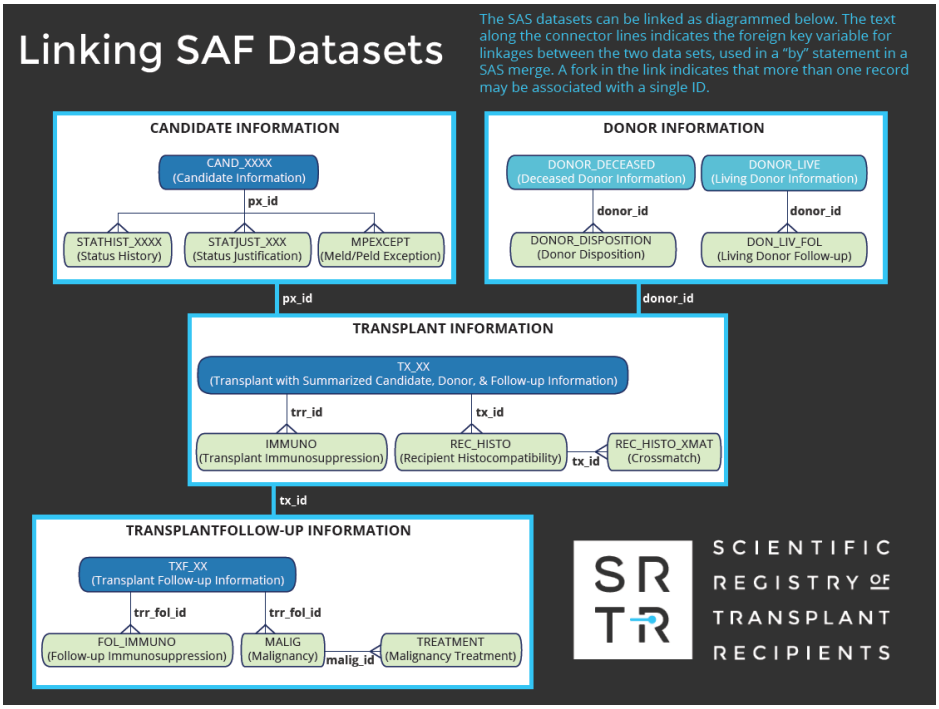
Our aim is to use the data for *general benefit* identifying variables that are significant risk factors for the patient's survival probability since transplantation, and build prediction models.

2.2 Variable pre-selection and data pre-processing

The data provided by UNOS include 62294 patients who underwent liver transplant surgery between 2005 and 2015. In figure 2.3, it is shown that around 5500 transplantations were performed each year. It is noticeable that since 2013 this number increased and in 2015 exceeded 6000 reaching a peak of 6199 transplantations. This may be related to the growing demand of liver transplantations in the USA, due to age

[1] Dictionary for variables details is provided at: https://www.srtr.org/requesting-srtr-data/saf-data-dictionary/.

Figure 2.2: How the standard analysis datasets are linked.



limits enlargement. However, it could also mean an improvement in the registration system.

One of the major challenges in the dataset was to find a proper way to select the subset of variables we are interested in. Overall, UNOS contained 657 variables in the Standard Analysis File (SAF) for both donors and patients (candidates and recipients). These regarded:

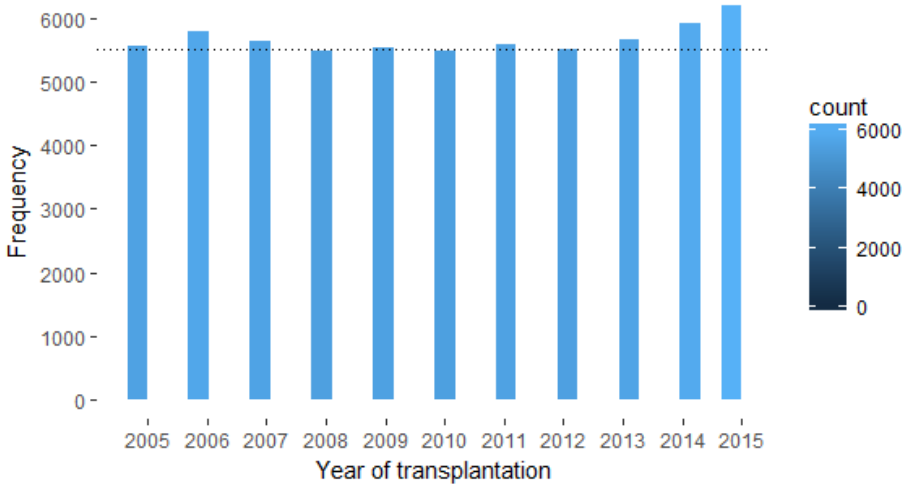
- **identification variables** like unique encrypted person id, unique encrypted donor id, candidate listing center, OPO serving transplant center.
- **important dates** such as transplant date, graft failure date, cohort censoring date, death date, graft follow-up date.
- **status variables**: death status (in 1, 3, 5 year and later), graft failure-free status (in 1, 3, 5 year and later).
- **demographic variables** such as age, gender, race, ethnicity, socioeconomic status and education level.
- **behavioral variables** e.g. smoking history, alcohol consumption, physical activity level, cocaine or other drug history.
- **physiological variables** for example blood type, etiology (cause of disease), laboratory measurements for arginine, serum creatinine, serum sodium, total bilirubin etc.

Variables that are potential risk factors may be used as covariates (confounders, explanatory variables) to estimate overall and graft failure-free survival post-transplantation.

Our main outcome variables (measures of clinical benefit) are:

- *Overall survival* (OS): time in years from the date of transplantation to death from any cause.
- *Graft failure-free survival* (FFS): time in years from the date of transplantation till graft-failure or death, whichever event occurs first.

Figure 2.3: Number of liver transplantations per year.



Making a selection of important variables from a list of around 600 is not a straightforward task. Therefore, variable pre-selection is based on the following factors: 1)clinical experts from Leiden University Medical Center, 2)percentage per variable missing, 3)its clinical importance. Furthermore, 4) redundancy (recoded variable for example age of the donor in years and in months), 5) superfluous-constant (numerical variable with one value or categorical variables with a single level) will be disregarded, 6) frequency of each categorical variable (we excluded or modified those with less than 1% of sample size at a level to create more balanced classes).

For choice 2, only variables with less than 40% missing data were imputed, as otherwise it would be infeasible to be reconstructed with plausible values based on the distribution of observed data and they would be noisy factors (Van Buuren et al., 1999).

Regarding number 6), it was decided to avoid extremely unbalanced and rare variables (classes of few observations are poorly distributed) that could not be a generalizable risk factor and would pose serious threats to our modeling procedure.

The variables were divided in two groups a) donor related variables and b) patient (candidate or recipient) related variables. To elaborate on this, for group b there were several variables referring to both candidates and recipients as for instance last encephalopathy, diabetes or hypertension status. For those preselected variables, we kept the ones corresponding to the recipients as they are more relevant for the project. Keeping the variables for the candidates would be more relevant when the focus is on the waiting list mortality.

Since 2001, the model for end-stage liver disease (MELD) was published as a prognostic indicator for adult patients with advanced chronic liver disease (Kamath et al., 2001). This model is a numeric scale ranging from 6 (less ill) to 40 (seriously ill),

used for liver transplant candidates aged 12 and older. It assesses the severity of chronic liver disease based on sound clinical and statistical validity, prioritizing allocation of liver transplants. It uses 4 routine lab test results: *bilirubin*, which measures how effectively the liver excretes bile, *INR* (prothrombin time), which measures the livers ability to make blood clotting factor, *creatinine* a substance which measures kidney functionality (severe liver disease is often associated with kidney impairment) and *serum sodium*, which assesses the severity of conditions like portal hypertension. These risk factors were included in the list of variables while MELD score was excluded, as it would confound variable selection of the prediction model.

The next pre-processing step was to simplify the levels of several polytomous categorical variables that would cause inconsistency in our analysis and would blur the functionality of Cox regression models. More precisely, there were several donor and patient variables with levels unknown (U) or even indeterminate (I), not done (ND) or pending (PD). We decided to remove these levels and include the small number of individuals (less than 1% and most frequently around 0.005 %) in the level "No". This makes the levels of the factors less complicated.

For the rest of the variables the following changes were applied:

- Candidate albumin was transformed from continuous to categorical with levels "(≥ 2)" and "< 2"
- Blood type for donor and patient levels A-A1-A2 were inserted to "group A", A1B-A2B-AB to "group AB", B to "group B" and O to "group O".
- For donor history of diabetes "No" remained as it is and "Yes, 0-5 years" - "Yes, 6-10 years" - "Yes, $>$ 10 years" - "Yes, duration unknown" were put to the category "Yes".

  Donors variables that were summarized following the same criteria are: history of cancer. For candidates: diabetes, encephalopathy status, angina and peptic ulcer and for recipients acute rejection episode.
- Donor's and candidate's race was changed from "Black", "White","Asian", "Multi", "Native", "Pacific" to just "Black", "White", "Other".
- Candidate's education was modified to levels "None/low/undefined education", "Medium education" and "Higher education".
- Recipient's functional status was simplified to 3 levels: "No assistance", "Some assistance" and "Total assistance" from 15 levels that existed originally in SRTR data dictionary.

In section A.1 of the appendix, two tables with the pre-selected variables for the donor and the patient are shown. A total of 106 risk factors, 52 covariates regarding donor characteristics and 54 for patient (candidate or recipient) characteristics. Out of those, 23 variables are continuous and concern demographics such as age, height, weight, body mass index as well as laboratory measurements e.g. hematocrit, serum sodium or warm ischemia time. The rest of the medical risk factors are categorical: dichotomous or less frequently polytomous. For brief explanations about medical terminology look appendix C. R-code for the data preparation is provided in the Appendix section D.1.1.

2.3 Descriptive Statistics

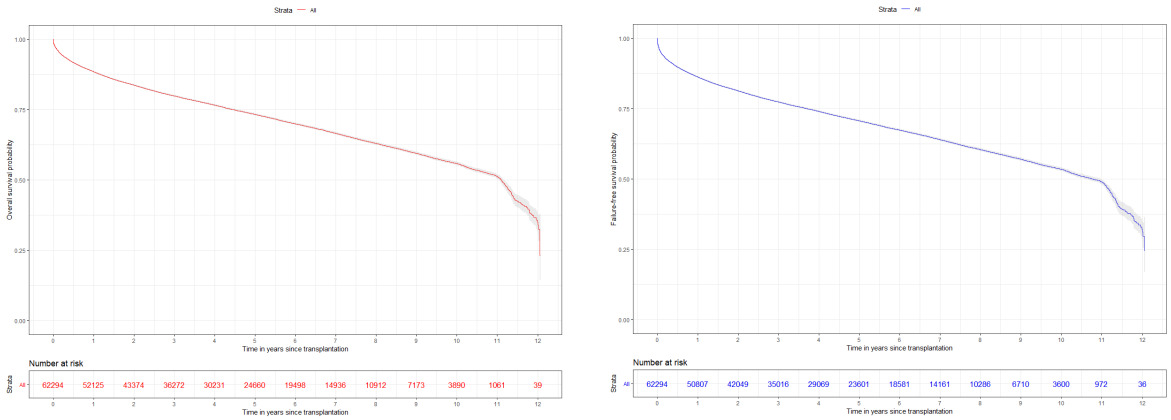
In this section we will discuss extensively the descriptive statistics that quantitatively summarize the outcome variables and our pre-selected risk factors. We will refer shortly to the numbers of missingness for each variable but we won't focus on the consequences of it as this is part of section 2.4.

For continuous variables, histograms of typical features such as age and BMI of donors and patients are shown, and we provide key measures of the 23 variable's central tendencies (min, 1st quantile, median, 3rd quantile, max and missing values) in table 2.1.

Categorical variables, that are the majority in our dataset, are presented using pie charts, bar-plots or by reporting their frequencies/proportions. 72 out of the 83 categorical variables are dichotomous and 11 polytomous.

Data collection included 60 Organ Procurement Organization (OPO) centers that took part in the recruitment of donors and 53 OPO centers, where candidates were listed for transplantation. Organ procurement and transplantation network (OPTN) has registered the death of 17431 unique persons starting from 5 January 2005 till 24 February 2017. Another important date is the candidate's cohort censoring time: 30 November 2016. Patients were followed up from the day of their transplantation until death or last contact. Graft failures were reported if occurred.

Figure 2.4: Kaplan-Meier survival curves for overall (left panel) and failure-free survival (right panel).



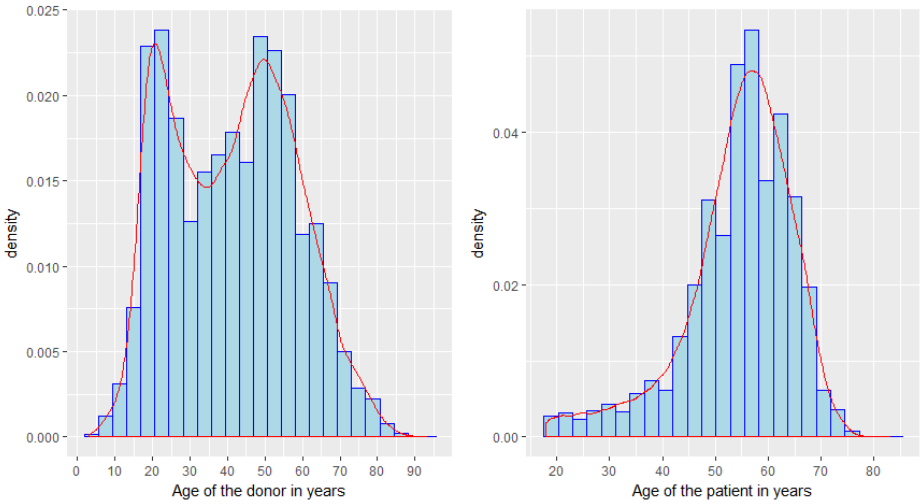
Overall and failure-free survival were estimated by applying Kaplan-Meier's (Kaplan and Meier, 1958) methodology (figure 2.4). Some patients in the study have been followed for more than 12 years.

Survival probability of liver recipients depends on a variety of complicated factors. Liver transplantation seems to prolong significantly the life of patients. Overall survival probability estimated from Kaplan-Meier method is 88.5% (88.2% - 88.7%), 79.9% (79.6% - 80.2%), 73.2% (72.9% - 73.7%) at 1, 3 and 5 years after surgery. Event-free (first event between graft failure and death) survival probability is 86.3% (86.0% - 86.6%), 77.4% (77.1% - 77.8%), 70.7% (70.3% - 71.1%) at 1, 3 and 5 years after surgery. Median overall survival time is 11.1 years (SD = 0.1) and median failure-free survival time is 10.8 years (SD = 0.2). There are 520 patients that died in the very beginning (same day as transplantation).

Histograms for age and BMI - separately for donors and patients - are shown in

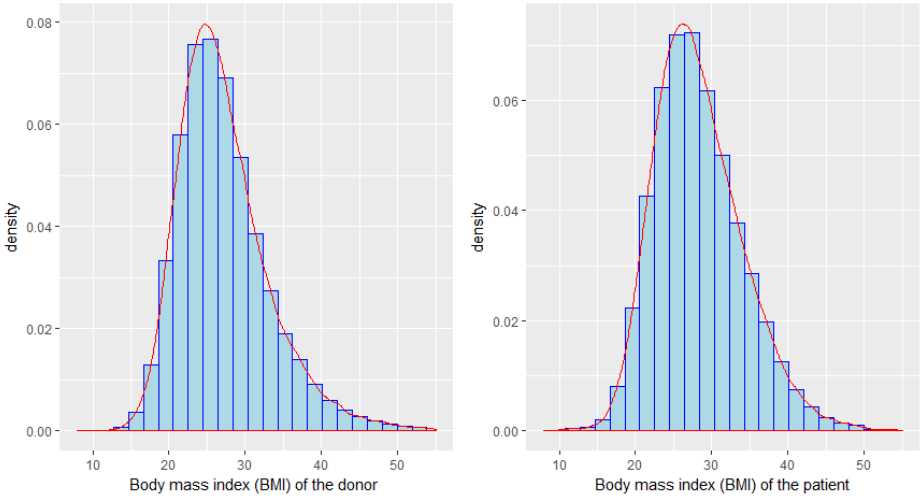
graphs 2.5 and 2.6. Age for donors range from 3 to 93 years, the distribution is bimodal with peaks around 20 and 50 years. The age for recipients of the grafts goes from 18 to 93, the distribution is normal with mean 54.3 years. None the liver donors in the dataset was alive. They are split in two groups "Donor after Brain-Dead" (DBD) and "Donor Circulatory Dead" (DCD). To the first category belong donors who are brain dead but their heart was still operating under mechanical support, whereas to the second category belong donors whose heart had stopped working during liver extraction. 59032 donors belonged to group DBD (94.8%) and 3262 (5.2%) to group DCD.

Figure 2.5: Histograms of age for donors and patients.



The distribution of BMI for donors and patients are very similar. Mean BMI is 27.3 and and 28.3 points [2] for donors and patients respectively.

Figure 2.6: Histograms of Body Mass Index for donors and patients.



All continuous variables are described in table 2.1. It contains extra demographic characteristics e.g. height, weight for donors and patients, and a series of laboratory

[2] Body Mass Index (BMI) is measured in kg/m^2.

Table 2.1: Descriptives for the continuous variables.

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	Missing
Donor age	3.0	26.0	42.0	41.3	54.0	93.0	0
Donor height	66.0	165.0	172.7	171.5	180.0	215.9	0
Donor weight	17.8	67.0	78.1	80.5	91.2	221.6	0
Donor BMI	9.6	23.1	26.3	27.3	30.3	79.0	1
Donor log-SGOT	0.1	3.3	3.8	3.9	4.4	8.9	53
Donor serum creatinine	0.1	0.8	1.1	1.5	1.5	26.0	21
Donor log-BUN	0.3	2.4	2.8	2.8	3.2	5.5	20
Donor total bilirubin	0.1	0.5	0.7	0.9	1.1	81.0	71
Donor log-SGPT	0.1	3.1	3.6	3.7	4.2	8.4	58
Donor serum sodium	101.0	141.0	147.0	147.2	153.0	200.0	69
Donor INR	0.1	1.1	1.3	1.4	1.4	94.0	1509
Donor hematocrit	0.3	27.0	30.4	31.0	34.5	75.0	231
Patient age	18.0	50.0	56.0	54.3	61.0	83.0	0
Patient cold ischemic time	0.0	5.0	6.3	6.8	8.0	48.0	0
Patient length of stay	0.0	7.0	10.0	16.0	17.0	739.0	5844
Patient BMI	7.5	24.2	27.6	28.3	31.9	72.9	1556
Patient warm ischemic time	0.0	30.0	38.0	41.1	48.0	240.0	17187
Patient last bilirubin	0.0	1.8	4.0	9.1	11.2	99.0	3
Patient last INR	0.5	1.3	1.6	1.9	2.2	99.0	3
Patient last serum creatinine	0.0	0.8	1.1	1.6	1.8	27.8	3
Patient last serum sodium	100.0	133.0	137.0	136.2	139.0	169.0	4
Patient height	15.6	165.1	172.7	172.1	180.3	225.0	1181
Patient weight	22.5	70.2	82.3	84.2	96.2	200.0	956

measurements. From these, the largest part has been mentioned repeatedly in the clinical literature for liver transplantations as prognostic factors (Wiesner et al., 2001); (Wiesner et al., 2003); (Kartoun et al., 2017), needless to say that bilirubin, INR, creatinine and serum sodium are fundamental ingredients used to estimate the candidate's MELD score (Q-A for transplant candidates). Length of stay is measured in days since transplantation and it is around 16 days for most of the patients. 22 patients remained hospitalized one year after their operation and only 3 patients 500 days.

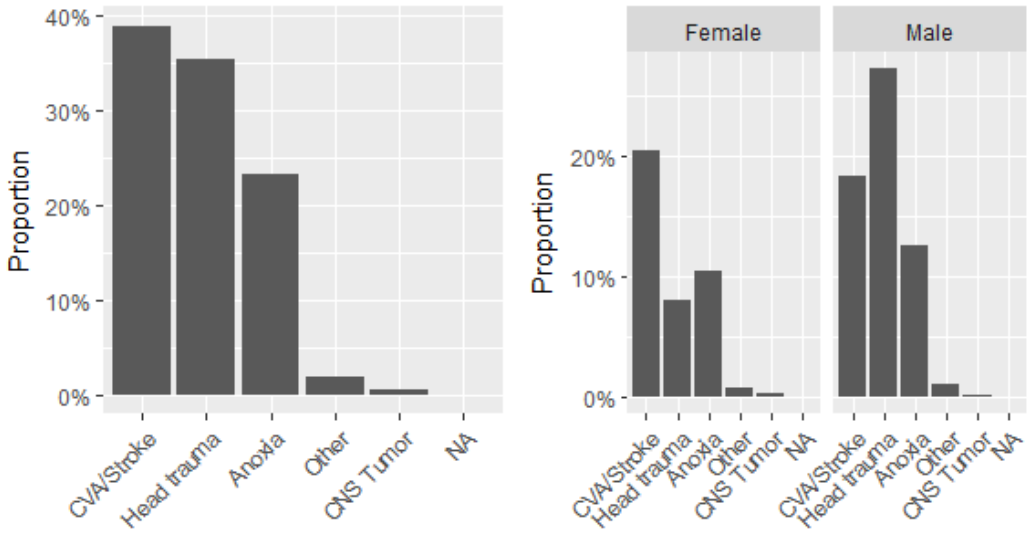
Subsequently, we present some bar-plots of polytomous risk factors. Bars of the total proportion are reordered in descending order.

We visualize the donor's cause of death in figure 2.7. Around 40% of the donors died from Cerebrovascular accident (CVA) or Stroke. Head trauma follows at 35% and then Anoxia with around 24%. Categories Other and CVS tumor are very rare. For the women, the most frequent category is CVA/Stroke whereas for the men Head trauma.

The most common cause of liver disease in total is Hepatitis C Virus (HCV) with approximately 28% then alcoholic close to 20%, other cirrhosis with 17% and malignant with a bit more than 11%. The rest of possible etiologies are in smaller percentages. HCV is the most frequent in men and other cirrhosis-HCV in women.

Additional information about other risk factors is shown below:

Figure 2.7: Bar-plots for donor's cause of death. Left panel: total proportion and right panel: proportion by gender.



- Donor blood group "O" 29876 persons (47.96%), "A" 23029 (37%), "B" 7381 (11.8%) and "AB" with 2008 (3.2%). Candidate blood groups: "O" 27964 (44.9%), "A" 22747 (36.5%), "B" 8427 (13.5%) and "AB" 3156 (5%).

- Candidate's education: "None/low/undefined education" 11562 (18.6%), "Medium education" 37884 (60.8%) and "Higher education" 12845 (20.6%) and 3 missing values. Education of the candidate may be related with the style of his life.

- Donor's number of transfusions at hospitalization: "None" including 34089 in total(54.7%), "1-5" with 19639 (31.5%), "6-10" with 5004 (8%) and "Greater than 10" with 3519 (5.6%) with 43 missing values.

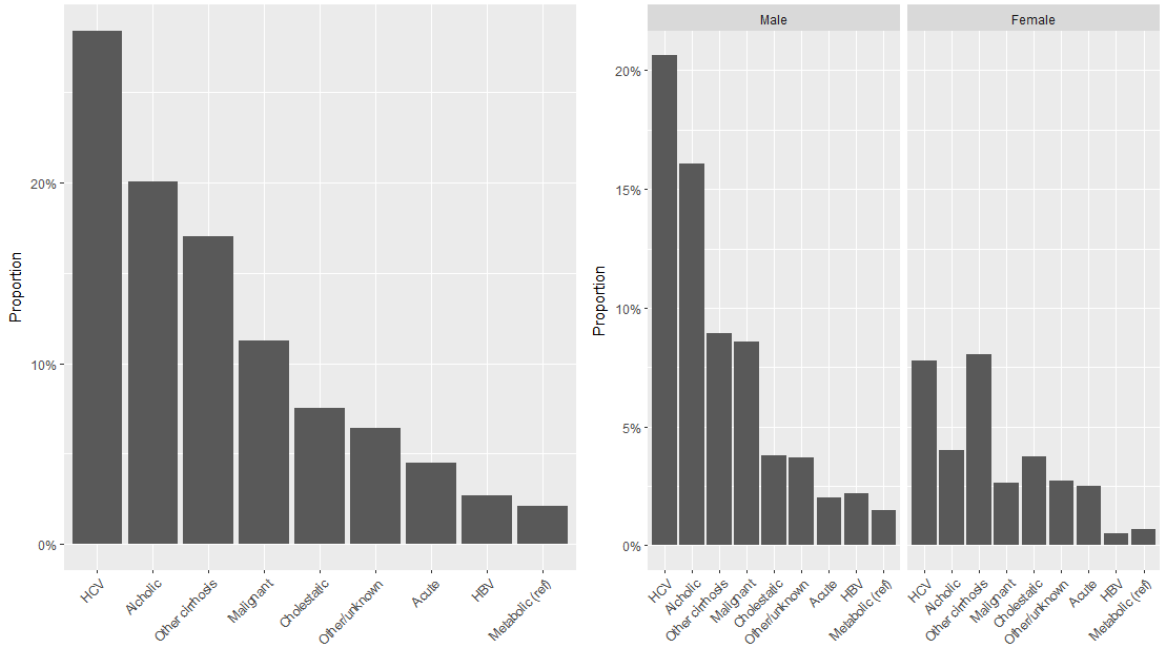
- Shared candidate's competition: "Local share" (71.3%), "Regional share" 14968 (24%) and "National share" 2924 (4.7%). It is expected that regional or especially national competition will play a role in the survival probability of the patients as local programs will be much more personalized.

- Recipient: pre-treatment status: "Not hospitalized" 41797 (or 67.1%), "hospitalized" 11762 (18.9%) and 8734 (14%) in "Intense care unit" (1 missing). Allocation is based on the MELD score and on the donor-recipient matches and that's why so many patients that underwent surgery were not hospitalized.

- Recipient/s functional status with the levels representing the patient's ability to perform daily activities such as changing cloths or going down the stairs. Three categories are provided to the data: "Some assistance" 23968 (38.5%), "Total assistance" 22122 (35.5%) and the least frequent "No assistance" 16203 (26%).

Pie charts for donor and patients for different factors are shown in Fig. 2.9 - 2.11. 49078 donors were White, 11232 Black and 1984 Other. On the other hand, 52468 patients were White, 6264 Black and 3562 Other.

Figure 2.8: Bar-plots for patient's cause of disease (etiology). Left panel: total proportions bar chart and right panel: proportions per patient's gender.



In figure 2.10, the proportions of male and female donors and patients is illustrated. From the donors, 37202 were male and 25092 female. The numbers are similar for the patients with 41968 being male and 20326 female.

Donor's and patient's ethnicities are shown in figure 2.11. 53829 donors were not Latino and 8465 Latino, whereas for the patients 53888 were not Latino and 8406 Latino.

In table 2.2 all the frequencies for "No-Yes" binary risk factors is shown.

Another binary group covariates with levels "Negative"-"Positive" have the following frequencies:

- Donor/s Anti-CMV: 21801 (35%) - 40491 (65%) with 2 missing.
- Donor/s Anti-HCV: 59959 (96.3%) - 2335 (3.7%).
- Donor/s HCV antibody status: 59866 (96.1%) - 2342 (3.9%) with 86 missing.
- Donor/s Hepatitis B core antibody status: 59067 (94.8%) - 3139 (5.2%) with 88 missing.
- Recipient: Pre-Treatment Serology Test Results Cytomegalovirus: 21821 (35%) - 39547 (65%) with 926 missing.
- Recipient: HBV surface antigen: 58311 (93.6%) - 3044 (6.4%) with 939 missing values.

Below frequencies of other risk factors are reported:

- Recipient's albumin "Greater or equal with 2": 58536 (94%) and "Smaller than 2": 3755 (6%) with 3 missing values.

Table 2.2: Binary risk factors with "No-Yes" levels.

Variable	No	Yes	Missing	Variable	No	Yes	Missing
don_prerecov_diuretics	24240	38051	3	enceph	22339	39951	4
don_ddavp	53562	8728	4	portal_vein	59558	2733	3
don_insulin	22991	39301	2	portal_hyperten_bleed	44789	1803	15702
don_inotrop_support	28957	33334	3	prev_abdom_surg	34342	27951	1
don_hist_cigarette_gt20_pkyr	46342	15951	1	ascites	43943	18346	5
don_hist_cocaine	53221	9071	2	last_dial_prior_week	54238	8056	0
don_hist_other_drug	40722	21570	2	diab	48271	14020	3
don_meet_cdc_high_risk	54436	7857	1	hcv	36654	25638	2
don_hist_diab	55188	6788	318	hcc	45530	16764	0
don_htn	40130	21812	352	malig	53670	3011	5613
don_hist_cancer	59859	2148	287	hbv	48427	12928	939
don_protein_urine	35603	26690	1	rec_work_income	53103	9055	136
don_anti_hbc	59085	3209	0	rec_immuno_maint_meds	1867	60426	1
don_prerecov_steroids	15892	46398	4	rec_acute_rej_episode	58756	3537	1
don_anti_convuls	54537	2667	5090	rec_tumor	45312	1874	15108
don_anti_hyperten	47066	15224	4	rec_bacteria_perit	42629	4853	14812
don_vasodil	53515	8775	4	can_variceal_bleeding	47669	2167	12458
don_heparin	3032	59260	2	rec_tipss	56947	5346	1
don_arginine	27606	34684	4	can_peptic_ulcer	46726	2000	13568
don_clinical_infect	28853	33438	3	can_angina_cad	44392	1324	16578
don_infect_blood	57026	5268	0	can_drug_treat_hyperten	36343	13025	12926
don_infect_lu	35049	27245	0	can_cereb_vasc	49500	437	12357
don_infect_urine	55160	7134	0	can_periph_vasc	49286	576	12432
don_heavy_alcohol	53082	9211	1	can_acpt_abo_incomp	61461	833	0
don_tattoos	42438	19854	2	can_acpt_extracorp_li	60896	1388	10
don_hla_typ	962	60485	847	can_acpt_li_seg	9078	53216	0
don_hist_prev_mi	59995	2295	4	can_acpt_hbc_pos	23613	38681	0
don_pulm_cath	55531	6762	1	can_acpt_hcv_pos	37292	25002	0
don_prev_gastro_disease	56468	4484	1342				

Figure 2.9: Pie charts for race of donors and patients.

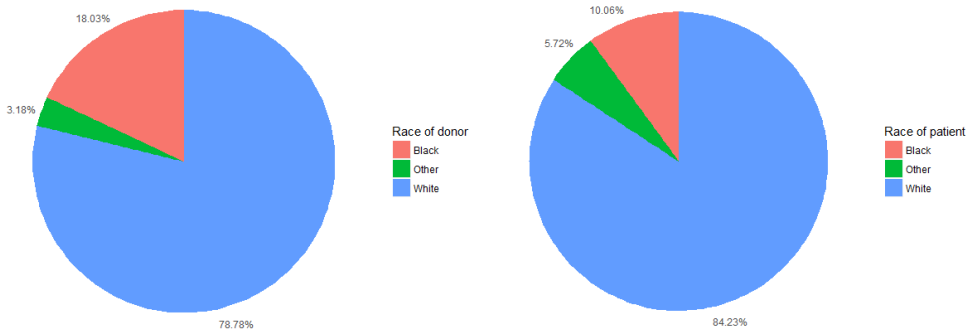
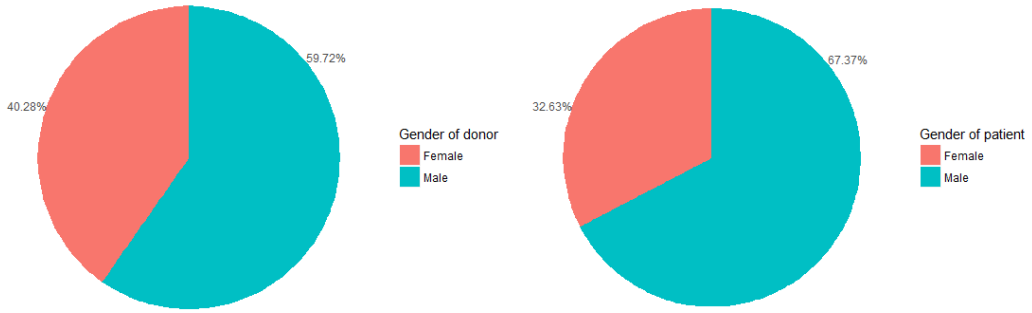


Figure 2.10: Pie charts for gender of donors and patients.

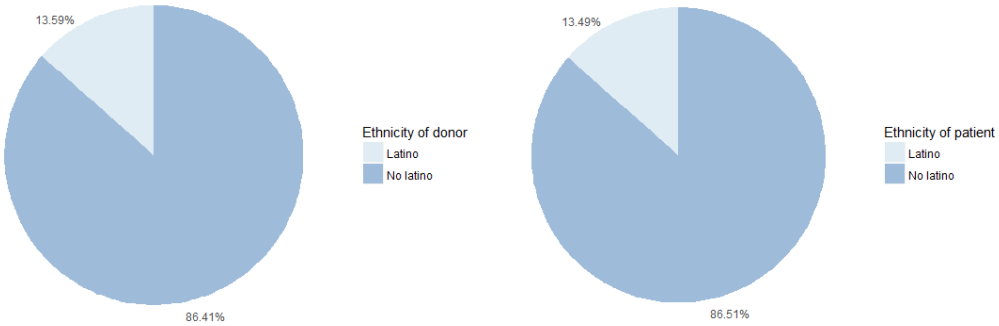


- Recipient: Split type "Whole organ": 61506 (98.7%) and "Split" 788 (1.3%).
- Patient on Life Support: "No life-support": 57192 (91.8%) and "Life-support": 5102 (8.2%).
- "Did the patient have a transplant surgery before?": "No re-transplantation": 58214 (93.5%) and "Re-transplantation": 4080 (6.5%).

2.4 Missing values - Imputation technique

In this section, we focus on the missing data and we discuss how to impute the missing values. They are common in medical studies as the gathering of the data is based on the doctors, expertized personnel, laboratory measurements but also on the patients themselves. If something goes wrong - for instance if the lab measuring device stops working or the doctor forgets to note down the finding or even if the patient does not

Figure 2.11: Pie charts for ethnicity of donors and patients.



show up in an appointment for personal reasons - this will result in another missing observation in the data set.

From UNOS dataset, we have pre-selected 2 outcome variable pairs (for overall survival and for failure-free survival) and 106 risk factors. As already discussed in section 2.2, we selected risk factors with less than 40% missingness.

The final dataset contains just 2.5% missingness overall. There are 14643 patients with at least 5 variables missing, 3776 with at least 10 missing and just 8 patients with more than 15 missing (only one of them had 36 variables missing).

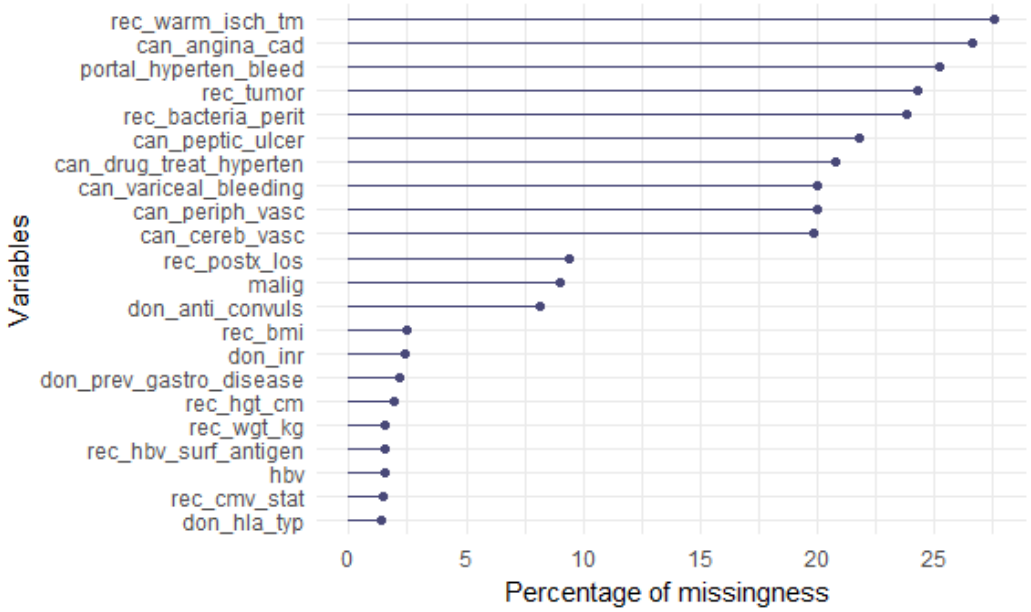
76 out of the 106 covariates had at least one unrecorded value. In figure 2.12 the percentage of missing values per risk factor is shown for the variables that had more than 1% missing observations. Warm ischemic time, which was not measured for 17187 patients (27.6%), is the variable with the highest number of missingness. The rest of the variables with more than 20% missingness are comorbidity: angina 16578 (26.6%), spontaneous portal hypertensive bleeding 15702 (25.2%), incidental tumor found at time of transplant 15108 (24.3%), spontaneous bacterial peritonitis 14812 (23.8%), peptic ulcer disease 13568 (21.8%) and drug treated systemic hypertension 12926 (20.7%).

Despite the fact that 76 variables have missing values, only 22 have more than 1% missingness and 10 more than 10% missingness. This shows that the amount of missingness in our data is actually small. Thereby, the imputation process will not affect severely the distributions of observed data.

Next, we investigated the missing data pattern because it influences the amount of information that can be transferred between variables. Imputation can be more precise if other variables are complete for those cases that are to be imputed. The opposite also holds. Predictors are more powerful if they are non-missing in rows that are - to a large degree - incomplete (Buuren, 2012).

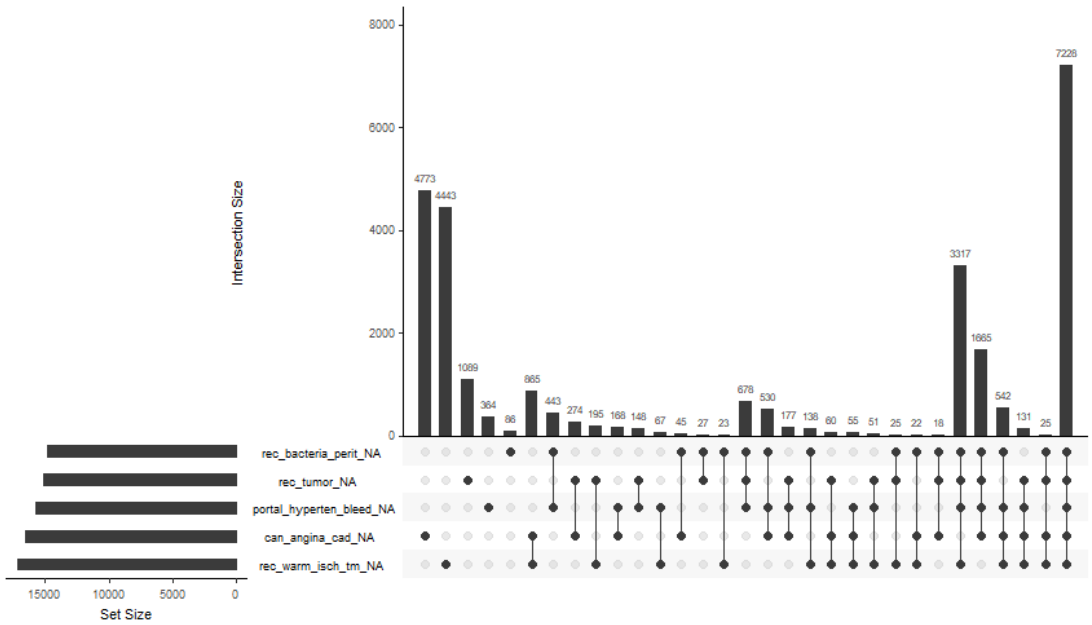
In the data, 27411 cases were complete. The rest of the cases contained at least one missing value. Due to the large number of variables we focused on the 5 with the largest percentage of missingness. These are warm ischemic time, angina, spontaneous portal hypertensive bleeding, incidental tumor found at time of transplant and spontaneous bacterial peritonitis; all referring to the patients. In figure 2.13 all possible combinations of missingness for these 5 variables are illustrated. The

Figure 2.12: Percentages of missingness per variable with more than 1% missing.



combination where of all of them are missing contains 7229 patients and is the most frequent group. The other most frequent groups are: angina missing, warm ischemic time missing and two combinations of 4 variables (the one without angina and the other without warm ischemic time). This suggests that imputation for warm ischemic time and angina may produce more accurate results compared to those for the other 3 variables.

Figure 2.13: Patterns of missing values for the 5 highest missing variables.



There are 3 mechanisms for missing values:

- Missing Completely At Random (MCAR): where the probability of being missing is the same for all cases.

- Missing At Random (MAR): the probability of being missing is the same only within groups defined by the observed data.
- Not Missing At Random (NMAR): the probability of being missing is different between variables.

For UNOS data, MCAR assumption is not so plausible because of the large number of cases and the different locations of the hospitals. Thus, we will assume that MAR holds. Imputation methods are generally applicable for this mechanism, but tend to be biased under the stricter NMAR assumption.

One strategy would be to proceed with complete case analysis of the 27411 patients, but this would mean a huge waste of data from the original 62294 rows. Furthermore, this may lead to invalid results (underestimation or overestimation of survival) if the excluded groups of patients represent a subsample from the entire sample (Van Buuren et al., 1999). For this reason, we decided to apply the imputation method to reconstruct the missing data.

As a part of a small sensitivity analysis, we explored the results of imputing missing data that we created artificially in complete cases dataset. Knowing the missingness on the entire dataset, we selected 5 variables that have many missing values 3 categorical: Spontaneous portal hypertensive bleeding (`portal_hyper_bleed`), Incidental tumor found at time of transplant (`rec_tumor`) and Drug Treated Systemic Hypertension (`can_drug_treat_hyper`) and 2 continuous: Warm ischemic time (`rec_warm_isch_tm`) and Length of Stay (`rec_postx_los`) as can be seen in graph 2.12. This process was repeated for 5, 10, 20 and 30% missingness on those variables. For the specification model we used the entire set of the 106 predictors, but also the 2 pairs of the response variables for overall and failure-free survival for higher precision and smaller bias.

We performed sensitivity analysis using 3 different R packages `MICE`, `AMELIA` and `randomForestSRC`. MICE (Multiple Imputation by Chained Equations) is one of the most commonly used packages for imputations (Stef van Buuren and Karin Groothuis-Oudshoorn, 2011). It uses fully conditional specification to impute incomplete multivariate data, assumes missing data are MAR, and predicts them based on the observed. Data are imputed variable by variable, specifying an imputation model for each one. By default, predictive mean matching is used for continuous variables and logistic regression for categorical.

AMELIA (James Honaker et al., 2018) imputes data using a bootstrapping based Expectation Maximization (EM) algorithm. It first creates a bootstrapped version of the original, it estimates key statistics by EM and then imputes missing data. The fact that it uses an EM algorithm makes it faster and robust to imputing. This package makes two assumptions about the data: 1) they have a multivariate normal distribution and 2) they are missing at random (MAR). The assumption of multivariate normality is a joint modeling approach.

Lastly, we tested randomForestSRC (Ishwaran et al., 2018). This package uses a lot of different algorithms for imputing missing data which are based on random forests. However, we decided to employ the original missForest algorithm as described by (Stekhoven and Bühlmann, 2012). This is a non-parametric imputation method that does not make explicit assumptions about the functional form of the data. It builds a random forest model for each variable. Then, it uses the model to predict missing values by using information based on the observed values. More specifically, a

Table 2.3: Imputation error with 10 percent missingness per variable.

	MICE	AMELIA	RANDOMFOREST
portal_hyperten_bleed	0.067	0.124	0.036
rec_tumor	0.080	0.128	0.039
can_drug_treat_hyperten	0.330	0.392	0.254
rec_warm_isch_tm	0.436	0.459	0.321
rec_postx_los	0.425	0.396	0.267

Table 2.4: Imputation error with 30 percent missingness per variable.

	MICE	AMELIA	RANDOMFOREST
portal_hyperten_bleed	0.071	0.114	0.034
rec_tumor	0.087	0.136	0.042
can_drug_treat_hyperten	0.338	0.395	0.252
rec_warm_isch_tm	0.764	0.755	0.546
rec_postx_los	0.734	0.732	0.523

forest of trees is grown to impute the data (150 trees were specified). Followingly, split statistics are calculated by selecting variables at random and from those a random subset is selected and defined as multivariate "pseudo-responses". A multivariate composite splitting rule is applied to each of the multivariate regression problems. The procedure is repeated until convergence of the algorithm. This is the most accurate of all random forests algorithms as all possible variable combinations are checked as responses, but also the most computationally expensive. More details about random forests can be found in chapter 3 section 3.3.

To assess the performance of the imputations, we used 2 separate measures. For the 3 categorical variables, the proportion of falsely classified entries (misclassification error) and for the 2 continuous variables the Normalized Root Mean Squared Error (NRMSE) defined as:

$$NRMSE = \sqrt{\frac{\mathrm{E}\big((\mathrm{X}_{\mathrm{true}} - \mathrm{X}_{\mathrm{imputed}})^2\big)}{\sigma^2(X_{true})}}.$$

This measure takes into account not only the mean difference between the true and the imputed continuous values but also their actual variance. Good performance benchmarks are for both measures values close to 0 and poor values close to 1.

The results for 5, 10, 20 and 30% missing data were similar. For concision, we show the results only for 10 and 30% missingness (tables 2.3 and 2.4). The missForest algorithm outperformed the other methods for both categorical and continuous variables resulting in smaller proportion of error. As expected, the errors were larger for the continuous variables as these kind of variables contain by nature a lot of information and is impossible to impute them by their exact value.

To recap, MICE uses full conditional specification whereas AMELIA uses a joint modeling approach. On the other hand, missForest does not make any implicit assumptions about data structures and usually is competitive/outperforms other methods of imputation especially in data settings where complex interactions and

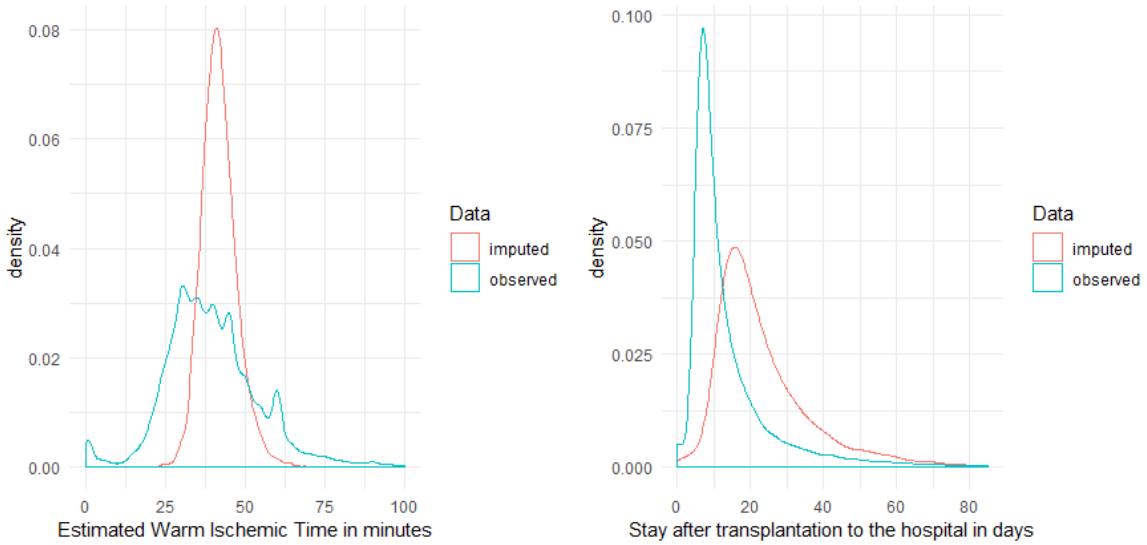
non-linear relations are suspected. Based on sensitivity analysis results we use the missForest exhaustive algorithm (reached the stopping criterion in 4 iterations) as our imputation method trading computationally intensity with accuracy.

We examined adding the Nelson-Aalen estimate of cumulative hazards H(T_1) and H(T_2) - instead of the overall and/or the failure-free survival times (T_1 and T_2 respectively) - together with the 106 covariates and the status variables because it can slightly improve the accuracy of imputations according to White and Royston (2009). However the correlation between H(T_1) and T_1 was 0.989 and between H(T_2) and T_2 0.988. Thus, it matters little for our data and we proceeded with the survival times.

We chose to impute our data only once and not multiple times because of the large number of observations (62294) and the very small amount of overall missingness (2.5%). Creating multiple data replicates would preserve the inherent uncertainty about them, but would be very computationally intensive and would complicate the variable selection as the modeling process would have to be repeated for each replicate for a set of 106 predictors. The imputation model accounts for the process that created the missing values and preserves the relations in the continuous and the categorical covariates imputed.

The last step of the imputation process involved inspecting the imputed data. We did this separately for the categorical and for the continuous variables. The percentage of missingness for variables with more than 1% missing is shown in figure 2.12.

Figure 2.14: Distributions of observed and imputed data. Left panel: "Warm ischemic time" and right panel: "Length of stay".



From the categorical variables, 12 dichotomous had more than 1000 missing entries. Most specifically variables:

- (From donor) Anticonvulsants (5090 missing) and Previous Gastrointestinal Disease (1342 missing).

- (From candidate) Drug Treated Systemic Hypertension (12101 missing), Variceal Bleeding within last two weeks (12458 missing), Peptic Ulcer Disease (13568 missing), Angina (16578 missing), Symptomatic Cerebrovascular Disease (12357 missing) and Symptomatic Peripheral Vascular Disease (12432 missing).

- (From recipient) Spontaneous portal hypertensive bleeding (15702 missing), Incidental tumor found at time of transplant (15108 missing), Spontaneous Bacterial Peritonitis (14812 missing), Pre-transplant malignancy (5613 missing).

Those variables except for Drug Treated Systemic Hypertension belong to very unbalanced classes with by far highest frequency at level No. For all these variables the imputation method yielded No as the best guessing value. For Drug Treated Systemic Hypertension the imputation process produced 12101 No and 825 Yes.

Only 5 continuous variables had more than 1000 missing observations. These were risk factors: Warm ischemic time of recipient (17187), Length of stay (5844), Donor/s INR (1509), BMI of recipient (1556) and height of recipient (1181). Summary statistics (min, 1st quantile, median, mean, 3rd quantile, max) of their distributions before and after the imputations were computed. The missForest algorithm imputed the missing values in such a way that the distributions of the imputed and the observed data are similar. In figure 2.14, the distributions of observed versus imputed data for Warm ischemic time and Length of stay are illustrated.

For Warm ischemic time, the imputed data were generated with a high frequency from the central part of the normal distribution of the observed values. For the variable "Length of stay", the imputed data were slightly moved to the right compared to the observed as the distribution of the observed data is right skewed, which affected the imputation process.

Chapter 3

Overview of the methods

In this chapter, an overview of the statistical methods for variable selection will be presented on 3.1 and 3.2 respectively. Then, Random Forests and Neural Networks are discussed in sections 3.3 and 3.4. The emphasis is on theoretical background, technical details, the connection with survival analysis, software implementation, suggestions in literature and interpretability of the models.

3.1 Cox Proportional Hazards regression

This section provides details about Cox proportional hazards model. In section 3.1.1 theory is presented. In section 3.1.2 the assumptions of a Cox model are illustrated, while in 3.1.3 the strategy regarding the coding of covariates is presented. Subsection 3.1.4 gives a review of some model selection strategies and 3.1.5 provides possible extensions in modeling when proportional hazards assumption is violated.

3.1.1 General framework

In survival analysis the focus is on the expected time duration till the occurrence of an event. This event is in this case death of the patient for overall survival (OS) or the first event between graft failure or death for failure-free survival (FFS).

The literature of survival analysis techniques is steadily increasing over years. However, when looking at non-parametric techniques there is a particular method that have been used for years and is the most popular.

The Kaplan-Meier estimator (Kaplan and Meier, 1958) - or product limit estimator- is a non parametric statistic that is used for the estimation of the survival function $S(t) = P(T > t)$. It is given by the formula:

$$\widehat{S}(t) = \prod_{i:\ t_i \leq t} \left(1 - \frac{d_i}{n_i}\right) \tag{3.1}$$

where t_i is the time when at least one event occurred and d_i the number of events at t_i.

This statistic is used to estimate the fraction of patients alive at certain time points after treatment. The log-rank test is commonly used to test for differences in survival times between two or more groups.

Kaplan-Meier curves and the log-rank test are cumbersome for quantitative predictors. In such cases the well known Cox Proportional Hazards (PH) regression

model is used (Cox, 1972). It is a semi-parametric model, as the component related to predictor variables is parametric while the estimate of survival function is fully non-parametric and doesn't make any assumption about the underlying distribution of survival times.

Suppose that data based on sample size n consist of the triple $(Y_j, C_j, X_j(t))$, $j = 1, ..., p$, with Y_j time on study for the j-th patient, C_j event indicator with $C_j = 1$ if event has occurred and $C_j = 0$ if life time is right-censored.

Let $(X_{i1}...X_{ip})$ be the values of the covariates of patient j. The hazard function of the Cox PH model with time-fixed covariates has the form:

$$\lambda(t|X_i) = \lambda_0(t) \exp(\beta_1 X_{i1} + \cdots + \beta_p X_{ip}) = \lambda_0(t) \exp(\beta^T X),$$

where $\lambda_0(t)$ is the arbitrary baseline hazard.

Using this model we can specify the hazard function at time t for each patient with a vector of covariates $X = (X_1, .., X_p)$.

Treating all patients as statistically independent of each other, we get the following partial likelihood:

$$L(\beta) = \prod_{i:C_i=1} L_i(\beta) \quad with \quad L_i(\beta) = \frac{\lambda(Y_i|X_i)}{\sum_{j:Y_j \geq Y_i} \lambda(Y_i|X_j)}$$

Taking the logarithm of it, the corresponding log-partial likelihood can be written as:

$$\ell(\beta) = \sum_{i:C_i=1} \left(X_i \cdot \beta - \log \sum_{j:Y_j \geq Y_i} \theta_j \right) \tag{3.2}$$

where $\theta_j = \exp(x_j \beta)$.

This function is then maximized over β to estimate the model parameters.

3.1.2 Model assumptions

Assumptions underling the Cox model are (Editors et al.):

- censoring of the observations is non-informative. This means that the mechanisms that cause the censoring of individual subjects are not related to the probability of an event occurring.
- proportionality of hazards (hazard functions between strata are proportional over time).
- linearity (additivity) in the effects of variables.

The proportional hazards assumption can be tested with the R function `cox.zph()`. A p-value smaller than 0.05 indicates non-proportionality between strata. For this project, having to consider 106 variables and a large amount of patient it is inevitable that violations of the assumptions will be present for a number of variables.

3.1.3 Covariates coding

For a Cox model, qualitative variables can be used in a regression analysis just as quantitative. However, more care need to be taken in the way they are coded and interpreted. There are many ways of coding qualitative variables. For dichotomous variables - like gender - it is straightforward and how the coding is applied is irrelevant.

For multilevel variables, there are more choices in the coding schemes. For our project to be consistent in all sections, we will consider a coding into a series of binary factors. To elaborate on this, supposing that a variable "X" has 3 levels "A", "B" and "C" with level "A" the reference, then this will be split into two new variables "X_B" and "X_C", where "X_B" is 1 if the patient belongs to category "B" or 0 otherwise and "X_C" is 1 if the patient belongs to category "C" or 0 otherwise.

The procedure as described above can be used for all 83 factors in the dataset. This coding scheme is employed in R by the `model.matrix()` function.

3.1.4 Model selection strategies

There are generally several approaches for model selection in Cox regression. They include two statistical ways in general: p-values and the Akaike Information Criterion (AIC). Collett (2014) suggests a model selection approach that includes 4 steps. This approach assumes that all variables are considered equally and can be described as follows:

1. Fit a univariate model for each covariate, and identify the predictors significant at some level p_1, say 0.20 using one of the global test criteria (Wald test, Likelihood ratio test or Score test).

2. Fit a multivariate model with all significant univariate predictors, and use backward selection to eliminate non-significant variables at some level p_2, say 0.10.

3. Starting with model identified in step 2, consider each of the non-significant variables from step (1) using forward selection, with significance level p_3, say 0.10.

4. Use stepwise regression with significance level $p_4 = 0.05$ to select the final model.

A widely used method for variable selection is AIC, which examines the likelihood and the number of parameters included in the model.

$$AIC = -2 \log L + kp \tag{3.3}$$

where L is the likelihood function, k is some predetermined constant and p is the number of covariates.

This statistic aims to balance the need for a model which fits the data very well to that of having a simple model with few parameters. Smaller values of the criterion suggest a better fit to the data. If a variable added to the model increases AIC, this indicates that the variable is unnecessary.

Both statistical techniques p-values and AIC can be used in automatic variable selection procedures such as forward selection, backward elimination or stepwise selection. Forward selection, starts with no predictors in the model, iteratively adds the

most contributive predictors and stops when the improvement is no longer statistically significant. Backward elimination starts with all predictors in the model (full model), iteratively removes the least contributive predictors, and stops when you have a model where all predictors are statistically significant. Last stepwise selection is a combination of forward and backward.

To select prognostic variables from Cox model automatic variable selection procedures were tested for UNOS data. AIC ignores stochastic errors inherited at the stage of variable selection and lacks of stability as discussed by (Breiman, 1996a) so different predictors may be selected in multiple runs of the procedures. For this reason, a numerically stable version of backward elimination will be applied.

3.1.5 Extensions of modeling

The Cox model can be extended to deal with violations of the proportional hazards assumption. This can be done in two ways:

1. Stratification, when the data within a stratum are expected to be relatively more similar to each other

2. Use more complicated models, inserting manually interactions between covariates and time and modeling non linear effects of variables by transformations or by expanding the design matrix to include basis functions (e.g. polynomial terms).

Using stratification, a different baseline hazard may exist for each stratum. A disadvantage here is that hazard ratios cannot be obtained for the variable used as stratum. Therefore, considering each variable regarding donor and patient in UNOS dataset as prognostic, stratification won't be used to correct for proportionality violations.

Moreover, having 106 variables it is unrealistic to assess the effect of pairwise interactions between them as this would mean testing $\binom{106}{2} = 5565$ possible interactions. Hence, to identify proper Cox models only the 106 prognostic factors will be used.

3.2 Feature selection for Cox with LASSO

In this section Least Angle Shrinkage and Selection Operator (LASSO) is discussed. Subsection 3.2.1 provides the theoretical background and in section 3.2.2 a small discussion is provided.

3.2.1 General framework

LASSO was introduced by Tibshirani (1996) as a method for estimation of regression models when many predictors are available. The method was expanded for variable selection in the Cox PH model (Tibshirani, 1997) with focus on fixed covariates.

The aim of LASSO is to minimize the log-partial likelihood given in (3.2) subject to the sum of absolute values of parameters bounded by a constant. In mathematical notation it can be written as:

$$\hat{\beta} = argmin[\ell(\beta)], \quad \text{subject to} \sum_{\mathrm{j=1}}^{\mathrm{p}} |\beta_{\mathrm{j}}| \leq \mathrm{s} \tag{3.4}$$

with s a user specified positive parameter.

The constraint $\sum_{j=1}^{p} |\beta_j| \leq s$, provides the advantage that some of the coefficients are shrunken exactly to zero. As a consequence LASSO is a tool for feature selection and construction of a parsimonious model.

Equation (3.4) can also be rewritten as:

$$\hat{\beta} = argmin_{\beta}\Big(\ell(\beta) + \lambda_{Lasso} \sum_{j=2}^{p} |\beta_j|\Big). \tag{3.5}$$

The quantity $\sum_{j=2}^{p} |\beta_j|$ is also known as the L_1-norm and performs regularization to the log-partial likelihood. The λ_{Lasso} is a non-negative constant that assigns the amount of penalization. Larger values for the the parameter mean larger penalty to the β coefficients and enlarged shrinkage towards zero. It can be shown that there is always a value in λ_{Lasso} such that all covariates are shrunken to exactly zero. Furthermore, $\lambda_{Lasso} = 0$ corresponds to the traditional Cox proportional hazards regression.

LASSO can be used to address the following question: "Which are the most relevant attributes to describe patient's survival?".

Tuning the hyper-parameter

As already mentioned the tuning parameter s in equation (3.4) or λ_{Lasso} equivalently in equation (3.5) plays a major role in the feature selection procedure. It is a controlling mechanism for the variance of the model. Higher values reduce further the variance but introduce at the same time more bias (variance-bias trade off). It also influences considerably the prediction performance of the model.

To find a suitable value for this parameter cross-validation needs to be performed. The goal is to minimize prediction error; here in terms of the cross-validated log-partial likelihood (CVPL) (Verweij and Van Houwelingen, 1993)

$$CVPL(s) = -\frac{1}{n} \sum_{i=1}^{n} \big(\ell(\hat{f}^{(-i)}(s)) - \ell^{(-i)}(\hat{f}^{(-i)}(s))\big). \tag{3.6}$$

This is usually done in 5 or 10 fold cross-validation. The exact strategy we followed will be discussed later on.

Tibshirani proposed an iterative quadratic programming process for tuning the hyper-parameter. The problem with this method was that for $p >> n$ the algorithm could not directly be applied. Later on the Least Angle Regression (LARS) algorithm became popular (Efron et al., 2004). This algorithm - having same order of magnitude of computational effort as ordinary least squares applied to the full set of covariates - can implement the LASSO with a simple modification.

The next step for the computationally efficiency of the parameter tuning was the L1-regularization path algorithm by Park and Hastie (2007) that selects variables according to the amount of penalization of L1-norm. This algorithm determines the entire path of the coefficient estimates as λ_{Lasso} varies $[\hat{\beta}(\lambda) : 0 < \lambda < \infty]$. Starting with $\lambda = \lambda_{max}$ that makes $\hat{\beta}(\lambda)$ non-zero, it computes a series of solution sets. By

generating the regularization path instead of computing the solutions at several fixed values of λ_{Lasso}, it identifies the order in which the variables enter or leave the model. An important finding here is that the L1-regularization is a smoother and less greedy version of the forward selection - backward elimination methods as mentioned in section 3.1.4.

The last step in the improvement of the optimization procedure was the use of cyclical coordinate descent computed along a regularized path (Friedman et al., 2010) which has the added ability of handling sparse features. This algorithm is implemented in the R package `glmnet`.

3.2.2 Things to consider

We conclude this section with some considerations about the LASSO. A benefit of it is that by penalizing the size of L1-norm of coefficients performs variable selection naturally. However, for a correct analysis it requires initial standardization of the regressors as well as a transformation of the categorical variables in a dummy coding style as described in section 3.1.3. The standardization can be done manually or is performed automatically by `glmnet` (Friedman et al., 2018).

A limitation in the prognostic factor selection of LASSO is that in cases that there are grouped variables (highly correlated one another) it tends to select one variable from each group ignoring the rest. This can affect the modeling procedure removing from the predictors variables that could be of prognostic value.

A possible extension to the LASSO is the elastic net that penalizes the log-partial likelihood with the L_1 norm and the L_2 norm at the same time, where $L_2 = \sum_{j=2}^{p}(\beta_j)^2$ is known as the Ridge penalty. This has the functionality of feature selection and shrinkage at the same time. However, for this project which aim is variables selection for building prediction models we will use only LASSO.

3.3 Random Survival Forests

In section 3.3.1 some theory is presented while in 3.3.2 the application to survival data is discussed. Details about implementation are provided in section 3.3.3. Results are discussed in section 3.3.4.

3.3.1 General framework

Random forests (RF) (Breiman Leo, 2001) are a learning method. It can be used for regression, classification, survival analysis and other statistical methods. The main idea of RF is to get a series of decision trees - which can capture complex interactions but are notorious as high variance methods - and obtain a collection of them averaging their characteristics. In this way, weak (unstable) learners are changed into strong learners.

Breiman (1994) introduced bagging (or bootstrap aggregation) as a machine learning algorithm to improve stability and accuracy for regression and classification. Bagging is a special case of averaging many noise models. Suppose that we fit a model to training data $Z = (x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N)$, obtaining prediction $\hat{f}_x$ for input

x. With bagging, we can average this prediction over a collection of bootstrapped [1] samples and reduce its variance. Fitting the model for each bootstrap sample gives a prediction $\hat{f^{*b}}(x)$ $(b = 1, \cdots, B)$. Then, bagging estimate is calculated as

$$\hat{f}_{bag}(x) = \frac{1}{B}\hat{f^{*b}}(x).$$

RF are essentially a modification of bagging building an army of de-correlated trees and averages them (Taylor, 2011). RF are quite popular due to their high level performance and flexibility. No assumptions need to be made during this ensemble process.

The construction of a general random forest can be described in the following steps (Hastie et al., 2001):

- Draw B times a bootstrap sample Z^* from the training data.
- Grow a separate tree of the forest T_b using the bootstrapped data. To grow the tree, the following 3 steps are repeated recursively: 1) Select m variables at random from p, 2) Pick the best variable/split point among the m 3) Split the node into two daughter nodes.

  The procedure is repeated until the minimum node size n_{min} is reached for each terminal node of the tree.
- Output the ensemble trees $(T_b)_1^B$.

Then predictions can be made in regression by averaging the ensembled trees and in classification by taking the majority vote of class predictions.

Decision trees - if grown sufficiently deep - have relatively low bias. Each tree generated with bagging is identically distributed. Therefore, the bias of bagged trees is the same as bias of individual trees. To reduce the bias, a sort of randomization procedure has to be implemented. Breiman introduced random selection of a subset of variables in the first recursive step of the algorithm. Trees are grown in an adaptive way to remove bias which at the same time reduces variance. Thus, the variance of RF is substantially reduced not only due to bootstrapped data sampling, but also due to random variable selection. Indeed, reducing the number of variables m will reduce the correlation between any pair of trees therefore reducing the average variance.

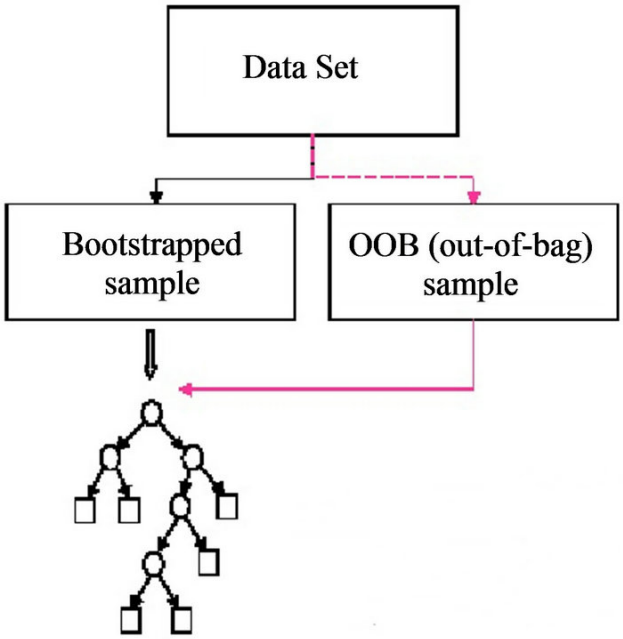
Out-of-bag samples

In this small sub-section a special feature of RF is illustrated. These machine learning techniques can collect out-of-bag (OOB) samples (Breiman, 1996b). During bootstrapping each tree leaves out about 37% of the cases. These left out cases can be used to form accurate estimates of important quantities. These estimated outputs are used to improve accuracy of the forests. This unique characteristic of RF makes them more appealing as the OOB samples can be used to obtain prediction error which is almost identical to the error estimate that can be obtained through N-fold cross validation.

Each tree is grown with the bootstrapped samples on the one hand, with leave one out cross validation being performed for the non-bootstrapped samples on the other hand (figure 3.1). OOB error estimates can be used to train the model and are good approximations of the generalization error.

[1] Bootstrapping is a statistical technique that falls under the broader heading of re-sampling.

Figure 3.1: Example of growing a decision tree. For each tree the sample is split into two parts the bootstrapped and the out-of-bag. This process is repeated B times to obtain an ensemble of trees.



3.3.2 Random forests for survival analysis

Random Survival forests (RSF) are random forests adapted for survival analysis of right censored data. Response is the pair (T_i, δ_i) with T_i survival time for the i^{th} patient and δ_i status indicator. RSF have inherited the appealing properties of RF. They are free of assumptions and can handle data with many predictor variables. Combining the predictions of many trees (averaging them), they show good prediction performance. Note that in case of growing a tree with a single variable it is equivalent to univariate Cox regression.

A general description of RSF algorithm is (Ishwaran and Kogalur, 2007):

Algorithm 1: Random survival forest algorithm

1. Draw **ntree** bootstrap samples from the original data
2. Grow a tree for each sample. At each node randomly select **mtry** predictors for splitting. Split on predictor using a survival splitting criterion. Each node is split on a specific predictor in such a way that maximizes survival difference across daughter nodes.
3. Grow the tree to full size. Constraint is that a terminal node (final node of the tree) should have at least **nodesize** unique deaths.
4. Calculate an ensemble cumulative hazard estimate combining information from the **ntree** trees (calculate one for each individual).
5. Compute an OOB error rate for the ensemble using B trees, $\forall B \in (1, \cdots, ntree)$.

Some questions that arise naturally for RSF are:
- how to define an appropriate node split for growing a tree?

- how to estimate the terminal node?
- what is the meaning of prediction?

We will give an answer to these questions in the following subsections. [2]

Two split rules are used in general to grow survival trees 1) log-rank splitting and 2) log-rank score splitting.

Splitting rule: Log-rank

Log-rank test for survival trees was introduced by Segal (1988). This rule is robust even if proportionality of hazards assumption does not hold. Given a node h, a proposed split on a real value x has the form:

$$x \leq c \quad \& \quad x > c.$$

The split on c will define the left and the right daughter node. The algorithm finds the split that maximizes survival difference between the 2 daughter nodes.

Let $t_1 < t_2, \cdots, < t_n$ the ordered death times in parent node h, denote by $d_{k,l}$ the number of deaths, $Y_{k,l}$ the individuals at risk in a specific time t_k in left daughter node for $k \in \{1, \ldots, m\}$. Similarly, $d_{k,r}$ and $Y_{k,r}$ for the right daughter node. Let $Y_{k,s}$ be the number of individuals alive at t_k in daughter $s \in (l, r)$ or those having an event at t_k.

$$Y_{k,l} = \#\{i : T_i \geq t_k,\ x_i \leq c\}, \quad Y_{k,r} = \#\{i : T_i \geq t_k,\ x_i > c\},$$

with x_i the value of x for individual $i = 1, \cdots, n$. Define $Y_k = Y_{k,l} + Y_{k,r}$ and $d_k = d_{k,l} + d_{k,r}$, n_s the total number of observations in daughter s. Thus, $n = n_l + n_r$, where $n_l = \#\{i : x_i \leq c\}$ and $n_r = \#\{i : x_i > c\}$. The log-rank statistic for split at value c for variable x is as follows:

$$L(x, c) = \frac{\sum_{k=1}^{m} \left(d_{k,l} - Y_{k,l} \frac{d_k}{Y_k} \right)}{\sqrt{\sum_{k=1}^{m} \frac{Y_{k,l}}{Y_k} \left(1 - \frac{Y_{k,l}}{Y_k} \right) \left(\frac{Y_k - d_k}{Y_k - 1} \right) d_k}}. \tag{3.7}$$

Node separation can then be measured by $(|L(x, c)|)$. Larger values are indicative of greater discrimination between the two groups and therefore a better split. The aim of the recursive algorithm at this point is to find the best variable x^* and split value c^* such that $|L(x^*, c^*)| \geq |L(x, c)| \quad \forall (x, c)$.

Splitting rule: Log-rank score

Hothorn and Lausen (2003) proposed the following splitting rule. The x variables randomly selected have been ordered so that $x_1 \leq x_2 \leq \cdots \leq x_n$. Compute the ranks for each survival time T_j. Define

$$a_j = \delta_j - \sum_{k=1}^{\Gamma_j} \frac{\delta_k}{n - \Gamma_k + 1},$$

[2] To be consistent with all mathematic notations, we use those proposed in the official website of software of implementation https://kogalur.github.io/randomForestSRC/theory.html.

with ($\Gamma_k = \#\{t : T_t \leq T_k\}$) the index of order T_k. The the log-rank score test is given by:

$$S(x,c) = \frac{\sum_{x_k \leq c}(a_j - n_l \overline{a})}{\sqrt{n_l \left(1 - \frac{n_l}{n}\right) s_a^2}}, \tag{3.8}$$

where $\overline{a}$ and s_a^2 are the sample mean and variance of $\{a_j : j = 1, \ldots, n\}$. The log-rank split ($|S(x,c)|$) is measure of node separation and the algorithm will maximize it over x and c to find the best split.

Estimates for terminal nodes

Terminal nodes are the final leaves of the trees. **Nodesize** parameter plays an important role when growing the tree towards the terminal nodes. It guarantees that the average node size across forest is at least nodesize. Another parameter that plays the role of regulator of tree growth is the **nodedepth** that defines how deep the tree should keep growing. It forces termination when depth reaches a pre-specified value. It can be seen how parameter nodesize affects the recursive algorithm 1.

Kaplan and Meier (1958) estimator is used to estimate survival of a terminal node. Let m be the distinct death times in the terminal node h. Let d_k be the number of deaths, Y_k the individuals at risk. Kaplan-Meier is defined as in (3.1). Then the cumulative hazard estimate(CHE) of the terminal node is given by the Nelson-Aalen estimator:

$$\hat{H}(t) = \sum_{t_k \leq t} \frac{d_k}{Y_k}. \tag{3.9}$$

For H terminal nodes in total in a tree, there are H estimates. Naming for each individual i the feature vector $\mathbf{X}_i$, the conditional CHE is

$$\hat{H}(t|\mathbf{X}_i) = \hat{H}_h(t), \text{ if } \mathbf{X}_i \in h.$$

We can specify the OOB estimator for all trees averaging the CHE statistic over all trees. To make this more illustrative, we define two more quantities: $\hat{H}_b(t|X)$ the cumulative hazard estimate for each tree b and

$$I_{i,b} = \begin{cases} 1 & \text{if } i \in OOB \\ 0 & \text{if } i \notin OOB. \end{cases}$$

Then the OOB ensemble CHE for individual i is as follows:

$$\hat{H}_e^*(t|\mathbf{X}_i) = \frac{\sum_{b=1}^{\texttt{ntree}} I_{i,b} \hat{H}_b(t|\mathbf{X}_i)}{\sum_{b=1}^{\texttt{ntree}} I_{i,b}}. \tag{3.10}$$

The quantity $\hat{H}_e^*(t|\mathbf{X}_i)$ can be used to get total OOB prediction error.

Conservation of events principle

The fundamental idea behind survival trees is the conservation of events principle (Ishwaran et al., 2008b). It is used to define ensemble mortality, a new type of predicted outcome for survival data. This principle asserts that the sum of estimated CHE over

time is equal to the total number of deaths, therefore the total number of deaths is conserved within each terminal node h.

$$\sum_{i=1}^{n(h)} \hat{H}_h(T_{i,h}) = \sum_{i=1}^{n(h)} \delta_{i,h}. \tag{3.11}$$

It can also be shown that that the total number of deaths is also conserved in a tree grown from the original non-bootstrapped data. Using this general principle a predicted outcome for mortality can be defined.

Measuring prediction error

A way to estimate prediction error in survival analysis is $E = (1-C)$, with C being the Harrell's concordance index (Van Houwelingen and Putter, 2011) $[0, 1]$ which measures how well the predictor ranks survival of two random individuals taking into account censoring of individuals.

Here we present the C-index in accordance with the random survival forests. If $t_1^*, \cdots, t_M^*$ are unique time points, the total predicted outcome of an individual (or else called *ensemble survival mortality*) is:

$$M_i = \sum_{k=1}^{M} \hat{H}_e^*(t_k^* | \mathbf{X}_i). \tag{3.12}$$

If the mortality estimate M_i of individual i is higher than this mortality for individual j then subject i has higher risk. Final error rate is $E = (1 - C)$. As benchmarks an error of 0 means perfect prediction and an error of 0.5 random guessing. We will give a summary of all predictive performance measures we use in the next chapter.

3.3.3 An overview about the existing software

Ishwaran et al. (2008b) have implemented Breiman's random forests in the R package `randomForestSRC` where SRC stands for survival, regression and classification (Kogalur U and Ishwaran H, 2018). Survival random forests were developed for right censored survival data. This is the best documented library regarding random forests for survival analysis. Package `randomForest` (Andy Liaw et al., 2018) for random forest in regression and classification lacks implementation for survival analysis.

The forest grows **ntree** times using the recursive binary algorithm as described in 1. The algorithm introduces randomization through bootstrapping at root nodes and randomizing selection of p out of P x-variables through parameter **mtry**. Resulting object in `randomForestSRC` can be extracted to make inferences. Due to its recursive nature iterations can be parallelized so that the trees grow in parallel. This can effectively reduce the computational time.

The main entry point is the function *rfsrc()* which can be used to training data and then predictions can be produced on new data using the *predict()* function. Overall error rate is returned if test data contain a y-outcome. Furthermore, the package has the ability to impute missing data in accurate ways as shown in section 2.4.

Tuning parameters

As already discussed, **mtry** is one of the hyper-parameters that can be tuned during training to achieve smaller prediction error. This parameter is the most fundamental as it controls an important part of randomness during the growth of several decision trees. UNOS data contain 106 covariates for both OS and FFS.

Suggested value for this parameter is $\lceil\sqrt{mtry}\rceil = \lceil\sqrt{106}\rceil = 11$. Small values of mtry mean that only a small part of the variables gets randomly selected each time. In the case of $mtry = 11$, each has a probability of $\frac{11}{106} = 10.4\%$. Therefore, it is possible that higher values may be needed to select a larger subset of the variables each time. As also proposed in Hastie et al. (2001), increasing the number of relevant variables does not compromise the performance of RF as they are robust in increasing the number of noisy variables.

Another important parameter for the topology of the tree is **nsplit** that allows the user to specify the split points number at which an x-variable is tested using one of the splitting rules (log-rank, log-rank score). Let $\mathbf{X}_i$ be the P-dimensional feature for case i such that $\mathbf{X}_i = (X_{i1}, \ldots, X_{i\mathtt{P}})$; let x the observed value for the p^{th} feature in the data set, where $p \in \{1, \ldots, \mathtt{P}\}$. Whether x is continuous or categorical a proposed split in node h is of the form $x \leq c$ and $x > c$. This defines left and right daughter nodes according to one of $\mathbf{X}_i$. A deterministic split on x demands considering all unique values of x at node h. When the variable is categorical the package treats the factors with an immutable 1:1 mapping of categories to integers. For example, a polytomous variable with 6 levels will be mapped into 6 integers. Split on a factor with f levels requires dividing them in two complementary subsets. A deterministic split like in the example requires examining all complementary pairing of levels with in this case will be

$$\binom{6}{1} + \binom{6}{2} + \binom{6}{3} = 41.$$

In general for the case of f levels the possible splits (complementary pairings) are $2^{f-1} - 1$. Factors with many levels increase a lot the computational complexity.

To avoid deterministic splitting that can be very inefficient, parameter **nsplit** with $nsplit > 0$ can be used to trigger a randomized selection of exactly nsplit points for each of the *mtry* variables within a node h. Another reason to use the *nsplit* parameter is the nature of our data. The vast majority of the variables are dichotomous or polytomous. Loh and Shih (1997) showed that decision trees tend to favor splits on continuous variables. As a consequence, we can use the *nsplit* parameter to reduce the amount of bias. The value chosen should be quite small.

We now discuss about strategy regarding two parameters of the package that play an important role in the topology of the growing tree (briefly mentioned in section 3.3.4). These are *nodesize* and *nodedepth*. *Nodesize* controls the average node size of the forest, whereas *nodedepth* forces termination of splitting in case a user specified value is reached. Setting $nodedepth = 0$ means defining a root node with depth zero (non-enlarging tree). *Nodesize* is a key parameter to achieve building a good model. Large values in *nodesize* will essentially force the forest to under-grow. This can cause problems with the accuracy. On the other hand, small values can also be a burden as each tree will keep growing on, and more and more noisy variables will get involved. Basically, pruning of a tree can be performed by increasing *nodesize* or decreasing *nodedepth*. Taking things discussed here into account, we will tune parameter *nodesize*

in a grid of points 20 to 120 to achieve smaller generalization error and we won't use parameter *nodedepth* as it is partially controlled by *nodesize*.

Overall the hyper-parameter tuning will be based on a 3-dimensional space containing parameters *mtry*, *nsplit* and *nodesize*. We can grow a forest varying the number of **ntree** trees so that we can find a number of parameters that will be large enough not to undermine OOB error.

3.3.4 Interpretation

In survival random forests variables can be selected based on their variable importance (VIMP). According to Ishwaran (2007) to calculate VIMP for a variable x, OOB cases are dropped down their in-bag survival tree. During splitting process use **random splitting** when x is encountered to create a daughter node. After creating the tree, CHE is calculated and averaged as in algorithm 1. VIMP for variable x is

$$VIMP = E_{randomized} - E_{original}.$$

With this intuitive method large importance values indicate variables with strong predictive ability. On the other hand, zero or negative values are indicative of non-predictive variables. Another interpretation can be given for VIMP under C-index. As the concordance index estimates the probability of correct classification between two cases, VIMP calculates the increase or decrease on test data in the absence of x.

However VIMP does not estimate the change in prediction error for a random forest with and without a variable, but the change in **test data** if x was not present, given original forest grown under x. An example to illustrate this consider a case when two variables are highly correlated and both predictive. Most probably both will have large VIMP values. In this scenario, dropping one variable and regrowing the forest might affect VIMP for the variable which was not dropped (value will increase). Though there is a high chance the prediction error will remain the same as the non-dropped variable is a good representative of the removed one.

VIMP can also be used to extract importance information for a group of x-variables. Interactions between variables can be identified. For UNOS data interactions between variables have not been taken into account as this would complicate the process and the computational intensity.

Another advantage of RSF is that variable selection can be performed with minimal depth - maximal subtrees (Ishwaran et al., 2010). Minimal depth is an order statistic that can capture the predictive of a variable in a tree. This method assesses the predictive value of a variable by computing its depth compared to the root node of a tree. Variables that have a smaller minimal depth value are more predictive.

Maximal subtree for a variable x is the largest subtree whose root node splits on x. There can be more than one maximal subtrees for a variable. Note that a maximal subtree may not exist if there are no splits on a particular variable. The shortest distance from root of the tree to root of the closest maximal subtree of x is the minimal depth of x. Mean value of the minimal depth distribution over all trees is used to decide if a variable's minimal depth is small enough to be regarded a good predictor. Contrary to VIMP, minimal depth is independent of the error rates. It only depends on the topology of forest trees.

Here, two different tools were discussed that can be used when many variables are present to provide interpretation: VIMP and minimal depth. Both methods have

the following two disadvantages. They cannot show if a variable has a positive or a negative effect and they refer to a variable as an entity (effects of factor levels are not provided) which weakens slightly their interpretability.

3.4 Neural Networks for survival analysis

In this section we introduce a neural network approach for UNOS. Subsection 3.4.1 provides the general framework that has been developed for neural networks; in section 3.4.2 some considerations about the models are outlined. In section 3.4.3 a short introduction tailored to survival data is provided while in section 3.4.4 the strategy applied in this thesis is discussed. Finally, subsection 3.4.5 illustrates the software used in this thesis and 3.4.6 tries to illuminate the "black boxes".

3.4.1 General framework

The progress that has been made in computational power in the 21^{st} century has allowed experimentation with *deep learning* which belongs to a broader family of machine learning. Deep learning methods are based on learning data representations through exposure to them. Artificial neural networks or neural networks (NN) are non-linear statistical models that belong to this family. The inputs are a number of features, then linear combinations of the inputs are extracted and the output (or target vector) is modeled as non-linear function of the features. These systems are inspired from biological neural networks that aimed at analyzing the human brain activity (van Gerven and Bohte, 2017).

A neural network is based on a collection of connected units which are called *nodes* or *neurons*. The units transmit signal to one another, then the signal is processed and transmitted to additional neurons connected to it. The connections between artificial neurons are called *edges*. Artificial neurons and edges have a weight which adjusts as learning proceeds. It increases or decreases the strength of the signal at a connection according to if it is positive or negative.

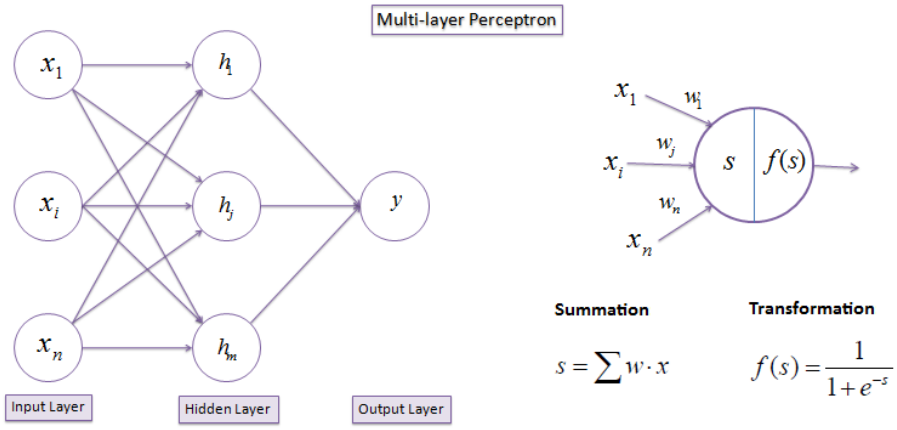
The most basic version of a neuron is a perceptron (Minsky and Papert, 1969). It is a type of linear classifier that makes its predictions based on a linear function combining a set of weights with the feature vector. Each input gets scaled up or down by a weight value assigned to it. All signals are summed up to a single value and an offset called *bias* is added as well. Afterwards a suitable transformation function is applied to them. A typical example of a perceptron is shown in figure 3.2. The transformation function (or most commonly called activation function) applied is the sigmoid. If the activation function is the identity $f(s) = s$ then the entire model collapses to a linear model in the inputs.

Neural networks have a layered structure. The input layer picks up the signals and passes them to the next layer that is called "hidden". A network may have more than one hidden layers that connect with the previous and transmit signals towards the output layer (last layer of the NN). The simplest form of a NN is the single layer feed-forward perceptron with the input layer, one hidden layer and the output layer. Usually the networks are fully connected meaning that every neuron in layer k is connected to all neurons in layer $(k+1)$.

Unlike traditional methods, NNs cannot be configured to work in a specific way. Knowledge is acquired from their environment through learning. The larger the

amount of data that we feed to the NN the better the model will learn. Inter-neuron connection strengths called *weights* are used to store the knowledge. We seek for values of them that perform well on the training data (Hastie et al., 2001). At the initialization step all nodes have small random weights and random biases. As the learning process goes on weights and biases are gradually shifted so that the next result is a bit closer to the desired output.

Figure 3.2: An example of a multi-layer perceptron. On the left side a typical topology of a single layer feed forward NN is illustrated whereas on the right side the perceptron.



Output transformations

When fitting a neural network, the output function $g_k(T)$ allows for a final transformation of the vector of outputs T. For regression, a common option is the identity function $g_k(T) = T_k$. For multi-class classification the *softmax* function $g_k(T) = \frac{e^{T_k}}{\sum_{i=1}^{m} e^{T_m}}$ is used. For a 2-class classification problem the *sigmoid* function $g_k(T) = \frac{1}{1+e^{-T_k}}$ has to be used.

Fitting neural networks

To estimate the unknown weights, a cost function is used according to the specified output(s). Denote the set of weights θ for a one hidden layer feed-forward neural network as follows:

1) $(\alpha_{0m}, \alpha_m) \quad m = 1, ..., M$ (p+1)M weights

2) $(\beta_{0k}, \beta_k) \quad k = 1, ..., K$ (M+1)k weights with p the number of features, M the number of nodes in the hidden layer and k the number of outputs.

The cost function of observations of data pairs (x, y) is defined as

$$C(\theta) = E[((f(x) - y)^2].$$

For regression the sum of squared errors is used as error function

$$C(\theta) = \sum_{k=1}^{K} \sum_{i=1}^{N} (y_{ik} - f_k(x_i))^2. \tag{3.13}$$

For classification we use the cross-entropy (or deviance) function:

$$C(\theta) = -\sum_{i=1}^{N}\sum_{k=1}^{K}(y_{ik} log f_k(x_i)). \tag{3.14}$$

The aim of the training of the network is to find the proper weights that lead towards the minimization of the cost function. The general approach to minimize $C(\theta)$ is by gradient descent (back-propagation using the chain rule). Some regularization to be added to the cost function is needed. More details will follow in this chapter.

Back-propagation

The back-propagation algorithm accelerated the training of multi-layer networks (Werbos, 1988). Back-propagation is a method that calculates the gradient of the loss (cost) function in a NN with respect to the weights. Mathematical representation of stochastic gradient descent algorithm is:

$$w_{ij}(t+1) = w_{ij}(t) + \eta\frac{\partial C}{\partial w_{ij}} + \xi(t) \tag{3.15}$$

where η is the learning rate and $\xi(t)$ is a stochastic term that adds some randomization to the procedure. As can be seen in (3.15) the weights of iteration (t+1) adjust according to the weights of the previous iteration with the use of chain rule in cost function with respect to the weights by the learning rate and adding extra term $\xi(t)$.

3.4.2 Aspects to be considered in the training process

1. The weights of the solution are related to the size of the prognostic factors. Therefore, scaling of the inputs such as standardization ensures the effective allocation of the weights. Categorical variables need to be coded in a dummy style as discussed in 3.1.3.

2. Neural networks depend on weights among the interconnected layers. For this reason, when the number of inputs is large the total number of weights can be massive as shown in 3.4.1 dependent on the number of hidden nodes M in the hidden layer. As a result, the model is over-parametrized and needs a substantial amount of training data to learn. Moreover, the optimization problem is non convex and the solutions may be unstable if the learning process is insufficient.

3. There are many learning algorithms that can be used in a NN. Some of them are gradient descent, Newton's method, Quasi-Newton method. They have numerous trade-offs, but almost any algorithm will work well with the correct hyper-parameters found on training. If the model, cost function and learning algorithm are selected appropriately - with trial and error - the resulting NN can become robust.

4. Number of hidden layers is one of parameters that can be tuned. More hidden layers offer greater flexibility to the network to find the non-linear relations that imitate the pattern of the data. There is no rule. Usually the choice is found through experimentation, even if training can be notoriously computationally intensive for many hidden layers.

3.4.3 Strategies for survival analysis setting

Neural networks are flexible and can handle big data without setting any constraints. NN might be more effective in complex analysis than the traditional Cox model. Their application has been extended to survival analysis over the years. Classification networks are often applied to prognosis problems, due to the lack of survival analysis software for neural networks.

Methods estimate either the survival probability $S(t)$ or the unconditional probability of death $p(t)$ whereas others estimate the conditional hazard $h(t)$ (or conditional probability of dying). In this thesis focus is on the application of feed-forward neural networks although generalizations such as convolutional NN where in hidden layers not all nodes communicate to each other are provided.

Regarding the error (cost) function, the cross entropy specified in (3.14) is the most popular choice due to its applicability in multi-class classification problems. A simplest version of the cross entropy function is the *binary cross-entropy* that has to be used in cases of 2-class classification to obtain model probabilities.

$$C(\theta) = -\sum_{n=1}^{N}(y_n log f(x_n) + (1-y_n) log(1-f(x_n))) \tag{3.16}$$

with θ the vector of weights and

$$f(x_n) = \frac{1}{1+e^{-\theta x_n}}$$

the sigmoid transformation function.

Topology of the networks is specified by input-hidden-output layers. In these methods there is a large amount of variation. Approaches can be distinguished on how the authors dealt with the censoring mechanism. Some networks have k output nodes - where k denotes k separate time intervals - while others have a single output node. According to Ripley and Ripley methods that deal with censoring use a different log-likelihood from that in standard software. Among methods that can be applied to standard software, Biganzoli et al. (1998) and Lapuerta P (1995) are the most satisfactory. Both require some preprocessing of the data to deal with censored data.

Method of Biganzoli

Before applying the methodology, data have to be transformed into a longitudinal format. Survival times needs to be divided into a set of k non-overlapping intervals which denote months or years. Each patient on the training set is replicated for the number of intervals that he/she is alive.

This approach models the conditional probability of dying $h(t)$ using $P(\textit{die in ith interval} \mid \textit{survive first i - 1 intervals, } x) = f(\eta_i)$, with $f(\eta_i)$ the sigmoid (logistic) function. The contribution to the log-likelihood can be expanded as $\sum log(f(\eta_i))$ over the intervals at which the specific patient is at risk. This is computed by having an additional input to the neural network specifying the time interval i for which $f(\eta_i)$ is required, and entering each patient into the training set for each time interval until death or the end of follow-up. The output node is one large target vector with 0 in case an event has not occurred in a particular time interval and 1 if the event did occur at this interval.

Biganzoli's method first estimates $f_i = P(t_i - 1 \leq T < t_i | T > t_i - 1)$ and then $S(t) = (1 - f_1) \cdots (1 - f_k)$. This method will be used in this thesis.

Method of Lapuerta

Before fitting a final NN, this method defines K separate networks with the purpose of imputing death periods for patients lost to follow up. In case a patient is right censored, he/she would be present to the networks corresponding to the intervals until the event is predicted. Assume the data are split into 5 time periods and an event can occur in periods 1, 2, 3 and 4 with period 5 representing no event. A patient censored in the first period will be presented in the 2nd network which will predict an event or not. In the case of no event the patient will be also presented in the third network and prediction will be repeated otherwise 0 will be imputed in the latter case. If a patient reaches period 5 without an event in all previous intervals a index of 1 is specified.

The aim of the imputation process on the censored observations is to create a K node output that can be trained as a multi-class classification problem. Lapuerta used sigmoid transfer functions for all the networks. With this method, the unconditional probability of death $p_i = P(t_{i-1} \leq T < t_i)$ is estimated and survival is estimated as $s_k = p_{k+1} = 1 - p_1 - \cdots - p_k$ 1 minus the cumulative probabilities across classes. Lapuerta's method is quite cumbersome as it needs multiple neural networks to deal with censoring. Biganzoli's approach will be applied in this thesis. It results in a single output node in the network topology and handles censoring in a more intuitive way.

3.4.4 Details about the implemented strategy

Feed Forward Artificial Neural Networks (FFANN) are equivalent to non-linear multivariate regression models. Particular cases of such a models with only input and output layer are equivalent to generalized linear models (GLM) with the logistic link function. Error terms are defined implicitly by the error function (section 3.4.1). It can be shown that FFANN with logistic outputs are non-linear regression models for conditional probability estimation (Bishop, 1996).

To implement this method which is connected with the theory of partial logistic regression, the survival times at K disjoint intervals $0 < t_1 < \cdots < t_k$ need to be split. Defining the survival function $S(t_k) = P(t > t_k)$, the discrete hazard rate (also called the conditional failure probability) is defined as:

$$h_k = \frac{S(t_{k-1}) - S(t_k)}{S(t_{k-1})}. \tag{3.17}$$

From equation (3.17) the related survival function can be obtained

$$S(t) = \prod_{k:t_k \leq t} (1 - h_k) \tag{3.18}$$

The contribution to the likelihood function of the *i-th* patient will be given by the product of conditional survival probabilities for the time intervals in which he/she is observed and the conditional failure probability in the interval in which the event of interest occurs.

Let d_{ik} be the censoring indicator equal to 1 in interval A_k, representing the event of interest or equal to 0 otherwise, the likelihood can be written as

$$L = \prod_{i=1}^{n} \prod_{k=1}^{k_i} h_{ik}^{d_{ik}} (1 - h_{ik})^{1-d_{ik}}. \tag{3.19}$$

By assuming that the observations in each time interval are independent across intervals, the discrete time model can be fit. Considering the Cox proportional hazard regression model as

$$\frac{h_k(x_i)}{1 - h_k(x_i)} = \frac{h_k(0)}{1 - h_k(0)} e^{\beta^T x_i}$$

where $h_k(0)$ is the baseline hazard and defining $\theta_k = log(\frac{h_k(0)}{1-h_k(0)})$ the model can be rewritten as:

$$h_k(x_i) = \frac{exp(\theta_k + \beta^T x_i)}{1 + exp(\theta_k + \beta^T x_i)}. \tag{3.20}$$

The partial logistic regression model with neural networks (PLANN) can be achieved with modeling of the joint dependence of hazards from time t_k and the covariate vector x_i. Taking the negative logarithm of equation (3.19), it is possible to obtain a equation that is equivalent to the binary cross entropy error function (3.16). Using this error function in a NN with no hidden nodes and logistic activation function, a linear logistic regression model such as in (3.20) is obtained. Target variable is represented by d_{ik}.

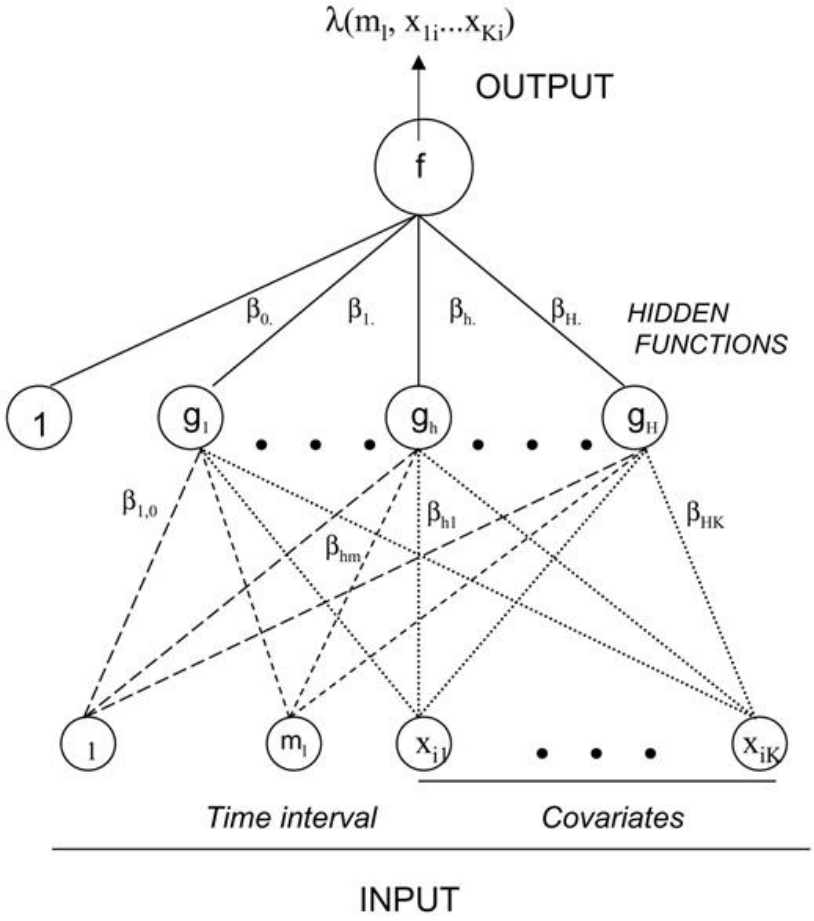
A generalization of the linear regression model for NN is the partial logistic regression model (PLANN) by the addition of a hidden layer with neurons and 1 output node. The PLANN model can be viewed in form of a diagram in figure 3.3. The input layer includes the prognostic factors (covariates), the time interval m_l and the bias unit (1). Hidden layer has H units plus a bias unit. Finally, the single output unit computes smoothed estimates of discrete hazard rates $\lambda(m_l, x_{1i}, \cdots x_{Ki})$. For the original PLANN method Biganzoli used the logistic activation function for both the hidden and the output layer (functions g and f).

Note that PLANN is not constrained to proportional hazards assumption as the interaction of time and covariate effects are modeled implicitly. Some very useful characteristics of this modeling technique is that output can be used to explore the shape of the hazard function or to find personalized survival predictions for the patients if ids are assigned to them.

It is expected that the time interval will play an important role in the modeling process as later time intervals would increase the hazard of death of graft failure for the patients. The data set used in this thesis contains only time-fixed covariates. Then replicating patients for each time interval means that the only variable that will be different between two rows corresponding to the same patient will be the interval. For time-dependent covariates, use of the PLANN strategy would be straightforward as data need always to be in a longitudinal long format with each row being represented by a time interval for each patient.

In this thesis the PLANN method will be implemented in the R software and more specifically in the state-of-the-art, powerful and flexible `keras` library (JJ Allaire and François Chollet, 2018). The number of parameters will be tuned using 5-fold cross validation in order to achieve high accuracy and a good generalization performance. A separate subsection will focus on explaining the library capabilities, possible parameters to tune and ways of evaluating the prediction error. As we

Figure 3.3: Visualization of the feed forward PLANN model. Dashed lines: connection between nodes and input-hidden layer. Solid lines: connection between nodes and hidden-output layer.



mentioned in 3.4.2, all prognostic factors must be standardized beforehand to allow for algorithm convergence.

Data transformation

To use the PLANN method the training data must be transformed from the wide format to the appropriate long format. Say that we have a short format of our data such as in table 3.1 where we have created a variable *interval* that splits the survival time variable (*patientsurvival*) in 12 distinct intervals (1,2,...,12) each one representing a year since transplantation. Patient 1 will belong to the fifth interval and he is censored (death = 0) and patient 2 to the 3rd interval and died at this point (death = 1).

To create the training data in a longitudinal format, transformation will be applied based on the interval variable. Each row on the long format corresponds to a distinct year. Last interval will be the one observed in wide format. An example is given in table 3.2. We have created a variable status that is 0 when the patient did not die on this interval and is 1 if the event occurred on that particular interval. Thus for example, patient 1 who is censored at interval 5 will be present for the intervals 1, 2, 3, 4 and 5 and patient 2 will be present on 3 intervals 1, 2, 3 with status 1 in the third.

For the test set as the status of each patient is supposed to be unknown, a patient

Table 3.1: Example of UNOS data in wide standardized format for overall survival.

don_genderM	donorage	enceph	hbvY	patientsurvival	interval	death	id
1	-0.554	0	0	4.98	5	0	1
0	0.162	1	1	2.87	3	1	2

Table 3.2: Example of UNOS data for overall survival in long standardized format for the training set. Variable status shows if a patient is alive (0) or dead (1) at a particular interval.

don_genderM	donorage	enceph	hbvY	interval	status	id
1	-0.554	0	0	1	0	1
1	-0.554	0	0	2	0	1
1	-0.554	0	0	3	0	1
1	-0.554	0	0	4	0	1
1	-0.554	0	0	5	0	1
0	0.162	1	1	1	0	2
0	0.162	1	1	2	0	2
0	0.162	1	1	3	1	2

has to be replicated for all 12 intervals. All variables have to be standardized with the categorical transformed as indicators. Id vector can be used for each patient to obtain personalized hazard probabilities and from them individual survival probabilities.

After transformation of categorical variables (as described in chapter 2.3) into indicators and standardization of the continuous variables in the final dataset there are 128 prognostic variables. Adding the interval variable *in total 129 variables* will be used as inputs to the neural network. The same procedure is applied to the dataset for overall survival and for failure-free survival. The long format of the two datasets will be different as FFS is defined as the first event between graft-failure and death.

3.4.5 Software of implementation

We will perform analysis in the R library `keras` (Chollet et al., 2015) which is an interface for the original neural network library written in Python programming language (Python Core Team, 2015). It runs on top of Tensorflow (Abadi et al., 2016) which is a symbolic math library used for machine learning. `Keras` contains implementations of commonly used neural network building blocks such as layers, objectives, activation functions, optimizers. One of the main advantages of the package is that it allows the use of distributed training of deep learning models on clusters of graphics processing units. Therefore, even if the dataset used in this thesis is large in terms of rows and columns, the library `keras` can deal with much larger datasets and has gained increasing attention in the last years for deep learning applications.

In subsection 3.4.1 NNs as layers of perceptrons together and a multilayer perceptron model were shown in figure 3.2. The input layer take the feature inputs, connects them to hidden layers through proper weights and hidden layers connect them to output layer.

Before using `keras`, the data have to be normalized and split into training and test sets. The exact way of formulation of training and test data is shown in section 3.4.4. The model has to be fine-tuned so that it performs better in unseen data. Data to be passed to the *fit* function need to be in a matrix format. Details about the cross-validation will be discussed in chapter 4.

To construct the network, we need to initialize a sequential model that is the multi-layer perceptron. There are many activation functions available for the hidden and output layers. For this project we focus on: 1) The sigmoid (logistic) activation function

$$f(x) = \frac{1}{1 + e^{-x}}, \tag{3.21}$$

2) the hyperbolic tangent (tanh)

$$f(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{3.22}$$

and 3) the rectified linear unit (ReLU) defined as

$$f(x) = x^{+} = \max(0, x) \tag{3.23}$$

ReLU function is defined is the positive part of its argument, where x is the input to a neuron. ReLU has been demonstrated to enable better training of deeper NN, compared to a wide set of other possible activation functions (Glorot et al., 2011). Biganzoli used the sigmoid activation function for both the hidden and output layer (Biganzoli et al., 1998). It is really interesting to study how the most popular modern transfer function will perform for the activation of hidden layers. For the output layer, the sigmoid activation function must be used as so as to ensure that the output of the NN is smoothed hazard probability (3.20).

After manually setting the architecture of the model (input-hidden-output layers), the estimation is performed using the most suitable loss function. Here, we specified the binary cross-entropy of equation (3.16). Another flexibility of the package is that it offers a large variety of optimizers, Adam (Adaptive Moment Estimation) which uses stochastic optimization, Stochastic Gradient Descent (SGD). This algorithm is also known as incremental gradient descent and it is a stochastic approximation of gradient descent (3.15). It is called stochastic because samples are selected randomly (or shuffled) in the order they appear in training set.

Having built the basic components, the model can be trained for a number of different epochs (number of times the algorithm sees the entire data) and set the batch size (the number of training examples in one backward or forward pass). We set the default number of epochs at 10. For the batch size we have tried different values throughout the process since it is related to the memory space needed. Setting higher batch sizes accelerates training.

Tuning parameters

`Keras` library is very flexible and there are a lot of options for hyper-parameters to tune. To avoid expanding in a vast grid of point combinations, we selected 5 parameters to tune.

- *Node size* of hidden layer(s). This parameter is one of the most fundamental as it defines the number of weights the network will have and consequently the

amount of complexity of it. Having 129 inputs in total the optimum node size parameter will be in the range 5 - 130 so that the network is able to transfer signals to the single output node in the most efficient way.

- *Dropout rate* of neurons. Over-fitting is a serious problem in NNs. Especially deep neural networks with multiple hidden layers face this problem very often because of the very big number of training weights in comparison of the sample size. Dropout rate is a technique that can deal with this problem (Srivastava et al., 2014). Using dropout in the `keras` interface, we randomly drop units (together with their connections) from NN during training by randomly selecting nodes to be dropped-out with a given probability. This procedure prevents units from adapting too much and offers major improvements over other regularization techniques. We included 4 different values of dropout rate 0.1, 0.2, 0.3 and 0.4. Note that the parameter is only used during training of a model and not used when validating it.

- *Learning rate* of SGD. This is parameter η in (3.15) also declared as step size of weight iteration. Some suggested values to use here are 0.001, 0.01, 0.1.

- *Momentum* of SGD algorithm. This method is often better and faster than simple SGD. It helps accelerate gradient vectors in the right directions, thus leading to faster converging. Momentum is a moving average of the gradients. An intuitive way to write SGD with momentum is

$$V_t = \beta V_{t-1} + \alpha \nabla_w C(W, X, y), \quad W = W - V_t \tag{3.24}$$

  with C the error function, α the learning rate and β the momentum, W the weights vector and V_t a new sequence. Some values suggested by experts to use are 0.8 and 0.9.

- *Weak class weight*. This is a parameter that can be used in case of unbalanced classes in `keras`. In the data set only 29% has experienced the event of interest for OS and 31.3% for FFS while the rest are censored. Considering this unbalance, we double the weight the minority class (or consider weight 1).

More things that can be tuned are the activation functions of hidden layer(s), optimizers etc.

3.4.6 Interpretability of the networks

After finishing the tuning process (through cross-validation), we can fit the final trained model and predict hazard probabilities on the test data. This can be done in `keras` with the built-in function *proba*. Then the survival function for each individual can be obtained from the hazard rates by using (3.18). Personalized survival probabilities can be plotted.

For single hidden layer FFANN, the algorithm developed first by Garson (1991) and later by Goh (1995) - provides information about the weights mechanism. The idea behind the algorithm is that inputs with larger connection weights produce greater intensities of signal transfer. As a result, these inputs will be more important for the model.

In general, positive connections indicate increasing effects on neurons and negative decreasing. Garson's algorithm can be used to determine relative importance of each input variable, partitioning the weights in the network. The absolute values of connection weights are used to specify % of importance.

Algorithm 2: Garson weights algorithm

Input: input-hidden , hidden-output connection weights

Output: relative importance R_j

1 $\forall$ hidden neuron $i = 1, ..., m$, multiply the absolute value of the hidden-output layer connection weight by the absolute value of the hidden-input layer connection weight. Repeat $\forall$ input $j = 1, ..., p$ to obtain P_{ij}

2 $\forall$ i, $Q_{ij} = \frac{P_{ij}}{\sum_{i=1}^{m} \sum_{j=1}^{p} P_{ij}}$

3 $\forall$ j, $S_j = \sum_{i=1}^{p} Q_{ij}$

4 $\forall$ j, $R_j = \frac{S_j \cdot 100}{\sum_{j=1}^{p} S_j}$

The algorithm has the drawback that it does not provide the direction of relationships so it remains unclear if the relative importance indicates a positive or negative effect. However, it can be a useful tool to identify important variables and check the stability of the weights. To elaborate on this, NNs contain an inherit amount of randomness being optimized stochastically. This means that it is possible to repeat the prediction with the same or other highly performed hyper-parameter combinations to check the stability of the network in predicting relative importance of the prognostic factors.

For NNs with multiple hidden layers the well known "black-boxes" is appropriate, as the algorithm is not implementable and extracting useful insights from them may be a tall order except from another approach is used to illuminate them. Some strategies do exist although the implementation for datasets with many inputs is a major problem.

Chapter 4

Predictive performance measures

An essential question in survival analysis is "how can we predict risk of future events?". To answer this question, the common strategy is to build several risk prediction models applying different methods. In chapter 3, we described different approaches that can be used: Cox regression models, Neural Networks and Random Forests. All methods have trade-offs between model assumptions, interpretation and computational efficiency. To assess and compare predictions between different methods is one of the aims of this thesis. *Predictive accuracy* can be assessed using different measures. It is important to point out that risk prediction in survival analysis is done in terms of probabilities.

In sections 4.1 and 4.2 the traditional statistical methods for survival data are presented, whereas in section 4.3 performance evaluation metrics from machine learning field using *a simple confusion matrix*. The chapter ends with a discussion about the strategy we followed to avoid over-fitting during model building phase.

4.1 Concordance index

To evaluate prognostic models, we want to be able to distinguish individuals who will have the event from those who will not. The most popular measure of discrimination in survival analysis is the C-index (or concordance index) advocated by Harrell et al. (1996). This measure calculates the proportion of pairs of observations for which the survival times order and model predictions are concordant. It takes into account censoring.

Let $t_1, t_2, \cdots, t_n$ and and L_i denote unique times in the data and the predicted risk outcome for individual i respectively. Subject i has a worse predicted outcome than individual j if $L_i > L_j$. The following steps are used to calculate the C-index:

1. Form all possible pairs of observations in data
2. Omit the pairs for those events that shorter survival time is censored. Other pairs (i, j) omitted are: if $T_i = T_j$ unless $(\delta_i = 1, \delta_j = 0)$ or $(\delta_i = 0, \delta_j = 1)$ or $(\delta_i = 1, \delta_j = 1)$. Denote the resulting combinations of pairs by $\mathbb{S}$, and the total possible number by $|\mathbb{S}|$.
3. If $(T_i \neq T_j)$, count 1 for each $s \in \mathbb{S}$ in which the shorter time had the worse predicted outcome.
4. If $(T_i \neq T_j)$, count 0.5 for each $s \in \mathbb{S}$ in which $L_i = L_j$.

5. If $(T_i = T_j)$, count 1 for each $s \in \mathbb{S}$ in which $L_i = L_j$.
6. If $(T_i = T_j)$, count 0.5 for each $s \in \mathbb{S}$ in which $L_i \neq L_j$.
7. Concordance index $C = \frac{concordant\ pairs}{possible}$.

It can be written as (Van Houwelingen and Putter, 2011):

$$C = \frac{\sum_{i \in D} \left(\#(j \in R(t_i); x_j < x_i) + 0.5 * \#(j \in R(t_i), j \neq i; x_j = x_i) \right)}{\sum_{i \in D} \left(Y(t_i) - 1 \right)} \tag{4.1}$$

with $R(t)$ the risk set of those still alive at some event time t.

This statistic is equivalent to the area under the receiver operating characteristic curve (ROC). A concordant index of 0.5 means that the model predicts no better than random. A c-statistic between 0.8 and 0.9 indicates excellent diagnostic accuracy and a model with concordance higher than 0.6 is considered clinically useful.

4.2 Brier score and integrated Brier score

C-index provides a rank statistic that measures the association between true survival time and predictor. A different measure with time-dependent nature is the *Brier scoring rule* defined as:

$$\mathrm{BS}(\mathrm{y}, \hat{\mathrm{S}}(\mathrm{t_0}|\mathrm{x})) = (\mathrm{y} - \hat{\mathrm{S}}(\mathrm{t_0}|\mathrm{x}))^2 \tag{4.2}$$

where $y = 1\{t > t_0\}$ is the actual observation ignoring censoring.

The expected value of Brier score with respect to a new observation y_{new} of true model $S(t_0|x)$ is:

$$E(\mathrm{BS}(\mathrm{y_{new}}, \hat{\mathrm{S}}(\mathrm{t_0}|\mathrm{x}))) = \mathrm{S}(\mathrm{t_0}|\mathrm{x})(1 - \mathrm{S}(\mathrm{t_0}|\mathrm{x})) + (\mathrm{S}(\mathrm{t_0}|\mathrm{x}) - \hat{\mathrm{S}}(\mathrm{t_0}|\mathrm{x}))^2 \tag{4.3}$$

Brier score consists of true variation $S(t_0|x)(1-S(t_0|x))$ and model error $(S(t_0|x) - \hat{S}(t_0|x))^2$. In practice $S(t_0|x)$ is unknown so the two components are not separable. A solution is to multiply the Brier score by the Kaplan-Meier estimate to scale the score.

Perfect prediction exists if $S(t_0|x) = 0$ or $(S(t_0|x) = 1$. In case of a model that predicts randomly and $\hat{S}(t_0|x) = 0.5$ according to (4.2) the Brier score will be $(0 - 0.5)^2 = 0.25$ for no event and $(1 - 0.5)^2 = 0.25$ for event. A model that assigns risk probabilities with a uniform distribution U(0, 1) would give a Brier score of 0.33. The most important benchmark is the expected Brier score of a prediction model without any predictors. In general, the model with smallest score is the best.

Equation (4.2) ignores censoring. To calculate Brier score, when censored observations are present, we have to deal with those censored before time t_0. (Graf et al., 1996) proposed to skip these observations and weight appropriately the ones that we use. This can be done by using the inverse probability of censoring weighting (IPCW) and multiplying by $\frac{1}{\hat{C}(t^*|x)}$. Here, $t^* = t$ for those who experienced an event before time t_0 and $t^* = t_0$ for individuals still at risk at t_0. $\hat{C}(t^*|x)$ is an estimate of $P(T_{cens} > t|x)$.

Brier score is calculated at different time-points. As an overall measure of prediction error, the Integrated Brier Score (IBS) is widely used to summarize the prediction error curves (Mogensen et al., 2012).

$$\text{IBS}(\text{Err}, \tau) = \frac{1}{\tau} \int_0^{\tau} \text{Err}(u, \hat{S}) du \tag{4.4}$$

where Err is the estimated predictive performance at each $0 < \tau < t_{max}$.

4.3 Metrics of confusion matrix

In machine learning, a well known way to evaluate classification's performance is the confusion (error) matrix (Stehman, 1997). Rows represent the predicted class, whereas columns the actual class. An example of a confusion matrix for binary classification is given in table 4.1 - a contingency table with 2 dimensions. An "Event" corresponds to status 1 for OS or FFS and "No Event" corresponds to status 0 (censoring). "A" are the true positives, "B" the false positives, "C" the false negatives and "D" the true negatives. A medical example for false positive is the doctor announcing to a patient that will die within the next year while he managed to survive. The equivalent example for false negative case is the doctor announcing to the patient that will survive for the next year while he/she dies. Both cases are unwanted and especially the one with the false positive.

Table 4.1: Example of a 2x2 confusion matrix used in binary classification.

	Actual	
Predicted	Event	No Event
Event	A	B
No Event	C	D

There are several metrics that can be defined through this matrix.

- **Accuracy**: the number of correct predictions divided by the total number

$$Accuracy = \frac{A + D}{A + B + C + D} \tag{4.5}$$

- **Sensitivity (or Recall)**: measures the proportion of actual positives that are correctly identified (given that event is a positive prediction). Recall is a measure of completeness. High values recommend many true positives and low values many false negatives.

$$Sensitivity = \frac{A}{A + C} \tag{4.6}$$

- **Specificity**: measures the proportion of actual negatives correctly identified. High values indicate many true negatives and low many false positives.

$$Specificity = \frac{D}{B + D} \tag{4.7}$$

- **Precision**: calculates the proportion of positive results which is a measure of exactness.

$$Precision = \frac{A}{A+B} \tag{4.8}$$

- **F1score**: the harmonic mean of precision and sensitivity. It is used to get an estimate of the balance between those 2 metrics.

$$F1score = \frac{2 * precision * recall}{precision + recall}. \tag{4.9}$$

All these metrics are based on a probability threshold for predicted survival (a value $\in [0, 1]$). This defines the expected status (1 = death, 0 = censored) for each patient and can be used for comparison with the real status to get a matrix as in table 4.1. For UNOS data, as we want all decisions to be made on equal footing, the probability threshold will be set at 0.5. Different threshold values can be used in order to take stricter or more lenient decisions regarding the expected status of each patient.

4.4 Building and evaluating models

Problems with over-fitting

In case of over-fitting difference between predictive performance on same data and on external data can be substantial. Building and evaluating a model on the same data can be dangerous. The model may fit the data perfectly and the apparent error rate be very small. However, using the same model on new individuals could produce a large error since this model lacks of generalizability (Van Houwelingen and Le Cessie, 1990).

General strategy

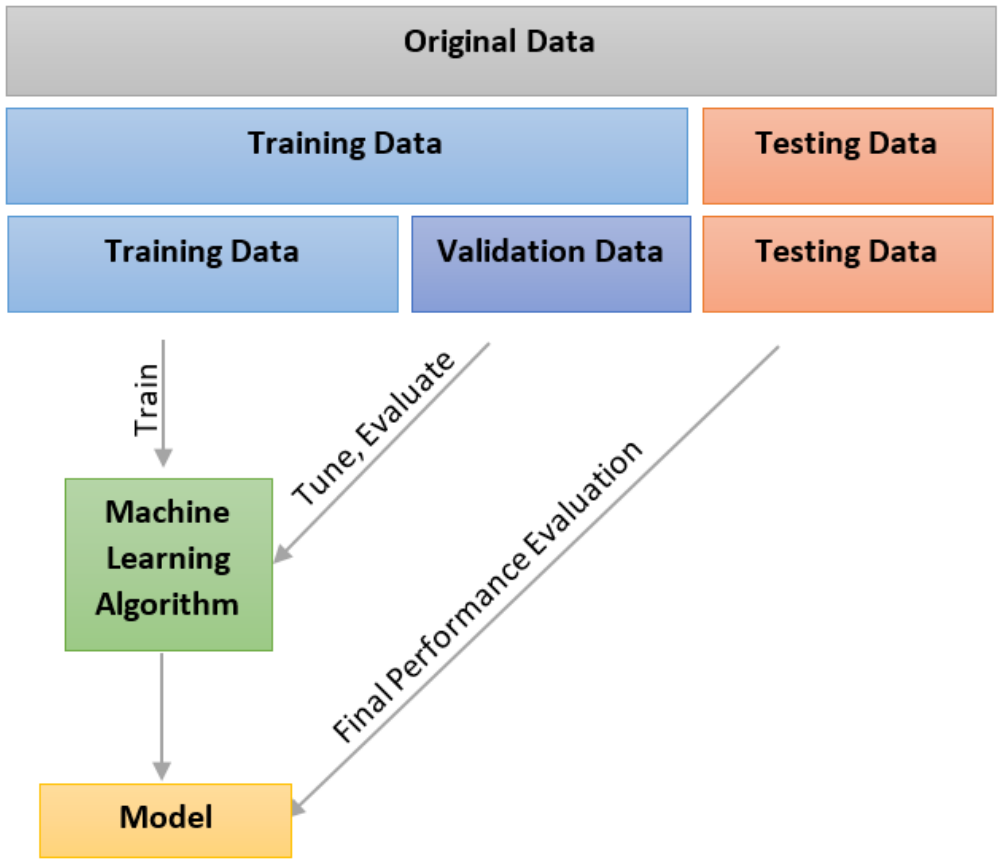
To build and evaluate prediction models we will use a general strategy, to avoid over-fitting, known as **split sample approach**. Models will be built in one part of the data and tested in another part. The original dataset will be split in $\frac{2}{3}$ (training data) and $\frac{1}{3}$ (test or hold-out data). An illustration of the procedure is shown in figure 4.1. Training data are used for model development and testing data are used to evaluate the performance of the final model specified.

Each method - as described in chapter 3 - has its own characteristics and thereby requires a separate strategy. Machine learning techniques like variables selection with LASSO penalization, neural networks and random survival forest require tuning of hyper-parameters to create a model that fits very well to training data but also has good predictive performance on new "unseen" data (testing data).

To build a highly generalizable model, we split the training data into *5*-folds and performed cross-validation. Each time 4 folds were used to train a model and the left-out fold is used to validate its performance. Based on a proper statistic the parameters were tuned on a grid of randomized points. For the LASSO method, the hyper-parameter was tuned using the cross validated log partial likelihood as described in equation (3.6). For random survival forest $E = (1-C)$ was utilized with C calculated by the ensemble mortality score (3.12), while for neural networks all hyper-parameters were tuned using the binary cross entropy loss function (3.16).

A note of difference between other machine learning techniques and random forests needs to be mentioned here. Random forests have a unique capability as out-of-bag samples are collected during bootstrapping (discussed in section 3.3.1). These samples are used to estimate prediction error in a procedure which is almost identical to leave-one-out cross validation. Therefore the training and the validation is performed automatically. Parameters can be tuned by finding the best combinations that decrease the out-of-bag error of the train set. However, to make a fair comparison of all methods, 5-fold cross-validation with the same folds will be used for tuning.

Figure 4.1: Procedure followed for data splitting.



Methods to deal with over-fitting on complete data

In addition to the data split method, there are other methods:

- k-fold cross-validation on whole data.
- *Bootstrap cross-validation*: bootstrap is implemented with or without replacement (sub-sampling bootstrap). Models are trained in bootstrap samples and validated on the complete data (Fu et al., 2005).
- *0.632 bootstrap*: linear combination of the apparent error and the bootstrap cross-validated error

$$\text{Boot632Err}(t, \hat{S}) = (1 - 0.632) * \text{AppErr}(t, \hat{S}) + 0.632 * \text{BootCvErr}(t, \hat{S}),$$

called 0.632 because for large n, it selects $1 - e^{-1} \approx 0.632$ of the original data points at least once at each bootstrap sample. A generalization of it is the 0.632+ bootstrap proposed by Efron and Tibshirani (1997).

Since bootstrap samples contain substantially less information than the full data sample, bootstrapping is a very efficient way to deal with over-fitting because it leads to underestimation of expected performance.

Model variability

This chapter ends with a short discussion about model variability. During data splitting process, 2/3 of the complete data were specified as training and 1/3 as test data. It is expected that models built under different training samples will be different in performance. This model uncertainty is included as a substantial part of prediction error (Gerds and van de Wiel, 2011).

Chapter 5

Application

In this chapter, the methods illustrated in Chapter 3 will be applied to the UNOS data. In section 5.1 results about Cox proportional hazards models are presented. In sections 5.2 and 5.3 results for random survival forests and neural networks for survival analysis are illustrated. Sections are split into two parts: overall survival (OS) and failure-free survival (FFS). At the end of the sections a small discussions about the findings is presented.

5.1 Cox Proportional Hazards models

In this section we describe the results for Cox regression models. Variable selection using backward and the LASSO method are discussed. Subsections are devoted separately to overall and failure-free survival (5.1.3, 5.1.4 for backward selection and 5.1.5, 5.1.6 for selection with LASSO).

5.1.1 OS: Interpretation of the Cox model using all variables

Cox proportional hazards model was fit for all 106 prognostic factors on the training data. Hazard ratios along with their 95% confidence intervals for overall survival are shown in appendix B table B.1. There are in total 128 levels in the 106 variables of the dataset. The most important factor is *immuno-maintenance medicine*. Having received the medicine decreases hazard by factor 0.1 or 90%. Another covariate which decreases hazard by factor 0.7 or 30% is *accepting an extra-corporeal liver*. Covariate *income* decreases hazard by factor 0.814. The most detrimental factors are *previous transplantation* which increases hazard by 1.573 or 57.3%, *incidental tumor* (39.1%), *angina* (33.7%), *life support* (30.2%) and *diabetes* (23.3%).

5.1.2 FFS: Interpretation of the Cox model using all variables

Hazard ratios along with 95% confidence intervals for failure-free survival are shown in appendix B table B.2. Strongest predictor is *immuno-maintenance medicine*. Being in the group of patients that received the drugs decreases hazard by factor 0.081 or 91.9%. Beneficial prognostic factors are *candidate accepted extra-corporeal liver* group yes (decreases hazard 31.1%) and *race other* versus white decreases hazard by factor 0.834.

Most detrimental factors: *patient was re-transplanted before* which increases hazard by factor 1.503 or 50.3%; *donor type DCD* (Donor after Cardiac Death) versus group DBD (Donor Brain-Dead) which increases hazard by factor 1.414 or 41.4 %. *Incidental tumor* increases hazard by 34.5%. Compared to overall survival, *donor type* seems to have an increased effect on event-free survival (no graft-failure or death).

5.1.3 Overall survival: Backward elimination

In the original UNOS dataset, out of 62294 cases, about 70.9% of the patients were censored and 29.1% dead. On the training and test set same proportions exist approximately without stratified sampling due to the large sample size. Regarding quantiles of survival time on training set, 25% was 1.71 years, 50% 3.95 years and 75% 6.96 years.

From stepwise regression models, we tested the backward and forward elimination methods on the training set using the `stepAIC()` function of the `MASS` package in R (W. N. Venables and B. D. Ripley, 2002). Such selection methods are quite notorious as being unstable because they use a greedy approach to select variables removing the least contributing with respect to AIC criterion. Backward elimination selected 59 variables at one run, but 54 to 61 variables were selected when we repeated the procedure. With the forward procedure, 55 variables were selected at one run with 51 to 56 in case of reexamination.

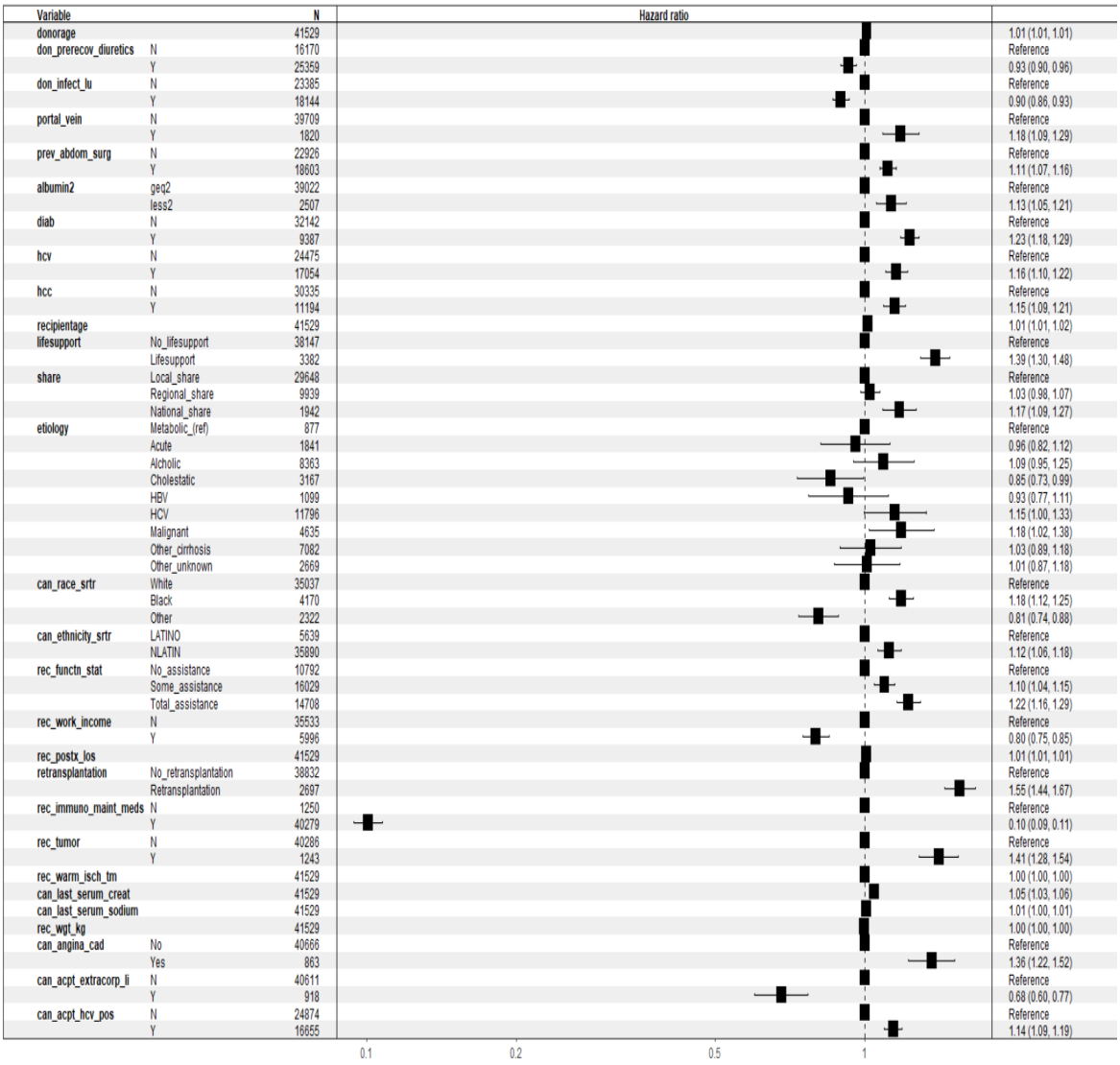
Previous modeling strategies are quite unstable, so to obtain a stable model which selects the same number of variables each time we used the `fastbw()` function of the `rms` package in R (Frank E Harrell Jr, 2018). This method is a numerically stable backward elimination on factors (Lawless and Singhal, 1978), even though slightly inefficient. It estimates the full model and computes approximate Wald statistics by computing conditional maximum likelihood estimates - assuming multivariate normality of estimates. Factors that require multiple degrees of freedom are dropped or retained as a group. In our data - with the AIC rule - 28 variables were selected out the 106 which led to a much simpler and parsimonious Cox model. We will refer to this model from now on as *"Cox backward"*.

Rec_immuno_maint_meds, *rec_postx_los*, *lifesupport* and *diab* violate the proportional hazards assumption ($p << 0.05$). In this project, we compare simple Cox models with machine learning techniques. Therefore we only report the results based on Cox models without interactions.

Interpretation of Cox backward model

A forest plot summarizing the hazard ratios along with their 95% confidence intervals of the Cox backward regression model is shown in figure 5.1. The most important prognostic factor is *immuno-maintenance medicine*. Having received the medicine reduces the hazard of dying by factor 0.1 or 90%. Moreover, being in the group of patients that *accepted the extra-corporeal liver* decreases the mortality risk by 0.68 or 32%. Other variables worth mentioning are *retransplantation*, *tumor* and *life support*. A patient who got a graft before (*retransplantation*), has tumor or is under life support increase hazard by factors 1.55, 1.41 and 1.39 respectively. Other important variables selected were *age of donor and recipient*, *weight of the recipient*, *liver etiology group*, *if the patient has working income*, *albumin group* and *last serum creatinine*, *serum sodium* that are active components of MELD (section 2.2).

Figure 5.1: Forest plot for Cox backward modelOve (OS).

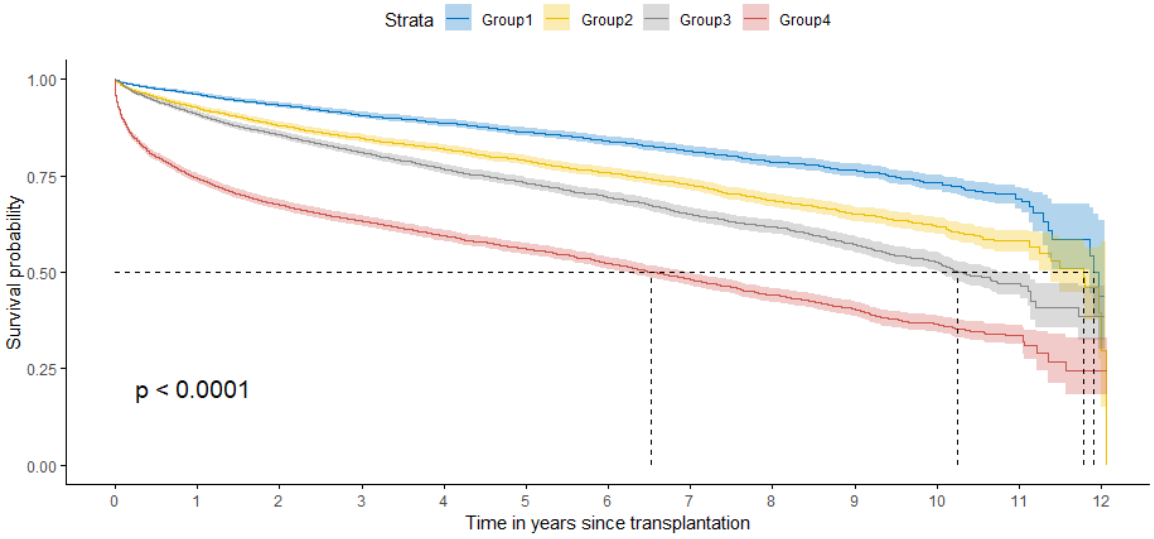


Concordance indices and linear predictors

Concordance probability is the frequency of concordant pairs among all pairs of subjects and is calculated based on the risk predictions (or equivalently using linear predictors) for each individual (Harrell et al., 1996). C-index is related with the number of variables. Higher C-index values are observed for models with more prognostic factors. Cox model with all 106 covariates on the training set had a C-index of 0.687 (se = 0.003), whereas Cox backward C-index of 0.682 (se = 0.003). This means that the discriminative ability of a model with 78 variables less than the full model is barely reduced. Model based on 28 variables selected is capable of making distinctions in the ranking list of patients survival times. Similarly on the test set of the full model predicted C-index was 0.678 (se = 0.014) and on the reduced model 0.676 (se = 0.014).

We discretized the linear predictors of the backward elimination model (logarithmic risk predictors) into 4 equally sized groups of patients on test set using the 25, 50 and 75% quantiles of linear predictors estimated on training set. Patients in group 1 have the lowest risk and patients in group 4 the highest risk. Outcome is shown in figure

Figure 5.2: Survival curves for 4 groups of linear predictors on test set for OS. Group 1: 0-25% quantile, Group 2: 25-50%, Group 3: 50-75% and Group 4: 75-100%. Vertical lines correspond to the median survival per group. Horizontal line corresponds to median survival probability.



5.2. Overall, there is good discrimination between them. Global χ^2 log-rank (test of difference in survival) has a very small p-value. Group 4 has a median survival of 6.5 years much smaller than that of the other 3 risk groups. It is also noticeable that groups 2 and 3 are barely distinguishable for the first 2 years post transplantation. New individuals entering the study can be placed in one of those groups according to predicted linear predictor summarizing information of variables.

5.1.4 Failure-free survival: Backward elimination

In complete UNOS data, 68.7% were censored whereas 31.3% had a graft-failure/died with training and test sets having identical proportions without the need of stratified sampling. Quantiles of survival time on training set were similar to that of OS: 25% was 1.39 years, 50% 3.8 years, 75% 6.84 years.

Backward selection using the `stepAIC()` function selected 59 variables and forward selection 56 variables. `fastbw()` function for backward selection (subsection 5.1.3) yielded a stable model using the AIC criterion. 32 variables were selected leading again in a more parsimonious model. Compared to the equivalent model for OS more variables were selected.

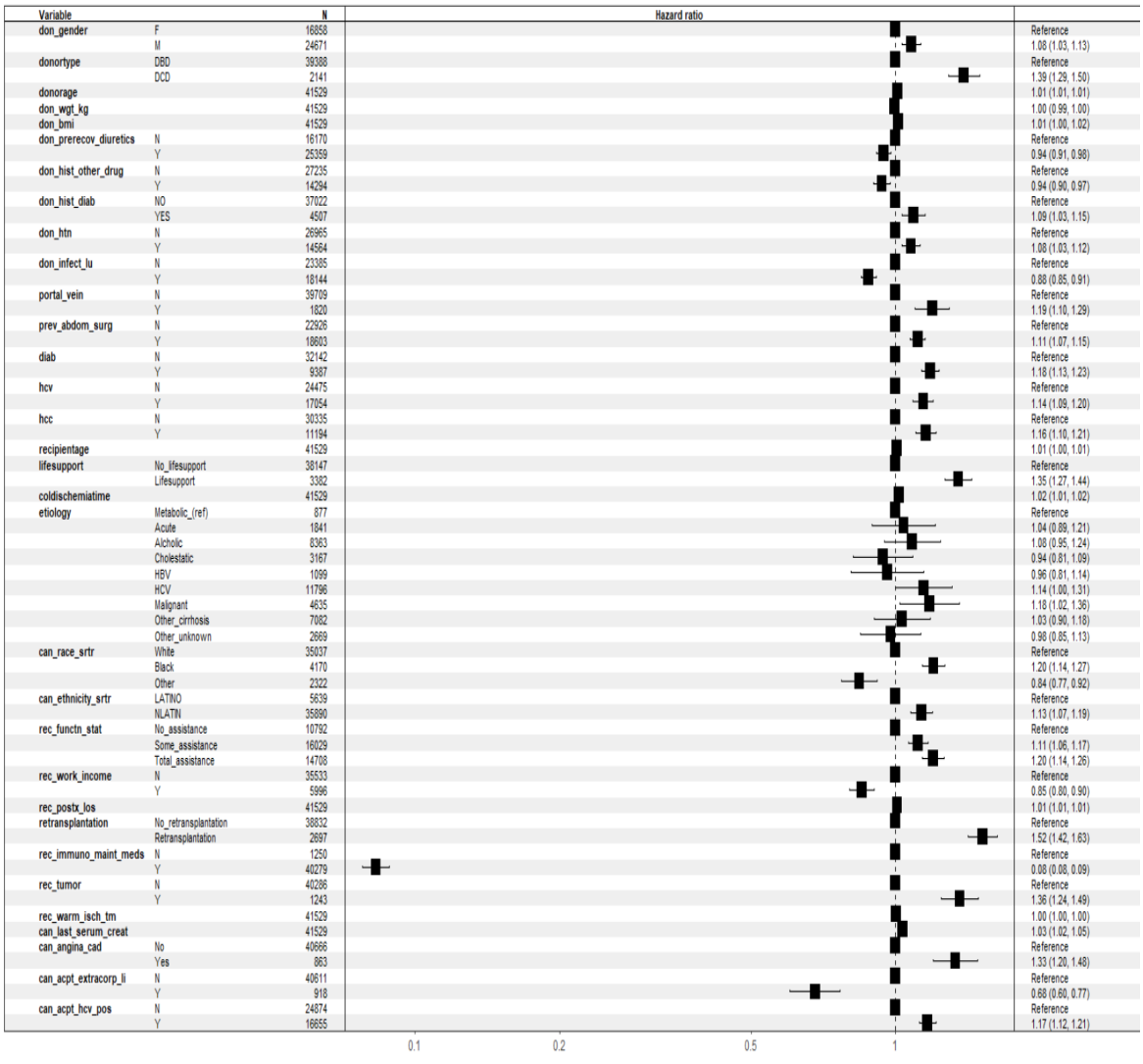
Among other variables *rec_immuno_maint_meds*, *rec_functn_stat* and *rec_postx_los* ($p << 0.05$) violated the PH assumption.

Interpretation of Cox backward model

Forest plot for the Cox backward regression model is presented in figure 5.3. Most important variable is *rec_immuno_maint_meds*. Immuno-maintenance medicine decreases hazard by factor 0.08 or 92%. A patient that has been transplanted before has an increased hazard rate of 1.52. Other important prognostic factor is donor type. *Graft from a Donor after Cardiac Death* (DCD) increases the hazard of

graft failure or death by factor 1.39. More hazard ratios with the high impact are: *can_acpt_extracorp_li* (Did the candidate accept an extra-corporeal liver?), *lifesupport* (Was patient on life support?) and *rec_tumor* (incidental tumor found at time of transplant).

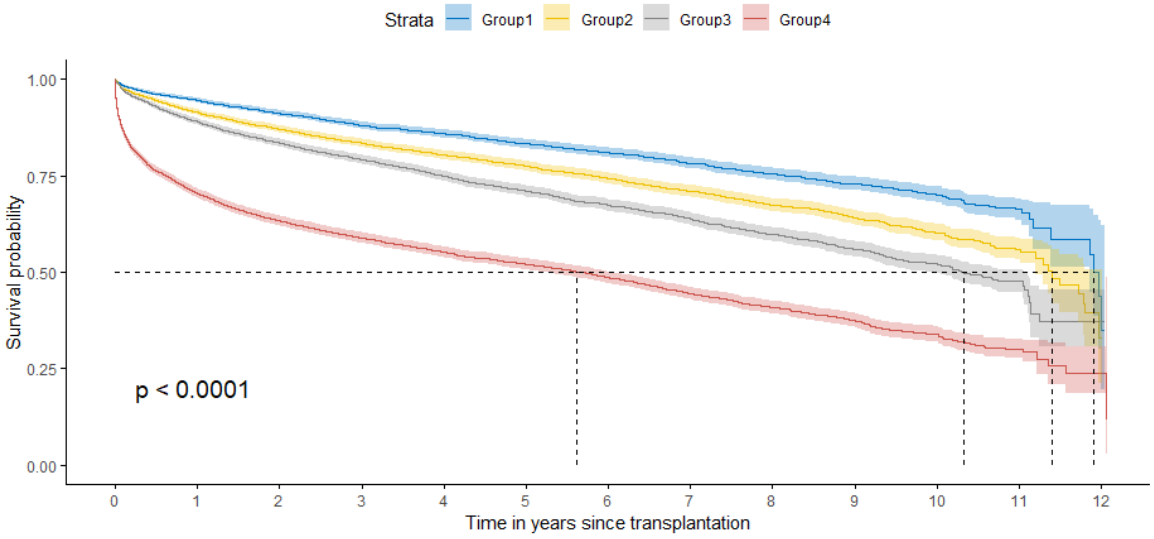
Figure 5.3: Forest plot for Cox backward model (FFS).



Concordance indices and linear predictors

Cox model with all 106 variables had a C-index of 0.684 (se = 0.003) on the train set and of 0.674 (se = 0.014) on the test set. Equivalently the concordance for Cox backward was 0.681 (se = 0.003) on train set and 0.673 (se = 0.015) the predicted on test set. Survival probabilities are shown in figure 5.4. For group 4 (highest risk group) the median survival probability is 5.6 years. For groups with lower risk it is higher than 10 years.

Figure 5.4: Survival curves for 4 groups of linear predictors on test set for FFS. Group 1: 0-25% quantile, Group 2: 25-50%, Group 3: 50-75% and Group 4: 75-100%. Vertical lines correspond to the median survival per group. Horizontal line corresponds to median survival probability.



5.1.5 Overall survival: Selection with LASSO

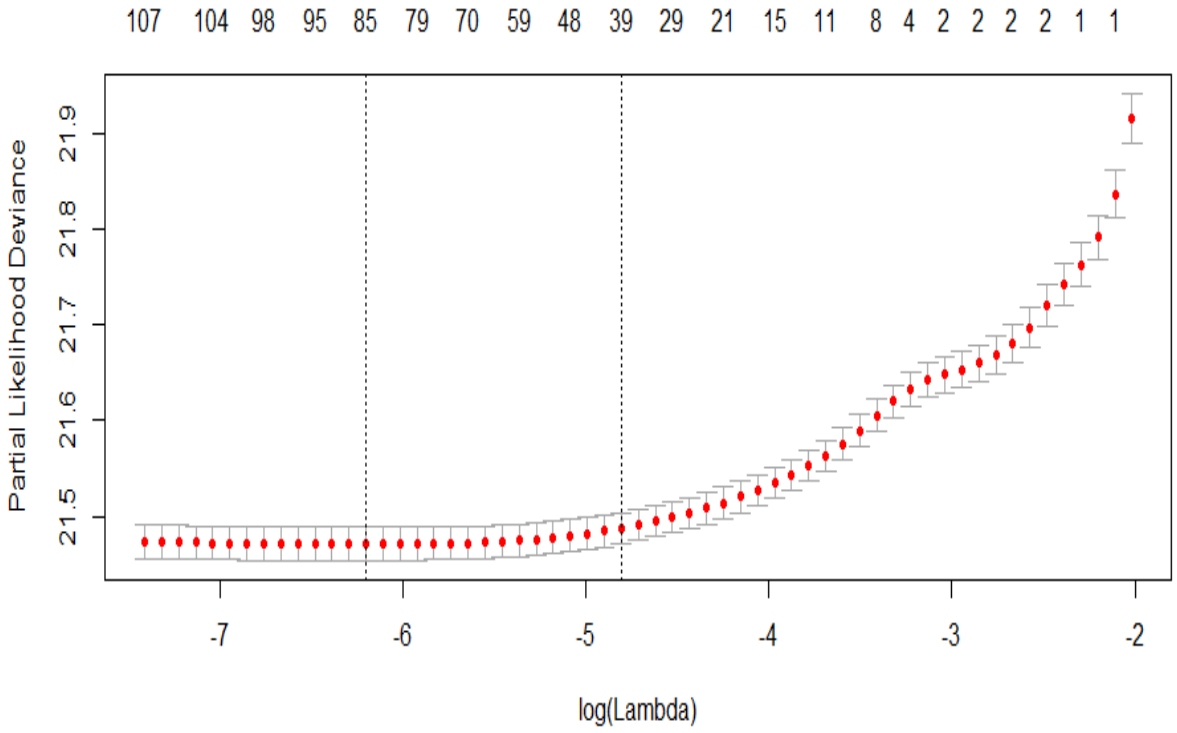
LASSO selects variables that provide a strong contribution to the Cox model by setting coefficients of unimportant variables to zero. Grid search was employed on the training set using 5-fold cross-validation to find the penalty parameter λ_{LASSO} that minimizes the cross-validated partial likelihood deviance as presented in equation 3.6. Package `glmnet` uses a cyclical coordinate descent algorithm computed along a regularized path (Jerome Friedman, 2010). It traces the minimum λ_{LASSO} value that sets all coefficients to zero and it decreases the amount of penalization so that more and more variables are selected. In this way it performs a targeted grid search over a range of suitable values for training set. Results are illustrated in figure 5.6.

Optimal λ_{min} was found close to 0.002. 85 prognostic variables were selected (categorical variables were transformed to indicators 3.1.3). `Glmnet` also calculates the best regularized model with cross-validation error within 1 standard error from λ_{min}. This ad-hoc rule is a compromise between the best λ and another larger λ value that leads to a more parsimonious model. The final value λ_{1se} was found very close to 0.008 which selects 39 variables. We will prefer λ_{1se} as our final choice as it produces an easier to interpret model. Results are provided in appendix B table B.3. Strongest 3 prognostic factors are *immuno-maintenance meds:Yes*, which decreases hazard by 89%, *retransplantation* which increases hazard by 46% and *life-support* which increases hazard by 25%.

Examining variable selection stability and predictive ability

LASSO aims to find the best model that yields a sparse solution but may select different covariates depending on the random seed used to create the folds when choosing the penalty parameter. 5 or 10 fold cross-validation is generally recommended. To investigate the stability of the variable selection, we repeated the cross-validation grid

Figure 5.5: Variable selection with LASSO for overall survival. Graph shows the cross validation along with upper and lower standard deviation. Left vertical line: λ_{min}, right vertical line: λ_{1se}. Number of variables selected are at the top of the plot.



search 250 times and we recorded λ_{min}, λ_{1se}, the variables selected and predicted C-index on the training data.

Concordance index for models with λ_{1se} was between 0.683 and 0.686 indicating a comparable discriminative ability. The list of different selected variables is shown in table 5.1. Different repeats selected from 19 to 61 variables. Generally 61 variables were selected at least once and 19 always. This table demonstrates how dependent results of cross-validation are to the folds. Note that to make a fair comparison in performance of different machine learning techniques, we use 5-fold cross validation with the exact same folds to train the models.

For the LASSO models using the equivalent λ_{min} 80 to 95 variables were selected at different repeats (table B.4). C-index was almost identical for all models ranging from 0.686 to 0.687. Note that variable *life-support* (level Lifesupport) which has a high hazard ratio (1.25) in table B.3 has been selected in only 3 of the 250 repeats. The strongest prognostic factor *immuno-maintenance medicine* was selected 9 times. These results suggest that the ad-hoc 1se rule might not always be a good choice since important prognostic factors can be ignored. Model defined by λ_{min} selects many more prognostic variables.

This is one of the pitfalls of LASSO method. When there are grouped variables (highly correlated) it tends to select one variable from each group ignoring the others. Clinical experience suggested that variable *rec_acute_rej_episodeYes* (Did patient have any acute rejection episode?) is related with physical weakness of the patient to receive

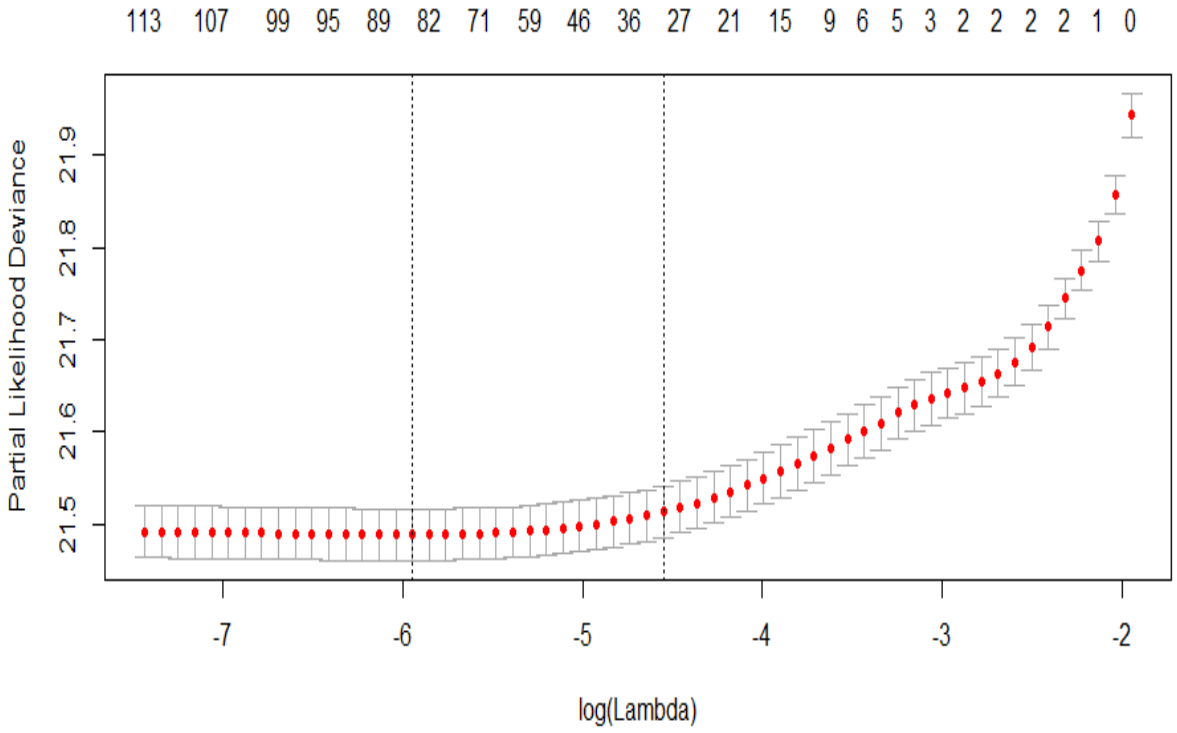
immuno-maintenance medicine. This might affect the selection process. In table 5.1 this phenomenon occurs with *rec_acute_rej_episode:Yes* being selected 250 times and *rec_immuno_maint_meds:Yes* in only 9 cases.

All in all - considering the discriminative ability - model with λ_{1se} produced C-index of 0.683 to 0.686 over the repeats and model with λ_{min} C-index of 0.686 or 0.687. Therefore, there is hardly any difference in the performance of the models. This is the main reason for preferring LASSO model with λ_{1se} as it can reach almost the same performance with a much smaller number of variables. In other words, loss in predictive performance is unimportant compared to gain in interpretability.

Table 5.1: Variables selected with LASSO from 250 repeats of cross-validation using λ_{1se} for OS.

Variable	Times	Variable	Times
albumin2less2	250	encephY	186
can_acpt_abo_incompY	250	don_ethnicity_srtrNLATIN	154
can_acpt_hbc_posY	250	last_dial_prior_weekY	154
can_last_inr	250	hbvY	113
can_peptic_ulcerYes	250	portal_veinY	113
diabY	250	rec_bacteria_peritY	113
donorraceOther	250	rec_postx_los	113
etiologyAlcholic	250	rec_tumorY	113
maligY	250	can_race_srtrOther	76
portal_hyperten_bleedY	250	don_meet_cdc_high_riskY	76
pretxstatusHOSPITALIZED	250	donorcodCNS_Tumor	76
pretxstatusIC_UNIT	250	can_variceal_bleedingY	48
rec_acute_rej_episodeYes	250	can_educationMedium_education	27
rec_bmi	250	don_hla_typY	27
rec_functn_statSome_assistance	250	etiologyCholestatic	27
rec_functn_statTotal_assistance	250	prev_abdom_surgY	27
rec_hbv_surf_antigenP	250	rec_wgt_kg	27
rec_work_incomeY	250	don_htnY	12
splitsplit	250	donorraceBlack	12
ascitesY	249	hccY	12
don_infect_bloodY	249	rec_cmv_statP	12
don_hist_diabYES	247	can_aboGroup_O	9
don_anti_hcvP	240	can_drug_treat_hypertenY	9
etiologyHBV	240	can_ethnicity_srtrNLATIN	9
etiologyHCV	240	etiologyOther_unknown	9
hcvY	240	rec_immuno_maint_medsY	9
can_last_serum_creat	220	don_genderM	5
shareRegional_share	220	don_log_bun	5
can_race_srtrBlack	186	lifesupportLifesupport	3
don_hist_cocaineY	186	recipientsexMale	3
don_txfus_terminal_hosp_numGreater_than_10	186		

Figure 5.6: Variable selection with LASSO for failure-free survival. Graph shows the cross validation along with upper and lower standard deviation. Left vertical line: λ_{min}, right vertical line: λ_{1se}. Number of variables selected are at the top of the plot.



5.1.6 Failure-free survival: Selection with LASSO

We employed again 5-fold cross validation for FFS. λ_{min} was found at 0.0026 with 82 variables being selected. λ_{1se} was specified at 0.011 choosing 32 variables. Results for λ_{1se} are presented in appendix B table B.5. Strongest 3 prognostic factors are *immumo-maintenance meds:Yes* (decreases hazard by 91%), *re-transplantation* (increases hazard by 36%) and *life-support* (increases hazard by 24%).

Examining variable selection stability and predictive ability

5-fold cross-validation was repeated 250 times on the training set using different seeds for creating the folds. For λ_{1se} concordance was 0.683-0.684 indicating small differences between different seeds. List of variables selected at different repeats is shown in table 5.2. In total, 59 variables were chosen at least once and 21 at all repeats. Variable *rec_immuno_maint_meds* was chosen 79 times and *life-support* not even once. Therefore, parsimony comes at a cost of uncertainty in variable selection.

For the LASSO models using λ_{min}, 81 to 99 variables were selected at multiple runs (table B.6). C-index on training set was 0.684 or 0.685. As for overall survival, difference in discriminative ability between models with λ_{1se} and λ_{min} is minimal. Therefore, LASSO model with λ_{1se} will be preferred as it provides more parsimonious models.

Table 5.2: Variables selected with LASSO from 250 repeats of cross-validation using λ_{1se} for FFS.

Variable	Times	Variable	Times
albumin2less2	250	pretxstatusHOSPITALIZED	217
can_acpt_abo_incompY	250	shareRegional_share	217
can_acpt_hbc_posY	250	can_race_srtrOther	179
can_last_inr	250	can_last_serum_creat	125
can_peptic_ulcerYes	250	don_aboGroup_O	125
diabY	250	rec_tumorY	125
don_ethnicity_srtrNLATIN	250	ascitesY	79
don_hist_diabYES	250	can_educationMedium_education	79
don_infect_bloodY	250	don_txfus_terminal_hosp_numGreater_than_10	79
donorraceOther	250	rec_immuno_maint_medsY	79
hbvY	250	last_dial_prior_weekY	52
maligY	250	rec_bacteria_peritY	52
portal_hyperten_bleedY	250	rec_cmv_statP	52
pretxstatusIC_UNIT	250	rec_wgt_kg	52
rec_acute_rej_episodeYes	250	can_aboGroup_O	28
rec_bmi	250	donorcodCNS_Tumor	28
rec_functn_statSome_assistance	250	can_variceal_bleedingY	11
rec_functn_statTotal_assistance	250	don_htnY	11
rec_hbv_surf_antigenP	250	don_log_bun	11
rec_work_incomeY	250	rec_postx_los	11
splitsplit	250	shareNational_share	11
don_anti_hcvP	244	don_hist_cigarette_gt20_pkyrY	5
don_meet_cdc_high_riskY	244	don_prerecov_diureticsY	5
etiologyAlcholic	244	can_cereb_vascY	2
hcvY	244	don_ddavpY	2
encephY	235	don_genderM	2
etiologyHBV	235	portal_veinY	2
can_race_srtrBlack	217	prev_abdom_surgY	2
don_hist_cocaineY	217	rec_warm_isch_tm	2
etiologyHCV	217		

5.2 Random Survival Forests models

In this section results for RSF are presented (for OS section 5.2.1 and for FFS 5.2.2). The section ends with a small discussion (5.2.3).

5.2.1 Overall survival

Tuning the parameters

In section 3.3.3 we discussed different parameters that can be trained for RSF. Grid search is in this context a 3-dimensional space with parameters *nodesize*, *mtry* and *nsplit*. Other trainable parameters are splitting rule and the number of trees. Training data include 41529 observations and 106 variables. Because of the complexity of the dataset, random forest algorithm required a large amount of time to grow a forest of trees. Analysis was performed in a hyper-computer which operates under Unix [1] system.

Initially, we run a preliminary grid search for the 3 tuning parameters using the OOB train error (1 - C) as an accuracy measure. Each time a forest of 500 trees was grown using a stratified bootstrap sub-sampling of 1000 individuals. Possible values for the parameters were: *nodesize* (5, 10, 25, 40, 55, 70) , *mtry* (11, 17, 23, 29, 35, 41, 47) and *nsplit* (3, 5, 7, 9, 11). Based on the log-rank rule the combination that resulted in the smallest OOB training error of 0.302 was (nodesize = 70, mtry = 35, nsplit = 7). Based the log-rank score rule, best combination was (nodesize = 70, mtry = 11, nsplit = 9) with OOB error 0.3403.

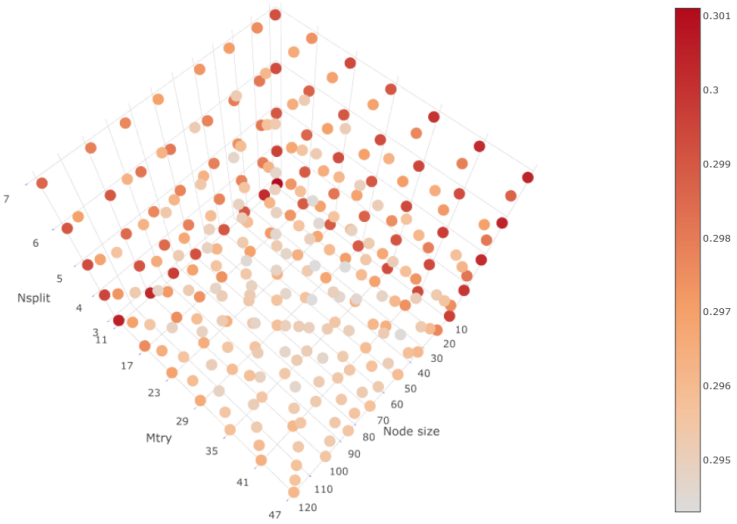
We repeated the grid search with 500 trees for a subsample of 5000 individuals for *nodesize* (10, 20, 35, 50, 70, 85, 100, 120) , *mtry* (11, 17, 23, 29, 35, 41, 47) and *nsplit* (3, 4, 5, 6, 7). Computational time was dramatically increased. A single iteration of forest growing lasted approximately 3 minutes for the log-rank rule and 30 minutes for the log-rank score rule. Best combinations were (nodesize = 50, mtry = 35, nsplit = 5) for the log-rank rule with OOB training error 0.294 and (nodesize = 50, mtry = 11, nsplit = 5) for the log-rank split rule with error 0.335. These results suggest that the log-rank rule is much more accurate and computationally more efficient for UNOS data than log-rank score rule.

OOB error can be a shortcut for tuning the parameters as it is possible to estimate the error of different combinations at once in the training set. For regression and classification random forests OOB tends to overestimate the true generalization error (Janitza, 2017). When using RFS it is not clear if OOB error can be a good alternative to cross-validation error. Therefore, best parameters of OOB error and 5-fold cross validation error will be compared. Needless to say that, as the aim of the thesis is the comparison of different methods, parameter tuning should be performed in the same folds as for the other methods to get comparable results.

We performed 5-fold cross validation employing the same folds used for LASSO. We looked for the combination that minimizes the mean error of the train set for the log-rank splitting rule. Same combination (nodesize = 50, mtry = 35, nsplit = 5) was the best. Mean error over the folds was 0.295. Visualization of the grid search is shown in figure 5.7. From this plot it can be observed that best combination are located around the center of grid of the 280 combinations (gray dots).

[1] We utilized Leiden university's mathematical institute server http://pub.math.leidenuniv.nl/~kampertmmd/Tukey.html

Figure 5.7: Grid search on a 3D space for RSF with log-rank split rule: OS. Minimum error of 5-fold cross validation was found at combination (nodesize = 50, mtry = 35, nsplit = 5).



Note that the number of forest trees is related with the stability of the forest algorithm. More trees lead to more stable forests (Strobl et al., 2009). Having selected the best combination for mtry-nodesize-nsplit, we examined the number of trees that stabilizes OOB error for different blocks of trees (5 to 500). Graph 5.8 suggests that around 250 trees error is stabilized. From now on we will use 250 trees instead of the 500 used for hyper-parameter training to reduce computational time.

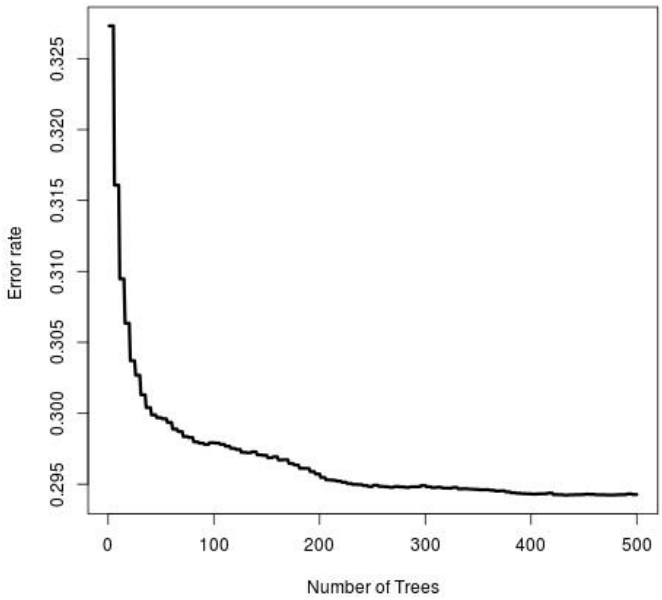
We grow the final RSF for overall survival using the log-rank split rule, ntree = 250, nodesize = 50, mtry = 35 and nsplit = 5. Note that this forest regards exact time points (all survival times in the data). In chapter 6, 5-fold cross-validation is repeated at 12 distinct time points.

Final model for OS

In this subsection, we report the results for the tuned model. Figure 5.9 shows the number of leaves per tree and the most frequent variables used by the log-rank rule to determine a split. The majority of the trees have grown 145 to 165 terminal nodes. This number was regulated by the parameter nodesize that controls the average nodesize of trees. Parameter has been set to 50 allowing on average 50 deaths at the terminal nodes of the trees. Smaller numbers of nodesize would create deeper trees and larger number shallower trees. Note that deeper trees might include more noisy variables which would increase the OOB error rate. On the other hand, larger values may cause the forest to under-grow compromising again accuracy.

Examining variables for splitting, *rec_postx_los* (length of stay at the hospital) is by far the most frequent. Variables in decreasing order are recipient and donor age, *can_last_serum_sodium* and *rec_hgt_cm*. Noteworthy, all variables are continuous. It is known that decision trees tend to favor splitting on continuous variables Loh and Shih (1997). Bias was reduced using 5 random splits points for each candidate x-variable. This may mitigated the amount of bias. Nevertheless, the algorithm for the forest using

Figure 5.8: Number of trees as a function of error rate for OS.



the *log-rank* rule artificially preferred numeric variables as they are more informative than categorical.

Tables 5.3 and 5.4 show the 20 variables with the highest VIMP on the training and the ones predicted on test set. Variable *rec_postx_los* has the highest importance on train set (0.057) and 2nd highest on the test set (0.045). In addition, variable *rec_immuno_maint_meds* has the 2^{nd} highest importance on the train set (0.038) and the highest on the test set (0.045). These 2 variables are the strongest predictors. The next 4 most predictive factors are *hcv* (HCV serology status) recipient and donor age as well as can_acpt_hcv_pos (did candidate receive an HCV antibody serology status?). For both training and test set there were more than 10 variables with negative importance indicating that these variables are noisy factors. Indicatively, such variables on both training and test set were *don_hist_cocaine* (has donor history of cocaine?), *don_infect_urine* (donor infection source urine), don_arginine (donor's arginine) and *don_ddavp* (donor meds DDAVP).

Using stratified sub-sampling in the survival forests, confidence intervals and the standard errors of VIMP (Ishwaran and Lu, 2018) were estimated. The subsample forest has inherited the same tuning parameters as the original grown forest. It is figure number 5.10. The 2 covariates with the highest importance are *rec_postx_los* and *rec_immuno_maint_meds*. The other 6 most important variables (less than 1% VIMP) are *hcv*, *recipientage*, *donorage*, *can_acpt_hcv_pos*, *etiology* and *diab*.

Another method discussed in 3.3.4 which can be used to identify predictive factors is minimal depth. It assesses the importance of a variable using the topology of the tree by computing its depth compared to the root node of a tree. Outcomes for OS are shown in table 5.5. Predictors with the minimal depth were once again *rec_immuno_maint_meds* and *rec_postx_los*. There is one variable don_sodium that despite having a low order minimal depth of 8.620, it has a negative VIMP. This demonstrates the difference between VIMP and minimal depth. VIMP is estimated as

Table 5.3: Variables with highest importance on training set using random survival forests (OS).

Variable	VIMP	Variable	VIMP
rec_postx_los	0.0575	hcc	0.0016
rec_immuno_maint_meds	0.0377	pretxstatus	0.0014
hcv	0.0058	lifesupport	0.0014
recipientage	0.0055	can_race_srtr	0.0014
donorage	0.0053	rec_tumor	0.0008
can_acpt_hcv_pos	0.0043	rec_functn_stat	0.0006
etiology	0.0033	can_last_bili	0.0006
diab	0.0018	don_htn	0.0005
retransplantation	0.0018	rec_work_income	0.0005
can_last_serum_creat	0.0016	can_last_inr	0.0005

Table 5.4: Most important predicted variables on test set using random survival forests (OS).

Variable	VIMP	Variable	VIMP
rec_immuno_maint_meds	0.0449	pretxstatus	0.0012
rec_postx_los	0.0426	hcc	0.0011
hcv	0.0050	lifesupport	0.0011
donorage	0.0049	can_race_srtr	0.0009
can_acpt_hcv_pos	0.0037	rec_functn_stat	0.0006
recipientage	0.0033	last_dial_prior_week	0.0006
etiology	0.0025	don_htn	0.0005
retransplantation	0.0020	rec_tumor	0.0005
can_last_serum_creat	0.0018	can_last_inr	0.0004
diab	0.0012	rec_work_income	0.0004

Table 5.5: Most important variables using minimal depth on grown training forest for overall survival.

Variable	Depth	VIMP	Variable	Depth	VIMP
rec_immuno_maint_meds	1.164	0.0377	diab	8.460	0.0018
rec_postx_los	1.364	0.0575	don_sodium	8.620	-0.0001
recipientage	4.632	0.0055	don_tot_bili	8.724	0.0002
donorage	5.176	0.0053	can_acpt_hcv_pos	8.924	0.0043
can_last_serum_creat	6.832	0.0016	can_last_bili	8.968	0.0006
can_last_serum_sodium	7.676	0.0002	can_last_inr	9.372	0.0005
etiology	7.924	0.0033	don_inr	9.584	0.0002
hcv	8.088	0.0058	don_hgt_cm	9.668	0.0000
retransplantation	8.252	0.0018	can_race_srtr	9.704	0.0014
rec_hgt_cm	8.340	0.0004	don_hematocrit	9.756	0.0001

Figure 5.9: Number of terminal nodes for each tree in the forest (left panel) and number of times a split occurred on 5 most frequent variables for forest (right panel) for OS.

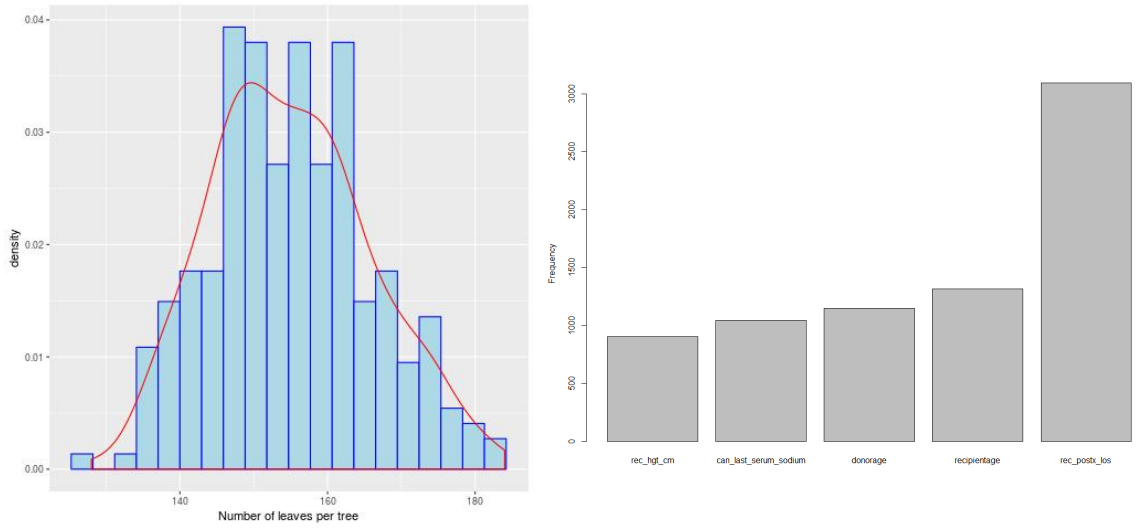
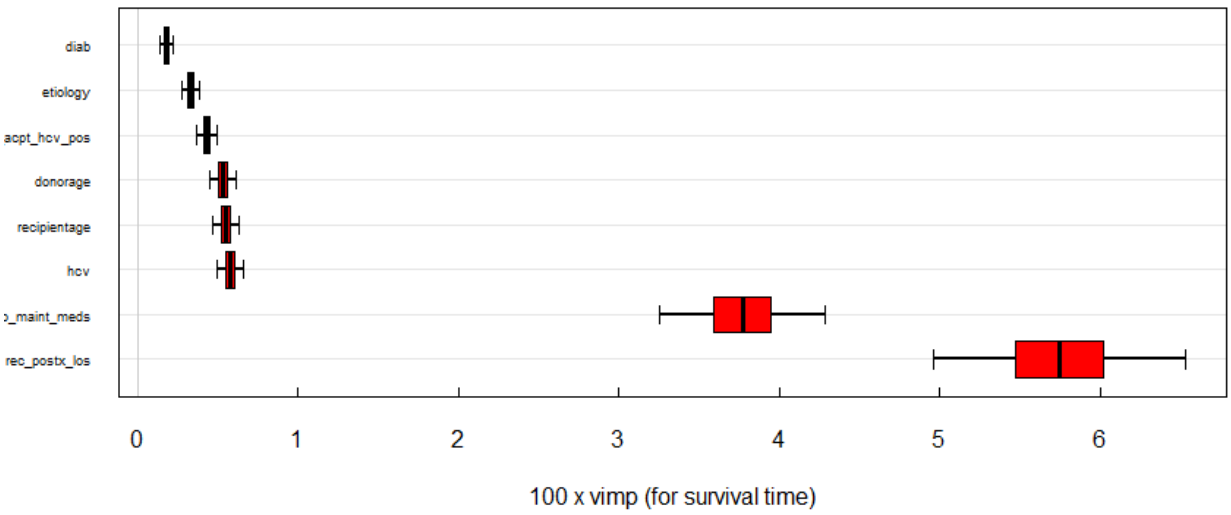


Figure 5.10: Confidence intervals and standard errors for VIMP using sub-sampling (OS). On the x-axis VIMP is scaled by 100.



the difference in error of a particular variable before and after its permutation, whereas minimal depth is related to the topology of the forest.

Comparison of RSF with Cox model in OOB error

Here, we make a comparison between RSF and the Cox model with all predictors for OS. For random survival forests OOB can be calculated easily, because during bootstrapping around 63% of the samples is used to grow a forest of trees and 37% stays hold-out. For RSF, we used bootstrap sub-sampling of 5000 patients on the training

set for each of 250 trees because of the computational intensity and the memory demanding forest R object (1 gb large). Note also that sub-sampling can increase the generalization performance of the forest on new "unseen" data. However, this parameter could also be tuned to achieve the lowest possible error.

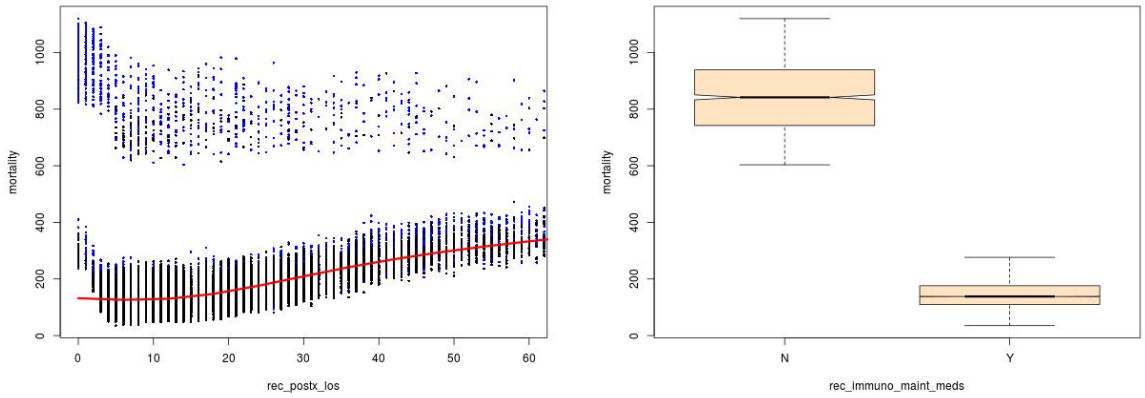
We performed out-of-bag Cox analysis 250 times. Stratified sampling with replacement (bootstrap sub-sampling) of 5000 patients from the training data was performed . In bag data were used to train the model and out-of-bag data were used to validate. Error rate for Cox model was calculated as 1 - C = 0.33. RSF had an error of 0.296. This difference in accuracy performance demonstrates the ability of RSF to perform much better than Cox in terms of OOB analysis due to their natural construction. For cohesion we want to mention that error rate for RSF is calculated using the ensemble cumulative hazard estimate the equivalent of linear predictors for the Cox model.

Plots of mortality

Plots of mortality for the 2 most predictive variables for overall survival *length of stay at hospital* and *intake of immuno-maintenance meds* are shown in 5.11. Mortality values represent estimated risk for each individual calibrated to the scale of number of events. For instance, if an individual i has a mortality value of 100 then in case all individuals had the same value as i, an average of 100 events would be anticipated. The total number of events is equal to the sum of estimated CHE according to the conservation of events principle.

Anticipated mortality (smoothed red line) increases as length of stay in hospitalization is longer than 10 days. Furthermore, patients who got *immuno_maint_meds* have a marginal mortality of close to 100 while the group who could not take medication shows mortality of 850.

Figure 5.11: Marginal effects of variables *rec_posx_los* and *rec_immuno_maint_meds* on mortality (overall survival). On the left plot points in blue correspond to events and black points to censored observations.



5.2.2 Failure-free survival

Short discussion for FFS is provided in this section.

Tuning the parameters

A) Preliminary grid search for training OOB error (almost identical to leave-one-out cross-validation) using bootstrap sub-sampling of 1000 persons and 500 trees for possible parameter values *nodesize* (5, 10, 25, 40, 55, 70) , *mtry* (11, 17, 23, 29, 35, 41, 47) and *nsplit* (3, 5, 7, 9, 11):

- Log-rank splitting rule best combination was (nodesize = 70, mtry = 41, nsplit = 5) with OOB error 0.301.
- For log-rank score splitting rule (nodesize = 55, mtry = 11, nsplit = 3) with OOB error 0.348.

B)Grid search for OOB training error using bootstrap sub-sampling of 5000 persons and 500 trees, for possible parameter values *nodesize* (10, 20, 35, 50, 70, 85, 100, 120) , *mtry* (11, 17, 23, 29, 35, 41, 47) and *nsplit* (3, 4, 5, 6, 7):

- Log-rank splitting rule best combination (nodesize = 70, mtry = 23, nsplit = 6) with OOB error 0.2946.
- Log-rank score splitting rule (nodesize = 50, mtry = 11, nsplit = 6) with OOB error 0.3416.

The log-rank splitting rule outperformed the log-rank score rule as in the case of overall survival.

C) 5-fold cross validation for the log-rank rule with 500 trees and bootstrapping sub-sampling of 5000 patients resulted in the same hyper-parameter combination (nodesize = 70, mtry = 23, nsplit = 6). Mean error over the 5-folds was 0.295. Similarly to the case of overall survival, 5-fold cross validation and direct training OOB error suggested same parameters. Visualization of the 5-fold cross validation is shown in figure 5.12. Best combinations are located close to the center of grid (dark blue dots).

Next step was to identify the number of trees that stabilize the OOB training error to get a stable forest. Search was applied in block of trees from 5 till 500. Figure 5.13 shows that with 200-250 trees error rate stabilizes at around 0.295. As a consequence, we decided to use 250 trees to grow the forest for FFS to have same size as OS.

Final model for FFS

Number of leaves per tree and 5 most frequent variables are depicted in graph 5.14. Most of the trees have grown 100-120 terminal nodes. In contrast to OS, terminal nodes were 145-165. The large difference is caused by parameter *nodesize*. For OS, the best choice for nodesize was 50 whereas for FFS best choice is 70 leading to shallower trees. From the most frequent variables, *rec_posx_los* was the one mostly used. In comparison with OS, categorical variable *rec_immuno_maint_meds* was often used for splitting. This might be an indication that amount of bias in forest is reduced by setting *nsplit* = 6 meaning randomized splitting.

Variables with the highest VIMP on training and those predicted on test set are shown in tables 5.6 and 5.7. *Rec_postx_los* is the strongest predictor with VIMP 0.066 on train and predicted VIMP of 0.047 on test set. As for OS, the 2nd most predictive

Table 5.6: Variables with highest importance on training set using random survival forests (FFS).

Variable	VIMP	Variable	VIMP
rec_postx_los	0.0662	can_race_srtr	0.0014
rec_immuno_maint_meds	0.0417	pretxstatus	0.0011
donorage	0.0071	donortype	0.0009
can_acpt_hcv_pos	0.0047	hcc	0.0009
hcv	0.0042	can_last_serum_creat	0.0008
recipientage	0.0027	diab	0.0007
etiology	0.0022	rec_tumor	0.0006
retransplantation	0.0020	can_last_bili	0.0005
don_htn	0.0015	can_last_inr	0.0005
lifesupport	0.0014	prev_abdom_surg	0.0004

Table 5.7: Most important predicted variables on the test set using random survival forests (FFS).

Variable	VIMP	Variable	VIMP
rec_postx_los	0.0473	lifesupport	0.0010
rec_immuno_maint_meds	0.0415	pretxstatus	0.0008
donorage	0.0069	can_last_serum_creat	0.0008
can_acpt_hcv_pos	0.0038	diab	0.0007
hcv	0.0038	hcc	0.0006
etiology	0.0020	donorcod	0.0006
retransplantation	0.0018	donortype	0.0005
recipientage	0.0013	rec_tumor	0.0004
don_htn	0.0013	rec_functn_stat	0.0004
can_race_srtr	0.0010	last_dial_prior_week	0.0003

Table 5.8: Most important variables using minimal depth on grown training forest for failure-free survival.

Variable	Depth	VIMP	Variable	Depth	VIMP
rec_immuno_maint_meds	1.516	0.0417	hcv	9.480	0.0042
rec_postx_los	1.960	0.0662	can_last_inr	9.516	0.0005
donorage	4.144	0.0071	don_sodium	9.768	0.0001
recipientage	6.044	0.0027	don_tot_bili	9.880	0.0002
can_acpt_hcv_pos	8.056	0.0047	don_hgt_cm	10.056	0.0001
retransplantation	8.128	0.0020	can_last_bili	10.128	0.0005
can_last_serum_sodium	8.352	0.0002	don_inr	10.384	0.0002
etiology	8.644	0.0022	pretxstatus	10.640	0.0011
can_last_serum_creat	8.684	0.0008	can_race_srtr	10.648	0.0014
rec_hgt_cm	9.048	0.0003	don_hematocrit	10.796	0.0001

Figure 5.12: Grid search on a 3D space for RSF with log-rank split rule: FFS. Minimum error of 5-fold cross validation was found at combination (nodesize = 70, mtry = 23, nsplit = 6).

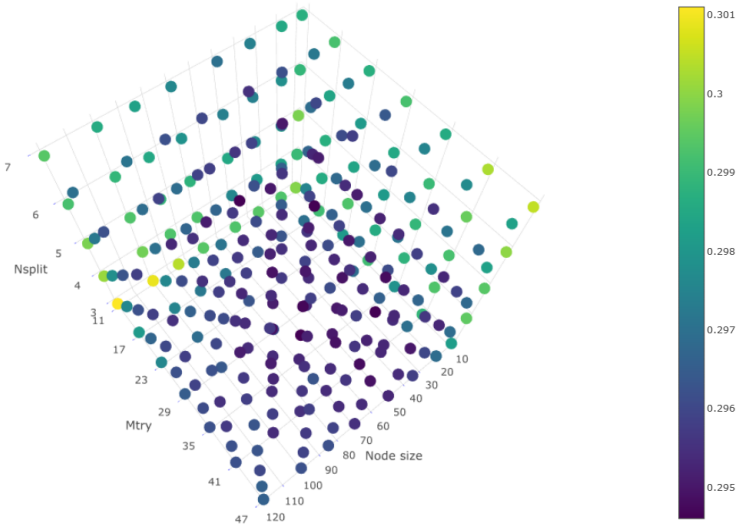
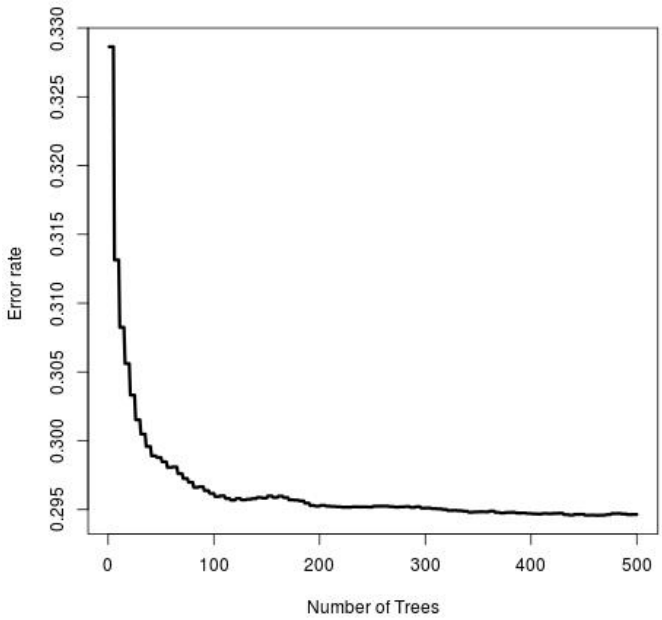


Figure 5.13: Number of trees as a function of error rate for FFS.



variable is *rec_immuno_maint_meds* (0.042 error decrease on train and 0.041 on test). Variables with negative importance on both sets were *don_vasodil* (donor vasodilators) and *rec_cmv_stat* (Pre-Tx Serology Test Results: Cytomegalovirus).

Confidence intervals and standard errors for VIMP using stratified sub-sampling are illustrated in figure 5.15.

Interpretation of the topology of the trees can be found in table 5.8. For the most predictive variables, we show minimal depth compared to root node and the

Figure 5.14: Number of terminal nodes for each tree in the forest (left panel) and number of times a split occurred on 5 most frequent variables for all trees (right panel) for FFS.

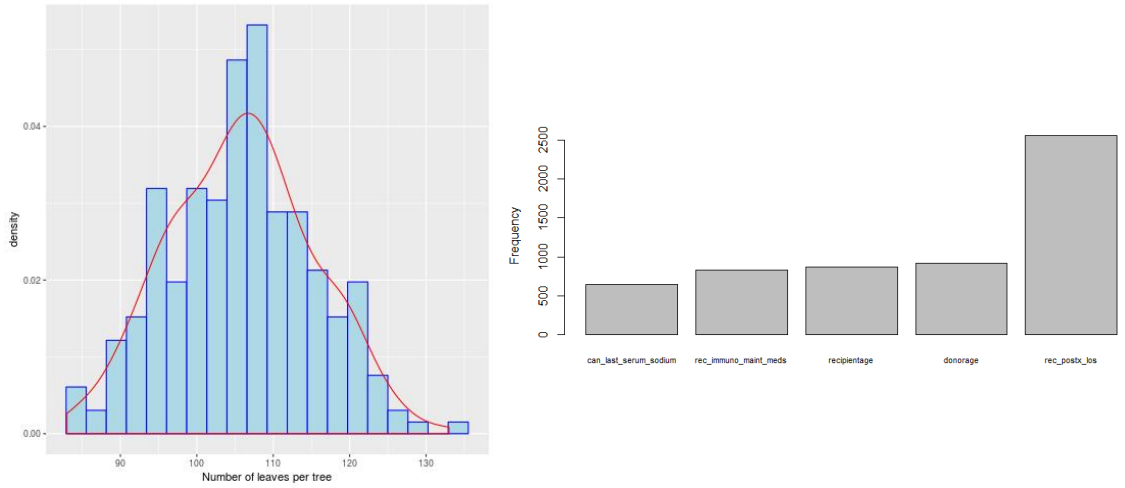
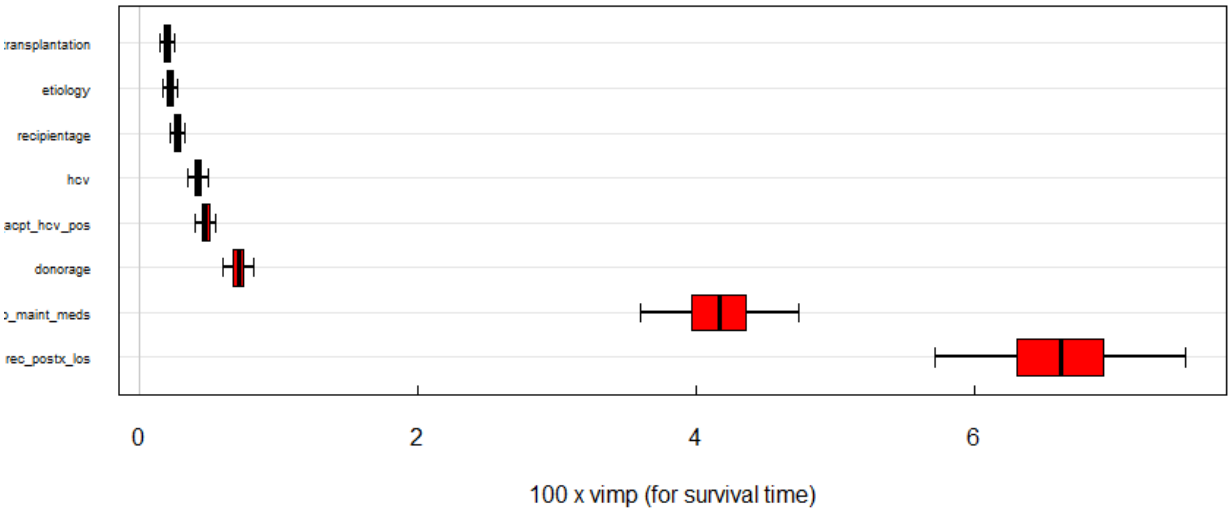


Figure 5.15: Confidence intervals and standard errors for VIMP using sub-sampling (FFS). On the x-axis VIMP is scaled by 100.



equivalent VIMP. Variables with the smallest average depth were *immuno-maintenance suppression* (1.516) and *length of stay* (1.960). Note again that minimal depth and VIMP are two separate methods for interpretability so their results may be different. On closer examination variables *retransplantation* (Did patient have a transplant surgery before?), *don_htn* (was donor hypertensive?) and *lifesupport* (was patient on life support?) have stronger impact for FFS compared to OS.

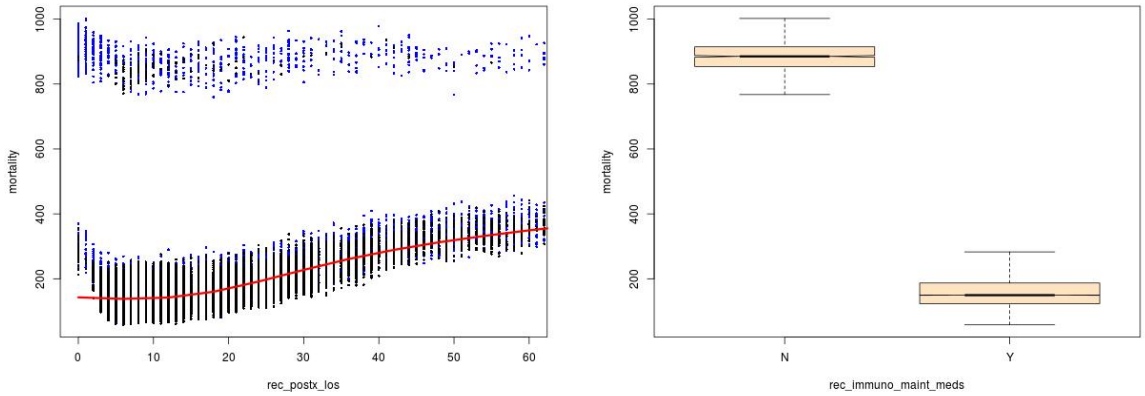
Comparison of RSF with Cox model in OOB error

Comparison of OOB training errors was performed for RSF and Cox model with all variables present. For OOB Cox analysis we used a stratified subsample of 5000 patients and we iterated 250 times. Error rate was 0.333 for Cox model and 0.295 for RSF. Similarly with OS, RSF outperformed Cox model for OOB analysis.

Plots of mortality

Marginal mortality plots of *rec_postx_los* and *rec_immuno_maint_meds* are shown in figure 5.16. Numbers are again calibrated to the scale of events. Graphs are very similar to those of overall survival. Mortality values are slightly increased as failure-free survival captures the first event between liver-failure and death.

Figure 5.16: Marginal effects of variables *rec_posx_los* and *rec_immuno_maint_meds* on mortality (failure-free survival). On the left plot points in blue correspond to events and black points to censored observations.



5.2.3 Discussion

In this section, we applied random survival forests an ensemble tree method which is an extension of classification and regression trees (CART) for right censored survival data.

For RSF, there is a measure used for model training. This is defined as 1 - C, where C is the concordance (is calculated using the ensemble mortality score). C-index relies on ordering of subjects and it is invariant to time dependency and therefore is a well established global measure. A carefully trained RSF can achieve high level performance. Furthermore, as there is no limitation in the number of trees that can be grown, a forest can achieve great stability naturally after a particular block size of trees.

We discussed the strategies we followed to tune parameters over a 3-D grid using 2 alternatives: training OOB error and the default method used in machine learning: 5-fold cross-validation. Both resulted in the same best combination showing that for UNOS data OOB error might be a fast alternative for estimation of the true generalization error.

During training process, we encountered computational obstacles (memory and time limitations) which downgraded the level of search efficiency and we chose to use stratified sub-sampling of 5000 individuals. This limited the size of individuals used to grow each tree but may actually reduced the generalization error on test set as extra randomization was inserted in the process. This number of patients was chosen without cross validating the parameter. It is possible that another subset of patients would produce better performance on the test set.

From the splitting rules, *log-rank* was by far more accurate and computationally more efficient compared to the rival *log-rank score* rule. In general, hyper-parameters pointed larger values for *nodesize* and *mtry* than the package defaults: 3 for *nodesize* and 11 for *mtry*. Larger values of *nodesize* and *mtry* led to shallower but more accurate trees. A more representative subset of variables was selected using $mtry = 23$ or 35.

Parameter *ntree* was also examined to define a suitable number of trees that does not affect the stability of the forest. Plots 5.9, 5.14 show how *nodesize* can regulate the tree depth substituting parameter *nodedepth* (maximum depth) which was another candidate parameter for tuning (discussed in section 3.3.3).

An extensive emphasis was given to the ability of RSF to give relative interpretation through VIMP and minimal depth selection. Consistently variables *rec_postx_los* and *rec_immuno_maint_meds* were found as strongest predictors for both overall and failure-free survival. Variables *retransplantation*, *don_htn* and *lifesupport* had increased importance for FFS. RSF has the ability to identify non-prognostic factors by assessing negative VIMP. Graphs 5.11, 5.16 showed the marginal impact for the 2 most predictive variables for mortality (expected total number of deaths).

Note that interpretation is not restricted to individual variables. Package of implementation randomForestSRC allows the user to estimate joint VIMP (VIMP for combinations of 2 or more variables). In this way it is possible to find combinations of variables that are the most important for survival. However, as at least all pairwise combinations of variables need to be examined this is computationally infeasible for UNOS dataset with 106 variables.

Variable *length of stay* was given increased attention by RSF compared to the Cox models and was found consistently as one of the 2 strongest prognostic factors by VIMP and minimal depth methods on the data. It is known that RF favor splitting on continuous variables (Loh and Shih, 1997), so this finding needs to be further investigated to see if *length of stay* is so strong a predictor for RSF. Software of implementation `randomForestSRC` allows specification of parameter *nsplit* (number of split points randomly selected per candidate variable) to mitigate amount of bias. Despite the use of it, there were several warnings for selection bias for both OS and FFS (most frequently used split variables, most important variables for VIMP and especially minimal depth). Wright et al. (2017) recently showed that unbiased split variable selection can be achieved for RSF using maximally selected rank statistics to the split point. Optimal split is determined using a statistical test for binary splits which adjusts for multiple testing in numerical variables. More emphasis should be given in the future in determining if RSF with the default *log-rank* split rule are a suitable method for datasets with both numerical and categorical predictors.

5.3 Neural Networks for survival analysis

We applied Biganzoli's method on UNOS data. As discussed in chapter 3 section 3.4.4 , data were transformed to a long format and time variable into 12 distinct time intervals representing years since transplantation. Prognostic factors as well as time interval were inserted in the network as inputs. Output node was a large target vector with 0 if an event did not occur on that interval and 1 if an event did occur.

Figure 5.17: Distributions of event times for UNOS data. Panel A: Overall survival, Panel B: Failure-free survival.

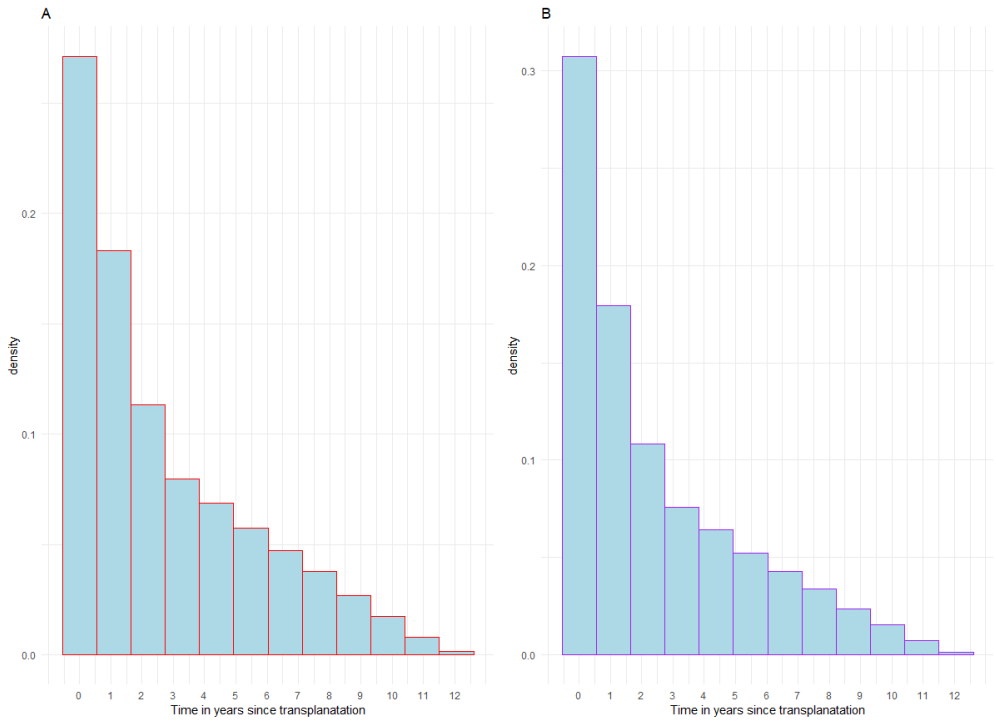


Figure 5.17 shows distribution of event times for overall and failure-free survival.

Steps to construct a confusion matrix

- Split the training data (2/3 of the original dataset) into 5 folds. Each time 4 folds were used to create the long training format (section 3.2) and 1 fold was used to create a validation set. For the validation set, as the exact survival time was supposed to be unknown, we replicated each patient for all 12 intervals.
- Feed the network in `R` (package `keras`) using as inputs the 128 variables and the distinct time interval. Different activation functions were applied for the input - hidden layer. More precisely 3 activation functions were investigated the sigmoid, the hyperbolic tangent (tanh) and the rectified linear unit (ReLU). We used dense layers and we inserted different parameters of dropout.
- Output layer contained a single unit with the sigmoid activation function between hidden and output layer. Binary cross entropy was used to specify the conditional probability of dying at each interval of the single output unit. Optimizer algorithm used was the stochastic gradient descent (SGD) with different parameters of learning rate and momentum.

- Process was repeated for 10 epochs. Trying to diminish class unbalance between censoring-events ratio (70.9% - 29.1% for OS and 68.7% - 31.3% for FFS) , we tested the behavior of setting the weight of event class to 2 (weak class).
- After the model was fitted, the condition hazard probabilities were estimated on validation set. For each individual, the probabilities at each interval were found $S(t) = \prod_{k:t_k \leq t}(1 - h_k)$.
- Survival probability of the actual time interval was detected for each patient.
- Using as cut-off of 0.5 predict the class (dead - alive) for a specific individual. When the survival probability was larger than 0.5 person was assumed alive (class 0). In a different case, class 1 was specified.
- Knowing the true status of individuals, construct a confusion matrix for the predicted class versus the actual class (example on table 4.1). Having calculated this matrix a variety of metrics can be used namely: accuracy, sensitivity, specificity, precision and f1 score.
- As neural networks rely on minimization of the cross-entropy error the value of binary cross-entropy was used to identify tuning combinations that lead in high performance in terms of accuracy.
- Another metric used for survival data was included - namely the integrated Brier score (IBS) (section 4.2).

Biganzoli used only the sigmoid (logistic) activation function for both input-hidden and hidden-output layers (Biganzoli et al., 1998). Here 2 more activation functions for the input-hidden layer are investigated: the hyperbolic tangent (tanh) and the rectified linear unit (ReLU). Biganzoli's network is a single hidden layer feed forward neural network. We will test a NN with 2-hidden layers with the second layer having same number of nodes as the first. We compare the performance of such a network that has an overabundance of weights with a single hidden layer NN. It is important to investigate if a higher level of performance can be reached or if an overfit is obtained.

In NN for classification problems, accuracy is the performance measure that is usually considered to select the best tuning parameters of cross-validation. Note that in neural networks for survival analysis there is not a well established measure of cross-validation performance. For this reason we use the following strategy:

1. we identify the 2 best combinations of tuning parameters that minimize the cross entropy for each one of the three activation functions.
2. We report accuracy, sensitivity, specificity, precision, f1 score and IBS on a table.
3. Based on a close inspection of all metrics of part 2, we select the tuning parameters that result in a good model overall giving priority to accuracy.

5.3.1 Overall survival

Experiments for one hidden layer part 1

The number of parameters that can be tuned with the `keras` library is large. Here we focus on tuning of 5 parameters *node size, dropout rate, learning rate, weak class*

Table 5.9: Best 2 combinations per activation function in terms of cross-entropy for neural networks with 1 hidden layer: overall survival (experiments part 1).

	Activation function of input-hidden layer nodes					
	1) Sigmoid	2) Sigmoid	1) Tanh	2) Tanh	1) ReLU	2) ReLU
Total weights	3931	6551	3931	5241	10481	13101
Node size	30	50	30	40	80	100
Dropout	0.2	0.2	0.2	0.2	0.2	0.2
Learning rate	0.1	0.1	0.1	0.1	0.1	0.1
Momentum	0.9	0.9	0.9	0.9	0.9	0.9
Weak class	1	1	1	1	1	1
Cross entropy	0.208	0.208	0.208	0.208	0.202	0.202
Accuracy	0.696	0.706	0.685	0.706	0.707	0.700
Sensitivity	0.096	0.086	0.116	0.088	0.142	0.147
Specificity	0.943	0.960	0.919	0.960	0.938	0.927
Precision	0.408	0.478	0.388	0.482	0.487	0.454
F1 score	0.155	0.145	0.174	0.146	0.220	0.222
IBS	0.174	0.185	0.172	0.195	0.184	0.180

weight and momentum. The 129 input features require great care to produce suitable connection weights between the neurons. As part of the first experiments we performed grid search using:

- node size (10, 20, 30, 40, 50, 60, 70, 80, 90, 100)
- dropout rate (0.2, 0.3, 0.4)
- learning rate (0.01, 0.1)
- weak class weight (1, 2)
- momentum (0.8, 0.9)

This resulted in a 5-D grid search of 240 combinations.

During cross-validation of NN, we are looking for a combination that minimizes the cross entropy. Such a model usually reaches a high accuracy as well. Best combinations are presented in table 5.9. The values of dropout rate, learning rate, momentum and weak class weight were the same for all combinations. Node sizes were low for the sigmoid and the tanh functions. Lowest cross-entropy was reached by the 2 combinations of ReLU function with highest accuracy 0.707 by ReLU model 1. Models with ReLU activation achieved a much higher sensitivity performance identifying more patients that died. All combinations have poor performances in sensitivity. Specificity was the highest for sigmoid combination 2 and tanh combination 2 (0.96). ReLU combinations dominated in terms of F1 score. Finally, IBS reached lowest score for combination with 30 nodes of tanh function. Based on different metrics another tuning combination can be selected.

Table B.7 in appendix B shows results for the best models in terms of accuracy. High accuracy values were found in more combinations than those that minimized cross-entropy loss function. Highest accuracy was reached by ReLU activation function

Table 5.10: Best 2 combinations per activation function in terms of cross-entropy for neural networks with 1 hidden layer: overall survival (experiments part 2).

	Activation function of input-hidden layer nodes					
	1) Sigmoid	2) Sigmoid	1) Tanh	2) Tanh	**1) ReLU**	2) ReLU
Total weights	9171	9826	9171	11136	**11791**	14411
Node size	70	75	70	85	**90**	110
Dropout	0.1	0.1	0.1	0.1	**0.1**	0.1
Learning rate	0.2	0.2	0.2	0.2	**0.2**	0.2
Momentum	0.9	0.9	0.8	0.8	**0.9**	0.9
Cross entropy	0.205	0.205	0.206	0.206	**0.199**	0.199
Accuracy	0.700	0.693	0.706	0.702	**0.707**	0.704
Sensitivity	0.135	0.150	0.113	0.121	**0.158**	0.149
Specificity	0.931	0.916	0.949	0.940	**0.932**	0.932
Precision	0.455	0.431	0.498	0.459	**0.496**	0.478
F1 score	0.207	0.221	0.181	0.190	**0.237**	0.226
IBS	0.176	0.171	0.187	0.183	**0.191**	0.186

(0.709) at both combinations. Nevertheless, very high accuracy values on the table come at a cost in sensitivity - which is a warning sign for over-fitting.

Next, we examined the combinations of hyper-parameters with different activation functions that lead to the lowest IBS (summarized Brier score). Results are given in appendix B.8. Note that the IBS observed for those hyper-parameter combinations is much lower than that of tables B.7 and 5.9. All activation functions achieved low IBS with the sigmoid having the lowest 0.127 and 0.128. Sensitivity was increased and reached 0.311 using the hyperbolic tangent activation function. However, accuracy was dramatically decreased and cross-entropy substantially increased. All combinations had node sizes 10 or 20. This means that best models in terms of IBS score have bias towards very small node sizes. Usually node size of 10 or 20 are very low to capture non-linearities in networks with more than 100 features and at least node size of 50-60 is required to get a well behaving network.

Experiments for one hidden layer part 2

For the second part of the experiments, we focused more on large node sizes. This parameter is the most fundamental for a NN as the number of weights that need to be generated is based on this parameter. Larger node sizes imply more weights. The combinations of parameters tested were:

- node size (70, 75, 80, 85, 90, 95, 100, 105, 110, 115, 120, 125, 130)
- dropout rate (0.1, 0.2, 0.3)
- learning rate (0.01, 0.1, 0.2)
- momentum (0.8, 0.9)

Weak class weight was kept stable at 1. In total there were 234 combinations on a 4-D grid space.

From now on, we present only the tables for the best combinations of hyper-parameters based on the minimized value of cross-entropy. Tables with respect to maximum accuracy and minimum IBS for this part of the experiments can be found in the appendix (tables B.9 and B.10).

In table 5.10 that lowest cross-entropy error is achieved with the same combination of dropout = 0.1 and learning rate = 0.2 for all cases. Node size of best models varies from 70 to 110 nodes. A feed forward NN with one hidden layer and 110 nodes has 14411 = (129 + 1)*110 + (110 + 1)*1 weights which means 14411 parameters to be estimated!

Lowest cross-entropy error was again found by the NN with ReLU activation functions. Highest accuracy was achieved by combination 1 of ReLU in the table (0.707). This combination had also the highest sensitivity and F1 score. Lowest IBS values were found for the combinations with the sigmoid activation function. Combination that is best for the majority of the metrics is that with ReLU activation function node size = 90, dropout = 0.1, learning rate = 0.2 and momentum 0.9. An alternative could be the sigmoid activation function, node size = 75 dropout = 0.1, learning rate = 0.2 and momentum = 0.8

Final parameter selection one hidden

In the previous subsection, we distinguished a number of tuning parameters that can be selected. ReLU activation function is best option as it leads to higher accuracy, sensitivity and F1 score. On the other hand, the sigmoid function produces consistently lower IBS. In terms of precision and specificity there is no clear winner. Considering as the best NN the one that can lead to the minimization of the loss function and the highest accuracy possible, we select the network with (activation function = ReLU, node size = 90, dropout rate = 0.1, learning rate = 0.2, momentum = 0.9 and weak class weight = 1).

Examining stability of the network and relative importance

Neural networks are notorious for being quite unstable. Massive amount of training is needed to achieve a stable network. To examine this, we used the parameters chosen in the previous paragraph to apply the network repeating 4 times. Figure 5.18 shows the results. It can be seen that 3 iterations produced similar survival probabilities and Brier scores but one iteration (blue lines) produced a large difference in results.

To further examine stability we used Garson's connection weights method - for the same runs of the network - to get the relative importance of the variables. Top 5 variables are visualized in graph 5.19. Variable *immuno-maintenance* has by far the highest relative importance. The 2^{nd} most contributing variable is *time interval*. For the other 3 most contributing variables there is a divergence from network to network. In graph 5.19A, variables length of stay, re-transplantation and life support in 5.19B: *re-transplantation*, *life support* and *Latin ethnicity of recipient*, in 5.19C: *length of stay*, *tumor* and *re-transplantation* and in 5.19D: *re-transplantation*, *Latin ethnicity* and *Symptomatic Cerebrovascular disease*. Therefore, the network is able to capture consistently the 2 most contributing variables for the prediction but it is unstable concerning the others.

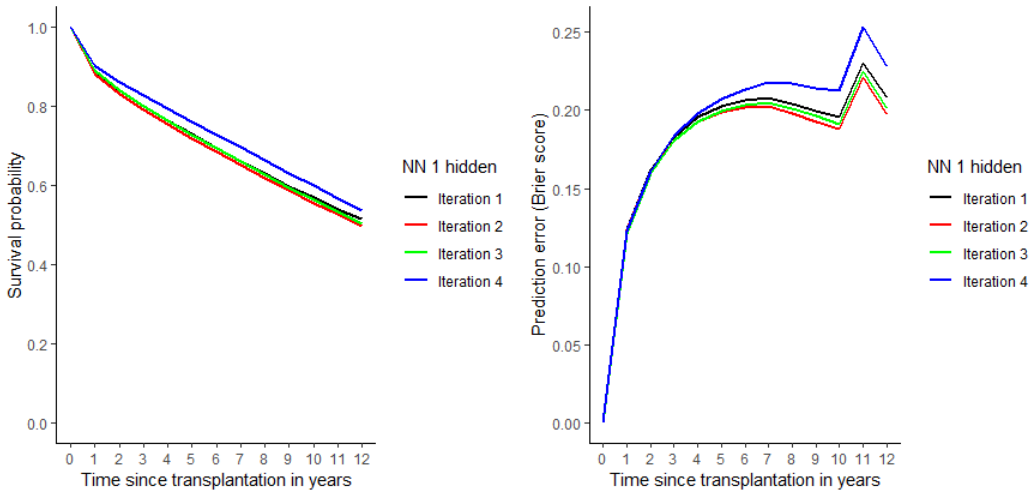
Relative importance for top 20 variables can be observed in table 5.19. All prognostic factors have more than 1% contribution to the weight distribution of the

Table 5.11: The 20 variables with highest relative importance using Neural Network with 1 hidden layer (OS).

Variable	Rel. Imp.	Variable	Rel. Imp.
rec_immuno_maint_medsY	0.1355	prev_abdom_surgY	0.0151
interval	0.0790	portal_hyperten_bleedY	0.0151
rec_postx_los	0.0377	pretxstatusIC_UNIT	0.0146
retransplantationRetransplantation	0.0335	etiologyOther_unknown	0.0130
don_hla_typY	0.0299	etiologyMalignant	0.0129
rec_functn_statTotal_assistance	0.0286	can_ethnicity_srtrNLATIN	0.0126
lifesupportLifesupport	0.0247	maligY	0.0125
rec_tumorY	0.0231	etiologyHCV	0.0124
donortypeDCD	0.0172	recipientage	0.0122
can_acpt_extracorp_liY	0.0155	last_dial_prior_weekY	0.0118

network. Strongest predictors for the network are *rec_immuno_maint_medsY* (0.136) and *interval* (0.079). Variable *rec_postx_los* is the 3rd most contributing, yet is not as predictive as it has been shown in the results of the random forests. The forth and fifth most predictive variables are *retransplantation* (0.038) and *don_hla_typY* (0.030). Note that instability of the network can affect its relative importance (look figure 5.19).

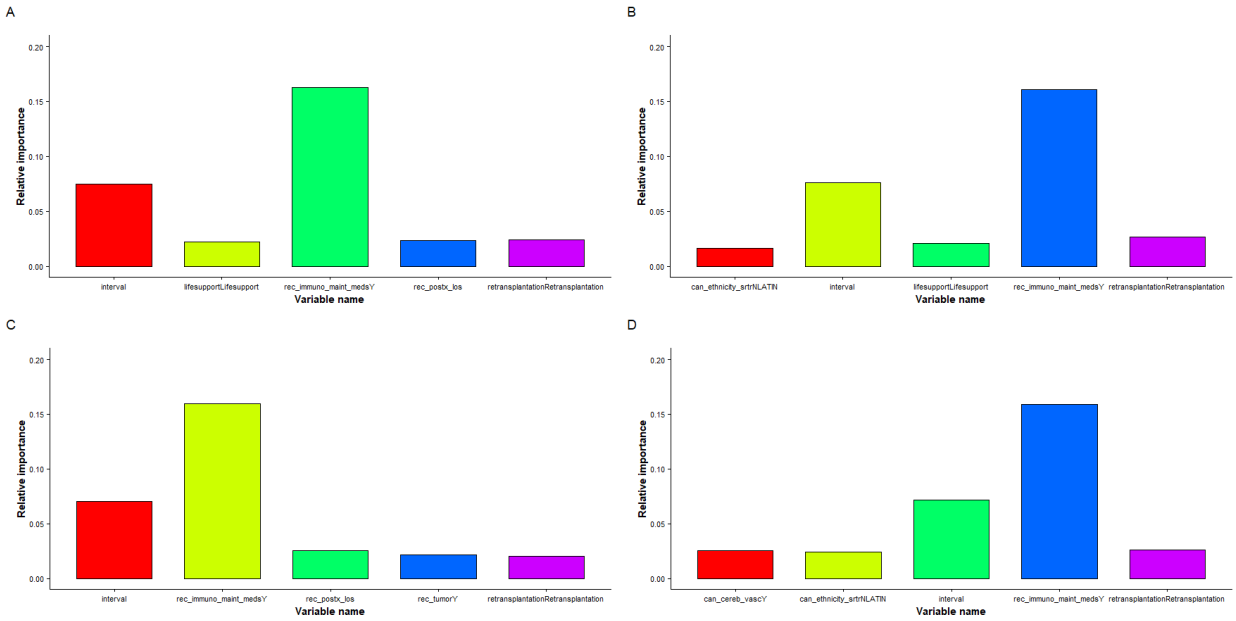
Figure 5.18: Mean survival probabilities over time (left panel) and Brier score over time (right panel) for a neural network with 1 hidden layer, ReLU activation function with node size = 90, dropout rate = 0.1, learning rate = 0.1, momentum = 0.8 and weak class weight = 1 run 4 times (OS).



Experiments for two hidden layers part 1

We repeated the 5-fold cross-validation adding another hidden layer to the neural network. Hyper-parameters search was made on a similar grid to that of 1 hidden layer (part 1). Please note that node size here refers to the size of the first as well as

Figure 5.19: Relative importance for top 5 variables using Garson connection weights method on multiple runs. Neural network with 1 hidden layer, ReLU activation function node size = 90, dropout rate = 0.1, learning rate = 0.1, momentum = 0.8 and weak class weight = 1 (OS).



that of second hidden layer. For instance, node size of 50 means node size of the first hidden layer 50 and node size of the second hidden layer 50. In this way, we avoided adding another tuning parameter that would increase the grid space dramatically.

In the right panel of figure 5.20, we can see that as the node size increases there is an exponential increase in the number of weights for neural network with 2 hidden layers.

Aim is to investigate if there are differences in the performance measures of a NN with 2 hidden layers compared to a simpler NN with a single hidden layer. Performance metrics are presented (table 5.12) for the best 2 combinations in terms of cross entropy for the 3 activation functions. Same activation function is used for the input-hidden 1 and hidden 1-hidden 2 layer.

All combinations have dropout = 0.2 learning rate = 0.1, momentum 0.9 and weak class weight = 1, same as for the best cross-entropy neural networks with 1 hidden layer (table 5.9). ReLU function achieved the minimum cross-entropy value (0.209). Accuracy and sensitivity is decreased. IBS is also decreased especially for ReLU function. It can be observed that the sigmoid function has led to dramatic over-fitting with the specificity value close to 1 and the sensitivity value close to 0. Interestingly, the tanh function has a better overall performance than ReLU. It reached better accuracy, specificity and precision for its combinations. Inspecting closely the 2 combinations we give an advantage to combination 1 due to higher sensitivity, f1 score and lower IBS score. Values in sensitivity $\rightarrow 0$ and specificity $\rightarrow 1$ are a warning of increased over-fitting.

Combinations with the lowest IBS (close to 0.12), results were similar to that of table B.8. Node size for each hidden layer was 10, dropout 0.4, learning rate 0.01, momentum 0.8 or 0.9 and weak class weight 2. Cross-entropy had values of

Figure 5.20: Left panel: A representation of a feed-forward neural network with 2 hidden layers (output layer has 1 node in our case). Right panel: Number of weights of 1 and 2 hidden layer feed-forward neural networks as a function of node size.

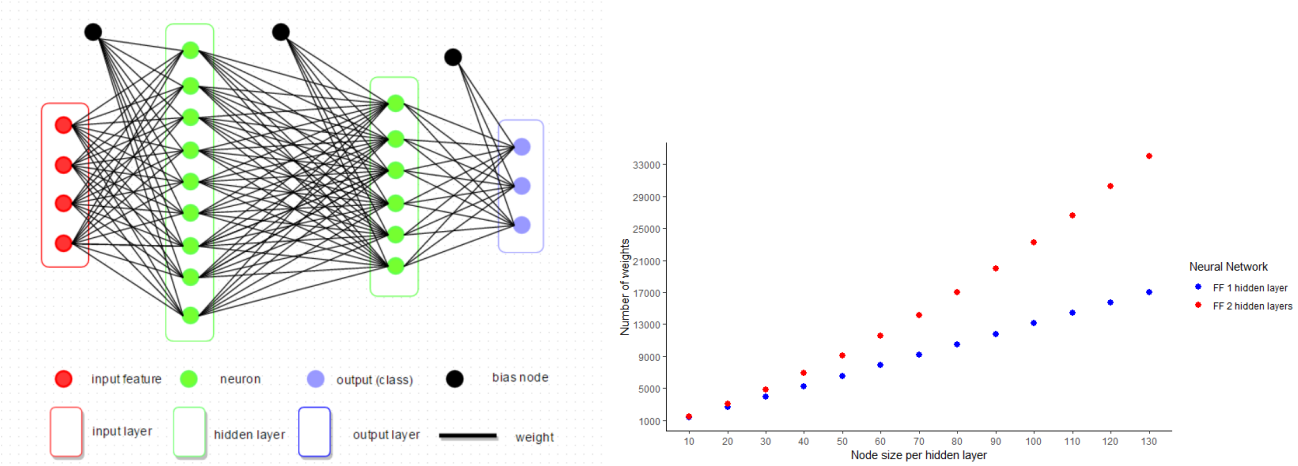


Table 5.12: Best 2 combinations per activation function in terms of cross-entropy for neural networks with 2 hidden layers: overall survival (experiments part 1).

	Activation function of input-hidden 1, hidden 1-hidden 2 layers					
	1) Sigmoid	2) Sigmoid	1) Tanh	2) Tanh	1) ReLU	2) ReLU
Total weights	4861	6881	23201	14141	19981	23201
Node size (each hidden)	30	40	100	70	90	100
Dropout	0.2	0.2	0.2	0.2	0.2	0.2
Learning rate	0.1	0.1	0.1	0.1	0.1	0.1
Momentum	0.9	0.9	0.9	0.9	0.9	0.9
Weak class	1	1	1	1	1	1
Cross entropy	0.223	0.223	0.209	0.210	0.209	0.209
Accuracy	0.705	0.703	0.693	0.703	0.682	0.687
Sensitivity	0.007	0.010	0.101	0.074	0.104	0.091
Specificity	0.992	0.987	0.937	0.962	0.920	0.931
Precision	0.259	0.241	0.416	0.447	0.348	0.356
F1 score	0.014	0.019	0.160	0.127	0.159	0.145
IBS	0.174	0.171	0.180	0.188	0.172	0.176

approximately 0.38 far from the best values of 0.21. Accuracy of the combinations was very poor (below 0.5). Thus, a NN with such small node size cannot capture the non-linearities of the input variables and therefore the minimum values combinations as a function of IBS are not representative of an appropriate network.

Experiments for two hidden layers part 2

We repeated the experiments over the following grid space of 126 combinations:

- node size (hidden 1 - hidden 2) (70, 80, 90, 100, 110, 120, 130)
- dropout rate (0.1, 0.2, 0.3)
- learning rate (0.01, 0.1, 0.2)
- momentum (0.8, 0.9)
- weak class weight 1

The same parameter was identified for dropout, learning rate and momentum. Cross entropy was the lowest for models with the ReLU activation function. This time ReLU obtained higher sensitivity values than tanh. Sensitivity and F1 score of the sigmoid functions was very low indicating over-fitting. Lowest IBS score was obtained by ReLU parameters 2. ReLU best combinations surpassed tanh in most metrics.

Final parameter selection two hidden

Choosing between model with ReLU activation or model with tanh we selected in the experiments part 1 (look table 5.12) the combination with (activation = "ReLU", node size = 100, dropout rate = 0.1, learning rate = 0.2, momentum = 0.9 and weak class weight = 1) has an advantage in all metrics but IBS. This is the choice for overall survival with 2 hidden layers.

Examining stability of the network

To check stability of NN, we run the final model 4 times (graph 5.21). The increased number of weights did not provide a more stable network. Values of the performance metrics differ from repeat to repeat even though values are close to the cross-validated ones. Iteration 2 produced a network with the worst Brier score. As a result, the network might obtain values in a certain range although instability and uncertainty in predictions is high.

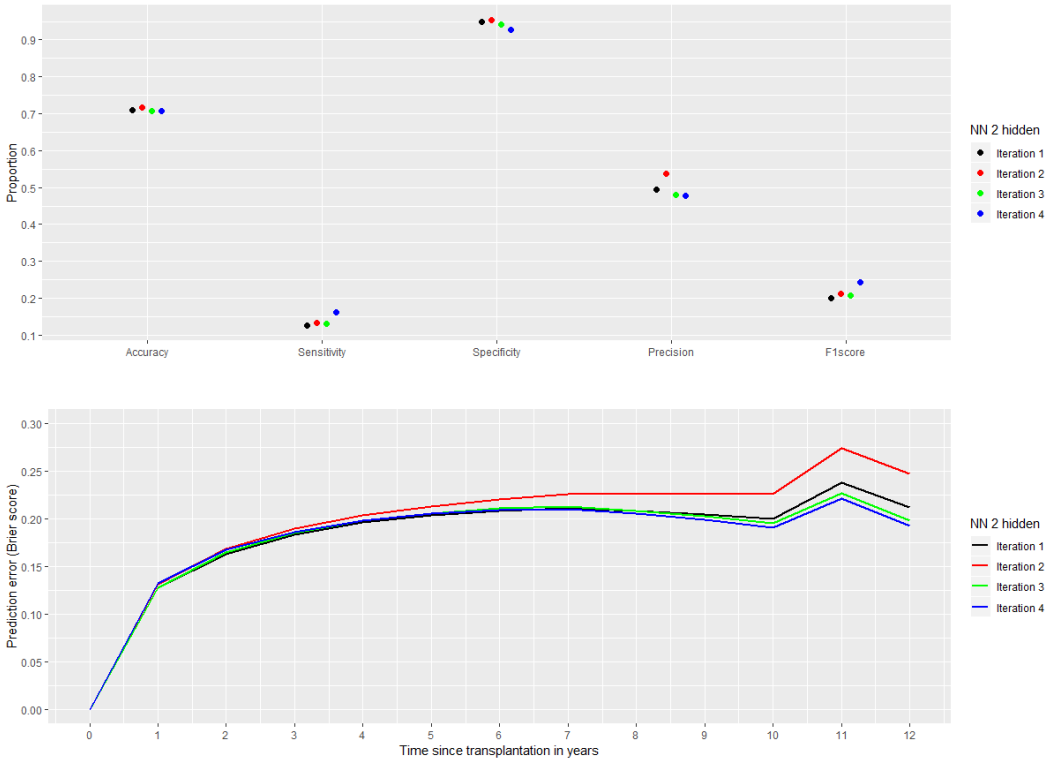
5.3.2 Failure-free survival

All experiments were repeated for failure-free survival. We refer to the hyper-parameters being selected for NN with 1 and 2 hidden layers, as well as stability issues and relative importance. Tables of best tuning parameters with respect to binary cross-entropy can be viewed in appendices section B.2.2.

Table 5.13: Best 2 combinations per activation function in terms of cross-entropy for neural networks with 2 hidden layers: overall survival (experiments part 2).

	Activation function of input-hidden 1, hidden 1-hidden 2 layers					
	1) Sigmoid	2) Sigmoid	1) Tanh	2) Tanh	1) ReLU	2) ReLU
Total weights	14141	16961	16961	19981	**23201**	26621
Node size (each hidden)	70	80	80	90	**100**	110
Dropout	0.1	0.1	0.1	0.1	**0.1**	0.1
Learning rate	0.2	0.2	0.2	0.2	**0.2**	0.2
Momentum	0.9	0.9	0.9	0.9	**0.9**	0.9
Weak class	1	1	1	1	**1**	1
Cross entropy	0.217	0.217	0.206	0.206	**0.203**	0.203
Accuracy	0.687	0.692	0.699	0.703	**0.709**	0.696
Sensitivity	0.041	0.036	0.131	0.134	**0.139**	0.141
Specificity	0.953	0.961	0.932	0.937	**0.943**	0.924
Precision	0.263	0.274	0.493	0.506	**0.507**	0.437
F1 score	0.070	0.063	0.200	0.205	**0.217**	0.212
IBS	0.177	0.181	0.180	0.184	**0.186**	0.175

Figure 5.21: Multiple runs of the feed-forward NN with 2 hidden layers with activation function ReLU for input - hidden 1 and hidden 1 - hidden 2 layers: Overall survival. Top panel: proportion for 5 performance metrics, Bottom panel: Time dependent Brier score.



Experiments for one hidden layer part 1

Tuning parameter combinations:

- node size (10, 20, 30, 40, 50, 60, 70, 80, 90, 100)

- dropout rate (0.2, 0.3, 0.4)
- learning rate (0.01, 0.1)
- weak class weight (1, 2)
- momentum (0.8, 0.9)

Corresponding is table B.11 in appendix B. Minimum cross-entropy is achieved by ReLU functions (0.216). Sigmoid and ReLU are better in terms of accuracy compared to tanh. In terms of sensitivity, ReLU combination 2 (node size = 90) with 0.17 is the best, but the worst in specificity (0.897) even if value is very high. Same combination has highest F1score (0.243) and second best IBS (0.179). In general sigmoid, performed well for much lower node sizes than ReLU and had the highest specificity. We prefer ReLU combination 1 (node size = 80), as it has the highest accuracy of all combinations (0.685), the second best sensitivity (0.156), best precision (0.495) and second best F1score (0.236).

Experiments for one hidden layer part 2

Grid search was done on the following combinations:

- node size (70, 75, 80, 85, 90, 95, 100, 105, 110, 115, 120, 125, 130)
- dropout rate (0.1, 0.2, 0.3)
- learning rate (0.01, 0.1, 0.2)
- momentum (0.8, 0.9)

Results are presented in appendix table B.12. Again ReLU function combinations achieved the lowest cross entropy error with combination 1 reaching the 2^{nd} best accuracy, best sensitivity (0.179) and best F1score (0.263). Sigmoid combination 2 has the highest accuracy (0.688), highest specificity (0.937) and precision (0.513).

Final parameter selection one hidden

Combination with activation sigmoid node size = 75, dropout = 0.1, learning rate 0.2, momentum 0.9, weak class weight = 1 has the best performance metrics overall.

Examining stability of the network and relative importance

Figure 5.22 shows the mean survival probabilities for all patients and time-dependent Brier score. Curves are quite close indicating a more stable performance. In terms of Brier score there is some difference for iteration 2 (red line). Looking at the top 5 variables in relative importance (figure 5.23) names and order in importance is identical suggesting again a much stabler network compared to the equivalent for OS with 1 hidden layer. In table 5.14 the top 20 factors are shown. *Immuno-maintenance suppression* (0.191) is by far the most important followed by *time interval* (0.107), *length of stay* (0.038), *donor type* (donation after circulatory death) with 0.034 and *retransplantation* with 0.029. The remaining variables have less than 0.02 importance. *Donor type*, *recipient: Spontaneous portal hypertensive bleeding*, *races Black and Other* and *portal vein thrombosis* (yes) have an increased relative importance compared to OS.

Figure 5.22: Mean survival probabilities over time (left panel) and Brier score over time (right panel) for a neural network with 1 hidden layer, sigmoid activation function with node size = 75, dropout rate = 0.1, learning rate = 0.2, momentum = 0.9 and weak class weight = 1 run 4 times (FFS).

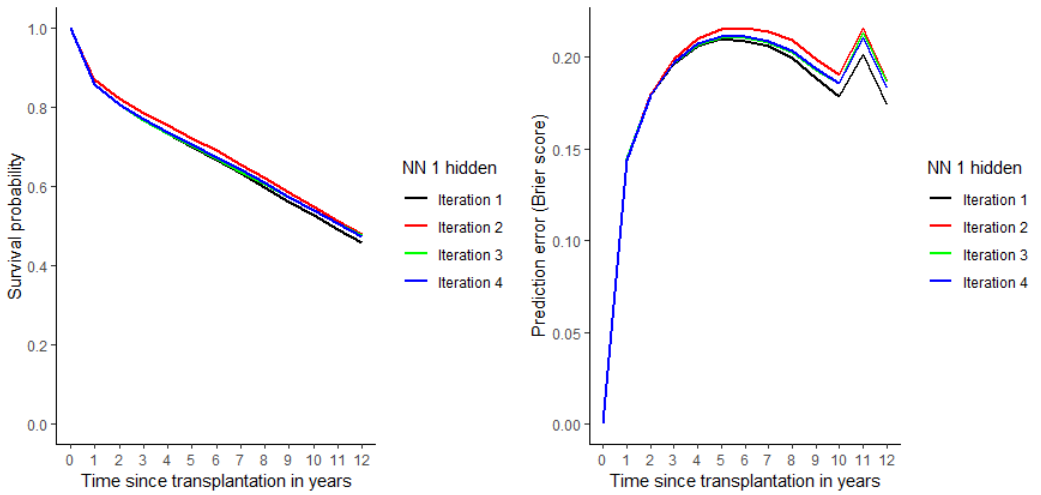
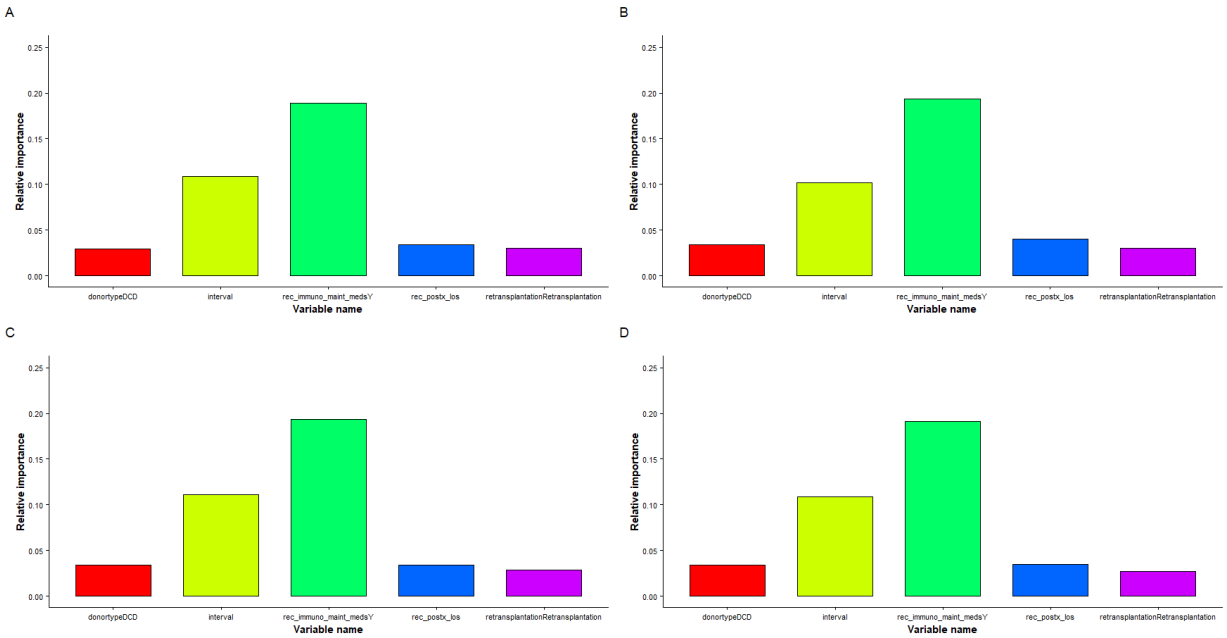


Figure 5.23: Relative importance for top 5 variables using Garson connection weights method on multiple runs. Neural network with 1 hidden layer, sigmoid activation function with node size = 75, dropout rate = 0.1, learning rate = 0.2, momentum = 0.9 and weak class weight = 1 (FFS).



Experiments for two hidden layers part 1

Hyper-parameters:

- node size (10, 20, 30, 40, 50, 60, 70, 80, 90, 100)
- dropout rate (0.2, 0.3, 0.4)

Table 5.14: The 20 variables with highest relative importance using Neural Network with 1 hidden layer (FFS).

Variable	Rel. Imp.	Variable	Rel. Imp.
rec_immuno_maint_medsY	0.1913	prev_abdom_surgY	0.0152
interval	0.1065	can_race_srtrOther	0.0139
rec_postx_los	0.0377	can_acpt_hcv_posY	0.0132
donortypeDCD	0.0337	portal_veinY	0.0122
retransplantationRetransplantation	0.0294	can_ethnicity_srtrNLATIN	0.0121
lifesupportLifesupport	0.0181	albumin2less2	0.0111
rec_functn_statTotal_assistance	0.0176	don_meet_cdc_high_riskY	0.0106
portal_hyperten_bleedY	0.0169	don_hcv_statPositive	0.0096
rec_tumorY	0.0166	don_hla_typY	0.0092
can_race_srtrBlack	0.0166	pretxstatusIC_UNIT	0.0090

- learning rate (0.01, 0.1)
- weak class weight (1, 2)
- momentum (0.8, 0.9)

Experiments are presented in table B.13. Similarly with the case of 2 hidden layers for OS, sigmoid function collapses in term of sensitivity and F1 score close to zero. This means that is an unsuitable choice in terms of event prognosis. Functions tanh and ReLU have much lower value of cross-entropy and much better performance in terms of sensitivity. Tanh combinations have the best accuracy values. Combination 1 has slightly less accuracy and specificity, but better sensitivity, F1 score and smaller IBS. Therefore, it is more preferable.

Experiments for two hidden layers part 2

Grid search:

- node size (hidden 1 - hidden 2) (70, 80, 90, 100, 110, 120, 130)
- dropout rate (0.1, 0.2, 0.3)
- learning rate (0.01, 0.1, 0.2)
- momentum (0.8, 0.9)
- weak class weight 1

The reader can look at table B.14. ReLU combinations have the smallest cross entropy values with tanh close. Again tanh performs better in accuracy, specificity and precision.

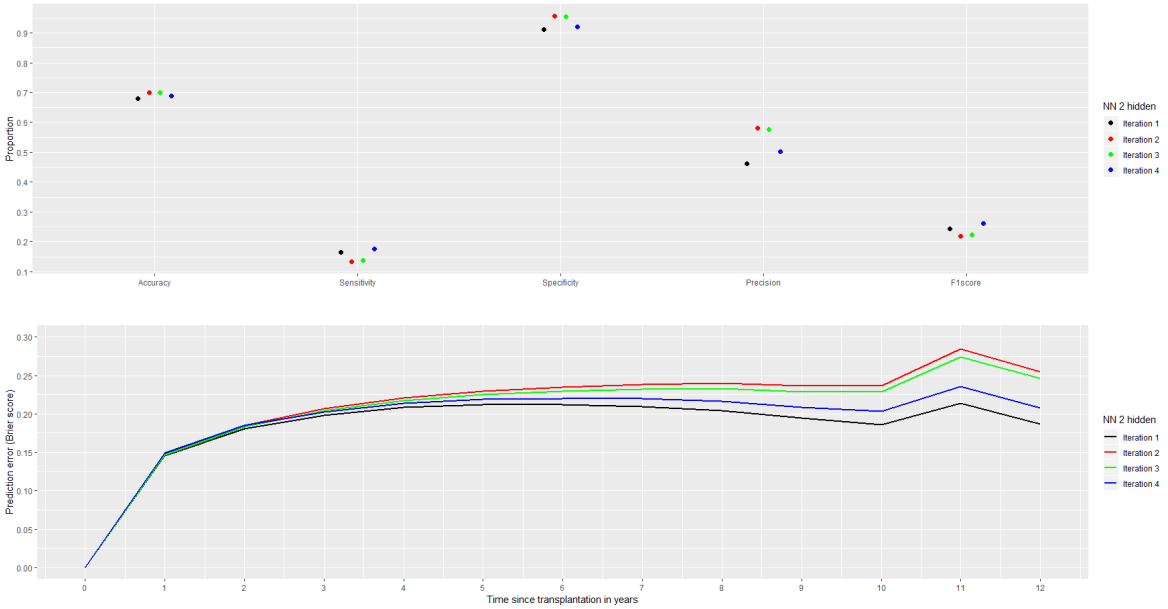
Final parameter selection two hidden

Between the two models, the one with activation tanh function of input-hidden 1 and hidden 1 - hidden 2 layers combination 1 in table B.14 reached higher accuracy, sensitivity, precision and F1 score. To sum up, final choice is (activation function tanh, node size = 100, dropout = 0.1, learning rate = 0.2, momentum = 0.9 and weak class = 1).

Examining stability of the network

We run the neural network 4 times to assess stability. Figure 5.24 shows instability for all metrics. In 4 runs of the network repeats 2 and 3 (red and green) produced similar performances, whereas 1 and 4 different ones. This network showed the highest instability. It might be due to the 23201 parameters that need to be estimated for the weights. Another is that the selection of the hyper-parameters is suboptimal.

Figure 5.24: Multiple runs of the feed-forward NN with 2 hidden layers, activation function tanh for input - hidden 1 and hidden 1 - hidden 2 layers: Failure-free survival. Top panel: proportion for 5 performance metrics, Bottom panel: Time dependent Brier score.



5.3.3 Discussion

Models-relative importance-stability issues

We applied Biganzoli's PLANN method and an extension using tanh and ReLU activation functions for input-hidden layer. A two hidden layer network with identical node sizes was fit. Tuning the hyper-parameters we observed that for overall survival activation functions sigmoid and ReLU produced the best models. ReLU had consistently the best combinations of minimized cross entropy for accuracy and sigmoid lower IBS. After performing 2 sets of experiments we selected combination (activation function ReLU, node size 90, dropout rate 0.1, learning rate 0.2, momentum 0.9

and weak class weight 1). There were indications of instability in the mean survival probabilities, Brier score. Relative importance using Garson connection weight method (Garson, 1991) showed that 2 variables *immune-suppression* (0.136) and *time interval* (0.08) were consistently the most contributing. Third most important variable was *length of stay* with an instability regarding variables 4 and 5 between *life support*, *Latin ethnicity*, *incidental tumor* and *symptomatic Cerebrovascular disease*.

A neural network with 2 hidden layers showed that sigmoid activation function collapses in terms of sensitivity and f1score to zero thus producing a network that predicts only alive patients. This is caused by the fact that sigmoid activation saturates at either 0 or 1 tail with the gradient at those regions being almost zero (Isaac Changhau). On the contrary, *ReLU* and *tanh* are by nature more flexible avoiding this pitfall. Combination chosen was (activation ReLU, node size 100, dropout rate 0.1, learning rate 0.2, momentum 0.9 and weak class weight 1) for OS and (activation function tanh, node size 100, dropout 0.1, learning rate 0.2, momentum 0.9 and weak class 1) for FFS. Instability was present (in performance metrics and Brier score) especially for the second network.

For FFS, NN with 1 hidden layer activation functions sigmoid and ReLU had the best performance in the majority of the metrics. Combination selected was (activation sigmoid, node size 75, dropout 0.1, learning rate 0.2, momentum 0.9, weak class weight 1). Network showed remarks of stability with relative importance graphs being identical for the top 5 variables in 4 repeats. *Immune-maintenance suppression* was by far the most important (0.191) having 5% increase in relative importance compared to OS followed by *time interval* (0.107) receiving again increased importance by 3%, *length of stay* (0.038), *donor type* (donation after circulatory death) 0.034 and *re-transplantation* (0.029).

Metrics investigation

Tables containing the 2 best combinations of different functions with respect to cross entropy show that accuracy and IBS are competitive. Larger values in accuracy come at a price of a larger value in IBS and the other way around. Same holds for sensitivity and specificity. Larger values for sensitivity result in smaller values of specificity. It would be desirable to find combinations that balance performance between those metrics.

The metrics defined in section 4.3 provide the following information:

- Specificity $\rightarrow 1$ means that true negatives D $\rightarrow 1$ and false positives B $\rightarrow 0$.
- Sensitivity $\rightarrow 0$ means that true positives A $\rightarrow 0$ and false negatives C $\rightarrow 1$. Such a network would predict that all patients stay alive being unsuitable.
- F1 score $\rightarrow 0$ means that sensitivity $\rightarrow 0$ or precision $\rightarrow 0$.

Each one of these combinations provides an unrealistic and possibly over-fitted network. For this reason, we avoided selecting combinations that lead to such metrics during tuning.

Considerations

A pitfall in the literature of neural networks for survival analysis is the absence of a well-established global performance measure. While the training of the network through cross-validation depends on the error loss function minimization, regularization techniques are often used to avoid reaching the global minimum as such

a network is possible to overfit. To establish a network with good performance we specified 6 performance metrics: accuracy, sensitivity, specificity, precision, f1 score and integrated Brier score. Different parameter combinations resulted in superiority of a different metric. This demonstrates the flexibility of NN as one can choose the metric of his choice to get the best network possible according to his preference or can combine all measures to get an overall selection. Validation of the performance is needed to have a representative and stable network. When we examined the networks selected via cross-validation tuning only the one for FFS with sigmoid activation function showed stability. Networks with 2 hidden layers showed large warnings of instability. It might be the case that the massive amount of training weights hinders the process.

For neural networks with 1 hidden layer, a warning sign extra to variability of performance metrics can be variation of variable contribution. This relative importance method is based on the connection weights (algorithm 2 in chapter 3). Therefore, variation in contribution means variation in weights causing instability. The construction of a global prediction measure that can be consistently stable for tuned parameters requires future research.

Two alternative solutions to get insights about relative importance of variables are schematic representation of the network and the sign of the weights and fitting many neural networks dropping variables that are suspected to be important. However,both are not straightforward for a network with 129 inputs.

The PLANN approach implemented here has been proposed by Biganzoli. Trying to be stay updated with the philosophy of neural networks from modern area we used the `keras` package that offers tremendous capabilities. Nevertheless, great capabilities bring great responsibilities as tuning number of parameters and grid search combinations are not limited. There will always be another combination not yet charted that can lead to a competitive network.

Over the last year, new literature regarding neural networks has been published in survival analysis field. For instance deep neural networks that employ Cox proportional hazards method to model interactions between patient covariates and treatment effectiveness to provide personalized recommendations (Fotso, 2018). It is of interest to compare different methods performance on medical data sets under varied proportions of numerical and categorical variables to identify cases where one method will be more suitable than another.

Chapter 6

Comparison between methods

6.1 Comparisons at exact time points

In previous chapter we discussed different statistical methods (SM) that can be used to estimate clinical outcomes for UNOS data. Data contain 106 variables and two different response pairs (time and status) one for overall survival (time until death) and another for failure-free survival (time to death or graft failure). A default choice is to use Cox model containing all prognostic factors. We considered two strategies for model selection: backward elimination and penalized Cox regression using LASSO (section 5.1). Two techniques from the area of machine learning (ML) - random survival forests and survival neural networks - (sections 5.2 and 5.3) were employed to estimate the quantities of interested.

For the neural networks transformation of the data had to be made for 12 distinct time intervals $(0, 1]$,..., $(11, 12]$ years. For the other methods exact time points were used. To compare performance and predictive ability of all models some transformation of time points is required. This aspect will be discussed in section 6.2. In this section, we compare only methods with exact time points: Cox model with all variables, Cox backward model, Cox model selected with LASSO and the random survival forest model. Comparisons will be made separately for overall and failure-free survival.

Table 6.1: Example of a 2x2 confusion matrix for simple classification setting.

	Actual	
Predicted	Event	No Event
Event	A	B
No Event	C	D

The strategies we followed are presented and include global measures, prediction error curves in terms of Brier score and C-index as well as survival curves for new hypothetical patients.

A) **Global performance measures**:

1. 5 metrics from the confusion matrix namely: **accuracy**, **sensitivity**, **specificity**, **precision** and **f1 score**. We compare the event status of last observed survival times with the survival probability predicted at this time. In

case of a survival probability ≤ 0.5 the predicted event status is 1 and in the different scenario 0 (no-event).

2. **Integrated Brier score (IBS)**: a summary of the time-dependent Brier score. It can be regarded as a global measure.

3. **Concordance index (C-index)**: computes the proportion of concordant pairs over all possible combinations. For the Cox models, it is constructed based on the prognostic index. A pair of observations is concordant if subject with highest prognostic index has shorter survival time. For RSF, C-index is based on ensemble mortality (number of expected deaths given that all subjects had same values). A pair of observations is concordant if subject with highest ensemble mortality has shorter survival time.

These measures will be presented in a table.

B) Prediction error curves using the time-dependent Brier score and time-dependent concordance index curves. Brier score is a weighted average of the squared deviations between the observed survival status and the predicted survival probability. Brier score is calculated over time so prediction error curves can be obtained based on the measure. Concordance index is usually a global measure. However, the same index can also be calculated at different specified time points using inverse probability of censoring weighting (IPCW) to adjust for right censored observations. For both methods R package `pec` (Thomas Gerds, 2018) can be used to estimate time-dependent Brier score and time-dependent concordance index. The curves will be estimated under 4 different cases:

- To the training set to obtain the **apparent training Brier score and C-index**.
- To the test set to obtain the **generalized Brier score and C-index**.
- To the training set using cross-validation based on bootstrap re-sampling B = 100 times to obtain the **bootstrap cross-validated (BootCV) Brier score and C-index**.
- To the training set using cross-validation based on bootstrap 632 re-sampling B = 100 times to obtain the **bootstrap 632 cross-validated (Boot632CV) Brier score and C-index**.

It is interesting to investigate if splitting the data in training (2/3) and test set (1/3) is necessary to avoid over-fitting (details in section 4.4) or if techniques of modern area such as bootstrap combined with cross-validation can provide safe inferences using training data at once.

C) Comparison of **predicted survival curves for new synthetic patients**. In the previous parts, when we discussed different Cox models and machine learning techniques a number of prognostic factors showed (high) predictive ability. The strongest predictor for Cox models - NN and the 2^{nd} strongest for RSF (in terms of VIMP) was *immuno-maintenance medicine*. Furthermore, RSF indicated *length of stay* as the most prognostic. To visualize the importance of variables, the predicted survival probabilities for new hypothetical patients will be plotted. Two other interesting variables in clinical point of view are *age of the recipient* and *incidental*

tumor found at time of surgery. These variables have been identified as prognostic in all models.

For the case of continuous variables: *length of stay* and *accidental tumor* the 10%, 50% and 90% quantiles on the training set will be used to construct different scenarios of hypothetical patients. For the categorical variables *immuno-maintenance suppression* and *incidental tumor*, we will use the different levels yes or no. To construct the full synthetic data set we use the median values for each of the 23 continuous variables and the mode factor level for the 72 dichotomous and the 11 polytomous categorical variables.

The subsections of the comparison are structured in a similar way as before. Division is made for overall and failure-free survival.

6.1.1 Comparisons: Overall survival

Models compared:

- Cox model with all variables (106 prognostic factors).
- Cox model using backward elimination (28 variables selected).
- 5-fold cross-validated Cox regression with LASSO with tuned parameter $\lambda_{1se} = 0.008$ (39 variables selected).
- A 5-fold cross validated random survival forest with tuned parameters: split rule = "log-rank", nodesize = 50, mtry = 35, nsplit = 5 and ntree = 250. Stratified bootstrap sub-sampling of 5000 patients per tree was used.

Global performance measures

Table 6.2 shows the performance measures for all models. Model of random survival forest with parameters mtry=35, nodesize=50, nsplit=5, ntree=250 and log-rank splitting rule outperforms Cox models according to all metrics. Accuracy of RSF is 0.687, whereas from the Cox models the best performance in accuracy is by the model including all 106 prognostic factors. Same pattern follows for the rest of the metrics of the equivalent confusion matrix. Best sensitivity (0.256), specificity (0.710), precision (0.044) and f1 score (0.076) is achieved by the trained RSF. Precision and f1 score have not been used in survival analysis literature as a way of evaluation different models. Very poor performance is shown by all models. The low f1 score is mostly determined by the low precision as this measure is the balanced accuracy between precision and sensitivity (section 4.3). An interpretation of the confusion metrics can be given as follows: the quite low sensitivity value means that true positives (A) are in approximately 1/4 ratio compared to the false negatives (C). This means that the models tend to predict that patients will survive than die. Specificity values for all models are very close. Examining table 6.1 a value of 0.71 means that true negatives (D) are in 2/3 ratio compared to false positives (B). Thus, all models predict correct survival (no-event) in the majority of the cases. The low value in precision means that true positives (A) are way less than false positives (B). Again this translates to poor predictive ability of events.

From the other 2 measures that capture the overall deviation of survival probabilities from the true status (IBS measure) and the discriminative ability of

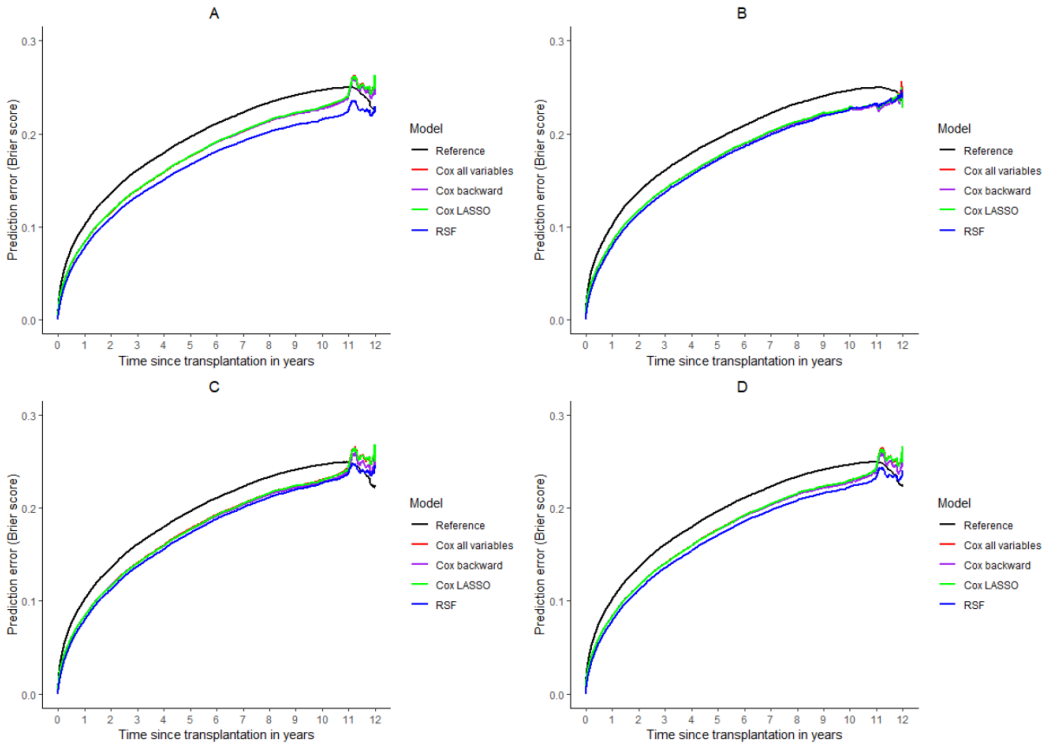
Table 6.2: Global performance measures for models at exact time points (OS).

	Accuracy	Sensitivity	Specificity	Precision	F1 score	IBS	C-index
Cox all variables	0.685	0.241	0.709	0.043	0.073	0.138	0.678
Cox backward	0.684	0.227	0.708	0.039	0.067	0.138	0.677
Cox LASSO	0.684	0.234	0.708	0.042	0.071	0.138	0.678
RSF	0.687	0.256	0.710	0.044	0.076	0.134	0.701

the models (C-index), the RSF is shown to have a consistently better prediction performance. C-index of the RSF model is 0.701 much higher than C-index of the Cox models. The forest is able to discriminate better patients with low and high risk. With respect to IBS, RSF has a reduced error but difference in performance is not so obvious.

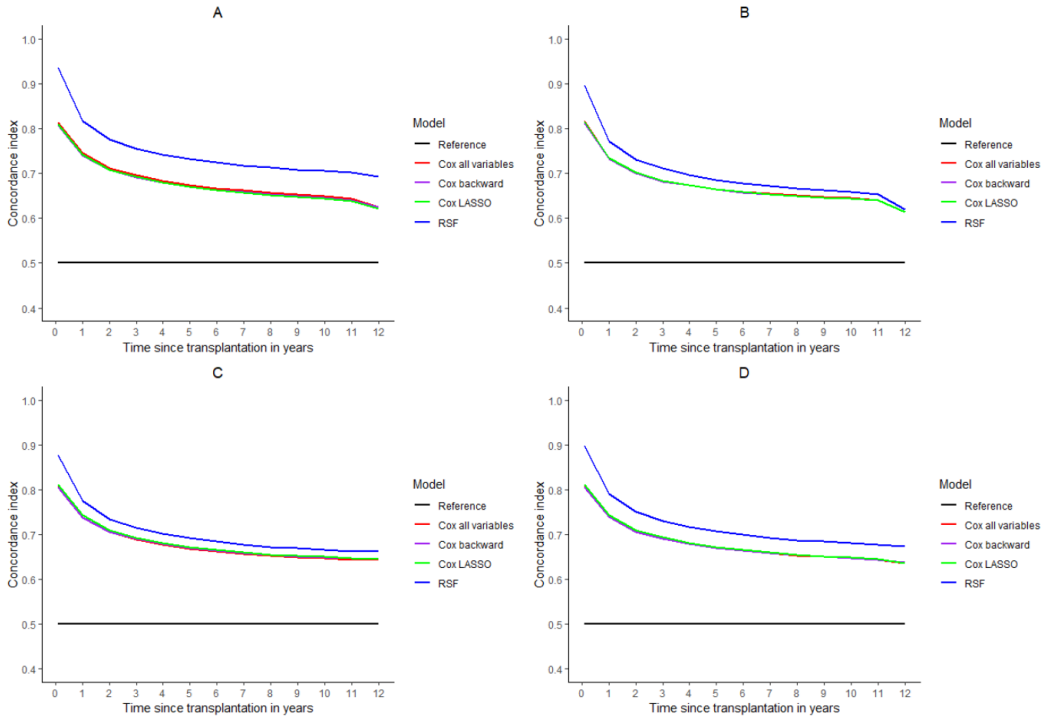
Prediction curves of error and C-index

Figure 6.1: Prediction error curves for all models using exact time points (OS). A: apparent error, B: generalization error (test error), C: OOB train error, D: linear combination of 0.328*(apparent error) + 0.632*(OOB train error). For graphs C and D bootstrapped cross-validation has been performed 100 times.



To further investigate results in table 6.2 we provide the time dependent prediction error curves (Brier score) and the time-dependent concordance for all models under 4 scenarios detailed earlier. Errors have an increasing trend over time. In figure 6.1 panel A, RSF (blue line) has lower prediction error than Cox models on training data. However, apparent error is usually avoided to evaluate the performance of models as it can be non-representative of the generalization error. Figures B (test error) and

Figure 6.2: Time-dependent C-index for all models using exact time points (OS). A: apparent error, B: generalization error (test error), C: OOB train error, D: linear combination of 0.328*(apparent error) + 0.632*(OOB train error). For graphs C and D bootstrapped cross-validation has been performed 100 times.



C (BootCV train error) show much smaller distances in prediction error between the models. Error in figure D (Boot632CV) is somewhere in between as it uses both apparent and BootCV train error weighted by 0.328 and 0.632 respectively. Note that all models have smaller prediction errors compared to the reference model.

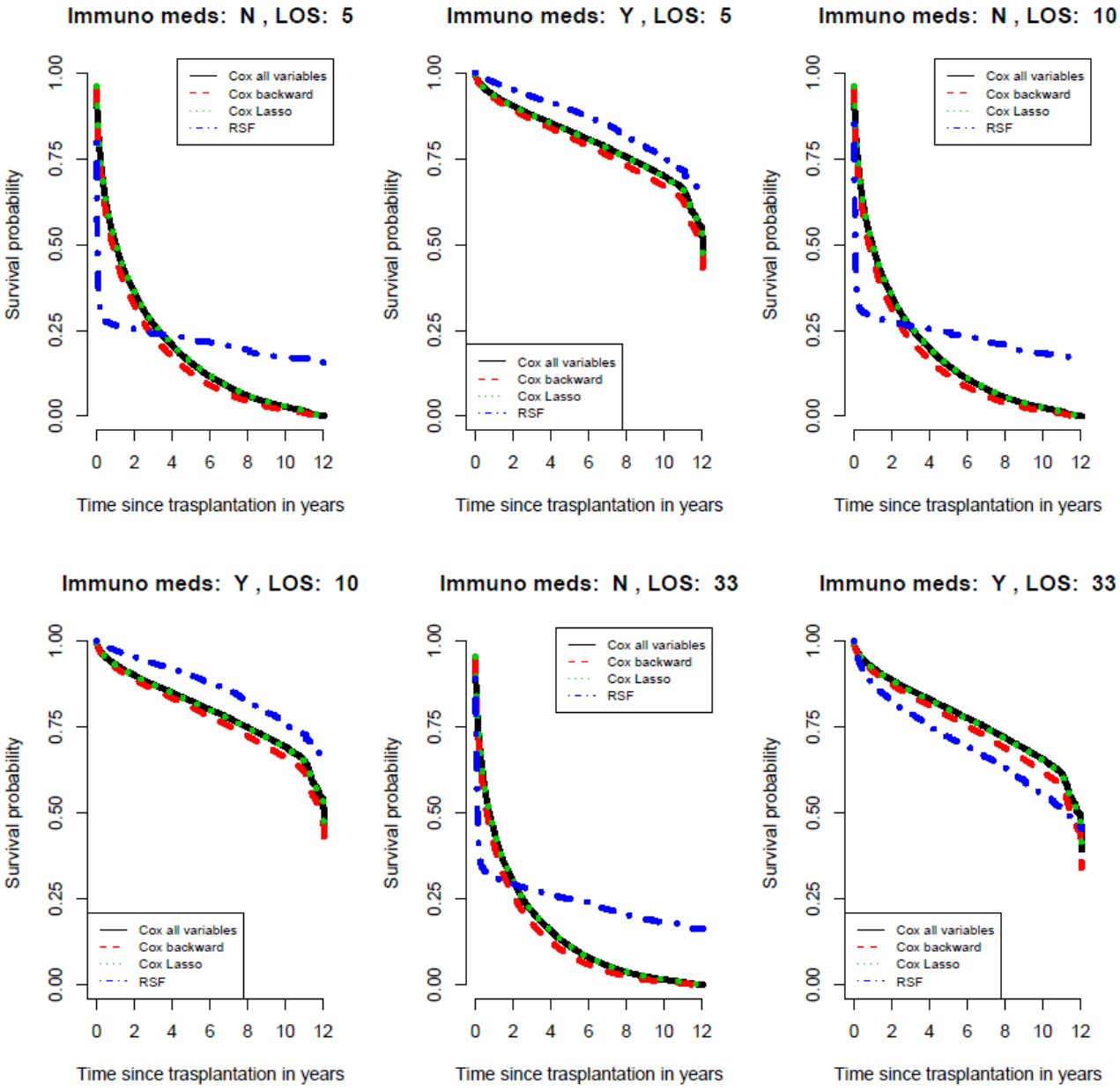
Figure 6.2 shows the concordance index over time. In all panels, concordance is the highest on the first year and decreases later on. RSF has an outstandingly higher performance on the training data (apparent error). This illustrates a well-known behavior of the forests to overfit training data. A way smaller difference is observed on the test set in figure B. Figures C and D refer to BootCV and Boot632CV C-index. C closely mimics B whereas in D blue line is in between. RSF show a stronger discriminatory ability. Hardly any difference is observed between the Cox models.

The indiscernible differences between B, C show that cross-validation based on bootstrap with re-sampling is a useful method to assess and compare the predictive power of various modeling strategies on the same data preventing at over-fitting. Bootstrap 632 can also be a good alternative. However, it is influenced by the apparent error (weighted by 0.328) and can be compromised in cases when apparent error differs significantly from test error. Bootstrap based cross-validation allows data to be used for both training and testing performance, an attractive alternative to traditional methods used to split the data into training set (2/3) and validation set (1/3).

Prediction curves for synthetic patients

On plot 6.3 Cox models follow a smooth strict decreasing behavior whereas RSF exhibits extreme decrease in survival probability for *immuno meds: No*. The shape

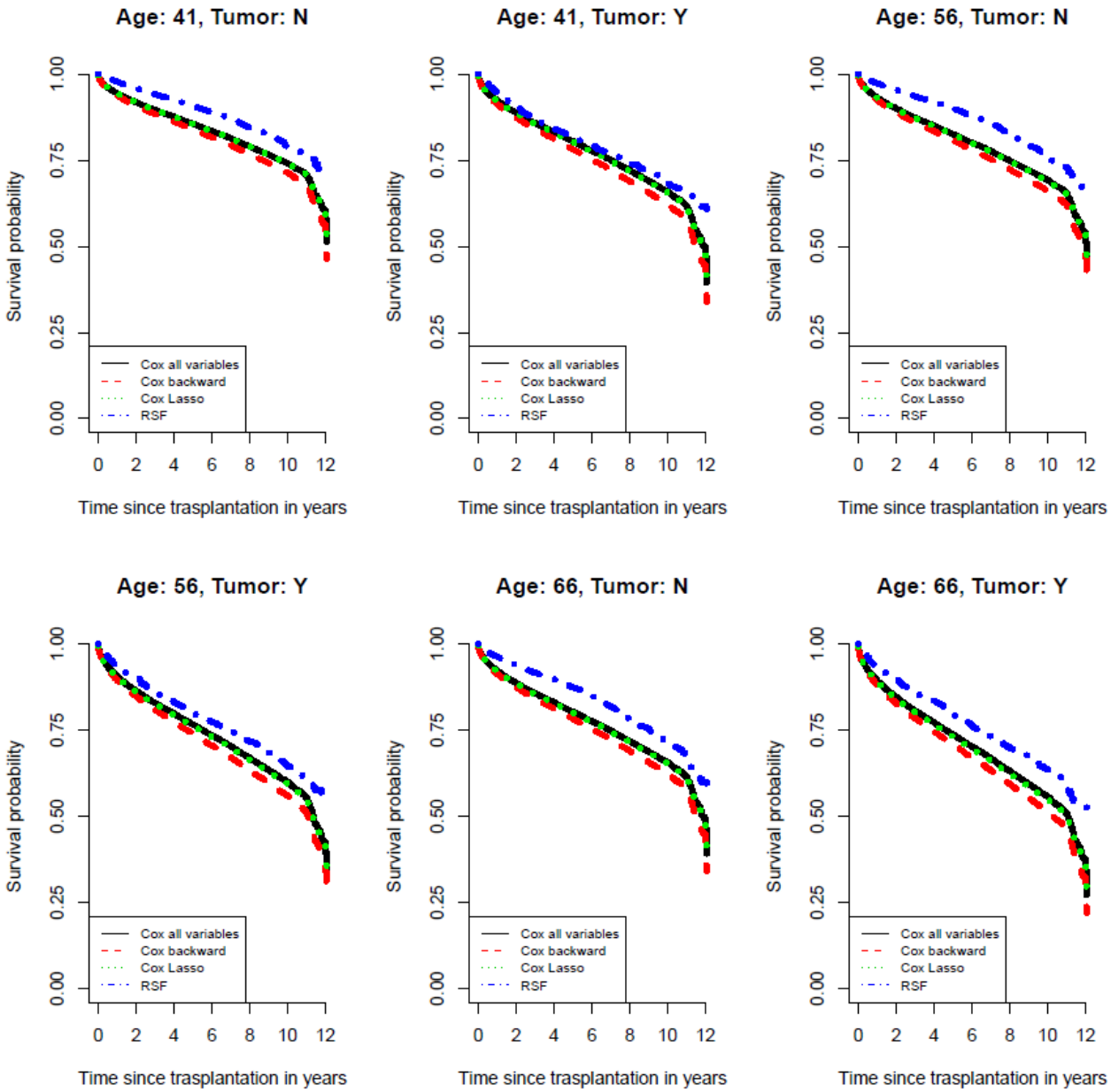
Figure 6.3: Overall survival for 6 new hypothetical patients for variables immuno-maintenance suppression and length of stay. Applied values for immuno meds are No (N) and Yes (Y). For length of stay values 5, 10 and 33 correspond to the 10%, 50% and 90% quantiles on the training data.



of the survival curve estimated by the RSF method may be related with the binary splitting process in the algorithm that allows for stepwise non-linear predictions. We want to emphasize that prognostic factor *rec_immuno_maint_meds* is very unbalanced and out of 41529 cases on the training set, only 1250 patients did not receive the immune-suppressants. Consequently, a very different behavior between those 2 groups can be possible.

Groups of patients that did not intake *immuno-maintenance medicine* have very low survival probabilities since the beginning of the post-operative period. For RSF probability of survival reduces dramatically to 0.3 since the first few months after liver

Figure 6.4: Overall survival curves of 6 new hypothetical patients for variables recipient age and incidental tumor. Values for tumor are No (N) or Yes (Y). For recipient age values 41, 56 and 66 years correspond to the 10%, 50% and 90% quantiles on the training data.



transplantation. A longer *length of stay* increases marginally the survival chance. For the Cox models, survival probability is a bit less than 0.5 as well after the second year of transplantation. On the other hand, patients that took the medicine have a much increased probability of survival. Examining the effect of variable *length of stay* for 5 and 10 days of hospitalization (given that the patient received the medicine) RSF predicts a higher survival probability than Cox models. This pattern is reversed in figure 6.3 right panel bottom line where RSF predicts lower survival probabilities. This fact is related with the importance of variable *length of stay* for RSF. Using VIMP and minimal depth *length of stay* was one of strongest predictors. This suggests that RSF

might do conservative survival predictions when variables are very prognostic and less conservative predictions otherwise. Last the Cox model using backward selection (red lines) predicted lower survival in all scenarios.

In figure 6.4 survival probabilities remain above 0.5 for almost all follow-up time. *Recipient age* and *incidental tumor* have reduced impact compared to *length of stay - immuno meds*. Cox model with all variables and the Cox model with LASSO give almost identical predictions. Lower survival predictions are made by the Cox backward model and higher with the random survival forest. Lines are in accordance with the previous graph. For UNOS data, in case a variable has strong prognostic value Cox models can predict in a stricter and smoother manner than RSF. Random survival forests can emphasize the importance of a variable providing extreme non-linear predictions. For less important variables, RSF produce higher survival predictions than Cox.

Note that an obvious limitation to such visual inspections is predicting probabilities for variables that are not selected by the backward elimination or the LASSO method.

6.1.2 Comparisons: Failure-free survival

Models compared:

- Cox model with all variables (106 prognostic factors).
- Cox model using backward elimination (32 variables selected).
- 5-fold cross-validated Cox regression with LASSO with tuned parameter $\lambda_{1se} = 0.0106$ (32 variables selected).
- A 5-fold cross validated random survival forest with tuned parameters: split rule = "log-rank", nodesize = 70, mtry = 23, nsplit = 6 and ntree = 250. Stratified bootstrap sub-sampling of 5000 patients per tree was used.

Global performance measures

Table 6.3 shows the performance of the 4 models for FFS. Results are similar with the RSF model outperforming Cox models for all measures. Notably there is a decrease in accuracy, specificity for all models and at the same time an increase in sensitivity, precision and f1 score. This is an indication that models for failure-free survival can identify slightly more events. Note that an event here is the first between graft-failure and death. Interestingly, Cox backward - LASSO model have the highest accuracy. RSF performs better than Cox models in sensitivity (0.313) and accuracy (0.666).

Examining the more specialized global measures of survival analysis IBS and C-index results are also very similar. RSF performs much better in terms of C-index (0.699) when compared to Cox models (C-index 0.673-0.674). For the integrated Brier score, RSF has the smallest error of 0.142. Both measures have a slight decrease compared to OS (see table 6.2). This is an indication that FFS has an added complexity. Metrics support that RSF model with tuned parameters mtry=23, nodesize=70, nsplit=6, ntree=250 with the log-rank splitting rule is a better discriminative model. Precision and consequently f1 score attain poor performance but slightly boosted compared to overall-survival.

Table 6.3: Global performance measures for models at exact time points (FFS).

	Accuracy	Sensitivity	Specificity	Precision	F1 score	IBS	C-index
Cox all variables	0.659	0.278	0.685	0.056	0.094	0.149	0.674
Cox backward	0.660	0.270	0.685	0.052	0.087	0.149	0.673
Cox LASSO	0.660	0.271	0.685	0.053	0.088	0.149	0.673
RSF	0.666	0.313	0.688	0.057	0.097	0.142	0.699

Prediction curves of error and C-index

The equivalent error curves (Brier score) and time-dependent C-index curves of the models for failure-free survival are provided in appendix (graphs B.1 and B.2). The errors are slightly increased and the C-indexes decreased. RSF performs slightly better than the Cox models in terms of Brier score and considerably better in terms of C-index. C-index of RSF on training set (panel A) is much higher than the C-index on test set (panel B) or that on the bootstrapped cross-validated training set (C and D). Errors and C-indexes in graphs B and C are very similar.

Prediction curves for synthetic patients

Graphs B.3 and B.4 in appendix B illustrate failure-free survival for new hypothetical patients.

6.1.3 Discussion

In this part we compared 3 Cox models: 1) the model with all variables, 2) the model selected from backward elimination, 3) the selected from LASSO and a machine learning model with a tuned random survival forest. The 3 Cox models exhibited similar performances for OS and FFS. Between them, the one with all variables was slightly better in terms of C-index but not in terms of IBS. However, under comparison with the RSF trained model, Cox models were inferior in all quantified metrics. Largest difference was observed in C-index (discriminative ability of the models).

We examined the prediction error curves and time-dependent C-indexes for all models under 4 cases: 1) prediction on training set without adjustment, 2) prediction on the test set, 3) prediction on training set with bootstrap cross-validation repeated B = 100 times and 4) prediction the training set with bootstrap 632 cross-validation repeated B = 100 times. Plots of error and C-index versus time showed superiority of RSF. In apparent error, RSF showed an even higher superiority that is not realistic as RSF can naturally adjust very good on the training data because they were tuned using them. A generalization error can be found on the test set that contains "unseen" observations or a good approximation can be found on the training set combining prediction with bootstrapped based cross-validation. The repetition of the process B = 100 times was used to reduce randomness and increase accuracy of findings.

RSF seem to be an attractive alternative to the Cox models in terms of prediction as it can distinguish more efficiently between low and high risk patients. However, we need to consider as well that considerate effort is needed to implement a forest with highly tuned hyper-parameters. The large sample size of UNOS data and the 106 predictors slow down the process dramatically and a considerate amount of time

is required for 5-fold cross-validation. Earlier we discussed a fast alternative to that - the OOB train error which however is not proven to be an unbiased procedure.

Survival predictions of new hypothetical patients for variables present in all models showed differences. RSF gave a higher survival probability than Cox models in most cases but lower for a very prognostic factor level (*immuno meds: No*) or large quantile (90%) of the very prognostic variable *length of stay*. On the other hand, all Cox models gave similar predictions, with the Cox backward elimination model consistently producing lower probabilities.

6.2 Comparisons at discrete time points - intervals

In this section, comparison of 6 models will be discussed. These are: a) Cox model with all variables, b) Cox model with backward elimination and c) Cox model with variables selected with LASSO, d) random survival forest, e) neural network with one with one hidden layer and f) neural network with 2 hidden layers. In section 5.3 we discussed the implementation of neural networks using the partial logistic method. Training and test data were transformed in a long format. For the training data, each patient was repeated according to the number of years he/she was present in the study while for the test data every patient was repeated for 12 time intervals since the survival time was supposed to be unknown.

The necessary transformation of the data to this format requires special attention to perform comparison between NN method and the other methods. One could choose to compare all models using the time intervals for the NNs and exact time points for Cox and RSF. However, this strategy will be avoided as it introduces bias. To mitigate bias, a simple solution is to change the exact survival times with 12 distinct times 1, 2, ..., 12 denoting years since transplantation. It is of interest to investigate how the models perform compared to section 6.1.1. Exact survival times include more information than discrete. It is anticipated that the performance of well established survival measures such as C-index and Brier score will decrease. Confusion matrix is based on the observed survival times and as the discrete time points are 12 while the exact time points were more than 4000 on the training set, the performance measures based on the confusion matrix are likely to produce higher values.

It is important to stress that changing the time points means changing partially the response variable. As a consequence, both Cox models and RSF models constructed before need to be re-fit. For instance the backward elimination method may select a different number of variables. Methods dependent on hyper-parameters such as selection with LASSO and RSF require re-tuning. To follow a cohesive strategy, 5-fold cross-validation will be repeated. Final tuned parameters are reported separately for OS and FFS.

Comparison will be based on global performance measures, the time-dependent Brier score, the mean survival curves estimated for all patients, predicted survival curves applied to the same synthetic patients as well as interpretability of models. A detailed description is given below:

A) Global measures of performance considered are

1. the 5 confusion matrix measures: **accuracy**, **sensitivity**, **specificity**, **precision** and **f1 score**

2. the **integrated Brier score (IBS)**

3. the **C-index**.

B) Plots of the **time-dependent Brier score** and **mean survival probabilities** over time.

C) **Predicted survival curves for hypothetical patients**. Variables used are *immuno-maintenance medicine*, *length of stay*, *recipient age* and *incidental tumor*. For categorical factors the two levels No or Yes are considered whereas for the continuous variables their 10, 50 and 90% quantiles on train set. The rest of the covariates were kept constant to their median value if they are continuous and to their mode value if dichotomous or polytomous.

D) **Interpretability of the different methods**. Cox models provide interpretation through the hazard ratios. RSF has 2 methods to obtain interpretation VIMP and minimal depth whereas for neural networks the Garson's connection weight method was used. Note that the method was only applied to neural networks with one hidden layer. All hazard ratios produced per Cox model, and VIMP, minimal depth for the re-tuned RSF can be found in appendix B sections B.4.1 and B.4.2. Effects are quite similar with those of the exact time points. Differences in the importance of variables per method are discussed. Division is made for overall and failure-free survival.

6.2.1 Comparisons: Overall survival

Models compared are:

- Cox model with all variables (106 prognostic factors).
- Cox model using backward elimination (29 variables selected).
- 5-fold cross-validated Cox regression with LASSO with tuned parameter $\lambda_{1se} = 0.0082$ (37 variables selected).
- 5-fold cross validated random survival forest with tuned parameters: split rule = "log-rank", nodesize = 50, mtry = 23, nsplit = 7 and ntree = 300. Stratified bootstrap sub-sampling of 5000 patients per tree was used.
- 5-fold cross-validated neural network with 1 hidden layer and parameters chosen: activation function = "ReLU", node size = 90, dropout rate = 0.1, learning rate = 0.2, momentum = 0.9 and weak class weight = 1.
- 5-fold cross-validated neural network with 2 hidden layers and parameters chosen: activation = "ReLU", node size = 100, dropout rate = 0.1, learning rate = 0.2, momentum = 0.9 and weak class weight = 1.

Global performance measures

Table 6.4 shows the summary of global performance measures. RSF provides the best model in terms of accuracy (0.715) and sensitivity (0.531) and shows the highest C-index (0.694) and the second lowest IBS (0.161). Error and concordance are slightly increased and decreased respectively compared to exact time points (table 6.2). This suggests that in a survival analysis setting more accurate results can be obtained when the time variable contains the highest possible amount of information.

Neural network models are comparable with the Cox models in terms of accuracy but worse in sensitivity. Nevertheless, they give the best results in specificity, precision and f1 score. IBS which summarizes Brier score is inferior for these models. Cox models are very close to each other for all examined metrics.

Table 6.4: Global performance measures for all models. Neural network 1h refers to a neural network with one hidden layer, neural network 2h to a neural network with 2 hidden layers (OS).

	Accuracy	Sensitivity	Specificity	Precision	F1 score	IBS	C-index
Cox all variables	0.702	0.427	0.721	0.096	0.156	0.166	0.683
Cox backward	0.697	0.343	0.714	0.054	0.093	0.159	0.683
Cox LASSO	0.702	0.423	0.721	0.092	0.151	0.166	0.683
RSF	0.715	0.531	0.728	0.115	0.189	0.161	0.694
Neural Network 1h	0.703	0.162	0.922	0.457	0.240	0.188	-
Neural Network 2h	0.697	0.181	0.906	0.440	0.257	0.182	-

To choose a model based on survival specialized measures leads to RSF. From the metrics of the confusion matrix, neural networks dominate in precision and specificity. This can be interpreted as follows: inspecting the confusion matrix 6.1 precision 0.45 means an almost 1:1 ratio between true positives (A) and false positives (B). Therefore, the networks can identify an event correctly 1/2 times. A specificity of 90% means that 9/10 true negatives (D) and not false positives (B) were found. Networks can identify patients as alive in the majority of the cases correctly. Note however that both neural networks presented here showed indications of instability so they can be regarded as a risky choice. RSF and Cox models have a very stable performance so they can be regarded a safer choice.

Prediction error curves and mean survival probabilities

Figure 6.5 shows that Cox models and RSF have almost identical error lines, with the Cox backward (red line) slightly better at the beginning. Neural networks have also similar trajectories. For the first 9 years the have higher prediction error where later they perform slightly better with the network with 2 hidden layers better. Looking at mean estimated survival probabilities, Cox backward has the highest mean value and the neural network with 2 hidden the lowest with differences being minimal.

Prediction curves for synthetic patients

In graph 6.6 predictions made by Cox models and the machine learning techniques are very different for *immuno meds level No*. Cox models produce smooth decreasing lines over time, whereas machine leaning models give an extreme decrease in survival probability from 1st year. For greater quantiles in *length of stay* (10, 33 days), survival probabilities of RF , NN slightly increase. For 5 days, RSF predicts survival probabilities of around 0.3 and NN models around 0.23 with probabilities almost stabilizing later on. As *length of stay* increases, NN give increased survival probabilities and RSF marginally increased. From the Cox models, backward elimination (red lines) gives by far the highest survival probabilities. Looking at sub-panels where *immuno meds: Yes*, predicted lines are much more clustered with RSF giving higher survival

Figure 6.5: Time-dependent Brier score (top panel) and mean survival probability over time (bottom panel) for all models (OS).

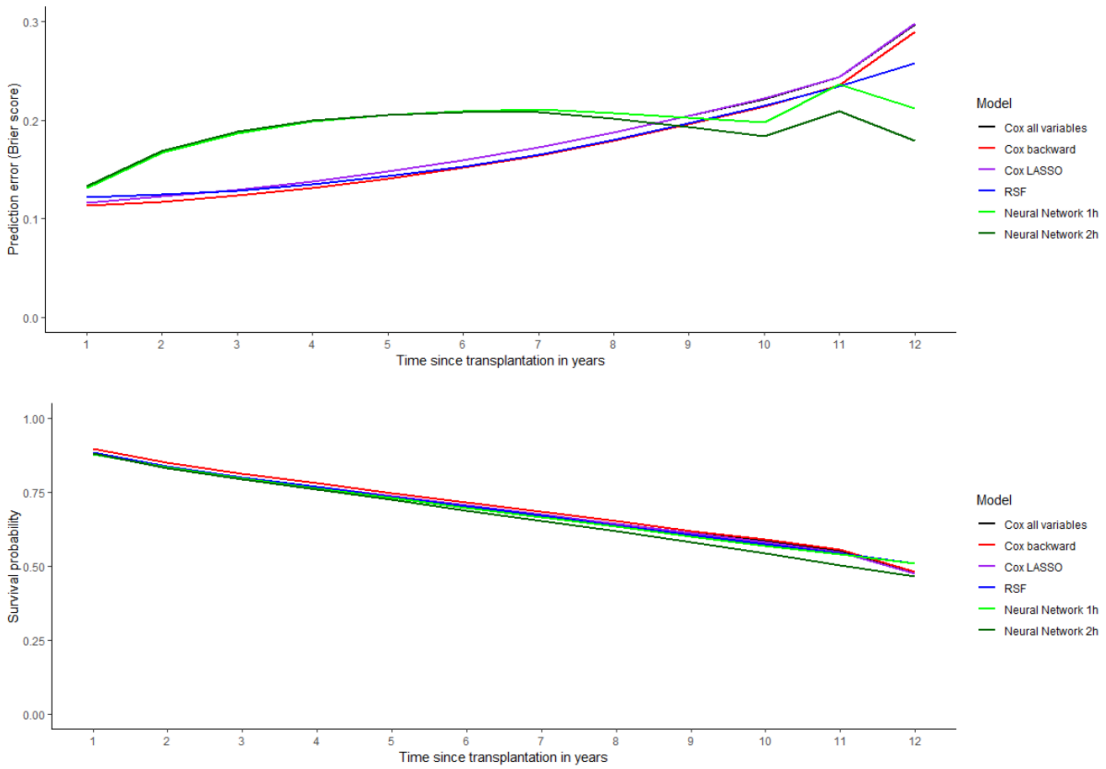
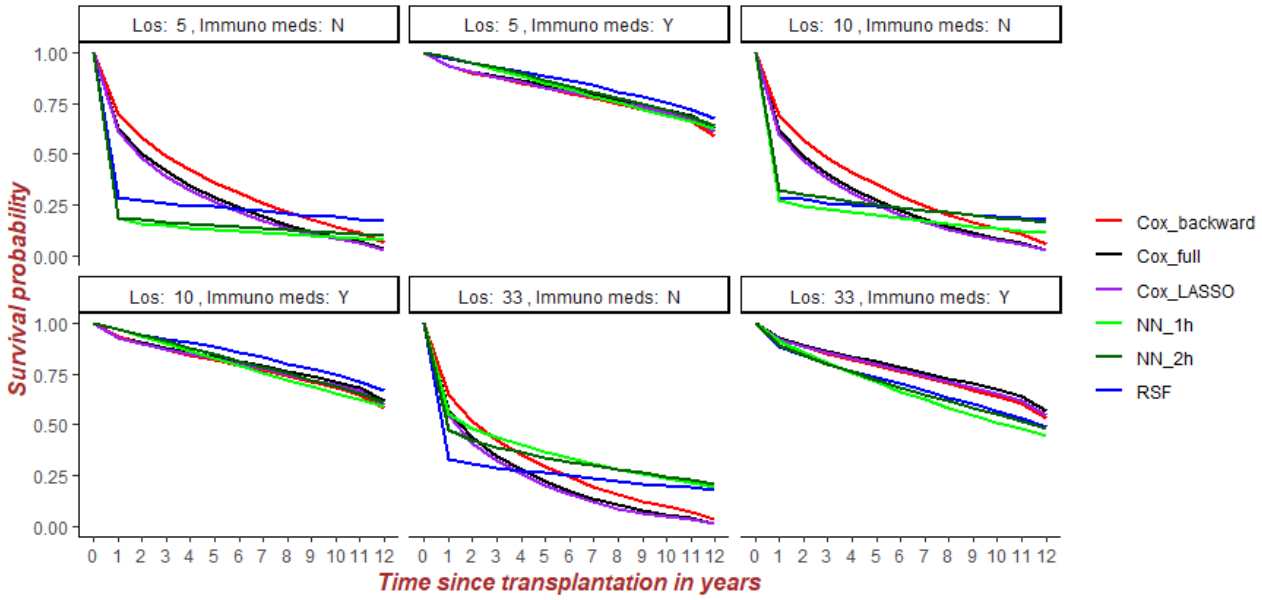
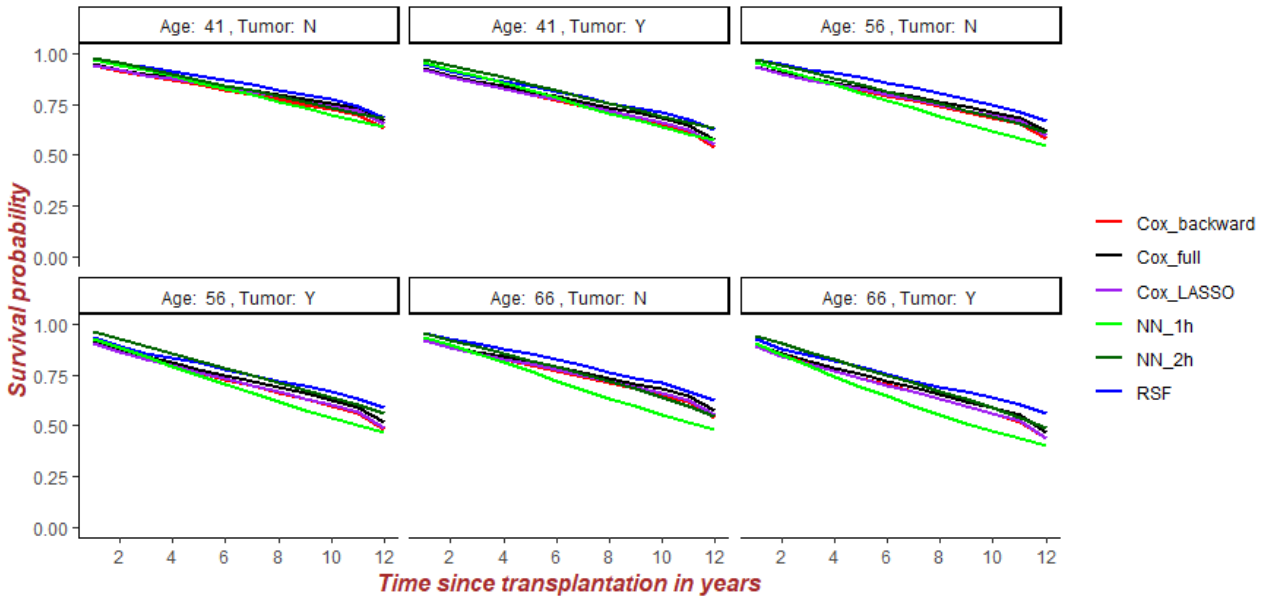


Figure 6.6: Overall survival for 6 new hypothetical patients and variables immuno-maintenance suppression and length of stay. Applied values for immuno meds are No (N) or Yes (Y). For length of stay applied values 5, 10 and 33 correspond to the 10%, 50% and 90% quantiles on the training data.



predictions for *length of stay* 5, 10 days. For 33 days, RF and NN give by far more

Figure 6.7: Overall survival for 6 new hypothetical patients and variables recipient age and incidental tumor. Applied values for tumor are No (N) or Yes (Y). For recipient age applied values 41, 56 and 66 years correspond to the 10%, 50% and 90% quantiles on the training data.



conservative survival predictions.

In this plot, the nature of the methods is revealed. Cox models make proportional hazards assumptions and consequently this leads to smooth strictly decreasing survival lines. On the other hand, RF and NN do not make any assumptions for data structure. As a result, they can give extreme survival predictions when a variable's level is very prognostic - such as here *immuno meds: No*. A difference with exact time points is that Cox backward model consistently gave the highest survival predictions where in figure 6.3 provided the lowest.

Looking at figure 6.7 all models produce very similar survival predictions for the 6 different combinations of recipient's age and incidental tumor. Most usually, RSF give the highest survival probabilities and NN with 1 hidden layer the lowest. This is related with the fact that *rec_tumorY* is one of the 5 most prognostic variables for the neural network with 1 hidden layer. Of course, the two different neural networks can give survival predictions that differ.

Overall, the loss of survival information with the discretization of more than 4000 exact survival time points to 12 discrete time points did not affect the Cox models and the RSF to a large extent. Some larger Brier scores and lower C-indexes were observed, but predictions in terms of survival curves for new synthetic patients were pretty similar. The model based on backward elimination produced the lowest Brier score and gave higher survival probabilities than the other Cox models.

Interpretability of the methods

Table 6.5 shows the levels of the variables that have the highest increase or decrease in hazard ratios compared to their reference level (*can_race_srtr* has reference level White). Variables are also present in the Cox backward and the Cox LASSO models.

For the Cox LASSO models their hazard ratios are shrunken because LASSO not only selects variables by setting the coefficients of unimportant variables to zero but also shrinks the effects of variables selected.

The most influential variables are *immuno-maintenance medicine* and *retransplantation*. Being in the group that received the medicine decreases hazard of death by factor 0.14, 0.191 and 0.201 for Cox model with all variables, Cox backward and Cox LASSO respectively. Having been re-transplanted before increases hazard by factor 1.517, 1.436 or 1.357 for the 3 models respectively. Note that both *candidate's race* Black and Other have a large difference compared to reference level White.

Table 6.5: Hazard ratios of the 10 most influential variable levels for the Cox models at discrete time points (OS).

	Cox all variables	Cox backward	Cox LASSO
retransplantationRetransplantation	1.517	1.436	1.357
rec_tumorY	1.375	1.351	1.176
can_angina_cadYes	1.347	1.357	1.179
lifesupportLifesupport	1.226	1.224	1.150
diabY	1.232	1.227	1.163
can_race_srtrBlack	1.203	1.195	1.131
can_race_srtrOther	0.802	0.804	0.951
rec_work_incomeY	0.813	0.796	0.850
can_acpt_extracorp_liY	0.774	0.815	0.976
rec_immuno_maint_medsY	0.140	0.191	0.201

When examining the most predictive variables for RSF in terms of VIMP and minimal depth (shown in tables B.17 and B.18), *length of stay* comes first for VIMP (0.052) and second for minimal depth (1.77). *Immuno-maintenance medicine* follows with 0.028 in VIMP and is the most predictive for minimal depth (1.543). What's more, other contributing variables are *donor's and recipient's age* but with much smaller VIMP and much higher minimal depth values. Out of the 20 most predictive variables 6 are continuous in VIMP and 14 in minimal depth showing clearly that RSF tend to favor continuous variables during splitting (Loh and Shih, 1997). *nsplit* parameter was used as a way to mitigate bias by forcing random selection of exactly *nsplit* points per candidate variable. However, it wasn't enough to deal satisfactory with the bias inserted by the log-rank split rule. This shows that when variable selection is biased then both VIMP and especially minimal depth that is calculated based on the topology of the tree will be biased.

Length of stay and *immuno meds* seem to have a strong predictive ability. *Immuno meds* were by far the most prognostic factor for the Cox models but here *length of stay* is also of the same if not larger importance according to RSF. Of note is that *length of stay* is one of the selected variables for both backward elimination and penalized regression which shows that the variable can be very prognostic.

The 20 variables with the highest relative importance for the neural network with one hidden layer are shown in table 5.11. *Immuno-maintenance meds* (0.136) are the most predictive at a large margin from the other variables. *Length of stay* (0.038) is the third most prognostic after time interval with *retransplantation* (0.034) and *was donor hla typed* (0.030), *functional status* (0.029), *life support* (0.025) and *incidental tumor* (0.023) at a close margin. As a comparison with RSF, only two continuous

variables are part of the list with the 20 most prognostic. *Extra-corporeal liver:Yes* is one of the 10 most influential variables. Note that is variable was also shown to be important in all Cox models but not influential by RSF.

To sum up, variables *immuno meds*, *retransplantation*, *life-support* and *incidental tumor* were shown as (very) prognostic in all models. RSF identified *length of stay* as very prognostic and neural networks as the third most prognostic. *Donor age* and *recipient's age* were of also in the 5 most predictive variables according to RSF.

6.2.2 Comparisons: Failure-free survival

Models compared are:

- Cox model with all variables (106 prognostic factors).
- Cox model using backward elimination (33 variables selected).
- 5-fold cross-validated Cox regression with LASSO with tuned parameter $\lambda_{1se} = 0.0086$ (36 variables selected).
- 5-fold cross validated random survival forest with tuned parameters: split rule = "log-rank", nodesize = 50, mtry = 41, nsplit = 5 and ntree = 300. Stratified bootstrap sub-sampling of 5000 patients per tree was used.
- 5-fold cross-validated neural network with 1 hidden layer and parameters chosen: activation function = "sigmoid", node size = 75, dropout rate = 0.1, learning rate = 0.2, momentum = 0.9 and weak class weight = 1.
- 5-fold cross-validated neural network with 2 hidden layers and parameters: activation = "tanh", node size = 100, dropout rate = 0.1, learning rate = 0.2, momentum = 0.9 and weak class weight = 1.

Global performance measures

Brier score for Cox models and RSF is increased and C-index is decreased. Confusion metrics are slightly increased for all models. These findings are in accordance with OS.

Results are presented in table 6.6. Differences do exist compared to table 6.3 of exact time points for FFS. Best model in accuracy (0.687), sensitivity (0.493) and C-index (0.692) is by the RSF. Backward elimination model achieved again the best IBS (0.176) with RSF having the second lowest (0.182). As previously, Cox models have a similar performance in the majority of the metrics with Cox LASSO being the best in accuracy, sensitivity and specificity.

NN models have again the lowest sensitivity but at the same time the highest specificity, precision and f1 score. Their performance is comparable with the other models in terms of accuracy and IBS. This differs with OS where neural network models had a much worse integrated Brier score compared to the rest. Neural network with 2 hidden layers is better between the 2 in all measures but IBS. However, in the application chapter it was shown that NN with 1 hidden layer and activation function sigmoid in both input-hidden and hidden-output layers (original PLANN approach) was stable whereas the neural network with 2 hidden layers and activation function of

Table 6.6: Global performance measures for all models. Neural network 1h refers to a neural network with one hidden layer, neural network 2h to a neural network with 2 hidden layers (FFS).

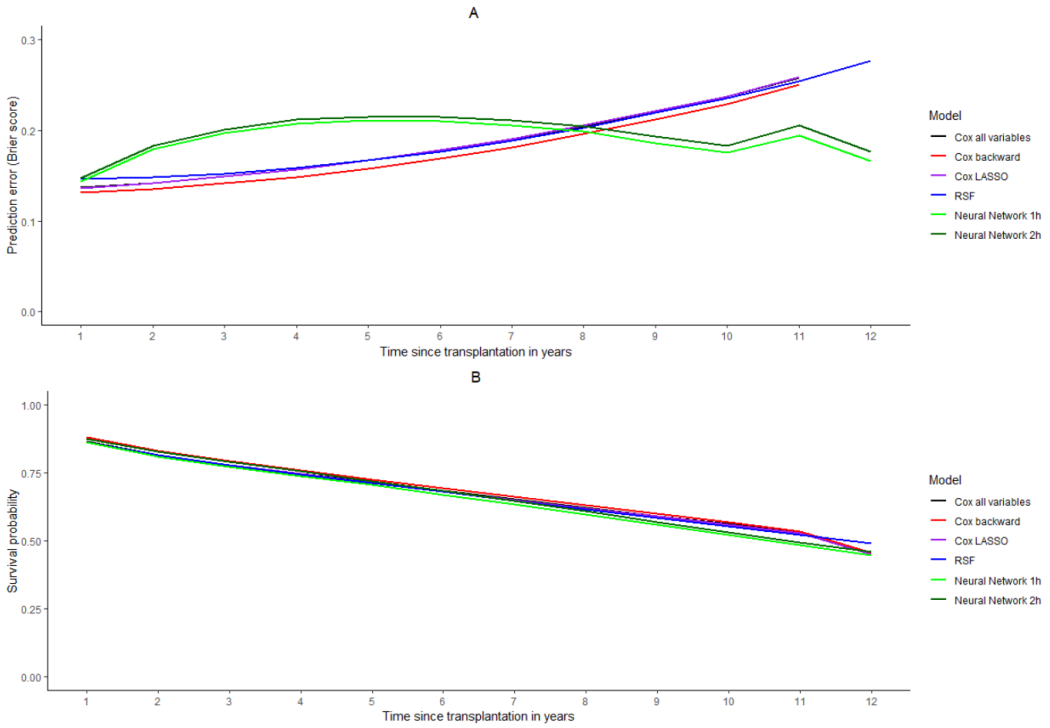
	Accuracy	Sensitivity	Specificity	Precision	F1 score	IBS	C-index
Cox all variables	0.679	0.450	0.699	0.117	0.183	0.183	0.679
Cox backward	0.674	0.381	0.692	0.068	0.116	0.176	0.679
Cox LASSO	0.681	0.458	0.700	0.114	0.183	0.184	0.679
RSF	0.687	0.493	0.704	0.128	0.204	0.182	0.692
Neural Network 1h	0.673	0.160	0.906	0.435	0.234	0.183	-
Neural Network 2h	0.680	0.171	0.911	0.467	0.250	0.188	-

input - hidden layer tanh was quite unstable. Therefore, the one with 1 hidden layer should be preferred.

RSF outperformed the other methods in concordance and gave similar if not better results in most metrics. RSF demonstrate remarkable stability in error after a particular number of trees (look chapter 5, figure 5.8) it can be motivated as a safe choice. However, tuning of such a model needs a considerate amount of time to get a highly performing combination, and interpretability can be challenging and favoring continuous prognostic factors.

Prediction error curves and mean survival probabilities

Figure 6.8: Time-dependent Brier score (top panel) and mean survival probability over time (bottom panel) for all models (FFS).



Plots in 6.8 show the Brier score and mean survival probabilities of all models. In panel A, small differences in error can be observed between Cox and RSF models

with Cox backward (red line) achieving the lowest error. Both backward and LASSO models are truncated at 11 years. On the other hand, NN models have almost identical evolution over time and reach better performance after 9 years. Looking at the distribution of event times (graph 5.17) most of them occurred during the first 2 years, so prediction after 8 years is not of great value.

Mean survival lines of all methods are almost identical with the neural network with one hidden layer producing marginally lower survival probability after 9 years.

Prediction curves for synthetic patients

Curves are presented in appendix B (figures B.5 and B.6).

Interpretability of the methods

Variables with the highest increase or decrease in hazard ratios for the Cox models are shown in 6.7. There is one variable *can_acpt_extracorp_liY* that was not selected by the LASSO model. The rest of these variables were selected in both Cox backward and Cox LASSO.

Similarly with OS, the most influential variables are *immuno-maintenance medicine:Yes* and *retransplantation*. Being in the group that received the meds decreases hazard by factors 0.126, 0.187 and 0.198 respectively in the 3 models. Patients that have been re-transplanted before have an increased hazard of 46.2, 40.1 and 31.6% given that the other variables are kept constant. Interestingly, *life-support* and *diabetes* have decreased hazard ratios for failure-free survival. They have been replaced in the table by variables *donor type level DCD* (donor after circulatory death) and *accept an HCV antibody positive donor?:Yes* with both increasing hazard of graft-failure or mortality.

Table 6.7: Hazard ratios of the 10 most influential variable levels for the Cox models at discrete time points (FFS).

	Cox all variables	Cox backward	Cox LASSO
retransplantationRetransplantation	1.462	1.401	1.316
donortypeDCD	1.393	1.240	1.187
rec_tumorY	1.332	1.307	1.132
can_angina_cadYes	1.311	1.306	1.139
can_acpt_hcv_posY	1.213	1.178	1.157
can_race_srtrBlack	1.207	1.205	1.137
can_race_srtrOther	0.824	0.832	0.964
rec_work_incomeY	0.857	0.852	0.890
can_acpt_extracorp_liY	0.786	0.847	-
rec_immuno_maint_medsY	0.126	0.187	0.198

Examining tables B.22 and B.23 in appendix B, the most prognostic variables at VIMP are *length of stay* (0.065) and *immuno meds* (0.027) with *donor age* (0.008), *accept an hcv antibody positive donor* (0.006) and *hcv serology status* (0.005) the rest of the top 5 while the other predictors have very small VIMP. Out of the 20 variables presented, 5 are numerical. In terms of minimal depth *immuno meds* (0.917) and *length of stay* (1.39) are the shallower with *donor age* (3.79), *recipient age* (6.307) and *accept an hcv positive antibody donor* (7.243) the others of the top 5. From the

20 overall, 14 are numerical. This is again a warning sign that continuous variables have been significantly favored during binary partitioning of the trees. Predictors *retransplantation*, *candidate's race*, *donor type* and *life support* are prognostic.

Relative importance of the neural network is shown in table 5.14. *Immuno meds* are the most important predictor (0.191) with *time interval* the second most important (0.107). *Length of stay* (0.038), *donor type:DCD* (0.034) and *retransplantation* (0.029) follow. Compared to OS, *accept an hcv antibody positive donor* (0.013) has an increased importance.

Summarizing, all models pointed to an increased importance of variable *can_acpt_hcv_posY* with *donor type:DCD* also more contributing. Cox models had decreased hazard ratios for *life-support* and *diabetes*. RSF had an increased VIMP for *length of stay* and for *hcv serology status* and NN an increased relative importance for *immuno meds* and *time interval*.

6.2.3 Discussion

We compared 6 methods using a variety of metrics. 5 of them - accuracy, sensitivity, specificity, precision and f1 score - refer to the confusion matrix constructed when comparing the event status of last observed survival time for each patient with survival probability predicted at this time point. Other global metrics used were the C-index and the integrated Brier score (IBS). From time-dependent performance measures Brier score was utilized.

Before comparison artificial change of the survival times was made for Cox models and RSF. Times were discretized into 12 distinct years 1, 2,..., 12 to mimic the neural network's 12 time intervals. Results showed small information loss for survival measures. Brier score increased and concordance index decreased leading to slightly worse performance of models. At the same time the 5 metrics related to confusion matrix did marginally inflate because the 12 discrete time points are much less than the exact. These findings show a negative association between confusion metrics and IBS-concordance that was also observed during the neural network cross-validation where higher values in accuracy were related to higher values in Brier score thus increased error.

Changing the time in the outcome pair (time, status) required to repeat all the cross-validations from the beginning. As expected new hyper-parameters were found to lead to highly performing models for Cox regression with LASSO and the random survival forest. This demonstrates a well known issue in models trained with ML. Even for a slight change in the data, parameter tuning must be repeated at an exhaustive grid to find a highly performing combination on training set that will result in a promising generalization error on the test set.

When we examined all global measures for OS and FFS results were similar with the trained neural networks of FFS having a comparable performance with the Cox models or RSF in terms of IBS. The trained NN models for overall survival were much inferior in terms of IBS. RSF outperformed in accuracy, sensitivity especially in C-index and had the 2^{nd} highest IBS. Note that a C-index was not found for NN because a natural ordering of subjects was not feasible in the way the data were longitudinally transformed. From the rest of the measures NN outlasted in specificity and precision - f1 score. This means that such models can be useful when scientific interest is on predicting alive patients correctly or to predict (true events) / (false

events) with a higher ratio. For the NN no proper measure is established to evaluate the model for survival analysis and as a result we selected the parameters in a more loose way combining the performance of many measures and the minimized cross-entropy. This showed warning signs of network instability and therefore choosing a network with 1 or 2 hidden layers has a degree of uncertainty in performance. On the other hand RSF and Cox models showed remarkable stability and can be regarded as much safer options.

Examining the time-dependent performance of the models, Cox backward and RSF had the smaller Brier score for the first 8 years with the 2 NN performing better later on. Survival prediction mechanism was examined and all models predicted almost the same mean survival probabilities. Individual predictions revealed difference for the very prognostic level *immuno meds No*. RSF and NN showed extreme predictions whereas Cox models gave more smooth and stricter predictions over time.

Interpretability of the models was of particular interest. For both OS and FFS the most influential hazard ratios for Cox models were for variables *immuno meds* and *retransplantation*. For RSF highest VIMP was provided by *length of stay* and *immuno meds* and for NN highest relative importance for *immuno meds* and *time interval* with *length of stay* next. Note that for the Cox models an interpretation is more handy for categorical variables because numerical variables cannot show their impact at once. RSF showed warning remarks of favoring continuous variables. When comparing OS with FFS, *donor type DCD* (donor after circulatory death) and *accept an hcv antibody positive donor* had the highest increase in hazard ratios, VIMP and relative importance.

All methods were able to provide insights about prognostic factors in their own way. Cox model with hazard ratios has not only the ability to measure the importance of a variable but also its sign. On the other hand, both VIMP and relative importance can only provide absolute importance without providing the direction of the relations. Neural networks can provide the effects of different levels of a variable whereas RSF can only show the total effect of a variable.

In terms of implementation time, RSF required a considerate amount of time and effort for tuning due to the complexity of UNOS dataset. A random subset of 5000 patients was selected for each grown tree. This increased the randomization and thus probably the generalization performance but it is possible that another subset combination or even bootstrapping (with replacement) the whole list of patients could improve forest performance even further. For NN data transformation was needed and tuning was performed on a 5-D space with 3 different transformation functions. This fact in combination with the lack of a proper performance measure made the combination selection demanding. Implementation of the Cox models was fast with even the variable selection LASSO requiring less than a minute.

Neural networks and random survival forests offer a very wide toolbox which can be extensively explored. For the case of PLANN approach, an extension could be made to more activation functions for the input-hidden layer with keras offering a wide list of choices. Would another activation function guarantee a larger stability to the network? Needless to mention that the hyper-parameter choices are massive. This demonstrates that time investment and experience is needed to facilitate a neural that will be highly performing and stable. RSF have a well documented library of implementation in R. It would be interesting to see how a forest would perform under default package choice of mtry = 11, nodesize = 3, nsplit = 0 (meaning deterministic splitting). In chapter

3, we discussed about deterministic splitting and the fact that it requires examining all complementary pairing of levels. Thus, we decided deliberately to use randomized splitting setting parameter nsplit > 0 to ease computational complexity. Can a default choice model outperform Cox models and neural networks?

NN and RSF can scientifically contribute to prediction modeling for liver transplantation if treated appropriately. RSF both for exact and discrete time points outperformed significantly the Cox models in C-index performance, so it can be a great value tool for categorization of patients in high or low risk groups. Neural networks are able to separate true events from false events and Cox models are salutary in terms of straightforward interpretability and have fast implementation. It is a matter of choice how one would select which one to use as long as he is aware of the advantages and disadvantages.

Chapter 7

Discussion

7.1 Machine learning in medical applications

With the rise of computational power and technology on 21^{st} century, more and more data have been collected in the medical field to identify trends and patterns which will allow building of better allocation systems for patients, more accurate prognosis and diagnosis as well as more accurate identification of prognostic factors. During the past few years, machine learning (ML) has received an increased attention in the medical area with methods and applications being more necessary than ever.

ML methods include deep learning, random forests, support vector machines, Bayesian networks, artificial neural networks and other advanced algorithmic techniques. These methods - and deep learning in particular - can automatically diagnose diseases making diagnostics cheaper and more accessible for people all over the world (Markus Schmitt). ML utilize algorithms to learn patterns in data and can take decisions very fast. However, they need a large amount of data to be trained appropriately - the larger the better. A field of medicine that ML has great potential is personalized treatment. Different patients respond to treatments and medicine in a different way. Therefore, personalized treatment can be used to increase the lifespan of patients. With ML, characteristics indicating that a patient will have a particular response to a particular treatment can be revealed.

In survival analysis field, implementation of machine learning techniques is still limited compared to other areas. Possible reasons associated are: 1) the complications that are brought by the censoring mechanism to time to event data which makes the implementation of machine learning methods arduous and 2) the limited amount of interpretability which renders these methods "black boxes" in the eye of medical personnel and doctors. Very often, statistical models (SM) are preferred due to their ease of use and straightforward interpretation.

Predicting survival of patients after liver transplantation is one of the most challenging areas in medicine due to the dependency from both donor and recipient characteristics. With machine learning, models can be built that predict if a patient will lose an organ after liver transplantation within a specific time horizon (Jarmulski et al., 2018). Graft failure or primary non-function can be predicted at liver transplant decision time with the use of ML methodology (Lau et al., 2017) thus assisting use of donor liver resources and ensuring prioritization of high risk groups of patients. Note that accurate survival predictions will result in better treatment decision.

In this thesis, prognostic factors were identified that affect either overall survival

(time to death) or failure-free survival (minimum time to death or graft failure) for patients from USA after liver transplantation. Both machine learning and statistical models were used and attention was given to the interpretability. A comparison of predictive performance for different models was applied using measures widely used in survival analysis and machine learning for simple classification.

7.2 Choosing between machine learning and statistical modeling

Following (Harrell Frank) ML can be distinguished from SM with the following:

1. **Uncertainty**. SM take uncertainty into account by specifying a probabilistic model to the data. For survival analysis, the Cox proportional hazards semi parametric model is often used. On the contrary, ML techniques take uncertainty into account by nature. Neural networks are random stochastic processes where randomness is initialized through the weights which are later tuned using an appropriate optimization algorithm. Furthermore, random forests are injected randomization in 2 ways: bootstrapping a number of patients at each tree and selecting a subset of variables for growing its nodes.

2. **Structure**. Statistical models typically assume additivity of predictors. Non-linearities and interaction effects can be manually specified to extend the models in cases where forms of non-linearity are suspected. For Cox models, the assumption of proportionality of hazards is made. On the other hand, ML models do not make any assumptions about the structure of the data having ultimate flexibility. An easy way to distinguish ML models from SM is considering the prognostic variables. ML does not give special emphasis to additive effects while SM favor by default additivity of predictor effects.

3. **Empiricism**. SM have identified parameters of interest beforehand, whereas ML techniques learn patterns from the data they are fed allowing for high order interactions automatically.

In this thesis two machine learning methods for time to event data were used: random survival forests (Ishwaran et al., 2008a) and a partial logistic artificial neural network (PLANN) (Biganzoli et al., 1998).

SM are often used to select a subset of predictors which are good prognostic factors for the outcome of interest. Here, two such models were estimated. A Cox model with backward elimination and a Cox model using the least angle and selection operator (LASSO) (Tibshirani, 1997).

ML is more an algorithmic approach which aims at the minimization of a particular loss function. Training of these models is needed to identify the best combination of tuning parameters. On the contrary, SM do not require training offering a fast implementation. Notably, there are intersections between ML and statistical regression models. Applying a neural network with the linear (identity) activation function $f(x) = x$ is equal to fitting a linear model at the inputs. Also applying a random survival forest with one variable is identical to univariate Cox regression.

Discussion in the medical field about ML is very popular nowadays. Many scientists are skeptical because of a number of pitfalls as incorrect parameter tuning, suboptimal

selection of performance measures and lack of interpretability (Chen and Asch, 2017). However, machine learning and artificial intelligence have demonstrated great successes especially in high signal-noise situations e.g. visual - sound recognition, video games and more recently in medical applications such as image recognition and disease diagnosis.

SM may be preferable if:

- signal-noise ratio is not large; which is very common in medical applications.
- interest is in the uncertainty of predictions. Cox model and any regression model can measure uncertainty through confidence intervals.
- additivity is the dominant way that predictors affect the outcome. SMs contain inherently this assumption.
- sample size is not large.
- interest is on the effect of special variables such as a treatment.
- emphasis is on the interpretability.

ML may be preferable if:

- signal-noise ratio is large.
- overall performance is the goal.
- non-additivity/complexity is expected to be strong.
- sample size is large.
- interest is not on the effect of a specific variable.
- emphasis is not on interpretability.

This distinction of SM and ML can be useful in taking decisions in which case to use one approach or the other.

For this project, UNOS data contained 62294 cases and 106 variables, 52 regarding the donor and 54 regarding the patient - a high dimensional setting for medical data. A pitfall of Cox models for large sample sizes is that p-values get shrunk and more and more variables get significant. In contrast, ML application is favored by large sample sizes.

7.3 Approach and review of findings

In this manuscript a variety of methods both from statistics and computer science were examined for UNOS liver transplantation data and prognostic factors were identified post operatively. From statistical methods (SM) a numerically stable version of backward elimination was applied and penalized Cox regression using LASSO (Tibshirani, 1997). From machine learning (ML) techniques, random forest and neural network extensions for survival analysis were employed. More specifically, random survival forest (RSF) by (Ishwaran et al., 2008b) and the partial logistic artificial neural

network (PLANN) by (Biganzoli et al., 1998). In chapter 6, comparison of the models performance was carried out. Comparisons were split into two parts 1) comparisons at exact time points (all survival times) where the Cox model with all 106 prognostic factors, Cox backward, Cox LASSO and RSF were compared and 2) comparisons at 12 discrete time points-intervals where 6 models were compared: Cox model with the 106 prognostic factors, Cox model with backward selection, Cox model with LASSO, RSF, a neural network with 1 hidden layer and a neural network with 2 hidden layers. Two clinical outcomes were examined overall survival (OS) and failure-free survival (FFS).

UNOS data include 62294 individuals and contained initially more than 600 variables. 106 variables were preselected from whom 72 were dichotomous, 11 polytomous and 23 continuous. Missing values were imputed to reconstruct the full dataset using an exhaustive random forest algorithm (Stekhoven and Bühlmann, 2012). Strategies for variable pre-selection and data imputation were presented with great detail in chapter 2.

In chapter 3, an overview of all methods was described where emphasis was on theoretical background, technical details, the connection with survival analysis and software of implementation. Predictive performance measures were discussed in chapter 4. Measures that are commonly used (Van Houwelingen and Putter, 2011) are the concordance index, Brier and integrated Brier score. For machine learning under single classification setting accuracy, sensitivity (or recall), specificity, precision and f1 score were used. For model building, the *split sample* approach was used. 2/3 of the complete UNOS data were used to train and develop models and 1/3 was used to test models performance. In case of models where tuning of parameters was required - for instance Cox regression with LASSO, random survival forest or neural networks - training was performed using 5-fold cross-validation and the same folds for all methods to get a fair comparison.

In chapter 5, the main part of the results for different models is illustrated. For OS, 70.9% of the patients were censored with 25, 50 and 75 % quantiles of survival time 1.71, 3.95 and 6.96 years, while for FFS 68.7% were censored with survival time quantiles 1.39, 3.8 and 6.84 years respectively. Cox backward method from `rms` package (Frank E Harrell Jr, 2018) selected 28 variables for OS with most prognostic factors *immuno-maintenance medicine* (90% decrease in hazard for level yes), *re-transplantation* (55% increase for level yes), *incidental tumor* (41% increase), *life support* (39% increase) and *extra-corporeal liver* (32% decrease for level yes). For FFS, 32 variables were selected with most prognostic factors *immuno-maintenance medicine* (92% decrease in hazard for level yes), *re-transplantation* (52% increase for level yes), *donor type* (39% increase for donor after cardiac death vs donor after brain death), *extra-corporeal liver* (32% decrease in hazard for level yes), *incidental tumor* (36% hazard increase) and *life-support* (35% hazard increase).

Variable selection with LASSO was performed using the ad-hoc 1 standard error rule to get larger λ_{LASSO} penalty and thus a more parsimonious and easier to interpret model. For overall survival, 39 variables were selected using 5-fold cross validation in the expanded data set with binary coding of polytomous variables (128 variables). Repetition of the 5-fold cross validation 250 times showed that only 19 variables were stably selected in all repeats with *immuno-maintenance medicine* being selected only 9 times which shows instability in the variable selection. Looking at the prediction performance, different fits of λ_{min} and λ_{1se} varied little in discriminative ability. The

strongest 3 prognostic factors were *immumo-maintenance meds:Yes*, which decreases hazard by 89%, *re-transplantation* which increases hazard by 46% and *life-support*, which increases hazard by 25%, given the other covariates are kept constant.

For FFS, 32 variables were selected from 5-fold cross validation. Repeating, 21 variables were always selected with *immuno-maintenance meds* selected 79 times, thus showing higher selection stability for FFS. Performance was almost identical for different fits with λ_{min} and λ_{1se}. Strongest 3 prognostic factors were *immumo-maintenance meds:Yes*, which decreases hazard by 91%, *re-transplantation* which increases hazard by 36% and *life-support* which increases hazard by 24%.

For RSF, an extensive grid search was performed on a 3-D space to identify best values for parameters *nodesize* (determines average node size across forest), *mtry* (determines number of randomly selected variables) and *nsplit* (determines the number of split points at which an x-variable is tested) based on forest error (1 - C), where C is the concordance index calculated using the ensemble mortality score ordering. Two different split rules *log-rank* and *log-rank score* were used with the *log-rank* proving superior in accuracy and more efficient in terms of computational time.

Random forest has a unique characteristic which is collection of out-of-bag samples (OOB). During bootstrapping about 63% of the cases are used to grow a tree and 37% of the cases are left out. These left-out/OOB instances can be used to form estimates of important quantities and obtain prediction error which is almost identical to that of a leave-one-out cross validation. For UNOS data, tuned parameters from 5-fold cross validation and out-of-bag error were the same for OS and FFS. This suggests that for survival data, OOB error can be used as a fast alternative for parameter tuning when searching at this particular 3-D space. However, research has shown that *mtry* parameter is related with OOB error bias (Janitza, 2017). Therefore, research is needed for different number of tuning parameters and grid choices to determine if OOB is an appropriate choice.

Furthermore, after tuning *nodesize*, *mtry* and *nsplit*, parameter *ntree* (number of trees) was examined to identify a suitable number of trees that stabilizes OOB error. More trees can reduce variability and increase stability and thus consistency of the outcome. For both OS and FFS 250 trees proved to be a sufficiently large choice.

Moreover, the 5 variables most frequently used by the *log-rank* splitting rule were shown. For OS, all of them were continuous and for FFS all but *immuno-maintenance meds* were continuous which illustrates a biased trend of RSF towards continuous variable selection (Loh and Shih, 1997) during binary partitioning especially considering that for UNOS data 83 out of the 106 variables were categorical (dichotomous or polytomous). This effected was also reflected in the interpretability of the variables.

Two methods were used to get interpretation of the forest: variable importance (VIMP) and minimal depth. VIMP measures the difference in overall error rate before and after a predictor's permutation and minimal depth how close to the root node a predictor was first used as a split. Measures are unrelated to each other with VIMP being dependent on the error and minimal depth on the topology of the forest. For OS the most prognostic variables identified by VIMP at exact time points were *length of stay* (0.058), *immuno meds* (0.038), and *hcv serology status* (0.0058) at a far smaller margin. Similarly, in terms of minimal depth *immuno meds* (1.164) and *length of stay* (1.364) were by far the shallower variables. For FFS at exact time points, VIMP was the highest for *length of stay* (0.066), *immuno meds* (0.042) and *donor age* (0.007) at

a far smaller margin. In minimal depth, *immuno-maintenance medicine* (1.516) and *length of stay* (1.960) were again the most important.

To fit a survival neural network, a specific data transformation to longitudinal type format was required for both training and test set. Patients were divided into 12 time intervals denoting years since transplantation and the intervals were used as part of the input variables to add extra information. Output of the network was the conditional failure probability. A detailed description of the PLANN is given in 3.4.3.

Grid search was performed on a 5-D space to identify best values of parameters *nodesize* that defines the number of weights of the network and consequently the amount of complexity, *dropout rate* that randomly selects the amount of nodes to be dropped-out with a given probability, *learning rate* which is the step size of weight iteration, *momentum* which helps accelerate gradient vectors and *weak class weight* that defines the weight of minority class. In the original PLANN approach, Biganzoli used a single hidden layer with activation function of both input-hidden and hidden-output layer the *sigmoid*. In this thesis, we employed two more activation functions for input-hidden layer: the rectified linear unit *ReLU* that is the most widely used activation function for neural networks and the hyperbolic tangent *tanh*. The partial logistic neural network was expanded in 2 hidden layers with same *nodesize* and identical activation functions for input-hidden 1 and hidden 1 - hidden 2 layers. Tuning of the parameters was based on combinations that lead to the minimization of the binary cross-entropy error for each activation function. The lack of a global measure that can be used to evaluate the performance of the 5-fold cross validation forced us to use a flexible approach including a number of measures for evaluation namely: accuracy, sensitivity, specificity, precision, f1 score and the integrated Brier score (IBS) from survival. Parameters were chosen based on the combinations that produce the best model performance for the majority of the measures based on the minimized cross-entropy error. Special attention was given to accuracy as it is the most widely used measure for evaluation of classification neural networks.

For overall survival, the best overall performance in cross-validation for a 1 hidden layer NN was observed for activation functions *sigmoid* and *ReLU*. *Sigmoid* had consistently the lowest IBS and *ReLU* the highest accuracy. The final selected parameters were examined in terms of stability and indications of instability were found. Relative importance through the connection weights method by Garson (1991) showed that variables *immuno-maintenance* (0.136) and *time interval* (0.079) were the most contributing for the predictions of the network. Third most contributing variable was *length of stay* (0.038) then *re-transplantation* (0.034) and *donor HLA typed:Yes* (0.030).

For failure-free survival, the final network had the *sigmoid* activation function and was similar to Biganzoli's PLANN approach. This network in particular showed stability. Examination of the most contributing variables using Garson's algorithm revealed that *immuno meds* have an increased relative importance for FFS (0.191) compared to OS. *Time interval* was also more contributing having relative importance of 0.107. The rest of the top 5 variables were *length of stay* (0.038), *donor type:DCD* (donation after circulatory death, 0.034) and *re-transplantation* (0.029). Factors with increased importance were *portal hypertensive bleeding*, *races Black and Other* as well as *portal vein thrombosis*.

The transition to a neural network with 2 hidden layers showed that *sigmoid* activation function is improper as such network is over-fitting predicting only alive

patients. This is caused by the fact that sigmoid function saturates and turns off the gradients for multiple hidden layers (Isaac Changhau). On the other hand, *ReLU* and *tanh* were able to give good combinations in the global performance measures. For OS, *ReLU* activation functions were selected for the input-hidden 1 and hidden 1-hidden 2 layers whereas for FFS *tanh*. Examination of stability on those networks showed that both were unstable.

The main aim of this thesis was to evaluate the predictive performance of the three Cox models: 1) Cox model with the 106 prognostic factors, 2) Cox with backward elimination, 3) Cox with LASSO; and two machine learning techniques: 1) Random survival forests and 2) Neural networks. The data transformation for the neural network to 12 time intervals made the direct comparison with Cox models and the RSF unfair as Cox models and RSF use exact time points. Therefore, it was decided to separate the comparisons into two parts 1) comparisons at exact time points and 2) comparison at distinct time points where Cox models and RSF also summarize survival information in 12 distinct time points 1, 2, ..., 12 years.

Comparison at exact time points for both OS and FFS showed that RSF outperforming Cox models in global performance measures from survival analysis and confusion matrix metrics. In terms of C-index RSF was considerably better showing a model that is very good at discriminating between low and high risk patients. A different approach of evaluating the model's performance on the training set was discussed using bootstrapped based cross-validation repeated $B = 100$ times. It was shown that it can be an attractive alternative to the split data method for data analysis. Finally, prediction curves for new hypothetical patients showed differences. RSF predicted higher survival probabilities than Cox models in most of the cases but much lower at the first 2 years for *immuno meds:No*.

Comparison at discrete time points was made for the 3 Cox models, a RSF model and 2 models for NN - 1 and 2 hidden layers. Note that for survival neural networks, concordance index was not defined as there is no natural ordering of the subjects and this measure was not applicable. 5-fold cross-validation was repeated for the models that required re-tuning and new parameters were identified. Compared with the exact time points, the Cox models and RSF reported a slight decrease in concordance and a slight increase in integrated Brier score (IBS). The 5 confusion matrix measures had slightly increased values. RSF had the best performance in accuracy, sensitivity and C-index and the second best in IBS. Neural networks had the best performance in specificity, precision and f1 score and the Cox backward model had the best performance in IBS. A major difference between OS and FFS was observed for the IBS of the neural networks. For OS the neural networks exhibited a much higher error compared to other methods, whereas in FFS their performance was comparable to the others. Note that for Brier score, neural networks outperformed the other models after 8 years, even if this is not of great value as the vast majority of events was observed at first 2-3 years.

All methods were compared at discrete points in terms of interpretability. Cox model with all variables and Cox with backward elimination had higher hazard ratios compared to penalized Cox with LASSO; a fact attributed to LASSO's ability not only to select but also to shrink coefficients. For both OS and FFS the most contributing variable was *immuno meds* and then *re-transplantation*. RSF using the VIMP method identified *length of stay* and *immuno meds* as the strongest predictors for both survival outcomes. For neural networks, the *immuno meds* were by far the

most prognostic followed by *time interval* and *length of stay* with relative importance increased for FFS. *Donor type DCD* (donor after circulatory death) and *accept an hcv antibody positive donor* are more contributing for FFS whereas *life-support*, *diabetes* less contributing. Variable *can_acpt_extracorp_li* (did patient accept an extra corporeal liver) was prognostic for the Cox models but was not identified in the most prognostic variables by ML techniques.

7.4 Limitations and considerations

During data pre-processing in chapter 2, several variables had to be modified to create more balanced classes while others had to be removed completely due to large percentage of missingness. This might limit the generalization of the findings to the standard analysis files (SAF) of SRTR. In general, thorough pre-processing of the data is required to get a format which can be used for analysis. A strong recommendation to UNOS personnel is to adjust the levels of polytomous prognostic factors in a more compact and balanced way. To explain this, very unbalanced classes and factors consisting of more than 6 levels can undermine the applicability, the efficiency but also the interpretability of statistical methods.

Results in chapters 5 and 6 are with respect to UNOS data and patients from the United States. Generalization of the results for patients who underwent liver transplantation outside USA can be misleading. This is due to the fact that survival probability is affected by regional status, diet, different cultural traditions and many other health and socioeconomic factors.

To reconstruct the 2.5% of missing data in preselected variables, single imputation was used. 2 other alternatives could have been used: to proceed with the 27411 complete cases (out of 62294) or to use multiple imputation. The first alternative would mean a huge waste of data and could lead to biased survival inference so it was rejected at once. The second option was much more attractive as multiple data replicates would preserve the inherit uncertainty of the observations. Nevertheless, it was decided to apply single imputation as multiple data sets would dramatically increase computational time and amount of effort required for implementation. Note, yet that a single imputation accounted for the process that created the missing values and preserved the relations in the continuous and the categorical covariates.

Another limitation in this project is the Cox model specification. It was decided for simplicity to use naive Cox models that incorporate only single variable effects. 2 machine learning techniques RSF and NN were compared. Note that for ML techniques non-linearities between variables are modeled automatically. Therefore, more sophisticated Cox models could have been considered using variable-time or variable-variable interactions based on doctors expertise and statistical findings. It is of interest to see how such model would behave compared to the RSF that showed superiority in both exact and discrete time points.

Considering time of implementation, RSF proved to be cumbersome for UNOS data. The memory of a laptop was not sufficient, and 5-fold cross-validation required a significant amount of time; thus a supercomputer was utilized. Neural networks did not have a default library for survival implementation and data transformation from a wide to a long format was required for each patient. Furthermore, an extensive grid search in a 5-D space for 3 activation functions was performed. On the contrary, Cox models had a easy and fast implementation.

Stability of the models was considered for both statistical models and the machine learning techniques. Cox proportional hazards model (Cox, 1972) is remarkably stable with the algorithm producing the same hazard ratios every time. From the machine learning models that are inherently based on mechanisms injecting randomization, RSF can achieve remarkable stability in OOB error through the *ntree* parameter. A sufficient number of trees can be selected for safe performance. On the other hand, 3 of the 4 NN applied in this thesis showed apparent remarks of instability of performance which renders them a risky choice.

Pitfall of survival neural networks is the lack of a global measure that can be used to determine a good combination for the hyper-parameters. In chapter 5, tuning of the parameters was based on the minimization of the cross-entropy error and selection was made based on a combination that can offer a good overall performance. Nevertheless, such choice can lead to criticism (Chen and Asch, 2017).

Interpretation of models is of paramount importance for medical personnel. Cox model is offering an easy interpretation through hazard ratios for both categorical and continuous variables. Hazard ratios for continuous variables are associated with one unit increase. However, the actual effect is hard to perceive at once. For techniques from the area of machine learning, there is usually a lack of attention in interpretability. For neural networks with one hidden layer the connection weights method of Garson (1991) can provide insights. Predictors importance is quantified in terms of relative importance and is associated with the weights values that are transmitted from input to output node. Each level of a variable receives its relative importance as factors are binary coded. For random survival forests two methods can provide interpretation VIMP and minimal depth where the first is associated with the error before and after the permutation of a prognostic factor and the second with forest topology. A limitation of the methods is that interpretation is available only for a variable entity and not for all levels if categorical. Note that both relative importance in NN and VIMP, minimal depth for RSF indicate only the direction of the relationships, so it remains unclear if it indicates a positive or a negative effect. For the neural networks with 2 hidden layers interpretation is a major problem, so the well known "black-boxes" is appropriate.

The models could not be directly compared at exact time points because of the necessary data transformation for NN with patients being divided into 12 time intervals denoting years since transplantation. Therefore, comparisons were split in two folds: exact time points for the Cox models and RSF and discrete time points for Cox models, RSF and NN. For discrete time points a number of global measures was examined for all models. For neural networks, the concordance index was not applicable because of the lack of unique ordering of survival times. A C-index may be constructed using an artificial ordering of patients.

A cut-off probability of 0.5 for survival times was considered to build a confusion matrix at the last observed time point for each patient. A patient with predicted survival probability higher than 0.5 was categorized as non-event otherwise as an event and was compared with the real status. However, based on medical application a stricter or a more lenient cut-off can be used to evaluate model performance. It is of interest to see how different models would perform in terms of machine learning metrics (accuracy, sensitivity, specificity, precision, f1 score) under different probabilistic thresholds.

For many years the Cox model has been established as dominant for time to event

data. Here we presented two alternatives from machine learning. RSF performed very well for both exact and discrete time points and demonstrated great potential in allocating patients in high and low risk groups. This can be of benefit for doctors to identify groups of patients in whose they should pay more attention. NN may not exhibited a better performance in survival analysis metrics but they showed potential in classification setting. High performance was observed in terms of precision which illustrates their ability to identify true events versus false events in a much higher ratio than the other models. This makes them more reliable when prediction of an event is made (death or graft failure).

7.5 Recommendations for future research

This thesis revealed a number of fields where future research is needed. First of all, predictive performance of the rival models was compared under certain censoring distributions for overall (70.9% censoring) and failure-free survival (68.7% censoring). It would be of interest to see how the models will perform under different censoring percentages.

From the machine learning techniques, random survival forests and survival neural networks were utilized. As mentioned before, other techniques can be applied such as support vector machines, Bayesian networks. A comparison of performance between these techniques with methods applied here would add extra information to the discussion between ML and SM. Can they be competitive with the methods applied here? Is it possible to get interpretability from them? Which variables of the 52 regarding donor and 54 regarding the patient would show strong predictive ability?

With 106 variables on the dataset - 23 continuous and the rest categorical - RSF seemed to favor the continuous variables during binary partitioning with the *log-rank* split rule. For OS, the 5 most frequently used variables during splitting were continuous and for FFS 4 of the 5 even if *nsplit* parameter was applied to trigger a randomized selection of exactly *nsplit* points for each of the *mtry* variables. Looking at variable importance (VIMP), *length of stay* was the strongest predictor for OS and FFS with *recipient age* and *donor age* also among the most prognostic variables. On the other hand, for Cox models and neural networks these variables were also prognostic but to a smaller degree. For Cox models, those variables resulted in small hazard ratios. For neural networks with one hidden layer, variable *length of stay* was the third most prognostic at a much smaller margin compared to *immuno-maintenance meds* and *time interval* with the *age of donor and recipient* having small relative importance. These results demonstrate that there was a bias towards the selection of continuous variables from the RSF. Further investigation is needed under different ratios between continuous and categorical variables for other datasets. (Loh and Shih, 1997) have showed that classification trees algorithms tend to be biased towards selecting variables that afford more splits. If this is the case for survival trees, research is necessary to investigate how to reduce bias.

Deep learning with neural networks is gaining more and more attention. In this manuscript, the PLANN approach was examined that estimates conditional hazard $h(t)$ probabilities. More approaches in literature estimate either the survival probability $S(t)$ or the unconditional probability of death $p(t)$ (Ripley and Ripley). All these methods require a-priori data transformation and can have one or multiple output nodes. For example Lapuertas method (Lapuerta P, 1995), defines K separate

networks to deal with censoring. Comparison of such networks could be made with Biganzoli's approach and its extensions (different activation functions, 2 hidden layers).

Moreover, new state-of-the art feed forward neural networks have come to light for survival analysis. They calculate directly a proportional hazards model as part of the output node. A direct implementation of such a network is provided in library *cox-nnet* of Python programming language (Ching et al., 2018). However, in R programming language there is a lack of a software of direct implementation. This could be part of research.

A notorious characteristic of neural networks is lack of stability. A trained network may diverge considerably in performance when fit under the exact same parameters for the exact data. In this thesis, 4 neural network models were fit under different activation functions and one or two hidden layers (two for OS and two for FFS). However, rerunning of such networks showed signs of instability in 3 cases. A possible reason for the instability is the lack of a proper validation measure for their performance in survival analysis. Tuning the hyper-parameters using 5-fold cross validation was based on the minimization of the binary cross-entropy for 3 different activation functions with final combination selected based on the overall performance of the network on different measures. This choice did not lead to stable networks. Therefore, it is of paramount importance to identify a global error measure (invariant to time) which can be reliable for network training. From the machine learning perspective, a highly performing network in accuracy is merely the goal. In survival analysis the C-index and the Brier score are commonly used metrics. The first was not implementable and the second was susceptible to time variation so the integrated Brier score a global summary measure was utilized. The use of it did not provide reasonable results as errors demonstrated bias towards very small node sizes.

A very important part for medical applications is the interpretability of the model. Statistical models are usually preferred because they offer a straightforward interpretation about model coefficients. In machine learning, interpretability is overlooked which makes them unsuitable for clinicians. For RSF and NN different methods were used to extract interpretation for the prognostic factors (VIMP / minimal depth, relative importance respectively). Cox models estimate relative risks whereas RSF and NN quantify the effects of either variable entities or levels. Therefore, the construction of a global measure of additivity is needed so that a direct comparison of SMs and ML can be made.

Adequate health care will soon be impossible without data supervision from modern machine learning methodologies to take personalized decisions for patients. If only special emphasis is given in the creation of state-of-the-art methods which can be used to get straightforward interpretability from the "black boxes", will the doctors feel comfortable to use them on a regular basis.

7.6 Contribution and novelty

This thesis aimed primarily to contribute in the discussion about machine learning (ML) and statistical modeling (SM) offering a comparison between 2 ML techniques: 1)random survival forest (RSF) (Ishwaran et al., 2008a) and 2) survival neural network of Biganzoli et al. (1998) (PLANN) with 3 SM originating from the well-known Cox model family: 1)The Cox model with all 106 prognostic factors, 2)A Cox model with variables selected from backward elimination and 3)A Cox model with variables

selected using LASSO. Prognostic factors were identified and a special effort was given to attain interpretability from machine learning techniques.

The following points are covered in this manuscript:

- Medical information about liver transplantations and how the United Network of Organ Sharing (UNOS) and the Scientific Registry of Transplant Recipients (SRTR) contribute to the process.
- A variety of statistical strategies for data cleaning and variable pre-selection from a set of more than 600 variables.
- Statistical strategies for data imputation and a comparison between 3 packages of the R programming language (R Core Team, 2014) for data reconstruction.
- An extensive analysis of all methods for two survival outcomes **overall** (time to death) and **failure-free** (time to death or graft-failure) survival.
- A short comparison of stepwise statistical methods (forward, backward) for variable selection in a Cox proportional hazards model and utilization of a stable backward elimination method.
- A short comparison of variable selection with LASSO under two regularization parameter choices λ_{min} and the more ad-hoc λ_{1se} rule which provides a larger penalty and thus a more parsimonious model.
- Reporting of the most important prognostic factors for each method for survival of liver transplanted patients post-operatively.
- An extensive application of 2 machine learning techniques RSF and PLANN. A novel expansion of the original neural network proposed by Biganzoli (1 hidden layer with activation function "sigmoid" for input-hidden and hidden-output layer) has been performed. Hereto, two new activation functions 1) the rectified linear unit (ReLU) and 2) the hyperbolic tangent (tanh) have been used for activation of the input-hidden layer. A further expansion of the network includes 2 hidden layers with identical node size and activation functions for input-hidden 1 and hidden 1-hidden 2 layers. Three activation functions were employed a)"sigmoid", b)"tanh", c)"ReLU".
- A comparison between methods under two different scenarios 1)exact time points (the 3 Cox models and RSF are compared) and 2)discrete time points (or intervals), where all models are compared. This distinction ensures a fair comparison between the methods.
- Investigation of methods predictive performance under metrics from 2 different areas survival analysis and classification setting from machine learning. From survival analysis the **C-index**, **integrated Brier score (IBS)**, **the time-dependent Brier score** and **the time-dependent C-index** have been utilized. From machine learning, measures of a simple confusion matrix considered are **accuracy**, **sensitivity**, **specificity**, **precision** and **f1 score**. To the best of our knowledge, the last two have not been applied before in medical literature for model comparison.

- A confirmation of many literature findings about advantages and disadvantages of the methods with special emphasis given to stability of each method.
- Discussions about the interpretability of the models and comparison with respect to implementation effort needed.
- Debate about which model discriminates better between low and high risk patients and which offers more accurate predictions at the last observed time point for each patient.
- Discussions at the end of the sections and illustration of project limitations.
- Summary about machine learning perspective in medical applications and suggestions for further investigation.

Appendix A

Data description

A.1 Description of the variables

Table A.1: Variables regarding the donor.

Variable	Interpretation	Levels (first reference)
don_gender	Donor's gender	Female, Male
don_ethnicity_srtr	SRTR donor ethnicity	Latino, Nlatin
donortype	Type of donor	DBD, DCD
donorrace	Race of donor	White, Black, Other
donorage	Age of donor in years	-
don_abo	Donor/s blood type	A, AB, B, O
don_hgt_cm	Donor's height in centimeters	-
don_wgt_kg	Donor's weight in kilograms	-
don_bmi	Donor's body mass index (BMI)	-
don_log_sgot	Log(x + 1) transformed SGOT/AST	-
donorcod	Donor/s cause of death	Anoxia,CNS_Tumor,CVA_Stroke, Head_trauma, Other
don_creat	Donor's serum creatinine	-
don_anti_cmv	Donor's Anti-CMV	Negative, Positive
don_anti_hcv	Donor's Anti-HCV	Negative, Positive
don_prerecov_diuretics	Donor's pre-recovery meds: Diuretics	No, Yes
don_ddavp	Donor's meds: DDAVP	No, Yes
don_insulin	Donor's insulin	No, Yes
don_inotrop_support	Donor's inotropic support	No, Yes
don_hist_cigarette_gt20_pkyr	Donor's Cigarette Use > 20 packages per year - Ever	No, Yes
don_hist_cocaine	Donor's history of cocaine	No, Yes
don_hist_other_drug	Donor's history of other drug	No, Yes
don_meet_cdc_high_risk	Does donor meet high risk guidelines?	No, Yes
don_hist_diab	Donor's history of diabetes	No, Yes
don_htn	Donor's history of hypertension	No, Yes
don_hist_cancer	Donor's history of cancer	No, Yes
don_log_bun	Log(x + 1) transformed BUN	-
don_tot_bili	Donor's total bilirubin	-
don_log_sgpt	Log(x + 1) transformed SGPT/ALT	-
don_protein_urine	Donor's protein in urine	No, Yes
don_sodium	Donor/s last serum sodium	-
don_inr	Donor/s INR	-
don_hematocrit	Donor's hematocrit	-
don_anti_hbc	Donor Anti-HBC	Negative, Positive
don_prerecov_steroids	Donor's pre-recovery meds: Steroids	No, Yes
don_anti_convuls	Donor's meds: Anticonvulsants	No, Yes
don_anti_hyperten	Donor's meds: Antihypertensives	No, Yes
don_vasodil	Donor Vasodilators	No, Yes
don_heparin	Donor's heparin	No, Yes
don_arginine	Donor's arginine	No, Yes
don_txfus_terminal_hosp_num	Donor/s number of transfusions at hospitalization	None, 1-5, 6-10, >10
don_clinical_infect	Donor clinical infection	No, Yes
don_infect_blood	Donor infection source blood	No, Yes
don_infect_lu	Donor infection source lungs	No, Yes
don_infect_urine	Donor infection source urine	No, Yes
don_heavy_alcohol	Donor Heavy Alcohol Use (heavy= 2+ drinks/day)	No, Yes
don_tattoos	Donor tattoos	No, Yes
don_hla_typ	Was donor HLA typed	No, Yes
don_hist_prev_mi	Donor: history of previous Myocardial infarction	No, Yes
don_pulm_cath	Donor: Was a pulmonary artery catheter placed?	No, Yes
don_hcv_stat	Donor's HCV antibody status	Negative, Positive
don_hbc_stat	Donor's Hepatitis B core antibody status	Negative, Positive
don_prev_gastro_disease	Donor Previous Gastrointestinal Disease	No, Yes

Table A.2: Variables regarding the patient.

Variable	Interpretation	Levels (first reference)
enceph	Candidate: Last encephalopathy	No, Yes
portal_vein	Recipient: Risk factors: portal vein thrombosis	No, Yes
portal_hyperten_bleed	Recipient: Spontaneous portal hypertensive bleeding	No, Yes
prev_abdom_surg	Recipient: Risk factors: Previous Abdominal Surgery	No, Yes
ascites	Candidate: ascites	No, Yes
last_dial_prior_week	Candidate: Last non-blank val. of dialysis within prior week	No, Yes
albumin2	Candidate: Last Albumin	$\geq 2, < 2$
diab	Candidate: Diabetes	No, Yes
hcv	Recipient: HCV serology status	No, Yes
hcc	Candidate: Ever Approved for an HCC exception	No, Yes
malig	Recipient: Pretransplant malignancy	No, Yes
recipientage	Recipient/s age in years	-
split	Recipient: Split type	whole_organ, split
lifesupport	Patient on Life Support	No_lifesupport, Lifesupport
hbv	Recipient: HBV Core Antibody	No, Yes
coldischemiatime	Recipient: Total Cold Ischemic Time	-
share	Candidate: share	Local, Regional, National
recipientsex	Recipient/s gender	Female, Male
etiology	Candidate: etiology (cause of disease)	Metabolic (ref.), Acute, Alcoholic, Cholestatic, HBV, HCV, Malignant, Other_cirrhosis, Other_unknown
pretxstatus	Recipient: pre-treatment status	Not hospitalized, Hospitalized, Intense care unit
can_race_srtr	Candidate/s race	White, Black, Other
can_ethnicity_srtr	Candidate/s ethnicity	Latino, Nlatin
can_abo	Patient/s Blood type	A, AB, B, O
rec_functn_stat	Recipient/s functional status	No_assistance, Some_assistance, Total_assistance
rec_work_income	Recipient: Working for income	No, Yes
rec_postx_los	Patient/s stay at the hospital after transplantation in days	-
rec_bmi	Recipient/s body mass index (BMI)	-
retransplantation	Did patient have a transplant surgery before?	No_retransplantation, retransplantation
rec_cmv_stat	Recipient: Pre-Tx Serology Test Results: Cytomegalovirus	Negative, Positive
rec_hbv_surf_antigen	Recipient: HBV surface antigen	Negative, Positive
rec_immuno_maint_meds	Are any meds given for maintainance or anti-rejection?	No, Yes
rec_acute_rej_episode	Did patient have any acute rejection episode?	No, Yes
rec_tumor	Recipient: Incidental tumor found at time of transplant	No, Yes
rec_warm_isch_tm	Recipient: Warm ishcemic time	-
rec_bacteria_perit	Recipient: Risk factors - Spontaneous Bacterial Peritonitis	No, Yes
can_variceal_bleeding	Candidate: Variceal Bleeding within last two weeks	No, Yes
rec_tipss	Recipient: Transjugular Intrahepatic Portacaval Stent Shunt (TIPSS)	No, Yes
can_last_bili	Candidate/s Last Bilirubin	-
can_last_inr	Candidate/s Last INR	-
can_last_serum_creat	Candidate/s Last Serum Creatinine mg/dl	-
can_last_serum_sodium	Candidate/s Last Serum Sodium	-
can_education	Candidate: Educational status	None_low_undefined, Medium, Higher
rec_hgt_cm	Recipient: Height in centimeter	-
rec_wgt_kg	Recipient: Weight in kilograms	-
can_peptic_ulcer	Candidate: Peptic Ulcer Disease	No, Yes
can_angina_cad	Candidate: Angina	No, Yes
can_drug_treat_hyperten	Candidate: Drug Treated Systemic Hypertension	No, Yes
can_cereb_vasc	Candidate: Symptomatic Cerebrovascular Disease	No, Yes
can_periph_vasc	Candidate: Symptomatic Peripheral Vascular Disease	No, Yes
can_acpt_abo_incomp	Did the candidate accept an incompatible blood type?	No, Yes
can_acpt_extracorp_li	Did the candidate accept an extra-corporeal Liver?	No, Yes
can_acpt_li_seg	Candidate: accept liver segment	No, Yes
can_acpt_hbc_pos	Candidate: Accept an Hepatitis B Core Antibody Positive Donor?	No, Yes
can_acpt_hcv_pos	Candidate: Accept an HCV Antibody Positive Donor?	No, Yes

Appendix B

Application

B.1 Cox Proportional Hazards models

Table B.1: Cox proportional hazards model with all prognostic factors: Hazard ratios and lower and upper 95% confidence intervals at exact time points (Overall Survival). The 5 most detrimental factor levels are colored in red and the 5 most beneficial are colored in green.

Variable	exp(coef)	lower .95	upper .95	Variable	exp(coef)	lower .95	upper .95
don_genderM	1.062	1.010	1.117	ascitesY	1.038	0.995	1.083
don_ethnicity_srtrNLATIN	0.952	0.899	1.009	last_dial_prior_weekY	1.071	1.006	1.141
donortypeDCD	1.164	1.063	1.274	albumin2<2	1.113	1.038	1.194
donorraceBlack	0.978	0.929	1.031	diabY	1.233	1.180	1.288
donorraceOther	1.119	1.010	1.241	hcvY	1.132	1.074	1.193
donorage	1.007	1.006	1.009	hccY	1.167	1.105	1.232
don_aboGroup AB	0.946	0.787	1.136	maligY	1.063	0.978	1.154
don_aboGroup B	0.902	0.779	1.045	recipientage	1.014	1.012	1.016
don_aboGroup O	0.958	0.854	1.076	splitsplit	1.104	0.931	1.310
don_hgt_cm	0.999	0.993	1.006	lifesupportLifesupport	1.302	1.200	1.413
don_wgt_kg	0.998	0.991	1.006	hbvY	1.049	0.999	1.101
don_bmi	1.005	0.985	1.026	coldischemiatime	1.007	1.001	1.012
don_log_sgot	1.022	0.988	1.056	shareRegional share	1.018	0.974	1.064
donorcodCNS Tumor	1.106	0.873	1.400	shareNational share	1.151	1.060	1.250
donorcodCVA/Stroke	1.049	0.992	1.110	recipientsexMale	1.018	0.964	1.074
donorcodHead trauma	0.998	0.939	1.061	etiologyAcute	0.967	0.822	1.137
donorcodOther	1.104	0.967	1.261	etiologyAlcholic	1.066	0.927	1.227
don_creat	1.000	0.985	1.014	etiologyCholestatic	0.851	0.730	0.993
don_anti_cmvP	0.998	0.959	1.039	etiologyHBV	0.952	0.783	1.157
don_anti_hcvP	1.230	0.685	2.208	etiologyHCV	1.144	0.991	1.320
don_prerecov_diureticsY	0.936	0.901	0.972	etiologyMalignant	1.195	1.029	1.388
don_ddavpY	0.975	0.924	1.028	etiologyOther cirrhosis	1.021	0.886	1.177
don_insulinY	1.019	0.980	1.060	etiologyOther/unknown	1.002	0.861	1.166
don_inotrop_supportY	0.999	0.962	1.039	pretxstatusHOSPITALIZED	1.071	1.008	1.138
don_hist_cigarette_gt20_pkyrY	1.021	0.978	1.067	pretxstatusIC UNIT	1.107	1.019	1.202
don_hist_cocaineY	0.991	0.934	1.052	can_race_srtrBlack	1.195	1.126	1.268
don_hist_other_drugY	0.939	0.897	0.983	can_race_srtrOther	0.809	0.736	0.889
don_meet_cdc_high_riskY	0.988	0.927	1.054	can_ethnicity_srtrNLATIN	1.144	1.079	1.213
don_hist_diabYES	1.043	0.981	1.108	can_aboGroup AB	0.971	0.835	1.128
don_htnY	1.043	0.994	1.094	can_aboGroup B	1.098	0.953	1.266
don_hist_cancerY	1.086	0.988	1.194	can_aboGroup O	1.025	0.912	1.151
don_log_bun	1.009	0.970	1.050	rec_functn_statSome assistance	1.095	1.043	1.150
don_tot_bili	1.013	1.000	1.027	rec_functn_statTotal assistance	1.162	1.091	1.237
don_log_sgpt	0.988	0.956	1.021	rec_work_incomeY	0.814	0.765	0.867
don_protein_urineY	1.020	0.982	1.059	rec_postx_los	1.006	1.005	1.006
don_sodium	1.001	0.999	1.003	rec_bmi	1.008	0.984	1.032
don_inr	1.007	0.997	1.018	retransplantationRetransplantation	1.573	1.462	1.693
don_hematocrit	0.999	0.996	1.003	rec_cmv_statP	0.971	0.934	1.009
don_anti_hbcP	1.051	0.745	1.481	rec_hbv_surf_antigenP	0.925	0.831	1.029
don_prerecov_steroidsY	1.002	0.958	1.047	rec_immuno_maint_medsY	0.100	0.094	0.107
don_anti_convulsY	1.028	0.945	1.119	rec_acute_rej_episodeYes	1.076	1.000	1.158
don_anti_hypertenY	0.998	0.953	1.045	rec_tumorY	1.391	1.268	1.525
don_vasodilY	1.001	0.949	1.056	rec_warm_isch_tm	1.002	1.001	1.003
don_heparinY	1.073	0.995	1.158	rec_bacteria_peritY	1.016	0.952	1.085
don_arginineY	0.987	0.950	1.026	can_variceal_bleedingY	1.132	1.032	1.241
don_txfus_terminal_hosp_num1-5	1.001	0.958	1.046	rec_tipssY	1.071	1.005	1.140
don_txfus_terminal_hosp_num6-10	1.037	0.962	1.118	can_last_bili	1.001	0.999	1.002
don_txfus_terminal_hosp_num>10	0.948	0.867	1.036	can_last_inr	0.998	0.981	1.014
don_clinical_infectY	0.945	0.881	1.013	can_last_serum_creat	1.037	1.023	1.052
don_infect_bloodY	1.025	0.954	1.102	can_last_serum_sodium	1.007	1.003	1.010
don_infect_luY	0.942	0.881	1.007	can_educationMedium education	0.966	0.923	1.011
don_infect_urineY	1.002	0.938	1.071	can_educationHigher education	0.927	0.875	0.983
don_heavy_alcoholY	0.997	0.945	1.052	rec_hgt_cm	1.003	0.995	1.011
don_tattoosY	1.010	0.966	1.056	rec_wgt_kg	0.995	0.987	1.003
don_hla_typY	1.009	0.879	1.159	can_peptic_ulcerYes	1.101	1.002	1.211
don_hist_prev_miY	1.069	0.977	1.170	can_angina_cadYes	1.337	1.199	1.491
don_pulm_cathY	1.004	0.949	1.061	can_drug_treat_hypertenY	0.998	0.954	1.043
don_hcv_statPositive	0.874	0.486	1.570	can_cereb_vascY	1.179	0.973	1.430
don_hbc_statPositive	0.952	0.673	1.347	can_periph_vascY	1.021	0.865	1.206
don_prev_gastro_diseaseY	1.031	0.960	1.106	can_acpt_abo_incompY	0.903	0.778	1.048
encephY	1.025	0.982	1.069	can_acpt_extracorp_liY	0.700	0.617	0.793
portal_veinY	1.185	1.088	1.290	can_acpt_li_segY	0.954	0.905	1.006
portal_hyperten_bleedY	1.091	0.994	1.198	can_acpt_hbc_posY	0.971	0.931	1.013
prev_abdom_surgY	1.120	1.077	1.165	can_acpt_hcv_posY	1.152	1.100	1.205

Table B.2: Cox proportional hazards model with all prognostic factors: Hazard ratios and lower and upper 95% confidence intervals at exact time points (Failure-free Survival). The 5 most detrimental factor levels are colored in red and the 4 most beneficial are colored in green.

Variable	exp(coef)	lower .95	upper .95	Variable	exp(coef)	lower .95	upper .95
don_genderM	1.063	1.013	1.116	ascitesY	1.030	0.988	1.073
don_ethnicity_srtrNLATIN	0.944	0.893	0.998	last_dial_prior_weekY	1.036	0.975	1.102
donortypeDCD	1.414	1.299	1.539	albumin2<2	1.090	1.018	1.166
donorraceBlack	1.012	0.963	1.064	diabY	1.177	1.128	1.228
donorraceOther	1.088	0.984	1.202	hcvY	1.115	1.060	1.173
donorage	1.009	1.007	1.010	hccY	1.172	1.112	1.235
don_aboGroup AB	0.942	0.791	1.123	maligY	1.035	0.955	1.122
don_aboGroup B	0.889	0.771	1.025	recipientage	1.006	1.004	1.008
don_aboGroup O	0.911	0.814	1.019	splitsplit	1.151	0.977	1.355
don_hgt_cm	0.999	0.992	1.005	lifesupportLifesupport	1.287	1.188	1.393
don_wgt_kg	0.997	0.990	1.004	hbvY	1.040	0.993	1.090
don_bmi	1.007	0.987	1.027	coldischemiatime	1.015	1.010	1.020
don_log_sgot	1.034	1.001	1.068	shareRegional share	1.030	0.987	1.075
donorcodCNS Tumor	1.116	0.889	1.400	shareNational share	1.134	1.048	1.228
donorcodCVA/Stroke	1.049	0.993	1.108	recipientsexMale	1.037	0.984	1.092
donorcodHead trauma	1.020	0.961	1.082	etiologyAcute	1.053	0.903	1.228
donorcodOther	1.082	0.951	1.230	etiologyAlcholic	1.060	0.927	1.212
don_creat	0.996	0.982	1.010	etiologyCholestatic	0.932	0.806	1.078
don_anti_cmvP	0.982	0.945	1.021	etiologyHBV	1.008	0.837	1.215
don_anti_hcvP	1.190	0.678	2.089	etiologyHCV	1.139	0.994	1.306
don_prerecov_diureticsY	0.939	0.905	0.974	etiologyMalignant	1.178	1.021	1.359
don_ddavpY	0.947	0.900	0.997	etiologyOther cirrhosis	1.033	0.902	1.183
don_insulinY	1.040	1.002	1.080	etiologyOther/unknown	0.965	0.835	1.115
don_inotrop_supportY	0.998	0.962	1.036	pretxstatusHOSPITALIZED	1.041	0.981	1.104
don_hist_cigarette_gt20_pkyrY	1.007	0.966	1.050	pretxstatusIC UNIT	1.063	0.981	1.152
don_hist_cocaineY	0.958	0.905	1.014	can_race_srtrBlack	1.195	1.129	1.265
don_hist_other_drugY	0.954	0.913	0.998	can_race_srtrOther	0.834	0.761	0.913
don_meet_cdc_high_riskY	0.958	0.901	1.019	can_ethnicity_srtrNLATIN	1.154	1.091	1.221
don_hist_diabYES	1.088	1.026	1.153	can_aboGroup AB	0.997	0.863	1.151
don_htnY	1.071	1.024	1.121	can_aboGroup B	1.138	0.992	1.306
don_hist_cancerY	1.083	0.990	1.186	can_aboGroup O	1.086	0.970	1.216
don_log_bun	1.004	0.966	1.043	rec_functn_statSome assistance	1.119	1.067	1.173
don_tot_bili	1.014	1.002	1.027	rec_functn_statTotal assistance	1.168	1.099	1.241
don_log_sgpt	0.979	0.949	1.010	rec_work_incomeY	0.861	0.811	0.914
don_protein_urineY	1.007	0.971	1.044	rec_postx_los	1.007	1.007	1.008
don_sodium	1.000	0.998	1.002	rec_bmi	1.014	0.991	1.038
don_inr	1.007	0.997	1.017	retransplantationRetransplantation	1.503	1.400	1.614
don_hematocrit	1.001	0.997	1.004	rec_cmv_statP	0.973	0.937	1.010
don_anti_hbcP	1.127	0.810	1.568	rec_hbv_surf_antigenP	0.898	0.810	0.995
don_prerecov_steroidsY	1.008	0.966	1.053	rec_immuno_maint_medsY	0.081	0.076	0.086
don_anti_convulsY	1.042	0.961	1.131	rec_acute_rej_episodeYes	1.109	1.034	1.189
don_anti_hypertenY	0.985	0.942	1.030	rec_tumorY	1.345	1.229	1.471
don_vasodilY	1.041	0.988	1.096	rec_warm_isch_tm	1.002	1.001	1.003
don_heparinY	1.071	0.995	1.153	rec_bacteria_peritY	1.037	0.973	1.105
don_arginineY	0.988	0.952	1.026	can_variceal_bleedingY	1.112	1.015	1.217
don_txfus_terminal_hosp_num1-5	1.010	0.968	1.054	rec_tipssY	1.058	0.995	1.124
don_txfus_terminal_hosp_num6-10	1.062	0.988	1.142	can_last_bili	1.000	0.998	1.002
don_txfus_terminal_hosp_num>10	0.954	0.875	1.039	can_last_inr	0.994	0.978	1.010
don_clinical_infectY	0.962	0.899	1.029	can_last_serum_creat	1.031	1.017	1.045
don_infect_bloodY	1.036	0.967	1.110	can_last_serum_sodium	1.006	1.002	1.009
don_infect_luY	0.912	0.855	0.972	can_educationMedium education	0.959	0.918	1.003
don_infect_urineY	0.989	0.928	1.054	can_educationHigher education	0.910	0.860	0.962
don_heavy_alcoholY	1.022	0.971	1.076	rec_hgt_cm	1.006	0.999	1.014
don_tattoosY	1.016	0.973	1.061	rec_wgt_kg	0.994	0.986	1.001
don_hla_typY	1.011	0.886	1.153	can_peptic_ulcerYes	1.107	1.010	1.212
don_hist_prev_miY	1.024	0.939	1.117	can_angina_cadYes	1.304	1.171	1.452
don_pulm_cathY	0.999	0.947	1.055	can_drug_treat_hypertenY	1.000	0.958	1.045
don_hcv_statPositive	0.882	0.502	1.550	can_cereb_vascY	1.149	0.951	1.388
don_hbc_statPositive	0.887	0.635	1.238	can_periph_vascY	1.085	0.924	1.274
don_prev_gastro_diseaseY	1.013	0.946	1.085	can_acpt_abo_incompY	0.950	0.826	1.093
encephY	1.016	0.976	1.058	can_acpt_extracorp_liY	0.689	0.610	0.777
portal_veinY	1.194	1.100	1.296	can_acpt_li_segY	0.984	0.935	1.035
portal_hyperten_bleedY	1.050	0.959	1.150	can_acpt_hbc_posY	0.952	0.914	0.991
prev_abdom_surgY	1.118	1.077	1.161	can_acpt_hcv_posY	1.183	1.132	1.236

Table B.3: Penalized Cox regression model with the LASSO method for OS using λ_{1se} at exact time points. Only the 39 non-zero coefficients are presented.

Variable	coef	exp(coef)
donortypeDCD	0.014	1.014
donorage	0.007	1.007
don_prerecov_diureticsY	-0.024	0.976
don_hist_other_drugY	-0.014	0.986
don_htnY	0.023	1.024
don_clinical_infectY	-0.010	0.990
don_infect_luY	-0.041	0.960
portal_veinY	0.043	1.044
portal_hyperten_bleedY	0.004	1.004
prev_abdom_surgY	0.078	1.081
last_dial_prior_weekY	0.038	1.038
albumin2less2	0.016	1.016
diabY	0.160	1.173
hcvY	0.160	1.173
hccY	0.044	1.045
recipientage	0.012	1.012
lifesupportLifesupport	0.225	1.252
coldischemiatime	0.001	1.001
shareNational_share	0.043	1.044
etiologyCholestatic	-0.102	0.903
etiologyHCV	0.032	1.032
etiologyMalignant	0.043	1.044
pretxstatusIC_UNIT	0.033	1.033
can_race_srtrBlack	0.122	1.130
can_race_srtrOther	-0.042	0.959
rec_functn_statTotal_assistance	0.083	1.087
rec_work_incomeY	-0.179	0.836
rec_postx_los	0.006	1.006
rec_bmi	0.001	1.001
retransplantationRetransplantation	0.378	1.460
rec_immuno_maint_medsY	-2.197	0.111
rec_tumorY	0.196	1.216
rec_warm_isch_tm	0.001	1.001
can_variceal_bleedingY	0.011	1.011
can_last_serum_creat	0.024	1.024
can_last_serum_sodium	0.001	1.001
can_angina_cadYes	0.171	1.186
can_acpt_extracorp_liY	-0.190	0.827
can_acpt_hcv_posY	0.112	1.119

Table B.4: Variables selected with LASSO from 250 repeats of cross-validation using λ_{min} for OS.

Variable	Times	Variable	Times
albumin2<2	250	etiologyCholestatic	250
ascitesY	250	etiologyHBV	250
can_aboGroup_O	250	etiologyHCV	250
can_acpt_abo_incompY	250	etiologyOther_unknown	250
can_acpt_extracorp_liY	250	hbvY	250
can_acpt_hbc_posY	250	hccY	250
can_acpt_li_segY	250	hcvY	250
can_drug_treat_hypertenY	250	last_dial_prior_weekY	250
can_educationMedium_education	250	lifesupportLifesupport	250
can_ethnicity_srtrNLATIN	250	maligY	250
can_last_inr	250	portal_hyperten_bleedY	250
can_last_serum_creat	250	portal_veinY	250
can_peptic_ulcerYes	250	pretxstatusHOSPITALIZED	250
can_periph_vascY	250	pretxstatusIC_UNIT	250
can_race_srtrBlack	250	prev_abdom_surgY	250
can_race_srtrOther	250	rec_acute_rej_episodeYes	250
can_variceal_bleedingY	250	rec_bacteria_peritY	250
diabY	250	rec_bmi	250
don_aboGroup_O	250	rec_cmv_statP	250
don_anti_cmvP	250	rec_functn_statSome_assistance	250
don_anti_hcvP	250	rec_functn_statTotal_assistance	250
don_ethnicity_srtrNLATIN	250	rec_hbv_surf_antigenP	250
don_genderM	250	rec_hgt_cm	250
don_hist_cancerY	250	rec_immuno_maint_medsY	250
don_hist_cocaineY	250	rec_postx_los	250
don_hist_diabYES	250	rec_tumorY	250
don_hla_typY	250	rec_wgt_kg	250
don_htnY	250	rec_work_incomeY	250
don_infect_bloodY	250	recipientsexMale	250
don_inotrop_supportY	250	retransplantationRetransplantation	250
don_log_bun	250	shareRegional_share	250
don_log_sgpt	250	splitsplit	250
don_meet_cdc_high_riskY	250	don_ddavpY	249
don_prev_gastro_diseaseY	250	don_hbc_statPositive	249
don_sodium	250	don_prerecov_diureticsY	249
don_txfus_terminal_hosp_num1to5	250	rec_warm_isch_tm	249
don_txfus_terminal_hosp_num6to10	250	don_log_sgot	193
don_txfus_terminal_hosp_num>10	250	can_last_serum_sodium	112
don_vasodilY	250	coldischemiatime	112
donorcodCNS_Tumor	250	recipientage	112
donorcodHead_trauma	250	don_bmi	44
donorraceBlack	250	don_prerecov_steroidsY	44
donorraceOther	250	don_protein_urineY	44
donortypeDCD	250	shareNational_share	44
encephY	250	can_aboGroup_B	10
etiologyAcute	250	don_heparinY	10
etiologyAlcholic	250	rec_tipssY	10

Table B.5: Penalized Cox regression model with the LASSO method for FFS using λ_{1se} at exact time points. The 32 non-zero coefficients are presented.

Variable	coef	exp(coef)
donortypeDCD	0.174	1.190
donorage	0.008	1.008
don_prerecov_diureticsY	-0.011	0.989
don_hist_other_drugY	-0.005	0.995
don_hist_diabYES	0.020	1.021
don_htnY	0.035	1.036
don_infect_luY	-0.062	0.940
portal_veinY	0.027	1.027
prev_abdom_surgY	0.063	1.065
diabY	0.113	1.119
hcvY	0.148	1.160
hccY	0.036	1.037
recipientage	0.005	1.005
lifesupportLifesupport	0.211	1.235
coldischemiatime	0.007	1.007
shareNational_share	0.005	1.005
etiologyCholestatic	-0.019	0.982
etiologyHCV	0.006	1.006
etiologyMalignant	0.003	1.003
pretxstatusIC_UNIT	0.002	1.002
can_race_srtrBlack	0.109	1.115
can_race_srtrOther	-0.001	0.999
rec_functn_statTotal_assistance	0.043	1.044
rec_work_incomeY	-0.113	0.893
rec_postx_los	0.007	1.007
retransplantationRetransplantation	0.309	1.362
rec_immuno_maint_medsY	-2.363	0.094
rec_tumorY	0.129	1.138
can_last_serum_creat	0.013	1.013
can_angina_cadYes	0.107	1.113
can_acpt_extracorp_liY	-0.161	0.851
can_acpt_hcv_posY	0.126	1.134

Table B.6: Variables selected with LASSO from 250 repeats of cross-validation using λ_{min} for FFS.

Variable	Times	Variable	Times
albumin2<2	250	etiologyOther_cirrhosis	250
ascitesY	250	etiologyOther_unknown	250
can_aboGroup_O	250	hbvY	250
can_acpt_abo_incompY	250	hccY	250
can_acpt_hbc_posY	250	hcvY	250
can_acpt_li_segY	250	last_dial_prior_weekY	250
can_cereb_vascY	250	lifesupportLifesupport	250
can_drug_treat_hypertenY	250	maligY	250
can_educationMedium_education	250	portal_hyperten_bleedY	250
can_ethnicity_srtrNLATIN	250	portal_veinY	250
can_last_inr	250	pretxstatusHOSPITALIZED	250
can_last_serum_creat	250	pretxstatusIC_UNIT	250
can_peptic_ulcerYes	250	prev_abdom_surgY	250
can_race_srtrBlack	250	rec_acute_rej_episodeYes	250
can_race_srtrOther	250	rec_bacteria_peritY	250
can_variceal_bleedingY	250	rec_bmi	250
coldischemiatime	250	rec_cmv_statP	250
diabY	250	rec_functn_statSome_assistance	250
don_aboGroup_O	250	rec_functn_statTotal_assistance	250
don_anti_hcvP	250	rec_hbv_surf_antigenP	250
don_anti_hypertenY	250	rec_immuno_maint_medsY	250
don_ddavpY	250	rec_postx_los	250
don_ethnicity_srtrNLATIN	250	rec_tumorY	250
don_genderM	250	rec_warm_isch_tm	250
don_hgt_cm	250	rec_wgt_kg	250
don_hist_cigarette_gt20_pkyrY	250	rec_work_incomeY	250
don_hist_cocaineY	250	recipientage	250
don_hist_diabYES	250	retransplantationRetransplantation	250
don_hist_other_drugY	250	shareNational_share	250
don_hla_typY	250	shareRegional_share	250
don_htnY	250	splitsplit	250
don_infect_bloodY	250	can_last_serum_sodium	249
don_log_bun	250	don_anti_cmvP	249
don_meet_cdc_high_riskY	250	don_bmi	249
don_prerecov_diureticsY	250	etiologyCholestatic	249
don_prerecov_steroidsY	250	can_aboGroup_AB	234
don_prev_gastro_diseaseY	250	can_educationHigher_education	234
don_sodium	250	don_log_sgot	234
don_txfus_terminal_hosp_num1to5	250	donorcodHead_trauma	234
don_txfus_terminal_hosp_num6to10	250	don_clinical_infectY	187
don_txfus_terminal_hosp_num>10	250	don_infect_urineY	187
don_vasodilY	250	can_last_bili	98
donorcodCNS_Tumor	250	don_tot_bili	98
donorraceBlack	250	etiologyAcute	98
donorraceOther	250	don_inotrop_supportY	42
encephY	250	don_creat	15
etiologyAlcholic	250	can_acpt_extracorp_liY	2
etiologyHBV	250	can_periph_vascY	2
etiologyHCV	250	don_anti_convulsY	2

B.2 Neural networks for survival analysis

B.2.1 Overall survival

Table B.7: Best 2 combinations per activation function in terms of accuracy for 1 hidden layer and overall survival (experiments part 1).

	Activation function of input-hidden layer nodes					
	1) Sigmoid	2) Sigmoid	1) Tanh	2) Tanh	1) ReLU	2) ReLU
Total weights	13101	6551	5241	6551	9171	13101
Node size	100	50	40	50	70	100
Dropout	0.2	0.2	0.2	0.2	0.3	0.4
Learning rate	0.1	0.1	0.1	0.1	0.1	0.1
Momentum	0.9	0.9	0.9	0.9	0.9	0.9
Weak class	1	1	1	1	1	1
Cross entropy	0.208	0.208	0.208	0.208	0.204	0.204
Accuracy	0.704	0.706	0.706	0.708	0.709	0.709
Sensitivity	0.095	0.086	0.088	0.091	0.123	0.127
Specificity	0.954	0.960	0.960	0.961	0.950	0.947
Precision	0.461	0.478	0.482	0.515	0.503	0.502
F1 score	0.157	0.145	0.146	0.153	0.197	0.202
IBS	0.182	0.185	0.195	0.195	0.188	0.185

Table B.8: Best 2 combinations per activation function in terms of integrated Brier score for neural network with 1 hidden layer: overall survival (experiments part 1).

	Activation function of input-hidden layer nodes					
	1) Sigmoid	2) Sigmoid	1) Tanh	2) Tanh	1) ReLU	2) ReLU
Total weights	1311	1311	1311	1311	1311	2621
Node size	10	10	10	10	10	20
Dropout	0.4	0.3	0.3	0.3	0.4	0.4
Learning rate	0.01	0.01	0.1	0.01	0.01	0.01
Momentum	0.8	0.8	0.9	0.8	0.8	0.8
Weak class	2	2	2	2	2	2
Cross entropy	0.368	0.367	0.346	0.356	0.362	0.359
Accuracy	0.472	0.482	0.533	0.540	0.514	0.511
Sensitivity	0.207	0.202	0.311	0.221	0.208	0.230
Specificity	0.581	0.597	0.624	0.670	0.639	0.627
Precision	0.169	0.171	0.255	0.216	0.192	0.202
F1 score	0.186	0.185	0.280	0.219	0.200	0.215
IBS	0.127	0.128	0.131	0.134	0.132	0.132

Table B.9 shows that even accuracy of 0.719 was achieved with a combination with activation the tanh function. In table B.10 it can be observed that node sizes vary from 75 to 125 and better performance in terms of accuracy is achieved for the smallest values of IBS compared to that of table B.9.

Table B.9: Best 2 combinations per activation function in terms of accuracy for neural network with 1 hidden layer: overall survival (experiments part 2).

	Activation function of input-hidden layer nodes					
	1) Sigmoid	2) Sigmoid	1) Tanh	2) Tanh	1) ReLU	2) ReLU
Total weights	12446	10481	17031	9826	14411	15721
Node size	95	80	130	75	110	120
Dropout	0.3	0.3	0.3	0.2	0.2	0.1
Learning rate	0.2	0.2	0.2	0.2	0.2	0.1
Momentum	0.9	0.9	0.9	0.9	0.9	0.8
Cross entropy	0.207	0.206	0.211	0.208	0.202	0.203
Accuracy	0.711	0.713	0.714	0.719	0.711	0.712
Sensitivity	0.108	0.099	0.075	0.111	0.139	0.132
Specificity	0.958	0.965	0.976	0.969	0.946	0.950
Precision	0.519	0.541	0.577	0.600	0.524	0.534
F1 score	0.179	0.168	0.131	0.186	0.218	0.210
IBS	0.184	0.188	0.216	0.210	0.190	0.193

Table B.10: Best 2 combinations per activation function in terms of integrated Brier score for neural network with 1 hidden layer: overall survival (experiments part 2).

	Activation function of input-hidden layer nodes					
	1) Sigmoid	2) Sigmoid	1) Tanh	2) Tanh	1) ReLU	2) ReLU
Total weights	13101	16376	12446	13101	13101	9826
Node size	100	125	95	100	100	75
Dropout	0.3	0.3	0.1	0.2	0.3	0.1
Learning rate	0.1	0.2	0.2	0.2	0.1	0.2
Momentum	0.8	0.9	0.9	0.9	0.8	0.9
Cross entropy	0.214	0.207	0.208	0.209	0.205	0.201
Accuracy	0.680	0.694	0.647	0.658	0.691	0.689
Sensitivity	0.055	0.133	0.223	0.165	0.141	0.171
Specificity	0.937	0.925	0.821	0.860	0.917	0.901
Precision	0.265	0.422	0.386	0.379	0.415	0.422
F1 score	0.090	0.202	0.257	0.215	0.210	0.241
IBS	0.169	0.169	0.164	0.165	0.171	0.171

B.2.2 Failure-free survival

Table B.11: Best 2 combinations per activation function in terms of cross-entropy for neural networks with 1 hidden layer: failure-free survival (experiments part 1).

	Activation function of input-hidden layer nodes					
	1) Sigmoid	2) Sigmoid	1) Tanh	2) Tanh	1) ReLU	2) ReLU
Total weights	3931	5241	7861	9171	10481	11791
Node size	30	40	60	70	80	90
Dropout	0.2	0.2	0.2	0.2	0.2	0.2
Learning rate	0.1	0.1	0.1	0.1	0.1	0.1
Momentum	0.9	0.9	0.9	0.9	0.9	0.9
Weak class	1	1	1	1	1	1
Cross entropy	0.222	0.222	0.221	0.221	0.216	0.216
Accuracy	0.679	0.683	0.678	0.654	0.685	0.669
Sensitivity	0.119	0.112	0.139	0.154	0.156	0.170
Specificity	0.935	0.945	0.925	0.882	0.927	0.897
Precision	0.464	0.485	0.475	0.429	0.495	0.433
F1 score	0.188	0.180	0.210	0.215	0.236	0.243
IBS	0.183	0.188	0.188	0.175	0.191	0.179

Table B.12: Best 2 combinations per activation function in terms of cross-entropy for neural networks with 1 hidden layer: failure-free survival (experiments part 2).

	Activation function of input-hidden layer nodes					
	1) Sigmoid	**2) Sigmoid**	1) Tanh	2) Tanh	1) ReLU	2) ReLU
Total weights	9171	**9826**	9171	13101	16376	17031
Node size	70	**75**	70	100	125	130
Dropout	0.1	**0.1**	0.1	0.1	0.1	0.1
Learning rate	0.2	**0.2**	0.1	0.1	0.2	0.1
Momentum	0.9	**0.9**	0.8	0.8	0.9	0.9
Cross entropy	0.218	**0.218**	0.220	0.220	0.212	0.213
Accuracy	0.670	**0.688**	0.685	0.676	0.687	0.680
Sensitivity	0.164	**0.146**	0.134	0.158	0.179	0.172
Specificity	0.902	**0.937**	0.937	0.913	0.919	0.912
Precision	0.436	**0.513**	0.508	0.463	0.509	0.475
F1 score	0.237	**0.227**	0.211	0.233	0.263	0.251
IBS	0.175	**0.188**	0.193	0.182	0.195	0.189

Table B.13: Best 2 combinations per activation function in terms of cross-entropy for neural networks with 2 hidden layers: failure-free survival (experiments part 1).

	Activation function of input-hidden 1, hidden 1-hidden 2 layers					
	1) Sigmoid	2) Sigmoid	1) Tanh	2) Tanh	1) ReLU	2) ReLU
Total weights	9101	11521	19981	9101	19981	23201
Node size (each hidden)	50	60	90	50	90	100
Dropout	0.2	0.2	0.2	0.2	0.2	0.2
Learning rate	0.1	0.1	0.1	0.1	0.1	0.1
Momentum	0.9	0.9	0.9	0.9	0.9	0.9
Weak class	1	1	1	1	1	1
Cross entropy	0.238	0.238	0.223	0.224	0.223	0.223
Accuracy	0.674	0.672	0.677	0.680	0.659	0.657
Sensitivity	0.019	0.021	0.109	0.100	0.142	0.142
Specificity	0.973	0.970	0.937	0.946	0.895	0.893
Precision	0.252	0.252	0.443	0.466	0.386	0.391
F1 score	0.035	0.038	0.174	0.163	0.207	0.206
IBS	0.178	0.177	0.187	0.190	0.174	0.176

Table B.14: Best 2 combinations per activation function in terms of cross-entropy for neural networks with 2 hidden layers: failure-free survival (experiments part 2).

	Activation function of input-hidden 1, hidden 1-hidden 2 layers					
	1) Sigmoid	2) Sigmoid	**1) Tanh**	2) Tanh	1) ReLU	2) ReLU
Total weights	14141	16961	**23201**	14141	26621	30241
Node size (each hidden)	70	80	**100**	70	110	120
Dropout	0.1	0.1	**0.1**	0.1	0.1	0.1
Learning rate	0.2	0.2	**0.2**	0.2	0.2	0.2
Momentum	0.9	0.9	**0.9**	0.9	0.9	0.9
Weak class	1	1	**1**	1	1	1
Cross entropy	0.231	0.231	**0.219**	0.220	0.216	0.216
Accuracy	0.662	0.667	**0.691**	0.693	0.681	0.686
Sensitivity	0.055	0.051	**0.150**	0.134	0.167	0.154
Specificity	0.940	0.949	**0.938**	0.949	0.916	0.929
Precision	0.304	0.317	**0.542**	0.550	0.487	0.505
F1 score	0.092	0.088	**0.232**	0.215	0.247	0.234
IBS	0.182	0.184	**0.192**	0.196	0.187	0.191

B.3 Comparisons at exact time points

B.3.1 Comparisons: Overall survival

Table B.15: Hazard ratios of the most influential variable levels for the Cox models at exact time points (OS).

	Cox all variables	Cox backward	Cox LASSO
retransplantationRetransplantation	1.573	1.550	1.460
rec_tumorY	1.391	1.410	1.216
can_angina_cadYes	1.337	1.360	1.186
lifesupportLifesupport	1.302	1.390	1.252
diabY	1.233	1.230	1.173
can_race_srtrBlack	1.195	1.180	1.130
rec_work_incomeY	0.814	0.800	0.836
can_race_srtrOther	0.809	0.810	0.959
can_acpt_extracorp_liY	0.700	0.680	0.827
rec_immuno_maint_medsY	0.100	0.100	0.111

B.3.2 Comparisons: Failure-free survival

Table B.16: Hazard ratios of the most influential variable levels for the Cox models at exact time points (FFS).

	Cox all variables	Cox backward	Cox LASSO
retransplantationRetransplantation	1.503	1.520	1.362
donortypeDCD	1.414	1.390	1.190
rec_tumorY	1.345	1.360	1.138
can_angina_cadYes	1.304	1.330	1.113
can_race_srtrBlack	1.195	1.200	1.115
can_acpt_hcv_posY	1.183	1.170	1.134
rec_work_incomeY	0.861	0.850	0.893
can_race_srtrOther	0.834	0.840	0.999
can_acpt_extracorp_liY	0.689	0.680	0.851
rec_immuno_maint_medsY	0.081	0.080	0.094

Figure B.1: Prediction error curves for all models using exact time points (FFS). A: apparent error, B: generalization error (test error), C: OOB train error, D: linear combination of 0.328*(apparent error) + 0.632*(OOB train error). For graphs C and D bootstrapped cross-validation has been performed 100 times.

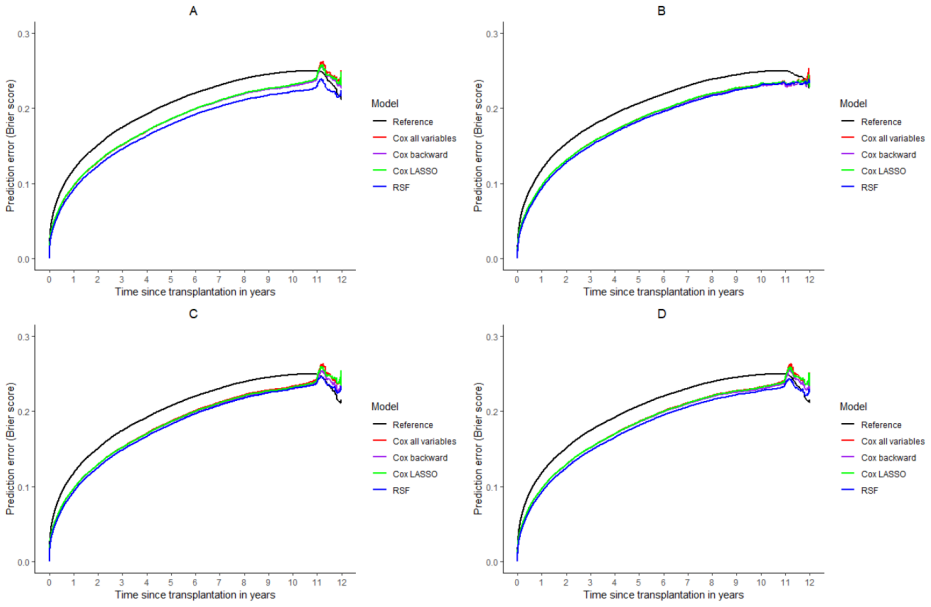


Figure B.2: Time-dependent C-index for all models using exact time points (FFS). A: apparent error, B: generalization error (test error), C: OOB train error, D: linear combination of 0.328*(apparent error) + 0.632*(OOB train error). For graphs C and D bootstrapped cross-validation has been performed 100 times.

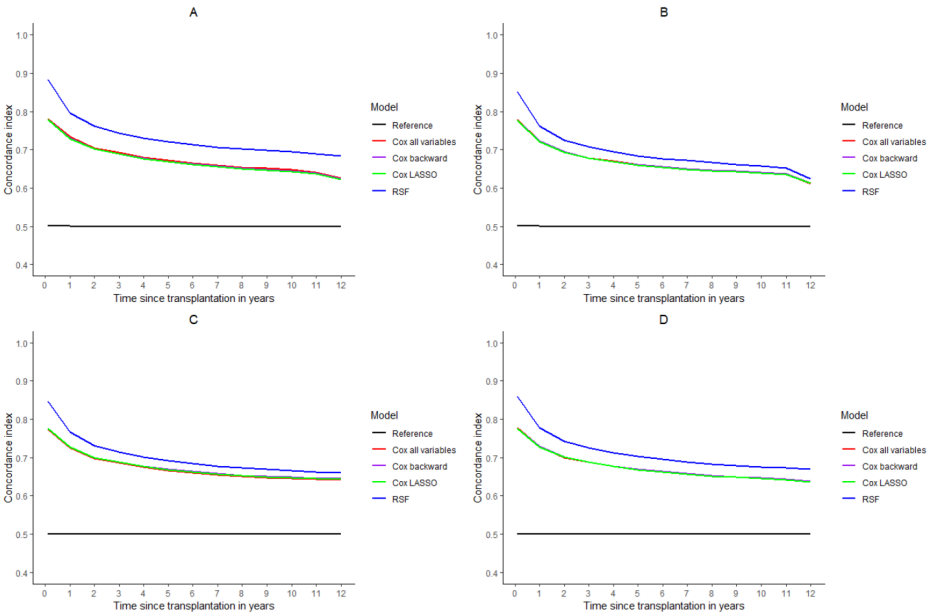


Figure B.3: Failure-free survival of 6 new hypothetical patients for variables immuno-maintenance suppression and length of stay. Applied values for immuno meds are No (N) and Yes (Y). For length of stay values 5, 10 and 33 correspond to the 10%, 50% and 90% quantiles on the training data.

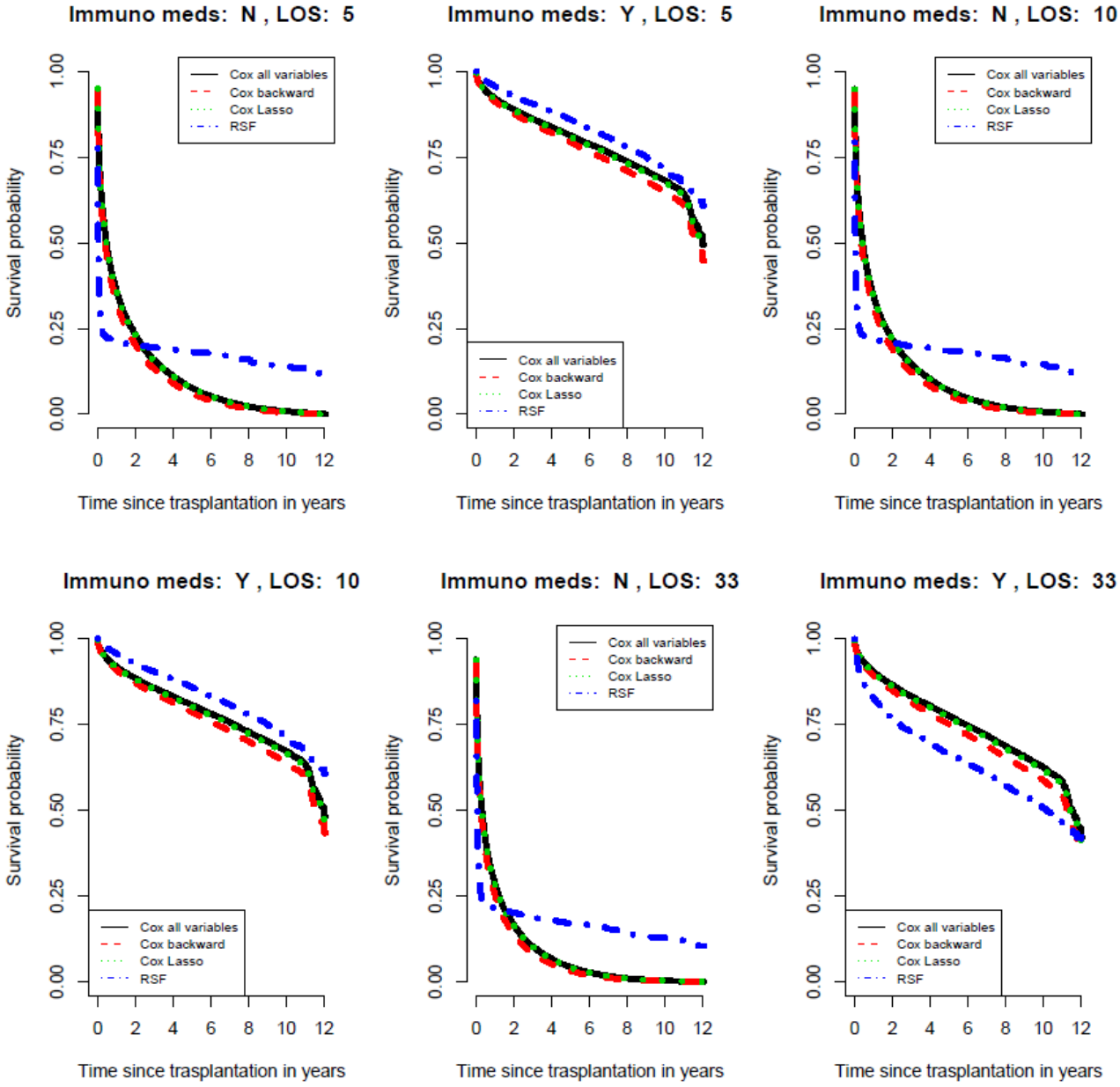
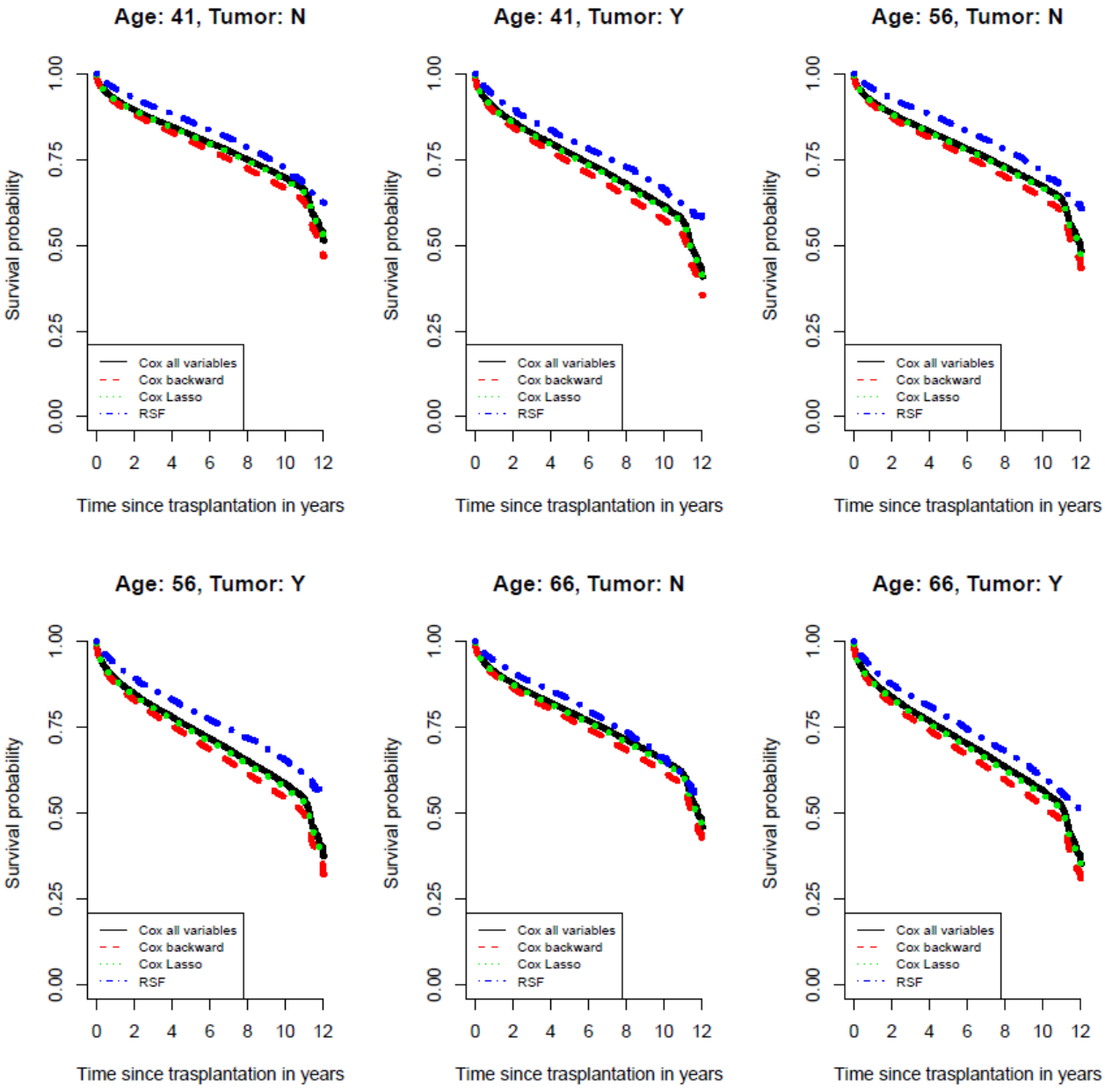


Figure B.4: Failure-free survival of 6 new hypothetical patients for variables recipient age and incidental tumor. Values for tumor are No (N) or Yes (Y). For recipient age values 41, 56 and 66 years correspond to the 10%, 50% and 90% quantiles on the training data.



B.4 Comparisons at distinct time points

B.4.1 Comparisons: Overall survival

Table B.17: Variables with highest importance using random survival forests at discrete time points (OS).

Variable	VIMP	Variable	VIMP
rec_postx_los	0.0518	can_race_srtr	0.0011
rec_immuno_maint_meds	0.0282	lifesupport	0.0011
donorage	0.0057	hcc	0.0010
hcv	0.0054	pretxstatus	0.0009
recipientage	0.0050	rec_tumor	0.0008
can_acpt_hcv_pos	0.0049	rec_work_income	0.0007
etiology	0.0035	can_last_bili	0.0007
diab	0.0016	can_last_inr	0.0006
can_last_serum_creat	0.0016	rec_functn_stat	0.0005
retransplantation	0.0016	don_htn	0.0005

Table B.18: Most important variables using minimal depth on grown forest at discrete time points (OS).

Variable	Depth	Variable	Depth
rec_immuno_maint_meds	1.5433	don_tot_bili	8.3500
rec_postx_los	1.7700	can_acpt_hcv_pos	8.3500
recipientage	4.3533	can_last_inr	8.5767
donorage	4.3633	don_inr	9.0100
can_last_serum_creat	6.6567	don_sodium	9.0400
etiology	7.1533	rec_warm_isch_tm	9.1767
can_last_serum_sodium	7.6367	don_hgt_cm	9.1833
rec_hgt_cm	8.1400	coldischemiatime	9.2000
retransplantation	8.2367	don_hematocrit	9.4000
can_last_bili	8.3433	hcv	9.4667

Table B.19: Cox proportional hazards model with all prognostic factors: Hazard ratios and lower and upper 95% confidence intervals at discrete time points (Overall Survival). The 5 most detrimental factor levels are colored in red and the 5 most beneficial are colored in green.

Variable	exp(coef)	lower .95	upper .95	Variable	exp(coef)	lower .95	upper .95
don_genderM	1.056	1.004	1.110	ascitesY	1.036	0.992	1.081
don_ethnicity_srtrNLATIN	0.953	0.900	1.009	last_dial_prior_weekY	1.066	1.001	1.136
donortypeDCD	1.169	1.068	1.280	albumin2<2	1.126	1.050	1.208
donorraceBlack	0.980	0.930	1.033	diabY	1.232	1.180	1.288
donorraceOther	1.115	1.005	1.236	hcvY	1.120	1.062	1.180
donorage	1.007	1.005	1.009	hccY	1.151	1.089	1.215
don_aboGroup_AB	0.939	0.782	1.128	maligY	1.078	0.992	1.170
don_aboGroup_B	0.910	0.786	1.053	recipientage	1.013	1.011	1.015
don_aboGroup_O	0.959	0.854	1.077	splitsplit	1.097	0.925	1.301
don_hgt_cm	0.999	0.992	1.006	lifesupportLifesupport	1.226	1.130	1.331
don_wgt_kg	0.999	0.992	1.006	hbvY	1.052	1.003	1.104
don_bmi	1.004	0.983	1.025	coldischemiatime	1.008	1.002	1.013
don_log_sgot	1.022	0.989	1.057	shareRegional_share	1.010	0.966	1.056
donorcodCNS_Tumor	1.113	0.879	1.409	shareNational_share	1.136	1.046	1.234
donorcodCVA_Stroke	1.055	0.997	1.117	recipientsexMale	1.013	0.959	1.069
donorcodHead_trauma	1.007	0.947	1.070	etiologyAcute	0.977	0.831	1.149
donorcodOther	1.099	0.963	1.255	etiologyAlcholic	1.075	0.934	1.237
don_creat	0.999	0.985	1.014	etiologyCholestatic	0.852	0.731	0.994
don_anti_cmvP	1.006	0.966	1.047	etiologyHBV	0.957	0.787	1.164
don_anti_hcvP	1.228	0.680	2.217	etiologyHCV	1.140	0.987	1.316
don_prerecov_diureticsY	0.933	0.898	0.969	etiologyMalignant	1.198	1.031	1.391
don_ddavpY	0.975	0.924	1.028	etiologyOther_cirrhosis	1.020	0.885	1.175
don_insulinY	1.005	0.966	1.045	etiologyOther_unknown	1.011	0.868	1.176
don_inotrop_supportY	0.997	0.959	1.036	pretxstatusHOSPITALIZED	1.084	1.020	1.151
don_hist_cigarette_gt20_pkyrY	1.031	0.987	1.077	pretxstatusIC_UNIT	1.121	1.032	1.218
don_hist_cocaineY	0.993	0.936	1.053	can_race_srtrBlack	1.203	1.134	1.277
don_hist_other_drugY	0.934	0.892	0.977	can_race_srtrOther	0.802	0.730	0.882
don_meet_cdc_high_riskY	0.978	0.918	1.043	can_ethnicity_srtrNLATIN	1.141	1.076	1.210
don_hist_diabYES	1.054	0.992	1.120	can_aboGroup_AB	0.981	0.844	1.140
don_htnY	1.043	0.995	1.094	can_aboGroup_B	1.086	0.943	1.251
don_hist_cancerY	1.095	0.997	1.203	can_aboGroup_O	1.025	0.912	1.151
don_log_bun	1.008	0.969	1.048	rec_functn_statSome_assistance	1.074	1.023	1.128
don_tot_bili	1.013	1.000	1.027	rec_functn_statTotal_assistance	1.115	1.047	1.187
don_log_sgpt	0.988	0.956	1.021	rec_work_incomeY	0.813	0.764	0.866
don_protein_urineY	1.012	0.974	1.051	rec_postx_los	1.007	1.006	1.007
don_sodium	1.000	0.998	1.002	rec_bmi	1.007	0.983	1.032
don_inr	1.007	0.996	1.017	retransplantationRetransplantation	1.517	1.410	1.632
don_hematocrit	1.000	0.997	1.004	rec_cmv_statP	0.971	0.934	1.009
don_anti_hbcP	1.073	0.758	1.517	rec_hbv_surf_antigenP	0.916	0.824	1.019
don_prerecov_steroidsY	1.004	0.961	1.050	rec_immuno_maint_medsY	0.140	0.131	0.149
don_anti_convulsY	1.029	0.945	1.119	rec_acute_rej_episodeYes	1.105	1.027	1.189
don_anti_hypertenY	0.993	0.948	1.039	rec_tumorY	1.375	1.254	1.508
don_vasodilY	1.012	0.959	1.067	rec_warm_isch_tm	1.002	1.001	1.003
don_heparinY	1.041	0.965	1.123	rec_bacteria_peritY	1.024	0.959	1.094
don_arginineY	0.988	0.950	1.027	can_variceal_bleedingY	1.127	1.028	1.236
don_txfus_terminal_hosp_num1to5	1.004	0.961	1.050	rec_tipssY	1.066	1.001	1.136
don_txfus_terminal_hosp_num6to10	1.042	0.967	1.124	can_last_bili	1.000	0.998	1.002
don_txfus_terminal_hosp_num>10	0.960	0.878	1.049	can_last_inr	0.994	0.978	1.011
don_clinical_infectY	0.936	0.873	1.004	can_last_serum_creat	1.038	1.024	1.052
don_infect_bloodY	1.031	0.959	1.108	can_last_serum_sodium	1.006	1.003	1.010
don_infect_luY	0.941	0.880	1.006	can_educationMedium_education	0.957	0.914	1.001
don_infect_urineY	1.004	0.939	1.072	can_educationHigher_education	0.914	0.862	0.969
don_heavy_alcoholY	1.001	0.949	1.056	rec_hgt_cm	1.004	0.996	1.012
don_tattoosY	0.997	0.953	1.043	rec_wgt_kg	0.995	0.987	1.003
don_hla_typY	1.038	0.903	1.192	can_peptic_ulcerYes	1.087	0.989	1.195
don_hist_prev_miY	1.056	0.965	1.155	can_angina_cadYes	1.347	1.208	1.503
don_pulm_cathY	1.018	0.963	1.077	can_drug_treat_hypertenY	0.998	0.954	1.043
don_hcv_statPositive	0.857	0.474	1.550	can_cereb_vascY	1.150	0.948	1.394
don_hbc_statPositive	0.939	0.661	1.334	can_periph_vascY	1.034	0.876	1.221
don_prev_gastro_diseaseY	1.018	0.949	1.093	can_acpt_abo_incompY	0.908	0.782	1.053
encephY	1.022	0.980	1.066	can_acpt_extracorp_liY	0.774	0.683	0.876
portal_veinY	1.170	1.074	1.274	can_acpt_li_segY	0.948	0.899	0.999
portal_hyperten_bleedY	1.100	1.002	1.207	can_acpt_hbc_posY	0.955	0.916	0.996
prev_abdom_surgY	1.109	1.067	1.154	can_acpt_hcv_posY	1.188	1.136	1.243

Table B.20: Cox backward model at distinct time points for overall survival. 29 variables were selected.

	exp(coef)	lower .95	upper .95
donortypeDCD	1.154	1.065	1.251
donorage	1.008	1.007	1.009
don_prerecov_diureticsY	0.940	0.906	0.975
don_hist_other_drugY	0.930	0.893	0.968
don_infect_luY	0.901	0.867	0.936
portal_veinY	1.158	1.064	1.261
prev_abdom_surgY	1.102	1.060	1.144
albumin2less2	1.122	1.047	1.202
diabY	1.227	1.176	1.281
hcvY	1.142	1.085	1.201
hccY	1.139	1.082	1.198
recipientage	1.012	1.010	1.014
lifesupportLifesupport	1.224	1.130	1.325
coldischemiatime	1.009	1.004	1.015
etiologyAcute	0.962	0.819	1.129
etiologyAlcholic	1.099	0.955	1.264
etiologyCholestatic	0.859	0.737	1.001
etiologyHBV	0.926	0.771	1.113
etiologyHCV	1.141	0.988	1.316
etiologyMalignant	1.188	1.024	1.380
etiologyOther_cirrhosis	1.017	0.883	1.171
etiologyOther_unknown	1.016	0.874	1.181
pretxstatusHOSPITALIZED	1.112	1.058	1.169
pretxstatusIC_UNIT	1.175	1.095	1.260
can_race_srtrBlack	1.195	1.128	1.265
can_race_srtrOther	0.804	0.734	0.881
can_ethnicity_srtrNLATIN	1.122	1.062	1.186
rec_work_incomeY	0.796	0.750	0.845
rec_postx_los	1.007	1.006	1.007
retransplantationRetransplantation	1.436	1.337	1.541
rec_immuno_maint_medsY	0.191	0.179	0.204
rec_tumorY	1.351	1.233	1.480
rec_warm_isch_tm	1.002	1.001	1.003
can_last_serum_creat	1.041	1.028	1.054
rec_wgt_kg	0.998	0.997	0.999
can_angina_cadYes	1.357	1.219	1.511
can_acpt_extracorp_liY	0.815	0.721	0.921
can_acpt_hcv_posY	1.159	1.112	1.208

Table B.21: Penalized Cox regression model with the LASSO method at distinct time points for OS using λ_{1se}. Only the 37 non-zero coefficients are presented.

Variable	coef	exp(coef)
donortypeDCD	0.017	1.017
donorage	0.006	1.006
don_prerecov_diureticsY	-0.022	0.978
don_hist_other_drugY	-0.019	0.981
don_hist_diabYES	0.001	1.001
don_htnY	0.020	1.020
don_clinical_infectY	-0.014	0.986
don_infect_luY	-0.044	0.957
portal_veinY	0.014	1.014
prev_abdom_surgY	0.057	1.059
last_dial_prior_weekY	0.031	1.031
albumin2less2	0.018	1.018
diabY	0.151	1.163
hcvY	0.143	1.154
hccY	0.030	1.030
recipientage	0.010	1.010
lifesupportLifesupport	0.140	1.150
coldischemiatime	0.001	1.001
shareNational_share	0.011	1.012
etiologyCholestatic	-0.096	0.908
etiologyHCV	0.020	1.020
etiologyMalignant	0.035	1.036
pretxstatusIC_UNIT	0.027	1.027
can_race_srtrBlack	0.123	1.131
can_race_srtrOther	-0.051	0.951
rec_functn_statTotal_assistance	0.038	1.039
rec_work_incomeY	-0.162	0.850
rec_postx_los	0.006	1.006
rec_bmi	-0.001	0.999
retransplantationRetransplantation	0.305	1.357
rec_immuno_maint_medsY	-1.606	0.201
rec_tumorY	0.162	1.176
can_last_serum_creat	0.022	1.022
can_last_serum_sodium	0.001	1.001
can_angina_cadYes	0.165	1.179
can_acpt_extracorp_liY	-0.024	0.976
can_acpt_hcv_posY	0.137	1.146

B.4.2 Comparisons: Failure-free survival

Figure B.5: Failure-free survival for 6 new hypothetical patients and variables immuno-maintenance suppression and length of stay. Applied values for immuno meds are No (N) or Yes (Y). For length of stay applied values 5, 10 and 33 correspond to the 10%, 50% and 90% quantiles on the training data.

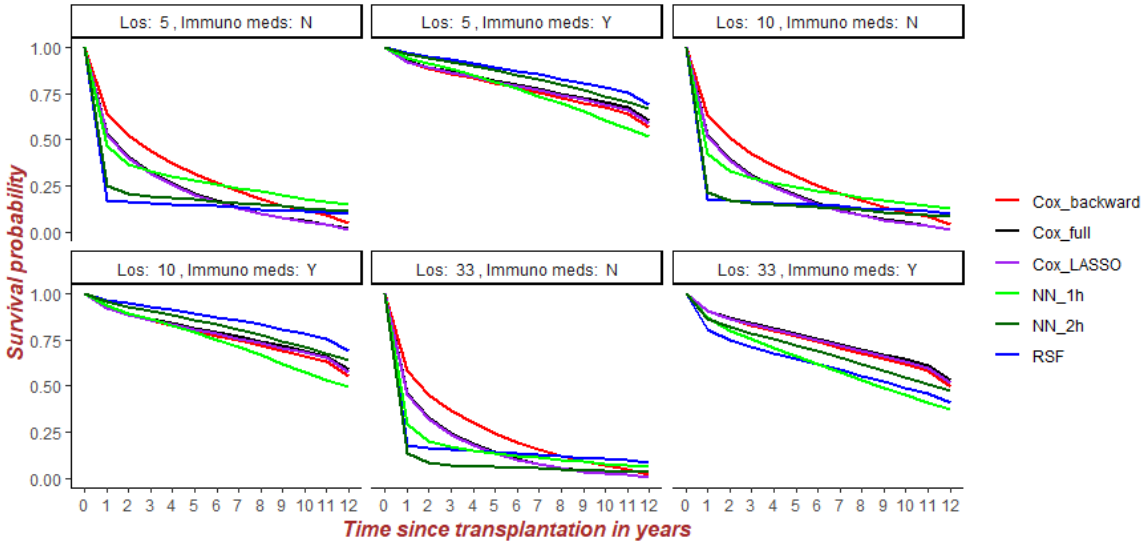


Figure B.6: Failure-free survival for 6 new hypothetical patients and variables recipient age and incidental tumor. Applied values for tumor are No (N) or Yes (Y). For recipient age applied values 41, 56 and 66 years correspond to the 10%, 50% and 90% quantiles on the training data.

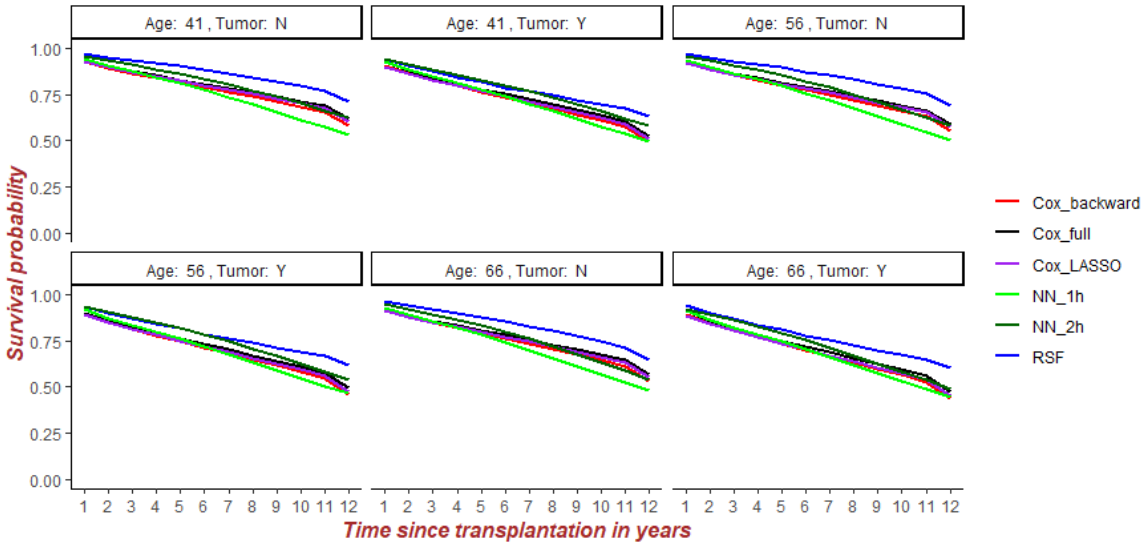


Table B.22: Variables with highest importance using random survival forests at discrete time points (FFS).

Variable	VIMP	Variable	VIMP
rec_postx_los	0.0653	donortype	0.0009
rec_immuno_maint_meds	0.0265	don_htn	0.0009
donorage	0.0075	lifesupport	0.0009
can_acpt_hcv_pos	0.0055	can_last_serum_creat	0.0009
hcv	0.0046	diab	0.0009
recipientage	0.0023	pretxstatus	0.0009
etiology	0.0020	rec_tumor	0.0007
retransplantation	0.0017	can_last_bili	0.0006
can_race_srtr	0.0014	rec_functn_stat	0.0005
hcc	0.0011	don_infect_lu	0.0004

Table B.23: Most important variables using minimal depth on grown forest at discrete time points (FFS).

Variable	Depth	Variable	Depth
rec_immuno_maint_meds	0.9167	rec_hgt_cm	8.7933
rec_postx_los	1.3900	don_tot_bili	8.8467
donorage	3.7900	retransplantation	8.9800
recipientage	6.3067	don_hgt_cm	9.0567
can_acpt_hcv_pos	7.2433	don_sodium	9.2500
can_last_serum_creat	8.1333	don_inr	9.3033
etiology	8.3233	can_race_srtr	9.3100
can_last_inr	8.4600	coldischemiatime	9.4833
can_last_serum_sodium	8.7433	hcv	9.6000
can_last_bili	8.7533	rec_warm_isch_tm	9.6900

Table B.24: Cox proportional hazards model with all prognostic factors: Hazard ratios and lower and upper 95% confidence intervals at discrete time points (Failure-free Survival). The 5 most detrimental factor levels are colored in red and the 4 most beneficial are colored in green.

Variable	exp(coef)	lower .95	upper .95	Variable	exp(coef)	lower .95	upper .95
don_genderM	1.057	1.007	1.109	ascitesY	1.033	0.991	1.076
don_ethnicity_srtrNLATIN	0.948	0.897	1.002	last_dial_prior_weekY	1.048	0.986	1.115
donortypeDCD	1.393	1.279	1.516	albumin2<2	1.097	1.025	1.175
donorraceBlack	1.013	0.963	1.065	diabY	1.183	1.134	1.234
donorraceOther	1.088	0.984	1.202	hcvY	1.108	1.053	1.166
donorage	1.009	1.007	1.010	hccY	1.148	1.090	1.211
don_aboGroup_AB	0.938	0.787	1.117	maligY	1.060	0.978	1.148
don_aboGroup_B	0.893	0.775	1.029	recipientage	1.006	1.004	1.008
don_aboGroup_O	0.915	0.818	1.023	splitsplit	1.137	0.965	1.338
don_hgt_cm	0.998	0.991	1.004	lifesupportLifesupport	1.195	1.104	1.294
don_wgt_kg	0.998	0.991	1.005	hbvY	1.045	0.997	1.094
don_bmi	1.003	0.984	1.023	coldischemiatime	1.014	1.009	1.019
don_log_sgot	1.033	1.000	1.067	shareRegional_share	1.017	0.975	1.061
donorcodCNS_Tumor	1.115	0.889	1.399	shareNational_share	1.117	1.032	1.209
donorcodCVA_Stroke	1.058	1.002	1.117	recipientsexMale	1.028	0.976	1.083
donorcodHead_trauma	1.023	0.964	1.085	etiologyAcute	1.037	0.890	1.209
donorcodOther	1.079	0.949	1.227	etiologyAlcholic	1.051	0.920	1.202
don_creat	0.995	0.982	1.009	etiologyCholestatic	0.926	0.800	1.070
don_anti_cmvP	0.996	0.958	1.035	etiologyHBV	1.002	0.831	1.208
don_anti_hcvP	1.198	0.679	2.113	etiologyHCV	1.124	0.980	1.288
don_prerecov_diureticsY	0.935	0.902	0.970	etiologyMalignant	1.166	1.011	1.346
don_ddavpY	0.952	0.904	1.002	etiologyOther_cirrhosis	1.020	0.891	1.167
don_insulinY	1.020	0.982	1.059	etiologyOther_unknown	0.976	0.845	1.128
don_inotrop_supportY	0.995	0.959	1.033	pretxstatusHOSPITALIZED	1.054	0.994	1.118
don_hist_cigarette_gt20_pkyrY	1.021	0.979	1.065	pretxstatusIC_UNIT	1.085	1.002	1.176
don_hist_cocaineY	0.967	0.913	1.023	can_race_srtrBlack	1.207	1.140	1.278
don_hist_other_drugY	0.942	0.902	0.985	can_race_srtrOther	0.824	0.752	0.902
don_meet_cdc_high_riskY	0.954	0.897	1.015	can_ethnicity_srtrNLATIN	1.140	1.077	1.206
don_hist_diabYES	1.090	1.029	1.155	can_aboGroup_AB	1.004	0.869	1.159
don_htnY	1.067	1.019	1.116	can_aboGroup_B	1.125	0.981	1.290
don_hist_cancerY	1.096	1.002	1.200	can_aboGroup_O	1.082	0.967	1.211
don_log_bun	1.006	0.969	1.045	rec_functn_statSome_assistance	1.089	1.039	1.142
don_tot_bili	1.014	1.002	1.027	rec_functn_statTotal_assistance	1.109	1.044	1.179
don_log_sgpt	0.979	0.948	1.010	rec_work_incomeY	0.857	0.808	0.910
don_protein_urineY	1.001	0.965	1.038	rec_postx_los	1.008	1.007	1.008
don_sodium	0.999	0.997	1.001	rec_bmi	1.012	0.989	1.036
don_inr	1.006	0.996	1.017	retransplantationRetransplantation	1.462	1.362	1.569
don_hematocrit	1.002	0.999	1.005	rec_cmv_statP	0.974	0.938	1.011
don_anti_hbcP	1.117	0.800	1.559	rec_hbv_surf_antigenP	0.895	0.808	0.992
don_prerecov_steroidsY	1.009	0.967	1.053	rec_immuno_maint_medsY	0.126	0.118	0.135
don_anti_convulsY	1.035	0.954	1.122	rec_acute_rej_episodeYes	1.134	1.058	1.216
don_anti_hypertenY	0.986	0.943	1.031	rec_tumorY	1.332	1.218	1.458
don_vasodilY	1.040	0.988	1.096	rec_warm_isch_tm	1.002	1.001	1.002
don_heparinY	1.038	0.965	1.117	rec_bacteria_peritY	1.031	0.968	1.099
don_arginineY	0.991	0.955	1.029	can_variceal_bleedingY	1.088	0.994	1.191
don_txfus_terminal_hosp_num1to5	1.015	0.973	1.059	rec_tipssY	1.050	0.988	1.116
don_txfus_terminal_hosp_num6to10	1.061	0.987	1.141	can_last_bili	1.000	0.998	1.001
don_txfus_terminal_hosp_num>10	0.968	0.888	1.055	can_last_inr	0.992	0.976	1.007
don_clinical_infectY	0.944	0.883	1.010	can_last_serum_creat	1.030	1.016	1.044
don_infect_bloodY	1.048	0.978	1.123	can_last_serum_sodium	1.005	1.002	1.009
don_infect_luY	0.923	0.866	0.985	can_educationMedium_education	0.953	0.912	0.996
don_infect_urineY	0.988	0.927	1.053	can_educationHigher_education	0.905	0.855	0.957
don_heavy_alcoholY	1.027	0.976	1.082	rec_hgt_cm	1.007	0.999	1.015
don_tattoosY	1.000	0.958	1.044	rec_wgt_kg	0.994	0.986	1.002
don_hla_typY	1.031	0.903	1.176	can_peptic_ulcerYes	1.095	1.000	1.200
don_hist_prev_miY	1.018	0.934	1.110	can_angina_cadYes	1.311	1.177	1.459
don_pulm_cathY	1.017	0.964	1.074	can_drug_treat_hypertenY	0.997	0.955	1.041
don_hcv_statPositive	0.867	0.491	1.532	can_cereb_vascY	1.114	0.922	1.346
don_hbc_statPositive	0.890	0.635	1.248	can_periph_vascY	1.096	0.933	1.287
don_prev_gastro_diseaseY	1.001	0.936	1.072	can_acpt_abo_incompY	0.960	0.834	1.103
encephY	1.021	0.980	1.063	can_acpt_extracorp_liY	0.786	0.697	0.885
portal_veinY	1.173	1.081	1.273	can_acpt_li_segY	0.975	0.927	1.026
portal_hyperten_bleedY	1.077	0.984	1.178	can_acpt_hbc_posY	0.938	0.901	0.977
prev_abdom_surgY	1.104	1.063	1.146	can_acpt_hcv_posY	1.213	1.161	1.267

Table B.25: Cox backward model at distinct time points for failure-free survival. 33 variables were selected.

	exp(coef)	lower .95	upper .95
donortypeDCD	1.337	1.240	1.442
donorage	1.008	1.007	1.010
don_hgt_cm	0.998	0.996	0.999
don_prerecov_diureticsY	0.944	0.911	0.978
don_hist_other_drugY	0.931	0.895	0.968
don_htnY	1.077	1.033	1.122
don_infect_luY	0.890	0.858	0.923
portal_veinY	1.150	1.060	1.248
prev_abdom_surgY	1.091	1.051	1.132
diabY	1.181	1.133	1.231
hcvY	1.123	1.069	1.180
hccY	1.126	1.072	1.182
recipientage	1.006	1.004	1.008
lifesupportLifesupport	1.226	1.149	1.308
coldischemiatime	1.014	1.009	1.019
etiologyAcute	1.030	0.885	1.198
etiologyAlcholic	1.069	0.936	1.222
etiologyCholestatic	0.938	0.812	1.084
etiologyHBV	0.953	0.801	1.134
etiologyHCV	1.125	0.981	1.289
etiologyMalignant	1.156	1.002	1.333
etiologyOther_cirrhosis	1.027	0.897	1.175
etiologyOther_unknown	1.001	0.867	1.156
can_race_srtrBlack	1.205	1.140	1.273
can_race_srtrOther	0.832	0.762	0.908
can_ethnicity_srtrNLATIN	1.104	1.046	1.166
rec_functn_statSome_assistance	1.072	1.023	1.123
rec_functn_statTotal_assistance	1.114	1.057	1.174
rec_work_incomeY	0.852	0.803	0.903
rec_postx_los	1.007	1.007	1.008
retransplantationRetransplantation	1.401	1.308	1.501
rec_immuno_maint_medsY	0.187	0.175	0.199
rec_acute_rej_episodeYes	1.119	1.045	1.199
rec_tumorY	1.307	1.195	1.429
rec_warm_isch_tm	1.002	1.001	1.003
can_last_serum_creat	1.033	1.021	1.046
can_educationMedium_education	0.950	0.909	0.992
can_educationHigher_education	0.909	0.860	0.962
rec_hgt_cm	1.004	1.002	1.006
rec_wgt_kg	0.998	0.997	0.999
can_angina_cadYes	1.306	1.174	1.452
can_acpt_extracorp_liY	0.847	0.754	0.953
can_acpt_hcv_posY	1.178	1.132	1.226

Table B.26: Penalized Cox regression model with the LASSO method at distinct time points for FFS using λ_{1se}. Only the 36 non-zero coefficients are presented.

Variable	coef	exp(coef)
donortypeDCD	0.171	1.187
donorage	0.007	1.007
donorcodCVA_Stroke	0.003	1.003
don_prerecov_diureticsY	-0.022	0.978
don_hist_other_drugY	-0.026	0.974
don_hist_diabYES	0.024	1.024
don_htnY	0.035	1.035
don_clinical_infectY	-0.014	0.987
don_infect_luY	-0.057	0.944
portal_veinY	0.020	1.021
prev_abdom_surgY	0.048	1.049
last_dial_prior_weekY	0.012	1.012
diabY	0.118	1.125
hcvY	0.133	1.142
hccY	0.030	1.030
recipientage	0.004	1.004
lifesupportLifesupport	0.119	1.127
coldischemiatime	0.006	1.006
etiologyCholestatic	-0.022	0.979
etiologyHCV	0.014	1.014
etiologyMalignant	0.019	1.019
pretxstatusIC_UNIT	0.018	1.018
can_race_srtrBlack	0.129	1.137
can_race_srtrOther	-0.037	0.964
rec_functn_statTotal_assistance	0.005	1.005
rec_work_incomeY	-0.117	0.890
rec_postx_los	0.007	1.007
retransplantationRetransplantation	0.275	1.316
rec_immuno_maint_medsY	-1.619	0.198
rec_acute_rej_episodeYes	0.008	1.008
rec_tumorY	0.132	1.141
rec_warm_isch_tm	0.001	1.001
can_last_serum_creat	0.015	1.015
can_last_serum_sodium	0.001	1.001
can_angina_cadYes	0.130	1.139
can_acpt_hcv_posY	0.146	1.157

Appendix C

Medical terminology

Important functions of the liver:

1. It creates a fluid called bile that helps body digest food.
2. It removes waste products and other toxins from the blood.
3. It produces proteins and cholesterol.

Diseases such as hepatitis and cirrhosis can damage liver and prevent it from being functional.

Medical terms described below are sorted in alphabetical order:

- **Arginine**: is an amino acid that is used in the biosynthesis of proteins.
- **Ascites**: is the abnormal buildup of fluid in the abdomen. Symptoms may include increased abdominal size, increased weight, abdominal discomfort, and shortness of breath. Complications can include spontaneous bacterial peritonitis.
- **Bilirubin**: is excreted by bile and urine, and elevated levels may indicate certain diseases. It is responsible for the yellow color of bruises and the yellow discoloration in jaundice. Its subsequent breakdown products, such as stercobilin, cause the brown color of feces. A different breakdown product - urobilin - is the main component of the straw-yellow color in urine.
- **BUN**: Blood urea nitrogen (BUN) is a medical test that measures the amount of urea nitrogen found in blood. The liver produces urea in the urea cycle as a waste product of the digestion of protein.
- **Cerebrovascular Disease**: It refers to a group of conditions that can lead to a cerebrovascular event, such as a stroke. These events affect the blood vessels and blood supply to the brain. If a blockage, malformation, or hemorrhage prevents the brain cells from getting enough oxygen, brain damage can result.
- **Cytomegalovirus (CMV)**: The word comes from Greek cyto-, "cell", and megalo-, "large". CMV is a genus of viruses. Natural hosts are humans and monkeys. Human cytomegalovirus is the species that infects humans. Diseases include mononucleosis, and pneumonia.

- **DDAVP**: Desmopressin (DDAVP) is a medication used to treat diabetes insipidus, bed-wetting, hemophilia A, von Willebrand disease, and high blood urea levels.
- **Anti-CMV**: Anti- cytomegalovirus (CMV) antibodies.
- **Anti-HCV**: Anti-hepatitis C virus.
- **Donor Type DBD vs DCD**: A)DBD (donor after brain-dead): beating heart cadavers (heart is still operating under mechanical support), B)DCD (donor after circulatory death): non-heart beating donor.
- **Extra-corporeal**: It is a medical procedure which is performed outside the body. In this procedure, blood is taken from a patient's circulation to have a process applied to it before it is returned to the circulation. All the apparatus carrying the blood outside the body constitute the extra corporeal circuit.
- **HBC**: Hepatitis B core antigen is a hepatitis B viral protein. It is an indicator of active viral replication; this means the person infected with Hepatitis B can likely transmit the virus on to another person (i.e. this person is infectious).
- **HCC**: Hepatocellular carcinoma (HCC) is a primary malignancy of the liver and occurs predominantly in patients with underlying chronic liver disease and cirrhosis. The cell(s) of origin are believed to be the hepatic stem cells, although this remains the subject of investigation. Tumors progress with local expansion, intra-hepatic spread, and distant metastases. HCC is now the third leading cause of cancer deaths worldwide, with over 500,000 people affected.
- **Hematocrit**: The hematocrit (Ht), is the volume percentage of red blood cells in blood. It is normally 47% +-5% for men and 42% +-5% for women. It is considered an integral part of a person's complete blood count results, along with hemoglobin concentration, white blood cell count, and platelet count.
- **Heparin**: Is a type of medication which is used as an anticoagulant (blood thinner). More specifically, it treats and prevents deep vein thrombosis, pulmonary embolism, and arterial thromboembolism. It is also used in the treatment of heart attacks and unstable angina. Other applications include test tubes and kidney dialysis machines.
- **Hepatitis B Core Antibody**: It is as test which looks for antibodies called IgM (anti-HBC) in blood. Such test can assess whether you are actively infected with the hepatitis B virus (HBV).
- **HLA-Matching**: The term "HLA" is short for Human Leukocyte Antigens. They are proteins in the immune system that determine whether a patient will react against a donor transplant. The best possible donor is one whose HLA is closely matched to the patient. HLA-matching is much more complicated than blood typing. There are many HLA markers which make a persons cell unique. A close HLA match may lower risk for complications after transplantation.

- **Inotrope**: It is an agent that alters the force or energy of muscular contractions. Negatively inotropic agents weaken the force of muscular contractions. Positively inotropic agents increase the strength of them. For some patients, inotropic support was provided.

- **INR**: International normalized ratio (INR) is blood-clotting test. It is used to measure how quickly blood forms a clot, compared with normal clotting time. When doctor is evaluating the function of your liver, a high INR usually means that the liver is not working as well as it could, because it is not making the blood clot normally.

- **Myocardial Infarction**: It is commonly known as a heart attack. It occurs when blood flow decreases or stops to a part of the heart, causing damage to the heart muscle. The most common symptom is chest pain or discomfort which may travel into the shoulder, arm, back, neck, or jaw.

- **Peptic Ulcer**: Peptic ulcers are open sores developed on the inside lining of stomach and the upper portion of small intestine. The most common symptom of a peptic ulcer is stomach pain.

- **Peripheral Vascular Disease**: PVD is a blood circulation disorder which causes the blood vessels outside of your heart and brain to narrow, block, or spasm. This can happen in your arteries or veins. PVD typically causes pain and fatigue, often in your legs, and especially during exercise.

- **Portal Hypertension**: It is hypertension (high blood pressure) in the hepatic portal system made up of the portal vein and its branches - which drain from most of the intestine to the liver.

- **Portal Vein Thrombosis**: It is a form of venous thrombosis affecting the hepatic portal vein, which can lead to portal hypertension and reduction in the blood supply to the liver.

- **Serology**: It is the study of blood serum.

- **Serum Albumin**: It is the main protein of human blood plasma. It binds water, cations fatty acids, hormones, bilirubin, thyroxine and pharmaceuticals. Its main function is to regulate the oncotic pressure of blood.

- **Serum Creatinine**: It is a blood measurement which is an important indicator of renal health because it is an easily measured by-product of muscle metabolism that is excreted unchanged by the kidneys.

- **Serum Sodium**: Measurement of serum sodium is routine in assessing electrolyte, acid-base, and water balance, as well as renal function. Sodium accounts for approximately 95% of the osmotically active substances in the extracellular compartment, provided that the patient is not in renal failure or does not have severe hyperglycemia.

- **SGPT/ALT**: The alanine aminotransferase (ALT) test is a blood test that checks for liver damage. Doctors can use this test to find out if a disease, drug, or injury has damaged the liver. Our body uses ALT to break food into energy.

Normally, ALT levels in the blood are low. If your liver is damaged, it will release more ALT into your blood and levels will rise. Note that ALT used to be called serum glutamic-pyruvic transaminase (SGPT).

- **SGOT/AST**: The SGOT test is a blood test. It helps to determine how well the liver is functioning by measuring levels of aspartate aminotransferase in the blood. Too much of this enzyme can indicate a problem, such as liver damage.
- **Spontaneous Bacterial Peritonitis**: SBP is the development of a bacterial infection in the peritoneum causing peritonitis, despite the absence of an obvious source for the infection. It occurs almost exclusively in people with portal hypertension (increased pressure over portal vein), usually because of cirrhosis of the liver.
- **TIPSS**: Transjugular intrahepatic portosystemic shunt (TIPS or TIPSS) is an artificial channel within the liver that establishes communication between the inflow portal vein and the outflow hepatic vein. It is used to treat portal hypertension (often due to liver cirrhosis), which frequently leads to intestinal bleeding, life-threatening esophageal bleeding (esophageal varices) and the build-up of fluid within the abdomen (ascites).
- **Variceal Bleeding**: Esophageal varix is a tortuous dilatation of an esophageal vein, especially in the distal portion. It results from any condition that causes portal hypertension, typically cirrhosis of the liver.
- **Vasodilators**: It Is a class of drugs often used to treat cardiovascular conditions, such as hypertension. Vasodilators are a group of medicines that dilate (open) blood vessels, which allows blood to flow more easily. They are used to treat or prevent: 1)High blood pressure (hypertension), 2)Heart failure, 3)Preeclampsia (high blood pressure during pregnancy), 4) Angina (chest pain caused by reduced blood flow to the heart), 5) Pulmonary hypertension (high blood pressure that affects the arteries of your lungs).

Web sources utilized:

1. https://aasldpubs.onlinelibrary.wiley.com
2. https://emedicine.medscape.com
3. https://www.everydayhealth.com
4. https://www.healthline.com
5. https://www.hepatitis.va.gov
6. https://www.mayoclinic.org
7. https://medical-dictionary.thefreedictionary.com
8. https://www.medicalnewstoday.com
9. https://www.webmd.com
10. https://en.wikipedia.org

Appendix D

R-code

D.1 Data description

D.1.1 Variable preselection and data preprocessing

```
install_packages <- c("foreign", "Hmisc", "plyr")
for (i in 1:length(install_packages)){
  if (!install_packages[i] %in% installed.packages()){
    install.packages(install_packages[i])
  }
}
library(foreign)
library(Hmisc)
library(plyr)

# read data file in sav format
unos <- read.spss("UNOS_file.sav", to.data.frame = TRUE)
# translate characters from upper to lower case
names(unos) <- casefold(names(unos))

# some factor variables have as part of their levels empty
# strings or NA so we drop them as levels
for (j in colnames(unos)) {
  if (is.factor(unos[, j])) {
    unos[, j] <- factor(unos[, j], exclude = c(NA, " ", "  ",
                                               "   ", "    "))
  }
}

# Fix dates
# Dates in SPSS are recorded in seconds since October 14, 1582
# the date of the beginning of the Julian calendar.
# To transform the value to the R "Date" format, you simply need
# to turn seconds into days and specify the origin.
spss2date <- function(x) as.Date(x/(24*60*60),
                                 origin = "1582-10-14")
# graft failure date
```

```
unos$tfl_graft_dt <- spss2date(unos$tfl_graft_dt)
unos$rec_tx_dt <- spss2date(unos$rec_tx_dt) # transplant date
# date patient died
unos$tfl_death_dt <- spss2date(unos$tfl_death_dt)
# cohort date
unos$tfl_endtxfu <- spss2date(unos$tfl_endtxfu)

# POSSIBLE RESPONSES (in years) - STATUS VARIABLES
year <- 365.25 # define days per year
# patient survival in years
unos$patientsurvival <- unos$patientsurvival / year
# death state: because as.numeric translates to 1, 2 we subtract 1
unos$death <- as.numeric(unos$death) - 1
# graft failure free survival in years
unos$gs_ffs <- unos$gs_ffs / year
# failure free state in numeric
unos$gs_ffsstate <- as.numeric(unos$gs_ffsstate) - 1

# remove value from 4 patients that had unrealistic BMI
unos$rec_bmi[unos$rec_bmi > 80] <- NA

# candidate's last encephalopathy used for MELD
# levels 1: None, 2: 1-2, 3: 3-4
unos$enceph <- as.numeric(unos$can_last_enceph) - 1
unos$enceph[unos$enceph > 1] <- 1

# history of portal vein thrombosis with 3 levels: N, U, Y
unos$portal_vein <- as.numeric(unos$can_portal_vein) - 2
# change portal vein thrombosis to 0, 1
unos$portal_vein[unos$portal_vein<0] <- 0

# Did Patient receive 5 or more units of packed red blood cells
# within 48 hours prior to transplantation
# due to spontaneous portal hypertensive bleeding
unos$portal_hyperten_bleed <-
as.numeric(unos$rec_portal_hyperten_bleed) - 2
unos$portal_hyperten_bleed[unos$portal_hyperten_bleed <0] <- 0

# previous upper abdominal surgery
unos$prev_abdom_surg <- as.numeric(unos$rec_prev_abdom_surg) - 2
unos$prev_abdom_surg[unos$prev_abdom_surg<0] <- 0

# candidate last ascites (used for MELD)
unos$ascites <- as.numeric(unos$can_last_ascites) - 2
# turned it to 0-1 from (N, U, Y) = (1, 2, 3)
unos$ascites[unos$ascites < 0] <- 0

# Last non-blank val. of dialysis within prior week
unos$last_dial_prior_week <-
as.numeric(unos$can_last_dial_prior_week) - 2
unos$last_dial_prior_week[unos$last_dial_prior_week < 0] <- 0
```

```
lvls <- levels(unos$pretxstatus) # pre-treatment status
unos$pretxstatus <- as.numeric(unos$pretxstatus)
unos$pretxstatus <- factor(unos$pretxstatus,
                           levels=3:1, labels=rev(lvls))

# candidate last albumin (used for MELD)
unos$albumin2 <- 0
unos$albumin2[unos$can_last_albumin < 2] <- 1
unos$albumin2[is.na(unos$can_last_albumin)] <- NA
unos$albumin2 <- factor(unos$albumin2,
                        levels=0:1,
                        labels=c(">=2", "<2"))

# candidate diabetes type
unos$diab <- as.numeric(unos$can_diab_ty) - 1
unos$diab[unos$diab>=4] <- 0
unos$diab[unos$diab>1] <- 1

# HCV serology status levels N = Negative, ND = not done,
# P = positive, U = unknown
unos$hcv <- as.numeric(unos$rec_hcv_stat) - 2
# set everything else than positive to zero
unos$hcv[unos$hcv != 1] <- 0

# Ever Approved for an HCC exception: already 0 or 1
unos$hcc <- unos$canhx_mpxcpt_hcc_approve_ind

# Pretransplant Malignancy with levels N, U, Y
unos$malig <- as.numeric(unos$rec_malig) - 2
unos$malig[unos$malig<0] <- 0

# HBV: Core Antibody//Core Antibody with levels N = negative,
# ND = not done, P = positive, U = unknown
unos$hbv <- as.numeric(unos$rec_hbv_antibody) - 2
unos$hbv[unos$hbv != 1] <- 0

# drop 376 variables because they are redundant or have many NA's
source("drop_list.R")

# now create lists of variables for ids, dates
ids_list <- c("pers_id", # Unique person ID to replace SSN
              "donor_id", # Encrypted Unique Donor ID (all donors)
              "px_id", # Patient Identifier
              "trr_id", # Unique identifier for TRR - unique key
              "tx_id", # Unique identifier for Transplant
              "rec_opo_id", # OPO Serving Transplant Center
              "donor_id_vessel", # Encrypted Vessel Donor ID
              "can_listing_ctr_id", # CAN_LISTING_ Center ID
              "pers_nexttx_trr_id", # Unique identifier for TRR
              "ctr_id" # Unique identifier for ctr_cd/ctr_ty
```

```
)

dates_list <- c("rec_tx_dt", # Transplant Date
                "don_recov_dt", # Recovery Date (donor to OR)
                # Death date determined from SSA database
                "pers_ssa_death_dt",
                "pers_optn_death_dt", # OPTN Death date
                "pers_restrict_death_dt", # Restricted death date
                "rec_px_stat_dt", # Patient Status/Date
                "rec_dischrg_dt", # Transplant Discharge Date
                "rec_admission_dt", # Date of Admission to Tx Center
                "rec_txfer_dt", #  Transfer Date
                "rec_fail_dt", # Date of Graft Failure
                "tfl_endtxfu", # End of cohort follow up
                "can_init_act_stat_dt", # Date First Active Status
                "rec_pretx_lab_dt", # Pretransplant Lab Date
                "can_listing_dt", # Listing Date
                "tfl_graft_dt", # Graft Failure date
                "tfl_death_dt", # Date Patient died
                "can_activate_dt", # candidate activation date
                "can_tiebreaker_dt", # Tiebreaker Date
                 # Date of Latest Bone Marrow Transplant
                "can_prev_bone_marrow_dt",
                 # most recent serum_creat date
                "don_alloc_ecd_serum_creat_dt",
                "can_endwlfu", # candidate censoring time
                "tfl_lafudate", # Last Graft Follow up Date
                "pers_nexttx", # transplant date
                "transplantyear" # year of the transplantation
)

unos_dates <- unos[, dates_list]
unos_ids <- unos[, ids_list]
unos <- unos[, !colnames(unos) %in% c(ids_list, dates_list)]

########################################
# Transformations of the variables
########################################

unos$don_citizenship <- as.factor(unos$don_citizenship)
unos$don_citizenship <- factor(unos$don_citizenship,
                               levels = c(1, 2, 3, 4, 5))
unos$don_citizenship <- factor(unos$don_citizenship,
                           labels = c("US Citizen", "Resident Alien",
                                      "Non-Resident Alien,
                                      Specify Country",
                                      "Non-US Citizen/US Resident",
                                      "Non-US Citizen/Non-US
                                      Resident"))

unos$can_citizenship <- factor(as.factor(unos$can_citizenship),
```

```
levels = c(1, 2, 3, 4, 5, 6),
labels = c("US Citizen", "RESIDENT ALIEN",
"NON-RESIDENT ALIEN, Year Entered US",
"Non-US Citizen/US Resident",
"Non-US Citizen/Non-US Resident,
Traveled to US for Reason Other Than Transplant",
"Non-US Citizen/Non-US Resident, Traveled to US for Transplant"))

unos$don_cad_don_cod <- factor(as.factor(unos$don_cad_don_cod),
levels = c(1, 2, 3, 4, 999),
labels = c("ANOXIA", "CEREBROVASCULAR/STROKE", "HEAD TRAUMA",
           "CNS TUMOR", "OTHER SPECIFY"))


unos$don_death_circum <- factor(as.factor(unos$don_death_circum),
levels = c(1, 2, 3, 4, 5, 6, 997),
labels = c("MVA", "SUICIDE", "HOMICIDE", "CHILD-ABUSE",
           "NON-MVA", "DEATH FROM NATURAL CAUSES", "NONE OF THEM"))


unos$don_hist_diab <- factor(as.factor(unos$don_hist_diab),
levels = c(1, 2, 3, 4, 5),
labels = c("NO", "YES, 0-5 YEARS", "YES, 6-10 YEARS",
           "YES, >10 YEARS", "YES, DURATION UNKNOWN"))

unos$don_hist_hyperten <- factor(as.factor(unos$don_hist_hyperten),
levels = c(1, 2, 3, 4, 5),
labels = c("NO", "YES, 0-5 YEARS", "YES, 6-10 YEARS",
           "YES, >10 YEARS", "YES, DURATION UNKNOWN"))

unos$don_hist_cancer <- factor(as.factor(unos$don_hist_cancer),
                               exclude = c(NA, "998"))

unos$don_a1 <- as.factor(unos$don_a1)
unos$don_a2 <- as.factor(unos$don_a2)
unos$don_b1 <- as.factor(unos$don_b1)
unos$don_b2 <- as.factor(unos$don_b2)
unos$don_dr1 <- as.factor(unos$don_dr1)
unos$don_dr2 <- as.factor(unos$don_dr2)
unos$rec_dgn <- as.factor(unos$rec_dgn)

unos$rec_tx_procedure_ty <-
factor(as.factor(unos$rec_tx_procedure_ty),
levels = c(701, 702, 703, 704),
labels = c("Whole Liver", "Partial Liver",
         "Split Liver",
         "Whole Liver with Pancreas (Technical Reasons)"))

unos$rec_med_cond <- factor(as.factor(unos$rec_med_cond),
levels = c(3, 2, 1),
labels = c("NOT HOSPITALIZED",
```

```
            "HOSPITALIZED",
            "IN INTENSIVE CARE UNIT"))

unos$rec_functn_stat <- as.factor(unos$rec_functn_stat)

unos$rec_physc_capacity <-
factor(as.factor(unos$rec_physc_capacity),
levels = c(1, 2, 3, 996),
labels = c("No Limitations", "Limited Mobility",
            "Wheelchair bound or more limited",
            "Not applicable (patient hospitalized)"))

unos$rec_primary_pay <- as.factor(unos$rec_primary_pay)
unos$rec_acute_rej_episode <-
factor(as.factor(unos$rec_acute_rej_episode),
                  levels = c(1, 2, 3),
                  labels = c("Yes,
at least one episode treated with anti-rejection agent",
            "Yes, none treated with additional anti-rejection agent",
            "No"))

# final MELD of candidate
unos$can_last_srtr_lab_meld <- unos$can_last_srtr_lab_meld - 6200

unos$can_last_ascites <- factor(as.factor(unos$can_last_ascites),
                                        levels = c(1, 2, 3),
                       labels = c("Absent", "Slight", "Moderate"))

unos$can_last_enceph <- factor(as.factor(unos$can_last_enceph),
                                      levels = c(1, 2, 3),
                                labels = c("None", "1-2", "3-4"))

unos$can_education <- factor(as.factor(unos$can_education),
levels = c(1, 2, 3, 4, 5, 6, 996, 998),
labels = c("NONE", "GRADE SCHOOL (0-8)",
            "HIGH SCHOOL (9-12) or GED",
            "ATTENDED COLLEGE/TECHNICAL SCHOOL",
            "ASSOCIATE/BACHELOR DEGREE",
            "POST-COLLEGE GRADUATE DEGREE",
            "N/A (< 5 YRS OLD)",
            "UNKNOWN"))

unos$can_dgn <- as.factor(unos$can_dgn) # candidate main diagnosis

unos$can_peptic_ulcer <- factor(as.factor(unos$can_peptic_ulcer),
                                     levels = c(1, 2, 3, 4, 998),
             labels = c("No", " Yes, active within the last year",
            "Yes, not active within the last year",
            "Yes, activity unknown",
            "Unknown"))
```

```
 unos$can_angina_cad <- factor(as.factor(unos$can_angina_cad),
 levels = c(1, 2, 3, 4, 998),
 labels = c("No", "Yes, and documented Coronary Artery Disease",
            "Yes, with no documented Coronary Artery Disease",
            "Yes, but Coronary Artery Disease unknown",
            "Status Unknown"))
unos$can_med_cond <-  factor(as.factor(unos$can_med_cond),
levels = c(3, 2, 1),
labels = c("NOT HOSPITALIZED", "HOSPITALIZED NOT IN ICU",
           "IN INTENSIVE CARE UNIT"))

unos$can_functn_stat <- as.factor(unos$can_functn_stat)
unos$can_physc_capacity <-
factor(as.factor(unos$can_physc_capacity),
levels = c(1, 2, 3, 996),
labels = c("No Limitations", "Limited Mobility",
           "Wheelchair bound or more limited",
           "Not Applicable (< 1 year old or hospitalized)"))

unos$can_dial <-  factor(as.factor(unos$can_dial),
                         levels = c(1, 2, 3, 4, 5, 999),
labels = c("No dialysis", "Hemodialysis", "Peritoneal Dialysis",
            "CAVH: Continuous Arteriovenous Hemofiltration",
            "CV VH: Continuous Venous/Venous Hemofiltration",
            "Dialysis-Unknown Type was performed"))

unos$don_txfus_terminal_hosp_num[
                           unos$don_txfus_terminal_hosp_num == 998]
                           <- NA # 0, 1, 2, or 3 transfusions

unos$don_hcv_stat <- as.factor(unos$don_hcv_stat)
# Hepatitis B Core Antibody Status
unos$don_hbc_stat <- as.factor(unos$don_hbc_stat)
# exclude one donor with wrong BMI
unos$don_bmi[unos$don_bmi > 80] <- NA
# make a summary of UNOS data
description <- describe(unos)
capture.output(print(description),
               file = "Unos_summary.txt")

# file 3: grouping variables for donor and patient
# first transform three very right skewed variables
unos$don_log_sgot <- log(unos$don_sgot + 1)
unos$don_log_bun <- log(unos$don_bun + 1)
unos$don_log_sgpt <- log(unos$don_sgpt + 1)

donor_vars <- c("don_gender", "don_ethnicity_srtr", "donortype",
                "donorrace", "donorage", "don_abo", "don_hgt_cm",
                "don_wgt_kg", "don_bmi", "don_log_sgot", "donorcod",
                 "don_creat", "don_anti_cmv",  "don_anti_hcv",
                 "don_prerecov_diuretics", "don_ddavp",
```

```
                  "don_insulin", "don_inotrop_support",
                  "don_hist_cigarette_gt20_pkyr", "don_anti_hbc",
                  "don_hist_cocaine", "don_hist_other_drug",
                  "don_meet_cdc_high_risk", "don_hist_diab",
                  "don_htn", "don_hist_cancer", "don_log_bun",
                  "don_tot_bili", "don_log_sgpt", "don_protein_urine",
                  "don_sodium", "don_inr", "don_hematocrit",
                  "don_prerecov_steroids",  "don_anti_convuls",
                  "don_anti_hyperten", "don_vasodil", "don_heparin",
                  "don_arginine", "don_txfus_terminal_hosp_num",
                  "don_clinical_infect", "don_infect_blood",
                  "don_infect_lu", "don_infect_urine",
                  "don_heavy_alcohol", "don_prev_gastro_disease"
                  "don_tattoos", "don_hla_typ", "don_hist_prev_mi",
                  "don_pulm_cath", "don_hcv_stat", "don_hbc_stat",
                  )

# which are defined for both candidates and recipients:
# portal_vein, prev_abdom_surg, malig, age (rec_age_at_tx,
# can_age_at_listing), life_support, dgn,
# med_cond, functn_stat, physc_capacity, work_income, bmi,
# artificial_li, tipss, hgt_cm, wgt_kg

# variables in the dataset that are comorbidities
# rec_bacteria_perit: spontaneous bacterial peritonitis,
# enceph: encephalopathy, diabetes, ascites, can_cereb_vasc,
# hypertension, can_periph_vasc: peripheral vascular disease
# keep the most recent patient measurements

patient_vars <- c("enceph", "portal_vein", "portal_hyperten_bleed",
                  "prev_abdom_surg", "ascites",
                  "last_dial_prior_week", "rec_functn_stat",
                  "albumin2", "diab", "hcv", "hcc", "malig",
                  "recipientage", "split", "lifesupport", "hbv",
                  "coldischemiatime", "share", "recipientsex",
                  "etiology","pretxstatus",  "can_race_srtr",
                  "can_ethnicity_srtr", "can_abo", "can_last_bili",
                  "rec_work_income", "rec_postx_los", "rec_bmi",
                  "retransplantation", "rec_cmv_stat",
                  "rec_hbv_surf_antigen", "rec_immuno_maint_meds",
                  "rec_acute_rej_episode", "rec_tumor",
                  "rec_warm_isch_tm",  "rec_bacteria_perit",
                  "can_variceal_bleeding", "rec_tipss",
                  "can_last_inr", "can_last_serum_creat",
                  "can_last_serum_sodium",  "can_education",
                  "rec_hgt_cm", "rec_wgt_kg", "can_peptic_ulcer",
                  "can_angina_cad", "can_drug_treat_hyperten",
                  "can_cereb_vasc", "can_periph_vasc",
                  "can_acpt_abo_incomp", "can_acpt_extracorp_li",
                  "can_acpt_li_seg", "can_acpt_hbc_pos",
                  "can_acpt_hcv_pos")
```

```
response_variables <- c("patientsurvival", "death",
                        "gs_ffs", "gs_ffsstate")
unos <- unos[, c(donor_vars, patient_vars, response_variables)]

##############################
# new transformations
##############################

# First for the donor
unos$don_abo <- revalue(unos$don_abo,
c("A         " = "Group A", "A1         " = "Group A",
"A1B       " = "Group AB",  "A2         " = "Group A",
"A2B       " = "Group AB", "AB        " = "Group AB",
"B         " = "Group B", "O         " = "Group O"))
unos$don_abo <- relevel(unos$don_abo, ref = "Group O")

unos$don_anti_cmv <- revalue(unos$don_anti_cmv,
                             c("I      " = "N",
                               "N      " = "N", "ND     " = "N",
                               "P      " = "P", "U      " = "N"))

unos$don_anti_hcv <- revalue(unos$don_anti_hcv,
c("I      " = "N", "N      " = "N", "ND     " = "N",
  "P      " = "P", "U      " = "N"))

unos$don_txfus_terminal_hosp_num <-
as.factor(unos$don_txfus_terminal_hosp_num)
unos$don_txfus_terminal_hosp_num <-
revalue(unos$don_txfus_terminal_hosp_num,
        c("0" = "None", "1" = "1-5",
          "2" = "6-10",
          "3" = "Greater than 10"))

unos$don_prerecov_diuretics <- revalue(unos$don_prerecov_diuretics,
c("N   " = "N", "U   " = "N", "Y   " = "Y"))

unos$don_ddavp <- revalue(unos$don_ddavp,
c("N   " = "N", "U   " = "N", "Y   " = "Y"))

unos$don_insulin <- revalue(unos$don_insulin,
c("N   " = "N", "U   " = "N", "Y   " = "Y"))

unos$don_inotrop_support <- revalue(unos$don_inotrop_support,
c("N   " = "N", "U   " = "N", "Y   " = "Y"))
unos$don_hist_cigarette_gt20_pkyr <-
revalue(unos$don_hist_cigarette_gt20_pkyr,
        c("N   " = "N", "U   " = "N", "Y   " = "Y"))

unos$don_hist_cocaine <- revalue(unos$don_hist_cocaine,
c("N   " = "N", "U   " = "N", "Y   " = "Y"))
```

```
unos$don_hist_other_drug <- revalue(unos$don_hist_other_drug,
c("N  " = "N", "U  " = "N", "Y  " = "Y"))
unos$don_meet_cdc_high_risk <- revalue(unos$don_meet_cdc_high_risk,
c("N  " = "N", "U  " = "N", "Y  " = "Y"))

unos$don_hist_diab <- revalue(unos$don_hist_diab,
c("NO" = "NO", "YES, 0-5 YEARS" = "YES",
"YES, 6-10 YEARS" = "YES", "YES, >10 YEARS" = "YES",
"YES, DURATION UNKNOWN" = "YES"))

# make hypertension factor
unos$don_htn <- factor(unos$don_htn, labels = c("N", "Y"))
unos$don_hist_cancer <- as.numeric(unos$don_hist_cancer) - 1
unos$don_hist_cancer[unos$don_hist_cancer > 0] <- 1
unos$don_hist_cancer <- factor(unos$don_hist_cancer,
                               labels = c("N", "Y")) # make it factor
unos$don_protein_urine <- revalue(unos$don_protein_urine,
c("N  " = "N", "U  " = "N", "Y  " = "Y"))
unos$don_anti_hbc <- revalue(unos$don_anti_hbc,
c("I    " = "N", "N    " = "N", "ND   " = "N",
"P    " = "P", "U    " = "N"))
unos$don_prerecov_steroids <-  revalue(unos$don_prerecov_steroids,
c("N  " = "N", "U  " = "N", "Y  " = "Y"))
unos$don_anti_convuls <-  revalue(unos$don_anti_convuls,
c("N  " = "N", "U  " = "N", "Y  " = "Y"))
unos$don_anti_hyperten <-  revalue(unos$don_anti_hyperten,
c("N  " = "N", "U  " = "N", "Y  " = "Y"))
unos$don_vasodil <-  revalue(unos$don_vasodil,
c("N  " = "N", "U  " = "N", "Y  " = "Y"))
unos$don_heparin <-  revalue(unos$don_heparin,
c("N  " = "N", "U  " = "N", "Y  " = "Y"))
unos$don_arginine <-  revalue(unos$don_arginine,
c("N  " = "N", "U  " = "N", "Y  " = "Y"))
unos$don_clinical_infect <- revalue(unos$don_clinical_infect,
c("N  " = "N", "U  " = "N", "Y  " = "Y"))

unos$don_infect_blood <- factor(unos$don_infect_blood,
                                labels = c("N", "Y"))
unos$don_infect_lu <- factor(unos$don_infect_lu,
                             labels = c("N", "Y"))
unos$don_infect_urine <- factor(unos$don_infect_urine,
                                labels = c("N", "Y"))
unos$don_heavy_alcohol <- revalue(unos$don_heavy_alcohol,
c("N  " = "N", "U  " = "N", "Y  " = "Y"))
unos$don_tattoos <- revalue(unos$don_tattoos,
c("N  " = "N", "U  " = "N", "Y  " = "Y"))

unos$don_hla_typ <- revalue(unos$don_hla_typ,
c("N  " = "N", "U  " = "N", "Y  " = "Y"))
unos$don_hist_prev_mi <- revalue(unos$don_hist_prev_mi,
c("N  " = "N", "U  " = "N", "Y  " = "Y"))
```

```
unos$don_hcv_stat <- revalue(unos$don_hcv_stat,
c("1" = "Positive", "2" = "Negative", "3" = "Negative",
  "5" = "Negative", "6" = "Negative", "7" = "Negative"))
unos$don_hcv_stat <-  relevel(unos$don_hcv_stat, ref = "Negative")

unos$don_hbc_stat <- revalue(unos$don_hbc_stat,
c("1" = "Positive", "2" = "Negative", "3" = "Negative",
  "5" = "Negative", "6" = "Negative", "7" = "Negative"))
unos$don_hbc_stat <-  relevel(unos$don_hbc_stat, ref = "Negative")
unos$don_prev_gastro_disease <-
revalue(unos$don_prev_gastro_disease,
c("N  " = "N", "U  " = "N", "Y  " = "Y"))

# Secondly for the patient
unos$enceph <- factor(unos$enceph, labels = c("N", "Y"))
unos$portal_vein <- factor(unos$portal_vein, labels = c("N", "Y"))
unos$portal_hyperten_bleed <- factor(unos$portal_hyperten_bleed,
labels = c("N", "Y")) # make it factor
unos$prev_abdom_surg <- factor(unos$prev_abdom_surg,
labels = c("N", "Y"))
unos$ascites <- factor(unos$ascites, labels = c("N", "Y"))
unos$last_dial_prior_week <- factor(unos$last_dial_prior_week,
labels = c("N", "Y")) # make it factor
unos$diab <- factor(unos$diab, labels = c("N", "Y"))
unos$hcv <- factor(unos$hcv, labels = c("N", "Y"))
unos$hcc <- factor(unos$hcc, labels = c("N", "Y"))
unos$hbv <- factor(unos$hbv, labels = c("N", "Y"))
unos$malig <- factor(unos$malig, labels = c("N", "Y"))
unos$can_race_srtr <- revalue(unos$can_race_srtr,
c("BLACK                          " = "Black",
"WHITE                          " = "White",
"ASIAN                          " = "Other",
"MULTI                          " = "Other",
"NATIVE                         " = "Other",
"PACIFIC                        " = "Other"))

# set White as reference category
unos$can_race_srtr <- relevel(unos$can_race_srtr, ref = "White")

unos$can_abo <- revalue(unos$can_abo,
c("A         " = "Group A", "A1        " = "Group A",
"A1B       " = "Group AB",  "A2        " = "Group A",
"A2B       " = "Group AB", "AB        " = "Group AB",
"B         " = "Group B", "O         " = "Group O"))

unos$can_abo <- relevel(unos$can_abo, ref = "Group O")
unos$rec_functn_stat <- revalue(unos$rec_functn_stat,
c("1" = "No assistance", "2" = "Some assistance",
   "3" = "Total assistance", "996" = "Total assistance",
   "998" = "No assistance", "2010" = "Total assistance",
```

```
    "2020" = "Total assistance", "2030" = "Total assistance",
    "2040" = "Total assistance",  "2050" = "Some assistance",
    "2060" = "Some assistance", "2070" = "Some assistance",
    "2080" = "No assistance", "2090" = "No assistance",
    "2100" = "No assistance"))

unos$rec_work_income <- revalue(unos$rec_work_income,
c("N  " = "N", "U  " = "N", "Y  " = "Y"))
unos$rec_cmv_stat <- revalue(unos$rec_cmv_stat,
c("N    " = "N", "ND   " = "N", "P    " = "P",
  "U    " = "N"))

unos$rec_hbv_surf_antigen <- revalue(unos$rec_hbv_surf_antigen,
c("N    " = "N", "ND   " = "N",
  "P    " = "P", "U    " = "N"))

unos$rec_acute_rej_episode <- revalue(unos$rec_acute_rej_episode,
 c("Yes, at least one episode treated with anti-rejection agent"
 = "Yes",
 "Yes, none treated with additional anti-rejection agent" = "Yes",
 "No" = "No"))
unos$rec_acute_rej_episode <- relevel(unos$rec_acute_rej_episode,
                                      ref = "No")
unos$rec_tumor <- revalue(unos$rec_tumor,
                          c("N  " = "N", "U  " = "N", "Y  " = "Y"))

unos$rec_bacteria_perit <- revalue(unos$rec_bacteria_perit,
                           c("N  " = "N", "U  " = "N", "Y  " = "Y"))
unos$can_variceal_bleeding <- revalue(unos$can_variceal_bleeding,
                          c("N  " = "N", "U  " = "N", "Y  " = "Y"))
unos$rec_tipss <- revalue(unos$rec_tipss,
                          c("N  " = "N", "U  " = "N", "Y  " = "Y"))

unos$can_education <- revalue(unos$can_education,
c("NONE" = "None/low/undefined education",
  "GRADE SCHOOL (0-8)" = "None/low/undefined education",
  "HIGH SCHOOL (9-12) or GED" = "Medium education",
  "ATTENDED COLLEGE/TECHNICAL SCHOOL" = "Medium education",
  "ASSOCIATE/BACHELOR DEGREE" = "Higher education",
  "POST-COLLEGE GRADUATE DEGREE" = "Higher education",
  "N/A (< 5 YRS OLD)" = "None/low/undefined education",
  "UNKNOWN" = "None/low/undefined education"))

unos$can_peptic_ulcer <- revalue(unos$can_peptic_ulcer,
c("No" = "No", " Yes, active within the last year" = "Yes",
  "Yes, not active within the last year" = "Yes",
  "Yes, activity unknown" = "Yes", "Unknown" = "No"))

unos$can_angina_cad <- revalue(unos$can_angina_cad,
c("No" = "No",
  "Yes, and documented Coronary Artery Disease" = "Yes",
```

```
  "Yes, with no documented Coronary Artery Disease" = "Yes",
  "Yes, but Coronary Artery Disease unknown" = "Yes",
  "Status Unknown" = "No"))

unos$can_drug_treat_hyperten <-
revalue(unos$can_drug_treat_hyperten,
c("N   " = "N", "U   " = "N", "Y   " = "Y"))
unos$can_cereb_vasc <- revalue(unos$can_cereb_vasc,
c("N   " = "N", "U   " = "N", "Y   " = "Y"))
unos$can_periph_vasc <- revalue(unos$can_periph_vasc,
c("N   " = "N", "U   " = "N", "Y   " = "Y"))

# save unos file with the selected variables
write.table(unos, "unos_selected.txt", sep="\t")
```

D.1.2 Descriptive statistics

```
install_packages <- c("knitr", "data.table", "ggplot2",
                      "gridExtra", "survminer", "survival",
                      "corrplot", "scales", "purrr")
for (i in 1:length(install_packages)){
  if (!install_packages[i] %in% installed.packages()){
    install.packages(install_packages[i])
  }
}
library(knitr)
library(data.table)
library(ggplot2)
library(gridExtra)
library(survminer)
library(survival)
library(corrplot)
library(scales)
library(purrr)

############################################################
# figure 2.3: Number of liver transplantations per year
############################################################
ggplot(data = unos, aes(x = transplantyear)) +
  geom_histogram(aes(fill = ..count..), bins = 30) +
  scale_x_continuous(name ="Year of transplantation",
                     breaks = seq(2005, 2015, by = 1)) +
                     scale_y_continuous(name ="Frequency",
                     breaks = seq(0, 6000, by = 1000)) +
  ylab("Frequency") +
  geom_hline(yintercept = 5500, linetype="dotted") +
  ggtitle(" ") +
  theme(panel.background = element_rect(fill = "white"))


vars <- colnames(unos)
```

```
factors_index <- sapply(vars, function(x) is.factor(unos[, x]))

# names for the categorical variables: nominal, polytomous, discrete
categorical_vars <- vars[factors_index] # 83 categorical variables

multilevel_index <- sapply(categorical_vars, function(x)
                           length(levels(unos[, x])) > 2)
# from them 11 are polytomous
multilevel_vars <- categorical_vars[multilevel_index]
# 72 dichotomous
dichotomous_vars <- categorical_vars[!multilevel_index]
summary(unos[, multilevel_vars])

# dichotomous variables with levels different than No, Yes
# (Negative - Positive)
dichotomous_cat1 <- c("don_gender", "don_ethnicity_srtr",
                      "donortype", "recipientsex",
                      "albumin2", "split", "lifesupport",
                      "can_ethnicity_srtr", "retransplantation")
summary(unos[, dichotomous_cat1])

# make description for dichotomous variables
# (levels: Negative, Positive)
dichotomous_cat2 <- c("don_anti_cmv", "don_anti_hcv",
                      "don_hcv_stat", "don_hbc_stat",
                      "rec_cmv_stat", "rec_hbv_surf_antigen")
summary(unos[, dichotomous_cat2])

# make description for dichotomous variables (levels: NO, YES)

dichotomous_cat3 <- dichotomous_vars[!(dichotomous_vars %in%
                                       c(dichotomous_cat1,
                                        dichotomous_cat2))]

summaries_cat3 = lapply(dichotomous_cat3,
                        function(x) summary(unos[, x]))
s0 = do.call("rbind", summaries_cat3)
# correct NA column
for (i in 1:nrow(s0)) {
  s0[i, 3] <- ifelse(as.numeric(s0[i , 1]) == as.numeric(s0[i, 3]),
                     0, s0[i, 3])
}
##############################################
# Create table 2.2: Binary risk factors
##############################################
s0 <-  cbind(dichotomous_cat3, s0)
s0 <- cbind(s0[1:29, ], s0[c(30:57,1), ])
s0[29, 5:8] <- " " # to erase the repeated value
tbl0 = kable(
  s0, col.names = rep(c("Variable", "No", "Yes", "Missing"), 2),
  format = "latex", booktabs = T,
```

```
  digits = 1, align = "c",
  caption = paste0("label{tab:descriptives_dichotomous}",
                   "Binary risk factors with No-Yes levels.")
)

# make a description table for the continuous variables
# 23 continuous predictors in total
continuous_vars <- vars[!factors_index][1:23]

var_names <- c("Donor age", "Donor height",
               "Donor weight", "Donor BMI",
               "Donor log-SGOT", "Donor serum creatinine",
               "Donor log-BUN", "Donor total bilirubin",
               "Donor log-SGPT", "Donor serum sodium",
               "Donor INR", "Donor hematocrit", "Patient age",
             "Patient cold ischemic time", "Patient length of stay",
             "Patient BMI", "Patient warm ischemic time",
             "Patient last bilirubin", "Patient last INR",
         "Patient last serum creatinine", "Patient last serum sodium",
               "Patient height", "Patient weight")
summaries = lapply(continuous_vars, function(x) summary(unos[, x]))
s = do.call("rbind", summaries)
s = round(s, digits = 2)
# correct NA column
for (i in 1:nrow(s)) {
  s[i, 7] <- ifelse(as.numeric(s[i , 1]) == as.numeric(s[i, 7]), 0,
                    round(s[i, 7], digits = 0))
}
rownames(s) = var_names
colnames(s) = c("Min.", "1st Qu.", "Median", "Mean",
                "3rd Qu.", "Max.", "Missing")

##########################################################
# table 2.1: Descriptives for the continuous variables
##########################################################
tbl = kable(
  s,
  format = "latex", booktabs = T,
  digits = 1, align = "c",
  caption = paste0("label{tab:descriptives_continuous}",
                   "Descriptives for the continuous variables.")
)

# Kaplan- Meier for outcomes of interest

f1 <- survfit(Surv(patientsurvival, death) ~ 1, data = unos)
f2 <- survfit(Surv(gs_ffs, gs_ffsstate) ~ 1, data = unos)
fits <- list(overall = f1, ffs = f2)

# function that returns last element of a vector
last <- function(x) { return( x[length(x)] ) }
```

```
# for overall survival
sum <- summary(f1)
last(sum$surv[sum$time < 1]) # 88.5 survives 1 year
# confidence intervals
last(sum$lower[sum$time < 1])
last(sum$upper[sum$time < 1])

last(sum$surv[sum$time < 3]) # 79.9% survives 3 years
last(sum$lower[sum$time < 3])
last(sum$upper[sum$time < 3])

last(sum$surv[sum$time < 5]) # 73.3% survives 5 years
last(sum$lower[sum$time < 5])
last(sum$upper[sum$time < 5])

# for failure-free survival
sum2 <- summary(f2)
last(sum2$surv[sum2$time < 1]) # 86.3 survives 1 year
last(sum2$lower[sum2$time < 1]) # 86.3 survives 1 year
last(sum2$upper[sum2$time < 1]) # 86.3 survives 1 year

last(sum2$surv[sum2$time < 3]) # 77.4% survives 3 years
last(sum2$lower[sum2$time < 3])
last(sum2$upper[sum2$time < 3])

last(sum2$surv[sum2$time < 5]) # 70.7% survives 5 years
last(sum2$lower[sum2$time < 5])
last(sum2$upper[sum2$time < 5])

# Using survminer
ggsurvplot(
  f1,
  size = 0.5,                 # change line size
  palette = "red",# custom color palettes
  conf.int = TRUE,          # Add confidence interval
  risk.table = TRUE,        # Add risk table
  censor = FALSE,
  pval = FALSE,
  risk.table.col = "strata",
  xlab = "Time in years since transplantation",
  ylab = "Overall survival probability",
  break.time.by = 1,     # break X axis in time intervals by 1 year
  risk.table.height = 0.15, # to adjust height
  ggtheme = theme_bw()      # Change ggplot2 theme
)

##############################################################
# figure 2.4: Kaplan-Meier survival curves for OS and FFS
##############################################################
ggsurvplot(
```

```
  f2,
  size = 0.5,                # change line size
  palette = "blue2",         # custom color palettes
  conf.int = TRUE,           # Add confidence interval
  risk.table = TRUE,         # Add risk table
  censor = FALSE,
  pval = FALSE,
  risk.table.col = "strata", # Risk table color by groups
  xlab = "Time in years since transplantation",
  ylab = "Failure-free survival probability",
  break.time.by = 1,     # break X axis in time intervals by 1 year
  risk.table.height = 0.15,
  ggtheme = theme_bw()      # Change ggplot2 theme
)

grid.arrange(plot1, plot2, nrow = 1, ncol = 2)

#####################################################################
#Figure 2.5: Histograms for age of the donor and the patient
#####################################################################
plot1 <- ggplot(unos, aes(x = donorage)) +
  geom_histogram(aes(y =..density..), bins = 25,
  fill=I("lightblue"),
  color = I("blue")) +
  scale_x_continuous(breaks = seq(0, 90, by = 10)) +
  labs(x = "Age of the donor in years") +
  geom_density(color = 2)

plot2 <- ggplot(unos, aes(x = recipientage)) +
  geom_histogram(aes(y =..density..), bins = 25,
  fill=I("lightblue"),
  color = I("blue")) +
  scale_x_continuous(breaks = seq(0, 90, by = 10)) +
  labs(x = "Age of the patient in years") +
  geom_density(color = 2)

grid.arrange(plot1, plot2, nrow = 1, ncol = 2, top = " ")

#####################################################################
#Figure 2.6: Histograms for Body Mass Index for donors-patients
#####################################################################
plot3 <- ggplot(unos, aes(x = don_bmi)) +
  geom_histogram(aes(y =..density..), bins = 25,
  fill=I("lightblue"), color = I("blue")) +
  scale_x_continuous(breaks = seq(10, 50, by = 5)) +
  xlim(8, 55) + labs(x = "Body mass index (BMI) of the donor ") +
  geom_density(color = 2)

plot4 <- ggplot(unos, aes(x = rec_bmi)) +
  geom_histogram(aes(y =..density..), bins = 25,
  fill=I("lightblue"), color = I("blue")) +
```

```
  scale_x_continuous(breaks = seq(10, 50, by = 5)) +
  xlim(8, 55) + labs(x = "Body mass index (BMI) of the patient") +
  geom_density(color = 2)

grid.arrange(plot3, plot4, nrow = 1, ncol = 2, top = " ")

################################################
# pie charts: Figures 2.9 - 2.11
################################################
# for donor's gender
value <- as.numeric(round(table(unos$don_gender,
                    useNA = "ifany")*100 /
                    length(unos$don_gender), digits = 2))
lbls <- as.factor(paste(value,"%", sep=""))
Donor_gender <- c("Female", "Male")

pie <- ggplot(data = NULL , aes(x = "", y = value,
              fill = Donor_gender))
# make a bar chart proportional to the number of cases
pie <- pie + geom_bar(width = 1,
                      stat = "identity",
                      position = "stack")
pie <- pie + coord_polar (theta = "y", start = 0)
pie <- pie + guides (fill = guide_legend (override.aes =
list (colour = NA), title = "Gender of donor"))
# to disappear unnecessary layout
pie <- pie + theme(panel.background = element_rect(fill = "white"),
                axis.title = element_blank() ,
                axis.ticks = element_blank ())
pie <- pie + scale_y_continuous(breaks = c(15, 80),
label = rev(levels(lbls)))

# for recipient's gender
value2 <- as.numeric(round(table(unos$recipientsex,
useNA = "ifany")*100 / length(unos$recipientsex), digits = 2))
lbls2 <- as.factor(paste(value2,"%", sep=""))

Patient_gender <- c("Male", "Female")
pie2 <- ggplot (data = NULL, aes(x = "", y = value2,
fill = Patient_gender))
# make a bar chart proportional to the number of cases
pie2 <- pie2 + geom_bar(width = 1, stat = "identity",
                        position = "stack")
pie2 <- pie2 + coord_polar(theta = "y", start = 0)
pie2 <- pie2 + guides (fill = guide_legend (override.aes =
list (colour = NA), title = "Gender of patient"))
pie2 <- pie2 + theme(panel.background =
                    element_rect(fill = "white"),
                    axis.title = element_blank() ,
                    axis.ticks = element_blank ())
pie2 <- pie2 + scale_y_continuous(breaks = c(15, 80),
```

```
label = rev(levels(lbls2)))
grid.arrange(pie, pie2, nrow = 1, ncol = 2, top = " ")

# pie charts for ethnicity
value <- as.numeric(round(table(unos$don_ethnicity_srtr,
useNA = "ifany")*100 / length(unos$don_ethnicity_srtr),
                          digits = 2))
lbls <- as.factor(paste(value,"%", sep = ""))

groups <- c("Latino", "No latino")
pie <- ggplot (data = NULL , aes(x = "", y = value, fill = groups))
pie <- pie + geom_bar(width = 1, stat = "identity",
                      position = "stack")
pie <- pie + coord_polar (theta = "y", start = 0)
pie <- pie + guides (fill = guide_legend (override.aes =
list (colour = NA), title = "Ethnicity of donor"))
pie <- pie + theme(panel.background = element_rect(fill = "white"),
                   axis.title = element_blank() ,
                   axis.ticks = element_blank ())

pie <- pie + scale_y_continuous(breaks = c(45, 90),
                                label = rev(levels(lbls))) +
                                scale_fill_brewer(palette = 3)

# for the recipient
value <- as.numeric(round(table(unos$can_ethnicity_srtr,
useNA = "ifany")*100 / length(unos$can_ethnicity_srtr),
                          digits = 2))
lbls <- as.factor(paste(value,"%", sep = ""))

groups <- c("Latino", "No latino")

pie2 <- ggplot (data = NULL , aes(x = "", y = value, fill = groups))
pie2 <- pie2 + geom_bar(width = 1, stat = "identity",
                        position = "stack")
pie2 <- pie2 + coord_polar (theta = "y", start = 0)
pie2 <- pie2 + guides (fill = guide_legend (override.aes =
list (colour = NA), title = "Ethnicity of patient"))
pie2 <- pie2 + theme(panel.background =
                   element_rect(fill = "white"),
                   axis.title = element_blank() ,
                   axis.ticks = element_blank ())
pie2 <- pie2 + scale_y_continuous(breaks = c(45, 90) ,
label = rev(levels(lbls))) +
scale_fill_brewer(palette = 3)

grid.arrange(pie, pie2, nrow = 1, ncol = 2, top = " ")

# similarly: pie charts for race...

############################################################
```

```
# Figures 2.7-2.8: Bar-plots of cause of death, etiology
############################################################
# re-order levels
reorder_size <- function(x) {
  factor(x, levels = names(sort(table(x), decreasing = TRUE)))
}
# bars are reordered in descending order
# first for donor's cause of death
levels(unos$don_gender)
unos$don_gender <- revalue(unos$don_gender, c("F  " = "Female",
                                              "M  " = "Male"))

plot1 <- ggplot(unos, aes(x = reorder_size(donorcod))) +
  geom_bar(aes(y = (..count..)/sum(..count..))) +
  xlab(" ") +
  scale_y_continuous(labels = scales::percent,
                     name = "Proportion") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

plot2 <- ggplot(unos, aes(x = reorder_size(donorcod))) +
  geom_bar(aes(y = (..count..)/sum(..count..))) +
  xlab(" ") +
  scale_y_continuous(labels = scales::percent,
  name = "Proportion") +
  facet_grid(~ don_gender) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

grid.arrange(plot1, plot2, nrow = 1, ncol = 2, top = " ")

# for patient's etiology
plot3 <- ggplot(unos, aes(x = reorder_size(etiology))) +
  geom_bar(aes(y = (..count..)/sum(..count..))) +
  xlab(" ") +
  scale_y_continuous(labels = scales::percent,
  name = "Proportion") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

plot4 <- ggplot(unos, aes(x = reorder_size(etiology))) +
  geom_bar(aes(y = (..count..)/sum(..count..))) +
  xlab(" ") +
  scale_y_continuous(labels = scales::percent,
  name = "Proportion") +
  facet_grid(~ recipientsex) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

grid.arrange(plot3, plot4, nrow = 1, ncol = 2, top = " ")
```

D.1.3 Missing values - Imputation technique

```
install_packages <- c("mice", "randomForestSRC", "Amelia",
                      "UpSetR", "VIM")
```

```
for (i in 1:length(install_packages)){
  if (!install_packages[i] %in% installed.packages()){
    install.packages(install_packages[i])
  }
}
library(mice)
library(randomForestSRC)
library(Amelia)
library(UpSetR)
library(VIM)

# missingness per row
which.max(rowSums(is.na(unos)))

# sum of NA in the full data set is 2.5%
sum(is.na(unos_selected)) /
(nrow(unos_selected)*ncol(unos_selected))

colSums(is.na(unos_selected)) # missing values per variable
colSums(is.na(unos_selected))[colSums(is.na(unos_selected))
                                        > 10000 ]
# 15 variables with more than 10000 NA
colSums(is.na(unos_selected)) < 10000]
colSums(is.na(unos_selected))[colSums(is.na(unos_selected)) > 0 &
colSums(is.na(unos_selected)) < 1000]

# show which variables have missing values out of the 110
vars_mis <- colnames(unos_selected)[sapply(colnames(unos_selected),
function(x) sum(is.na(unos_selected[, x]))) > 0]

################################################
# Figure 2.12: Percentage of missingness
# per variable with more than 1% missing
################################################
gg_miss_var(unos_selected[, vars_mis_greater1], show_pct = TRUE) +
  labs(y = "Percentage of missingness ") +
  scale_y_continuous(breaks = seq(0, 25, 5))
gg_miss_case(unos_selected) + labs(x = "Number of Cases")

################################################
# Figure 2.13: Patterns of missing values for
# the 5 highest missing variables
################################################
unos_selected %>%
as_shadow_upset() %>%
upset()

#################################
# Sensitivity analysis
#################################
```

```
unos1 <- read.table("unos_selected.txt")
unos_trial <- unos1[complete.cases(unos1), ]
# sum of NA in the full data set is 2.5%
sum(is.na(unos1)) / (nrow(unos1)*ncol(unos1))
unos_complete <- unos_trial # duplicate the complete dataset

# measures for continuous: Normalized RMSE
# for categorical : proportion of falsely classified entries (PFC)
# experiment with 4 different proportions of missingness
props <- c(0.05, 0.10, 0.20, 0.30)

set.seed(1)
index1 <- sample(1:nrow(unos_trial),
                 round(nrow(unos_trial)*props[1],
                 digits = 0), replace = FALSE)
index2 <- sample(1:nrow(unos_trial),
                 round(nrow(unos_trial)*props[1],
                 digits = 0), replace = FALSE)
index3 <- sample(1:nrow(unos_trial),
                 round(nrow(unos_trial)*props[1],
                 digits = 0), replace = FALSE)
index4 <- sample(1:nrow(unos_trial),
                 round(nrow(unos_trial)*props[1],
                 digits = 0), replace = FALSE)
index5 <- sample(1:nrow(unos_trial),
                 round(nrow(unos_trial)*props[1],
                 digits = 0), replace = FALSE)

unos_trial$portal_hyperten_bleed[index1] <- NA
unos_trial$rec_tumor[index2] <- NA
unos_trial$can_drug_treat_hyperten[index3] <- NA
unos_trial$rec_warm_isch_tm[index4] <- NA
unos_trial$rec_postx_los[index5] <- NA

### package MICE
mice_time_5p <- system.time(
mice_obj_5p <- mice(data = unos_trial, m = 1,
                    maxit = 5, seed = 12345)
)
# time elapsed mice_time_5p / 60

complete_mice_5p <-  mice::complete(mice_obj_5p, action = 1)

# check prediction error, misclassifications for MICE
# for portal_hyperten_bleed
m1 <- (1 - sum(diag(table(unos_complete$portal_hyperten_bleed,
                          complete_mice_5p$portal_hyperten_bleed) /
                          nrow(unos_complete)))) / props[1]
# for rec_tumor
m2 <- (1 - sum(diag(table(unos_complete$rec_tumor,
complete_mice_5p$rec_tumor) / nrow(unos_complete)))) / props[1]
```

```
# for can_drug_treat_hyperten
m3 <- (1 - sum(diag(table(unos_complete$can_drug_treat_hyperten,
                          complete_mice_5p$can_drug_treat_hyperten)
                          / nrow(unos_complete)))) / props[1]
# for rec_warm_isch_tm
m4 <- sqrt(mean((unos_complete$rec_warm_isch_tm -
complete_mice_5p$rec_warm_isch_tm)^2) /
var(unos_complete$rec_warm_isch_tm))
# for rec_postx_los
m5 <- sqrt(mean((unos_complete$rec_postx_los -
           complete_mice_5p$rec_postx_los)^2) /
           var(unos_complete$rec_postx_los))

m_5p <- c(m1, m2, m3, m4, m5)

# repeat imputations with package randomForestSRC
# using unsupervised splitting, original missForest algorithm
time_rf_5p <- system.time(
  complete_rf_5p <- impute.rfsrc(data = unos_trial, ntree = 150,
  mf.q = 1, max.iter = 5, nimpute = 1, do.trace = FALSE)
)

# repeat imputations with package randomForestSRC
# using unsupervised splitting, original missForest algorithm
time_rf_5p <- system.time(
  complete_rf_5p <- impute.rfsrc(data = unos_trial,
                                 ntree = 150, mf.q = 1,
                                 max.iter = 5, nimpute = 1,
                                 do.trace = FALSE)
)

# time elapsed time_rf_5p / 60

# for portal_hyperten_bleed
rf1 <- (1 - sum(diag(table(unos_complete$portal_hyperten_bleed,
complete_rf_5p$portal_hyperten_bleed) / nrow(unos_complete))))
/ props[1]
# for rec_tumor
rf2 <- (1 - sum(diag(table(unos_complete$rec_tumor,
complete_rf_5p$rec_tumor) / nrow(unos_complete)))) / props[1]
# for can_drug_treat_hyperten
rf3 <- (1 - sum(diag(table(unos_complete$can_drug_treat_hyperten,
                           complete_rf_5p$can_drug_treat_hyperten) /
                           nrow(unos_complete)))) / props[1]
# for rec_warm_isch_tm
rf4 <- sqrt(mean((unos_complete$rec_warm_isch_tm -
complete_rf_5p$rec_warm_isch_tm)^2)
/ var(unos_complete$rec_warm_isch_tm))

# for rec_postx_los
rf5 <- sqrt(mean((unos_complete$rec_postx_los -
```

```
complete_rf_5p$rec_postx_los)^2) / var(unos_complete$rec_postx_los))
rf_5p <- c(rf1, rf2, rf3, rf4, rf5)


### package AMELIA
noms <- names(Filter(is.factor, unos_trial))
complete_am_5p <- amelia(unos_trial, m = 1, p2s = 1,
noms = noms, frontend = FALSE, parallel = "snow")

complete_am_5p <- complete_am_5p$imputations$imp1

# for portal_hyperten_bleed
am1 <- (1 - sum(diag(table(unos_complete$portal_hyperten_bleed,
complete_am_5p$portal_hyperten_bleed) / nrow(unos_complete))))
/ props[1]
# for rec_tumor
am2 <- (1 - sum(diag(table(unos_complete$rec_tumor,
complete_am_5p$rec_tumor) / nrow(unos_complete)))) / props[1]
# for can_drug_treat_hyperten
am3 <- (1 - sum(diag(table(unos_complete$can_drug_treat_hyperten,
                           complete_am_5p$can_drug_treat_hyperten) /
                           nrow(unos_complete)))) / props[1]
# for rec_warm_isch_tm
am4 <- sqrt(mean((unos_complete$rec_warm_isch_tm -
complete_am_5p$rec_warm_isch_tm)^2) / var(unos_complete$rec_warm_isch_tm))

# for rec_postx_los
am5 <- sqrt(mean((unos_complete$rec_postx_los -
complete_am_5p$rec_postx_los)^2) / var(unos_complete$rec_postx_los))
am_5p <- c(am1, am2, am3, am4, am5)

sensitivity_5p <-  as.data.frame(cbind(MICE_error_5p = m_5p,
                                RANDOMFOREST_error_5p = rf_5p,
                                AMELIA_error_5p = am_5p))
rownames(sensitivity_5p) <- c("portal_hyperten_bleed", "rec_tumor",
"can_drug_treat_hyperten", "rec_warm_isch_tm", "rec_postx_los")
capture.output(print(sensitivity_5p), file = "results_5p.txt")

# Code is similar for the other proportions of missingness

#################################
# Data imputation
#################################

# check Nelson-Aalen estimates of cumulative hazard as a replacement
# of patient surviva;
# for overall survival
H0_t <- nelsonaalen(data = unos_selected,
                    timevar = patientsurvival,
                    statusvar = death)
cor(cbind(H0_t, Survival = unos_selected$patientsurvival))
```

```
# 0.989 correlation

# for failure-free survival
H0_t2 <- nelsonaalen(data = unos_selected,
                     timevar = gs_ffs,
                     statusvar = gs_ffsstate)
cor(cbind(H0_t2, Survival = unos_selected$gs_ffs))
# 0.988 correlation
# correlation almost 1, so for these data it matters little
# whether we take Ho(t) or T as a predictor.

#############################################################
#############################################################
# Original missForest algorithm for data imputation using
# unsupervised splitting.
# We grow a forest to impute the data. To proceed split statistics
# are calculated. If a node splits on a variable with
# missing data, the variable's missing data is imputed by randomly
# drawing values from non-missing in-bag data. The purpose of this
# is to make it possible to assign cases to daughter nodes based
# on the split. We used all possible variable combinations as
# responses, and split by the rest of the variables using
# multivariate composite splitting.
# Missing data for responses are imputed by prediction.
# The process is repeated using a new set of variables for responses
# (mutually exclusive to the previous fit), until all variables
# have been imputed. The procedure is repeated until convergence
# of the algorithm. Maximum number of iterations was set to 5.

# This is the most accurate of all imputation procedures that the
# randomForestSCR offers, but also by far the most
# computationally expensive one.
set.seed(12345)
time_rf_imput <- system.time(
  unos_complete <- impute.rfsrc(data = unos_selected,
                                ntree = 150, mf.q = 1,
                                max.iter = 5, do.trace = TRUE)
)

time_rf_imput / 3600 # elapsed time 8.50 hours

#############################################################
# possible diagnostics after imputations
#############################################################

mat1 <-
cbind(unos_selected$rec_warm_isch_tm[
      !is.na(unos_selected$rec_warm_isch_tm)],
      "observed")
mat2 <-
cbind(unos_complete$rec_warm_isch_tm[
```

```
which(is.na(unos_selected$rec_warm_isch_tm))], "imputed")
df <- as.data.frame(rbind(mat1, mat2))
df$V1 <- as.numeric(as.character(df$V1))
df$V2 <- as.factor(df$V2)
colnames(df) <- c("rec_warm_isch_tm", "Data")

#################################################################
# Figure 2.14: Distributions of observed and imputed data
#################################################################
plot1 <- ggplot(df, aes(rec_warm_isch_tm, colour = Data)) +
  geom_density() +
  xlab("Estimated Warm Ischemic Time in minutes") +
  xlim(c(0, 100)) +
  theme_minimal() +
  ggtitle("  ")


mat3 <- cbind(unos_selected$rec_postx_los[
!is.na(unos_selected$rec_postx_los)],
              "observed")
mat4 <- cbind(unos_complete$rec_postx_los[which(
is.na(unos_selected$rec_postx_los))],
              "imputed")

df2 <- as.data.frame(rbind(mat3, mat4))
df2$V1 <- as.numeric(as.character(df2$V1))
df2$V2 <- as.factor(df2$V2)
colnames(df2) <- c("rec_postx_los", "Data")

plot2 <- ggplot(df2, aes(rec_postx_los, colour = Data)) +
  geom_density(adjust = 1.5) +
  xlab("Stay after transplantation to the hospital in days") +
  xlim(c(0, 85)) +
  theme_minimal() +
  ggtitle(" ")

gridExtra::grid.arrange(plot1, plot2, ncol = 2)
```

D.2 Application

For all parts of analysis, the R code for failure-free survival is similar to that of overall survival. It can be implemented by replacing unos_overall by unos_failure_free, overall by ffs, ovr by ffs and also patientsurvival by gs_ffs, death by gs_ffsstate.

D.2.1 Cox proportional hazards models

Backward elimination

```
install_packages <- c("survival", "survminer", "rms", "ggplot2",
                      "grid", "gridExtra", "pec", "forestmodel")
```

```
for (i in 1:length(install_packages)){
  if (!install_packages[i] %in% installed.packages()){
    install.packages(install_packages[i])
  }
}

library(survival)
library(survminer)
library(rms)
library(ggplot2)
library(grid)
library(gridExtra)
library(pec)
library(forestmodel)
source("https://bioconductor.org/biocLite.R")
biocLite("survcomp")
library(survcomp)

load("unos_overall.RData")
unos_overall$patientsurvival <- unos_overall$patientsurvival +
                                (1/365)

set.seed(12345)
N <- nrow(unos_overall)
index <- sample(1:N, round(N/3), replace = FALSE)
training_overall <- unos_overall[-index, ] # create training set
test_overall <- unos_overall[index, ] # create test set
# quantiles of the training set
quantile(training_overall$patientsurvival)

#################################################################
# table B1: Cox proportional hazards model with all prognostic
# variables
#################################################################

# fit Cox full model
cox_full <- coxph(Surv(patientsurvival, death) ~.,
                  data = training_overall, x = TRUE, y = TRUE)

sort(cox_full$coefficients)

summ <- summary(cox_full)
df_cox_full <- round(summ$conf.int[, c(1, 3, 4)], 3)
df_cox_full1 <- df_cox_full[1:64, ]
df_cox_full2 <- df_cox_full[65:128, ]

names1 <- rownames(df_cox_full1)
names2 <- rownames(df_cox_full2)
rownames(df_cox_full1) <- NULL
rownames(df_cox_full2) <- NULL
```

```
df <- data.frame(Variable = names1, exp_coef = df_cox_full1[, 1],
                 lower95 = df_cox_full1[, 2],
                 upper95 = df_cox_full1[, 3],
                 Variable = names2, exp_coef = df_cox_full2[, 1],
                 lower95 = df_cox_full2[, 2],
                 upper95 = df_cox_full2[, 3])

kable(
  df,
  col.names = c("Variable", "exp(coef)", "lower .95",
                "upper .95", "Variable", "exp(coef)",
                "lower .95", "upper .95"),
  format = "latex", booktabs = T,
  digits = 3, align = "c",
  caption = paste0("label{tab:app_cox_full_ovr1}")
)


# Stepwise regression methods:

# Forward selection, which starts with no predictors in the model,
# iteratively adds the most contributive predictors,
# and stops when the improvement is no longer stat significant.
# Backward selection (or backward elimination), starts with all
# predictors in the model (full model), iteratively removes the
# least contributive predictors, and stops when you have a model
# where all predictors are statistically significant.
# Stepwise selection (or sequential replacement), a combination
# of forward and backward selections.
# You start with no predictors,
# sequentially add the most contributive predictors (like forward
# selection). After adding each new variable, remove any variables
# who no longer provide an improvement in the model fit.


cox_null <- coxph(Surv(patientsurvival, death) ~ 1,
                  x = TRUE, y = TRUE,
                  data = training_overall, method = "breslow")

### Backward elimination model
time_backward <- system.time(
  step_model <- stepAIC(cox_full, direction = "backward",
                        trace = TRUE)
)
summ_step <- summary(step_model)
# summ_step$concordance
# summ_step$call # 59 variables were selected
# time_backward / 3600
# AUC selected 59 predictors
impor_vars_AUC_back <- summ_step$call
```

```
# Forward-backward model selection are two greedy approaches
# to solve the combinatorial optimization problem of finding the
# optimal combination of features.
# You need to look for suboptimal, computationally efficient
# strategies. In Forward method you start with an empty model,
# and iterate over features. For each feature you train a model,
# and select feature which yields the best model according to
# your metric. In a similar  fashion you proceed by adding the
# next feature that yield the best
# improvement when combined with the already selected ones.
# In backward method you just invert the procedure: start with all
# features, and iteratively remove that one whose removal least
# affects performance, or leads to the biggest improvement.

### Forward selection model
time_forward <- system.time(
  step_model2 <- stepAIC(cox_null, direction="forward",
                         trace = FALSE,
                         scope = list(upper = cox_full,
                         lower = cox_null))
)
time_forward / 3600 # time to compute 1.10 hours
summ_step2 <- summary(step_model2)
summ_step2$call # it kept 55 variables
formula_AUC_forw <- summ_step2$call

### backward stepwise variable selection
# first create a cpc object
null_model <- cph(Surv(patientsurvival, death) ~ 1,
                  data = training_overall, method = "breslow")
full_model <- cph(Surv(patientsurvival, death) ~.,
                  data = training_overall, method = "breslow")

# fastbw() with AIC rule finds 28 variables
step_model3 <- fastbw(fit = full_model, rule = "aic")
# fastbw() with p-value rule finds 40 variables
step_model4 <- fastbw(fit = full_model, rule = "p")

# pec package has a built in function for fastbw()
# Backward variable selection in the Cox regression model
cox_back <- selectCox(formula = Surv(patientsurvival, death) ~.,
                      data = training_overall, rule = "aic")
class(cox_back)
cox_back$In

### create formula for coxph
vars_back <- cox_back$In # variables
f_back <- as.formula(paste("Surv(patientsurvival, death) ~",
                           paste(vars_back, collapse = "+")))
```

```
cox_back <- coxph(formula = f_back, x = TRUE, y = TRUE,
                  data = training_overall, method = "breslow")

attr(cox_back$terms, "term.labels")
temp <- cox.zph(cox_back)

#########################################################
# Figure 5.1: Forest plot for Cox backward model OS
#########################################################
library(forestmodel)
forest_model(cox_back, recalculate_width = F,
             recalculate_height = 10)

# graphical test of proportional hazards
# Systematic departures from a horizontal line are indicative
# of non-proportional hazards
temp <- cox.zph(cox_back)

# find the linear predictors for cox backward
pred_train_back <- predict(cox_back, type = "lp")
pred_test_back <- predict(cox_back,
                          newdata = test_overall[, -c(107, 108)],
                          type = "lp")

conc_back <-  concordance.index(pred_test_back,
                                surv.time =
                                test_overall$patientsurvival,
                                surv.event = test_overall$death)
conc_back$c.index
conc_back$lower
conc_back$upper

# split the linear predictors into 4 groups with equal patients
groups_train_back <- factor(cut(pred_train_back, c(-Inf,
                            quantile(pred_train_back, 0.25),
                            quantile(pred_train_back, 0.5),
                            quantile(pred_train_back, 0.75),
                            Inf)),
labels=c("Group1", "Group2", "Group3", "Group4"))

#############################################################
# Figure 5.2: Survival curves for 4 groups of linear predictors
#############################################################

groups_test_back <- factor(cut(pred_test_back, c(-Inf,
                           quantile(pred_train_back, 0.25),
                           quantile(pred_train_back, 0.5),
                           quantile(pred_train_back, 0.75),
                           Inf)),
labels=c("Group1", "Group2", "Group3", "Group4"))
```

```
fit_model4 <- survfit(Surv(patientsurvival, death) ~
                      groups_test_back,
                      data = test_overall)

ggsurvplot(
  fit_model4,
  size = 0.5,                # change line size
  palette = "jco",           # nice palettes are jco, uchicago
  conf.int = TRUE,           # Add confidence interval
  risk.table = FALSE,        # Add risk table
  censor = FALSE,
  pval = TRUE,
  risk.table.col = "strata", # Risk table color by groups
  xlab = "Time in years since transplantation",
  ylab = "Survival probability",
  break.time.by = 1,
  legend.labs = c("Group1", "Group2", "Group3", "Group4"),
  risk.table.height = 0.35,
  surv.median.line = "hv",
  ggtheme = theme_classic()     # Change ggplot2 theme

)

res2 <- surv_pvalue(fit_model4)
```

Selection with LASSO

```
install_packages <- c("survival", "glmnet", "ggplot2",
                      "gridExtra", "dplyr", "tidyverse")

for (i in 1:length(install_packages)){
  if (!install_packages[i] %in% installed.packages()){
    install.packages(install_packages[i])
  }
}
library(survival)
library(glmnet)
library(ggplot2)
library(gridExtra)
#library(dplyr)
#library(tidyverse)
#source("https://bioconductor.org/biocLite.R")
#biocLite("survcomp")
library(survcomp)

response_pair <- c("patientsurvival", "death")

# for the training set
X_train <- training_overall[,
!(colnames(training_overall) %in% response_pair)]
# create martix X for glmnet
X_train_ovr <- model.matrix(~., X_train)[, -1]
# dim(X_train_ovr) # 41529 x 129
# create matrix Y
Y_train_ovr <- training_overall[,
colnames(training_overall) %in% response_pair]
Y_survtrain_ovr <- Surv(Y_train_ovr$patientsurvival,
                        Y_train_ovr$death)

# for the test set
x_test <- test_overall[,
                !(colnames(test_overall) %in% response_pair)]
X_test_ovr <- model.matrix(~., x_test)[, -1]
Y_test_ovr <- test_overall[,
                  colnames(test_overall) %in% response_pair]
Y_survtest_ovr <- Surv(Y_test_ovr$patientsurvival,
                       Y_test_ovr$death)

# use these two datasets to fit Cox model in the selected features
training_ovr_extended <- as.data.frame(cbind(X_train_ovr,
                                             Y_train_ovr))
test_ovr_extended <- as.data.frame(cbind(X_test_ovr, Y_test_ovr))

library(doParallel)
if (!exists("cl")) {
```

```
  cl <- makeCluster(3)
  registerDoParallel(cl)
}
# getDoParWorkers()
# getDoParName()
# stopCluster(cl)

seed <- 12345
set.seed(seed)
folds <- createFolds(y = training_overall$patientsurvival,
                     k = 5, list = FALSE)

time_cv <- system.time(
  cv.fit <- cv.glmnet(x = X_train_ovr, y = Y_survtrain_ovr,
                      foldid = folds, parallel = TRUE,
                      family = "cox", grouped = TRUE,
                      maxit = 1000)
)
cv.fit$lambda.min  # value of lambda that gives minimum cvm
cv.fit$lambda.1se
plot(cv.fit$glmnet.fit)
plot(cv.fit$lambda, cv.fit$cvm)
plot(cv.fit)

############################################################
# Figure 5.5: Variable selection with LASSO (OS)
############################################################
# left vertical line: where cv error curve hits the minimum
# right vertical line: the most regularized model with cv-error
# within 1-sd of minimum
plot(cv.fit, sign.lambda = -1 )

active_index <- as.numeric(unlist(predict(cv.fit,
                                  newx = X_test_ovr,
                                  s = "lambda.1se",
                                  type = "nonzero")))
names_coefs <- colnames(test_ovr_extended)[1:128]
vars_active <- names_coefs[active_index]
coefficients <- coef(fit, s = cv.fit$lambda.1se)
# str(coefficients)
# available coefficients for test set
names_coefs <- unlist(coefficients@Dimnames[1])
# corresponding to lambda = cv.fit$lambda.min.
active_index <- which(coefficients != 0)
active_coefs <- coefficients[active_index]
df <- data.frame(Variable = vars_active,
                 Coef = round(active_coefs, 2),
                 Exp_Coef = round(exp(active_coefs), 2))
rownames(df) <- NULL

# Fit a generalized linear model via penalized maximum likelihood,
```

```
# alpha = 1 means Lasso
fit <- glmnet(x = X_train_ovr, y = Y_survtrain_ovr, alpha = 1,
              #lambda = cv.fit$lambda.1se,
              family = "cox", maxit = 1000)
fit$lambda
plot(fit)

#########################################################
# make a function that repeats the process 250 times
# for lambda_min and lambda_1se
#########################################################
repeats <- 250
var_names <- colnames(training_ovr_extended)[1:128]

lambdas_min <- vector(mode = "numeric", length = repeats)
lambdas_1se <- vector(mode = "numeric", length = repeats)
nr_coefs_min <- vector(mode = "numeric", length = repeats)
nr_coefs_1se <- vector(mode = "numeric", length = repeats)
cindices_min <- vector(mode = "numeric", length = repeats)
cindices_1se <- vector(mode = "numeric", length = repeats)
selection_list_min <- vector(mode = "list", length = repeats)
selection_list_1se <- vector(mode = "list", length = repeats)
seed <- 12345

for (i in 1:repeats) {

  cat("Started iteration i = ", i, "finding lambdas ...",  "\n")

  set.seed(seed + i)
  folds <- createFolds(y = training_overall$patientsurvival,
                       k = 5, list = FALSE)

  cv.fit <- cv.glmnet(x = X_train_ovr, y = Y_survtrain_ovr,
                      foldid = folds, parallel = TRUE,
                      family = "cox", grouped = TRUE,
                      maxit = 1000)
  # alpha = 1 means Lasso
  fit_min <- glmnet(x = X_train_ovr,
                    y = Y_survtrain_ovr, alpha = 1,
                    lambda = cv.fit$lambda.min,
                    family = "cox", maxit = 1000)

  fit_1se <- glmnet(x = X_train_ovr,
                    y = Y_survtrain_ovr, alpha = 1,
                    lambda = cv.fit$lambda.1se,
                    family = "cox", maxit = 1000)

  coefficients_min <- coef(fit_min, s = cv.fit$lambda.min)
  coefficients_1se <- coef(fit_1se, s = cv.fit$lambda.1se)

  selection_list_min[[i]] <-
```

```
  var_names[as.vector(coefficients_min@i)]
  selection_list_1se[[i]] <-
  var_names[as.vector(coefficients_1se@i)]

  lambdas_min[i] <- fit_min$lambda
  lambdas_1se[i] <- fit_1se$lambda
  nr_coefs_min[i] <- length(coefficients_min@x)
  nr_coefs_1se[i] <- length(coefficients_1se@x)
  lp_pred_min <- predict(fit_min, newx = X_train_ovr,
                         s = "lambda.min",
                         type="link") # gives linear predictors
  lp_pred_1se <- predict(fit_1se, newx = X_train_ovr,
                         s = "lambda.1se",
                         type="link") # gives linear predictors

  df_min <- data.frame(time = training_overall$patientsurvival,
                       status = training_overall$death,
                       lp = as.numeric(lp_pred_min))
  df_1se <- data.frame(time = training_overall$patientsurvival,
                       status = training_overall$death,
                       lp = as.numeric(lp_pred_1se))

  cat("Calculating concordance indices... ", "\n")
  cindices_min[i] <- concordance.index(x = df_min$lp,
                                       surv.time = df_min$time,
                                       surv.event =
                                       df_min$status)$`c.index`
  cindices_1se[i] <- concordance.index(x = df_1se$lp,
                                       surv.time = df_1se$time,
                                       surv.event =
                                       df_1se$status)$`c.index`
}

############################################################
# Variables selected lambda_min from 250 repeats
############################################################

# count how many times each variable got selected
sel_unlisted_min <- unlist(selection_list_min)
df_count_min <- as.data.frame(table(sel_unlisted_min))
df_count_min <- df_count_min[order(df_count_min$Freq,
                                   decreasing = TRUE), ]
colnames(df_count_min) <- c("Variable", "Times")

############################################################
# Table 5.1: Variables selected lambda_1se from 250 repeats
############################################################

sel_unlisted_1se <- unlist(selection_list_1se)
df_count_1se <- as.data.frame(table(sel_unlisted_1se))
df_count_1se <- df_count_1se[order(df_count_1se$Freq,
```

```
                                        decreasing = TRUE), ]
colnames(df_count_1se) <- c("Variable", "Times")

res_min <- data.frame(Lambdas = lambdas_min,
                      Nr_coefs = nr_coefs_min,
                      C_index = cindices_min)

res_1se <- data.frame(Lambdas = lambdas_1se,
                      Nr_coefs = nr_coefs_1se,
                      C_index = cindices_1se)
```

D.2.2 Random Survival forests

```
install_packages <- c("survival", "caret", "randomForestSRC",
                      "parallel", "prodlim", "pec")

for (i in 1:length(install_packages)){
  if (!install_packages[i] %in% installed.packages()){
    install.packages(install_packages[i])
  }
}
library(survival)
library(caret)
library(randomForestSRC)
library(parallel)
library(prodlim)
library(pec)
options(rf.cores = 20, mc.cores = 20)

##################################
# 5-fold cross validation
##################################

nodesize <- c(10, 20, 35, 50, 70, 85, 100, 120)
nsplit <- c(3, 4, 5, 6, 7)
mtry <- c(11, 17, 23, 29, 35, 41, 47)
combis <- expand.grid(nodesize, nsplit, mtry)

nfolds <- 5
set.seed(12345)
folds <- createFolds(training_overall$death, k = 5, list = TRUE)
cv_time <- matrix(0, nrow = nfolds, ncol = nrow(combis))
oob_error <- matrix(0, nrow = nfolds, ncol = nrow(combis))


for (i in 1:nfolds) {

  cat("Starting iteration i = ", i, "\n")
  indices <- folds[[i]]
  train_set <- training_overall[-indices, ]
  validation_set <- training_overall[indices, ]

  for (j in 1:nrow(combis)) {

    cat("Working on combination:", "nodesize: ",
        combis[j, 1],
        "nsplit: ", combis[j, 2],
        "and mtry: ", combis[j, 3], "\n")
    cv_time[i, j] <- {
      system.time(
        fit_ovr <- rfsrc(Surv(patientsurvival, death) ~ .,
```

```
        splitrule = "logrank",
                        nodesize = combis[j, 1],
                        nsplit = combis[j, 2],
                        data = train_set, mtry = combis[j, 3],
                        ntree = 500, seed = -12345,
                        ntime = 1000, sampsize = 1000))[3]
    }
    cat("Calculating the OOB validated prediction error ...",
        "\n")
    fit_val <- predict(fit_ovr, newdata = validation_set,
                       seed = -12345)
    oob_error[i, j] <- as.numeric(fit_val$err.rate[fit_val$ntree])
    cat("Error is: ", oob_error[i, j])

  }
}

df_logrank_ovr <- data.frame(Node_size = combis$Var1,
                             Nsplit = combis$Var2,
                             Mtry = combis$Var3,
                             Error = round(colMeans(oob_error),
                             4))

save.image("image_logrank_ovr_cv.Rdata")
cv_time / 60 # time in minutes
mean(cv_time / 60)
df_logrank_ovr[which.min(df_logrank_ovr$Error), ]
sd(df_logrank_ovr$Error)

a <- list(
  title = 'Node size',
  autotick = FALSE,
  ticks = "outside",
  dtick = 10,
  ticklen = 2,
  tickwidth = 1,
  tickcolor = toRGB("blue")
)

b <-  list(
  title = 'Mtry',
  autotick = TRUE,
  ticks = "outside",
  tick0 = -1,
  dtick = 6,
  ticklen = 2,
  tickwidth = 1,
  range = c(10, 47),
  tickcolor = toRGB("blue")
)
c <- list(
```

```
  title = 'Nsplit',
  autotick = FALSE,
  ticks = "outside",
  dtick = 1,
  ticklen = 2,
  tickwidth = 1,
  tickcolor = toRGB("blue")
)

################################################
# Figure 5.7: Grid search on a 3D space for RSF
################################################

#create 3D scatterplot
library(plotly)
plot_ly(df_logrank_ovr, x = ~Node_size, y = ~Mtry, z = ~Nsplit,
        marker = list(color = ~Error,
                      colorscale = "Red", # default is "Reds",
                      try "Viridis"
                      showscale = TRUE)) %>%
  add_markers() %>%
  layout(scene = list(xaxis = a,
                      yaxis = b,
                      zaxis = c),
         annotations = list(
           x = 1.10,
           y = 1.10,
           z = 0.90,
           text = 'mean OOB Error',
           xref = 'paper',
           yref = 'paper',
           showarrow = FALSE
         ))


###################################
# Finding OOB training error
###################################

nodesize <- c(10, 20, 35, 50, 70, 85, 100, 120)
nsplit <- c(3, 4, 5, 6, 7)
mtry <- c(11, 17, 23, 29, 35, 41, 47)
combis <- expand.grid(nodesize, nsplit, mtry)
cv_time <- vector(mode = "numeric", length = nrow(combis))
oob_error <- vector(mode = "numeric", length = nrow(combis))

for (j in 1:nrow(combis)) {

  cat("Working on combination nr:", j, "with nodesize: ",
      combis[j, 1],
      "nsplit: ", combis[j, 2], "and mtry: ",
```

```
      combis[j, 3],  "\n")

  cv_time[j] <- {
    system.time(
      fit_ovr <- rfsrc(Surv(patientsurvival, death) ~ .,
                       splitrule = "logrank",
                       nodesize = combis[j, 1],
                       nsplit = combis[j, 2],
                       data = training_overall,
                       mtry = combis[j, 3],
                       ntree = 500, seed = -12345,
                       sampsize = 5000,
                       ntime = 500, forest = FALSE))[3]
  }

  cat("Calculating the OOB prediction error of combination  ...",
      "\n")
  oob_error[j] <- as.numeric(fit_ovr$err.rate[fit_ovr$ntree])
  cat("Error is: ", oob_error[j], "\n")
}


df_logrank_ovr <- data.frame(Node_size = combis$Var1,
                             Nsplit = combis$Var2,
                             Mtry = combis$Var3,
                             Error = round(oob_error, 4))


##############################################
# Finding OOB error per number of trees
##############################################

fit_block <- rfsrc(Surv(patientsurvival, death) ~ .,
                   splitrule = "logrank", nsplit = 5,
                  data = training_overall, ntree = 500,
                  split.depth = "all.trees",
                  var.used = "all.trees", seed = -12345,
                  mtry = 35, nodesize = 50,
                  ntime = 500, sampsize = 5000, block.size = 5,
                  forest = TRUE
)

jpeg("plot_trees_ovr.jpg")
plot(fit_block)
dev.off()
cat("Plot of number of trees recorded!")

################################
# Fitting final model
################################
```

```
cat("Fitting random forest... ")
# final fit ovr
fit <- rfsrc(Surv(patientsurvival, death) ~ .,
             splitrule = "logrank", nsplit = 5,
             data = training_overall, ntree = 250,
             split.depth = "all.trees",
             var.used = "all.trees", seed = -12345,
             mtry = 35, nodesize = 50,
             ntime = 500, sampsize = 5000, forest = TRUE,
             importance = TRUE
)

fit$err.rate[fit$ntree]

jpeg("plot_oob_mort.png", width = 600)
plot.survival.rfsrc(fit, plots.one.page = FALSE)
dev.off()

##########################################################
# marginal effects of variable rec_immuno_maint_meds
##########################################################
jpeg("plot_mortal_meds.png", width = 600)
plot.variable(fit, xvar.names = "rec_immuno_maint_meds",
              surv.type = "mort")
dev.off()


##########################################################
# marginal effects of variable rec_postx_los
##########################################################
jpeg("plot_mortal_los.png", width = 600)
plot.variable(fit, xvar.names = "rec_postx_los", xlim = c(0, 60),
              surv.type = "mort")
dev.off()


##########################################################
# number of terminal nodes for each tree in the forest
##########################################################
leaf_count <- fit$leaf.count
jpeg("plot_nr_leaves.png")
ggplot(NULL, aes(x = leaf_count)) +
  geom_histogram(aes(y =..density..),
                 bins = 20,
                 fill=I("lightblue"),
                 color = I("blue")) +
  geom_density(color = 2) +
  labs(x = "Number of leaves per tree") +
  theme(plot.title = element_text(hjust = 0.5))
dev.off()
```

```
#####################################################
# most frequent variables used for the forest
#####################################################

vars_used <- tail(sort(fit$var.used), 10)
jpeg("plot_var_used.png", width = 750)
barplot(tail(sort(fit$var.used), 5), ylab = "Frequency")
dev.off()

# very important variables are used early
jpeg("plot_split_depth.png", width = 650)
hist(fit$split.depth, xlab = "Split depth per variable",
     col = "red", main = "", breaks = 20)
dev.off()

cat("play with node size to see change in node depth")
node_size <- c(10, 20, 30, 40, 50, 60, 70, 80)
forest_depth <- matrix(0, nrow = length(node_size), ncol = 106)
colnames(forest_depth) <- colnames(training_overall[1:106])
error <- vector(mode = "numeric", length = length(node_size))


for (i in 1:length(node_size)){

  cat("Starting iteration i = ", i, "with node size = ",
      node_size[i], "\n")
  fit_depth <- rfsrc(Surv(patientsurvival, death) ~ .,
                     splitrule = "logrank",
                     nsplit = 5, data = training_overall,
                     ntree = 250,
                     nodesize = node_size[i],
                     seed = -12345, mtry = 35,
                     split.depth = "all.trees",
                     ntime = 500, sampsize = 5000, forest = FALSE
  )

  forest_depth[i, ] <- colMeans(fit_depth$split.depth)
  error[i] <- fit_depth$err.rate[fit_depth$ntree]
  cat("Error rate is: ", error[i], "\n")

}

forest_depth <- as.data.frame(forest_depth)
forest_depth$rec_postx_los
rm(fit_depth)

#####################################################
# Most important variables using minimal depth
#####################################################

vars1 <- var.select(object = fit, method = "md",
```

```
                    conservative = "high",
                    refit = FALSE)

md <- vars1$varselect[1:20, ]
vars1$topvars
merged <- data.frame(Variable = rownames(md)[1:10],
                     Depth = md$depth[1:10],
                     VIMP = md$vimp[1:10],
                     Variable = rownames(md)[11:20],
                     Depth = md$depth[11:20],
                     VIMP = md$vimp[11:20])

######################################################
# Most important variables using VIMP on training set
######################################################
vars2 <- fit$importance
head(sort(vars2, decreasing = TRUE), 15)
length(which(vars2 > 0.0005))

a <- head(sort(vars2, decreasing = TRUE), 20)
df <- data.frame(Variable = attr(a, "names")[1:10],
                 VIMP = round(a, 4)[1:10],
                 Variable = attr(a, "names")[11:20],
                 VIMP = round(a, 4)[11:20])
rownames(df) <- NULL


######################################################
# Most important variables using VIMP on test set
######################################################
cat("Starting variable importance on test set...")
#test data vimp
vars3 <- vimp(object = fit, newdata = test_overall,
              importance = "permute")
print(vars3$importance)

b <- head(sort(vars3$importance, decreasing = TRUE), 20)
df2 <- data.frame(Variable = attr(b, "names")[1:10],
                  VIMP = round(b, 4)[1:10],
                  Variable = attr(b, "names")[11:20],
                  VIMP = round(b, 4)[11:20])
rownames(df2) <- NULL

########################################
# VIMP reversed boxplots
########################################

cat("Starting subsampling method for vimp ... ")
# plot subsample
fit_sub <- subsample(fit)
```

```
jpeg("plot_vimp_ci.png", width = 700)
plot.subsample(fit_sub, pmax = 8, cex = 0.59,
               xlab = "100 x vimp (for survival time)")
dev.off()

save.image("image_rf_ovr_final.Rdata")

# predictions on test set

fit_test <- predict(fit, newdata = test_overall, seed = -12345,
                    split.depth = "all.trees", forest = FALSE)

##############################################
# error on test set (1 - concordance)
##############################################

fit_test$err.rate[fit_test$ntree]

# always put the times in sorted order
probs_rf_ovr <- predictSurvProb(object = fit,
                                newdata = test_overall,
                                times =
                      sort(unique(test_overall$patientsurvival)))

save(probs_rf_ovr, file = "probs_rf_ovr.Rdata")

metrics_rf_ovr <-  metrics_ovr_su(probs_rf_ovr, test_overall)
pair_rfsrc <- as.data.frame(cbind(test_overall$patientsurvival,
                                  test_overall$death))
colnames(pair_rfsrc) <- c("time", "status")


library(doParallel)
if (!exists("cl")) {
  cl <- makeCluster(20)
  registerDoParallel(cl)
}

surv_f <- as.formula(Surv(patientsurvival, death) ~ .)
pec_f <- as.formula(Hist(patientsurvival, death) ~ 1)
## run cox/rfsrc models
## for illustration we use a small number of trees
cox_full <- coxph(surv_f, data = training_overall,
                  x = TRUE, y = TRUE)

################################################
# another way to tune the forest
################################################

# following lines of R-code specify an example
# quick tuning repeated 3 times
```

```
cat("Starting forest tuning ... ")

forest_tuning1 <- tune.rfsrc(Surv(patientsurvival, death) ~ .,
                             data = training_overall,
                             ntreeTry = 200,
                             nodesizeTry = c(5,
                             seq(10, 100, by = 10)),
                             trace = TRUE, maxIter = 20,
                             doBest = TRUE)

forest_tuning1$optimal

cat("Starting forest tuning 2 ... ")
forest_tuning2 <- tune.rfsrc(Surv(patientsurvival, death) ~ .,
                              data = training_overall,
                              ntreeTry = 200,
                              nodesizeTry = c(5,
                              seq(10, 100, by = 10)),
                              trace = TRUE, maxIter = 20,
                              doBest = TRUE)

forest_tuning2$optimal

cat("Starting forest tuning 3 ... ")
forest_tuning3 <- tune.rfsrc(Surv(patientsurvival, death) ~ .,
                              data = training_overall,
                              ntreeTry = 200,
                              nodesizeTry = c(5,
                              seq(10, 100, by = 10)),
                              trace = TRUE, maxIter = 20,
                              doBest = TRUE)

forest_tuning3$optimal


optimals <- data.frame(combis1 = forest_tuning1$optimal,
                       combis2 = forest_tuning2$optimal,
                       combis3 = forest_tuning3$optimal)


save(optimals, file = "tune_optimals_ovr.Rdata")

############################################################
# comparison of out of bag error for Cox vs random forest
############################################################
cat("out-of-bag Cox Analysis ...", "\n")
set.seed(12345)
cox_err <- sapply(1:250, function(b) {

  # stratified subsampling of 5000 from training overall
```

```
  set1 <- which(training_overall$death == 1)
  set0 <- which(training_overall$death == 0)
  ind1 <- sample(set1, length(set1) / 8.304 , replace = FALSE)
  ind0 <- sample(set0, length(set0) / 8.304, replace = FALSE)
  train_temp <- training_overall[c(ind0, ind1), ]
  training_sub <- train_temp[sample(1:nrow(train_temp),
                                    5000,
                                    replace = TRUE), ]
  # to follow the progress of the repeats every 10 steps
  if (b %% 10 == 0) cat("cox bootstrap:", b, "\n")
  # bootstrapping by sampling with replacement
  index <- sample(1:nrow(training_sub), nrow(training_sub),
  replace = TRUE)
  trainset <- training_sub[index, ]
  validationset <- training_sub[-index, ]
  cox_obj <- tryCatch({coxph(surv_f, trainset)},
  error=function(ex){NULL})
  if (is.list(cox_obj)) {
    # calculating C-index
    randomForestSRC:::cindex(training_sub$patientsurvival[-index],
                             training_sub$death[-index],
                             predict(cox_obj, validationset))
  } else NA
})

cat("\n OOB error rates\n\n")
cat("\tRSF : ", fit$err.rate[fit$ntree], "\n")
cat("\tCox regression : ", mean(cox_err, na.rm = TRUE), "\n")

##############################################
#  Comparing prediction error for Cox - RSF
##############################################

# check on training data
prederror <- pec(list("Cox full" = cox_full, "RSF" = fit),
                 data = training_overall,
                 formula = pec_f,
                 splitMethod = "none",
                 times = 0:12)

jpeg("pec_comp_train_none.png", width = 700)
plot(prederror, xlab = "Time in years since transplantation",
     xlim = c(0, 12))
dev.off()


# # check on test data
prederror2 <- pec(list("Cox full" = cox_full, "RSF" = fit),
                  traindata = training_overall,
                  data = test_overall, formula = pec_f,
                  splitMethod = "none",
```

```
                  times = 0:12)

jpeg("pec_comp_test_none.png", width = 700)
plot(prederror2, xlab = "Time in years since transplantation",
     xlim = c(0, 12))
dev.off()
save.image("image_pec_cox_rf_ovr.Rdata")


cat("Starting prediction error with bootCv ... ", "\n")
set.seed(12345)
prederror3 <- pec(list("Cox full" = cox_full, "RSF" = fit),
                  data = test_overall, formula = pec_f,
                  splitMethod = "BootCv", B = 100,
                  times = 0:12, verbose = TRUE)

jpeg("pec_comp_test_bootcv.png", width = 700)
plot(prederror3, xlab = "Time in years since transplantation",
     xlim = c(0, 12))
dev.off()


cat("Starting prediction error with boostrap 632 ... ", "\n")
set.seed(12345)
prederror4 <- pec(list("Cox full" = cox_full, "RSF" = fit),
                  data = test_overall, formula = pec_f,
                  splitMethod = "Boot632", B = 100,
                  times = 0:12, verbose = TRUE)

jpeg("pec_comp_test_boot632.png", width = 700)
plot(prederror4, xlab = "Time in years since transplantation",
     xlim = c(0, 12))
dev.off()


# repeat for the C-index
cox_null <- coxph(as.formula(Surv(patientsurvival, death) ~ 1),
                  data = training_overall, x = TRUE, y = TRUE)

cat("Starting C-index with none (train)  ... ", "\n")
cerror1  <- pec::cindex(list("Reference" = cox_null,
                             "Cox full" = cox_full,
                             "RSF" = fit),
                        formula = pec_f,
                        splitMethod = "none",
                        data = training_overall,
                        eval.times = c(0.1, seq(1, 12, 1)))

jpeg("cindex_comp_train_none.png", width = 700)
plot(cerror1, xlab = "Time in years since transplantation",
     xlim = c(0, 12))
```

```
dev.off()

cat("Starting C-index with none (test)  ... ", "\n")
cerror2  <- pec::cindex(list("Reference" = cox_null,
                             "Cox full" = cox_full,
                             "RSF" = fit),
                        formula = pec_f,
                        splitMethod = "none",
                        data = test_overall,
                        eval.times = c(0.1, seq(1, 12, 1)))

jpeg("cindex_comp_test_none.png", width = 700)
plot(cerror2, xlab = "Time in years since transplantation",
     xlim = c(0, 12))
dev.off()

save.image("image_cindex_cox_rf_ovr.Rdata")

# compute the bootstrap-crossvalidation estimate of
# the C-index at different time points
cat("Starting C-index with BootCv... ", "\n")
set.seed(12345)
cerror3  <- pec::cindex(list("Reference" = cox_null,
                             "Cox full" = cox_full,
                             "RSF" = fit),
                        formula = pec_f,
                        splitMethod = "BootCv",
                        B = 100,
                        data = test_overall,
                        pred.times = c(0.1, seq(1, 12, 1)),
                        eval.times = c(0.1, seq(1, 12, 1)))

jpeg("cindex_comp_test_bootcv.png", width = 700)
plot(cerror3, xlab = "Time in years since transplantation",
     xlim = c(0, 12))
dev.off()

cat("Starting C-index with Boot632... ", "\n")
set.seed(12345)
cerror4  <- pec::cindex(list("Reference" = cox_null,
                             "Cox full" = cox_full,
                             "RSF" = fit),
                        formula = pec_f,
                        splitMethod = "Boot632",
                        B = 100,
                        data = test_overall,
                        pred.times = c(0.1, seq(1, 12, 1)),
                        eval.times = c(0.1, seq(1, 12, 1)))

jpeg("cindex_comp_test_boot632.png", width = 700)
plot(cerror4, xlab = "Time in years since transplantation",
```

```
     xlim = c(0, 12))
dev.off()
save.image("image_rf_ovr_final.Rdata")
```

D.2.3 Neural networks for survival analysis: Overall survival

```
#####################################################
# Neural networks one hidden layer part 1
#####################################################

# Hint: to download keras library for R you need to
# have installed the Anaconda3

install_packages <- c("survival", "keras", "pec",
                      "caret", "e1071", "doParallel")

for (i in 1:length(install_packages)){
  if (!install_packages[i] %in% installed.packages()){
    install.packages(install_packages[i])
  }
}

library(keras)
library(survival)
library(pec)
library(caret)
library(e1071)
library(doParallel)
cl <- makeCluster(3)
registerDoParallel(cl)
# parallel::stopCluster(cl)

# set this at the beginning of R session to get
# reproducible results (it is tensor-flow seed)
# it disables GPU and enables parallelism only on CPU
# random number generator in tensor flow backend engine
use_session_with_seed(seed = 1235, disable_gpu = TRUE,
                      disable_parallel_cpu = TRUE,
                      quiet = FALSE)

# read the scaled training data for overall survival
training_ovr_scaled <- read.table("training_ovr_scaled.txt")
# read the scaled test data for overall survival
test_ovr_scaled <- read.table("test_ovr_scaled.txt")

# function that transforms the train data into long format
# for each patient create the interval (1 till time 12 years),
# survival and create an id number

data_train_creator <- function(data){

  N <- nrow(data)
  # assign survival times to 12 intervals, each is a 1-year period
  data$interval <- as.numeric(cut(data$patientsurvival,
```

```
                                breaks = 12))
  data$survival <- as.numeric(cut(data$patientsurvival,
                                breaks = 12))
  data$id <- 1:N
  n.times <- data$interval
  data_long <- data[rep(seq_len(N), times = n.times), ]

  # create the correct intervals
  for(i in unique(data_long$id)) {
    n_length <- length(data_long$interval[data_long$id == i])
    data_long$interval[data_long$id == i] <- 1:n_length
  }

  data_long$status <- vector(mode = "numeric",
  length = nrow(data_long))

  # put indication 1 on status at the interval that patient dies
  for (i in 1:nrow(data_long)) {
    if (data_long$death[i] != data_long$status[i] &&
        data_long$survival[i] == data_long$interval[i])
        data_long$status[i] <- 1
  }

  return(data_long)

}

# function that creates data in the right long format for
# test set for each patient the interval goes from 1 year
# till 12 years

data_test_creator <- function(data){

  N <- nrow(data)
  # assign survival times to 12 intervals
  data$interval <- max(as.numeric(cut(data$patientsurvival,
  breaks = 12)))
  # the true interval survival
  data$survival <- as.numeric(cut(data$patientsurvival,
                                  breaks = 12))
  data$id <- 70001:(70000 + N) # define the patient ids abstractly
  n.times <- data$interval
  data_long <- data[rep(seq_len(N), times = n.times), ]

  # create the correct intervals
  for(i in unique(data_long$id)) {
    n_length <- length(data_long$interval[data_long$id == i])
    data_long$interval[data_long$id == i] <- 1:n_length

  }
```

```
  data_long$status <- vector(mode = "numeric",
                             length = nrow(data_long))

# put indication 1 on status at the intervals on
# which a patient has died
  for (i in 1:nrow(data_long)) {
    if (data_long$death[i] == 1 &&
        data_long$survival[i] <= data_long$interval[i])
        data_long$status[i] <- 1
  }

  return(data_long)

}


# function to calculate Brier and Integrated Brier scores for NN
brier_nnet <- function(pair_df, prob_matrix) {
  # calculating censoring distribution
  so <-  Surv(pair_df$true_surv , pair_df$status)
  time <-  so[, 1]
  ot <-  order(time)
  cens <-  so[ot, 2 ]
  time <-  time[ot]
  N <-  nrow(so)
  hatcdist <- prodlim(Surv(time, cens) ~ 1, reverse = TRUE)
  # censoring weights (reverse Kaplan-Meier)
  csurv <-  predict(hatcdist, times = time, type = "surv")
  csurv [csurv == 0] <-  Inf # access infinite value to censored
  btime <-  time
  # matrix transpose of survival predictions
  survs <-  t(as.matrix(prob_matrix))
  btime.n <-  unique(time) # find the unique times
  bsc <- rep(0, length(btime.n))

  for (j in 1:length(btime.n)) {
    # indicator vectors for selecting relevant patients
    # the censored ones
    values1 <- as.integer(time <= btime.n[j] & cens == 1)
    values2 <- as.integer(time > btime.n[j] )

    inb <- survs[j , ]
    inb[which(inb == 1)] <- inb[which(inb == 1)]
    inb[which(inb == 0)] <- inb[which(inb == 0)]
    # Brier: bsc
    bsc[j] <- mean((0 - survs[j, ] )^2 * values1 * (1/csurv) +
                     (1 - survs [j, ])^2 * values2 * (1/csurv[j]))
  }
  bsc <- c(0, bsc) # Brier score from time 0

  # calculate integrated brier score
```

```
  timing <- 0:12
  indexa <- 2:length(timing)
  int_bs <- (diff(timing) %*% ((bsc[indexa - 1] +
             bsc[indexa])/2))/diff(range(timing))

  return(list(Brier = bsc[-1], Int_brier = int_bs))
}


#################################################################
# function that calculates weights, node size, cross entropy,
# accuracy, sensitivity, specificity
# Brier score and integrated Brier score: uses function brier_nnet
# arguments are trained_model: aka the fit_keras,
# validation_set and real_status that is the event status
#################################################################

measures_calculator <- function(trained_model, datanew, real_status) {

  df1 <- data.frame(hazard = predict_proba(trained_model,
                                           as.matrix(datanew[,
                                           c(1:128, 131)]),
                                           batch_size = 500))
  df1$id <- datanew$id # ids of the patients
  df1$survival <- datanew$survival # survival time in years
   # create personalized groups per id
  groups <- split(df1, f = df1$id)

  true_surv <- unlist(lapply(groups, function(x) {
  surv_obj <- x$survival
  true_res <- surv_obj[1]
  return(true_res)}
  ))
  group_probs <- lapply(groups, function(x)
  {x <- cumprod(1 - x$hazard)})
  pred_mat <- do.call("rbind", group_probs)

  relative_probs <- vector(mode = "numeric",
                           length = length(group_probs))
  for (i in 1:length(group_probs)){
  temp <- group_probs[[i]]
  ind <- which(1:12 == true_surv[i])
# find the relative survival probability of the
# real survival interval
  relative_probs[i] <- temp[ind]
  }
  N0 <- length(unique(df1$id)) # number of unique persons

  # in the data frame create random id numbers
  # to label the patients
  df2 <- data.frame(relative_probs, true_surv,
```

```
                    status = real_status,
                    id = (70001):(70000 + N0))
  df2$prediction <- 1 - round(df2$relative_probs, digits = 0)
  # auxiliary confusion table
  tabel <- table(df2$prediction, df2$status)
  confusion_mat <- confusionMatrix(tabel, positive = "1")
  accuracy <- sum(df2$prediction == df2$status) / nrow(df2)
  sensitivity <- sum(df2$status == 1 & df2$prediction == 1) /
                    colSums(tabel)[2]
  specificity <- sum(df2$status == 0 & df2$prediction == 0) /
                    colSums(tabel)[1]
  precision <- as.numeric(confusion_mat$byClass[5])
  recall <- as.numeric(confusion_mat$byClass[6])
  f1score <- as.numeric(confusion_mat$byClass[7])

  brier_obj <- brier_nnet(df2, prob_matrix = pred_mat)
  int_brier <- as.numeric(brier_obj$Int_brier)
  brier_set <- brier_obj$Brier
  all_weights <- get_weights(fit_keras)
  nr_weights <- (length(all_weights[[1]]) +
                 length(all_weights[[2]])
                 + length(all_weights[[3]]) +
                 length(all_weights[[4]]))

  return(list(weights = nr_weights,
          node_size = length(get_weights(fit_keras)[[2]]),
          cross_entropy =
          result$metrics$loss[result$params$epochs],
          accuracy = accuracy,
          sensitivity = as.numeric(sensitivity),
          specificity = as.numeric(specificity),
          Precision = precision,
          Recall = recall,
          F1score = f1score,
          Integrated_brier = int_brier,
          Brier_scores = brier_set))
}

##########################################
# set up the cross-validation
##########################################

# most popular optimization algorithms used are the Stochastic
# Gradient Descent (SGD), ADAM and RMSprop
# you need to tune certain parameters such as learning rate or
# momentum
# things that can be tuned are node_size, lr, regularizer_l2,
# epochs, batch_size, decay

nfolds <- 5
set.seed(12345)
```

```
folds <- createFolds(training_ovr_scaled$patientsurvival,
                     k = 5, list = TRUE)

node_size <- seq(10, 100, by = 10) # grid of node sizes
dropout_rate <- c(0.2, 0.3, 0.4)
lr <- c(0.01, 0.1)
class_weights <- c(1, 2)
momentum <- c(0.8, 0.9)
combis <- expand.grid(node_size, dropout_rate,
                      lr, class_weights, momentum)

# initialize objects
cv_error <- matrix(0, nrow = nfolds, ncol = nrow(combis))
cv_accuracy <- matrix(0, nrow = nfolds, ncol = nrow(combis))
cv_specificity <- matrix(0, nrow = nfolds, ncol = nrow(combis))
cv_sensitivity <- matrix(0, nrow = nfolds, ncol = nrow(combis))
cv_precision <- matrix(0, nrow = nfolds, ncol = nrow(combis))
cv_recall <- matrix(0, nrow = nfolds, ncol = nrow(combis))
cv_f1score <- matrix(0, nrow = nfolds, ncol = nrow(combis))
cv_weights <- matrix(0, nrow = nfolds, ncol = nrow(combis))
cv_intbrier <- matrix(0, nrow = nfolds, ncol = nrow(combis))


# Learning rate controls how much to update the weight
# at the end of each batch and the momentum controls how
# much to let the previous update influence the current
# weight update. The number of neurons in a layer is an
# important parameter to tune. Generally the number of neurons
# in a layer controls the representational capacity of the
# network, at least at that point in the topology.

for (i in 1:nfolds) {

  cat("Started iteration i = ", i, "\n")
  indices <- folds[[i]]
  cat("Creating the training set ...", "\n")
  # create the train set
  train_set <-  data_train_creator(
  training_ovr_scaled[-indices, ])
  cat("Creating the validation set ...", "\n")
  validation_set <- data_test_creator(data =
  training_ovr_scaled[indices, ]) # create the validation set
  # real status for the validation set
  event_status <- training_ovr_scaled[indices, ]$death

  # create the matrices to be used for keras library
  # predictors: 128 variables + interval
  train_x <- as.matrix(train_set[, c(1:128, 131)])
  dimnames(train_x) <- NULL # the object must have empty dimnames
  train_y <- train_set$status
  validation_x <- as.matrix(validation_set[, c(1:128, 131)])
```

```
# # the object must have empty dimnames
dimnames(validation_x) <- NULL
validation_y <- validation_set$status

for (j in 1:nrow(combis)) {

  cat("Testing combination number:", j,
  "of repeat", i, " out of 5", "\n")
  cat("calculating for node size:", combis[j, 1], ",
  dropout rate:", combis[j, 2], "\n",
  "and learning rate", combis[j, 3], "and weak class weight",
  combis[j, 4],
  "and momentum", combis[j, 5], "...", "\n")

  # start building the model
  fit_keras <- keras_model_sequential()
  # Add layers to the model
  # we create a densely connected ANN to the output
  fit_keras %>%
    layer_dense(units = combis[j, 1], activation = 'relu',
    input_shape = c(129)) %>%
    layer_dropout(rate = combis[j, 2]) %>%
    layer_dense(units = 1, activation = 'sigmoid')
    # for binary class classification problem
  fit_keras %>% compile(
    loss = 'binary_crossentropy',
    optimizer = optimizer_sgd(lr = combis[j, 3],
    momentum = combis[j, 5])
    )

  result <- fit_keras %>% fit(
  train_x,
  train_y,
  epochs = 10,
  batch_size = 500,
  validation_data = list(validation_x, validation_y),
  class_weight = list("0" = 1, "1" = combis[j, 4])
  #,callbacks = c(early_stopping)
  )

  # now that the model has run lets calculate the measures
  # the total weights are 129*node_size + bias_input
  # + node_size + bias_node_size
  values <- measures_calculator(trained_model = fit_keras,
                                datanew = validation_set,
                                real_status = event_status)

  cv_error[i, j] <- values$cross_entropy
  cv_weights[i, j] <- values$weights
  cv_accuracy[i, j] <- values$accuracy
  cv_sensitivity[i, j] <- values$sensitivity
```

```
    cv_specificity[i, j] <- values$specificity
    cv_precision[i, j] <- values$Precision
    cv_recall[i, j] <- values$Recall
    cv_f1score[i, j] <- values$F1score
    cv_intbrier[i, j] <- values$Integrated_brier

  }
}

df_relu_ovr <- round(cbind(node_size = combis[, 1],
                           dropout_rate = combis[, 2],
                           learning_rate = combis[, 3],
                           momentum = combis[, 5],
                           weak_weight = combis[, 4],
                           weights = colMeans(cv_weights),
                           cross_entropy = colMeans(cv_error),
                           accuracy = colMeans(cv_accuracy),
                           sensitivity =
                           colMeans(cv_sensitivity),
                           specificity =
                           colMeans(cv_specificity),
                           precision = colMeans(cv_precision),
                           recall = colMeans(cv_recall),
                           f1score = colMeans(cv_f1score),
                           integrated_brier =
                           colMeans(cv_intbrier)), digits = 3)

save(df_relu_ovr, file = "results_relu_ovr.Rdata")
save.image("image_relu_ovr.Rdata")

parallel::stopCluster(cl)

df_relu_ovr <- as.data.frame(df_relu_ovr)


#########################################################
# fitting the final models for overall survival
#########################################################

# Hint: to download keras library you need to
# have installed the Anaconda3 and TensorFlow for R

install_packages <- c("survival", "keras", "pec",
                      "caret", "e1071", "gridExtra",
                      "grid", "ggpubr")

for (i in 1:length(install_packages)){
  if (!install_packages[i] %in% installed.packages()){
    install.packages(install_packages[i])
  }
}
```

```
library(keras)
library(survival)
library(pec)
library(caret)
library(e1071)
library(gridExtra)
library(grid)
library(ggpubr)

use_session_with_seed(seed = 1235, disable_gpu = TRUE,
                      disable_parallel_cpu = TRUE,
                      quiet = FALSE)


# read the scaled training data for overall survival
training_ovr_scaled <- read.table("training_ovr_scaled.txt")
# read the scaled test data for overall survival
test_ovr_scaled <- read.table("test_ovr_scaled.txt")

load("training_ovr_long.Rdata")
load("test_ovr_long.Rdata")

# function to calculate Brier and Integrated Brier scores for nnet
brier_nnet <- function(pair_df, prob_matrix) {
  # calculating censoring distribution
  so <-  Surv(pair_df$true_surv , pair_df$status)
  time <-  so[, 1]
  ot <-  order(time)
  cens <-  so[ot, 2 ]
  time <-  time[ot]
  N <-  nrow(so)
  hatcdist <- prodlim(Surv(time, cens) ~ 1, reverse = TRUE)
  # censoring weights (reverse Kaplan-Meier)
  csurv <-  predict(hatcdist, times = time, type = "surv")
  csurv [csurv == 0] <-  Inf # access infinite value to censored
  btime <-  time
  # matrix transpose of survival predictions
  survs <-  t(as.matrix(prob_matrix))
  btime.n <-  unique(time) # find the unique times
  bsc <- rep(0, length(btime.n))

  for (j in 1:length(btime.n)) {
    # indicator vectors for selecting relevant patients
    # the censored ones
    values1 <- as.integer(time <= btime.n[j] & cens == 1)
    values2 <- as.integer(time > btime.n[j] )

    inb <- survs[j , ]
    inb[which(inb == 1)] <- inb[which(inb == 1)]
    inb[which(inb == 0)] <- inb[which(inb == 0)]
    # Brier: bsc
```

```
    bsc[j] <- mean((0 - survs[j, ] )^2 * values1 * (1/csurv) +
                     (1 - survs [j, ])^2 * values2 * (1/csurv[j]))
  }
  bsc <- c(0, bsc) # Brier score from time 0

  # calculate integrated brier score
  timing <- 0:12
  indexa <- 2:length(timing)
  int_bs <- (diff(timing) %*% ((bsc[indexa - 1] +
              bsc[indexa])/2))/diff(range(timing))

  return(list(Brier = bsc[-1], Int_brier = int_bs))
}

# function that calculates all measures

measures_calculator <- function(trained_model,
                                datanew, real_status) {

  df1 <- data.frame(hazard = predict_proba(trained_model,
                                           as.matrix(datanew[,
                                           c(1:128, 131)]),
                                           batch_size = 500))
  df1$id <- datanew$id # ids of the patients
  df1$survival <- datanew$survival # survival time in years
  # create personalized groups per id
  groups <- split(df1, f = df1$id)

  true_surv <- unlist(lapply(groups, function(x) {
    surv_obj <- x$survival
    true_res <- surv_obj[1]
    return(true_res)}
  ))
  group_probs <- lapply(groups, function(x) {
  x <- cumprod(1 - x$hazard)})
  pred_mat <- do.call("rbind", group_probs)

  relative_probs <- vector(mode = "numeric",
                           length = length(group_probs))
  for (i in 1:length(group_probs)){
    temp <- group_probs[[i]]
    ind <- which(1:12 == true_surv[i])
  # find the relative survival probability of
  # the real survival interval
    relative_probs[i] <- temp[ind]
  }
  N0 <- length(unique(df1$id)) # number of unique persons


  # in the data frame create random id numbers
  # to label the patients
```

```
  df2 <- data.frame(relative_probs, true_surv,
  status = real_status, id = (70001):(70000 + N0))
  df2$prediction <- 1 - round(df2$relative_probs, digits = 0)
  # create possible classes
  classes <- c(0, 1)
  # auxiliary confusion table
  tabel <- table(factor(df2$prediction, levels = classes),
                 factor(df2$status, levels = classes))
  confusion_mat <- confusionMatrix(tabel, positive = "1")
  accuracy <- sum(df2$prediction == df2$status) / nrow(df2)
  sensitivity <- sum(df2$status == 1 & df2$prediction == 1) /
                     colSums(tabel)[2]
  specificity <- sum(df2$status == 0 & df2$prediction == 0)
                     / colSums(tabel)[1]
  precision <- as.numeric(confusion_mat$byClass[5])
  recall <- as.numeric(confusion_mat$byClass[6])
  f1score <- as.numeric(confusion_mat$byClass[7])

  brier_obj <- brier_nnet(df2, prob_matrix = pred_mat)
  int_brier <- as.numeric(brier_obj$Int_brier)
  brier_set <- brier_obj$Brier
  all_weights <- get_weights(fit_keras)
  nr_weights <- (length(all_weights[[1]]) +
                 length(all_weights[[2]])
                 + length(all_weights[[3]]) +
                 length(all_weights[[4]]))

  return(list(weights = nr_weights,
              node_size = length(get_weights(fit_keras)[[2]]),
              cross_entropy =
              result$metrics$loss[result$params$epochs],
              accuracy = accuracy,
              sensitivity = as.numeric(sensitivity),
              specificity = as.numeric(specificity),
              Precision = precision,
              Recall = recall,
              F1score = f1score,
              Integrated_brier = int_brier,
              Brier_scores = brier_set))
}

fit_keras <- keras_model_sequential()
# Add layers to the model
# we create a densely connected ANN to the output
fit_keras %>%
  layer_dense(units = 90, activation = 'relu',
  input_shape = c(129)) %>%
  layer_dropout(rate = 0.1) %>%
  layer_dense(units = 1, activation = 'sigmoid')
fit_keras %>% compile(
  loss = 'binary_crossentropy', # for binary class classification
```

```
  optimizer = optimizer_sgd(lr = 0.2, momentum = 0.9)
  #, metrics = c("accuracy")
)


event_status <- test_ovr_scaled$death

# create the matrices to be used for keras library
# predictors: 128 variables + interval
train_x <- as.matrix(training_ovr_long[, c(1:128, 131)])
dimnames(train_x) <- NULL # the object must have empty dimnames
train_y <- training_ovr_long$status
test_x <- as.matrix(test_ovr_long[, c(1:128, 131)])
dimnames(test_x) <- NULL # the object must have empty dimnames
test_y <- test_ovr_long$status

result <- fit_keras %>% fit(
  train_x,
  train_y,
  epochs = 10,
  batch_size = 1000,
  validation_data = list(test_x, test_y),
  class_weight = list("0" = 1, "1" = 1)
)

# now that the model has run lets calculate the measures
values_final <- measures_calculator(trained_model = fit_keras,
                                    datanew = test_ovr_long,
                                    real_status = event_status)

df1 <- data.frame(hazard = predict_proba(fit_keras,
                                         as.matrix(test_ovr_long[,
                                         c(1:128, 131)]),
                                         batch_size = 500))
df1$id <- test_ovr_long$id # ids of the patients
df1$survival <- test_ovr_long$survival # survival time in years
# create personalized groups per id
groups <- split(df1, f = df1$id)

true_surv <- unlist(lapply(groups, function(x) {
  surv_obj <- x$survival
  true_res <- surv_obj[1]
  return(true_res)}
))
group_probs <- lapply(groups, function(x) {
                      x <- cumprod(1 - x$hazard)})
pred_mat_nn1h <- do.call("rbind", group_probs)

##############################################
# rerun the network 4 times...
##############################################
```

```
means1 <- colMeans(pred_mat_nn1h)
means2 <- colMeans(pred_mat_nn1h)
means3 <- colMeans(pred_mat_nn1h)
means4 <- colMeans(pred_mat_nn1h)

df1 <- data.frame(Time = 0:12, Probs = c(1, means1),
                  Model = "Iteration 1")
df2 <- data.frame(Time = 0:12, Probs = c(1, means2),
                  Model = "Iteration 2")
df3 <- data.frame(Time = 0:12, Probs = c(1, means3),
                  Model = "Iteration 3")
df4 <- data.frame(Time = 0:12, Probs = c(1, means4),
                  Model = "Iteration 4")


####################################
# mean survival probabilities
####################################

plot1 <- rbind(df1, df2, df3, df4)

obj1 <- ggplot(plot1, aes(x = Time, y = Probs, color = Model)) +
  geom_line(size = 0.9) +
  scale_x_continuous(breaks = 0:12) +
  scale_y_continuous(breaks = seq(0, 1, 0.2), limits = c(0, 1)) +
  xlab("Time since transplantation in years") +
  ylab("Survival probability") +
  theme_classic() +
  ggtitle(" ") +
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_color_manual(values=c("black", "red",
                              "green", "blue"),
                              name = "NN 1 hidden")

#####################################################
# Brier scores over time for each repeat
#####################################################

briers1 <- values_final$Brier_scores
briers2 <- values_final$Brier_scores
briers3 <- values_final$Brier_scores
briers4 <- values_final$Brier_scores
df5 <- data.frame(Time = 0:12, Brier = c(0, briers1),
                  Model = "Iteration 1")
df6 <- data.frame(Time = 0:12, Brier = c(0, briers2),
                  Model = "Iteration 2")
df7 <- data.frame(Time = 0:12, Brier = c(0, briers3),
                  Model = "Iteration 3")
df8 <- data.frame(Time = 0:12, Brier = c(0, briers4),
                  Model = "Iteration 4")
```

```
plot1 <- rbind(df5, df6, df7, df8)

obj2 <- ggplot(plot1, aes(x = Time, y = Brier, color = Model)) +
  geom_line(size = 0.9) +
  scale_x_continuous(breaks = 0:12) +
  #scale_y_continuous(breaks = seq(0.3, 1, 0.1),
  limits = c(0.3, 1)) +
  xlab("Time since transplantation in years") +
  ylab("Prediction error (Brier score)") +
  theme_classic() +
  ggtitle(" ") +
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_color_manual(values=c("black", "red",
                              "green", "blue"),
                              name = "NN 1 hidden")

grid.arrange(obj1, obj2, ncol=2)

############################################################
# this is Garsons's connection weight method
# you can use it to get interpretation of the weights of
# the neural network. It returns a list with the variable
# names and the the relative importance measure
############################################################

var_imp_keras <- function(model){

  list_weights <- get_weights(model)
  # list_weights[[1]] are the weights between input-hidden node
  # list_weights[[2]] are the weights between bias-hidden node
  # list_weights[[3]] are the weights between hidden node-output
  # list_weights[[4]] is the weight between bias-output
  # names input nodes , number input nodes
  names <- c(colnames(training_ovr_scaled)[1:128], "interval")
  n_input <- length(list_weights[[1]]) / length(list_weights[[2]])
  # number hidden nodes , number output nodes
  n_nodes <- length(list_weights[[3]])
  n_outputs <- length(list_weights[[4]])

  # dimensions for weight matrices
  ncols1 <- (n_input + 1)
  nrows1 <- n_nodes
  ncols2 <- (n_nodes + 1)
  nrows2 <- n_outputs
  length1 <- ncols1 * nrows1
  length2 <- ncols2 * nrows2

  # selecting weights
  mat1 <- rbind(list_weights[[2]], list_weights[[1]])
  mat1 <- t(mat1)
```

```
  weights1 <- mat1
  colnames(weights1) <- c("Bias", names)
  rownames(weights1) <- paste ("N" , seq (1:n_nodes) , sep = "")

  mat2 <- rbind(list_weights[[4]], list_weights[[3]])
  weights2 <- mat2
  rownames(weights2) <- c("Bias", paste ("N", seq (1:n_nodes),
                                         sep = ""))
  # calculating variable imp using connection weight method
  mega_mat <- matrix(NA, ncol = n_input, nrow = n_outputs)
  for (i in 1:n_input) {
    for (j in 1:n_outputs) {
      mega_mat[j, i] <- sum(weights1[, i + 1] *
                            weights2[2:(n_nodes + 1), j ])
    }
  }
  colnames(mega_mat) <- names
  mega_mat_abs <- abs(mega_mat)
  totals <- rowSums (mega_mat_abs)
  mega_mat_rel <- as.data.frame(mega_mat_abs/ totals)
  rels <- as.vector(as.numeric(mega_mat_rel))

  return(list(Names = names, Rels = rels))
}

# find the variable importance of the last fit_keras model
var_imp <- var_imp_keras(model = fit_keras)

df2 <- data.frame(name = var_imp$Names,
                  variable_importance = var_imp$Rels)
df2 <- df2[order(df2$variable_importance, decreasing = TRUE), ]

rel_imp <- data.frame(Variable = df2$name[1:10],
                      Importance = df2$variable_importance[1:10],
                      Variable = df2$name[11:20],
                      Importance = df2$variable_importance[11:20])
rownames(rel_imp) <- NULL

# plots have been repeated for iterations 1 to 4...

ggbarplot(data = df2[1:5, ], x = "name",
  y = "variable_importance", fill = rainbow(5)) +
  xlab("Variable name") +
  ylab("Relative importance") +
  ylim(c(0, 0.2)) +
  theme(axis.text=element_text(size = 8),
        axis.title=element_text(size=11, face="bold")) +
  labs(title = " ",  tag = "A")

############################################################
# plot of comparison of weights for NN with 1 hidden layer
```

```
# and NN with 2 hidden layers
#########################################################

nodesize <- seq(10, 130, by = 10)
weights_1h <-  (129 + 1)*nodesize  + (nodesize + 1)*1
weights_2h <- (129 + 1)*nodesize + (nodesize + 1)*(nodesize) +
              nodesize  + 1

w1 <- data.frame(NodeSize = nodesize, Weights = weights_1h,
                 Model = "FF 1 hidden layer")
w2 <- data.frame(NodeSize = nodesize, Weights = weights_2h,
                 Model = "FF 2 hidden layers")

plot3 <- rbind(w1, w2)

ggplot(plot3, aes(x = NodeSize, y = Weights, color = Model)) +
  geom_point(size = 2.5) +
  scale_x_continuous(breaks = seq(10, 130, by = 10)) +
  scale_y_continuous(breaks = seq(1000, 35000, by = 4000))  +
  xlab("Node size per hidden layer") +
  ylab("Number of weights") +
  theme_classic() +
  ggtitle(" ") +
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_color_manual(values = c("blue", "red"),
                     name = "Neural Network")

#################################################
# final model with 2 hidden layers
#################################################
fit_keras <- keras_model_sequential()
# Add layers to the model
# we create a densely connected ANN to the output
fit_keras %>%
  layer_dense(units = 100, activation = 'relu',
  input_shape = c(129)) %>%
  layer_dropout(rate = 0.1) %>%
  layer_dense(units = 100, activation = 'relu') %>%
  layer_dropout(rate = 0.1) %>%
  layer_dense(units = 1, activation = 'sigmoid')
fit_keras %>% compile(
  loss = 'binary_crossentropy',
  optimizer = optimizer_sgd(lr = 0.2, momentum = 0.9)
)

event_status <- test_ovr_scaled$death

# create the matrices to be used for keras library
# predictors: 128 variables + interval
train_x <- as.matrix(training_ovr_long[, c(1:128, 131)])
dimnames(train_x) <- NULL # the object must have empty dimnames
```

```
train_y <- training_ovr_long$status
test_x <- as.matrix(test_ovr_long[, c(1:128, 131)])
dimnames(test_x) <- NULL # the object must have empty dimnames
test_y <- test_ovr_long$status

result <- fit_keras %>% fit(
  train_x,
  train_y,
  epochs = 10,
  batch_size = 1000,
  validation_data = list(test_x, test_y),
  class_weight = list("0" = 1, "1" = 1)
)

# calculate the measures....
values_final <- measures_calculator(trained_model = fit_keras,
                                    datanew = test_ovr_long,
                                    real_status = event_status)

##########################################
# repeat the process 4 times...
##########################################

briers5 <- values_final$Brier_scores
briers6 <- values_final$Brier_scores
briers7 <- values_final$Brier_scores
briers8 <- values_final$Brier_scores
df9 <- data.frame(Time = 0:12, Brier = c(0, briers5),
                  Model = "Iteration 1")
df10 <- data.frame(Time = 0:12, Brier = c(0, briers6),
                   Model = "Iteration 2")
df11 <- data.frame(Time = 0:12, Brier = c(0, briers7),
                   Model = "Iteration 3")
df12 <- data.frame(Time = 0:12, Brier = c(0, briers8),
                   Model = "Iteration 4")

plotb <- rbind(df9, df10, df11, df12)

obja <- ggplot(plotb, aes(x = Time, y = Brier, color = Model)) +
  geom_line(size = 0.9) +
  scale_x_continuous(breaks = 0:12) +
  scale_y_continuous(breaks = seq(0, 0.3, 0.05),
                     limits = c(0, 0.3)) +
  xlab("Time since transplantation in years") +
  ylab("Prediction error (Brier score)") +
  #theme_classic() +
  ggtitle(" ") +
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_color_manual(values=c("black", "red",
                              "green", "blue"),
                              name = "NN 2 hidden")
```

```
############################################################
# Proportions of the 5 performance metrics per iteration
############################################################

acc1 <- data.frame(Proportion = values_final$accuracy,
                   Iteration = "Iteration 1",
                   Measure = "Accuracy")
acc2 <- data.frame(Proportion = values_final$accuracy,
                   Iteration = "Iteration 2",
                   Measure = "Accuracy")
acc3 <- data.frame(Proportion = values_final$accuracy,
                   Iteration = "Iteration 3",
                   Measure = "Accuracy")
acc4 <- data.frame(Proportion = values_final$accuracy,
                   Iteration = "Iteration 4",
                   Measure = "Accuracy")
df_acc <- rbind(acc1, acc2, acc3, acc4)

sens1 <- data.frame(Proportion = values_final$sensitivity,
                    Iteration = "Iteration 1",
                    Measure = "Sensitivity")
sens2 <- data.frame(Proportion = values_final$sensitivity,
                    Iteration = "Iteration 2",
                    Measure = "Sensitivity")
sens3 <- data.frame(Proportion = values_final$sensitivity,
                    Iteration = "Iteration 3",
                    Measure = "Sensitivity")
sens4 <- data.frame(Proportion = values_final$sensitivity,
                    Iteration = "Iteration 4",
                    Measure = "Sensitivity")
df_sens <- rbind(sens1, sens2, sens3, sens4)


spec1 <- data.frame(Proportion = values_final$specificity,
                    Iteration = "Iteration 1",
                    Measure = "Specificity")
spec2 <- data.frame(Proportion = values_final$specificity,
                    Iteration = "Iteration 2",
                    Measure = "Specificity")
spec3 <- data.frame(Proportion = values_final$specificity,
                    Iteration = "Iteration 3",
                    Measure = "Specificity")
spec4 <- data.frame(Proportion = values_final$specificity,
                    Iteration = "Iteration 4",
                    Measure = "Specificity")
df_spec <- rbind(spec1, spec2, spec3, spec4)


prec1 <- data.frame(Proportion = values_final$Precision,
                    Iteration = "Iteration 1",
```

```
                      Measure = "Precision")
prec2 <- data.frame(Proportion = values_final$Precision,
                      Iteration = "Iteration 2",
                      Measure = "Precision")
prec3 <- data.frame(Proportion = values_final$Precision,
                      Iteration = "Iteration 3",
                      Measure = "Precision")
prec4 <- data.frame(Proportion = values_final$Precision,
                      Iteration = "Iteration 4",
                      Measure = "Precision")
df_prec <- rbind(prec1, prec2, prec3, prec4)

f11 <- data.frame(Proportion = values_final$F1score,
                      Iteration = "Iteration 1",
                      Measure = "F1score")
f12 <- data.frame(Proportion = values_final$F1score,
                      Iteration = "Iteration 2",
                      Measure = "F1score")
f13 <- data.frame(Proportion = values_final$F1score,
                      Iteration = "Iteration 3",
                      Measure = "F1score")
f14 <- data.frame(Proportion = values_final$F1score,
                      Iteration = "Iteration 4",
                      Measure = "F1score")
df_f1 <- rbind(f11, f12, f13, f14)


plot_all <- rbind(df_acc, df_sens, df_spec, df_prec, df_f1)

objb <- ggplot(plot_all, aes(x=Measure, y=Proportion,
                             color = Iteration)) +
  geom_point(position=position_dodge(0.22), size = 2) +
  scale_y_continuous(breaks = seq(0, 1, by = 0.1)) +
  scale_color_manual(values=c("black", "red",
                              "green", "blue"),
                              name = "NN 2 hidden") +
                              scale_x_discrete(" ")
grid.arrange(objb, obja)

#################################################################
# REMINDER:
# for the final fit of failure-free survival code is very similar
# replace unos_overall -> unos_failure_free
#         overall -> ffs, ovr -> ffs, patientsurvival -> gs_ffs
#         death -> gs_ffsstate
#################################################################
```

D.3 Comparison of methods

D.3.1 Functions that calculate the measures

```
###################################
# file "the_functions.R"
###################################

install_packages <- c("survival", "pec",
                      "caret", "e1071")

for (i in 1:length(install_packages)){
  if (!install_packages[i] %in% installed.packages()){
    install.packages(install_packages[i])
  }
}

# libraries to call
library(survival)
library(pec)
library(caret)
library(e1071)

# function that gets the probability matrix and returns
# the relative metrics

metrics_ovr <- function(prob_matrix, test_set, p = 0.5) {

  relative_probs <- vector(mode = "numeric",
                           length = nrow(test_set))
  time <- sort(test_set$patientsurvival)
  for (i in 1:length(relative_probs)){
    temp <- prob_matrix[i, ]
    ind <- max(which(time == test_set$patientsurvival[i]))
    relative_probs[i] <- temp[ind]
  }

  # find expected status based on 0.5 probability cut-off
  expected_status <- ifelse(relative_probs > p, 0, 1)
  classes <- c(0, 1)
  # auxiliary confusion table
  tabel <- table(factor(test_set$death, levels = classes),
                 factor(expected_status, levels = classes))
  confusion_mat <- confusionMatrix(tabel, positive = "1")
  confusion_mat <- confusionMatrix(tabel, positive = "1")
  accuracy <- as.numeric(confusion_mat$overall[1])
  sensitivity <- as.numeric(confusion_mat$byClass[1])
  specificity <- as.numeric(confusion_mat$byClass[2])
  precision <- as.numeric(confusion_mat$byClass[5])
  recall <- as.numeric(confusion_mat$byClass[6])
  f1score <- as.numeric(confusion_mat$byClass[7])

  return(list(Accuracy = accuracy, Sensitivity = sensitivity,
              Specificity = specificity, Precision = precision,
```

```
              Recall = recall, F1score = f1score))

}

# this should be used in case you find the survival probabilities
# at sort-unique time points
metrics_ovr_su <- function(prob_mat_su, test_set, p = 0.5) {

  relative_probs <- vector(mode = "numeric",
                           length = nrow(test_set))
  time <- sort(unique(test_set$patientsurvival))
  for (i in 1:length(relative_probs)){
    temp <- prob_mat_su[i, ]
    ind <- max(which(time == test_set$patientsurvival[i]))
    relative_probs[i] <- temp[ind]
  }

  # find expected status based on 0.5 probability cut-off
  expected_status <- ifelse(relative_probs > p, 0, 1)
  classes <- c(0, 1)
  # auxiliary confusion table
  tabel <- table(factor(test_set$death, levels = classes),
                 factor(expected_status, levels = classes))

  confusion_mat <- confusionMatrix(tabel, positive = "1")
  accuracy <- as.numeric(confusion_mat$overall[1])
  sensitivity <- as.numeric(confusion_mat$byClass[1])
  specificity <- as.numeric(confusion_mat$byClass[2])
  precision <- as.numeric(confusion_mat$byClass[5])
  recall <- as.numeric(confusion_mat$byClass[6])
  f1score <- as.numeric(confusion_mat$byClass[7])

  return(list(Accuracy = accuracy, Sensitivity = sensitivity,
              Specificity = specificity, Precision = precision,
              Recall = recall, F1score = f1score))
}

# calculate Brier and Integrated Brier scores at all sorted times
brier_general <- function(pair_df, prob_matrix) {

  so <- Surv(pair_df$time, pair_df$status)
  time <- so[, 1]
  ot <- order(time)
  cens <- so[ot, 2]
  time <- time[ot] # ordered times
  N <- nrow(so)

  # find the reverse Kaplan-Meier distribution
  hatcdist <- prodlim(Surv(time, cens) ~ 1, reverse = TRUE)
  csurv <- predict(hatcdist, times = time, type = "surv")
  csurv[csurv == 0] <- Inf
```

```
  btime <- time # duplicate time vector

  survs <- t(as.matrix(prob_matrix)) # put it as matrix
  bsc <- rep(0, nrow(survs))

  for (j in 1:nrow(survs)) {
    values1 <- as.integer(time <= btime[j] & cens == 1)
    values2 <- as.integer(time > btime[j])
    inb <- survs[j,]

    inb[which(inb == 1)] <- inb[which(inb == 1)]
    inb[which(inb == 0)] <- inb[which(inb == 0)]
    bsc[j] <- mean((0 - survs[j, ])^2 * values1 * (1/csurv) +
                     (1 - survs[j, ])^2 * values2 * (1/csurv[j]))

    # calculate the integrated brier score
  }

  pos <- sindex(jump.times = btime, eval.times = 1:12)
  brier <- c(0, bsc[pos])

  # calculate integrated brier score starting from time point 0
  timing <- 0:(length(brier) - 1)
  indexa <- 2:length(timing)
  int_bs <- (diff(timing) %*% ((brier[indexa - 1] +
              brier[indexa])/2))/diff(range(timing))

  return(list(Brier = brier[-1], Int_brier = int_bs))
}

# this should be used when the probability matrix is found
# at sort unique time points
brier_general_su <- function(pair_df, prob_matrix_su) {

  so <- Surv(pair_df$time, pair_df$status)
  time <- so[, 1]
  ot <- order(time)
  cens <- so[ot, 2]
  time <- time[ot] # ordered times
  N <- nrow(so)

  # find the reverse Kaplan-Meier distribution
  hatcdist <- prodlim(Surv(time, cens) ~ 1, reverse = TRUE)
  csurv <- predict(hatcdist, times = time, type = "surv")
  csurv[csurv == 0] <- Inf
  btime <- time # duplicate time vector

  survs <- t(as.matrix(prob_matrix_su)) # put it as matrix

  bsc <- rep(0, nrow(survs))
```

```
  for (j in 1:nrow(survs)) {
    values1 <- as.integer(time <= btime[j] & cens == 1)
    values2 <- as.integer(time > btime[j])
    inb <- survs[j,]
    inb[which(inb == 1)] <- inb[which(inb == 1)]
    inb[which(inb == 0)] <- inb[which(inb == 0)]
    bsc[j] <- mean((0 - survs[j, ])^2 * values1 * (1/csurv) +
                   (1 - survs[j, ])^2 * values2 * (1/csurv[j]))

    # calculate the integrated brier score
  }

  brier <- c(0, bsc)

  # calculate integrated brier score
  timing <- 0:(length(brier) - 1)
  indexa <- 2:length(timing)
  int_bs <- (diff(timing) %*% ((brier[indexa - 1] +
             brier[indexa])/2))/diff(range(timing))

  return(list(Brier = brier[-1], Int_brier = int_bs))
}
```

D.3.2 Comparisons at exact time points: Overall survival

```
install_packages <- c("pec", "survival", "caret", "ggplot2",
                      "randomForestSRC", "knitr", "gridExtra")

for (i in 1:length(install_packages)){
  if (!install_packages[i] %in% installed.packages()){
    install.packages(install_packages[i])
  }
}

library(pec)
library(survival)
library(caret)
library(ggplot2)
library(randomForestSRC)
source("https://bioconductor.org/biocLite.R")
biocLite("survcomp")
library(survcomp)
library(knitr)
library(gridExtra)

source("the_functions.R")
load("unos_overall.RData")
unos_overall$patientsurvival <- unos_overall$patientsurvival
                                  + (1/365)

set.seed(12345)
N <- nrow(unos_overall)
index <- sample(1:N, round(N/3), replace = FALSE)
training_overall <- unos_overall[-index, ] # create training set
test_overall <- unos_overall[index, ] # create test set

################################################
# Cox models and RSF at exact time points
################################################

# Cox full and Cox KM
cox_full <- coxph(Surv(patientsurvival, death) ~.,
                  data = training_overall, x = TRUE, y = TRUE)

pi_cox_full <- predict(cox_full, newdata = test_overall)

cindex_cox_full <-  concordance.index(x = pi_cox_full,
                                      surv.time =
                                     test_overall$patientsurvival,
                                      surv.event =
                                      test_overall$death)$c.index

cox_null <- coxph(Surv(patientsurvival, death) ~ 1,
```

```
                  data = training_overall, x = TRUE, y = TRUE)

# model cox backward
cox_back <- selectCox(formula = Surv(patientsurvival, death) ~.,
                      data = training_overall, rule = "aic")
vars_back <- cox_back$In # variables
f_back <- as.formula(paste("Surv(patientsurvival, death) ~",
                           paste(vars_back, collapse = "+")))
# Cox backward in coxph object
cox_back <- coxph(formula = f_back, x = TRUE, y = TRUE,
                  data = training_overall, method = "breslow")

pi_cox_back <- predict(cox_back, newdata = test_overall)

cindex_cox_back <-  concordance.index(x = pi_cox_back,
                                      surv.time =
                                     test_overall$patientsurvival,
                                      surv.event =
                                      test_overall$death)$c.index


pec_f <- as.formula(Hist(patientsurvival, death) ~ 1)
surv_f <- as.formula(Surv(patientsurvival, death) ~ .)

# model Lasso
response_pair <- c("patientsurvival", "death")

# for the training set
X_train <- training_overall[, !(colnames(training_overall) %in%
                                response_pair)]
# create martix X for glmnet
X_train_ovr <- model.matrix(~., X_train)[, -1]
# dim(X_train_ovr) # 41529 x 129
Y_train_ovr <- training_overall[, colnames(training_overall) %in%
                                   response_pair] # create matrix Y
Y_survtrain_ovr <- Surv(Y_train_ovr$patientsurvival,
                        Y_train_ovr$death)

# for the test set
x_test <- test_overall[,
                       !(colnames(test_overall) %in% response_pair)]
X_test_ovr <- model.matrix(~., x_test)[, -1]
Y_test_ovr <- test_overall[, colnames(test_overall) %in%
                               response_pair]
# Surv function packages survival data into the form
# expected by glmnet
Y_survtest_ovr <- Surv(Y_test_ovr$patientsurvival,
                       Y_test_ovr$death)

# use these two datasets to fit Cox model in the selected features
training_ovr_extended <- as.data.frame(cbind(X_train_ovr,
```

```
                                          Y_train_ovr))
test_ovr_extended <- as.data.frame(cbind(X_test_ovr, Y_test_ovr))

seed <- 12345
set.seed(seed)
folds <- createFolds(y = training_overall$patientsurvival,
                     k = 5, list = FALSE)

cv.fit <- cv.glmnet(x = X_train_ovr, y = Y_survtrain_ovr,
                    foldid = folds, parallel = TRUE,
                    family = "cox", grouped = TRUE,
                    maxit = 1000)

active_index <- as.numeric(unlist(predict(cv.fit,
                                          newx = X_test_ovr,
                                          s = "lambda.1se",
                                          type = "nonzero")))
names_coefs <- colnames(test_ovr_extended)[1:128]
vars_active <- names_coefs[active_index]
rm(cv.fit)

f_lasso <- as.formula(paste("Surv(patientsurvival, death) ~",
                            paste(vars_active, collapse = "+")))


cox_lasso <- coxph(formula = f_lasso, x = TRUE, y = TRUE,
                   data = training_ovr_extended)

pi_cox_lasso <- predict(cox_lasso, newdata = test_ovr_extended)

cindex_cox_lasso <-  concordance.index(x = pi_cox_lasso,
                                       surv.time =
                                      test_overall$patientsurvival,
                                       surv.event =
                                       test_overall$death)$c.index

cox_pair <- as.data.frame(cbind(test_ovr_extended$patientsurvival,
                                test_ovr_extended$death))
colnames(cox_pair) <- c("time", "status")

time_su <- sort(unique(test_ovr_extended$patientsurvival))

#####################################################
# calculate survival probabilities and metrics
#####################################################

# find the probabilities of the models
cox_probs_full <- predictSurvProb(cox_full, test_overall,
                                  times = time_su)

brier_cox_full <-  brier_general_su(cox_pair, cox_probs_full)
```

```
metrics_cox_full <-  metrics_ovr_su(prob_mat_su = cox_probs_full,
                                    test_set = test_overall,
                                    p = 0.5)

cox_probs_back <- predictSurvProb(cox_back, test_overall,
                                  times = time_su)

brier_cox_back <-  brier_general_su(cox_pair, cox_probs_back)
metrics_cox_back <-  metrics_ovr_su(prob_mat_su = cox_probs_back,
                                    test_set = test_overall,
                                    p = 0.5)

cox_probs_lasso <- predictSurvProb(cox_lasso, test_ovr_extended,
                                   times = time_su)

brier_cox_lasso <-  brier_general_su(cox_pair, cox_probs_lasso)
metrics_cox_lasso <-  metrics_ovr_su(prob_mat_su =
                                     cox_probs_lasso,
                                     test_set = test_ovr_extended,
                                     p = 0.5)

# Random survival forests
model_rsf <- rfsrc(Surv(patientsurvival, death) ~ .,
                   splitrule = "logrank", nsplit = 5,
                   data = training_overall, ntree = 250,
                   seed = -12345, mtry = 35, nodesize = 50,
                   ntime = 500, sampsize = 5000,
                   forest = TRUE, importance = FALSE
)
fit_test <- predict(fit, newdata = test_overall, seed = -12345,
                    split.depth = "all.trees", forest = FALSE)
# always put the times in sorted order
probs_rf_ovr <- predictSurvProb(object = model_rsf,
                                newdata = test_overall,
                                times =
                      sort(unique(test_overall$patientsurvival)))


pair_rfsrc <- as.data.frame(cbind(test_overall$patientsurvival,
                                  test_overall$death))
colnames(pair_rfsrc) <- c("time", "status")
brier_model_rsf <-  brier_general_su(pair_rfsrc, probs_rf_ovr)
metrics_model_rsf <- metrics_ovr_su(prob_mat_su = probs_rf_ovr,
                                    test_set = test_overall,
                                    p = 0.5)

cindex_rsf <-  1 - fit_test$err.rate[fit_test$ntree] # 0.700572

##########################################################
# Global performance measures at exact time points
##########################################################
```

```
obj_cox_full <- data.frame(Accuracy = metrics_cox_full$Accuracy,
                           Sensitivity =
                           metrics_cox_full$Sensitivity,
                           Specificity =
                           metrics_cox_full$Specificity,
                           Precision = metrics_cox_full$Precision,
                           F1score = metrics_cox_full$F1score,
                           Int_brier = brier_cox_full$Int_brier,
                           C_index = cindex_cox_full)

obj_cox_back <- data.frame(Accuracy = metrics_cox_back$Accuracy,
                           Sensitivity =
                           metrics_cox_back$Sensitivity,
                           Specificity =
                           metrics_cox_back$Specificity,
                           Precision = metrics_cox_back$Precision,
                           F1score = metrics_cox_back$F1score,
                           Int_brier = brier_cox_back$Int_brier,
                           C_index = cindex_cox_back)

obj_cox_lasso <- data.frame(Accuracy = metrics_cox_lasso$Accuracy,
                           Sensitivity =
                           metrics_cox_lasso$Sensitivity,
                           Specificity =
                           metrics_cox_lasso$Specificity,
                           Precision =
                           metrics_cox_lasso$Precision,
                           F1score = metrics_cox_lasso$F1score,
                           Int_brier = brier_cox_lasso$Int_brier,
                           C_index = cindex_cox_lasso)

obj_model_rsf <- data.frame(Accuracy = metrics_model_rsf$Accuracy,
                            Sensitivity =
                            metrics_model_rsf$Sensitivity,
                            Specificity =
                            metrics_model_rsf$Specificity,
                            Precision =
                            metrics_model_rsf$Precision,
                            F1score = metrics_model_rsf$F1score,
                            Int_brier = brier_model_rsf$Int_brier,
                            C_index = cindex_rsf)

df_comp <- round(data.frame(rbind(obj_cox_full, obj_cox_back,
                            obj_cox_lasso, obj_model_rsf)), 4)

rownames(df_comp) <- c("Cox all variables", "Cox backward",
                       "Cox LASSO", "RSF")
colnames(df_comp) <- c("Accuracy", "Sensitivity", "Specificity",
                      "Precision", "F1 score", "IBS", "C-index")
```

```
kable(
  df_comp,
  format = "latex", booktabs = T,
  digits = 4, align = "c",
  caption = paste0("label{tab:chap5_exact_comp_ovr}")
)


##########################################
# Prediction curves of error and C-index
##########################################

# check on training data
prederrorb <- pec(list("Cox backward" = cox_back),
                  data = training_overall,
                  formula = pec_f,
                  splitMethod = "none",
                  times = 0:12)

prederrorc <- pec(list("Cox LASSO" = cox_lasso),
                   data = training_ovr_extended,
                   formula = pec_f,
                   splitMethod = "none",
                   times = 0:12)


# check on test data
prederror2b <- pec(list("Cox backward" = cox_back),
                   traindata = training_overall,
                   data = test_overall, formula = pec_f,
                   splitMethod = "none",
                   times = 0:12)

prederror2c <- pec(list("Cox LASSO" = cox_lasso),
                    traindata = training_ovr_extended,
                    data = test_ovr_extended, formula = pec_f,
                    splitMethod = "none",
                    times = 0:12)

cat("Starting prediction error with bootCv ... ", "\n")
set.seed(12345)
prederror3b <- pec(list("Cox backward" = cox_back),
                   data = training_overall, formula = pec_f,
                   splitMethod = "BootCv", B = 100,
                   times = 0:12, verbose = TRUE,
                   reference = FALSE)

prederror3c <- pec(list("Cox Lasso" = cox_lasso),
                    data = training_ovr_extended, formula = pec_f,
                    splitMethod = "BootCv", B = 100,
                    times = 0:12, verbose = TRUE,
```

```
                    reference = FALSE)

cat("Starting prediction error with boostrap 632 ... ", "\n")
set.seed(12345)
prederror4b <- pec(list("Cox backward" = cox_back),
                   data = training_overall, formula = pec_f,
                   splitMethod = "Boot632", B = 100,
                   times = 0:12, verbose = TRUE,
                   reference = FALSE)

prederror4c <- pec(list("Cox Lasso" = cox_lasso),
                    data = training_ovr_extended, formula = pec_f,
                    splitMethod = "Boot632", B = 100,
                    times = 0:12, verbose = TRUE,
                    reference = FALSE)

################################################
# Apparent training error
################################################
line1a <- data.frame(times = prederror$time,
                     pe = prederror$AppErr$Reference,
                     Model = "Reference")
line2a <- data.frame(times = prederror$time,
                     pe = prederror$AppErr$`Cox full`,
                     Model = "Cox all variables")

line3a <- data.frame(times = prederrorb$time,
                     pe = prederrorb$AppErr$`Cox backward`,
                     Model = "Cox backward")
line4a <- data.frame(times = prederrorc$time,
                     pe = prederrorc$AppErr$`Cox LASSO`,
                     Model = "Cox LASSO")
line5a <- data.frame(times = prederror$time,
                     pe = prederror$AppErr$RSF,
                     Model = "RSF")

df1a <- rbind(line1a, line2a, line3a, line4a, line5a)

a <- ggplot(df1a, aes(x = times, y = pe, color = Model)) +
  geom_line(size = 0.9) +
  xlab("Time since transplantation in years") +
  ylab("Prediction error (Brier score)") +
  scale_x_continuous(breaks = 0:12, limits = c(0, 12)) +
  ylim(c(0, 0.3)) +
  theme_classic() +
  ggtitle("A ") +
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_color_manual(values=c("black", "red", "purple",
                              "green", "blue"))

################################################
```

```
# Generalization (test) error
################################################

line1b <- data.frame(times = prederror2$time,
                     pe = prederror2$AppErr$Reference,
                     Model = "Reference")
line2b <- data.frame(times = prederror2$time,
                     pe = prederror2$AppErr$`Cox full`,
                     Model = "Cox all variables")
line3b <- data.frame(times = prederror2b$time,
                     pe = prederror2b$AppErr$`Cox backward`,
                     Model = "Cox backward")
line4b <- data.frame(times = prederror2c$time,
                     pe = prederror2c$AppErr$`Cox LASSO`,
                     Model = "Cox LASSO")
line5b <- data.frame(times = prederror2$time,
                     pe = prederror2$AppErr$RSF,
                     Model = "RSF")

df1b <- rbind(line1b, line2b, line3b, line4b, line5b)

b <- ggplot(df1b, aes(x = times, y = pe, color = Model)) +
  geom_line(size = 0.9) +
  xlab("Time since transplantation in years") +
  ylab("Prediction error (Brier score)") +
  scale_x_continuous(breaks = 0:12, limits = c(0, 12)) +
  ylim(c(0, 0.3)) +
  theme_classic() +
  ggtitle("B ") +
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_color_manual(values=c("black", "red", "purple",
                              "green", "blue"))

################################################
# for Bootstrap cross-validation OOB train error
################################################

line1 <- data.frame(times = prederror3$time,
                    pe = prederror3$BootCvErr$Reference,
                    Model = "Reference")
line2 <- data.frame(times = prederror3$time,
                    pe = prederror3$BootCvErr$`Cox full`,
                    Model = "Cox all variables")
line3 <- data.frame(times = prederror3b$time,
                    pe = prederror3b$BootCvErr$`Cox backward`,
                    Model = "Cox backward")
line4 <- data.frame(times = prederror3c$time,
                    pe = prederror3c$BootCvErr$`Cox Lasso`,
                    Model = "Cox LASSO")
line5 <- data.frame(times = prederror3$time,
                    pe = prederror3$BootCvErr$RSF,
```

```
                    Model = "RSF")

df1 <- rbind(line1, line2, line3, line4, line5)

c <- ggplot(df1, aes(x = times, y = pe, color = Model)) +
  geom_line(size = 0.9) +
  scale_x_continuous(breaks = 0:12, limits = c(0, 12)) +
  xlab("Time since transplantation in years") +
  ylab("Prediction error (Brier score)") +
  ylim(c(0, 0.3)) +
  theme_classic() +
  ggtitle("C ") +
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_color_manual(values=c("black", "red", "purple",
                              "green", "blue"))

#####################################################
# Bootstrap-cross validation 0.632*(OOB train error)
# + 0.328*(apparent error)
#####################################################

line1d <- data.frame(times = prederror4$time,
                     pe = prederror4$Boot632Err$Reference,
                     Model = "Reference")
line2d <- data.frame(times = prederror4$time,
                     pe = prederror4$Boot632Err$`Cox full`,
                     Model = "Cox all variables")
line3d <- data.frame(times = prederror4b$time,
                     pe = prederror4b$Boot632Err$`Cox backward`,
                     Model = "Cox backward")
line4d <- data.frame(times = prederror4c$time,
                     pe = prederror4c$Boot632Err$`Cox Lasso`,
                     Model = "Cox LASSO")
line5d <- data.frame(times = prederror4$time,
                     pe = prederror4$Boot632Err$RSF,
                     Model = "RSF")

df1d <- rbind(line1d, line2d, line3d, line4d, line5d)

d <- ggplot(df1d, aes(x = times, y = pe, color = Model)) +
  geom_line(size = 0.9) +
  scale_x_continuous(breaks = 0:12, limits = c(0, 12)) +
  xlab("Time since transplantation in years") +
  ylab("Prediction error (Brier score)") +
  ylim(c(0, 0.3)) +
  theme_classic() +
  ggtitle("D ") +
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_color_manual(values=c("black", "red", "purple",
                              "green", "blue"))
grid.arrange(a, b, c, d, ncol = 2)
```

```
########################################
# Time-dependent C-index
########################################

cat("Starting C-index with none (train)  ... ", "\n")
cerror1b  <- pec::cindex(list("Cox backward" = cox_back),
                         formula = pec_f,
                         splitMethod = "none",
                         data = training_overall,
                         eval.times = c(0.1, seq(1, 12, 1)))

cerror1c  <- pec::cindex(list("Cox LASSO" = cox_lasso),
                          formula = pec_f,
                          splitMethod = "none",
                          data = training_ovr_extended,
                          eval.times = c(0.1, seq(1, 12, 1)))
plot(cerror1, xlab = "Time in years since transplantation",
     xlim = c(0, 12))

cat("Starting C-index with none (test)  ... ", "\n")
cerror2b  <- pec::cindex(list("Cox backward" = cox_back),
                         formula = pec_f,
                         splitMethod = "none",
                         data = test_overall,
                         eval.times = c(0.1, seq(1, 12, 1)))

cerror2c  <- pec::cindex(list("Cox LASSO" = cox_lasso),
                          formula = pec_f,
                          splitMethod = "none",
                          data = test_ovr_extended,
                          eval.times = c(0.1, seq(1, 12, 1)))

plot(cerror2, xlab = "Time in years since transplantation",
     xlim = c(0, 12))

# bootcv C-index
cat("Starting C-index with BootCv... ", "\n")
set.seed(12345)
cerror3b  <- pec::cindex(list("Cox backward" = cox_back),
                         formula = pec_f,
                         splitMethod = "BootCv",
                         B = 100,
                         data = training_overall,
                         pred.times = c(0.1, seq(1, 12, 1)),
                         eval.times = c(0.1, seq(1, 12, 1)))

cerror3c  <- pec::cindex(list("Cox LASSO" = cox_lasso),
                          formula = pec_f,
                          splitMethod = "BootCv",
                          B = 100,
```

```
                          data = training_ovr_extended,
                          pred.times = c(0.1, seq(1, 12, 1)),
                          eval.times = c(0.1, seq(1, 12, 1)))

cat("Starting C-index with Boot632... ", "\n")
set.seed(12345)
cerror4b  <- pec::cindex(list("Cox backward" = cox_back),
                         formula = pec_f,
                         splitMethod = "Boot632",
                         B = 100,
                         data = training_overall,
                         pred.times = c(0.1, seq(1, 12, 1)),
                         eval.times = c(0.1, seq(1, 12, 1)))


cerror4c  <- pec::cindex(list("Cox LASSO" = cox_lasso),
                          formula = pec_f,
                          splitMethod = "Boot632",
                          B = 100,
                          data = training_ovr_extended,
                          pred.times = c(0.1, seq(1, 12, 1)),
                          eval.times = c(0.1, seq(1, 12, 1)))

##########################################
# Apparent C-index
##########################################

line6a <- data.frame(times = cerror1$pred.time,
                     pe = cerror1$AppCindex$Reference,
                     Model = "Reference")
line7a <- data.frame(times = cerror1$pred.time,
                     pe = cerror1$AppCindex$`Cox full`,
                     Model = "Cox all variables")
line8a <- data.frame(times = cerror1b$pred.time,
                     pe = cerror1b$AppCindex$`Cox backward`,
                     Model = "Cox backward")
line9a <- data.frame(times = cerror1c$pred.time,
                     pe = cerror1c$AppCindex$`Cox LASSO`,
                     Model = "Cox LASSO")

line10a <- data.frame(times = cerror1$pred.time,
                      pe = cerror1$AppCindex$RSF,
                      Model = "RSF")

df2a <- rbind(line6a, line7a, line8a, line9a, line10a)

e <- ggplot(df2a, aes(x = times, y = pe, color = Model)) +
  geom_line(size = 0.9) +
  scale_x_continuous(breaks = 0:12) +
  scale_y_continuous(breaks = seq(0.4, 1, 0.1),
  limits = c(0.4, 1)) +
```

```
  xlab("Time since transplantation in years") +
  ylab("Concordance index") +
  theme_classic() +
  ggtitle("A") +
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_color_manual(values=c("black", "red", "purple",
                              "green", "blue"))

##############################################
# C-index on test set
##############################################

line6b <- data.frame(times = cerror2$pred.time,
                     pe = cerror2$AppCindex$Reference,
                     Model = "Reference")
line7b <- data.frame(times = cerror2$pred.time,
                     pe = cerror2$AppCindex$`Cox full`,
                     Model = "Cox all variables")
line8b <- data.frame(times = cerror2b$pred.time,
                     pe = cerror2b$AppCindex$`Cox backward`,
                     Model = "Cox backward")
line9b <- data.frame(times = cerror2c$pred.time,
                     pe = cerror2c$AppCindex$`Cox LASSO`,
                     Model = "Cox LASSO")

line10b <- data.frame(times = cerror2$pred.time,
                      pe = cerror2$AppCindex$RSF,
                      Model = "RSF")


df2b <- rbind(line6b, line7b, line8b, line9b, line10b)

f <- ggplot(df2b, aes(x = times, y = pe, color = Model)) +
  geom_line(size = 0.9) +
  scale_x_continuous(breaks = 0:12) +
  scale_y_continuous(breaks = seq(0.4, 1, 0.1),
  limits = c(0.4, 1)) +
  xlab("Time since transplantation in years") +
  ylab("Concordance index") +
  theme_classic() +
  ggtitle("B") +
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_color_manual(values=c("black", "red", "purple",
                              "green", "blue"))

##################################################
# bootstrap cross-validated C-index
##################################################

line6c <- data.frame(times = cerror4$pred.time,
                     pe = cerror4$BootCvCindex$Reference,
```

```
                    Model = "Reference")
line7c <- data.frame(times = cerror4$pred.time,
                    pe = cerror4$BootCvCindex$`Cox full`,
                    Model = "Cox all variables")
line8c <- data.frame(times = cerror4b$pred.time,
                    pe = cerror4b$BootCvCindex$`Cox back`,
                    Model = "Cox backward")
line9c <- data.frame(times = cerror4c$pred.time,
                    pe = cerror4c$BootCvCindex$`Cox LASSO`,
                    Model = "Cox LASSO")

line10c <- data.frame(times = cerror4$pred.time,
                     pe = cerror4$BootCvCindex$RSF,
                     Model = "RSF")

df2c <- rbind(line6c, line7c, line8c, line9c, line10c)
g <- ggplot(df2c, aes(x = times, y = pe, color = Model)) +
  geom_line(size = 0.9) +
  scale_x_continuous(breaks = 0:12) +
  scale_y_continuous(breaks = seq(0.4, 1, 0.1),
  limits = c(0.4, 1)) +
  xlab("Time since transplantation in years") +
  ylab("Concordance index") +
  theme_classic() +
  ggtitle("C") +
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_color_manual(values=c("black", "red", "purple",
                              "green", "blue"))

############################################################
# Bootstrap-cross validation 0.632*(OOB train C-index)
# + 0.328*(apparent C-index)
############################################################

line6 <- data.frame(times = cerror4$pred.time,
                    pe = cerror4$Boot632Cindex$Reference,
                    Model = "Reference")
line7 <- data.frame(times = cerror4$pred.time,
                    pe = cerror4$Boot632Cindex$`Cox full`,
                    Model = "Cox all variables")
line8 <- data.frame(times = cerror4b$pred.time,
                    pe = cerror4b$Boot632Cindex$`Cox back`,
                    Model = "Cox backward")
line9 <- data.frame(times = cerror4c$pred.time,
                    pe = cerror4c$Boot632Cindex$`Cox LASSO`,
                    Model = "Cox LASSO")
line10 <- data.frame(times = cerror4$pred.time,
                    pe = cerror4$Boot632Cindex$RSF,
                    Model = "RSF")

df2 <- rbind(line6, line7, line8, line9, line10)
```

```
h <- ggplot(df2, aes(x = times, y = pe, color = Model)) +
  geom_line(size = 0.9) +
  scale_x_continuous(breaks = 0:12) +
  scale_y_continuous(breaks = seq(0.4, 1, 0.1),
  limits = c(0.4, 1)) +
  xlab("Time since transplantation in years") +
  ylab("Concordance index") +
  theme_classic() +
  ggtitle("D") +
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_color_manual(values=c("black", "red", "purple",
                              "green", "blue"))
grid.arrange(e, f, g, h, ncol = 2)

#####################################################
#prediction curves for hypothetical patients
#####################################################

# make a function that identifies the nature of your variable

most_common <- function(x) {
  if (is.numeric(x)) return(median(x))
  else {
    tx <- table(x)
    m <- which(tx == max(tx))[1]
    fac <- factor(names(m), levels = levels(x))
    return(fac)
  }
}

vars <- colnames(training_overall)[1:106]
freqa <- lapply(vars, function(x)
                most_common(training_overall[, x]))

df_synth_normal <- do.call(cbind.data.frame, freqa)
colnames(df_synth_normal) <- vars

# repeat the data frame 6 times
df_synth_normal <- df_synth_normal[rep(seq_len(1), times = 6), ]

#########################################################
# survival curves for tumor and recipient age
#########################################################

# predictions for new synthetic patients
tumor <- factor(levels(training_overall$rec_tumor))
age <- quantile(training_overall$recipientage, c(0.10, 0.5, 0.90))

# create new hypothetic patient
combis <- with(training_overall, expand.grid(
```

```
  rec_tumor = tumor, # increasing age to different
  recipientage = age
))

df_synth_normal$recipientage <- combis$recipientage
df_synth_normal$rec_tumor <- combis$rec_tumor

df_synth_expanded$recipientage <- combis$recipientage
df_synth_expanded$rec_tumorY <- as.numeric(combis$rec_tumor) - 1

synth_cox_full <- predictSurvProb(object = cox_full,
                                  newdata = df_synth_normal,
                                  times = time_su)

time_val <- c(0, time_su)

# plots for Cox model
cols <- c("black", "red", "green", "lightblue", "blue", "purple")
plot(time_val, c(1, synth_cox_full[1, ]),
     type = "l",
     ylim = c(0,1),
     col = cols[1],
     lwd = 0.5,
     xlab = "Years since transplantation",
     ylab = "survival probability",
     main =
     "Survival Curves for hypothetical patients with Cox full")

for (i in 2:nrow(synth_cox_full)){
  lines(time_val, c(1, synth_cox_full[i, ]), type = "l",
        col = cols[i])
}
legend("bottomleft", paste("Age: ", df_synth_normal$recipientage,
                           ", Tumor: ",
                           df_synth_normal$rec_tumor),
       lty = 1, col = cols, bty = "o", cex = 0.8)

model_rsf <- rfsrc(Surv(patientsurvival, death) ~ .,
                   splitrule = "logrank", nsplit = 5,
                   data = training_overall, ntree = 250,
                   seed = -12345, mtry = 35, nodesize = 50,
                   ntime = 500, sampsize = 5000,
                   forest = TRUE, importance = FALSE)

model_names <- c("Cox all variables", "Cox backward",
                 "Cox Lasso", "RSF")
cols <- c("black", "red", "green2", "blue")

#jpeg("plot1.png", width = 600)
pdf(file = "Rplots1.pdf")
par(mfrow=c(2, 3))
```

```
lapply(1:6, function(x) {

  plotPredictSurvProb(cox_full, newdata = df_synth_normal[x, ],
                      lty = 1, col = "black", legend = TRUE,
                      xlab = "Time since trasplantation in years")

  title(paste0("Age: ", df_synth_normal$recipientage[x],
               ", Tumor: ", df_synth_normal$rec_tumor[x]))
  legend("bottomleft", legend = model_names,
         lty = 1:4, col = cols, bty = "o", cex = 0.8)
  plotPredictSurvProb(cox_back, newdata = df_synth_normal[x, ],
                      add = TRUE, lty = 2, col = "red")
  plotPredictSurvProb(cox_lasso, newdata = df_synth_expanded[x, ],
                      add = TRUE, lty = 3, col = "green2")
  plotPredictSurvProb(model_rsf, newdata = df_synth_normal[x, ],
                      add = TRUE, lty = 4, col = "blue")
})
dev.off()


#################################################################
# hypothetical patients with rec_immuno meds and length of stay
#################################################################

# survival curves for immuno meds and length of stay

# predictions for new synthetic patients
immuno_meds <-
factor(levels(training_overall$rec_immuno_maint_meds))
los <- quantile(training_overall$rec_postx_los,
                c(0.10, 0.5, 0.90))

# create new hypothetic patient
combis2 <- with(training_overall, expand.grid(
# increasing age to different
  rec_immuno_maint_meds = immuno_meds,
  rec_postx_los = los
))

df_synth2_normal$rec_immuno_maint_meds <-
combis2$rec_immuno_maint_meds
df_synth2_normal$rec_postx_los <- combis2$rec_postx_los

df_synth2_expanded$rec_immuno_maint_medsY <-
as.numeric(combis2$rec_immuno_maint_meds) - 1
df_synth2_expanded$rec_postx_los <- combis2$rec_postx_los

synth_cox_full2 <- predictSurvProb(object = cox_full,
                                   newdata = df_synth2_normal,
                                   times = time_su)
time_val <- c(0, time_su)
```

```
# plots for Cox model
cols <- c("black", "red", "green", "lightblue", "blue", "purple")
plot(time_val, c(1, synth_cox_full2[1, ]),
     type = "l",
     ylim = c(0,1),
     col = cols[1],
     lwd = 0.5,
     xlab = "Years since transplantation",
     ylab = "survival probability",
     main =
     "Survival Curves for hypothetical patients with Cox full")

for (i in 2:nrow(synth_cox_full2)){
  lines(time_val, c(1, synth_cox_full2[i, ]),
        type = "l", col = cols[i])
}
legend("bottomleft", paste("Immuno meds: ",
                           df_synth2_normal$rec_immuno_maint_meds,
                           ", LOS: ",
                           df_synth2_normal$rec_postx_los),
       lty = 1, col = cols, bty = "o", cex = 0.6)

model_rsf <- rfsrc(Surv(patientsurvival, death) ~ .,
                   splitrule = "logrank", nsplit = 5,
                   data = training_overall, ntree = 250,
                   seed = -12345, mtry = 35, nodesize = 50,
                   ntime = 500, sampsize = 5000,
                   forest = TRUE, importance = FALSE)

model_names <- c("Cox all variables", "Cox backward",
                 "Cox Lasso", "RSF")
cols <- c("black", "red", "green3", "blue")
legend_pos <- c("topright", "bottomleft", "topright",
                "bottomleft", "topright", "bottomleft")

pdf(file = "Rplots2.pdf")
par(mfrow=c(2, 3))
lapply(1:6, function(x) {

  plotPredictSurvProb(cox_full, newdata = df_synth2_normal[x, ],
                      lty = 1, col = "black", legend = TRUE,
                      xlab = "Time since trasplantation in years")

  title(paste("Immuno meds: ",
              df_synth2_normal$rec_immuno_maint_meds[x],
              ", LOS: ", df_synth2_normal$rec_postx_los[x]))
  legend(legend_pos[x], legend = model_names,
         lty = 1:4, col = cols, bty = "o", cex = 0.8)
  plotPredictSurvProb(cox_back, newdata = df_synth2_normal[x, ],
                      add = TRUE, lty = 2, col = "red")
```

```
  plotPredictSurvProb(cox_lasso,
                      newdata = df_synth2_expanded[x, ],
                      add = TRUE, lty = 3, col = "green3")
  plotPredictSurvProb(model_rsf, newdata = df_synth2_normal[x, ],
                      add = TRUE, lty = 4, col = "blue")
})
dev.off()

#####################################################
# synthetic dataset in expanded format
#####################################################
df_synth_expanded <- data.frame(model.matrix(~. ,
                                data = df_synth_normal)[, -1])

# for one variable
expand <- model.matrix(~. , data = df_synth_normal)[, -1]
expand_df <- data.frame(t(expand))
rownames(expand_df)

training_overall_x <- training_overall[, 1:106]
for(i in 1:length(colnames(training_overall_x))) {
  if(class(training_overall_x[,i]) == "numeric" ||
class(training_overall_x[,i]) == "integer") {
    training_overall_x[, i] <-
    as.vector(scale(training_overall_x[, i])) }
}

# synthetic dataset for scaled format of the data

freqb <- lapply(vars, function(x) most_common(
                training_overall_x[, x]))

df_synth_scaled <- do.call(cbind.data.frame, freqb)
colnames(df_synth_scaled) <- vars
df_synth_scaled <- df_synth_scaled[rep(seq_len(1), times = 6), ]
df_synth_scaled <- data.frame(model.matrix(~.,
                              data = df_synth_scaled)[, -1])
colnames(df_synth_scaled)
```

D.3.3 Comparisons at discrete time points - intervals: Overall survival

```
# Hint: to download keras library you need to
# have installed the Anaconda3 and TensorFlow for R
install_packages <- c("glmnet", "keras", "randomForestSRC",
                      "pec", "survival", "caret")

for (i in 1:length(install_packages)){
  if (!install_packages[i] %in% installed.packages()){
    install.packages(install_packages[i])
  }
}
library(glmnet)
library(keras)
library(randomForestSRC)
library(pec)
library(survival)
library(caret)
source("https://bioconductor.org/biocLite.R")
biocLite("survcomp")
library(survcomp)

use_session_with_seed(seed = 1235, disable_gpu = TRUE,
                      disable_parallel_cpu = TRUE,
                      quiet = FALSE)

source("the_functions.R")
load("unos_overall.RData")
unos_overall$patientsurvival <- unos_overall$patientsurvival
                                  + (1/365)

unos_overall$patientsurvival <-
as.numeric(cut(unos_overall$patientsurvival,
           breaks = 12))

set.seed(12345)
N <- nrow(unos_overall)
index <- sample(1:N, round(N/3), replace = FALSE)
training_overall <- unos_overall[-index, ] # create training set
test_overall <- unos_overall[index, ] # create test set

# Cox full and Cox KM
cox_full <- coxph(Surv(patientsurvival, death) ~.,
                  data = training_overall, x = TRUE, y = TRUE)

cox_null <- coxph(Surv(patientsurvival, death) ~ 1,
                  data = training_overall, x = TRUE, y = TRUE)

pi_cox_full <- predict(cox_full, newdata = test_overall)
```

```
cindex_cox_full <-  concordance.index(x = pi_cox_full,
                                      surv.time =
                                     test_overall$patientsurvival,
                                      surv.event =
                                      test_overall$death)$c.index

# model cox backward
cox_back <- selectCox(formula = Surv(patientsurvival, death) ~.,
                      data = training_overall, rule = "aic")
vars_back <- cox_back$In # variables
f_back <- as.formula(paste("Surv(patientsurvival, death) ~",
                           paste(vars_back, collapse = "+")))
# Cox backward in coxph object
cox_back <- coxph(formula = f_back, x = TRUE, y = TRUE,
                  data = training_overall, method = "breslow")

pi_cox_back <- predict(cox_back, newdata = test_overall)

cindex_cox_back <-  concordance.index(x = pi_cox_back,
                                      surv.time =
                                     test_overall$patientsurvival,
                         surv.event = test_overall$death)$c.index

# model Lasso
response_pair <- c("patientsurvival", "death")

# for the training set
X_train <- training_overall[,
               !(colnames(training_overall) %in% response_pair)]
X_train_ovr <- model.matrix(~., X_train)[, -1]
# dim(X_train_ovr) # 41529 x 129
Y_train_ovr <- training_overall[,
colnames(training_overall) %in% response_pair] # create matrix Y
Y_survtrain_ovr <- Surv(Y_train_ovr$patientsurvival,
                        Y_train_ovr$death)

# for the test set
x_test <- test_overall[, !(colnames(test_overall) %in% response_pair)]
X_test_ovr <- model.matrix(~., x_test)[, -1]
Y_test_ovr <- test_overall[, colnames(test_overall) %in% response_pair]
# Surv function packages survival data into
# the form expected by glmnet
Y_survtest_ovr <- Surv(Y_test_ovr$patientsurvival,
                       Y_test_ovr$death)

# use these two datasets to fit Cox model in the selected features
training_ovr_extended <- as.data.frame(cbind(X_train_ovr,
                                             Y_train_ovr))
test_ovr_extended <- as.data.frame(cbind(X_test_ovr, Y_test_ovr))
```

```
seed <- 12345
set.seed(seed)
folds <- createFolds(y = training_overall$patientsurvival,
                     k = 10, list = FALSE)

cv.fit <- cv.glmnet(x = X_train_ovr, y = Y_survtrain_ovr,
                    foldid = folds, parallel = TRUE,
                    family = "cox", grouped = TRUE,
                    maxit = 1000)
cv.fit$lambda.1se

active_index <- as.numeric(unlist(predict(cv.fit,
                                          newx = X_test_ovr,
                                          s = "lambda.1se",
                                          type = "nonzero")))
names_coefs <- colnames(test_ovr_extended)[1:128]
vars_active <- names_coefs[active_index]

f_lasso <- as.formula(paste("Surv(patientsurvival, death) ~",
                            paste(vars_active, collapse = "+")))

cox_lasso <- coxph(formula = f_lasso, x = TRUE, y = TRUE,
                   data = training_ovr_extended)

pi_cox_lasso <- predict(cox_lasso, newdata = test_ovr_extended)

cindex_cox_lasso <-  concordance.index(x = pi_cox_lasso,
                        surv.time = test_overall$patientsurvival,
                        surv.event = test_overall$death)$c.index

cox_pair <- as.data.frame(cbind(test_ovr_extended$patientsurvival,
                                test_ovr_extended$death))
colnames(cox_pair) <- c("time", "status")

time_su <- sort(unique(test_ovr_extended$patientsurvival))

# find the probabilities of the models
cox_probs_null <- predictSurvProb(cox_null, test_overall,
                                  times = time_su)

cox_probs_full <- predictSurvProb(cox_full, test_overall,
                                  times = time_su)

brier_cox_full <-  brier_general_su(cox_pair, cox_probs_full)
metrics_cox_full <-  metrics_ovr_su(prob_mat_su = cox_probs_full,
                                    test_set = test_overall,
                                    p = 0.5)

cox_probs_back <- predictSurvProb(cox_back, test_overall,
                                  times = time_su)
```

```
brier_cox_back <-  brier_general_su(cox_pair, cox_probs_back)
metrics_cox_back <-  metrics_ovr_su(prob_mat_su = cox_probs_back,
                                    test_set = test_overall,
                                    p = 0.5)

cox_probs_lasso <- predictSurvProb(cox_lasso, test_ovr_extended,
                                   times = time_su)

brier_cox_lasso <-  brier_general_su(cox_pair, cox_probs_lasso)
metrics_cox_lasso <-  metrics_ovr_su(prob_mat_su =
                                     cox_probs_lasso,
                                     test_set = test_ovr_extended,
                                     p = 0.5)

# Random survival forests
model_rsf <- rfsrc(Surv(patientsurvival, death) ~ .,
                   splitrule = "logrank", nsplit = 7,
                   data = training_overall, ntree = 300,
                   seed = -12345, mtry = 23, nodesize = 50,
                   ntime = 500, sampsize = 5000,
                   forest = TRUE, importance = FALSE
)

fit_test <- predict(model_rsf,
                    newdata = test_overall,
                    seed = -12345,
                    split.depth = "all.trees",
                    forest = FALSE)
cindex_rsf <-  1 - fit_test$err.rate[fit_test$ntree] # 0.6942

# always put the times in sorted order
probs_rf_ovr <- predictSurvProb(object = model_rsf,
                                newdata = test_overall,
                                times =
                          sort(unique(test_overall$patientsurvival)))

pair_rfsrc <- as.data.frame(cbind(test_overall$patientsurvival,
                                  test_overall$death))
colnames(pair_rfsrc) <- c("time", "status")
brier_model_rsf <-  brier_general_su(pair_rfsrc, probs_rf_ovr)
metrics_model_rsf <- metrics_ovr_su(prob_mat_su = probs_rf_ovr,
                                    test_set = test_overall,
                                    p = 0.5)

# Fit the neural networks

training_ovr_scaled <- read.table("training_ovr_scaled.txt")
test_ovr_scaled <- read.table("test_ovr_scaled.txt")
load("training_ovr_long.Rdata")
load("test_ovr_long.Rdata")
```

```
data_test_creator <- function(data){

  N <- nrow(data)
  # assign survival times to 12 intervals, each is a 1-year period
  data$interval <- max(as.numeric(cut(data$patientsurvival,
                       breaks = 12)))
  data$survival <- as.numeric(cut(data$patientsurvival,
                    breaks = 12)) # the true interval survival
  data$id <- 70001:(70000 + N) # define the patient ids abstractly
  n.times <- data$interval
  data_long <- data[rep(seq_len(N), times = n.times), ]

  # create the correct intervals
  for(i in unique(data_long$id)) {
    n_length <- length(data_long$interval[data_long$id == i])
    data_long$interval[data_long$id == i] <- 1:n_length
  }

  data_long$status <- vector(mode = "numeric",
                             length = nrow(data_long))

  # put indication 1 on status at the intervals
  # on which a patient has died
  for (i in 1:nrow(data_long)) {
    if (data_long$death[i] == 1 &&
        data_long$survival[i] <= data_long$interval[i])
        data_long$status[i] <- 1
  }

  return(data_long)
}

measures_calculator <- function(trained_model,
                                datanew, real_status) {

  df1 <- data.frame(hazard = predict_proba(trained_model,
                                           as.matrix(datanew[,
                                           c(1:128, 131)]),
                                           batch_size = 500))
  df1$id <- datanew$id # ids of the patients
  df1$survival <- datanew$survival # survival time in years
  groups <- split(df1, f = df1$id)

  true_surv <- unlist(lapply(groups, function(x) {
    surv_obj <- x$survival
    true_res <- surv_obj[1]
    return(true_res)}
  ))
  group_probs <- lapply(groups, function(x) {
  x <- cumprod(1 - x$hazard)})
  pred_mat <- do.call("rbind", group_probs)
```

```
relative_probs <- vector(mode = "numeric",
length = length(group_probs))
for (i in 1:length(group_probs)){
  temp <- group_probs[[i]]
  ind <- which(1:12 == true_surv[i])
  relative_probs[i] <- temp[ind]
}
N0 <- length(unique(df1$id)) # number of unique persons

# in the data frame create random id numbers
# to label the patients
df2 <- data.frame(relative_probs, true_surv,
status = real_status,
id = (70001):(70000 + N0))
df2$prediction <- 1 - round(df2$relative_probs, digits = 0)
# create possible classes
classes <- c(0, 1)
# auxiliary confusion table
tabel <- table(factor(df2$prediction, levels = classes),
               factor(df2$status, levels = classes))
confusion_mat <- confusionMatrix(tabel, positive = "1")
accuracy <- sum(df2$prediction == df2$status) / nrow(df2)
sensitivity <- sum(df2$status == 1 & df2$prediction == 1) /
                   colSums(tabel)[2]
specificity <- sum(df2$status == 0 & df2$prediction == 0) /
                   colSums(tabel)[1]
precision <- as.numeric(confusion_mat$byClass[5])
recall <- as.numeric(confusion_mat$byClass[6])
f1score <- as.numeric(confusion_mat$byClass[7])

brier_obj <- brier_nnet(df2, prob_matrix = pred_mat)
int_brier <- as.numeric(brier_obj$Int_brier)
brier_set <- brier_obj$Brier
all_weights <- get_weights(fit_keras)
nr_weights <- (length(all_weights[[1]]) +
               length(all_weights[[2]])
               + length(all_weights[[3]])
               + length(all_weights[[4]]))

return(list(weights = nr_weights,
            node_size = length(get_weights(fit_keras)[[2]]),
            cross_entropy =
            result$metrics$loss[result$params$epochs],
            accuracy = accuracy,
            sensitivity = as.numeric(sensitivity),
            specificity = as.numeric(specificity),
            Precision = precision, Recall = recall,
            F1score = f1score,
            Integrated_brier = int_brier,
            Brier_scores = brier_set))
```

```
}

# model with one hidden layer
set.seed(12345)

fit_keras <- keras_model_sequential()
# Add layers to the model

# we create a densely connected ANN to the output
fit_keras %>%
  layer_dense(units = 90, activation = 'relu',
  input_shape = c(129)) %>%
  layer_dropout(rate = 0.1) %>%
  layer_dense(units = 1, activation = 'sigmoid')
fit_keras %>% compile(
  loss = 'binary_crossentropy', # for binary class classification
  optimizer = optimizer_sgd(lr = 0.2, momentum = 0.9)
)
event_status <- test_ovr_scaled$death

# create the matrices to be used for keras library
# predictors: 128 variables + interval
train_x <- as.matrix(training_ovr_long[, c(1:128, 131)])
dimnames(train_x) <- NULL # the object must have empty dimnames
train_y <- training_ovr_long$status
test_x <- as.matrix(test_ovr_long[, c(1:128, 131)])
dimnames(test_x) <- NULL # the object must have empty dimnames
test_y <- test_ovr_long$status

result <- fit_keras %>% fit(
  train_x,
  train_y,
  epochs = 10,
  batch_size = 1000,
  validation_data = list(test_x, test_y),
  class_weight = list("0" = 1, "1" = 1)
  #,callbacks = c(early_stopping)
)

# now that the model has run lets calculate the measures
metrics_model_nn1h <- measures_calculator(trained_model =
                                            fit_keras,
                                            datanew = test_ovr_long,
                                            real_status =
                                            event_status)

df1 <- data.frame(hazard = predict_proba(fit_keras,
                                           as.matrix(test_ovr_long[,
                                           c(1:128, 131)]),
                                           batch_size = 500))
df1$id <- test_ovr_long$id # ids of the patients
```

```
groups <- split(df1, f = df1$id)

group_probs <- lapply(groups, function(x) {
                      x <- cumprod(1 - x$hazard)})
pred_mat_nn1h <- do.call("rbind", group_probs)

set.seed(12345)
# model with 2 hidden layers
# find the metrics for combination node_size = 100,
#dropout rate = 0.1,
# learning rate 0.2, momentum 0.9, weak class weight 1
fit_keras2 <- keras_model_sequential()
# we create a densely connected ANN to the output
fit_keras2 %>%
  layer_dense(units = 100, activation = 'relu',
  input_shape = c(129)) %>%
  layer_dropout(rate = 0.1) %>%
  layer_dense(units = 100, activation = 'relu') %>%
  layer_dropout(rate = 0.1) %>%
  layer_dense(units = 1, activation = 'sigmoid')
fit_keras2 %>% compile(
  loss = 'binary_crossentropy', # for binary class classification
  optimizer = optimizer_sgd(lr = 0.2, momentum = 0.9)
)

event_status <- test_ovr_scaled$death

# create the matrices to be used for keras library
# predictors: 128 variables + interval
train_x <- as.matrix(training_ovr_long[, c(1:128, 131)])
dimnames(train_x) <- NULL # the object must have empty dimnames
train_y <- training_ovr_long$status
test_x <- as.matrix(test_ovr_long[, c(1:128, 131)])
dimnames(test_x) <- NULL # the object must have empty dimnames
test_y <- test_ovr_long$status

result <- fit_keras2 %>% fit(
  train_x,
  train_y,
  epochs = 10,
  batch_size = 1000,
  validation_data = list(test_x, test_y),
  class_weight = list("0" = 1, "1" = 1)
  #,callbacks = c(early_stopping)
)

metrics_model_nn2h <- measures_calculator(trained_model =
                                          fit_keras2,
                                          datanew = test_ovr_long,
                                          real_status =
                                          event_status)
```

```
df2 <- data.frame(hazard = predict_proba(fit_keras2,
                                         as.matrix(test_ovr_long[,
                                         c(1:128, 131)]),
                                         batch_size = 1000))

df2$id <- test_ovr_long$id # ids of the patients
df2$survival <- test_ovr_long$survival # survival time in years
groups2 <- split(df2, f = df2$id)
group_probs2 <- lapply(groups2, function(x) {
                       x <- cumprod(1 - x$hazard)})
pred_mat_nn2h <- do.call("rbind", group_probs2)

##########################################################
# Global performance measures at distinct time points
##########################################################

obj_cox_full <- c(metrics_cox_full$Accuracy,
                  metrics_cox_full$Sensitivity,
                  metrics_cox_full$Specificity,
                  metrics_cox_full$Precision,
                  metrics_cox_full$F1score,
                  brier_cox_full$Int_brier,
                  cindex_cox_full)

obj_cox_back <- c(metrics_cox_back$Accuracy,
                  metrics_cox_back$Sensitivity,
                  metrics_cox_back$Specificity,
                  metrics_cox_back$Precision,
                  metrics_cox_back$F1score,
                  brier_cox_back$Int_brier,
                  cindex_cox_back)

obj_cox_lasso <- c(metrics_cox_lasso$Accuracy,
                  metrics_cox_lasso$Sensitivity,
                  metrics_cox_lasso$Specificity,
                  metrics_cox_lasso$Precision,
                  metrics_cox_lasso$F1score,
                  brier_cox_lasso$Int_brier,
                  cindex_cox_lasso)

obj_model_rsf <- c(metrics_model_rsf$Accuracy,
                   metrics_model_rsf$Sensitivity,
                   metrics_model_rsf$Specificity,
                   metrics_model_rsf$Precision,
                   metrics_model_rsf$F1score,
                   brier_model_rsf$Int_brier,
                   cindex_rsf)

obj_model_nn1h <- c(metrics_model_nn1h$accuracy,
                   metrics_model_nn1h$sensitivity,
```

```
                    metrics_model_nn1h$specificity,
                    metrics_model_nn1h$Precision,
                    metrics_model_nn1h$F1score,
                    metrics_model_nn1h$Integrated_brier)

obj_model_nn2h <- c(metrics_model_nn2h$accuracy,
                    metrics_model_nn2h$sensitivity,
                    metrics_model_nn2h$specificity,
                    metrics_model_nn2h$Precision,
                    metrics_model_nn2h$F1score,
                    metrics_model_nn2h$Integrated_brier)

df_final <- round(data.frame(rbind(obj_cox_full, obj_cox_back,
                                   obj_cox_lasso, obj_model_rsf,
                                   obj_model_nn1h, obj_model_nn2h)),
                                   4)

colnames(df_final) <- c("Accuracy", "Sensitivity", "Specificity",
                        "Precision", "F1 score", "IBS", "C-index")
rownames(df_final) <- c("Cox all variables", "Cox backward",
                        "Cox LASSO", "RSF",
                        "Neural Network 1h", "Neural Network 2h")
df_final$`C-index`[5:6] <- c("-", "-")

##########################################
# Time-dependent Brier score plot
##########################################

line1 <- data.frame(times = time_su,
                    brier = brier_cox_full$Brier,
                    Model = "Cox all variables")
line2 <- data.frame(times = time_su,
                    brier = brier_cox_back$Brier,
                    Model = "Cox backward")
line3 <- data.frame(times = time_su,
                    brier = brier_cox_lasso$Brier,
                    Model = "Cox LASSO")
line4 <- data.frame(times = time_su,
                    brier = brier_model_rsf$Brier,
                    Model = "RSF")
line5 <- data.frame(times = time_su,
                    brier = metrics_model_nn1h$Brier_scores,
                    Model = "Neural Network 1h")
line6 <- data.frame(times = time_su,
                    brier = metrics_model_nn2h$Brier_scores,
                    Model = "Neural Network 2h")

df <- rbind(line1, line2, line3, line4, line5, line6)

a1 <- ggplot(df, aes(x = times, y = brier, color = Model)) +
  geom_line(size = 0.9) +
```

```
  scale_x_continuous(breaks = 0:12) +
  xlab("Time since transplantation in years") +
  ylab("Prediction error (Brier score)") +
  ylim(c(0, 0.3)) +
  theme_classic() +
  ggtitle(" ") +
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_color_manual(values=c("black", "red", "purple",
                              "blue", "green", "darkgreen"))

##############################################
# mean survival probabilities plot
##############################################

mean_probs_cox_null <- colMeans(cox_probs_null)
mean_probs_cox_full <- colMeans(cox_probs_full)
mean_probs_cox_back <- colMeans(cox_probs_back)
mean_probs_cox_lasso <- colMeans(cox_probs_lasso)
mean_probs_rsf <- colMeans(probs_rf_ovr)
mean_probs_nn1h <- colMeans(pred_mat_nn1h)
mean_probs_nn2h <- colMeans(pred_mat_nn2h)

line0 <- data.frame(times = time_su, prob = mean_probs_cox_null,
                    Model = "Reference")
line7 <- data.frame(times = time_su, prob = mean_probs_cox_full,
                    Model = "Cox all variables")
line8 <- data.frame(times = time_su, prob = mean_probs_cox_back,
                    Model = "Cox backward")
line9 <- data.frame(times = time_su, prob = mean_probs_cox_lasso,
                    Model = "Cox LASSO")
line10 <- data.frame(times = time_su, prob = mean_probs_rsf,
                    Model = "RSF")
line11 <- data.frame(times = time_su, prob = mean_probs_nn1h,
                    Model = "Neural Network 1h")
line12 <- data.frame(times = time_su, prob = mean_probs_nn2h,
                    Model = "Neural Network 2h")

df2 <- rbind(line7, line8, line9, line10, line11, line12)

a2 <- ggplot(df2, aes(x = times, y = prob, color = Model)) +
  geom_line(size = 0.9) +
  scale_x_continuous(breaks = 0:12) +
  xlab("Time since transplantation in years") +
  ylab("Survival probability") +
  ylim(c(0, 1)) +
  theme_classic() +
  ggtitle(" ") +
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_color_manual(values=c("black", "red", "purple",
                              "blue", "green", "darkgreen"))
grid.arrange(a1, a2)
```

```
################################################
# predictions on new synthetic patients
################################################

tumor <- factor(levels(test_overall$rec_tumor))
age <- quantile(training_overall$recipientage, c(0.10, 0.5, 0.90))

# create new hypothetic patient
combis <- with(training_overall, expand.grid(
  rec_tumor = tumor, # increasing age to different
  recipientage = age
))

df_synth_normal$recipientage <- combis$recipientage
df_synth_normal$rec_tumor <- combis$rec_tumor

df_synth_expanded$recipientage <- combis$recipientage
df_synth_expanded$rec_tumorY <- as.numeric(combis$rec_tumor) - 1

synth_cox_full <- predictSurvProb(object = cox_full,
                                  newdata = df_synth_normal,
                                  times = time_su)
synth_cox_back <- predictSurvProb(object = cox_back,
                                  newdata = df_synth_normal,
                                  times = time_su)
synth_cox_lasso <- predictSurvProb(object = cox_lasso,
                                   newdata = df_synth_expanded,
                                   times = time_su)

time_val <- time_su

# predictions with rsf
model_rsf <- rfsrc(Surv(patientsurvival, death) ~ .,
                   splitrule = "logrank", nsplit = 7,
                   data = training_overall, ntree = 300,
                   seed = -12345, mtry = 23, nodesize = 50,
                   ntime = 500, sampsize = 5000,
                   forest = TRUE, importance = FALSE
)

synth_rsf_ovr <- predictSurvProb(object = model_rsf,
                                 newdata = df_synth_normal,
                                 times = time_su)

# create hypothetical patient for synthetic scaled

tumor_scaled <- c(0, 1)
age_scaled <- quantile(training_ovr_scaled$recipientage,
                       c(0.10, 0.5, 0.90))
```

```
# create new hypothetic patient
combis_scaled <- with(training_ovr_scaled, expand.grid(
  rec_tumorY = tumor_scaled, # increasing age to different
  recipientage = age_scaled
))

df_synth_scaled$rec_tumorY <- combis_scaled$rec_tumorY
df_synth_scaled$recipientage <- combis_scaled$recipientage
# create random patient survival and death status
df_synth_scaled$patientsurvival <- c(2, 4, 5, 1, 9, 4)
df_synth_scaled$death <- c(1, 0, 1, 0, 0 ,0)
rownames(df_synth_scaled) <- NULL
data_synth_long <- data_test_creator(df_synth_scaled)

df_new <- data.frame(hazard = predict_proba(fit_keras,
                                            as.matrix(data_synth_long[,
                                                  c(1:128, 131)]),
                                                  batch_size = 1000))
df_new$id <- rep(1:6, each = 12) # ids of the patients
groups_new <- split(df_new, f = df_new$id)
group_probs_new <- lapply(groups_new, function(x) {
                          x <- cumprod(1 - x$hazard)})
pred_mat_new <- do.call("rbind", group_probs_new)

# for neural network with 2 hidden layers
df_new2 <- data.frame(hazard = predict_proba(fit_keras2,
                                            as.matrix(data_synth_long[,
                                                  c(1:128, 131)]),
                                                  batch_size = 500))
df_new2$id <- rep(1:6, each = 12) # ids of the patients
groups_new2 <- split(df_new2, f = df_new2$id)
group_probs_new2 <- lapply(groups_new2, function(x) {
                           x <- cumprod(1 - x$hazard)})
pred_mat_new2 <- do.call("rbind", group_probs_new2)

############################################################
# Survival curves for recipientage and tumor
############################################################

cases <- c("Age:  41 , Tumor:  N", "Age:  41 , Tumor:  Y",
           "Age:  56 , Tumor:  N", "Age:  56 , Tumor:  Y",
           "Age:  66 , Tumor:  N", "Age:  66 , Tumor:  Y")

pat1 <- data.frame(Times = time_val, Prob1 = synth_cox_full[1, ],
                Prob2 = synth_cox_back[1, ],
                Prob3 = synth_cox_lasso[1, ],
                Prob4 = synth_rsf_ovr[1, ],
                Prob5 = pred_mat_new[1, ],
                Prob6 = pred_mat_new2[1, ],
                Patient = cases[1])
pat2 <- data.frame(Times = time_val,
```

```
                    Prob1 = synth_cox_full[2, ],
                    Prob2 = synth_cox_back[2, ],
                    Prob3 = synth_cox_lasso[2, ],
                    Prob4 = synth_rsf_ovr[2, ],
                    Prob5 = pred_mat_new[2, ],
                    Prob6 = pred_mat_new2[2, ],
                    Patient = cases[2])
pat3 <- data.frame(Times = time_val,
                    Prob1 = synth_cox_full[3, ],
                    Prob2 = synth_cox_back[3, ],
                    Prob3 = synth_cox_lasso[3, ],
                    Prob4 = synth_rsf_ovr[3, ],
                    Prob5 = pred_mat_new[3, ],
                    Prob6 = pred_mat_new2[3, ],
                    Patient = cases[3])
pat4 <- data.frame(Times = time_val,
                    Prob1 = synth_cox_full[4, ],
                    Prob2 = synth_cox_back[4, ],
                    Prob3 = synth_cox_lasso[4, ],
                    Prob4 = synth_rsf_ovr[4, ],
                    Prob5 = pred_mat_new[4, ],
                    Prob6 = pred_mat_new2[4, ],
                    Patient = cases[4])
pat5 <- data.frame(Times = time_val,
                    Prob1 = synth_cox_full[5, ],
                    Prob2 = synth_cox_back[5, ],
                    Prob3 = synth_cox_lasso[5, ],
                    Prob4 = synth_rsf_ovr[5, ],
                    Prob5 = pred_mat_new[5, ],
                    Prob6 = pred_mat_new2[5, ],
                    Patient = cases[5])
pat6 <- data.frame(Times = time_val,
                    Prob1 = synth_cox_full[6, ],
                    Prob2 = synth_cox_back[6, ],
                    Prob3 = synth_cox_lasso[6, ],
                    Prob4 = synth_rsf_ovr[6, ],
                    Prob5 = pred_mat_new[6, ],
                    Prob6 = pred_mat_new2[6, ],
                    Patient = cases[6])

df_all_pat <- rbind(pat1, pat2, pat3, pat4, pat5, pat6)

ggplot(df_all_pat, aes(x = Times)) +
  geom_line(size = 0.8, aes(y  = Prob1, col = "Cox_full")) +
  geom_line(size = 0.8, aes(y  = Prob2, col = "Cox_backward")) +
  geom_line(size = 0.8, aes(y  = Prob3, col = "Cox_LASSO")) +
  geom_line(size = 0.8, aes(y  = Prob4, col = "RSF")) +
  geom_line(size = 0.8, aes(y  = Prob5, col = "NN_1h")) +
  geom_line(size = 0.8, aes(y  = Prob6, col = "NN_2h")) +
  scale_x_continuous(breaks = seq(2, 12, 2)) +
  xlab("Time since transplantation in years") +
```

```
  ylab("Survival probability") +
  ylim(c(0, 1)) +
  theme_classic() +
  ggtitle(" ") +
  theme(plot.title = element_text(hjust = 0.5)) +
  facet_wrap(~Patient) +
  scale_colour_manual(name = " ",
                      values=c(Cox_full = "black",
                               Cox_backward = "red",
                               Cox_LASSO = "purple",
                               RSF = "blue",
                               NN_1h = "green",
                               NN_2h = "darkgreen")) +
  theme(axis.title=element_text(face="bold.italic",
                                size="12", color="brown"),
                                legend.position="right")

#################################################################
# Survival curves for rec_immuno_maint_meds and rec_postx_los
#################################################################

immuno_meds <-
factor(levels(training_overall$rec_immuno_maint_meds))
los <- quantile(training_overall$rec_postx_los,
                c(0.10, 0.5, 0.90))

# create new hypothetic patient
combis <- with(training_overall, expand.grid(
  rec_immuno_maint_meds = immuno_meds,
  rec_postx_los = los
))

df_synth2_normal$rec_postx_los <- combis$rec_postx_los
df_synth2_normal$rec_immuno_maint_meds <-
combis$rec_immuno_maint_meds

df_synth2_expanded$rec_postx_los <- combis$rec_postx_los
df_synth2_expanded$rec_immuno_maint_medsY <-
as.numeric(combis$rec_immuno_maint_meds) - 1

synth_cox_full2 <- predictSurvProb(object = cox_full,
                                   newdata = df_synth2_normal,
                                   times = time_su)
synth_cox_back2 <- predictSurvProb(object = cox_back,
                                   newdata = df_synth2_normal,
                                   times = time_su)
synth_cox_lasso2 <- predictSurvProb(object = cox_lasso,
                                    newdata = df_synth2_expanded,
                                    times = time_su)
time_val <- time_su
```

```
synth_rsf_ovr2 <- predictSurvProb(object = model_rsf,
                                  newdata = df_synth2_normal,
                                  times = time_su)

# create hypothetical patient for synthetic scaled

immuno_meds_scaled <- c(0, 1)
los_scaled <- quantile(training_ovr_scaled$rec_postx_los,
                       c(0.10, 0.5, 0.90))

# create new hypothetic patient
combis_scaled2 <- with(training_ovr_scaled, expand.grid(
  rec_immuno_maint_medsY = immuno_meds_scaled,
  rec_postx_los = los_scaled
))

df_synth2_scaled$rec_immuno_maint_medsY <-
combis_scaled2$rec_immuno_maint_medsY
df_synth2_scaled$rec_postx_los <- combis_scaled2$rec_postx_los
# create random patient survival and death status
df_synth2_scaled$patientsurvival <- c(2, 4, 5, 1, 9, 4)
df_synth2_scaled$death <- c(1, 0, 1, 0, 1 ,0)
rownames(df_synth2_scaled) <- NULL
data_synth_long <- data_test_creator(df_synth2_scaled)

df_new <- data.frame(hazard = predict_proba(fit_keras,
                                            as.matrix(data_synth_long[,
                                            c(1:128, 131)]),
                                            batch_size = 500))
df_new$id <- rep(1:6, each = 12) # ids of the patients
groups_new <- split(df_new, f = df_new$id)
group_probs_new <- lapply(groups_new, function(x) {
                            x <- cumprod(1 - x$hazard)})
pred_mat_new <- do.call("rbind", group_probs_new)

df_new2 <- data.frame(hazard = predict_proba(fit_keras2,
                                             as.matrix(data_synth_long[,
                                             c(1:128, 131)]),
                                             batch_size = 1000))
df_new2$id <- rep(1:6, each = 12) # ids of the patients
groups_new2 <- split(df_new2, f = df_new2$id)
group_probs_new2 <- lapply(groups_new2, function(x) {
                             x <- cumprod(1 - x$hazard)})
pred_mat_new2 <- do.call("rbind", group_probs_new2)

cases <- c("Los:  5 , Immuno meds:  N",
           "Los:  5 , Immuno meds:  Y",
           "Los:  10 , Immuno meds:  N",
           "Los:  10 , Immuno meds:  Y",
           "Los:  33 , Immuno meds:  N",
           "Los:  33 , Immuno meds:  Y")
```

```
synth_cox_full2 <-  cbind(1, synth_cox_full2)
synth_cox_back2 <-  cbind(1, synth_cox_back2)
synth_cox_lasso2 <-  cbind(1, synth_cox_lasso2)
synth_rsf_ovr2 <-  cbind(1, synth_rsf_ovr2)
pred_mat_new <- cbind(1, pred_mat_new)
pred_mat_new2 <- cbind(1, pred_mat_new2)


pat1 <- data.frame(Times = time_val,
                   Prob1 = synth_cox_full2[1, ],
                   Prob2 = synth_cox_back2[1, ],
                   Prob3 = synth_cox_lasso2[1, ],
                   Prob4 = synth_rsf_ovr2[1, ],
                   Prob5 = pred_mat_new[1, ],
                   Prob6 = pred_mat_new2[1, ],
                   Patient = cases[1])
pat2 <- data.frame(Times = time_val,
                   Prob1 = synth_cox_full2[2, ],
                   Prob2 = synth_cox_back2[2, ],
                   Prob3 = synth_cox_lasso2[2, ],
                   Prob4 = synth_rsf_ovr2[2, ],
                   Prob5 = pred_mat_new[2, ],
                   Prob6 = pred_mat_new2[2, ],
                   Patient = cases[2])
pat3 <- data.frame(Times = time_val,
                   Prob1 = synth_cox_full2[3, ],
                   Prob2 = synth_cox_back2[3, ],
                   Prob3 = synth_cox_lasso2[3, ],
                   Prob4 = synth_rsf_ovr2[3, ],
                   Prob5 = pred_mat_new[3, ],
                   Prob6 = pred_mat_new2[3, ],
                   Patient = cases[3])
pat4 <- data.frame(Times = time_val,
                   Prob1 = synth_cox_full2[4, ],
                   Prob2 = synth_cox_back2[4, ],
                   Prob3 = synth_cox_lasso2[4, ],
                   Prob4 = synth_rsf_ovr2[4, ],
                   Prob5 = pred_mat_new[4, ],
                   Prob6 = pred_mat_new2[4, ],
                   Patient = cases[4])
pat5 <- data.frame(Times = time_val,
                   Prob1 = synth_cox_full2[5, ],
                   Prob2 = synth_cox_back2[5, ],
                   Prob3 = synth_cox_lasso2[5, ],
                   Prob4 = synth_rsf_ovr2[5, ],
                   Prob5 = pred_mat_new[5, ],
                   Prob6 = pred_mat_new2[5, ],
                   Patient = cases[5])
pat6 <- data.frame(Times = time_val,
                   Prob1 = synth_cox_full2[6, ],
```

```
                    Prob2 = synth_cox_back2[6, ],
                    Prob3 = synth_cox_lasso2[6, ],
                    Prob4 = synth_rsf_ovr2[6, ],
                    Prob5 = pred_mat_new[6, ],
                    Prob6 = pred_mat_new2[6, ],
                    Patient = cases[6])

df_all_pat <- rbind(pat1, pat2, pat3, pat4, pat5, pat6)

ggplot(df_all_pat, aes(x = Times)) +
  geom_line(size = 0.8, aes(y  = Prob1, col = "Cox_full")) +
  geom_line(size = 0.8, aes(y  = Prob2, col = "Cox_backward")) +
  geom_line(size = 0.8, aes(y  = Prob3, col = "Cox_LASSO")) +
  geom_line(size = 0.8, aes(y  = Prob4, col = "RSF")) +
  geom_line(size = 0.8, aes(y  = Prob5, col = "NN_1h")) +
  geom_line(size = 0.8, aes(y  = Prob6, col = "NN_2h")) +
  scale_x_continuous(breaks = 0:12) +
  xlab("Time since transplantation in years") +
  ylab("Survival probability") +
  ylim(c(0, 1)) +
  theme_classic() +
  ggtitle(" ") +
  theme(plot.title = element_text(hjust = 0.5)) +
  facet_wrap(~Patient) +
  scale_colour_manual(name = " ",
                      values=c(Cox_full = "black",
                               Cox_backward = "red",
                               Cox_LASSO = "purple",
                               RSF = "blue",
                               NN_1h = "green",
                               NN_2h = "darkgreen")) +
  theme(axis.title=element_text(face="bold.italic",
                                size="12", color="brown"),
        legend.position="right")
```

Bibliography

M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: A system for large-scale machine learning, 2016. URL https://ai.google/research/pubs/pub45381.

b. Andy Liaw, M. Wiener, and M. Andy Liaw. Package 'randomForest' Title Breiman and Cutler's Random Forests for Classification and Regression. 2018. doi: 10.1023/A:1010933404324⟩. URL https://www.stat.berkeley.edu/~breiman/RandomForests/.

C. F. Barker and J. F. Markmann. Historical overview of transplantation. *Cold Spring Harbor Perspectives in Medicine*, 2013. ISSN 21571422. doi: 10.1101/cshperspect.a014977.

E. Biganzoli, P. Boracchi, L. Mariani, and E. Marubini. FEED FORWARD NEURAL NETWORKS FOR THE ANALYSIS OF CENSORED SURVIVAL DATA: A PARTIAL LOGISTIC REGRESSION APPROACH. Technical report, 1998.

C. M. Bishop. Neural Networks: A Pattern Recognition Perspective. Technical report, 1996. URL http://www.ncrg.aston.ac.uk/.

L. Breiman. Bagging Predictors. Technical report, 1994. URL https://www.stat.berkeley.edu/~breiman/bagging.pdf.

L. Breiman. Heuristics of instability and stabilization in model selection. *Annals of Statistics*, 1996a. ISSN 00905364. doi: 10.1214/aos/1032181158.

L. Breiman. OUT-OF-BAG ESTIMATION. Technical report, 1996b. URL https://www.stat.berkeley.edu/~breiman/OOBestimation.pdf.

Breiman Leo. Random forests. 2001.

S. Buuren. *Flexible Imputation of Missing Data*, volume 20125245 of *Chapman & Hall/CRC Interdisciplinary Statistics Series*. Chapman and Hall/CRC, 3 2012. ISBN 978-1-4398-6824-9. doi: 10.1201/b11826. URL https://www.taylorfrancis.com/books/9781439868256.

J. H. Chen and S. M. Asch. Machine Learning and Prediction in Medicine Beyond the Peak of Inflated Expectations. *New England Journal of Medicine*, 2017. ISSN 0028-4793. doi: 10.1056/NEJMp1702071.

T. Ching, X. Zhu, and L. X. Garmire. Cox-nnet: An artificial neural network method for prognosis prediction of high-throughput omics data. 2018. doi: 10.1371/journal.pcbi.1006076. URL https://doi.org/10.1371/journal.pcbi.1006076.g001.

F. Chollet et al. Keras, 2015. URL https://keras.io.

D. Collett. *Modelling survival data in medical research*. 2014. ISBN 9781439856789.

D. R. Cox. Regression Models and Life-Tables. 1972. ISBN 00359246. doi: 10.1007/978-1-4612-4380-9{_}37.

S. Editors, K. Dietz, M. Gail, K. Krickeberg, J. Samet, A. T. Springer, N. York, B. Heidelberg, H. Kong, L. Milan, and P. Tokyo. Statistics for Biology and Health.

B. Efron and R. Tibshirani. Improvements on Cross-Validation: The .632+ Bootstrap Method. Technical Report 438, 1997. URL https://pdfs.semanticscholar.org/8e30/f02d667163ff52223efd57c0b48a0a9a7873.pdf.

B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. LEAST ANGLE REGRESSION. Technical Report 2, 2004.

S. Fotso. Deep Neural Networks for Survival Analysis Based on a Multi-Task Framework. 2018.

Frank E Harrell Jr. rms: Regression Modeling Strategies, 2018. URL https://CRAN.R-project.org/package=rms.

J. Friedman, T. Hastie, and R. Tibshirani. Regularization Paths for Generalized Linear Models via Coordinate Descent. *Journal of Statistical Software*, 2010. ISSN 1548-7660. doi: 10.18637/jss.v033.i01.

J. Friedman, T. Hastie, R. Tibshirani, N. Simon, B. Narasimhan, J. Qian, and . Maintainer. Package 'glmnet' Type Package Title Lasso and Elastic-Net Regularized Generalized Linear Models. Technical report, 2018.

W. J. Fu, R. J. Carroll, and S. Wang. Estimating misclassification error with small samples via bootstrap cross-validation. *Bioinformatics*, 21(9): 1979–1986, 5 2005. ISSN 1367-4803. doi: 10.1093/bioinformatics/bti294. URL http://www.ncbi.nlm.nih.gov/pubmed/15691862https://academic.oup.com/bioinformatics/article-lookup/doi/10.1093/bioinformatics/bti294.

G. Garson. Interpreting Neural Network Connection Weights. *AI Expert, 6*, pages 47–51, 1991.

T. A. Gerds and M. A. van de Wiel. Confidence scores for prediction models. *Biometrical Journal*, 53(2):259–274, 3 2011. ISSN 03233847. doi: 10.1002/bimj.201000157. URL http://doi.wiley.com/10.1002/bimj.201000157.

X. Glorot, A. Bordes, and Y. Bengio. Deep Sparse Rectifier Neural Networks. Technical report, 2011. URL http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf.

A. Goh. Back-propagation neural networks for modeling complex systems. *Artificial Intelligence in Engineering*, 9(3):143–151, 1 1995. ISSN 0954-1810. doi: 10.1016/0954-1810(94)00011-S. URL https://www.sciencedirect.com/science/article/pii/095418109400011S.

E. Graf, C. Schmoor, W. Sauerbrei, and M. Schumacher. Assessment and comparison of prognostic classification schemes for survival data. *Statistics in medicine*, 18 (17-18):2529–45, 1996. URL http://www.ncbi.nlm.nih.gov/pubmed/10474158.

F. E. Harrell, K. L. Lee, and D. B. Mark. Multivariable Prognostic Models: Issues in developing models, evaluating assumptions and adequacy and measuring and reducing errors. *Statistics in Medicine*, 15(4):361–387, 2 1996. doi: 10.1002/(SICI)1097-0258(19960229)15:4⟨361::AID-SIM168⟩3.0.CO;2-4. URL http://www.ncbi.nlm.nih.gov/pubmed/8668867http://www.ncbi.nlm.nih.gov/pubmed/8668867.

Harrell Frank. Road Map for Choosing Between Statistical Modeling and Machine Learning. URL http://www.fharrell.com/post/stat-ml/.

T. Hastie, R. Tibshirani, and J. Friedman. Springer Series in Statistics The Elements of Statistical Learning Data Mining, Inference, and Prediction. Technical report, 2001.

T. Hothorn and B. Lausen. On the exact distribution of maximally selected rank statistics. Technical report, 2003. URL www.elsevier.com/locate/csda.

Isaac Changhau. Activation Functions in Neural Networks. URL https://isaacchanghau.github.io/post/activation_functions/.

H. Ishwaran. Variable importance in binary regression trees and forests. *Electronic Journal of Statistics*, 2007. ISSN 19357524. doi: 10.1214/07-EJS039.

H. Ishwaran and U. B. Kogalur. Random Survival Forests for R. Technical Report 2, 2007.

H. Ishwaran and M. Lu. Standard errors and confidence intervals for variable importance in random forest regression, classification, and survival. *Statistics in Medicine*, 6 2018. ISSN 02776715. doi: 10.1002/sim.7803. URL http://www.ncbi.nlm.nih.gov/pubmed/29869423http://doi.wiley.com/10.1002/sim.7803.

H. Ishwaran, U. B. Kogalur, E. H. Blackstone, and M. S. Lauer. Random survival forests. *Annals of Applied Statistics*, 2008a. ISSN 19326157. doi: 10.1214/08-AOAS169.

H. Ishwaran, U. B. Kogalur, E. H. Blackstone, and M. S. Lauer. Random survival forests. *Annals of Applied Statistics*, 2008b. ISSN 19326157. doi: 10.1214/08-AOAS169.

H. Ishwaran, U. B. Kogalur, E. Z. Gorodeski, A. J. Minn, and M. S. Lauer. High-dimensional variable selection for survival data. *Journal of the American Statistical Association*, 2010. ISSN 01621459. doi: 10.1198/jasa.2009.tm08622.

H. Ishwaran, M. Udaya, and B. Kogalur. Title Random Forests for Survival, Regression, and Classification (RF-SRC). 2018. URL https://github.com/kogalur/randomForestSRC/issues/newhttp://web.ccs.miami.edu/~hishwaranhttp://www.kogalur.comhttps://github.com/kogalur/randomForestSRC.

A. James Honaker, G. King, M. Blackwell, and M. Matthew Blackwell. Title A Program for Missing Data. 2018.

S. Janitza. Silke Janitza On the overestimation of random forest's out-of-bag error On the overestimation of random forest's out-of-bag error. Technical report, 2017. URL http://www.stat.uni-muenchen.de.

W. Jarmulski, A. Wieczorkowska, M. Trzaska, M. Ciszek, L. Paczek, and L. Paczek. MACHINE-LEARNING MODELS FOR PREDICTING PATIENT SURVIVAL AFTER LIVER TRANSPLANTATION. *Computer Science*, 19(2):223, 5 2018. ISSN 1508-2806. doi: 10.7494/csci.2018.19.2.2746. URL http://journals.agh.edu.pl/csci/article/view/2746.

R. T. Jerome Friedman, Trevor Hastie. Regularization Paths for Generalized Linear Models via Coordinate Descent, 2010. URL http://www.jstatsoft.org/v33/i01/.

JJ Allaire and François Chollet. keras: R Interface to 'Keras'. 2018. URL https://CRAN.R-project.org/package=keras.

P. S. Kamath, R. H. Wiesner, M. Malinchoc, W. Kremers, T. M. Therneau, C. L. Kosberg, G. Damico, E. R. Dickson, and W. R. Kim. A model to predict survival in patients with end-stage liver disease, 2001. ISSN 02709139.

E. L. Kaplan and P. Meier. Nonparametric Estimation from Incomplete Observations NONPARAMETRIC ESTIMATION FROM INCOMPLETE OBSERVATIONS*. *Source Journal of the American Statistical Association*, 5313481(282):457–481, 1958. URL http://www.jstor.org/stable/2281868http://www.jstor.org/page/info/about/policies/terms.jsphttp://www.jstor.org.

U. Kartoun, K. E. Corey, T. G. Simon, H. Zheng, R. Aggarwal, K. Ng, and S. Y. Shaw. The MELD-Plus: A generalizable prediction risk score in cirrhosis. *PLoS ONE*, 2017. ISSN 19326203. doi: 10.1371/journal.pone.0186301.

Kogalur U and Ishwaran H. Title Random Forests for Survival, Regression, and Classification (RF-SRC). Technical report, 2018.

L. L. Lapuerta P, Azen SP. Use of neural networks in predicting the risk of coronary-artery disease. *Comp Biomed Res*, page 3852., 1995.

L. Lau, Y. Kankanige, B. Rubinstein, R. Jones, C. Christophi, V. Muralidharan, and J. Bailey. Machine-Learning Algorithms Predict Graft Failure After Liver Transplantation. *Transplantation*, 101(4):e125–e132, 4 2017. ISSN 0041-1337. doi: 10.1097/TP.0000000000001600. URL http://Insights.ovid.com/crossref?an=00007890-201704000-00025.

J. F. Lawless and K. Singhal. Efficient Screening of Nonnormal Regression Models. *Biometrics*, 34(2):318, 6 1978. ISSN 0006341X. doi: 10.2307/2530022. URL https://www.jstor.org/stable/2530022?origin=crossref.

W.-Y. Loh and Y.-S. Shih. SPLIT SELECTION METHODS FOR CLASSIFICATION TREES. Technical report, 1997.

Markus Schmitt. Artificial Intelligence in Medicine. URL https://www.datarevenue.com/en/usecases/artificial-intelligence-in-medicine?gclid=EAIaIQobChMIwY7w15-_3QIVirTtCh1Engm-EAAYASAAEgIWi_D_BwE.

R. M. Merion, D. E. Schaubel, D. M. Dykstra, R. B. Freeman, F. K. Port, and R. A. Wolfe. The survival benefit of liver transplantation. *American Journal of Transplantation*, 2005. ISSN 16006135. doi: 10.1111/j.1600-6143.2004.00703.x.

M. Minsky and S. Papert. *Perceptrons; an introduction to computational geometry*. MIT Press, 1969. ISBN 9780262130431.

U. B. Mogensen, H. Ishwaran, and T. A. Gerds. Evaluating Random Forests for Survival Analysis Using Prediction Error Curves. Technical report, 2012. URL http://www.jstatsoft.org/.

M. Y. Park and T. Hastie. L 1-regularization path algorithm for generalized linear models. Technical report, 2007.

Python Core Team. Python: A dynamic, open source programming language. Python Software Foundation., 2015. URL https://www.python.org/.

Q-A for transplant candidates. Questions and Answers for Transplant Candidates about the Liver Allocation System.

R Core Team. R: A Language and Environment for Statistical Computing, 2014. URL http://www.R-project.org/.

B. D. Ripley and R. M. Ripley. Artificial Neural Networks: Prospects for Medicine Neural Networks as Statistical Methods in Survival Analysis. Technical report.

M. R. Segal. Regression Trees for Censored Data Author(s). Technical Report 1, 1988. URL http://www.biecek.pl/statystykaMedyczna/2531894Tree.pdf.

N. Srivastava, G. Hinton, A. Krizhevsky, and R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Technical report, 2014.

Stef van Buuren and Karin Groothuis-Oudshoorn. Multivariate Imputation by Chained Equations in R. *Journal of Statistical Software*, 45:1–67, 2011. URL https://www.jstatsoft.org/v45/i03/.

S. V. Stehman. Selecting and interpreting measures of thematic classification accuracy. *Remote Sensing of Environment*, 1997. ISSN 00344257. doi: 10.1016/S0034-4257(97)00083-7.

D. J. Stekhoven and P. Bühlmann. Missforest-Non-parametric missing value imputation for mixed-type data. *Bioinformatics*, 2012. ISSN 13674803. doi: 10.1093/bioinformatics/btr597.

C. Strobl, J. Malley, and G. Tutz. An introduction to recursive partitioning: rationale, application, and characteristics of classification and regression trees, bagging, and random forests. *Psychological methods*, 14(4):323–48, 12 2009. ISSN 1939-1463. doi: 10.1037/a0016973. URL http://www.ncbi.nlm.nih.gov/pubmed/19968396http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC2927982.

J. M. G. Taylor. Random Survival Forests. Technical report, 2011. URL https://core.ac.uk/download/pdf/82632679.pdf.

A. A. Thomas Gerds. Package 'pec' Title Prediction Error Curves for Risk Prediction Models in Survival Analysis. Technical report, 2018. URL https://cran.r-project.org/web/packages/pec/pec.pdf.

R. Tibshirani. Regression Shrinkage and Selection via the Lasso. Technical Report 1, 1996.

R. Tibshirani. THE LASSO METHOD FOR VARIABLE SELECTION IN THE COX MODEL. *STATISTICS IN MEDICINE*, 16:385–395, 1997.

S. Van Buuren, H. C. Boshuizen, and D. L. Knook. Multiple imputation of missing blood pressure covariates in survival analysis. *Statistics in Medicine*, 1999. ISSN 02776715. doi: 10.1002/(SICI)1097-0258(19990330)18:6⟨681::AID-SIM71⟩3.0.CO;2-R.

M. van Gerven and S. Bohte. Editorial: Artificial Neural Networks as Models of Neural Information Processing. *Frontiers in Computational Neuroscience*, 11, 12 2017. ISSN 1662-5188. doi: 10.3389/fncom.2017.00114. URL http://journal.frontiersin.org/article/10.3389/fncom.2017.00114/full.

H. C. Van Houwelingen and H. Putter. Dynamic Prediction in Clinical Survival Analysis. 2011.

J. C. Van Houwelingen and S. Le Cessie. Predictive value of statistical models. *Statistics in Medicine*, 1990. ISSN 10970258. doi: 10.1002/sim.4780091109.

P. J. M. Verweij and H. C. Van Houwelingen. Cross-validation in survival analysis. *Statistics in Medicine*, 12(24):2305–2314, 12 1993. ISSN 02776715. doi: 10.1002/sim.4780122407. URL http://doi.wiley.com/10.1002/sim.4780122407.

W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer, fourth edition, 2002. URL http://www.stats.ox.ac.uk/pub/MASS4.

Werbos. Backpropagation: past and future. In *IEEE International Conference on Neural Networks*, 1988. ISBN 0-7803-0999-5. doi: 10.1109/ICNN.1988.23866.

I. R. White and P. Royston. Imputing missing covariate values for the Cox model. *Statistics in Medicine*, 2009. ISSN 02776715. doi: 10.1002/sim.3618.

R. Wiesner, E. Edwards, R. Freeman, A. Harper, R. Kim, P. Kamath, W. Kremers, J. Lake, T. Howard, R. M. Merion, R. A. Wolfe, R. Krom, P. M. Colombani, P. C. Cottingham, S. P. Dunn, J. J. Fung, D. W. Hanto, S. V. McDiarmid, J. M. Rabkin, L. W. Teperman, J. G. Turcotte, and L. R. Wegman. Model for end-stage liver disease (MELD) and allocation of donor livers. *Gastroenterology*, 2003. ISSN 00165085. doi: 10.1053/gast.2003.50016.

R. H. Wiesner, S. V. McDiarmid, P. S. Kamath, E. B. Edwards, M. Malinchoc, W. K. Kremers, R. A. Krom, and W. R. Kim. Meld and Peld: Application of survival models to liver allocation, 2001. ISSN 15276465.

M. N. Wright, T. Dankowski, and A. Ziegler. Unbiased split variable selection for random survival forests using maximally selected rank statistics. *Statistics in Medicine*, 2017. ISSN 10970258. doi: 10.1002/sim.7212.

List of Abbreviations

AIC Akaike Information Criterion

Boot632CV Bootstrap 632 Cross-validation

BootCV Bootstrap Cross-validation

CART Classification and Regression Trees

CHE Cumulative Hazard Estimate

CVPL Cross Validated Partial Likelihood

DCD Donor after Cardiac Death

EM Expectation Maximization

FFANN Feed Forward Artificial Neural Networks

FFS Failure-Free Survival

HCV Hepatitis C Virus

IBS Integrated Brier Score

IPCW Inverse probability of censoring weighting

LARS Least Angle Regression

LASSO Least Absolute Shrinkage and Selection Operator

MAR Missing At Random

MCAR Missing Completely At Random

MELD Model for End-Stage Liver Disease

MICE Multiple Imputation by Chained Equations

ML Machine Learning

NMAR Not Missing At Random

NN Artificial Neural Networks

NRMSE Normalized Root Mean Squared Error

OOB Out-of-bag

OPO Organ Procurement Organizations

OPTN Organ Procurement and Transplantation Network

OS Overall Survival

PH Proportional Hazards

PLANN Partial Logistic Artificial Neural Network

randomForestSRC Random Forests for Survival-Regression-Classification

ReLU Rectified Linear Unit

RF Random Forests

ROC Area under the receiver operating characteristic curve

RSF Random Survival Forest

SAF Standard Analysis Files

SGD Stochastic Gradient Descent

SM Statistical method

SRTR Scientific Registry of Transplant Recipients

tanh Hyperbolic Tangent

UNOS United Network for Organ Sharing

VIMP Variable Importance