



CZECH TECHNICAL UNIVERSITY IN PRAGUE  
Faculty of Nuclear Sciences and Physical Engineering



# **Estimating Kidney Transplantation Donor-Recipient Compatibility Using Machine Learning**

## **Odhad kompatibility dárce a příjemce pro transplantaci ledvin pomocí strojového učení**

Bachelor's Degree Project

Author: **Matěj Klouček**  
Supervisor: **Ing. Tomáš Kouřim**  
Consultant: **Ing. Pavel Strachota , Ph.D.**  
Academic year: 2022/2023

## ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student:	Matěj Klouček
Studijní program:	Matematické inženýrství
Studijní specializace:	Matematická informatika
Název práce (česky):	Odhad kompatibility dárce a příjemce pro transplantaci ledvin pomocí strojového učení
Název práce (anglicky):	Estimating kidney transplantation donor-recipient compatibility using machine learning

### Pokyny pro vypracování:

- 1) Seznamte se s principy strojového učení, jeho různými metodami a softwarem pro jeho implementaci.
- 2) Shromážděte informace potřebné pro pochopení rozhodujících faktorů při hledání kompatibilních párů dárce a příjemce. Prostudujte roli 'hlavního histokompatibilního komplexu (HLA)' při transplantaci ledvin.
- 3) Shromážděte, vyčistěte a analyzujte data z uskutečněných transplantací ledvin za účelem vytvoření souboru dat pro trénování modelu strojového učení.
- 4) Navrhněte algoritmus predikující kompatibilitu dárce a příjemce orgánu a ověřte jeho fungování na reálných i vygenerovaných datech.
- 5) Porovnejte přesnost navrhovaného algoritmu pro data pocházející z různých vzorků populace a případně navrhněte jeho úpravy.



Doporučená literatura:

- 1) R. O. Duda, P. E. Hart, D. G. Stork, Pattern Classification. John Wiley & Sons, 2000.
- 2) F. Chollet, Deep Learning with Python. Manning Publications Co., 2018.
- 3) M. Mohri, A. Rostamizadeh, A. Talwalkar, Foundations of Machine Learning. MIT Press, 2018.
- 4) J. J. Kim, S. V. Fuggle, S. D. Marks, Does HLA matching matter in the modern era of renal transplantation? Pediatric Nephrology 36, 2021, 31–40.
- 5) M. Wohlfahrtová, O. Viklický, R. Lischke, a kolektiv, Transplantace orgánů v klinické praxi. Grada Publishing a.s., Praha, 2021.

Jméno a pracoviště vedoucího bakalářské práce:

Ing. Tomáš Kouřim  
Mild Blue, s.r.o., Plzeňská 27, Praha 5

Jméno a pracoviště konzultanta:



Ing. Pavel Strachota , Ph.D.  
Katedra matematiky FJFI ČVUT,, Trojanova 13, 120 00 Praha 2

Datum zadání bakalářské práce: 31.10.2022

Datum odevzdání bakalářské práce: 2.8.2023

Doba platnosti zadání je dva roky od data zadání.

V Praze dne 31. října 2022

  
.....  
garant oboru  
  
.....  
vedoucí katedry



  
.....  
děkan

*Acknowledgment:*

I would like to thank ..... for (his/her expert guidance) and expres's my gratitude to ..... for (his/her language assistance).

*Author's declaration:*

I declare that this Bachelor's Degree Project is entirely my own work and I have listed all the used sources in the bibliography.

Prague, August 2, 2023

Matěj Klouček

## Odhad kompatibility dárce a příjemce pro transplantaci ledvin pomocí strojového učení

*Obor: Matematické inženýrství*

*Druh práce:* Bakalářská práce

*Konzultant:* Ing. Pavel Strachota , Ph.D., Katedra matematiky FJFI ČVUT

**Klíčová slova:** analýza přežívání, random survival forests, strojové učení, transplantace ledvin

## Estimating kidney transplantation donor-recipient compatibility using machine learning

[illegible]

**Key words:** Machine learning, Random survival forests, Renal transplantation, Survival analysis

# Contents

<b>Introduction</b>	<b>8</b>
<b>1 Machine Learning</b>	<b>9</b>
1.1 General Overview of Machine Learning . . . . .	9
1.1.1 Classification of Machine Learning Models . . . . .	9
1.2 Building a Machine Learning Model . . . . .	10
1.2.1 Data Preprocessing . . . . .	10
1.2.2 Evaluating Performance . . . . .	11
1.2.3 Feature Selection . . . . .	12
1.2.4 Hyperparameter tuning . . . . .	12
1.3 Decision Trees and Random Forests . . . . .	12
1.3.1 Decision Trees . . . . .	12
1.3.2 Ensemble Learning and Random Forests . . . . .	15
<b>2 Renal Transplantation</b>	<b>17</b>
2.1 Chronic Kidney Disease . . . . .	17
2.2 Pre-Transplant Procedures . . . . .	17
2.2.1 HLA Typing . . . . .	17
2.3 Goals . . . . .	18
<b>3 Survival Analysis</b>	<b>20</b>
3.1 Introduction to Survival Analysis . . . . .	20
3.1.1 Data Censoring . . . . .	20
3.1.2 Survival and Hazard Functions . . . . .	20
3.1.3 Hazard Ratio . . . . .	21
3.2 Survival Analysis Methods . . . . .	21
3.2.1 Kaplan-Meier Estimator . . . . .	21
3.2.2 Log-rank Test . . . . .	22
3.2.3 Cox Regression . . . . .	22
<b>4 Random Survival Forests</b>	<b>26</b>
4.1 Building a Random Survival Forests model . . . . .	26
4.2 Hyperparameters . . . . .	27
4.3 Harell's Concordance Index . . . . .	27
4.3.1 Calculating Harell's Concordance Index . . . . .	28
4.4 Permutation Variable Importance . . . . .	29
4.5 Other Methods . . . . .	29

<b>5</b>	<b>Data and Software Architecture</b>	<b>31</b>
5.1	Data Acquisition . . . . .	31
5.2	Software Architecture . . . . .	31
5.2.1	Python . . . . .	31
5.2.2	Jupyter Notebook . . . . .	32
5.2.3	scikit-learn and scikit-survival . . . . .	32
5.2.4	Other libraries . . . . .	32
5.2.5	Cluster Computing . . . . .	33
<b>6</b>	<b>Model Training</b>	<b>34</b>
6.1	Data Preprocessing . . . . .	34
6.1.1	Separating Living and Deceased Donors . . . . .	34
6.1.2	Reducing Number of Features . . . . .	35
6.1.3	Ensuring Correct Formatting . . . . .	36
6.1.4	Imputing Missing Values . . . . .	36
6.1.5	Feature Encoding . . . . .	36
6.2	Feature Selection . . . . .	37
6.3	Hyperparameter Tuning . . . . .	38
6.4	Example Usage . . . . .	38
<b>7</b>	<b>Model evaluation</b>	<b>40</b>
7.1	General evaluation . . . . .	40
7.2	Comparison to conventional models . . . . .	40
7.3	Performance on different population samples . . . . .	40
7.4	Usage for Computing Compatibility . . . . .	40
7.5	Possibilities for Future Research . . . . .	41
	<b>Conclusion</b>	<b>42</b>

# Introduction

Kidney transplantation is the single best treatment for patients with end-stage kidney disease as it has demonstrated the best quality of life improvement and the best survival rate amongst other possible treatments [2, 3]. However, demand for allografts greatly exceeds supply [1, 3] and finding a compatible donor-recipient pair is often a highly time-consuming task, which results in patients spending a prolonged time on the waiting list for potential transplantation. While on the waiting list, patients have to undergo dialysis multiple times a week for several hours, which supplements the functions of their failing kidneys, which greatly reduces the quality of the patient's life and puts an additional burden on the healthcare system. It is therefore crucial to minimize the time patients spend on the waiting list by optimizing the process of finding suitable donor-recipient pairs.

Even more important than the swiftness of the donor-recipient matchmaking, is the quality of the given match, as that plays the most crucial role in determining the patient's prospects. Currently, only a handful of factors are taken into consideration when looking for suitable donor-recipient pair, such as whether they are immunologically compatible, their medical histories and their blood type. These methods have been proven successful at minimizing the risk of an acute transplant rejection, however there may be other factors that influence the graft's long term performance that are currently unaccounted for. Therefore, in order to reduce the flow of people returning to the waiting list and to prevent premature deaths because of failed grafts, it is necessary to uncover these patterns and take them into account when computing compatibility between potential donors and recipients. This is where machine learning can play a crucial role, by analyzing large amounts of data and identifying patterns that can help find compatible pairs or conversely detect incompatible pairings that would otherwise undergo transplantation.

The research will involve analyzing data on past transplantations from the US-based *United Network for Organ Sharing (UNOS)* dataset, including information on both living and deceased donors, recipients and their post-transplant outcome. This data will be used to train and test machine learning models that can predict the patient's chances of surviving over time, given that they receive a graft from a particular donor. These can then be compared in order to decide which donor-recipient pairing are the most compatible. In particular, this paper will be focused on the Random Survival Forests model which has demonstrated a promising performance in related scientific papers [5]. Additionally, the research will also examine the performance of the developed models on different population samples within the UNOS dataset.

The results of this thesis will hopefully provide valuable insights into application of machine learning in the field of transplantation medicine. If the models developed in this research are found to be effective in predicting compatibility, it could lead to a better and more efficient matching of donors and recipients, resulting in risk reduction for the transplant patients as complications such as graft rejection would be less probable. Furthermore, this research will also provide a valuable contribution to the broader field of medical research, by demonstrating the potential of machine learning in improving the success rate of medical treatments and reducing the burden on the healthcare system.



# Chapter 1

## Machine Learning

### 1.1 General Overview of Machine Learning

Machine learning is a rapidly growing field within the larger discipline of artificial intelligence, which is concerned with the development of algorithms and statistical models that enable computers to autonomously improve their performance on a given task. The key idea behind machine learning is to develop algorithms that can identify patterns and relationships in large amounts of data, and then use these algorithms to make predictions or decisions based on newly introduced data.

In general, machine learning is a great tool when solving problems that would conventionally require an insurmountable amount of programming or when working with large amounts of data from which one could not easily extract information or identify meaningful patterns.

#### 1.1.1 Classification of Machine Learning Models

Machine learning models can be classified based on several parameters including the type of data used in training the models and the way they handle new data.

##### Classification by Data Type

*Supervised Learning* is a type of Machine Learning where models require a labeled dataset for their learning, which means that along with the data, the desired output needs to be provided to the computer as well. Supervised learning can be further subdivided into Regression and Classification, based on whether the desired output is a continuous numerical value or a discrete one respectively. The examples of Supervised Learning models include: *Linear* and *Logistic Regression*, *Random Forests*, *Decision Trees*, *Supporting Vector Machines* and *Neural Networks*.

The opposite of Supervised Learning is *Unsupervised Learning*, where models are trained on an unlabeled dataset with the aim of finding patterns in the given data or grouping data with similar characteristics. An example of Unsupervised Learning is *Clustering* which is used to find groups with shared characteristics, *Association* which is used to find relation between the input variables (also called predictors) in a given dataset, or *Dimensional reduction* which is used to simplify data in order to more easily extract information from it.

A combination of these, called *Semi-supervised Learning*, uses a combination of both labeled and unlabeled data and first uses an Unsupervised Machine Learning model to first cluster the data in order to classify the unlabeled data and then uses a Supervised machine Learning model on this newly labeled data. This is advantageous to using a simple Supervised Machine Learning model as it allows working

with larger amounts of data that would otherwise be unusable because of the lack of labeling, thus potentially resulting in higher accuracy of the model.

*Reinforcement Learning* is a specialized case of Semi-supervised learning, where the model is trained using feedback from the environment and is often used in cases where no labeled data exists or when the labeled dataset does not provide the best course of action. The learning system (called *agent* in this case) learns by performing actions from which it receives either rewards or penalties from the environment and based on these has to develop a strategy to maximize rewards (called *policy*) [6, 7].

### Instance-based vs. Model-based

*Instance-based* learning algorithms work by comparing the similarity of new input data to the training data, an example of this is the *k-Nearest Neighbors* algorithm, which finds *k* examples from the training data that have the most similar features to the given input and outputs either the most frequent or the average label value in this cohort.

On the other hand *Model-based learning* algorithms develop a mathematical function whose parameters are learned from the training data and predictions about new data are then accomplished by providing the newly introduced data as an input to the function. Model-based learning algorithms are for example *Linear Regression* and *Neural Networks*.

## 1.2 Building a Machine Learning Model

The process of building a machine learning model generally consists of the following steps:

### 1.2.1 Data Preprocessing

Before a machine learning model can even begin to be trained, it is necessary that the training data is formatted correctly. The process of transforming the dataset into a correct format is called *preprocessing* and is an integral part of the process of building almost every machine learning model. Preprocessing includes, but is not limited to, the following techniques.

#### Imputing

One of the most common challenges tackled during preprocessing is missing values. Majority of machine learning models are not capable of handling entries with missing features so unless there is another model at disposal that can handle missing values, the missing values need to be handled in one of the following ways.

The easiest option is to drop columns or rows with missing values from the dataset. This is fast and reliable, however it is not optimal as it can lead to a potential loss of important information that the model could learn from.

A preferable alternative is replacing the missing values based on some given rule. This process is called *imputing* and can be performed in many different ways. One of the most popular imputers is the `SimpleImputer()` provided by the scikit-learn library described in 5.2.3. For numerical features, it replaces missing values with either the mean or the median of the values in the column that the missing value is located in. In the case of categorical features, it replaces missing values with the most common category in the particular column. Also in the case of both categorical and numerical values, it can simply replace the missing values with a given constant value.

## Encoding Categorical Features

Another challenge tackled during preprocessing arises from the fact that most machine learning models only allow numerical features as input. As a result, categorical values need to be transformed into numerical features in a process called *encoding*. One of the most simple yet most efficient methods of encoding is one-hot encoding, which transforms a categorical column with unique values  $(x_1, \dots, x_n)$  into  $n$  columns. For each row, the value of the  $i$ -th newly created column is 1  $\iff$  the original categorical column had a value  $x_i$ . The other newly created columns are assigned a value of 0.

organ		organ_L_Kidney	organ_Pancreas	organ_R_Kidney
R_Kidney		0	0	1
L_Kidney	$\longrightarrow$	1	0	0
Pancreas		0	1	0
R_Kidney		0	0	1
Pancreas		0	1	0

## Splitting Data into Training, Validation and Test Sets

The goal of machine learning is to create models that are able to make predictions when faced with new data that the model hasn't seen during the training process. To achieve this, it is necessary to split the data into 3 parts: *Training set*, *Validation set* and *Test set*. If a model is trained using all available data, it may perform well on said data, but may be unable to generalize for new instances thus making it useless in practice, which is why it is important to keep some of the data aside for validation and testing. The datasets used for validation and testing are also referred to as the *hold-out* sets [6].

It is desirable to keep the majority of the data for training with the usual distribution being 70% for training and 15% for training and validation each, though with larger datasets, it is possible to allocate even higher percentage of the data to training.

Once the training of the model has been done using the training set, the performance of the model is tested using the test set, i.e. data that it hasn't seen before. A good metric for measuring the quality of the model is the *generalization error*, given by the error rate of the model on new cases. This however, is not suitable for models working with censored data, whose performances is usually evaluated using the C-index described in 4.3.

Better performance of the model can be achieved using a validation set, which gives the option to select the best values for the model's *hyperparameters* (specifics of a given machine learning model that are set before the training process begins). This is done by training the model multiple times on the training set with different hyperparameters and then comparing their performance on the validation set. Once the best model has been selected, it is then trained using both the training and the validation set and its performance is then measured using the test set [7].

### 1.2.2 Evaluating Performance

If the model performs well on the training set (e.g. it's generalization error is low), but poorly on the test set, it means that the model has learned unnecessary details (also called *noise*) from the training set which then hampers its ability to generalize for new instances. This phenomena is called *overfitting* and is a common problem that arises during the process of developing a machine learning model. Overfitting usually caused either by the data being too noisy (errors in the data, many outliers), the dataset being too small or by having too many irrelevant features. The above mentioned problems can be overcome by gathering more data, removing outliers and locating errors in the data or by simplifying the model

by choosing fewer features. Another way of solving the issue of overfitting is constraining how much the model can change the values of its parameters, thus making the model simpler and less prone to overfitting. This is called *regularization* [7].

Machine learning models can also face an opposite problem, i.e. not being able learn the underlying patterns in the data and thus being inaccurate on both the test and training data. This is called *underfitting* and can be solved by either using a more appropriate (and more complex) model for the given task, selecting better features to train the model on or reducing any constraints that might have been set to simplify the model in order to prevent overfitting [7].

### 1.2.3 Feature Selection

As mentioned in 1.2.2, one of the most common causes for overfitting is having too many features included in the model. This can be solved by only selecting the features that contribute the most to the model's performance. This can not only prevent overfitting and thus increase the model's accuracy scores [17] but also greatly increase explainability of the model [6] and speed up the training process.

In the case of classification decision trees and random forests described in 1.3, feature importance is most commonly computed using the *mean decrease in impurity* (MDI) also known *Gini importance*. For each feature used as an input for a given tree, the MDI is calculated as the mean improvement in the accuracy of the given tree when splitting on this feature. In the case of random forests, this importance is then averaged over all trees. This is a very efficient way of computing feature importance as it can be calculated during the training process. However this method can be biased towards overestimating importance of variables with bigger scaled values .

A more reliable, though more computationally demanding, alternative is the *permutation importance* described in 4.4.

### 1.2.4 Hyperparameter tuning

An integral part of building a well-performing machine learning model is selecting the best hyperparameters for a given algorithm. These parameters are not optimized by the learning algorithm itself and therefore require experimentation to find the best combination of values for them [6].

A common way of finding the best combination of hyperparameters is *grid search*, which works by creating a set of values for each hyperparameter and then evaluating the model's performance for every combination of these hyperparameters. For numerical parameters, a logarithmic scale is typically used to create a set of possible values (e.g. [0.01, 0.1, 1, 10, 100, 1000]), while for categorical hyperparameters, it is possible to simply iterate over all possible values. When the best combination of hyperparameters is found, it can be beneficial to experiment with values closer to the found optimal combination [6].

## 1.3 Decision Trees and Random Forests

### 1.3.1 Decision Trees

Decision Trees are a supervised, model-based machine learning method used for both regression and classification, whose main benefits are that it can handle complex and nonlinear relations in data. There are two types of decision trees based on the type of target variable: *Classification trees* and *Regression trees*.

Decision trees are in practice mostly binary trees where in each parent node (also called *decision node*), a specific attribute of the feature vector is examined and a split is made based on a given criteria whose specifics are learned during the training process. For example if a value of the given attribute

is below a specific threshold, the left branch is followed, otherwise the right branch is followed, with the threshold being set to maximize a certain performance metric of the model. Once the leaf node (also called *terminal node*) is reached, the example is assigned a probability of belonging to a given categorical value in case of a classification tree or assigned a real target value in case of a regression tree.

Different decision tree algorithms use different split rules also known as *criteria*. For example, one of the most common algorithms used for generating a classification tree is called ID3, which selects what attributes from the feature vector to split upon based on *Entropy*, *Information gain* or *Gini Impurity* of the subsets created by the split made on each feature. In the case of Regression Trees, the criterion used is often based on the reduction of *variance* or *standard deviation* between labels and their mean values. Another type of criterion is used by the decision trees in Random Survival Forests, described in more detail in 4, which use a log-rank test statistic in order to maximize the difference between the predicted survival time in each leaf node.

### 1.3.1.1 Building a Classification Tree

A classification tree using the ID3 algorithm is built as follows [6]:

Let  $C=\{1, 2, \dots, p\}$  be the set of possible categorical values,  $\{(x^{(i)}, y^{(i)})\}_{i=1}^N$  is a collection of labeled examples with numerical predictors, where  $N$  is the size of the collection,  $\mathbf{x}^{(i)} = (x_1^{(i)}, \dots, x_k^{(i)})$  is a  $k$ -dimensional feature vector of the example  $i \in \hat{N}$  and  $y^{(i)} \in C$  is its label. The decision tree model is denoted by  $f(\mathbf{x})$  which is a function that estimates the probability of a given example to be of class  $a \in C$  i.e. it is defined as

$$f(\mathbf{x}, a) \stackrel{\text{def}}{=} \Pr(y = a \mid \mathbf{x}), \quad (1.1)$$

where  $\mathbf{x}$  is a  $k$ -dimensional feature vector and  $y$  is a random variable describing the class of a given example.

Let  $S_i$  denote the set of labeled examples included in a given node  $i$ ,  $p^{S_i}(a)$  denote the proportion of the examples in  $S_i$  that belong to class  $a$  and let  $F_i$  denote the set of all possible features in a given node  $i$ . In the first step of the algorithm, the decision tree consists of only its root node that contains all of the labeled examples i.e.  $S_0 = \{(x^{(i)}, y^{(i)})\}_{i=1}^N$ , all features are available i.e.  $F_0 = \hat{k}$  and the proportion is given by

$$p^{S_0}(a) = \frac{\left| \left\{ (\mathbf{x}^{(i)}, y^{(i)}) \mid y^{(i)} = a, (\mathbf{x}^{(i)}, y^{(i)}) \in S_0 \right\} \right|}{|S_0|}. \quad (1.2)$$

The next step of the algorithm is to iterate over all possible features  $j \in F_0$  and all possible threshold values  $t$  (the possible threshold values can be chosen, for example, as midpoints between values of the predictors e.g. if the predictor  $j$  takes on the values  $[1, 2, 3]$  then we may choose the possible threshold values as  $[1.5, 2.5]$ ) and split  $S_0$  into two subsets defined as

$$S_{0-} \stackrel{\text{def}}{=} \{(\mathbf{x}, y) \mid (\mathbf{x}, y) \in S_0, x_j < t\} \quad \text{and} \quad S_{0+} \stackrel{\text{def}}{=} \{(\mathbf{x}, y) \mid (\mathbf{x}, y) \in S_0, x_j \geq t\}. \quad (1.3)$$

For each iteration (ordered pair  $(j, t)$ ), the quality of the split is calculated by using the algorithm's criterion. Assuming that the tree's criterion is entropy, the entropy of a given set of examples is given by

$$H(S) = - \sum_{a \in S} p^S(a) \log_2 p^S(a). \quad (1.4)$$

The quality of the given split is then determined by the weighted sum of the entropies of the two subsets created by the split i.e.

$$H(S_-, S_+) = \frac{|S_-|}{|S|} H(S_-) + \frac{|S_+|}{|S|} H(S_+), \quad (1.5)$$

where the goal is to minimize  $H(S_-, S_+)$ .

Once the best split has been found, each of the created subsets serves as a new decision node, i.e.  $S_1 := S_{0-}$ ,  $S_2 := S_{0+}$  and the branching continues considering only the attributes that were not previously used, i.e. given that  $j_0 \in F_0$  was chosen as the optimal predictor to split upon, then  $F_1 = F_2 = F_0 \setminus \{j_0\}$

The algorithm stops if either there are no further attributes to split upon, all possible decisions would reduce entropy less than some set amount or the tree reaches a set maximum depth (the minimum number of edges connecting the root to a leaf node).

When a new input  $\mathbf{x}$  is introduced in the form of a  $k$ -dimensional vector whose attributes are of the same data type as in the feature vectors  $\mathbf{x}^{(i)}$ , the decision tree is followed from the root down, evaluating the attributes of  $\mathbf{x}$  in each decision node until a leaf node is reached. Let  $S_n$  denote such leaf node, then for  $\forall a \in C$

$$f(\mathbf{x}, a) = p^{S_n}(a) = \frac{\left| \left\{ (\mathbf{x}^{(i)}, y^{(i)}) \in S_n \mid y^{(i)} = a \right\} \right|}{|S_n|}. \quad (1.6)$$

### 1.3.1.2 Building a Regression Tree

This time suppose the opposite case i.e. that the dataset consists of examples with categorical valued predictors and labels with continuous numerical values. As a result, the Standard Deviation Reduction algorithm would be good choice to build the regression tree.

Let  $X_j$  be the set of all possible categorical values for the feature  $j \in \hat{k}$ , where  $k$  is the total number of features in the dataset. Again let  $\left\{ (x^{(i)}, y^{(i)}) \right\}_{i=1}^N$  denote a collection of labeled examples, where  $N$  is the size of the collection,  $\mathbf{x}^{(i)} = (x_1^{(i)}, \dots, x_k^{(i)})$  is a  $k$ -dimensional feature vector of the example  $i \in \hat{N}$ , where  $x_j^{(i)} \in X_j$  for  $\forall j \in \hat{k}$  and  $y^{(i)} \in \mathbb{R}^+$  is the target value. The decision tree model will this time be a function that takes a  $k$ -dimensional feature vector with the same type of attributes as  $\mathbf{x}^{(i)}$ ,  $i \in \hat{k}$  as input, i.e. if  $\mathbf{x} = (x_1, \dots, x_k)$  then  $x_j \in X_j$  for  $\forall j \in \hat{k}$  and outputs a real valued estimation, i.e.  $f(\mathbf{x}) \in \mathbb{R}^+$ .

Let  $S_i$  denote the set of labeled examples included in a given node  $i$ ,  $F_i$  is the set of all possible features in the node  $S_i$ ,  $\sigma(S_i)$  is the standard deviation for labels in  $S_i$ , while  $\sigma(S_i, j)$  is standard deviation in  $S_i$  for feature  $j \in F_i$ .

In the first step of the algorithm, again the decision tree consists of only its root node that contains all labeled example i.e.  $S_0 = \left\{ (x^{(i)}, y^{(i)}) \right\}_{i=1}^N$ ,  $F_0 = \hat{k}$ . The standard deviation of all labels in  $S_0$  given by

$$\sigma(S_0) = \sqrt{\frac{1}{N} \sum_{i=1}^N (y^{(i)} - \bar{y})^2}, \quad (1.7)$$

where  $N$  is the number of examples in  $S_0$  and  $\bar{y}$  is the mean value of labels in  $S_0$ .

For  $\forall j \in F_0$ , the standard deviation for the feature  $j$  is then calculated by

$$\sigma(S_0, j) = \sum_{c \in X_j} p(c) \sigma(c) \quad (1.8)$$

where  $p(c)$  is the proportion of examples in  $S_0$  whose feature  $j$  belongs to the class  $c$ . Let  $N_c$  denote the number of examples in  $S_0$  whose feature  $j$  belongs to the class  $c$  i.e.  $N_c = \left| \left\{ (\mathbf{x}^{(i)}, y^{(i)}) \in S_0 \mid x_j^{(i)} = c \right\} \right|$ , then  $p(c)$  is given by

$$p(c) = \frac{N_c}{|S_0|} \quad (1.9)$$

and  $\sigma(c)$  is the standard deviation of labels in  $S_0$  whose feature  $j$  belongs to the class  $c$ , i.e.

$$\sigma(c) = \sqrt{\frac{1}{N_c} \sum_{i=1}^{N_c} (y_c^{(i)} - \bar{y}_c)^2} \quad (1.10)$$

where  $y_c^{(i)}$  are the labels of examples whose feature  $j$  belongs to the class  $c$  and  $\bar{y}_c$  is their mean value.

The best split is then chosen as the one with the highest *standard deviation reduction* given by

$$\Delta\sigma(S_0, j) = \sigma(S_0) - \sigma(S_0, j). \quad (1.11)$$

$S_0$  is then split into subsets based on the chosen feature  $j$  (the split is not necessarily binary), i.e.

$$S_c = \{(x^{(i)}, y^{(i)}) \in S_0 \mid x_j^{(i)} = c\} \quad \forall c \in X_j \quad (1.12)$$

where each  $S_c$  then serves as a new decision node and the process repeats.

The algorithm is stopped if either the selected reduction in variation is below a set threshold, there are no more features to split upon or if the tree reaches set maximum depth.

When the model is presented with a new input  $\mathbf{x}$ , it follows the tree from the root down until a leaf node is reached. Let  $S_n$  denote such a leaf node, then output values is given by the average value of the labels in  $S_n$ , i.e.

$$f(\mathbf{x}) = \frac{1}{|S_n|} \sum_{(x^{(i)}, y^{(i)}) \in S_n} y^{(i)}. \quad (1.13)$$

### 1.3.2 Ensemble Learning and Random Forests

*Ensemble learning* is a machine learning technique that combines the predictions of multiple simpler models in order to boost its overall performance on a given task. An example of an ensemble learning method are *random forests*, which build on the foundation of decision trees by aggregating the predictions made by a set amount of decision trees, allowing the model to make more accurate predictions than any single decision tree could make by itself. The key idea behind random forests is that every decision tree is trained using a different sample of input features, which helps reduce overfitting and improves the generalization capabilities of the model.

An example of a random forests model is training a group of classification trees on different random subsets of the training data and when faced with new input, predicting the class of the given input by choosing the class that got the most "votes" by the classification trees [7]. A class  $c$  gets a "vote" by the decision tree when the input  $\mathbf{x}$  is assigned the highest probability of belonging to  $c$ , i.e. class  $c \in C$  gets the vote  $\iff c = \arg \max_{a \in C} \Pr(y = a \mid \mathbf{x})$ , where  $C$  is the set of all possible classes.

#### 1.3.2.1 Building a Random Forest

The process of building a random forest model can be divided into four steps [6, 7]:

1. *Sampling the data:* In order to boost the performance of a random forests model (and other ensemble learning methods), it is vital to train the decision trees on different samples of the data in order to minimize risk of the predictors making the same kind of error. Samples are taken randomly from the dataset in order to create multiple different subsets of the data. This process can be divided based on whether the samples are taken "with replacement" or not i.e. whether instances of the data can be used in training of multiple predictors or just once, this is called *bagging* (abbreviation of *bootstrap aggregating*) and *pasting* respectively, with bagging being the generally more often used case for random forests.

2. *Feature sampling*: In order to even more differentiate the individual decision trees, only a random subset of the input features are considered when building a particular decision tree. The model will perform best when the input feature are independent of each other.
3. *Building the decision trees*: For each sample of the data and input features, a decision tree is built using a given split criterion. For even more randomized trees it is possible to randomly select the threshold values for each split in the decision trees, instead of searching for the best possible threshold. These are called *Extremely Randomized Trees*.
4. *Aggregating Predictions*: Random forests make predictions by aggregating the predictions made by the individual decision trees, for example in the case of classification task, the class with the biggest amount of votes is selected while in the case of a regression task, the predicted value is calculated by averaging the values predicted by the individual decision trees.

The most important hyperparameters to be set before the beginning of the training process are the number of trees in the forest, the size of the bootstrap sample, the size of the feature sample, and the split criterion used to build the decision trees.



## Chapter 2

# Renal Transplantation

### 2.1 Chronic Kidney Disease

The *chronic kidney disease* (CKD) is a medical condition in which the kidneys gradually lose their ability to filter waste products and excess fluids from the blood system. CKD is the main cause for renal transplantation worldwide as roughly 90% of all renal transplantation are performed to treat the *end-stage renal disease*, which is the final stage of CKD. The prevalence of CKD is estimated to be between 8% and 16% worldwide [11] and is observed to be on the rise, as in 2015, CKD caused roughly 1.2 million deaths compared to 409,000 in 1990. Symptoms of CKD include fatigue, nausea, loss of appetite, leg swelling and itching. As the disease progresses, it may result in health complications such as anemia, cardiovascular disease, bone disease and nerve damage. In the final stages when the kidneys fail altogether, the affected person needs to regularly undergo dialysis, which supplements the kidney's functions. However, this needs to be done several times a week and each session lasts between 3 to 5 hours, which results in a significant decrease in the quality of the patients life. Thus when possible, transplantation is a much preferred treatment for CKD.

### 2.2 Pre-Transplant Procedures

There are number of procedures that a patient has to undergo before they are admitted onto the kidney transplantation waiting list.

Firstly, their overall health is checked in order evaluate whether they are suitable candidate for the procedure. This includes blood tests, imaging studies, and other diagnostic tests to check for any underlying health conditions that could affect the success of the transplantation.

Secondly, a process called *HLA typing* is performed both on the patient and the recipient. HLA typing assumes paramount importance in determining the immunological compatibility between the donor and the recipient, which is necessary for a successful transplantation. Transplantation between a incompatible paring could lead to an acute transplant rejection, which happens when the recipient's immune system perceives the graft as foreign and rejects it.

#### 2.2.1 HLA Typing

The process of HLA typing involves testing the person's blood and tissue to determine their *human leukocyte antigen* (HLA) type, which is a protein that plays a key role in the body's immune response. Incompatible pairs are discarded while potentially compatible pairs have their blood tested against each other in a laboratory using a process called *cross-matching*.

*Cross-matching* involves testing the blood of the potential donor against the blood of the recipient to ensure that there are no antibodies that could cause a reaction. If the recipient's serum contains antibodies that react with the donor's cells, the complement will be activated and will cause the cells to burst, or lyse. This reaction can be seen under a microscope, and the degree of lysis can be graded to determine the strength of the reaction. When no reaction occurs, the cross-match is called *negative*, whereas if a reaction occurs, the cross-match is called *positive*. A positive cross-match does not automatically mean that the transplantation cannot proceed though, as a successful transplantation may still be possible if the reaction is not too severe, given that the patient is in favorable health condition and that they are given additional treatment with immunosuppressive drugs that suppress the activity of the immune system. This is often the case when other donors are not easily available.

The decision on which pairings to discard and which ones to perform a cross-match on is done by calculating a *HLA mismatch*. The HLA antigens are classified based on their biochemical and functional properties [1]: *HLA class I* and *HLA class II*. Class I antigens are present on the surface of most nucleated cells in the body, while class II antigens are specific to certain cells related to immune function [1]. These classes are then further subdivided into loci (a locus denotes a specific position on a chromosome): HLA-A, HLA-B and HLA-C in the case of HLA class I and HLA-DR, HLA-DQ, HLA-DP in the case of HLA class II. Each locus is further divided into alleles (an allele represents specific DNA sequences at a given locus): HLA-B27, HLA-DRB101, and so on [1]. When calculating the HLA mismatch, a score is given based on the number of allele mismatches for each subclass (HLA-A, HLA-DR, etc.). A higher score indicates a greater degree of immunological incompatibility in the pairing as the best transplant survival has been achieved with no mismatches at HLA-A, -B and -DR. The relationship between HLA mismatches and allograft survival time has been revealed thanks to the UNOS data, that will be used later [1].

Within the UNOS dataset, only the HLA-A, HLA-B and HLA-DR loci are being used computing mismatch. More specifically, the alleles of only two antigens are tracked for each of the mentioned loci. As illustrated in table 2.1 a mismatch is calculated for each locus (A, B, DR) by counting how many of the two antigens are different between the donor and the recipient. The overall HLA mismatch is then given by the sum of these locus mismatches. Note that D- represents donor's typing while R- represents recipient's typing.

Table 2.1: HLA Mismatch

DA1	DA2	DB1	DB2	DDR1	DDR2	RA1	RA2	RB1	RB2	RDR1	RDR2	AMIS	AMIS	DRMIS	HLAMIS
3.0	24.0	35.0	44.0	1.0	2.0	1.0	26.0	38.0	44.0	1.0	4.0	2.0	2.0	1.0	4.0
3.0	32.0	18.0	44.0	11.0	97.0	2.0	23.0	44.0	62.0	7.0	11.0	2.0	2.0	0.0	3.0
2.0	24.0	35.0	41.0	8.0	15.0	2.0	29.0	27.0	44.0	4.0	7.0	1.0	1.0	2.0	5.0
1.0	25.0	7.0	27.0	1.0	13.0	3.0	25.0	7.0	27.0	1.0	13.0	1.0	1.0	0.0	1.0
26.0	29.0	7.0	41.0	7.0	13.0	2.0	68.0	35.0	39.0	3.0	97.0	2.0	2.0	2.0	6.0

## 2.3 Goals

The aforementioned procedures have been very successful at reducing the risk of graft failure shortly after the transplantation. In recent years, improvements in these techniques have successfully minimized the risk of an acute transplantation rejection within the first year of the transplantation below 15% [2]. Nevertheless, it may be possible that there are yet undiscovered correlations effecting the long-term performance of the graft, that are not being accounted for in the current donor-recipient matchmaking process. The progress that has been made in the field of transplantation medicine has also resulted in

the abundance of data about both pre-transplant conditions as well as patient follow-up. This presents an opportunity for the use of machine learning techniques, aimed at uncovering these correlations.

Given that the majority of well-documented data originates from the last 30 years, which is significantly shorter than the average human lifespan, solely considering patients whose grafts have already failed would introduce a bias towards individuals who had a lower probability of survival from the outset. Therefore, it is necessary to include all patients in the analysis, including those with functioning grafts as well as those lost to follow-up (i.e. the status of their graft is unknown). The difficulty lies in the fact that conventional machine learning models do not inherently handle this type of data when considering the patient's survival time as the target variable. However, there exists a field of statistics that addresses these challenges called *survival analysis*.

## Chapter 3

# Survival Analysis

### 3.1 Introduction to Survival Analysis

Survival analysis is a collection of statistical procedures for data analysis for which the outcome variable of interest is time until an event occurs, usually referred to as *time-to-event* or *survival time*. [15]. More specifically, time-to-event refers to the time from the beginning of a follow-up i.e. the period during which the subject of a study is observed, usually from the date of diagnosis, the start of a treatment or transplantation, until the occurrence of an event of interest, usually meaning the death of a patient, disease incidence, relapse from remission, recovery or in the case of kidney transplantation: *graft failure*. Survival analysis involves several statistical techniques that are used during the pre-transplant procedure as a part of the process of finding compatible donor-recipient pairing, namely the *Kaplan-Meier estimator*, the *Log-Rank test* and the *Cox proportional hazards model* just to name a few. These will be explained in detail further in this chapter.

#### 3.1.1 Data Censoring

A crucial challenge that all survival analysis methods have to tackle is *data censoring*, which refers to the lack of data about the exact survival time of an individual, which happens, for example, as a result of a study ending before the said individual experiences an event or if the individual withdraws from the study. This kind of censoring is called *right censoring*, meaning that if the survival time were to be plotted along a horizontal axis, the event would happen to the right side of the cutoff point of the study, but it is unknown by how much.

*Left censored* data are also possible, but they are uncommon in the case of post-transplantation data as the date of transplantation is naturally almost always known. Left censoring can happen, for example, in the case of predicting the survival time after exposure to a virus, as the exact date of this occurrence is often unknown.

#### 3.1.2 Survival and Hazard Functions

The two fundamental functions of survival analysis are the *survival function* denoted by  $S(t)$  and the *hazard function*, also called the *instantaneous failure rate* denoted by  $h(t)$ . The survival function describes the probability of an individual *surviving* (i.e. not experiencing an event) longer, than a specified point in time  $t$ . In another words, it is defined as

$$S(t) = \Pr(T > t) \tag{3.1}$$

where  $T$  is a random variable describing the survival time of an individual.

In contrast, the hazard function, as hinted by its alternative name, gives the the instantaneous potential per unit time for the event to occur, given that the individual has survived up to time  $t$ , that is the hazard function is focused on *failing* (i.e. event occurring) compared to the survival function which is focused on *not failing* (i.e. surviving) [15]. What is meant by the word *potential*, is the probability of an event happening within a given time frame and as this this potential is *instantaneous*, it is given by the following limit

$$h(t) = \lim_{\Delta t \rightarrow 0} \frac{\Pr(t \leq T < t + \Delta t \mid T \geq t)}{\Delta t}. \quad (3.2)$$

While the survival functions may seem to be obvious option for describing models aimed at predicting survival time, it is usually the hazard function, with which the survival models are described by, since it provides a more direct measure of risk over time. Importantly, either of the function can be derived from the other one using the following formulae

$$S(t) = \exp \left[ - \int_0^t h(u) du \right] \quad (3.3)$$

$$h(t) = - \left( \frac{\frac{dS(t)}{dt}}{S(t)} \right) \quad (3.4)$$

### 3.1.3 Hazard Ratio

One of the most common goals of survival analysis is to describe a correlation between a given *exposure variable* ( i.e. a feature of an instance in the data) and the *outcome variable* (e.g. survival time). This relationship is described using the *hazard ratio* (HR), which describes the relative risk of an event occurring in one group compared to another , i.e. it is calculated as the ratio of the hazard functions of the two groups

$$\text{HR}(t) = \frac{h_1(t)}{h_2(t)} \quad (3.5)$$

A hazard ratio of 1, means that the probability of an event occurring at a given time is the same in both groups. If, for example, said groups are sorted by the value of a certain exposure variable and their hazard ratio is 1, this means that there is no relationship between said exposure variable and the outcome variable.

Analogously, a hazard ratio of 5 would mean that the first group has a five times larger hazard than the second group [15].

## 3.2 Survival Analysis Methods

### 3.2.1 Kaplan-Meier Estimator

One of the most widely used methods to estimate the survival function is the *Kaplan-Meier estimator* that works as follows [15]. Let there be a dataset containing  $n$  entries with each entry  $i$  containing information about the individual's survival time  $T_i$  and censoring status  $\delta_i$ , where  $\delta_i = 1$  if an event has been observed at time  $T_i$  and  $\delta_i = 0$  if the entry is right censored. Let  $\tau$  denote the set of all unique event times, i.e  $\tau := \{T_i \mid \delta_i = 1, i = 0, \dots, n\}$ , let  $f_j$  denote the number of individuals who fail at time  $t_j$  and let  $r_j$  denote the number of individuals at risk at time  $t_j$  (i.e. have yet to experience an event or are yet to be censored), then  $\forall t_j \in \tau$  the probability of an individual surviving past  $t_j$  for

$j = 1, \dots, |\tau|$  given that they have already survived until at least  $t_j$  is given by the conditional probability  $\Pr(T > t_j \mid T \geq t_j)$  that is deductible from the dataset as follows

$$\Pr(T > t_j \mid T \geq t_j) = 1 - \frac{f_j}{r_j}. \quad (3.6)$$

Note that the number of individuals at risk does not take in account the individuals that were previously censored, that is  $r_j = r_{j-1} - f_{j-1} - c_{j-1}$ , where  $c_{j-1}$  is the number of individuals that were censored during  $[t_{j-1}, t_j)$ .

The survival function is a function that outputs the probability of surviving past a given time  $t$ , therefore, thanks to the *chain rule* of probability, the *Kaplan-Meier curve*  $\hat{S}$  estimates the survival function at times  $t_1, \dots, t_{|\tau|}$  as follows [15]

$$\hat{S}(t_j) = \Pr(T > t_j) = \prod_{k=1}^j \Pr(T > t_k \mid T \geq t_k). \quad (3.7)$$

Thanks to this notation, the Kaplan-Meier formula is often referred to as the *product-limit formula* as it is a product of probabilities that is limited by the time point being observed. The Kaplan-Meier curve itself assumes that  $\hat{S}(t)$  remains constant over  $t_j \leq t < t_{j+1}$ , therefore the curve has a "stair-like" shape.

### 3.2.2 Log-rank Test

The log-rank test is statistical test used to compare the survival distribution of two or more groups and decide whether their survival curves are statistically equivalent or not [15]. The log-rank statistic is also used as a split-rule criterion for building survival trees that are explained in 4, where it is used to compare the difference in survival times observed between the two groups that are created by a possible split in a given node of a survival tree. Therefore, the following explanation of the log-rank statistic is only concerned with comparing two groups.

Assuming that a dataset contains information about two groups of individuals along with their survival time and censoring status, let  $\tau$  again denote the set of all unique event times regardless of group. Then  $\forall t_j \in \tau$  an expected count of failures  $e_j^{(i)}$  is computed for each group

$$e_j^{(1)} = \frac{r_j^{(1)}}{r_j^{(1)} + r_j^{(2)}} \cdot (f_j^{(1)} + f_j^{(2)}), \quad e_j^{(2)} = \frac{r_j^{(2)}}{r_j^{(1)} + r_j^{(2)}} \cdot (f_j^{(1)} + f_j^{(2)}) \quad (3.8)$$

where  $r_j^{(i)}$  is the number of individuals from group  $i$  at risk at time  $t_j$  and  $f_j^{(i)}$  is the number of failures in group  $i$  at time  $t_j$ .

Let  $O_i - E_i = \sum_{t_j \in \tau} f_j^{(i)} - e_j^{(i)}$  represent the sum of the differences between the observed and the expected counts of failures over all of the unique event times for group  $i$ . Then the log-rank statistic for the group  $i$  is given by

$$L^{(i)} = \frac{(O_i - E_i)^2}{\text{Var}(O_i - E_i)} \quad (3.9)$$

### 3.2.3 Cox Regression

*Cox regression*, also referred to as the *Cox proportional hazards model*, is a mathematical model used for analysis of time-to-event data. Specifically, it is used to model an individual's hazard function based on a set of given predictor variables [15]. It is commonly used in medical research to investigate the

factors that influence the duration until an event occurs. For example, it can be used to predict survival curves of patients who have undergone kidney transplantation and for identifying factors that impact the likelihood of graft failure.

### 3.2.3.1 Formula

The general formula of the Cox regression model describes the hazard function as

$$h(t, \mathbf{X}) = h_0(t) \exp \left( \sum_{i=1}^k \beta_i X_i \right) \quad (3.10)$$

where  $\mathbf{X} = (X_1, X_2, \dots, X_k)$  is a set of  $k$  predictor variables,  $h(t, \mathbf{X})$  represents the hazard function at time  $t$  and given predictor variables  $\mathbf{X}$ ,  $h_0(t)$  is an unspecified *baseline hazard function* and  $\beta = (\beta_1, \beta_2, \dots, \beta_k)$  represent constant coefficients associated with corresponding predictor variables. The fact that  $h_0$  is unspecified, makes Cox regression a semi-parametric model (in contrast to parametric models, whose functional form is known) [15]. Parametric models are more precise when fitted correctly, however it may not always be clear which parametric model is appropriate for a given problem. As a result, a semi-parametric model is a great tool when uncertain about which parametric model to use, as a semi-parametric model will typically approximate the results of a correct parametric model [15].

A core assumption behind the Cox proportional hazards model and thus by extension 3.10 is that the baseline hazard function is dependent only on time, whereas the second part of the expression  $e^{\sum_{i=1}^k \beta_i X_i}$  only involves the  $X$ s, which means the predictor variables are time-independent. This assumption is called the *proportional hazards* assumption. If time-dependent predictor variables were to be considered, a variation of the Cox model would be required called the *extended Cox model*.

To measure the effect of a particular predictor variable on the overall hazard, a *hazard ratio* is calculated without the need to know the baseline hazard function explicitly using only estimates of the  $\beta$  coefficients.

### 3.2.3.2 Partial Maximum Likelihood Estimation

The estimates for the  $\beta$  coefficients can be obtained using a *partial maximum likelihood estimation* method and are denoted as  $\hat{\beta}$ . The estimated model would then be

$$\hat{h}(t, \mathbf{X}) = \hat{h}_0(t) \exp \left( \sum_{i=1}^k \hat{\beta}_i X_i \right) \quad (3.11)$$

Given that we have a random sample of  $n$  entries  $Y_1, Y_2, \dots, Y_n$  with each entry containing information about the  $k$  predictor variables, survival time  $T$  and censoring status  $\delta$  and that the sample comes from an unknown joint probability distribution. Then the maximum likelihood estimation for  $\beta$  is given by

$$\hat{\beta} = \arg \max_{\beta \in \Theta} L(\beta) \quad (3.12)$$

where  $L$  is the maximum likelihood function given by

$$L(\beta) = \prod_{i=1}^n f_{Y_i}(y_i, \beta) \quad (3.13)$$

where  $f_{Y_i}$  is the univariate probability density function associated with the  $i$ -th random variable from the random sample.

However, one of the key features of the Cox model is that there is no assumed distribution for the survival time, therefore it is not possible to compute a *full* maximum likelihood [15]. Therefore, a *partial likelihood* is used, which only considers probabilities of the subjects who fail, i.e. with censoring status  $\delta = 1$ .

This means that the partial likelihood function can be written as

$$L = \prod_{j=1}^p L_j \quad (3.14)$$

where  $p$  is the number of failure times within the random sample and  $L_j$  is the likelihood of failing at the  $j$ -th time. The set of individuals at risk of failing at time  $j$  is denoted by  $R(t_j)$ . It is important to note that while the partial likelihood focuses only on the subjects that fail,  $R(t_j)$  also includes the individuals that will be censored later, thus they are also needed for the computation of  $L_j$ .  $L_j$  is defined as the ratio of the hazard function of the individual, who failed at time  $j$  over the sum of the hazards of individuals at risk at time  $j$ , i.e.

$$L_j(\beta) = \frac{h_0(t) \exp(\beta \cdot \mathbf{X}_j)}{\sum_{i \in R(t_j)} h_0(t) \exp(\beta \cdot \mathbf{X}_i)} \quad (3.15)$$

where  $\mathbf{X}_j$  denotes the vector of predictor variables describing the individual who failed at time  $j$  and  $\mathbf{X}_i$  for  $i \in R(t_j)$  denotes the vector of predictor variables describing the individuals at risk at time  $j$ . As  $h_0(t)$  cancels out, the partial likelihood function can be written as

$$L(\beta) = \prod_{j=1}^p \frac{\exp(\beta \cdot \mathbf{X}_j)}{\sum_{i \in R(t_j)} \exp(\beta \cdot \mathbf{X}_i)} \quad (3.16)$$

Once the partial likelihood function  $L$  is formed, the equation 3.12 is generally solved by maximizing the natural log of  $L$ , i.e. finding the roots of

$$\frac{\partial \ln L}{\partial \beta_i} = 0 \quad \text{for } i = 1, \dots, k \quad (3.17)$$

This gives us a system of equations that can be solved numerically [15].

### 3.2.3.3 Hazard Ratio Estimation

As described in 3.1.3, the hazard ratio is defined as the ratio of hazards of two different individuals. The estimation of the hazard ratio is therefore given by

$$\hat{HR} = \frac{\hat{h}(t, \mathbf{X}^*)}{\hat{h}(t, \mathbf{X})} \quad (3.18)$$

where  $\mathbf{X}^* = (X_1^*, X_2^*, \dots, X_n^*)$  and  $\mathbf{X} = (X_1, X_2, \dots, X_k)$  denote the sets of predictors of two different individuals. The ratio is commonly written so that the higher-valued hazard is in the numerator. This ensures that the ratio is always greater than 1. By using 3.11 we get that

$$\hat{HR} = \frac{\hat{h}_0(t) \exp\left(\sum_{i=1}^k \hat{\beta}_i X_i^*\right)}{\hat{h}_0(t) \exp\left(\sum_{i=1}^k \hat{\beta}_i X_i\right)} = \exp\left(\sum_{i=1}^k \hat{\beta}_i (X_i^* - X_i)\right) \quad (3.19)$$

meaning that the hazard ratio stays constant over time, i.e. the hazards are *proportional* and the effect of the predictor variables on the hazard rate remains constant over time.



Now, in order to compute the effect of a predictor variable  $X_j$  on the overall hazard while adjusting for other variables, let vectors  $\mathbf{X}^*$  and  $\mathbf{X}$  be defined as

$$\begin{cases} X_i^* = X_i & \text{for } i \neq j \\ X_i^* \neq X_i & \text{for } i = j \end{cases} \quad (3.20)$$

The estimated hazard ratio for the predictor variable  $X_j$  is then given by

$$\hat{HR} = \exp(\hat{\beta}_j (X_j^* - X_j)) \quad (3.21)$$

As hazard functions are always positive and the ratio is written in way as to ensure that it is always greater than 1, the greater the hazard ratio, the greater the predictor's influence on the overall hazard.

#### 3.2.3.4 Adjusted Survival Curves

Upon fitting a Cox regression model onto survival data, survival curves that adjust for the predictor variables (also called *adjusted survival curves*) are defined in a similar fashion as

$$S(t, \mathbf{X}) = S_0(t) \exp\left(\sum_{i=1}^k \beta_i X_i\right) \quad (3.22)$$

Using estimates  $S_0(t)$  and  $\hat{\beta}_i$  computed by the partial likelihood estimation method for  $S_0(t)$  and  $\beta_i$  respectively. We get an estimated survival function

$$\hat{S}(t, \mathbf{X}) = S_0(t) \exp\left(\sum_{i=1}^k \hat{\beta}_i X_i\right) \quad (3.23)$$

and by specifying the input vector of predictors  $\mathbf{X}$  we get a survival curve adjusted for the given predictors.

#### 3.2.3.5 Uses

The Cox regression model will later be used as a benchmark to compare the performance of the machine learning model to a conventional survival analysis model.

## Chapter 4

# Random Survival Forests

*Random Survival Forests* (RSF) is an ensemble machine learning method used for analyzing right-censored survival data. RSF combine the concepts of *survival analysis* and *random forests*, which allows it to handle issues commonly associated with conventional survival analysis methods, such as restrictive assumptions about the model's parameters (e.g. proportionality in the Cox regression model), the inability to handle nonlinear relations in the data and issues with missing data, as well as issues associated with regular random forest models, such as being restricted to regression and classification tasks and the inability to handle right-censored survival data [13].

Each tree in the forest outputs a cumulative hazard function  $H(t | \mathbf{x})$ , which takes a vector of predictive variables  $\mathbf{x}$  describing a given individual and time  $t$  as inputs, and assigns a probability of an event occurring to the individual at time  $t$ . The forest's overall prediction, i.e. *the ensemble cumulative hazard*, is then computed as the average of the individual tree predictions.

Random Survival Forests have been used in a wide range of applications, including medicine, engineering, and social sciences, to model survival outcomes and identify important prognostic factors.

### 4.1 Building a Random Survival Forests model

The algorithm for building a RSF model works as follows [13].

1. A set number of subsets is taken from the data using the bootstrap sampling method as described in 1, with each sample excluding on average 37% of the original data. The excluded data is referred to as *out-of-bag data* (OOB) and is later used for computing the ensemble cumulative hazard.
2. A *binary survival tree* (BST) is built for each of the bootstrap samples. The algorithm for growing a BST generally follows the same principles as described in 1.3.1, i.e. at each node the algorithm iterates over all possible features and values for them and decides on the best split. The best split is decided by maximizing the log-rank statistic 3.2.2, which compares the survival times of the two groups created by the potential split. The algorithm is restricted by a criterion that each node should contain at least 1 unique event time, which leads to a point when no new splits can be made meaning the tree has been fully grown.
3. Once a tree is fully grown, each terminal node outputs either a survival function or a *cumulative hazard function* (CHF) denoted by  $\hat{H}(t)$  which is calculated as follows:  
Let  $A$  denote the set of all terminal nodes, then each node  $j \in A$  contains a set of individuals  $(T_{1,j}, \delta_{1,j}), \dots, (T_{n(j),j}, \delta_{n(j),j})$ , where  $n(j)$  is the number of cases belonging to the node  $j$ ,  $T_{i,j}$  is the survival time of the  $i$ -th case belonging to the node  $j$ ,  $\delta_{i,j} \in \{0, 1\}$  is the censoring status of the

$i$ -th case at time  $T_{i,j}$ , with  $\delta_{i,j} = 0$  if the individual is right-censored (e.g. has a still functioning graft or has been lost to followup) and  $\delta_{i,j} = 1$  if an event has been observed at a given time  $T_{i,j}$  (e.g. graft failure). Let  $t_{1,j} < t_{2,j} < \dots < t_{N(j),j}$  denote the unique event times observed in node  $j$ , where  $N(j)$  is the number of observed unique event times and let  $f_{k,j}$  and  $r_{k,j}$  represent the number of individuals who experienced an event (fail) or were at risk at time  $t_{k,j}$  respectively. Then either a survival function is calculated using the Kaplan-Meier estimator 3.2.1, or a CHF for the node  $j$  is given by the Nelson-Maier estimator

$$\hat{H}_j(t) = \sum_{t_{k,j} \leq t} \frac{d_{k,j}}{Y_{k,j}} \quad (4.1)$$

When a new input vector  $\mathbf{x}$  is introduced to the model, the tree is followed from the top down until a terminal node  $j \in A$  is reached. The CHF of the whole tree is then given by

$$H(t | \mathbf{x}) = \hat{H}_j(t). \quad (4.2)$$

4. Next, two ensemble cumulative hazards functions are calculated. The first, called the *bootstrap ensemble CHF* (denoted by  $H_e^*$ ), simply averages over all grown survival trees. The second, called the *OOB ensemble CHF* (denoted by  $H_e^{**}$ ), averages over trees where a given input was not used in the training of the given tree i.e. is a case of the OOB data. Suppose an example  $\mathbf{x}_i$  is taken from the training dataset, let  $B$  denote the total number of bootstrap samples, and let  $H_b^*(t|\mathbf{x})$  denote the CHF predicted by the tree grown from the bootstrap sample  $b \in \hat{B}$  and let

$$I_{i,b} = \begin{cases} 1 & \text{if } \mathbf{x}_i \text{ belongs to the OOB subset of the dataset for bootstrap sample } b \\ 0 & \text{else, i.e. } \mathbf{x}_i \text{ was used in the training of a tree grown from the bootstrap sample } b \end{cases}.$$

The bootstrap ensemble CHF is then defined as

$$H_e^*(t | \mathbf{x}_i) = \frac{1}{B} \sum_{b=1}^B H_b(t | \mathbf{x}_i) \quad (4.3)$$

where  $H_b(t | \mathbf{x}_i)$  is the CHF of a tree grown from the bootstrap sample  $b$ , defined in (4.2). In contrast, the OOB ensemble CHF is defined as

$$H_e^{**}(t | \mathbf{x}_i) = \frac{\sum_{b=1}^B I_{i,b} H_b(t | \mathbf{x}_i)}{\sum_{b=1}^B I_{i,b}}. \quad (4.4)$$

## 4.2 Hyperparameters

TODO: List hyperparameters

## 4.3 Harell's Concordance Index

The prediction error of a random survival forests model is estimated using the *Harell's concordance index* (also known as *C-index*) [13], which is an metric commonly used in medical research to evaluate survival models. The C-index is defined as the ratio of correctly ordered pairs to comparable pairs [16] (explained in more detail in 4.3.1). The C-index is also closely related to the *Area under the ROC*

*curve* (short for Receiver Operating Characteristics curve) also known as AUC, which is a graphical plot that illustrates the performance of a binary classification model at different classification thresholds [14, Classification - ROC Curve and AUC].

Specifically, it plots the *true positive rate* (TPR) defined as

$$TPR = \frac{TP}{TP + FN} \quad (4.5)$$

where  $TP$  is the number of *true positives* (i.e. positive cases correctly classified by the model) and  $FN$  is the number *false negatives* (i.e. positive cases incorrectly classified as negative), against the *false positive rate* (FPR) defined as

$$FPR = \frac{FP}{FP + TN} \quad (4.6)$$

where  $FP$  is the number of *false positives* (i.e. negative cases incorrectly classified as positive) and  $TN$  is the number of *true negatives* (i.e. negative cases correctly classified as negative) [14]. Lowering the classification threshold results in more cases being classified as positive, thus generally increasing both the true positive and the false negative rate.

The area under the ROC curve can be interpreted as the probability that a given model will rank a random positive example more highly (e.g. it will assign a higher probability of being positive) than a random negative example. The C-index can attain values from 0 to 1, where 1 means a perfect model whose predictions are always correct and 0 being the opposite. A random classifier that assigns the predicted value with a uniform distribution, will have a C-index of 0.5. Generally, models with a C-index above 0.7 are considered to be good predictors. An example of a machine learning models that use the AUC for evaluation of their performance are classification decision trees 1.3.1 and by extension random forests used for classification .

The C-index works in similiar fashion to the AUC in the sense that it compares two random individuals, and checks whether the model assigns a more favorable hazard function to the patient with longer survival time.

### 4.3.1 Calculating Harell's Concordance Index

Not all individuals can be compared to each other as a result of censoring. The C-index is therefore calculated as follows [13]:

1. Find all possible pairs of cases across the test dataset while excluding the following cases:

- pairs  $((T_i, \delta_i), (T_j, \delta_j))$  where the case with the shorter survival time is right-censored  
i.e.  $T_i < T_j \wedge \delta_i = 0$
- pairs  $((T_i, \delta_i), (T_j, \delta_j))$  with equal survival times where both cases are also right-censored  
i.e.  $T_i = T_j \wedge \delta_i = \delta_j = 0$

The cases that are left are called *permissible* (or *comparable*) and we denote their set by  $P$ .

2. For each permissible pair  $((T_i, \delta_i), (T_j, \delta_j))$  a value for  $c_{i,j}$  is assigned as follows:

- (a) if  $T_i \neq T_j$  then

$$c_{i,j} = \begin{cases} 1 & \text{if } T_i < T_j \implies i \text{ has worse predicted outcome than } j \\ 0.5 & \text{if } T_i < T_j \implies \text{predicted outcomes of } i \text{ and } j \text{ are tied} \\ 0 & \text{else} \end{cases} \quad (4.8)$$

(b) if  $T_i = T_j \wedge \delta_i = \delta_j = 1$

$$c_{i,j} = \begin{cases} 1 & \text{if predicted outcomes of } i \text{ and } j \text{ are tied} \\ 0.5 & \text{else} \end{cases}$$

(c) if  $T_i = T_j \wedge \delta_i \neq \delta_j$

$$c_{i,j} = \begin{cases} 1 & \text{if } \delta_i = 1 \implies i \text{ has worse predicted outcome than } j \\ 0.5 & \text{else} \end{cases}$$

3. The C-index is then given by

$$C = \frac{1}{|P|} \sum_{i,j \in P, i \neq j} c_{i,j}. \quad (4.7)$$

Whether a given case has a better or a worse predicted outcome is decided as follows. Let  $t_1, \dots, t_n$  be a set of pre-chosen time points, then case  $i$  has a worse predicted outcome than case  $j$  if

$$\sum_{k=1}^n H_e^{**}(t_k | \mathbf{x}_i) < \sum_{k=1}^n H_e^{**}(t_k | \mathbf{x}_j) \quad (4.8)$$

where  $\mathbf{x}_l$  is a feature vector of the case  $l$ , and  $H_e^{**}$  is the OOB ensemble CHF defined in (4.4).

The prediction *error* ( $PE$ ) is then simply given by  $PE = 1 - C$ .

## 4.4 Permutation Variable Importance

Since survival trees use the log-rank statistic as a split criterion as opposed to impurity used in conventional decision trees, variable importance cannot be computed using the mean decrease in impurity method described in 1.2.3. Consequently, the *permutation feature importance* described in the original paper introducing random forests [10], which is fortunately also implemented within the scikit-learn package.

Permutation importance is calculated by first calculating a baseline metric by evaluating the model's performance when trained on the complete set of features. Next, a feature is permuted from the validation set and the model's performance is evaluated again. The permutation importance of a given variable is then given by the difference between the baseline metric and the model's performance after permuting said variable [17].

## 4.5 Other Methods

Random Survival Forest is not the only model that combines conventional machine learning techniques with survival analysis. The scikit-survival library 5.2.3 also provides implementations of *Survival Support Vector Machine* and *Survival Gradient Boosting*.

### Survival Support Vector Machine

The survival support vector machine is an extension of the conventional *support vector machine* (*SVM*), which is a supervised machine learning model that works by mapping the feature vector of each sample into high-dimensional space and calculating a hyperplane that separates them based on their

labels, called a *decision boundary*. When faced with a new input, the SVM makes a decision based on which side of the decision boundary the new case falls [6].

The survival SVM differs from the regular SVM in the sense that it uses survival time as well as censoring status as labels and uses both of these to calculate the hyperplane. The survival SVM can work with two types of problems: a regression problem which outputs predicted survival time and a ranking problem which ranks individuals based on their survival time. This is however slightly disadvantageous as these types of predictions are not easily transferrable to standard survival analysis outputs, namely the survival and hazard functions [16].

### Survival Gradient Boosting

Survival gradient boosting builds on the basis of regular *gradient boosting*, which is an ensemble supervised machine learning technique that works by combining predictions of many *weak learners*, models that are only slightly better than random guessing, and sequentially improving their performance by gradually optimizing a loss function by using its gradient. Gradient boosting models generally achieve greater accuracy than random forests, however take much longer to train [6, 16].

While in the case of conventional gradient boosting the loss function given by prediction error (e.g. mean squared error), in the case survival gradient boosting the loss function is the partial likelihood loss Cox regression described in 3.2.3.

The reasons for why the RSF model was chosen include the above described disadvantages of the aforementioned alternatives and promising accuracy that the RSF model displayed in the related articles [5, 3].

## Chapter 5

# Data and Software Architecture

### 5.1 Data Acquisition

The data used for training was obtained from the United Network for Organ Sharing (UNOS) which is a nonprofit organization that manages the organ transplant system in the United States. The organization oversees transplantation procedures (matching donors, ensuring fair graft allocation) for multiple organs such as kidneys, heart, lungs, liver, pancreas and intestines. As a result, UNOS maintains a large database of organ transplantations carried between 1984 and today. Specifically, there are more than a million entries regarding kidney transplantations, an amount sufficient for training a machine learning model.

The data can be obtained free of charge via <https://unos.org> (a VPN may be needed for access from outside the US) upon filling out the request form on the website and signing relevant documents regarding the use of the data that will be sent via a provided email. The data was obtained in the form of a tab-delimited file that was transformed into a MongoDB database using the following GitHub repository <https://github.com/ceharvs/transplant2mongo>. The database tables were then converted into a JSON format using the Database tools plugin in IntelliJ IDEA Ultimate.

Alongside the data from the UNOS database, another dataset was provided by the Czech Institute for Clinical and Experimental Medicine (IKEM), however the dataset was limited in size and thus was not suitable for training a machine learning model.

As the dataset provided by IKEM was the first to be acquired, the option to train the model partly on generated data was considered to supplement the limited size of the IKEM dataset, however as the much larger UNOS dataset was acquired, this was no longer necessary.

### 5.2 Software Architecture

A number of software tools and libraries were used for building the survival analysis machine learning model, with the backbone of the application being the Python based *scikit-survival* package, which is itself built upon the *scikit-learn* package. Additionally, the *pandas* and *numpy* libraries were used for both preprocessing of the data and evaluating the models performance and finally, the *matplotlib* library was used to provide graphical output in order visually validate models performance.

#### 5.2.1 Python

*Python* is an interpreted, general-purpose, high-level, object-oriented programming language that consistently ranks among the most used programming languages in the world [18]. The reasons for its popularity include relatively easy human readability, built-in data structures such as lists and dictionaries,

wide range of libraries that provide additional functionality, automatic memory management and the fact that Python is cross platform, meaning that code written on one machine can easily be run on different machines, regardless of their operating system. [19, 20]

### 5.2.2 Jupyter Notebook

*Jupyter Notebook* is a notebook authoring web application that provides tools for interactive computing with *computational notebooks* (also known as *notebook interface*). Computational notebooks are documents that enable users to integrate executable computer code, plain text, visualization and other interactive tools in a structured manner. [22] One of the fundamental advantages of computational notebooks is the ability to partition code into smaller self-contained units with each unit having the result of their execution attached to them. This facilitates easier analysis of results and makes information extraction more efficient. This proves to be beneficial especially in the fields of data analysis and machine learning where extracting information from data is of utmost importance. Jupyter Notebooks are primarily used for executing Python code. Nonetheless, as indicated by its name, Jupyter Notebooks also provide support for the Julia and R languages.

### 5.2.3 scikit-learn and scikit-survival

#### scikit-learn

*scikit-learn* is an open-source machine learning Python module. It is written in Python and C and it is the most frequently used library for machine learning [6]. *scikit-learn* provides tools for every part of the process of building a machine learning model, namely data preprocessing, model selection, fitting, and evaluation, among others [17].

#### scikit-survival

*scikit-survival* is a Python module built on top of *scikit-learn*. It utilizes the functionalities of *scikit-learn* for machine learning, while allowing the user to implement survival analysis methods in their models [16]. Included in the library are many of the standard data analysis methods including the Kaplan-Meier estimator 3.2.1 and Cox Regression 3.2.3 as well as methods combining survival analysis with machine learning methods such as the Random Survival Forests 4.

### 5.2.4 Other libraries

#### Pandas

*Pandas* is an open-source library built on top of the Python programming language used for manipulation with data and data analysis in general. The core of the library is the *DataFrame* object that is used to organize data in a tabular format. *Pandas* provides a multitude of tools for working with *DataFrames* including the ability to extract statistics about the data, handling of missing values, reshaping data sets, as well as the ability to import and export *DataFrames* in various formats such as .csv, .json and more [23].

#### NumPy

*NumPy* is an open-source Python library used for working with numerical data. More specifically, it provides tools for working with multidimensional numerical arrays and is widely used in other data analysis oriented libraries such as *Pandas*, *Matplotlib* and *scikit-learn*. The centerpiece of the library



is the *ndarray* which is a homogenous  $n$ -dimensional array object, that is equipped with methods to efficiently operate on it [24].

### Matplotlib

*Matplotlib* is a Python library based on NumPy used for creating both static and animated visualizations of data. These visualizations mainly appear in various types of plots such as line plots, histograms, scatter plots, and 3D plots. It draws inspiration from the MATLAB programming language and Matplotlib itself provides an extension called *Pyplot*, which is a collection of functions that replicate the behaviour of MATLAB for Python [25]. In the field of machine learning, matplotlib is often used to validate and evaluate the training process of a machine learning model in order to prevent overfitting or underfitting.

#### 5.2.5 Cluster Computing

Due to the large size of the tables extracted from the UNOS dataset, reaching up to 40GB in the JSON format, conducting any meaningful operations locally became infeasible since in order to do any sort of processing, the data needed to be first loaded into main memory, which, in my case, had a capacity of only 12GB. As a result, all computing was conducted remotely on the HELIOS high-performance computing cluster hosted at the Department of Mathematics, Faculty of Nuclear Sciences and Physical Engineering, Czech Technical University in Prague. The cluster can provide up to 384GB of main memory on a single node, which was more than enough for the needs of this project. The connection to the cluster was established using SSH in the Linux shell. To enhance practicality, I configured node forwarding from the remote node to localhost, which allowed me to interact with Jupyter notebooks directly via a web browser on my machine. More information about the cluster such as about the hardware specs can be found at [21].

## Chapter 6

# Model Training

The code used for training the machine learnign model including pipelines used for data preprocessing, training and evaluating the model are publicly available at [https://github.com/matejkloucek/bp\\_don\\_rec\\_compatibility](https://github.com/matejkloucek/bp_don_rec_compatibility) in the form of Jupyter Notebooks. Intructions on how to properly execute them are included in the README file located in the above GitHub repository.

### 6.1 Data Preprocessing

#### 6.1.1 Separating Living and Deceased Donors

There are 6 tables to be extracted from the UNOS dataset using MongoDB as mentioned in 5.1, though only 3 are relevant to this study. These 3 being:

- **Kidney\_Pancreas**, which contains information about kidney and pancreas transplantations performed between 1994 and 2022 in the US including information about both recipients and donors.
- **Living\_Donor** and **Deceased\_Donor**, which contain more detailed information about living and deceased donors respectively.

The **Kidney\_Pancreas** table also happens to be the largest in terms of file size. After converting the table to JSON format, the resulting file size reached to approximately 40GB. Consequently, loading the file into a Pandas DataFrame took a significant amount of time. Therefore, it proved advantageous to convert all tables from JSON to .csv format, which drastically reduced loading time. This efficiency can be attributed to the fact that unlike JSON, which stores data in a hierarchical structure using key-value pairs, the .csv format stores data in a tabular (and therefore faster to read from) format.

In its original form, the **Kidney\_Pancreas** dataset contained 1108884 entries and 470 columns. It contains information about both living and deceased donors, but the details regarding the donors are limited in scope, particularly in the case of living donors. To adress this limitation, a table join was performed with the **Living\_Donor** and **Deceased\_Donor** tables to provide more information about the donors. This merge was based on the **DONOR\_ID** column, with 174381 entries sharing a common **DONOR\_ID** between the **Kidney\_Pancreas** and **Living\_Donor** tables and 399822 entries sharing it between the **Kidney\_Pancreas** and **Deceased\_Donor**.

Preliminary experimentations with the data and RSF models have revealed that the features impacting graft survival vary between living donor and deceased donor transplantations. Consequently, the dataset has been split on based on the type of transplantation performed, separating living and decaed donor

transplantations. Subsequently, independent machine learning models were trained on each of the created subsets separately.

The two new datasets created by the merge and the subsequent split will from now be referred to as too as `Kidpan_Living` and `Kidpand_Deceased`. The `Kidpan_Living` table had 174381 entries and 649 columns, while the `Kidpand_Deceased` table had 399822 entries and 893 columns. In both cases, the number of columns is usually considered to be high for an effective machine learning model as it could lead to overfitting and poor explainability. Therefore, the number of features had to be reduced significantly.

### 6.1.2 Reducing Number of Features

In addition to the dataset, UNOS provides a .xlsx file called *STAR Files Data Dictionary* (STAR = Standard Transplant Analysis and Research), which contains information about what each of the columns represents. This dictionary was used in the following sections to provide context for the dataset's features.

#### Simultaneous Transplantations

The dataset contains information about both kidney and pancreas transplantations, as they are sometimes performed together in a medical procedure known as the *simultaneous kidney-pancreas transplant* (SKP) [28]. In fact, most (83%) of pancreas transplantation are performed in the context of SKP's [29]. However, the focus of this paper is solely on kidney transplantations and the number of SKPs is relatively low compared to the number of stand-alone kidney transplantation. Therefore individuals, who had undergone a SKP were dropped from the dataset and all columns regarding pancreas transplantation were dropped as well. These columns were selected manually based on their description in the STAR dictionary.

#### Follow-up Data

As the goal of this paper is to predict compatibility of patients prior to transplantation, any column that contains information about the patient's status after or at the time of the transplantation had to be removed from the dataset. This was also done by manually selecting such columns based on their description in the STAR dictionary. The only follow-up features that were retained are the `GTIME_KI` and `GSTATUS_KI` columns, which describe the graft's survival time and censoring status respectively.

#### Duplicate Columns

As the original `Kidney_Pancreas` table also contained some information about donors, duplicate columns were created upon merging with the `Living_Donor` and `Deceased_Donor` tables. Some were easy to identify as they shared the same name, while others had to be identified manually.

#### Irrelevant Columns

The STAR dictionary was also used to identify obviously irrelevant features and drop them to speed up the training process. These include IDs, codes and form statuses used internally by either UNOS or individual transplant centers.

### Deceased and Living Donor Columns

As mentioned above, the `Kidney_Pancreas` table itself contains limited information about both deceased and living donors. Therefore, columns regarding deceased donors had to be dropped from the dataset used for training of the model for living donors and vice versa. This was done excluding columns with high percentage of missing values ( $> 50\%$ ) as the columns regarding deceased donors would be left unfilled in the case of grafts from living donors. The opposite case was handled in the same fashion accordingly.

### Low Quality Data

Dropping columns with high percentage of missing values was also done to prevent a significant loss of information by having to either impute large amounts of missing values or discarding entries with missing features. Columns with only one unique value were dropped as well.

### HLA typing

As illustrated in the table 2.1, each locus of both the donor and the recipient has two columns. These columns are categorical and represent different alleles. The number of unique alleles exceeds 100 in some cases. This would create a problem when encoding categorical columns as it would create an enormous amount of new columns. Fortunately, these columns can be dropped from the dataset as they are already used for calculating HLA mismatch and therefore dropping them does not result in a significant loss of information.

Overall, these measures reduced the number of features to 137 in the case of `Kidpan_Living` and to `<number>` in the case of `Deceased_Donor`.

### 6.1.3 Ensuring Correct Formatting

Upon loading the dataset into a `DataFrame`, many of the columns had incorrect data types. Specifically there were two issues:

1. Categorical variables that were described using digits were incorrectly classified as numerical. This would lead to misleading results and had to be dealt with by identifying these columns using the STAR dictionary and changing their data type to `object` so that they could be encoded later.
2. Columns containing dates were formatted as strings and were thus identified as categorical. This would lead to problems when encoding since there would be a new column created for every unique time. This was dealt with by only using the year and setting its datatype to `float64`.

### 6.1.4 Imputing Missing Values

Missing values were imputed using scikit-learn's `SimpleImputer()` with the mean strategy for numerical features and `most_frequent` strategy for categorical features.

### 6.1.5 Feature Encoding

The RSF model requires all input variables to be numerical. However, many of the features in the dataset were categorical and therefore had to be encoded before they could be used for training the RSF model. Scikit-learn's `OneHotEncoder` was used to encode the categorical variables. Encoding the

whole Kidpan\_Living dataset resulted in the training dataset having 812 columns after merging with the categorical values. In the case of Kidpan\_Deceased, this number reached to <number>.

After feature selection described in 6.2, the encoded dataset had 111 columns in the case of Kidpan\_Living and <number> in the case of Kidpan\_Deceased. After training

## 6.2 Feature Selection

After training the model on the whole dataset, the performance of the model described by the C-index was 0.6345 for the living donor model and <number> for the deceased donor model. The hyperparameters used for this training were set to: `n_estimators=100`, `min_samples_split=10`, `min_samples_leaf=10` and `max_depth=None`.

In order to simplify the model and improve its performance, the best features were selected by calculating their importance using permutation importance described in 4.4. In comparison to variable importance calculated by the Gini importance 1.2.3, computing permutation importance is a very time consuming task. The scikit-learn's implementation of permutation importance offers a `n_repeats` parameter, which controls the number of times a feature is permuted in the dataset resulting in increased accuracy of the feature importance by reducing randomness. However, this also greatly increases the computational complexity. Thus, the variable importances were computed initially on smaller dataset, created by dropping all entries with missing features, with `n_repeats=5`. This was followed by a computation on a larger dataset with imputed missing values with `n_repeats=1`. The two resulting tables of variable importances were used to create a list of the 23 most important features. The variable importance of those features was then validated by computing permutation importance on a dataset consisting only of these features with `n_repeats=5`. In addition, the 4 columns regarding HLA typing (AMIS, BMIS, DRMIS, HLAMIS) were added to study their influence on the graft's long term survival, resulting in 27 features being selected in total.

The selected features along with their description and variable importance can be found in table 6.1.

Table 6.1: Variable Importance

Feature	Mean Importance	St. Dev. Importance	Description	Data type
AGE	0.024871	0.007475	Recipient's age	Numerical
AGEDON	0.012540	0.001580	Donor's age	Numerical
PRI_PAYMENT_TCR_KI	0.012119	0.001041	Recipient's primary payment source	Categorical
ETHCAT	0.010511	0.002499	Recipient's ethnicity category	Categorical
END_BMI_CALC	0.009909	0.001080	Recipient's BMI	Numerical
TOT_SERUM_ALBUM	0.009846	0.001971	Recipient's total serum albumin	Numerical
WORK_INCOME_TCR	0.009702	0.000435	Is recipient working for income	Categorical
PRE_TX_TXFUS	0.009500	0.001591	Did recipient receive transfusion	Categorical
ETHCAT_DON	0.009020	0.000407	Donor's ethnicity category	Categorical
DIABETES_DON	0.008933	0.000531	Donor's diabetes status	Categorical
DAYSWAIT_CHRON_KI	0.008736	0.000717	Days recipient spent on waiting list	Numerical
EBV_SEROSTATUS	0.008692	0.000600	Recipient's status of Epstein-Barr virus	Categorical
ON_DIALYSIS	0.008667	0.000418	Is recipient regularly administered dialysis	Categorical
PREV_TX_ANY	0.008667	0.000627	Recipient's past transplantations	Categorical
DAYSWAIT_ALLOC	0.008268	0.000643	Days recipient was prioritised for allocation	Numerical
HCV_SEROSTATUS	0.008215	0.000306	Recipient's Hepatitis C status	Categorical
HIST_CIG	0.008208	0.001030	Recipient's history of cigarette use	Categorical
FUNC_STAT_TCR	0.008014	0.000208	Recipient's overall health status	Categorical
KI_CREAT_PREOP	0.007738	0.000611	Donor's creatinine level	Numerical
DIAB	0.007714	0.000167	Recipient's diabetes status	Categorical
PERIP_VASC	0.007668	0.000300	Recipient's peripheral vascular disease status	Categorical
AMIS	0.006838	0.001141	Mismatch at the HLA-A locus	Numerical
DRMIS	0.005818	0.002427	Mismatch at the HLA-DR locus	Numerical
BMIS	0.003345	0.001171	Mismatch at the HLA-B locus	Numerical

Using the same hyperparameters as before, the C-index of the model using the reduced list of features improved to 0.6366.

### 6.3 Hyperparameter Tuning

While for training of the initial models and feature selection the dataset has been split into only the training and the test set, with 80% of the dataset being reserved for training and 20% for testing. For hyperparameter tuning, the dataset has been split into three folds: the training, validation and test sets with the ratio being 70% for training and 15% for validation and testing each. See 1.2.1 for why this was done.

After experimenting with different combinations of hyperparameters using grid search 1.2.4, the optimal parameters were found to be: `n_estimators=200`, `min_samples_split=10`, `min_samples_leaf=15` and `max_depth=None`. Using these hyperparameters for training the dataset resulted in the improvement of the model's C-index to 0.6374.

### 6.4 Example Usage

Once trained, the model can be used to predict allograft's survival as well as cumulative hazard functions based on information about the recipient and the donor as listed in table 6.1. In addition, the model can also be used to predict a risk score  $R$ , which is given by

$$R(\mathbf{x}) = \sum_{j=1}^{n(h)} H(T_j | \mathbf{x}) \quad (6.1)$$

where  $\mathbf{x}$  is an input feature vector,  $h$  is the terminal node that was reached when  $\mathbf{x}$  was introduced,  $n(h)$  is the number of unique event times in node  $h$  and  $H(t | \mathbf{x})$  is the cumulative hazard function defined in 4.2 [16]. This risk score could be a good way to quantify the compatibility between a potential donor-recipient pairing.

Illustrated in figures 6.1, 6.2 are predicted survival functions, cumulative hazard functions and risk scores of 5 randomly selected samples from the test set. In case they were censored, the dotted line represents their observed time of censoration, else, the dashed line represents their graft failure time.

Figure 6.1: Predicted survival functions

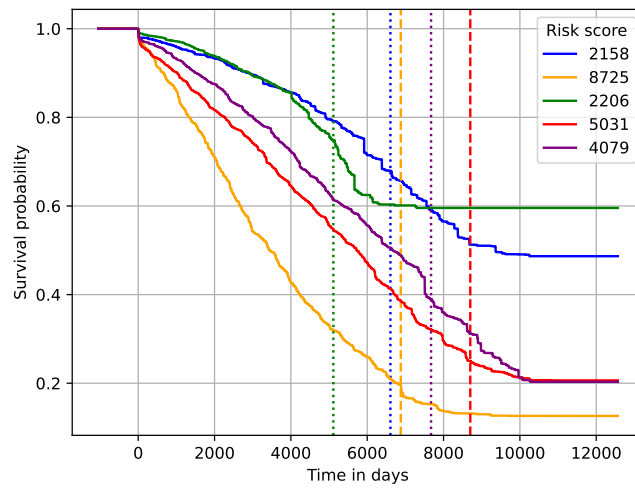
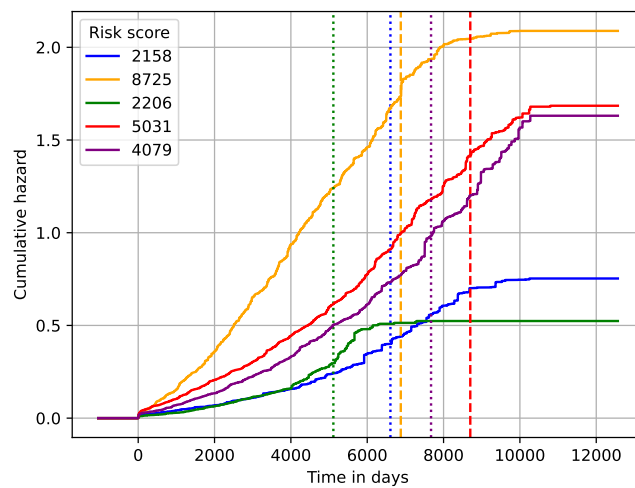


Figure 6.2: Predicted cumulative hazard functions



## Chapter 7

# Model evaluation

### 7.1 General evaluation

The accuracy of the model given by the C-index is lower than the one developed in [5] which had a C-index of 0.7. This is because they used predictors that are unknown prior to the transplantation itself. One such example is the *cold ischemia time*, which describes the time the graft spends outside the body of both the donor and recipient. This feature in particular has a significant impact on the recipient's survival curve, thus giving their model a significant boost in accuracy. However, as mentioned before, the goal of this paper is to develop a machine learning model that can be used for computing compatibility in order to match the suitable pairs of recipients and donors, which the one developed in [5] is not suited for.

TODO: Discuss the most important features.

### 7.2 Comparison to conventional models

TODO: Compare the model's performance to the Cox's Regression Model

### 7.3 Performance on different population samples

TODO: Try to locally differentiate ethnic groups within the UNOS database. How does the model perform on different population samples? Are different ethnicities more or less compatible?

### 7.4 Usage for Computing Compatibility

TODO: How this model could be used for choosing the most optimal donors. In what cases does the age bias matter and when not. (If we had more than one recipient and one (or more) donor(s) and we wanted to choose, which recipient should receive the donor's kidney, then the model would be biased to select the youngest recipient which could result in a case where there would be a donor that would be the best possible match for an older recipient, but a younger recipient would be selected since his survival curve would be the most favorable, even though there could be more suitable candidates for him.

However, this would not be an issue in the case where there is only 1 recipient and multiple donors. Then this model would be suitable for selecting the most compatible donor, by selecting the one, upon whose selection would the recipient have the best survival curve.



## **7.5 Possibilities for Future Research**

Handling the age bias + creating a model capable of picking the best match out of multitude of both donors and recipients.

# Conclusion

Text of the conclusion...

# Bibliography

- [1] S. J. Knechtle, L. P. Marson, P. J. Morris, *Kidney Transplantation: Principles and Practice*, Elsevier, 2020.
- [2] B. J. Nankivell, D. R. Kuypers. *Diagnosis and prevention of chronic kidney allograft loss*, The Lancet, Vol. 378, Issue 9800, 2011, 1428-1437.
- [3] S. Senanayake, N. White, N. Graves, H. Healy, K. Baboolal, S. Kularatna. *Machine learning in predicting graft failure following kidney transplantation: A systematic review of published predictive models*, International Journal of Medical Informatics, Vol. 130, 2019, 103957, <https://www.sciencedirect.com/science/article/pii/S1386505619302977>
- [4] D.D. Aufhauser Jr., et al. *Impact of prolonged dialysis prior to renal transplantation*. Clin Transplant, 2018 Jun, 32(6):e13260.
- [5] E. Mark, D. Goldsman, B. Gurbaxani, P. Keskinocak, J. Sokol. *Using machine learning and an ensemble of methods to predict kidney transplant survival*. PLoS ONE 14(1), 2019, e0209068, <https://doi.org/10.1371/journal.pone.0209068>
- [6] A. Burkov: *The Hundred-Page Machine Learning Book*. Andriy Burkov, 2019.
- [7] A. Géron: *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow 2nd Edition*. O'Reilly Media, 2019.
- [8] M. Mohri, A. Rostamizadeh, A. Talwalkar: *Foundations of Machine Learning*. MIT Press, 2018.
- [9] C. Strobl, A. Boulesteix, A. Zeileis, T. Hothorn, *Bias in random forest variable importance measures: illustrations, sources and a solution*. BMC Bioinformatics. 2007 Jan 25;8:25.
- [10] L. Breiman, *Random Forests*, Machine Learning, 45, 5-32, 2001
- [11] Chronic kidney disease: global dimension and perspectives
- [12] D. R. Cox, D. Oakes: *Analysis of Survival Data*. Chapman & Hall, 1984.
- [13] H. Ishwaran, U. B. Kogalur, E. H. Blackstone. M. S. Lauer: *Random survival forests*. Ann. Appl. Stat. 2 (3) 841 - 860, September 2008.
- [14] Google Developers, *Machine Learning Crash Course*, <https://developers.google.com/machine-learning/crash-course>
- [15] D. G. Kleinbaum, M. Klein, *Survival Analysis: A Self-Learning Text 3rd Edition*. Springer, 2012.

- [16] S. Pölsterl, *scikit-survival: A Library for Time-to-Event Analysis Built on Top of scikit-learn*. Journal of Machine Learning Research, vol. 21, no. 212, pp. 1–6, 2020.
- [17] Pedregosa et al., *Scikit-learn: Machine Learning in Python*, JMLR 12, pp. 2825-2830, 2011.
- [18] Github, *The state of open source software*. <https://octoverse.github.com/2022/top-programming-languages>
- [19] Python Software Foundation, *Python Documentation*, <https://docs.python.org/3/>
- [20] The Python Wiki, <https://wiki.python.org/moin/>
- [21] Pavel Strachota, *HELIOS cluster documentation*, <http://helios.fjfi.cvut.cz/>
- [22] Project Jupyter, *Jupyter Notebook Documentation*, <https://jupyter-notebook.readthedocs.io/en/latest/notebook.html>
- [23] The pandas development team, *Pandas*, <https://pandas.pydata.org/about/>
- [24] NumPy Developers, *NumPy*, [https://numpy.org/doc/stable/user/absolute\\_beginners.html](https://numpy.org/doc/stable/user/absolute_beginners.html)
- [25] Matplotlib Development Team, *Matplotlib*, <https://matplotlib.org/>
- [26] S. Krikov et. al., *Predicting Kidney Transplant Survival Using Tree-Based Modeling*. ASAIO Journal 53(5):p 592-600, September 2007. | DOI: 10.1097/MAT.0b013e318145b9f7
- [27] S. J.W. Willems, *Inverse Probability Censoring Weights for Routine Outcome Monitoring Data*
- [28] National Kidney Foundation, *A to Z Health Guide*, <https://www.kidney.org/atoz/content>
- [29] Jiang AT, BHSc, Rowe N, Sener A, Luke P. *Simultaneous pancreas-kidney transplantation: The role in the treatment of type 1 diabetes and end-stage renal disease*. Can Urol Assoc J. 2014 Mar;8(3-4)