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

České vysoké učení technické v Praze, Fakulta jaderná a fyzikálně inženýrská

Katedra: matematiky

Akademický rok: 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 histokompatiblního 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

B

....

garant oboru

Masáková

....

vedoucí katedry

....

děkan

Acknowledgment:
I would like to thank Ing. Tomáš Kouřim for his expert guidance and express my gratitude to Ing. Pavel Strachota, Ph.D. for his assistance with the formal aspects of this project.

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

Název práce:

Odhad kompatibility dárce a příjemce pro transplantaci ledvin pomocí strojového učení

Autor: Matěj Klouček

Obor: Matematické inženýrství

Zaměření: Matematická informatika

Druh práce: Bakalářská práce

Vedoucí práce: Ing. Tomáš Kouřim, Mild Blue, s.r.o.

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

Abstrakt: Se zvyšující se prevalencí onemocnění spojených s ledvinami je vylepšení procesu hledání vhodných párů dárců a příjemců zásadní pro zmírnění zátěže, kterou pro zdravotnický systém představují pacienti se selhávajícími ledvinami. Předchozí studie ukázaly, jak mohou metody strojového učení být použity pro predikování času přežití po transplantaci ledvin. V této studii je pro ohodnocení kompatibility mezi dárci a příjemci použit model náhodného přežívacího lesa, který predikuje dobu přežití na základě jejich pre-transplantačních metrik. Model je také použit pro zkoumání různých faktorů ovlivňujících přežití štěpu. Součástí studie je také podrobné prozkoumání algoritmů, které stojí za modelem náhodného přežívacího lesa, a analýzy přežití, která se běžně využívá v medicínském výzkumu.

Klíčová slova: strojové učení, rozhodovací strom, náhodný les, transplantace ledvin, analýza přežívání, náhodný přežívací les

Title:

Estimating Kidney Transplantation Donor-Recipient Compatibility Using Machine Learning

Author: Matěj Klouček

Abstract: With the increasing prevalence of kidney-related diseases, the improvement of the donor-recipient match-making process becomes crucial for alleviating some of the burden placed on the healthcare system by patients with failing kidneys. Several other studies have already demonstrated how machine learning methods could be used for predicting survival time after renal transplantation. In this study, the random survival forest model is used to evaluate compatibility between donors and recipients by predicting their survival time based on their pre-transplantation metrics. The model is then also used to investigate the various factors influencing graft survival. The study also includes an in-depth examination of the algorithms behind the random survival forests model, as well as survival analysis, a field of statistics commonly used in medical research.

Key words: Machine learning, Decision tree, Random forest, Renal transplantation, Survival analysis, Random survival forest

Contents

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 among other possible treatments [2, 3]. However, demand for transplants (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 due to the high cost and resources required for dialysis treatment. It is, therefore, crucial to minimize the time patients spend on the waiting list by optimizing the process of finding suitable donor-recipient pairings.

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 pairs, 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 a machine learning model that will evaluate the compatibility between a donor and a recipient by computing a risk score based on the patient's predicted survival and hazard functions, given that they receive a graft from that particular donor. This score can then be compared in order to select the most compatible donor-recipient pairing. 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 study 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. Within the domain of machine learning, computers develop algorithms and statistical models by learning from data without requiring explicit programming for their learning process.

The key idea behind machine learning is to create algorithms (models) capable of recognizing patterns and relationships within large datasets. These models are then utilized to make predictions or decisions using new, unseen data that was not part of their training process.

In general, machine learning is a great tool for solving problems that would conventionally require an infeasible amount of programming or when working with large amounts of data from which information is not easily extractable.

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 the desired output needs to be provided to the computer during the training process along with the data. Supervised learning can be further subdivided into *regression* and *classification* tasks based on whether the desired output is a continuous numerical value or a discrete one, respectively. Examples of supervised learning models include: *linear* and *logistic regression*, *decision trees*, *random forests*, *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 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. Typically, an unsupervised machine learning model is used first in order group similar examples and assign the unlabeled data a label based on the labeled example they were grouped with.

Then, a supervised machine learning model is used on this newly labeled data. However, models using the reverse order also exist. Using semi-supervised learning 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. Predictions about new data are then obtained by providing the newly introduced data as an input to the function. Model-based learning algorithms include *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 values, 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 either the features or the samples containing 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 otherwise 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 Section 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 feature with unique values $(x_1, \ldots x_n)$ into n columns as shown in Table 1.1. For each row, let y denote the original categorical value, i.e. $y \in (x_1, \ldots x_n)$ and let y_j denote the value of the encoded column $j \in \hat{n}$. Then

$$y_j = \begin{cases} 1 & \Longleftrightarrow y = x_j \\ 0 & \text{else} \end{cases} \tag{1.1}$$

Table 1.1: One-hot encoding

organ		organ_L_Kidney	organ_Pancreas	organ_R_Kidney
R_Kidney		0	0	1
L_Kidney	⟶	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 testing and validation each. However, with larger datasets, it is possible to allocate even higher percentage of the data to training.

Once the model has been trained 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 predictions that the model makes based on samples from the hold-out sets, i.e., samples it has not seen during the training process. Though not all models use generalization error for evaluation, for example, models focused on time-to-event predictions are usually evaluated using the C-index described in Section 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. its generalization error is low), but poorly on the hold-out sets, 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 phenomenon is called *overfitting* and is a common problem that arises during the process of developing a machine learning model. Overfitting is 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 the opposite problem, i.e., not being able learn the underlying patterns in the training data and thus being inaccurate on both the hold-out 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 Section 1.2.2, one of the most common causes for overfitting is having too many features included in the model. To solve this, only the features with the highest *feature importance* (i.e., contributing the most to the model's performance) are selected and used for training of the final model. This can prevent overfitting and thus increase the model's accuracy scores [17]. Moreover, this also greatly enhances the model's explainability [6] and speeds up the training process.

In the case of classification decision trees and random forests described in Section 1.3, feature importance is most commonly computed using the *mean decrease in impurity* (MDI) also known as *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. To address this issue, numerical features are often scaled to the $0 - 1$ range in a process called *normalization*.

A more reliable, though more computationally demanding, alternative is the *permutation importance* described in Section 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.q. $[0.01, 0.1, 1, 10, 100, 1000]$), while for categorical hyperparamaters, 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 benefit is 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 a *decision node*) a set of attributes of a feature vector is examined in order to make a split based on a given criteria. The specifics of this criteria are learned during the training process. For example, if a value of the given feature is below a specific threshold, the left branch is followed, otherwise the right branch is followed. The threshold is set to either maximize or minimize a certain performance metric of the model. Once the leaf node (also called *terminal node*) is reached, the example is assigned either a probability of belonging to a given categorical value in case of classification trees, or a numerical value in case of regression trees.

Different decision tree algorithms use different split rules also known as *criterions.* For example, one of the most common algorithms used for generating a classification tree is called ID3, which selects which 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 greater detail in Section 4, which use the log-rank test statistic in order to maximize the difference between the predicted survival in the child nodes.

1.3.1.1 Building a Classification Tree

A classification tree using the ID3 algorithm is built as follows [6]:
Let C=$\{1, 2, \ldots, p\}$ be the set of possible categorical labels, $\left\{\left(\mathbf{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{N}$ is a collection of labeled examples with numerical features, where N is the size of the collection, $\mathbf{x}^{(i)} = (x_1^{(i)}, \ldots, x_k^{(i)})$ is a n-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

$$f(\mathbf{x}, a) \overset{\text{def}}{=} \Pr(y = a \mid \mathbf{x}), \tag{1.2}$$

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

Let S_k denote a set of labeled examples belonging to node k, let $p^{S_i}(a)$ denote the proportion of the examples in S_k that belong to class a and let F_i denote the set of all possible features in a given node i (i.e. features that have not been used to make a split before).

In the first step of the algorithm, the decision tree consists only of its root node, which contains all of the labeled examples (i.e. $S_0 = \left\{\left(x^{(i)}, y^{(i)}\right)\right\}_{i=1}^{N}$), all features are available for splitting (i.e. $F_0 = \hat{n}$).

The proportion of the examples in S_k that belong to class a is given by

$$p^{S_k}(a) = \frac{\left|\left\{\left(\mathbf{x}^{(i)}, y^{(i)}\right) \in S_k \mid y^{(i)} = a\right\}\right|}{|S_k|}. \tag{1.3}$$

Next, the algorithm iterates over all possible features $j \in F_k$ and all possible threshold values $t_j \in T_j$ where T_j is the set of all possible threshold values for the feature j. The possible threshold values can be chosen, for example, as midpoints between observed values of the given feature, e.g. if $x_j \in [1, 2, 3]$ then the possible threshold values T_j may be chosen as $T_j = [1.5, 2.5]$. For each iteration (ordered pair $\left(j, t_j\right)$), S_k is split into two disjoint subsets defined as

$$S_{k-} \overset{\text{def}}{=} \left\{\left(\mathbf{x}^{(i)}, y^{(i)}\right) \in S_k \mid x_j^{(i)} < t_j\right\} \quad \text{and} \quad S_{0+} \overset{\text{def}}{=} \left\{\left(\mathbf{x}^{(i)}, y^{(i)}\right) \in S_k \mid x_j^{(i)} \geq t_j\right\}. \tag{1.4}$$

The quality of the split is then evaluated using the algorithm's criterion. Assuming that the tree's criterion is entropy, the entropy of a set of examples S_k belonging to node k is given by

$$H(S_k) = -\sum_{a \in S_k} p^{S_k}(a) \log_2 p^{S_k}(a). \tag{1.5}$$

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

$$H(S_{k-}, S_{k+}) = \frac{|S_{k-}|}{|S_k|} H(S_{k-}) + \frac{|S_{k+}|}{|S_k|} H(S_{k+}), \tag{1.6}$$

where S_{k-} and S_{k+} are disjoint subsets created by splitting S_k (note that $|S_{k-}| + |S_{k+}| = |S_k|$).

The best split (ordered pair $\left(\tilde{j}, \tilde{t_j}\right)$) is then given by minimizing the entropy of the split, i.e.,

$$\left(\tilde{j}, \tilde{t_j}\right) = \underset{j \in F, t_j \in T_j}{\arg\min} H(S_{k-}, S_{k+}). \tag{1.7}$$

Once the best split has been found, each of the created subsets then acts as a new decision node. The branching then continues with the exception that only the attributes that were not previously used are considered for splitting. In another words, given that $j_k \in F_k$ was chosen as the optimal feature to split upon, then only $F_k \setminus \{j_k\}$ are considered for splitting in the child nodes. The same logic applies to any subsequent splits.

The algorithm stops if there are either no further attributes to split upon, all possible decisions would reduce entropy less then a 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 **x**, with the same attributes as the feature vectors used in the training process, is introduced to the decision tree, the tree is followed from the root node down as follows. For each decision node, let j denote the feature that has been used for splitting the given node, and let t_j denote the threshold value learned during the training process for feature j. If $x_j < t_j$, the left branch is followed, else, the right branch is followed. This process repeats until a leaf node is reached. Let S_n denote the set of labeled examples belonging to this leaf node, then $\forall a \in C$

$$f(\mathbf{x}, a) = p^{S_n}(a) = \frac{\left|\left\{\left(\mathbf{x}^{(i)}, y^{(i)}\right) \in S_n \mid y^{(i)} = a\right\}\right|}{|S_n|}. \tag{1.8}$$

The example **x** is then classified as belonging to the class $c \in C$ if

$$c = \underset{a \in C}{\arg\max} f(\mathbf{x}, a) \tag{1.9}$$

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 features and labels with continuous numerical values. The Standard Deviation Reduction algorithm would be a good choice for building such a regression tree.

Let X_j be the set of all possible categorical values for the feature $j \in \hat{n}$,where n is the total number of features in the dataset. Again, let $\left\{\left(\mathbf{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{N}$ denote a collection of labeled examples, where N is the size of the collection, $\mathbf{x}^{(i)} = (x_1^{(i)}, \ldots, x_k^{(i)})$ is a k-dimensional feature vector of the example $i \in \hat{N}$, where $x_j^{(i)} \in X_j$, $\forall j \in \hat{k}$, and $y^{(i)} \in \mathbb{R}^+$ is the target value.

In this case, the decision tree will be represented a function $f(\mathbf{x}) \in \mathbb{R}^+$ that takes a n-dimensional feature vector, with the same type of attributes as $\mathbf{x}^{(i)}$, as input.

Let S_k denote the set of labeled examples included in node k, let F_k be the set of all possible features in the node k, let $\sigma(S_k)$ represent the standard deviation of labels in node k and let $\sigma'(S_k, j)$ be the standard deviation of labels in the node k for feature $j \in F_k$.

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

$$\sigma(S_0) = \sqrt{\frac{1}{N}\sum_{i=1}^{|S_0|}(y^{(i)} - \overline{y})^2}, \tag{1.10}$$

where $\overline{y}$ 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

$$\sigma'(S_0, j) = \sum_{c_j \in X_j} p\left(c_j\right)\sigma''\left(c_j\right) \tag{1.11}$$

where $p\left(c_j\right)$ is the proportion of examples in S_0 whose feature j belongs to the class $c_j \in X_j$, and $\sigma''\left(c_j\right)$ is the standard deviation of labels in S_0 whose feature j belongs to the class $c_j \in X_j$. $p(c)$ and $\sigma(c)$ are defined as follows.

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

$$p(c) = \frac{N_{c_j}}{|S_0|}. \tag{1.12}$$

Using the same notation for N_{c_j},$\sigma''(c)$ is given by

$$\sigma''(c) = \sqrt{\frac{1}{N_{c_j}}\sum_{i=1}^{N_{c_j}}\left(y_{c_j}^{(i)} - \overline{y}_{c_j}\right)^2} \tag{1.13}$$

where $y_{c_j}^{(i)}$ are the labels of samples whose feature j belongs to the class c_j and $\overline{y}_{c_j}$ is their mean value.

The best split is then given by maximizing the highest *standard deviation reduction* given by

$$\Delta\sigma(S_0, j) = \sigma(S_0) - \sigma'(S_0, j). \tag{1.14}$$

In another words, the node is split using feature $\tilde{j}$ given by

$$\tilde{j} = \arg\max_{j \in F_0} \Delta\sigma(S_0, j) \tag{1.15}$$

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

$$S_{c_j} = \left\{\left(x^{(i)}, y^{(i)}\right) \in S_0 \mid x_j^{(i)} = c_j\right\} \qquad \forall c_j \in X_j \tag{1.16}$$

where each S_{c_j} then acts 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 **x**, it follows the tree from the root down until a leaf node is reached. Let S_n be the set of examples belonging to such a leaf node. The tree's prediction is then given by the average value of the labels in S_n, i.e.

$$f(\mathbf{x}) = \frac{1}{|S_n|} \sum_{(\mathbf{x}^{(i)}, y^{(i)}) \in S_n} y^{(i)}. \tag{1.17}$$

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 is *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 different input features and different subsets of the training samples, which helps reduce overfitting and improves the generalization capabilities of the model.

1.3.2.1 Examples of Random Forests

A random forests model trained for a classification problem consists of an ensemble of classification trees. When making predictions based on an example it has not seen during the training process, the random forests model classifies the input as belonging to class *c*, if *c* got the highest number of *votes* by the individual classification trees [7]. A class *c* gets a vote by a classification tree when the input **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.

Similarly, a random forest built for a regression problem consists of an ensemble of regression trees. The prediction of the forest is given by taking the average of the predictions made by the individual regression trees, i.e.

$$\hat{f}(\mathbf{x}) = \frac{1}{N} \sum_{n=1}^{N} f_n(\mathbf{x}) \tag{1.18}$$

where **x** is an input feature vector, $\hat{f}$ is the forest's prediction, N is the total number of trees and f_n are the predictions made by the individual regression forests.

1.3.2.2 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 the training of multiple trees (*bagging*) or just once (*pasting*). Bagging is 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 features are independent of each other.

3. *Building the decision trees*: For each sample of the data and for each set of input features, a decision tree is built using a given split criterion. For even more randomized trees, it is possible to randomly select a 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 as shown in Section 1.3.2.1.

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

1.3.2.3 Weak and Strong Learners

The reason why random forests are effective, and therefore so popular, lies in the *law of large numbers*. This means that by combining predictions from multiple *weak learners*, each only slightly better than random guessing, the ensemble model achieves much higher accuracy (*strong learner*) [7, 10].

For example, consider a classification random forest as described in Section 1.3.2.1 that is designed to classify inputs into two categories. Suppose that the forest consists of 1000 classification trees whose predictions are correct only 51% of the time (weak learners), i.e. the probability that they will give their vote to the correct class is 51%. When the votes of all 1000 trees are combined, the probability that the correct class will receive more votes is 74.68%. This implies that the random forest will accurately categorize a newly introduced input with 74.68% success rate making it a fairly strong learner.

Chapter 2

Renal Transplantation

2.1 Chronic Kidney Disease

The *chronic kidney disease* (CKD) is a medical condition in which 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 *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 Current Strategies for Assessing Donor-Recipient Compatibility

There are a number of procedures that both the patient and a potential donor have to undergo before a transplantation. The results of these procedures are then used to assess the compatibility between the patient and the potential donor.

Firstly, their overall health is checked in order to evaluate whether they are a 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, such as history of diseases related to the functionality of kidneys. It is also necessary for the donor and the recipients to have compatible blood types.

Second, a process called *HLA typing* is performed both on the patient and the potential donor. 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 an immunologically 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 an adverse 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 observed 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 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]. Each locus has two alleles, one from each parent. 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. 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 used for computing mismatch. 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.

For example, in the first row of the Table 2.1, the HLA mismatch is 5 since both of the alleles at the HLA-B and HLA-DR locus differ, therefore, the HLA mismatch at the locus level is 2 for the HLA-B and HLA-DR loci. At the HLA-A locus, both the donor and the recipient have one identical allele (2), therefore, the mismatch at the HLA-A locus is only 1. In total, the HLA mismatch is 1 + 2 + 2 = 5.

Table 2.1: HLA Mismatch

donor's HLA typing						recipient's HLA typing						mismatch at locus level			total mismatch
DA1	DA2	DB1	DB2	DDR1	DDR2	RA1	RA2	RB1	RB2	RDR1	RDR2	AMIS	BMIS	DRMIS	HLAMIS
2	31	18	38	11	13	2	97	44	97	1	4	1	2	2	5
3	68	7	62	4	15	1	3	52	62	4	15	1	1	0	2
26	29	53	70	11	15	23	24	38	53	14	15	2	1	1	4
1	29	8	44	3	7	1	29	8	44	3	7	0	0	0	0
23	68	44	48	1	8	2	24	52	35	2	8	2	2	1	5

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 affecting the long-term performance of the graft, that are not being accounted for in the current donor-recipient matchmaking process. The progress 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 still 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, in 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 a 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 is not limited to medical applications, as it is often used, for instance, to model reliability in engineering, studying the time until the failure of mechanical system occurs.

Survival analysis involves several statistical techniques that are commonly used in medical research, 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, with the former two also being a vital part of the algorithm behind random survival forests described in Section 4.

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. Censoring may happen, for example, when a study ends before an 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 is also possible, but is uncommon in the case of post-transplantation data as the 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 [15]

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

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 instantaneous potential per unit time (usually days) 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., the 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 potential is *instantaneous*, it is given by the following limit [15]

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

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 survival models are described, since it provides a more direct measure of risk over time. Importantly, either of the functions can be derived from the other one using one of the following formulae [15]

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

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

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. It is calculated as the ratio of the hazard functions of the two groups:

$$\mathrm{HR}(t) = \frac{h_1(t)}{h_2(t)}. \tag{3.5}$$

A hazard ratio of 1 means that the probability of an event occurring at a given time is the same in both groups. For example, if the two groups are divided by their 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 for estimating survival functions is the *Kaplan-Meier estimator,* which works as follows [15]. Suppose we have a dataset containing n entries with each entry i containing information about the individual's survival time T_i and censoring status δ_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 at time T_i. Let τ denote the set of all unique event times, i.e $\tau := \{T_i \mid \delta_i = 1,\ i = 0, \ldots, n\}$, let f_j denote the 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., patients who have yet to experience an event or are yet to be censored). The probability of an individual surviving past $t_j \in \tau$, given that they have already survived until at least t_j, is then given by the conditional probability $\Pr(T > t_j \mid T \geq t_j)$ which is itself given by

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

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 outputs the probability of an individual 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 $t_1, \ldots, t_{|\tau|}$ as follows [15]

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

Due 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 a 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 Section 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.

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

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

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. The log-rank statistic for the group i is then given by

$$L^{(i)} = \frac{(O_i - E_i)^2}{\mathrm{Var}(O_i - E_i)}. \tag{3.9}$$

Note that for two groups: $O_1 - E_1 = -(O_2 - E_2)$.

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 for predicting survival curves of patients who have undergone kidney transplantation and for identifying factors that impact the likelihood of graft failure. The Cox regression model will also be used later as a benchmark to compare the performance of the machine learning model with a conventional survival analysis model.

3.2.3.1 Formula

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

$$h(t \mid \mathbf{x}) = h_0(t) \exp\left(\sum_{i=1}^{k} \beta_i x_i\right) \tag{3.10}$$

where $\mathbf{x} = (x_1, x_2, \dots x_n)$ is a n-dimensional feature vector, $h_0(t)$ is an unspecified *baseline hazard function,* and $\beta = (\beta_1, \beta_2, \dots, \beta_n)$ represents constant coefficients associated with corresponding features. 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 the correct parametric model [15].

A core assumption behind the Cox proportional hazards model, and thus by extension equation (3.10), is that the baseline hazard function is dependent only on time, whereas the second part of the expression $e^{\sum_{i=1}^{k} \beta_i X_i}$ is only dependent on $\mathbf{x}$, which means the features are time-independent. This assumption is called the *proportional hazards* assumption. If time-dependent features 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 β coefficients.

3.2.3.2 Partial Maximum Likelihood Estimation

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

$$\hat{h}(t \mid \mathbf{x}) = \hat{h_0}(t) \exp\left(\sum_{i=1}^{k} \hat{\beta}_i x_i\right) \tag{3.11}$$

Given that we have a random sample of N individuals $\left\{\left(\mathbf{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{N}$, where $\mathbf{x}^{(i)}$ are n-dimensional feature vectors describing the given individuals, and $y^{(i)} = \left(T^{(i)}, \delta^{(i)}\right)$ are ordered pairs, where $T^{(i)}$ and $\delta^{(i)}$ are the observed survival time and censoring status respectively. Assuming that the samples come from an unknown joint probability distribution, the maximum likelihood estimation for β is given by

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

where L is the maximum likelihood function. The maximum likelihood function is conventionally defined as

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

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] as defined in (3.13).

Therefore, a *partial likelihood* is used, which only considers probabilities of the subjects who fail, i.e. with censoring status $\delta^{(i)} = 1$. This means that the partial likelihood function can be written as

$$L = \prod_{j=1}^{p} L_j \tag{3.14}$$

where p is the number of failures 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 $R(t_j)$. Note that $R\left(t_j\right)$ represents a *set* of individuals at risk compared to r_j used before, which only represents the number of individuals at risk. This distinction in notation is made because it simplifies the equation (3.15). L_j is defined as a 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.

$$L_j(\beta) = \frac{h_0(t)\exp\left(\beta \cdot \mathbf{x}^{(j)}\right)}{\sum_{i \in R(t_j)} h_0(t)\exp\left(\beta \cdot \mathbf{x}^{(i)}\right)} \tag{3.15}$$

where $\mathbf{x}^{(j)}$ denotes the feature vector 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. Note that $\beta \cdot \mathbf{x}^{(j)}$ is a dot product. It is also important to note that in contrast to the notation used for Kaplan-Meier estimator, p represents the number of *all* events, not just the number of unique event times. This ensures that there is only one hazard function in the numerator of the (3.15) equation.

As $h_0(t)$ cancels out, the partial likelihood function can be written as

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

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

$$\frac{\partial \ln L}{\partial \beta_i} = 0 \qquad \text{for } i = 1, \ldots, k \tag{3.17}$$

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

3.2.3.3 Hazard Ratio Estimation

As described in Section 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

$$\hat{HR} = \frac{\hat{h}(t \mid \mathbf{x}^*)}{\hat{h}(t \mid \mathbf{x})} \tag{3.18}$$

where $\mathbf{x}^* = \left(x_1^*, x_2^*, \ldots, x_n^*\right)$ and $\mathbf{x} = (x_1, x_2, \ldots x_n)$ are feature vectors 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

$$\hat{HR} = \frac{\hat{h_0}(t)\exp\left(\sum_{i=1}^{k}\hat{\beta}_i x_i^*\right)}{\hat{h_0}(t)\exp\left(\sum_{i=1}^{k}\hat{\beta}_i x_i\right)} = \exp\left(\sum_{i=1}^{k}\hat{\beta}_i\left(x_i^* - x_i\right)\right) \tag{3.19}$$

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 feature j on the overall hazard while adjusting for other variables, let vectors $\mathbf{x}^*$ and $\mathbf{x}$ be defined as

$$\begin{cases} x_i^* = x_i & \text{for } i \neq j \\ x_j^* \neq x_j \end{cases} \tag{3.20}$$

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

$$\hat{HR} = \exp\left(\hat{\beta}_j\left(x_j^* - x_j\right)\right) \tag{3.21}$$

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.

Chapter 4

Random Survival Forests

Random survival forests (RSF) are an ensemble machine learning method used for analyzing right-censored survival data. RSF combines 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. It also adresses 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].

The building blocks of RSF are *survival trees*, which are a modification of decision trees (described in Section 1.3.1) designed to handle survival data. Each tree in the forest outputs a either a *cumulative hazard function* $H(t \mid \mathbf{x})$ or a *survival function* $S(t \mid \mathbf{x})$. Both take a vector of predictive variables $\mathbf{x}$, describing a given individual, and time t as inputs. The forest's overall prediction, i.e., the *ensemble cumulative hazard function* $H_e(t \mid \mathbf{x})$ or the *ensemble survival function* $S_e(t \mid \mathbf{x})$*,* is then computed as the average of the individual tree's predictions.

4.1 Building a Random Survival Forests Model

The algorithm for building a RSF model as implemented in the works as follows [13, 16].

1. A set number of subsets is taken from the training dataset using the bootstrap sampling method as described in Section 1, with each sample excluding, on average, 37% of the original data.

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 Section 1.3.1 , i.e., at each node the algorithm iterates over all possible features (the number of which can be limited) and values for them, and decides on the best split. The best split is decided by maximizing the log-rank statistic (described in Section 3.2.2), which compares the survival curves 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. This leads to a point where no new splits can be made, meaning that the tree has been fully grown. Other hyperparameters limiting the growth of the tree can also be set, which are described in Section 4.2.

3. Once a tree is fully grown, it outputs either a survival function or a cumulative hazard function (CHF), which are calculated as follows. Let A denote the set of all terminal nodes.

   (a) If the preferred output is a survival function, for every terminal node, that is $\forall j \in A$, a survival curve $\hat{S}_j(t)$ is computed using the Kaplan-Meier estimator in the same manner as described in Section 3.2.1, with the individuals from the given bootstrap sample belonging to the given terminal node j being used as the group on which the Kaplan-Meier survival curve is calculated on.

(b) Otherwise, for every terminal node $j \in A$, a CHF denoted by $\hat{H}_j(t)$ is calculated as follows. Suppose that each node $j \in A$ contains a set of individuals $\left(T_{1,j}, \delta_{1,j}\right), \ldots, \left(T_{n(j),j}, \delta_{n(j),j}\right)$, 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 $T_{i,j}$. Let $t_{1,j} < t_{2,j} < \ldots < t_{N(j),j}$ denote the unique event times observed in node j, where $N(j)$ is the total number of observed unique event times, and let $f_{k,j}$ and $r_{k,j}$ represent the number of individuals who either experienced an event (failed) or were at risk at time $t_{k,j}$ respectively. The CHF for the node j is then given by the Nelson-Maier estimator

$$\hat{H}_j(t) = \sum_{t_{k,j} \leq t} \frac{r_{k,j}}{f_{k,j}}. \tag{4.1}$$

4. When the RSF model is asked to make predictions for a feature vector $\mathbf{x}$, each survival tree is followed from the root node down in accordance with the rules it learned during the training process as described in Section 1.3.1.1, until a terminal node $i \in A$ is reached.

   (a) The survival function of the whole tree is then given by

$$S(t \mid \mathbf{x}) = \hat{S}_i(t), \tag{4.2}$$

   (b) Alternatively, the CHF of the tree is given by

$$H(t \mid \mathbf{x}) = \hat{H}_i(t). \tag{4.3}$$

5. Next, both the ensemble survival function $S_e(t \mid \mathbf{x})$ and the ensemble cumulative hazard function $H_e(t \mid \mathbf{x})$ are computed in a similar manner by averaging over the predictions made by all grown tress. More specifically, let N denote the number of survival trees in the ensemble.

   (a) The survival function is then given by

$$S_e(t \mid \mathbf{x}) = \frac{1}{N} \sum_{n=1}^{N} S_n(t \mid \mathbf{x}) \tag{4.4}$$

   where S_n is a survival function predicted by the n-th tree in the ensemble (defined in (4.2)).

   (b) Analogously, the ensemble cumulative hazard function is given by

$$H_e(t \mid \mathbf{x}) = \frac{1}{N} \sum_{n=1}^{N} H_n(t \mid \mathbf{x}) \tag{4.5}$$

   where H_n is a cumulative hazard function predicted by the n-th tree in the ensemble (defined in (4.3)).

4.2 Hyperparameters

Like any other machine learning model, the RSF model has multiple hyperparameters that define its behavior. As RSF are derived from conventional random forests, they also share many of the same hyperparameters. The most important hyperparameters that the scikit-survival's (see Section 5.2.3) implementation of RSF allows to be set are [16]:

- `n_estimators`: The number of survival trees in the forest. More trees generally result in better predictions, however, they also lead to longer training times, and the benefits gained by adding more trees diminish, reaching a point where the added computational complexity is not worth it [17].
- `max_depth`: The maximum allowed depth of a tree. If none is set, then tree is allowed to grow until all leaves contain less than `min_samples_split` [17].
- `min_samples_split`: The minimum number of samples required to split a node. As mentioned before, this parameter can limit the growth of survival trees and therefore speed up the training process. However, setting this parameter too high can result in underfitting as the model may fail to capture more complex relations in the data.
- `min_samples_leaf`: The minimum number of samples required to be in a leaf node. This will ensure more samples are used for calculating the survival / cumulative hazard functions. Therefore, raising this number will smooth the predictions [16]. However, setting this parameter too high may worsen the model's ability to make accurate predictions. If none is set, the default value is 1.
- `max_features`: The maximum number of features to be considered when looking for a best split. Having only a random subset of the features available when looking for the best split may help to differentiate the individual trees and reduce overfitting. However, setting this number too low may again reduce the accuracy of the model.

4.3 Harrell's Concordance Index

The prediction error of a random survival forests model is estimated using the *Harrell's concordance index* (also known as *C-index*) [13], which is a 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, the ROC curve plots the *true positive rate* (TPR) defined as

$$TPR = \frac{TP}{TP + FN} \tag{4.6}$$

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

$$FPR = \frac{FP}{FP + TN} \tag{4.7}$$

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 (AUC) 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 AUC, as well as 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 an AUC of 0.5. Generally, models with an AUC 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 (described in Section 1.3.1) and by extension random forests used for classification .

The C-index works in similar fashion to the AUC in the sense that it compares two individuals, and checks whether the model assigns a more favorable hazard function to the patient with longer observed survival time. More specifically, let $t_1, \dots t_n$ be a set of pre-chosen time points, then, we say that individual i has a *worse predicted outcome* than individual j if

$$\sum_{k=1}^{n} H_e(t_k \mid \mathbf{x}_i) > \sum_{k=1}^{n} H_e(t_k \mid \mathbf{x}_j) \tag{4.8}$$

where $\mathbf{x}_i$ and $\mathbf{x}_j$ are feature vectors describing the individuals i and j respectively, and H_e is the ensemble CHF defined in (4.5). However, not all individuals can be compared to each other due to censoring. Therefore, there is specific set of rules that define comparable pairs, which are explained in the next section.

4.3.1 Calculating Harrell's Concordance Index

The C-index is calculated as follows [13].

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

   - pairs $\big((T_i, \delta_i), (T_j, \delta_j)\big)$ where the case with the shorter survival time is right-censored, i.e., $T_i < T_j \ \wedge \ \delta_i = 0$
   - pairs $\big((T_i, \delta_i), (T_j, \delta_j)\big)$ 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 $\big((T_i, \delta_i), (T_j, \delta_j)\big)$, a value for $c_{i,j}$ is assigned as follows:

   (a) if $T_i \neq T_j$ and, without the loss of generality, suppose that $T_i < T_j$

   $$c_{i,j} = \begin{cases} 1 & \text{if } i \text{ has worse predicted outcome than } j \text{ (defined in (4.8))} \\ 0.5 & \text{if the predicted outcomes of } i \text{ and } j \text{ are tied (equality in (4.8))} \\ 0 & \text{if } j \text{ has worse predicted outcome than } i \end{cases}$$

   (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 $T_i = T_j \wedge \delta_i \neq \delta_j$ and, without the loss of generality, suppose that $\delta_i = 1$

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

3. The C-index is then given by

$$C = \frac{1}{|P|} \sum_{i,j \in P, i \neq j} c_{i,j} \tag{4.9}$$

where $|P|$ denotes the number of permissible pairs.

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], is used to compute variable importance. Fortunately, permutation importance is also implemented within the scikit-learn 5.2.3 package.

Permutation importance is computed by first calculating a baseline metric by evaluating the model's performance when trained on the complete set of features. Next, a feature is permutated from the validation set, and the model's performance is evaluated again. The permutation importance of a given variable is then given by the 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 (see Section 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 making predictions on 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 transferable 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 (described in Section 1.3.2.3), and sequentially improving their performance by gradually optimizing a loss function by using its gradient. Gradient boosting models generally achieve greater accuracy than random survival forests, but take much longer to train [6, 16].

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

The reasons for why the RSF model was chosen include the above-described disadvantages of the aforementioned alternatives and the 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 out between 1984 and 2022. 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 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 for visualization of the results. The following sections offer a brief introduction to each of the aforementioned software tools and libraries.

5.2.1 Python

Python is an interpreted, general-purpose, high-level, non-typed, 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 should in theory 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 knows 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 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 behavior of MATLAB for Python [25].

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 is far beyond the capacity of regular computers. 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 a sufficient for the needs of this project. The connection to the cluster was established using SSH in the Linux shell. To enhance practicality, remote nodes were forwarded onto localhost, which allowed working with Jupyter notebooks directly via a web browser on a local machine. More information about the cluster such as regarding the hardware specs can be found at [21].

Chapter 6

Model Training

The code used for training the machine learning model, including pipelines used for data preprocessing, training, and evaluating the random survival forest model are publicly available in the form of Jupyter Notebooks at: `https://github.com/matejkloucek/bp_don_rec_compatibility` .

Instructions 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 Section 5.1, but only 3 are relevant to this study. These 3 are:

- `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`, which contains detailed information about living donors.
- `Deceased_Donor`, which contains detailed information about deceased donors

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 address this limitation, a table join was performed with the `Living_Donor` and `Deceased_Donor` tables to provide more information about the donors. This joining was performed on the `DONOR_ID` column, which is a foreign key in the `Kidney_Pancreas` table that identifies the graft's donor for the given transplantation, and refers to a primary key in either the `Living_Donor` or the `Deceased_Donor` table. There were 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 results gained by experimenting with the RSF models have revealed that the features impacting graft survival vary between living donor and deceased donor transplantations. Consequently, this paper will focus solely on transplantations from a living donor. As a result, dataset has been split based on the type of transplantation performed, separating living and deceased donor transplantations. The subset resulting from this split containing information about the living donor transplantation will from now on be referred to as `Kidpan_Living`. The `Kidpan_Living` table had 174381 entries and 649 columns. This number of columns is usually considered to be too 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) [27]. In fact, most (83%) of pancreas transplantation are performed in the context of SKP's [28]. However, the focus of this paper is solely on kidney transplantations and the number of SKPs is relatively low compared to the number of standalone kidney transplantation. Therefore, individuals who had undergone a SKP were dropped from the dataset. 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.

Note for following chapters: Owing to the nature of the `GTIME_KI` and `GSTATUS_KI` columns, the survival times computed and predicted in the following sections and chapters describe the survival time of the graft itself, not necessarily the survival time of the patient, as the graft may fail but the patient may continue to live on either by returning to dialysis or by getting another transplant. The opposite may also happen, i.e., patient dies with a functioning kidney. Out of the total 174381 entries in the `Kidpan_Living` table, 106910 were censored while still alive, 40714 experienced graft failure, and 26732 patients died with a functioning graft. In terms of graft survival, the death of a patient with a functioning graft is considered a censoring event, given that the patient died of causes unrelated to kidneys. Nevertheless, for the sake of simplicity, the survival time of the graft will sometimes be referred to as the patient's or recipient's survival time.

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. This was done by 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 STAR dictionary was also used to check that no obviously relevant columns were lost in the process.

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.

In addition, as discussed in Section 2.2.1, the value of the HLA mismatch column is a sum of the mismatches at the A, B, and DR loci. Including both the overall HLA mismatch and the mismatch at the individual loci introduces multicollinearity to the model, thereby potentially compromising its accuracy. Therefore, the model was first trained using only the mismatches at the locus level, with the aim of discovering whether any of the locus-level mismatches had a larger impact on survival time than others. The results revealed that the mismatch at the HLA-DR locus had the biggest impact on survival, while the mismatch at the HLA-B locus had the smallest impact. Nevertheless, the accuracy of the model did not significantly differ between the case where only the overall HLA mismatch was used, and the case where only the individual locus-level mismatches were used. Subsequently, only the overall HLA mismatch was used for the training of the final model in order to simplify it.

Overall, these measures reduced the number of features to 136.

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 of the RSF model. Scikit-learn's `OneHotEncoder` was used to encode the categorical variables. After merging the encoded categorical columns with the numerical columns, the resulting dataset had 810 columns.

6.2 Feature Selection

In order to simplify the model and improve its performance, the best features were selected by calculating their importance using permutation importance described in Section 4.4. In comparison to variable importance calculated by the Gini importance (described in Section 1.2.3), computing permutation importance is a highly time consuming, as well as resource demanding task.

The scikit-learn's implementation of permutation importance allows computing of permutation importance to be done in parallel on multiple cores. However, due to the specifics of the implementation of random survival forests in the scikit-survival's package, this was extremely memory demanding and couldn't be completed even using the Helios cluster. Therefore, the computations were performed using only a single core, which resulted in computation times of up to 10 hours.

The scikit-learn's implementation of permutation importance offers a `n_repeats` parameter, which controls the number of times a feature is permutated 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 a 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=2` . The two resulting tables of variable importances were used to create a list of the 27 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=15`.

The selected features along with their description and variable importance can be found in Table 6.1. For categorical features, values of their most important encoded sub-feature are listed in this table.

Table 6.1: Variable Importance

Feature	Mean Importance	St. Dev. Importance	Description	Data type
AGE	0.025524	0.005794	Recipient's age	Numerical
AGEDON	0.015870	0.002372	Donor's age	Numerical
HLAMIS	0.014202	0.001809	overall HLA mismatch	Numerical
END_BMI_CALC	0.012784	0.001401	Recipient's BMI	Numerical
DRMIS	0.010725	0.001533	Mismatch at the HLA-DR locus	Numerical
ON_DIALYSIS	0.010116	0.001832	Is recipient regularly administered dialysis	Categorical
BMIS	0.009605	0.001134	Mismatch at the HLA-B locus	Numerical
DIAB	0.009528	0.002777	Recipient's diabetes status	Categorical
AMIS	0.009310	0.000711	Mismatch at the HLA-A locus	Numerical
ETHCAT	0.008813	0.003179	Recipient's ethnicity category	Categorical
PRE_TX_TXFUS	0.008088	0.001182	Did recipient receive transfusion	Categorical
PERIP_VASC	0.007778	0.000708	Recipient's peripheral vascular disease status	Categorical
DAYSWAIT_CHRON_KI	0.007472	0.000778	Days recipient spent on waiting list	Numerical
ETHCAT_DON	0.006864	0.000557	Donor's ethnicity category	Categorical
PRI_PAYMENT_TCR_KI	0.006821	0.000796	Recipient's primary payment source	Categorical
FUNC_STAT_TCR	0.006570	0.000299	Recipient's functional status	Categorical
TOT_SERUM_ALBUM	0.006915	0.001306	Recipient's total serum albumin	Numerical
HIST_CIG	0.006426	0.000307	Recipient's history of cigarette use	Categorical
HCV_SEROSTATUS	0.006025	0.000423	Recipient's hepatitis C status	Categorical
EBV_SEROSTATUS	0.006014	0.000245	Recipient's status of Epstein-Barr virus	Categorical
DIABETES_DON	0.005850	0.000000	Donor's diabetes status	Categorical
WORK_INCOME_TCR	0.005793	0.000187	Is recipient working for income	Categorical
DAYSWAIT_ALLOC	0.005418	0.000838	Days recipient was prioritised for allocation	Numerical
PREV_TX_ANY	0.005222	0.001113	Recipient's past transplantations	Categorical
KI_CREAT_PREOP	0.004947	0.000517	Donor's creatinine level	Numerical

After one-hot encoding the categrocial features in this reduced dataset, the final number of columns before training was 111.

6.2.1 Discussion of Features

As can be seen in the Table 6.1, the graft's survival time is significantly influenced by recipient age, followed by the donor's age. This finding aligns with expectations, as age often plays a pivotal role in various medical conditions. Similarly, HLA mismatch (described in Section 2.2.1]) emerges as a crucial determinant of graft survival time.

In addition to age and HLA mismatch, variables reflecting the recipient's overall health, such as BMI and functional status (ability to perform daily activities), are prominent features in determining graft survival time. Medical histories of both the recipient and donor, including conditions like diabetes and hepatitis, also play a role.

Unexpectedly, certain columns, such as `PRI_PAYMENT_TCR_KI` and `WORK_INCOME_TCR`, reflecting the patient's financial status, are included among the most influential factors. A possible explanation for this could be that patients with greater financial security can access better medical care, thus increasing their chances of survival. Additionally, these patients may afford healthier lifestyle choices, such as improved diet and exercise.

6.3 Hyperparameter Tuning

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

After experimenting with different combinations of hyperparameters using grid search (described in Section 1.2.4), the optimal parameters were found to be:

`n_estimators=200`

`min_samples_split=10`

`min_samples_leaf=15`

`max_depth=None`.

Using these hyperparameters and the reduced list of features, the RSF model achieved an accuracy given by the C-index of 0.638.

6.4 Model's Capabilities

Once trained, the model can be used to predict survival functions, as well as cumulative hazard functions of allografts based on the information regarding 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

$$R(\mathbf{x}) = \sum_{j=1}^{n} H_e\left(t_j \mid \mathbf{x}\right) \tag{6.1}$$

where $\mathbf{x}$ is an input feature vector, t_i represents the unique event times in the training data, n is the total number of unique event times in the training data, and $H_e(t \mid \mathbf{x})$ is the ensemble cumulative hazard function defined in (4.3) [16]. This risk score could be a good way to quantify the compatibility between a potential donor-recipient pairing.

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

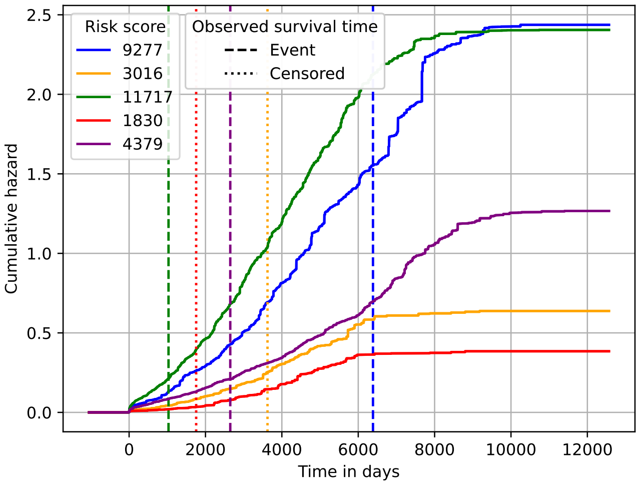

Figure 6.2: Predicted cumulative hazard functions

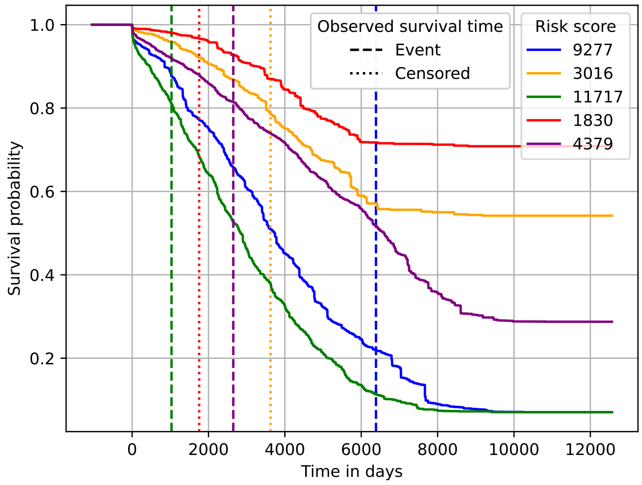

Figure 6.1: Predicted survival functions

6.4.1 Model's Limitation

With increasing time, the predicted survival curves eventually flatten, which is an inaccuracy of the model as survival curves cannot stay constant forever and must eventually decline to zero.

The plateauing of both of the curves is caused by the limited amount of samples belonging to the individual terminal nodes of the survival trees. More specifically, this limitation is caused by the high amount of censoring in the dataset with 55% in the complete `Kidney_Pancreas` dataset and 61% in the `Kipan_Living` dataset. This results in situations where many censored individuals are grouped in a terminal node while their censoring time extends beyond the last unique event time in the given terminal node. Consequently, the computed survival functions, using the Kaplan-Meier estimator in each survival tree, maintain a constant nonzero value beyond the last unique event time.

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 0.724. 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 prospects, thus giving their model a significant boost in accuracy. However, as mentioned before, the goal of this study 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 and therefore should only use information that is knows prior to the transplantation. Thus, the model developed in [5], while more accurate in predicting survival time, is not suitable for such a task.

7.1.1 Important Features

As discussed in Section 6.2.1, the age of the recipient is by far the biggest factor influencing the graft's survival time. The impact of the recipient's age on survival time can also be validated using the Kaplan-Meier estimator (described in Section 3.2.1). In Figure 7.1, survival curves were computed by selecting 1000 random samples from the test for each age group 0-14, 15-29, 30-44, 45-59, 60+. Next, survival curves of these cohorts were computed with the Kaplan-Meier estimator using observed survival time and censoring status.

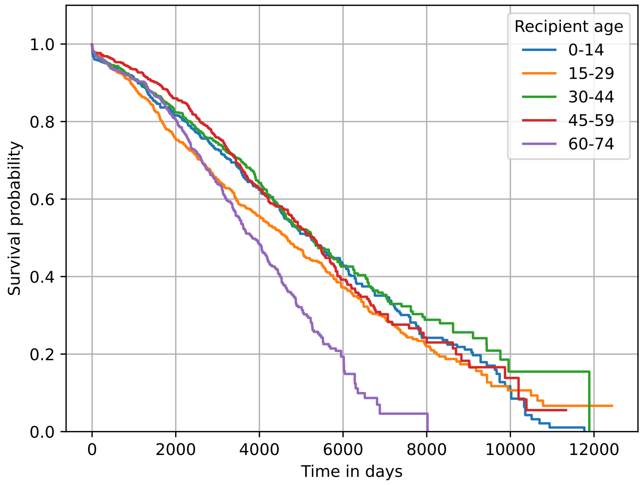

Figure 7.1: Kaplan-Meier survival curves by age group

From observation, age exhibits little influence on the graft's survival in the age groups 0-59 and only begins to have a significant impact in the 60+ age cohort. This trend can also be observed on the predictions made by the RSF model. Plotted in Figure 7.2 are averages of 1000 survival functions predicted for each of the age cohort using the same samples as for computing the survival curve in Figure 7.1.

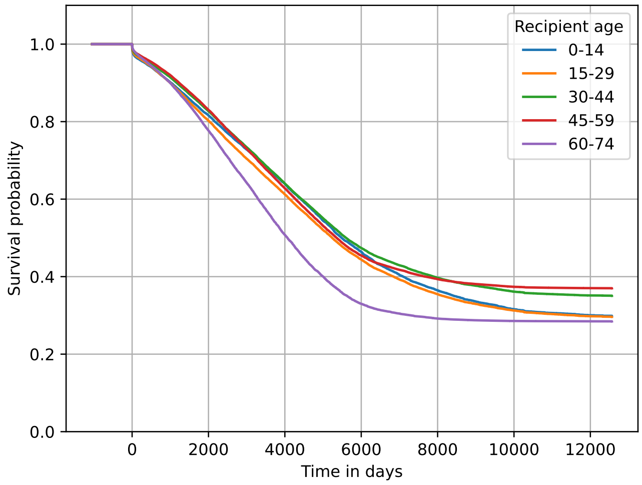

Figure 7.2: Averaged RSF predictions by age

Another feature with a significant variable importance is HLA mismatch (described in Section 2.2.1). By grouping samples by the level of their HLA mismatch and computing their survival curves using the Kaplan-Meier estimator. It can be observed in Figure 7.3 that recipients with zero HLA mismatches have more optimistic survival curves than others. It can also be observed that survival curves of recipients with few (but more than one) mismatches do not differ significantly from the survival curves of recipients with higher mismatch levels. Also visible in Figure 7.3, the distance between the survival curves of patients with some HLA mismatches and the ones without, seems to grow larger with time suggesting that HLA mismatch level is important not only for preventing acute rejections, but also for long term survival.

It is also important to note that zero HLA mismatch level is in practice mostly found only between identical twins as the chances of finding a perfectly matched unrelated (or related, but not identical twin) donor are extremely low. This is because HLA genes are inherited from parents and the number of different alleles is very high. This would therefore suggest that patients receiving a graft from their identical twin have a significantly higher chances of survival, while HLA mismatch does not seem to play a big factor for unrelated donor-recipient pairings as these, in practice, attain HLA mismatch greater than zero.

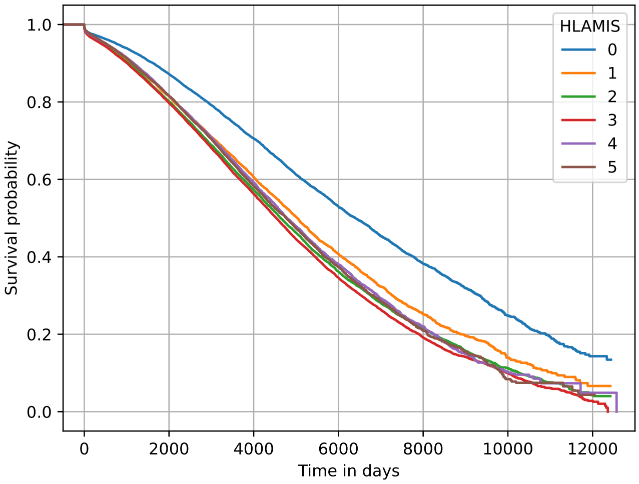

Figure 7.3: Kaplan-Meier survival curves by HLA mismatch

This also holds true for the predictions made by the RSF model (see Figure 7.4), which were computed by grouping the entries from the test set by their HLA mismatch, and then averaging the predictions made for each of the subsets. Note that the model's predictions start to deviate from the observed survival curves in Figure 7.3 around the 6000 days mark, which is a result of the plateauing effect described in Section 6.4.1.

7.1.2 Comparison to Conventional Models

The Cox Regression model (described in Section 3.2.3) fitted onto the same dataset had a C-index of 0.619, meaning that the random forests model clearly outperforms Cox regression when making predictions based on the features listed in Table 6.1. Though it is important to note, that the Cox regression model took a significantly less amount of time to fit and had considerably lesser memory requirements in comparison to the RSF model. This prompts the question as to whether such added computational complexity is worth the improved accuracy.

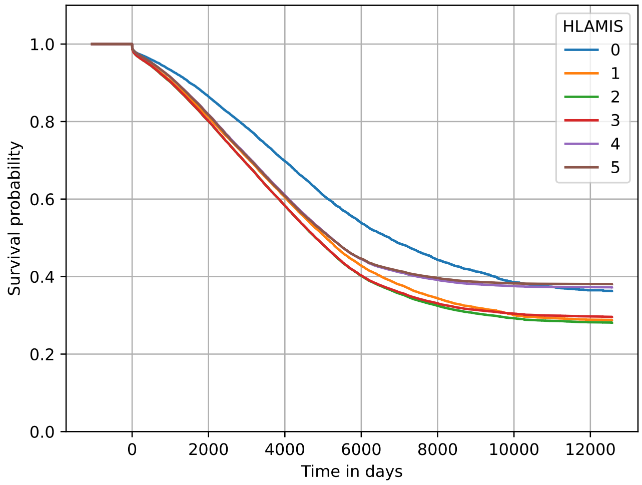

Figure 7.4: Averaged RSF predictions by HLA mismatch

7.2 Performance on different population samples

In order to investigate the influence of ethnic background on the outcome of a kidney transplantation, the dataset has been grouped by the ETHCAT and ETHCAT_DON categories representing the recipient's and the donor's ethnic categories respectively. Within the UNOS dataset, individuals are divided into 7 categories: White, Black, Hispanic, Asian, Native American, Pacific Islander and Multiracial. Consequently, the dataset was divided into 49 subsets. To ensure adequate sample sizes, only 10 subsets with the most entries were selected for further investigation. The selected subsets are listed in Table 7.1.

Table 7.1: Ethnicities of recipients and donors

ETHCAT	ETHCAT_DON	N. of entries
white	white	111515
hispanic	hispanic	19054
black	black	18308
asian	asian	4323
hispanic	white	4169
black	white	3989
white	hispanic	3045
asian	white	1866
white	black	1045
white	asian	751

Plotting the calculated Kaplan-Meier survival curves on a graph (see Figure 7.5) reveals that there are slight differences between the different groups. Namely, pairings where at least one from the donor-recipient pairing was Asian, had a more favorable survival curve. On the other hand, pairings where either the donor or the recipient were Black, had a slightly less favorable survival curve than the others.

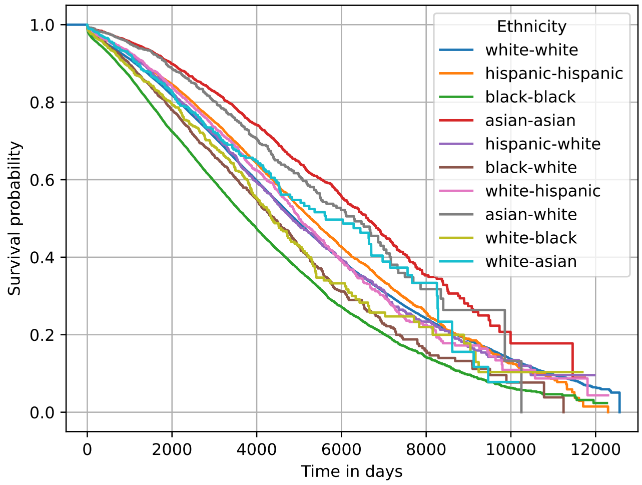

Figure 7.5: Kaplan-Meier survival curves by ethnicity

This is even more visible when focusing solely on the pairings where the recipient was white (see Figure 7.6). Also in Figure 7.6, it can be observed that transplantations between different ethnic categories in general perform no worse or better than transplantations between the same ethnic group.

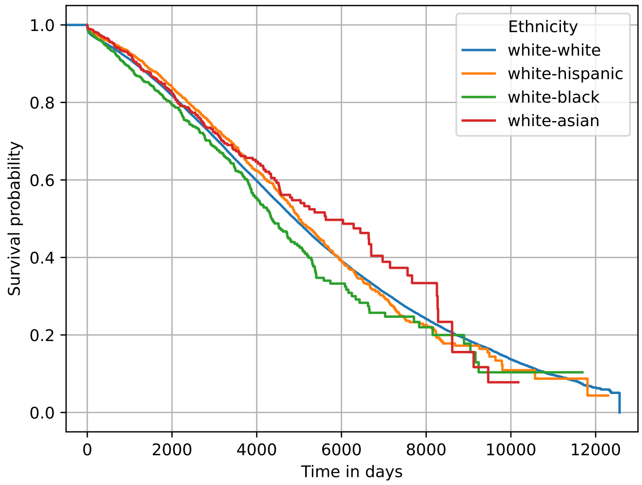

Figure 7.6: Kaplan-Meier survival curves: white recipients

These trends can also be observed on the predictions made by the RSF model (see Figure 7.7)

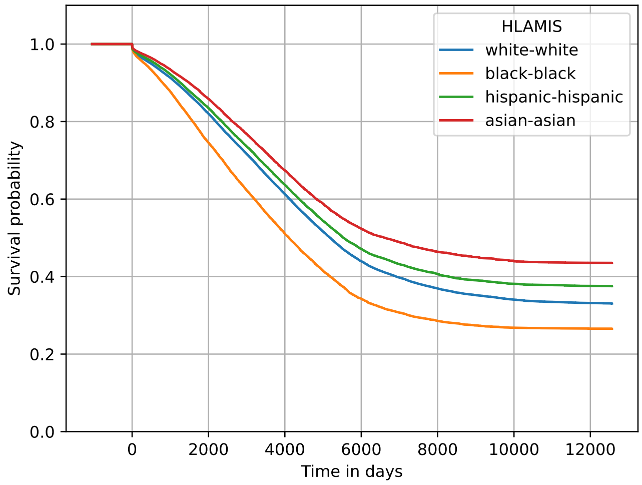

Figure 7.7: Averaged RSF predictions by ethnicity

7.3 Usage for Computing Compatibility

The RSF model could be used in cases where there is one patient looking for a donor and multiple donors are available. The model would asses their compatibility by calculating the risk score of each of the possible pairings. The pairing with the lowest risk score could be subsequently chosen for transplantation.

Following is an example of how the model could be used for computing compatibility and thus assist in the decision-making process of choosing the best donor-recipient paring.

7.3.1 Example of Compatibility Computation

Table 7.2 contains information about the patient, Table 7.3 contains information about 5 possible donors along with a risk score predicted by the RSF model, given that the donor's kidney was to be selected for the transplantation. The donor-recipient pairing with the lowest rick score would be selected for transplantation.

Table 7.2: Patient Information

Column	Value
CREAT_TRR	6.82
AGE	57
END_BMI_CALC	28.30
DAYSWAIT_CHRON_KI	211
TOT_SERUM_ALBUM	3.50
DAYSWAIT_ALLOC	211
PREV_TX_ANY	N
ON_DIALYSIS	Y
DIAB	2
WORK_INCOME_TCR	Y
PRI_PAYMENT_TCR_KI	1
PRE_TX_TXFUS	N
PERIP_VASC	Y
ETHCAT	1
FUNC_STAT_TCR	2090
HCV_SEROSTATUS	N
HIST_CIG	N
EBV_SEROSTATUS	P
DIAG_KI	3069

Table 7.3: Donor information + Risk Score predicted by RSF

	AGE_DON	DRMIS	AMIS	BMIS	HLAMIS	KI_CREAT_PREOP	ETHCAT_DON	DIABETES_DON	Risk Score
1	23	1	0	1	3	0.86	2	N	7036
2	30	1	1	1	3	0.86	1	N	7665
3	45	1	1	2	4	0.90	1	N	7878
4	49	0	0	0	0	0.90	1	N	5603
5	41	2	2	2	6	1.00	2	N	7734

In this case, the donor number 4 would be selected.

7.3.2 Age Bias

On the other hand, a situation for which the model might not be suited for is a case where there are multiple patients and one (or more) donor(s). If the model were to be used to find the best match in this scenario, the model would tend to select the youngest patient and discriminate against older patients as the recipient's age is the most important feature influencing the graft's survival time (as shown in Section 7.1.1). This however, could result in scenarios where a donor could be the best possible match for an elderly recipient by all other measures but the recipient's age, while for the younger patient, even better suited match might be found in the near future, however the model would still rank the younger patient as the best match solely because of his age.

Taking age out of the equation would not be a good solution as that would seriously damper the models accuracy. There also might be important correlations between age and other features. Preferably, the resulting risk score could be weighted based on the recipients age, e.g., if $\tilde{R}$ were to denote the adjusted risk score, R the risk score predicted by the RSF model, w a weight and $\mathbf{x}$ the pairing's feature vector, then the weighted risk score could be given by

$$\tilde{R}(\mathbf{x}) = w(\mathbf{x}) \cdot R(\mathbf{x}) \tag{7.1}$$

where w could, for example, be defined as

$$w(\mathbf{x}) = \left(\frac{x^{(AGE)}}{c}\right)^{-1} \tag{7.2}$$

where $x^{(AGE)}$ is the recipient's age attribute of the feature vector and c could be a constant such as the *life expectancy* in the given region.

However, as illustrated in Figure 7.1, the recipient's age only begins to play a factor in the 60+ age group. Therefore, using the formula suggested in (7.1) may put younger patients at a disadvantage, when compared to middle-aged patients. Consequently, an alternative solution could be to subtract a certain amount from the risk score of patients belonging to the 60+ age cohort. Using the same notation as before, the adjusted risk score could be given by

$$\tilde{R}(\mathbf{x}) = \begin{cases} R(\mathbf{x}) & \text{if } x^{(AGE)} < 60 \\ R(\mathbf{x}) - c & \text{else} \end{cases} \tag{7.3}$$

where c could be the difference between the average risk in the $0 - 59$ and 60 age cohorts. The average risk score of both of the cohorts was calculated by sampling 1000 samples from each of the cohorts, calculating a risk score for each of the samples, and finally taking their averages. The resulting difference in average risk scores is

$$c = \hat{R}_{0-59} - \hat{R}_{60+} = 2269 \tag{7.4}$$

where $\hat{R}_n$ represent the average risk score of the n age cohort.

7.4 Possibilities for Future Research

In spite of the RSF model's demonstrated superiority in accuracy in comparison to the Cox Regression method, which is commonly used for kidney transplant survival estimation, the C-index value of 0.638 indicates that the model is still far from being a reliable predictor. Therefore, a future research could focus on enhancing the model's accuracy by altering the model's algorithm or by working with higher quality data. For example, the UNOS dataset contains detailed medical information obtained at the time of transplantation along with abundant follow-up data. However, for the purposes of this study, a more expansive dataset regarding the pre-transplantation state of both the donor and the recipient would be advantageous. Consequently, a potential future research could entail gathering of such data.

Furthermore, it can also be expected, that future models will naturally improve their prediction accuracy due to the availability of larger quantities of non-censored as well as high-quality data.

Additionally, it may prove worthwhile to adapt the model for scenarios involving multiple donors and recipients, extending beyond the suggestions outlined in Section 7.3.2.

Conclusion

The increasing prevalence of the end-stage kidney disease imposes a great strain on the healthcare system. With the demand for kidneys greatly exceeding the supply, patients have to undergo months, sometimes even years of dialysis waiting list. Consequently, effective matchmaking of recipient parings is of the utmost importance, not only to ensure that people don't return to the waiting list but, even more importantly, to prevent premature deaths. Finding pairings that will ensure the longest possible survival time is crucial for achieving this goal. This however, is no easy task as there are complex factors affecting the compatibility of donors and patients, many of which may be currently unaccounted for.

Machine learning demonstrates a promising solution to this problem. As exemplified in this study, the random forests model, a machine learning model that builds on the foundation of survival analysis, outperforms the conventionally used Cox proportional hazards model in terms of the accuracy of predicted survival prospects following a renal transplantation. The random survival forest has the ability to evaluate the quality of a potential match based on either the predicted survival or cumulative hazard functions, and could thus be used as a part of the decision-making process of choosing the optimal donor-recipient pairing. In particular, the risk score derived from the cumulative hazard function could be used to quantify the compatibility between a recipient and a potential donor, given that any potential biases are addressed. As demonstrated, these models could also serve as a tool for examining the influence of various factors on the outcome of renal transplantations, both short-term and long-term.

Although further research is required to improve the model's reliability to a level suitable for clinical adoption, the model even in its premature stages is still able to reveal insights into the determinants impacting graft survival following renal transplantations.

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] V. Jha, et al., *Chronic kidney disease: global dimension and perspectives*, Lancet, 2013.

[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`

[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.

[27] National Kidney Foundation, *A to Z Health Guide*, `https://www.kidney.org/atoz/content`

[28] 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)