



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



Estimating Kidney Transplantation Donor-Recipient Compatibility Using Machine Learning

Odhad kompatibility dárce a příjemce pro transplantaci ledvin pomocí strojového učení

Bachelor's Degree Project

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

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

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

Pokyny pro vypracování:

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



Doporučená literatura:

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

Jméno a pracoviště vedoucího bakalářské práce:

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

Jméno a pracoviště konzultanta:


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


Datum zadání bakalářské práce: 31.10.2022

Datum odevzdání bakalářské práce: 2.8.2023

Doba platnosti zadání je dva roky od data zadání.

V Praze dne 31. října 2022


.....
garant oboru


.....
vedoucí katedry




.....
děkan

Acknowledgment:

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

Author's declaration:

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

Prague, August 2, 2023

Matěj Klouček

Odhad kompatibility dárce a příjemce pro transplantaci ledvin pomocí strojového učení

Obor: Matematické inženýrství

Druh práce: Bakalářská práce

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

Klíčová slova: klíčová slova (nebo výrazy) seřazená podle abecedy a oddělená čárkou

Estimating kidney transplantation donor-recipient compatibility using machine learning

[illegible]

Key words: keywords in alphabetical order separated by commas

Contents

Introduction	7
1 Machine Learning	8
1.1 General Overview of Machine Learning	8
1.1.1 Classification of Machine Learning Models	8
1.2 Training and Evaluating a Machine Learning Model	9
1.2.1 Training, Validation and Test sets	9
1.2.2 Overfitting and Underfitting	9
1.3 Decision Trees and Random Forests	10
1.3.1 Decision Trees	10
1.3.2 Ensemble Learning and Random Forests	13
2 Renal transplantation	14
2.1 Pre-Transplant Procedures, HLA Typing and other important factors	14
2.2 Survival analysis	14
2.2.1 Cox Regression	14
2.3 Random Survival Forests	16
2.3.1 Building a Random Survival Forests model	16
2.3.2 Split rules	18
2.3.3 Calculating Prediction Error	18
2.4 Findings of Similar Papers and Potential Improvements	19
3 Model training	20
3.1 Data Selection	20
3.2 Programming	20
3.3 Traing the model	20
3.4 Optimization	20
4 Model evaluation	21
4.1 General evaluation	21
4.2 Performance on different population samples	21
4.3 Comparison to previously developed models	21
Conclusion	22

Introduction

The use of machine learning in the field of medicine has been gaining traction in recent years, and one area where it has the potential to make a significant impact is in the field of organ transplantations, specifically in helping to find compatible donor-recipient pairs. The goal of this thesis is to explore the use of machine learning methods to predict compatibility of donors and recipients, with the ultimate goal of improving the success rate of kidney transplants and thus potentially saving lives and consequently reducing the burden on the healthcare system. The process of finding a compatible donor for a transplant patient is a complex and time-consuming task, involving a variety of factors including among others blood type, HLA typing and medical history of both the donor and the recipient. Despite advances in medical technology, the process of finding a compatible donor is still challenging, with many patients waiting years for a suitable match. 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 collecting and analyzing data on past transplantations from the US-based *United Network for Organ Sharing (UNOS)* dataset, including information on both living and deceased donors, recipients and their post-transplant outcome. This data will be used to train and test machine learning models that can predict compatibility between donors and recipients. The models will be evaluated using various metrics such as accuracy, precision, recall, and F1-score. Additionally, the research will also examine the performance of the developed models on different population samples, namely on the dataset provided by the *Czech Institute for Clinical and Experimental Medicine (IKEM)*.

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

Chapter 1

Machine Learning

1.1 General Overview of Machine Learning

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

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

1.1.1 Classification of Machine Learning Models

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

1.1.1.1 Classification by Data Type

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

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

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

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

Finally, *Reinforcement Learning* is a type of machine model, 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*) [1][2].

1.1.1.2 Instance-based vs. Model-based

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

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

1.2 Training and Evaluating a Machine Learning Model

1.2.1 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 generalise for new instances thus making it useless in practice, that's why it's important to keep some of the data aside for validation and testing, these two together are called *hold-out* sets [1].

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

Once the training of the model has been done using the training set, the performance of the model is tested using the test set, i.e. data that it hasn't seen before. A good metric for measuring the quality of the model is the *generalization error*, given by the error rate of the model on new cases.

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 [2].

1.2.2 Overfitting and Underfitting

If the generalization error of the model is low on the training set, but high on the test set, it means that the model has learned unnecessary details from the training set which then hampers its ability to generalise for new instances. This phenomena is called *overfitting* and is a common problem that arises during the process of developing a machine learning model. Overfitting usually caused either by the data

being too noisy (errors in the data, many outliers), the dataset being too small or by having too many irrelevant features. One way to overcome this is too addressed the above mentioned problems by gathering more data, removing outliers and inputing 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* [2].

Machine learning models can also face an opposite problem, i.e. not being able learn the underlying patterns in the data and thus being inaccurate on both the test and training data. This is called *underfitting* and can be solved by using a more appropriate (more complex) model for the given task, better selecting features to train the model on or reducing any constraints that might have set to simplify the model in order to prevent overfitting [2].

1.3 Decision Trees and Random Forests

1.3.1 Decision Trees

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

Decision trees are in practice mostly binary trees where in each parent node (also called *decision node*), a specific attribute of the feature vector is examined and a split is made based on a given criteria whose specifics are learned during the training process. For example if a value of the given attribute is below a specific threshold, the left branch is followed, otherwise the right branch is followed, with the threshold being set to maximize a certain performance metric of the model. Once the leaf node (also called *terminal node*) is reached, the example is assigned a probability of belonging to a given categorical value in case of a classification tree or assigned a real target value in case of a regression tree.

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

1.3.1.1 Building a Classification Tree

A classification tree using the ID3 algorithm is built as follows [1]:

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

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

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

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

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

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

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

For each iteration (ordered pair (j, t)), the quality of the split is calculated by using the algorithm's criterion. In the case of the ID3 algorithm, the criterion is the entropy of a given set of examples which is given by:

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

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

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

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

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

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

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

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

1.3.1.2 Building a Regression Tree

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

Let X_j be the set of all possible categorical values for the feature $j \in \hat{k}$, where k is the total number of features in the dataset. Again let $\{(x^{(i)}, y^{(i)})\}_{i=1}^N$ denote a collection of labeled examples, where N is the

size of the collection, $\mathbf{x}^{(i)} = (x_1^{(i)}, \dots, x_k^{(i)})$ is a k -dimensional feature vector of the example $i \in \hat{N}$, where $x_j^{(i)} \in X_j$ for $\forall j \in \hat{k}$ and $y^{(i)} \in \mathbb{R}^+$ is the target value. The decision tree model will this time be a function that takes a k -dimensional feature vector with the same type of attributes as $\mathbf{x}^{(i)}$, $i \in \hat{k}$ as input, i.e. if $\mathbf{x} = (x_1, \dots, x_k)$ then $x_j \in X_j$ for $\forall j \in \hat{k}$ and outputs a real valued estimation, i.e. $f(\mathbf{x}) \in \mathbb{R}^+$.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1.3.2 Ensemble Learning and Random Forests

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

An example of a random forests model is training a group of classification trees on different random subsets of the training data and when faced with new input, predicting the class of the given input by choosing the class that got the most "votes" by the classification trees [2] (a class c gets a "vote" by the decision tree when the input \mathbf{x} is assigned the highest probability of belonging to c , i.e. class $c \in C$ gets the vote $\iff c = \arg \max_{a \in C} \Pr(y = a | \mathbf{x})$, where C is the set of all possible classes).

1.3.2.1 Building a Random Forest

The process of building a random forest model can be divided into four steps [1][2]:

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 (or other predictors) on different samples of the data in order to minimize risk of the predictors making the same kind of error. Samples are taken randomly from the dataset in order to create multiple different subsets of the data. This process can be divided based on whether the samples are taken "with replacement" or not i.e. whether instances of the data can be used in training of multiple predictors or just once, this is called *bagging* (abbreviation of *bootstrap aggregating*) and *pasting* respectively, with bagging being the generally more often used case for random forests.
2. *Feature sampling*: In order to even more differentiate the individual decision trees, only a random subset of the input features are considered when building a particular decision tree. The model will perform best when the input features are independent of each other.
3. *Building the decision trees*: For each sample of the data and input features, a decision tree is built using a given split criterion. For even more randomized trees it is possible to randomly select the threshold values for each split in the decision trees, instead of searching for the best possible threshold. These are called *Extremely Randomized Trees*.
4. *Aggregating Predictions*: Random forests make predictions by aggregating the predictions made by the individual decision trees, for example in the case of classification task, the class with the biggest amount of votes is selected while in the case of a regression task, the predicted value is calculated by averaging the values predicted by the individual decision trees.

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

Chapter 2

Renal transplantation

2.1 Pre-Transplant Procedures, HLA Typing and other important factors

- Discuss why even kidney transplantation happen and how are they usually performed (pre-transplant procedures etc.)
- Explain HLA typing
- Maybe give some data about how hard it is to find a suitable match for a patient

2.2 Survival analysis

Survival analysis is a statistical method used to analyze and model time-to-event data, where the event of interest is the occurrence of a specific event, such as death, failure, or disease. In survival analysis, we are interested in estimating the probability of an event occurring over time and identifying factors that affect the timing of the event.

The data used in survival analysis typically include information on the time of occurrence of the event of interest and the status of the event at a particular time point. For example, in a clinical trial, the data may include the time from treatment initiation to relapse of a disease, and whether the patient is still alive or has died at a specific follow-up time.

Survival analysis involves several statistical techniques, including the Kaplan-Meier estimator, Cox proportional hazards model, and accelerated failure time models. The Kaplan-Meier estimator is used to estimate the survival function, which describes the probability of survival over time. The Cox proportional hazards model is used to model the hazard function, which describes the instantaneous risk of an event occurring at a particular time, as a function of covariates. The accelerated failure time models are used to model the log-transformed time-to-event data as a linear function of covariates.

Survival analysis is widely used in medical research, engineering, and social sciences to analyze time-to-event data and make predictions about future events. It is also used in business and economics to model customer retention, product failure, and other time-to-event phenomena.

2.2.1 Cox Regression

Cox regression, also known as the *Proportional hazards model* or *Cox proportional hazards model*, is a type of survival model that is used to analyze the relationship between the time before some event happens and one or more predictor variables. It is commonly used in medical research to investigate the factors that influence the time to an event, such as graft failure.

The Cox regression model assumes that the *hazard rate* (the risk of the event occurring at any given time) is proportional across different levels of the predictor variables. In other words, the effect of the predictor variables on the hazard rate remains constant over time.

Cox regression can be used to identify compatible pairs of donors and recipients for kidney transplantations by analyzing the survival time of transplanted kidneys and identifying factors that impact the likelihood of graft failure.

One important predictor variable in this context is the degree of human leukocyte antigen (HLA) matching between the donor and recipient. HLA is a group of genes that play a critical role in the immune system's ability to recognize and respond to foreign tissue, and HLA mismatches between donor and recipient can increase the risk of graft failure.

Using Cox regression allows analyzing the survival time of transplanted kidneys while controlling for other factors that may influence graft survival, such as age, sex, and underlying medical conditions and using the results to estimate the hazard ratio for HLA mismatch, which represents the relative risk of graft failure for each additional HLA mismatch. This information can then be used to identify compatible pairs of donors and recipients by selecting those with the lowest risk of graft failure based on their HLA compatibility and other relevant factors.

Cox regression models the relationship between survival time and predictor variables using hazard functions, which according to the model can be modeled as product of a baseline hazard function and an exponential function of the predictor variables.

The hazard function describes the probability of an event occurring (such as death or failure of a transplanted kidney) at a specific point in time, given that the event has not already occurred. It is denoted by $\lambda(t)$ and can be defined as:

$$\lambda(t) = \lim_{\Delta t \rightarrow 0} \frac{P(t \leq T < t + \Delta t \mid T \geq t)}{\Delta t} \quad (2.1)$$

where T is a random variable representing the time until the event occurs, and $P(t \leq T < t + \Delta t \mid T \geq t)$ represents the probability of the event occurring between time t and time $t + \Delta t$, given that the event has not already occurred by time t .

The Cox proportional hazards model assumes that the hazard function can be modeled as follows:

$$\lambda(t \mid \mathbf{X}) = \lambda_0(t) \exp(\beta_1 X_1 + \beta_2 X_2 + \dots + \beta_k X_k) \quad (2.2)$$

where $\lambda_0(t)$ represents the baseline hazard function that is not dependent on the predictor variables, k is the number of predictor variables, $\mathbf{X} = (X_1, X_2, \dots, X_k)$ is a vector of predictor variables and $\beta_1, \beta_2, \dots, \beta_k$ represent the coefficients associated with each predictor variable. The exponential function of the predictor variables represents the proportional change in the hazard function associated with a unit change in the corresponding predictor variable, holding all other predictor variables constant.

The hazard ratio (HR) is defined as the ratio of the hazard functions for two groups with different values of a predictor variable, while holding all other predictor variables constant. For example, let $\tilde{\mathbf{X}} = (X_1, \dots, X_i + 1, \dots, X_k)$ then:

$$\frac{\lambda(t \mid \tilde{\mathbf{X}})}{\lambda(t \mid \mathbf{X})} = \frac{\lambda_0(t) \exp(\beta_1 X_1 + \dots + \beta_i (X_i + 1) + \dots + \beta_k X_k)}{\lambda_0(t) \exp(\beta_1 X_1 + \dots + \beta_i X_i + \dots + \beta_k X_k)} = \frac{\exp(\beta_i) \prod_{j=1}^k \exp(\beta_j X_j)}{\prod_{j=1}^k \exp(\beta_j X_j)} = \exp(\beta_i)$$

Meaning then the hazard ratio for a given predictor associated with a one-unit increase in that variable is given by:

$$HR_i = \exp(\beta_i), \quad i \in \hat{k} \quad (2.3)$$

More broadly, the hazard ratio for two groups with varying predictor values $X_i^{(1)}$ and $X_i^{(2)}$ respectively can be written as:

$$HR_i = \exp(\beta_i (X_i^{(1)} - X_i^{(2)})) \quad (2.4)$$

As $\lambda_0(t)$ cancels out, the hazard ratio stays constant over time, in another words the hazards are *proportional*.

The Cox regression model estimates the coefficients $\beta_i, \forall i \in \hat{k}$ and the baseline hazard function $\lambda_0(t)$ using *maximum likelihood estimation*. The model can be used to predict the hazard function for different combinations of predictor variables, which can be used to estimate the probability of an event occurring at different points in time, given the values of the predictor variables.

The Cox regression model can also be used to estimate the survival function, which represents the probability of surviving beyond a given time t , given the values of the predictor variables. The survival function is defined as:

$$S(t | \mathbf{X}) = \exp\left(-\int_0^t \lambda(u | \mathbf{X}) du\right) \quad (2.5)$$

where $\lambda(u | \mathbf{X})$ represents the hazard function at time u , given the values of the predictor variables.

Overall, Cox regression provides a powerful tool for analyzing survival data and identifying the factors that influence the likelihood of an event occurring over time.

2.3 Random Survival Forests

Random Survival Forests (RSF) is an ensemble machine learning method used for analyzing right-censored survival data. RSF combine the concepts of *survival analysis* and *random forests*, which allows it to handle issues commonly associated with traditional survival analysis methods, such as restrictive assumptions about the model's parameters (e.g. proportionality in the Cox Regression model), the inability to handle nonlinear relations in the data and issues with missing data, as well as issues associated with regular random forest models, such as being restricted to regression and classification tasks and the inability to handle right-censored survival data [5].

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

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

2.3.1 Building a Random Survival Forests model

The algorithm for building a RSF model works as follows [5].

1. A set number of subsets is taken from the data using the bootstrap sampling method, with each sample excluding on average 37% of the original data. The excluded data are referred to as *out-of-bag data* (OOB) and they are used for computing the ensemble commulative hazard.

2. A *binary survival tree* (BST) is built for each of the bootstrap samples. The algorithm for growing a BST generally follows the same principles as described in 1.3.1, i.e. at each node the algorithm iterates over all possible features and values and decides on the best split, with the metric used for deciding the split being the difference between survival times predicted by the given child nodes (the split rule is explained in detail in 2.3.2). The goal in this case is to maximize the survival difference in order to separate dissimilar cases. Another difference between a BST and a regular decision tree is that each node contains not only information about the survival time denoted by T , but also binary information about the censoring status denoted by δ . The algorithm is restricted by a criterion that each node should contain at least 1 unique death, which leads to a point where no new splits can be made meaning the tree has been fully grown. Once this point is reached, each terminal node outputs a *cumulative hazard function* (CHF) denoted by $\hat{H}(t)$ which is calculated as follows:

Let A denote the set of all terminal nodes, then each node $j \in A$ contains a set of cases (individuals) $(T_{1,j}, \delta_{1,j}), \dots, (T_{n(j),j}, \delta_{n(j),j})$, where $n(j)$ is the number of cases belonging to the node j , $T_{i,j}$ is the survival time of the i -th case belonging to the node j , $\delta_{i,j} \in \{0, 1\}$ is the censoring status of the i -th case at time $T_{i,j}$, with $\delta_{i,j} = 0$ if the individual is right-censored (e.g. has a functioning graft) and $\delta_{i,j} = 1$ if an event has been observed at a given time $T_{i,j}$ (e.g. graft failure). Then let $t_{1,j} < t_{2,j} < \dots < t_{N(j),j}$ denote the time of the different events observed in node j , where $N(j) = \sum_{i=1}^{n(j)} \delta_{i,j}$ is the total number of observed events. The CHF for the given node is then given by the Nelson-Maier estimator

$$\hat{H}_j(t) = \sum_{t_{k,j} \leq t} \frac{d_{k,j}}{Y_{k,j}} \quad (2.6)$$

where $d_{k,j}$ is the number of events observed at time $t_{k,j}$ and $Y_{k,j}$ is the number of people at risk at time $t_{k,j}$ i.e. the number of individuals that are yet to experience an event.

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

$$H(t | \mathbf{x}) = \hat{H}_j(t) \quad (2.7)$$

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

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

The bootstrap ensemble CHF is then defined as

$$H_e^*(t | \mathbf{x}_i) = \frac{1}{B} \sum_{b=1}^B H_b(t | \mathbf{x}_i) \quad (2.8)$$

where $H_b(t | \mathbf{x}_i)$ is the CHF of a tree grown from the bootstrap sample b , defined in (2.7).

In contrast, the OOB ensemble CHF is defined as

$$H_e^{**}(t | \mathbf{x}_i) = \frac{\sum_{b=1}^B I_{i,b} H_b(t | \mathbf{x}_i)}{\sum_{b=1}^B I_{i,b}} \quad (2.9)$$

2.3.2 Split rules

TODO: Explain the log-rank statistic.

2.3.3 Calculating Prediction Error

The prediction error of random survival forest models is calculated using the *Harell's concordance index* (also known as *C-index*) [5], which is a metric commonly used in medical research as well as machine learning to evaluate the performance of predictive models. The C-index is equal to the area under the *ROC curve* (short for Receiver Operating Characteristics curve), which is a graphical plot that illustrates the performance of a binary classification model at different classification thresholds [6, Classification - ROC Curve and AUC].

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

$$TPR = \frac{TP}{TP + FN} \quad (2.10)$$

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

$$FPR = \frac{FP}{FP + TN} \quad (2.11)$$

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).

Lowering the classification threshold results in more cases being classified as positive, thus generally increasing both the true positive and the false negative rate.

The area under the ROC curve can be interpreted as the probability that a given model will rank a random positive example more highly (e.g. it will assign a more favourable hazard function) than a random negative example. The C-index can attain values from 0 to 1 with 1 meaning a perfect model whose predictions are always correct and 0 meaning it's always wrong. A random classifier that assigns the predicted value with a uniform distribution, will have a C-index of 0.5. Generally, models with a C-index above 0.7 are considered to be good predictors.

In the case of RSF, it isn't necessary to know the function of the ROC curve explicitly, rather the C-index is calculated as follows [5]:

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

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

The cases that are left are called *permissible* and we denote their set by P .

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

(a) if $T_i \neq T_j$ then

$$c_{i,j} = \begin{cases} 1 & \text{if } T_i < T_j \implies i \text{ has worse predicted outcome than } j \\ 0.5 & \text{if } T_i < T_j \implies \text{predicted outcomes of } i \text{ and } j \text{ are tied} \\ 0 & \text{else} \end{cases} \quad (2.13)$$

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

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

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

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

3. The C-index is then given by

$$C = \frac{1}{|P|} \sum_{i,j \in P, i \neq j} c_{i,j} \quad (2.12)$$

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

$$\sum_{k=1}^n H_e^{**}(t_k \mid \mathbf{x}_i) < \sum_{k=1}^n H_e^{**}(t_k \mid \mathbf{x}_j) \quad (2.13)$$

where \mathbf{x}_l is a feature vector of the case l , and H_e^{**} is the OOB ensemble CHF defined in (2.9).

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

2.4 Findings of Similar Papers and Potential Improvements

Chapter 3

Model training

3.1 Data Selection

- How did we obtain the data?
- The work we did pre-analysis: importing from MongoDB, creating .csv files, using Helios cluster
- Trimming the dataset, scripts to exclude obviously irrelevant data
- Data imputation: scripts to handle missing data
- Discuss right-censored data (Patients with still working grafts)

3.2 Programming

- packages: scikit-survival

3.3 Traing the model

Here I will possible include some code but mostly graphs.

3.4 Optimization

Trying different things to get better results.

Chapter 4

Model evaluation

4.1 General evaluation

Discussion about the results.

4.2 Performance on different population samples

Try to locally differentiate ethnic groups within the UNOS database, possibly try to fit the model onto the IKEM dataset.

4.3 Comparison to previously developed models

Discussion about how this work compared to the previous ones.

Conclusion

Text of the conclusion...

Bibliography

- [1] A. Burkov: *The Hundred-Page Machine Learning Book*. Andriy Burkov, 2019.
- [2] A. Géron: *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow 2nd Edition*. O'Reilly Media, 2019
- [3] M. Mohri, A. Rostamizadeh, A. Talwalkar: *Foundations of Machine Learning*. MIT Press, 2018.
- [4] D. R. Cox, D. Oakes: *Analysis of Survival Data*. Chapman & Hall, 1984.
- [5] H. Ishwaran, U. B. Kogalur, E. H. Blackstone. M. S. Lauer: *Random survival forests*. Ann. Appl. Stat. 2 (3) 841 - 860, September 200
- [6] Google Developers, *Machine Learning Crash Course*, <https://developers.google.com/machine-learning/crash-course>