# RECURRENT NEURAL NETWORKS FOR COMPLEX SURVIVAL PROBLEMS

# KARMAŞIK YAŞAM PROBLEMLERİ İÇİN YİNELEMELİ SİNİR AĞLARI

**PIUS SINDIYO MARTHIN**

**PROF. DR. NİHAL ATA TUTKUN**
**Supervisor**

Submitted to Graduate School of Science and Engineering of Hacettepe University

as a Partial Fulfillment to the Requirements

for the Award of the Degree of Doctor of Philosophy

in Statistics.

2023

# ABSTRACT


**RECURRENT NEURAL NETWORKS FOR COMPLEX SURVIVAL PROBLEMS**

**Pius Sindiyo MARTHIN**

**Doctor of Philosophy, Department of Statistics**

**Supervisor: Prof. Dr. Nihal ATA TUTKUN**

**May 2023, 130 Pages**

In this study, we introduce a novel deep learning technique (CmpXRnnSurv_AE) that obliterates the limitations imposed by traditional approaches and addresses the limitations of the existing deep learning systems to jointly predict the risk-specific probabilities and survival function for recurrent events with competing risks. We introduce the component termed Risks Information Weights (RIW) as an attention mechanism to compute the weighted cumulative incidence function (WCIF) and an external auto-encoder (ExternalAE) as a feature selector to extract complex characteristics among the set of covariates responsible for the cause-specific events. We train our model using synthetic and real data sets and employ the appropriate metrics for complex survival models for evaluation. As benchmarks, we selected both traditional, and machine learning models and our model demonstrates better performance across all datasets with the best weighted time dependent concordant index score of 92% and the weighted time dependent Brier score of 20%.



**Keywords:** Cumulative Incidence Function (CIF), Risk Information Weight (RIW), Autoencoders (AE), Survival analysis, Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM).

# ÖZET

## KARMAŞIK YAŞAM PROBLEMLERİ İÇİN YİNELEMELİ SİNİR AĞLARI

**Pius Sindiyo MARTHIN**

**Doktora, İstatistik Bölümü**

**Danışman: Prof. Dr. Nihal ATA TUTKUN**

**Mayıs 2023, 130 Sayfa**

Bu çalışmada, yarışan riskli tekrarlı olaylar için riske özel olasıklıkları ve yaşam fonksiyonunu birlikte tahmin etmek için geleneksel yaklaşımların dayattığı sınırlamaları ortadan kaldıran ve mevcut derin öğrenme sistemlerinin kısıtlamalarını gösteren yeni bir derin öğrenme tekniği (CmpXRnnSurv_AE) verilmiştir. Ağırlıklı kümülatif insidans fonksiyonunu (WCIF) hesaplamak için bir dikkat mekanizması olarak Risk Bilgi Ağırlıkları (RIW) adlı bileşeni ve nedene özgü olaylardan sorumlu ortak değişkenler kümesi arasından karmaşık özellikleri ayıklamak için bir özellik seçici olarak harici bir otomatik kodlayıcıyı (ExternalAE) önerilmiştir. Modelimiz yapay ve gerçek veri kümeleri kullanarak eğitilmiş ve değerlendirme için karmaşık yaşam modelleri için uygun ölçümler kullanılmıştır. Kıyaslama olarak hem geleneksel hem de makine öğrenimi modelleri seçilmiş ve önerilen modelin tüm veri kümelerinde %92'lik en iyi ağırlıklı zamana bağlı uyumlu uygunluk indeksi ve %20'lik ağırlıklı zamana bağlı Brier skor ile daha iyi performans gösterdiği sonucuna ulaşılmıştır.

**Anahtar Kelimeler:** Birikimli İnsidans Fonksiyonu, Risk Bilgi Ağırlığı, Otomatik Kodlayıcılar, Yaşam Çözümlemesi, Yinelemeli Sinir Ağları, Uzun-Kısa Vadeli Hafıza.

# ACKNOWLEDGMENT

First and foremost, I would like to express my deepest gratitude to my Lord and Savior Jesus Christ, for the gift of life and every help I ever required. My deepest gratitude to my thesis advisor, Prof. Dr. Nihal Ata Tutkun, for her guidance, support, and encouragement throughout my Ph.D. journey. Her unwavering belief in me, and her constant availability for discussion and feedback have been instrumental in the successful completion of this thesis.

I would also like to thank the members of my thesis committee, Prof. Dr. Duru Karasoy and Prof. Dr. Yasemin Yavuz, for their valuable feedback and suggestions on the various stages of my research. Their insights have been invaluable in shaping the direction and outcome of my work. I am also grateful to the Department of Statistics and Hacettepe University Graduate School of Science and Engineering for providing me with the assistance and opportunities necessary to conduct my research.

I would also like to acknowledge my colleagues and friends in the Department of Mathematics, University of Bremen, who have provided me with a stimulating and supportive environment for my research. Special thanks to Prof.Dr.Dr.h.c. Peter Maass, University of Bremen for his guidance and mentorship during my time at Bremen. I would like to express my sincere gratitude to the AGENS research group at the University of Bremen. Their camaraderie and encouragement have been greatly appreciated.

I would also like to thank my family and friends for their unwavering support and understanding throughout the duration of my Ph.D. I could not have done this without them. Finally, I would like to express my appreciation for the support and funding provided by the Turkish government through the Türkiye Scholarships Bursları fellowship. This funding was essential for the completion of my research.

Overall, I am grateful for the support, guidance, and opportunities provided by all of these individuals and organizations, which have been instrumental in the successful completion of my Ph.D. and this thesis. It is an honor to have had the opportunity to pursue my passion for research under such esteemed guidance and support.

For UNOS-OPTN(KIDPAN) dataset: This work was supported in part by Health Resources and Services Administration contract 234-2005-370011C. The content is the responsibility of the authors alone and does not necessarily reflect the views or policies of the Department of Health and Human Services, nor does mention of trade names, commercial products, or organizations imply endorsement by the U.S. Government. I would like to express my gratitude to the Massachusetts Institute of Technology (MIT ) for granting access to the Medical Information Mart for Intensive Care (MIMIC-III) dataset used in this research. The use of this dataset was instrumental in the completion of my study, and I am grateful for the effort put into collecting and sharing this valuable resource.

Pius Sindiyo MARTHIN
May 2023, Ankara

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# LIST OF ABBREVIATIONS

- OOB: Out-of-bag samples
- CHF: Cumulative hazard function
- RSCI: Risk-specific concordance index
- ROC: Receiver operating characteristic
- DMGP: Deep multi-task Gaussian process
- SVM: Support vector machine
- EHR: Electronic health records
- KM: Kaplan-Meier
- EN-Cox: Elastic net Cox
- MIMIC-III: Medical information Mart for intensive care
- AFT: Accelerated failure time
- MTMT: Multiple Times Point Multi-Task
- RNN: Recurrent Neural Networks
- LSTM: Long short-term memory
- AI: Artificial intelligence
- ICU: Intensive care unit
- CPH: Cox proportional hazards model
- CIF: Cumulative incidence function
- GRU: Gated recurrent unit
- MIT: Massachusetts Institute of Science and Technology
- AE: Auto encoders
- RF: Random forest
- RIW: Risks information weight
- WOS: Weighted overall survival
- WCIF: Weighted cumulative incidence function
- RSF: Random survival forest
- MLPs: A multilayer perceptron/Feed-forward neural network
- XOR: Exclusive or
- BPP: Backpropagation
- BPPT: Backpropagation through time

- MC: Monte Carlo
- SGD: Stochastic gradient descent
- UNOS: United Network for Organ Sharing
- OPTN: Organ Procurement and Transplantation Network
- KIDPAN: Kidney-pancreas dataset
- KGRFT: Kidney graft
- PGRFT: Pancreas graft
- ReduceLROnPlateau: Reducing learning rate on plateaus
- HCNN: Historical consistent neural network
- C-index: Time dependent concordat index
- CNN: Convolution neural networks

# LIST OF SYMBOLES

$K$: A class of $K$ competing risks

$K$: A vector of $K$ distinct events

$k$: A distinct event

$f \circ g$: A composite function

$X$: A random vector

$x$: A random variable

$H$: Distribution function

$X \to Y$: A map from space $X\ to\ Y$

$\odot$: Elements-wise multiplication/Hadamard product

$\emptyset$: An empty set

$R^n$: Set of n-tuples of real numbers

$\prod$: General multiplication

$f_\theta(x \to y)$: A transformation from *x to y* using function $f$

$\sigma$: A sigmoid function

$||.||_F$: Frobenius norm

$[X \oplus Y]$: Concatenation of vector $X\ and\ Y$

$\theta$: A distinct parameter

$\Theta$: A class of parameters

$I(.)$: An indicator function

$L$: A loss function

$\sum$: General summation

$M_{\theta \in \Theta}(.)$: A model /A hypothesis class

: A partial derivative

$C$: An observed survival/censorship time

$C$: A cumulative incidence function

$\Delta$: Infinitesimal change

$l_{\theta \in \Theta}(.)$: A partial loglikelihood function

$R$: A risk set

$T$: Survival time

$l_1$: $l_1 norm$

$\epsilon_{t,k}$: Trade-off hyperparameter for the $k^{th}$ risk between a pair of subjects at time t

# OBJECTIVE

In this study, we propose a deep learning technique to jointly predict the weighted cumulative incidence function (WCIF) and the weighted overall survival function (WOS) for recurrent events with competing risks. To account for the complex correlational structure, we introduce the component termed Risks Information Weights (RIW) as an attention mechanism to compute the WCIF. We also propose an external auto-encoder (ExternalAE) as a feature selector to extract relevance characteristics among the set of covariates responsible for the cause-specific events.

# 1. INTRODUCTION

Survival analysis is the most used approach in the modeling of time-to-event data (Singer & Willet, 2003). This technique has been widely utilized across various disciplines, including but not limited to medicine, engineering, and economics, to explore complex structural relationships among the set of covariates concerning the outcome of an entity (Cox, 1975; Johnson *et al.,* 2016). In the medical field, survival models can assist clinicians in running the appropriate diagnostics and making proper treatment arrangements to optimize overall costs and patient well-being (Luck *et al.,* 2017).

Despite being popular in modeling survival data, the traditional approach to survival analysis which includes both parametric, non-parametric, and semi-parametric techniques, faces limitations due to strong assumptions that limit the inference and prediction ability of the resulting models (Lee *et al.,* 2020). As a widely used semi-parametric approach for survival data, the Cox proportional hazards model (CPH) encounters interpretability issues and low performance due to the unknown distribution of the outcomes and proportionality assumption for the hazard ratio, respectively (Lee *et al.,* 2018; 2020).

When we encounter complex survival data, predicting the risks of failure or survival time distribution of an instance requires techniques that correctly model the complex correlation structure and the relationship between covariates and the outcome. Before the revival of deep learning recently, for simple survival problems, the traditional approach to survival analysis, such as the CPH model and its variants, gave competitively better performance compared to machine learning approaches (Wang *et al.,* 2019). This is because the traditional methods work well with censored data when the simplest data structure is employed or when the underlying stochastic process for the survival time is known (Lee *et al.,* 2018).

However, with the increase in survival data complexity, such as recurrent events with competing risks, traditional methods appeared to be overwhelmed (Gupta *et al.,* 2019). Recently, tremendous achievements have been attained in the field of machine learning. Machine learning practitioners can now train models with very high accuracy compared to many statistical approaches (Wang *et*

*al.,* 2020). Also, technological advancement has facilitated enormous collections and data storage that are freely accessible to the community (Du *et al.*, 2020). Therefore, survival model practitioners can now utilize Electronic Health Records (EHR) and big data from different sources to train machine learning models that are less limited to traditional assumptions (Coccia*,* 2020).

Despite the outstanding achievements of deep learning models over the traditional methods across disciplines, the field of survival analysis seems to need to catch up in employing these techniques, especially in the case of complex survival data. Although several deep learning models exist for survival analysis, as discussed in section 2, studies focusing on complex survival problems remain limited.

The existing deep learning models on complex survival problem, such as CRESA (Gupta *et al.,* 2019) and DeepHit (Lee *et al.,* 2018, 2020) relies on the basic cumulative incidence function (CIF), which poses a limitation on fully accounting for the complex correlational structure caused by recurrent events with competing risks. In addition, these models may suffer from performance deficits due to noise and redundant information in the data. In practice, recurrent events with competing risks survival data come from various sources such as EHR. These data are most likely subjected to noisy and redundant information, which may impact the model performance (Rietschel *et al.,* 2018). To attain reasonable performance on such data, it is necessary to design a model that dynamically accounts for the data structure's noise, redundancy, and complexity.

In this study, we propose a new deep learning technique to jointly predict the weighted cumulative incidence function (WCIF) and the weighted overall survival function (WOS) for recurrent events with competing risks. To account for the complex correlational structure, we introduce the component termed Risks Information Weights (RIW) as an attention mechanism to compute the WCIF. We also propose an external auto-encoder (ExternalAE) as a feature selector to extract relevance characteristics among the set of covariates responsible for the cause-specific events. The proposed model has multi-fold advantages, including complete freedom from traditional assumptions and noise tolerance by using the ExternalAE. Also, the network successfully addresses the complex correlational structures in the data by predicting the WCIF and facilitates feature transfers through the residual connections.

The organization of the thesis is given below:

- Chapter 2 introduces related studies on complex survival problems,
- Chapter 3 presents the research methodology,
- Chapter 4 discuss machine learning approach to complex survival problems,
- Chapter 5 presents the proposed network architecture for complex survival model,
- Chapter 6 introduces the loss functions for complex survival problems,
- Chapter 7 presents the general approach of training machine learning models,
- Chapter 8 introduces the benchmark models,
- Chapter 9 presents experimentations,
- Chapter 10 discuss the results, and
- Chapter 11 presents the concluding remarks and future work recommendations.

# 2. RELATED STUDIES

## 2.1. Traditional Approach to Survival Modeling

Survival analysis aggregates statistical procedures to analyze the time-to-event data (Cox, 1972). Time to an event data is experienced in different fields such as the medical field, time until the death or time until the recurrence of a particular health issue, engineering, time until the failure of equipment or part of the machinery equipment, in social studies, time until recidivism (re-arrest), in education field time until the dropout, in the economic field time until recession or loan repayment and many more. Due to various reasons, survival data happenstance the problem of censorship where some instances have incomplete information. Consequently, survival data analysis has been achieved in various ways, from traditional to machine learning approaches (Lee *et al.,* 2018).

Traditional approaches to survival analysis are organized into three groups: the non-parametric approach, the semi-parametric approach, and the parametric approach (Wang *et al.,* 2017). The non-parametric approach to survival analysis features the most traditional techniques, including the Kaplan Meier, also known as the product limit estimator, due to Kaplan and Meier (1958). KM is used to estimate the population survival function. The life table method, which computes the population survival curve in the case of large population size, is another non-parametric technique (Fine & Gray, 1999). The Nelson Aalen estimator is a counting process technique in the pack of non-parametric approaches to obtain the population hazard function.

The non-parametric approaches to survival analysis are suitable when the proportional hazards assumption fails to hold or when the underlying distribution for the event time is unknown (Gupta et al., 2019). Although these techniques assist the estimation of population survival functions and furnish light to the field of survival analysis, they do not involve any covariates apart from the survival time.

The semi-parametric approach to survival analysis has played a prominent role in modeling time-to-event data. The CPH model is a semi-parametric approach to survival analysis most popular

and widely adopted technique by survival model practitioners across different disciplines to analyze time-to-event data (Pénichoux *et al.,* 2015). Several variations of this model are available in the literature to cater to the needs, including the classic Cox model, Cox boost, regularized Cox, and time-dependent Cox model. Further, regularized Cox branches into Elastic Net (EN-Cox), Lasso Cox, Ridge Cox, and OSCAR Cox (Wang *et al.*, 2017).

These models utilize parametric and non-parametric techniques to produce more consistent estimators over a wide range of conditions. Despite its outstanding achievement, the CPH model suffers from a solid proportional hazards assumption and diminishing performance when modeling complex survival data (Lee et al., 2020).

The alternative to non and semi-parametric approaches to survival analysis is the parametric method encompassing techniques such as linear regression and accelerated failure time (AFT) models (Yao et al., 2017). Linear regression includes the Tobit model, Buckley-James model, and penalized regression in terms of weighted regression and structured regularization (Wang *et al.*, 2017). The parametric survival models assume a known distribution for the survival time. Most popular distributions, such as Exponential, Gompertz, Weibull, Logistic, Log-logistic, Normal, and Log-normal distributions, are assumed for the survival time (Wang *et al.*, 2017). Despite good performance in modeling survival data, the parametric approach is more restrictive to the prior knowledge of the distribution for the survival time which may lead to bias and inefficient estimates when the data fails to follow the desired distributions (Rietschel *et al.,* 2018).

### 2.2. Machine Learning Approach to Survival Analysis

Before the revival of deep learning in recent years, researchers across various disciplines have observed the prosperity of artificial intelligence (AI) systems, which significantly, have become the primary substitute for most traditional approaches. Despite the remarkable achievement of machine learning techniques in most areas, its application to survival analysis has lagged due to the difficulties imposed by censored information and the paucity of survival data. In today's literature, there exist several machine learning approaches to survival analysis which include but are not limited to survival trees, support vector machine (SVM), ensemble techniques such as

bagging survival trees, boosting, and random survival forest (RSF), artificial neural network and Bayesian approach (Wang et al., 2017). Although machine learning models have achieved high performance across different disciplines, in survival analysis, the CPH model has significantly competed with most of the above approaches, especially for simple survival tasks (Coccia, 2020).

### 2.3. The Revival of the Deep Learning Era

Recent technological advancement has facilitated more research in the deep learning field, and the results are outstanding. For the past decade, researchers have witnessed the breakthrough in artificial neural networks that led to remarkable transformation across various fields due to the application of deep learning technology. In the medical field, specifically cancer-related research, powerful classification models have been attained by training a deep neural network (Coccia, 2020).

In the survival analysis discipline, researchers are now leveraging AI systems to achieve reasonable accuracy, which is significantly better than the traditional approaches. Advanced deep learning techniques exist for survival analysis problems, including multi-task, active, and transfer learning (Wang et al., 2017). Below we briefly discuss the achievements attained so far by applying deep learning to survival analysis.

A deep learning technique called CRESA is introduced to model the probabilistic association between the inputs and outcomes distribution for recurring multiple events. This model is developed by starting with a single risk (stacked Long Short Term Memory (LSTM)), and the generalization to multiple risks follows. This model leverages the CIF by computing the joint distribution over the event times per competing risk across time steps (Gupta et al., 2019).

Despite good results compared to the baseline models, it needs to fully account for the complex correlational structure imposed by the recurrent events with competing risks. A Dynamic-DeepHit is an AI approach to complex survival problems designed specifically for the case of longitudinal data. This model successfully addressed the limitations of standard approaches to survival data

with competing risks such as joint modeling and land-marking (Lee et al., 2020). Despite the drastic improvement in discrimination power for complex events, this model is designed for the case of longitudinal data.

A temporal (multiple time points) multi-tasks learning framework (MTMT) for survival analysis problems that utilizes tensor representation is introduced by (Wang et al., 2020). Applying the survival dataset, the authors generated a model at every time point in a sequence, and the optimizations were jointly performed to ensure the sharing of standard features across tasks. Although this model has the added advantage of dynamically monitoring the survival status of instances on multiple events over time by computing the task-specific survival functions, it does not account for the case of recurrent events with competing risks.

In addition, the optimization techniques involve tedious tensor manipulations, and the model is only partially free from traditional assumptions. A Deep Recurrent Model for Survival Analysis (RNN-SURV) leverages the RNN for time-to-event data. This model is designed to exploit the survival data to compute the risk score and the survival distribution of each sample (Giunchiglia et al., 2018).

The authors applied several real datasets and comparisons using the concordance index (C-index) shows its superiority against the selected benchmark models. Despite the excellent performance and ability to personalize the treatment per patient, it is unsuitable for more complex survival data, such as recurrent events with competing risks.

A deep learning approach to survival analysis with competing risks called DeepHit is introduced to overcome the parametric assumptions made on traditional approaches to survival analysis. This model can handle the survival time predictions in the presence of competing risks.
The DeepHit not only smoothly models the competing risk scenarios but also handles time-variant covariates (Lee et al., 2018).

Using a time-dependent concordance index, the model was tested and compared with several states of the art models. Despite good performance, it cannot handle the case of recurrent events with

competing risks. A deep multi-tasks neural network that directly models the survival function instead of the hazard function to predict survival time for kidney graft patients is introduced by Luck et al., (2017). This model can be trained to learn survival time and rank in the Cox partial log-likelihood. Although the trained model was tested on various real datasets and outperformed state-of-the-art models based on the (C-index), it cannot handle the case of complex survival data such as recurrent events with competing risks.

The Deep Correlational Learning for Survival Prediction from Multi-Modality Data is introduced by Yao et al., (2017). The authors designed a deep network (DeepCorrSurv) to extract the most relevant features from pathological images and bio-molecular data by considering the correlational structure among the features. Despite its good performance, it is limited to more complex survival data, such as recurrent events with competing risks.

The Deep multi-task Gaussian process for survival analysis with competing risks is another application of AI to analyze survival data with multiple competing events. The authors viewed patients' survival times concerning competing risks as a result of a Deep Multi-task Gaussian Process (DMGP) and developed a non-parametric Bayesian model for survival data with competing risks Alaa et al., (2017). This method followed a Bayesian approach by parameterizing the vector-valued functions of the patient's characteristics and updating the posterior functions given the right-censored time-to-event data. The model was evaluated using both synthetic and real datasets. Comparison with the baseline techniques is achieved by using the cause-specific concordance index. Despite good performance, it is not wholly free from parametric assumptions. Also, the model cannot suffice in the case of recurrent events with competing risks.

The Deep Convolutional Network (CNN) for survival analysis with pathological data is introduced by Zhu et al., (2016). Due to the limitations of the traditional survival approaches, which use only the basics and handcrafted features, the authors designed a deep CNN for survival analysis. The design takes pathological images as input for the first time to improve prediction. Although the model shows a 12% increase in performance due to higher feature extraction using CNN, it is not suitable for more complex survival data, such as recurrent events with competing risks.

The limitations imposed by the traditional approach to survival modeling led to the introduction of a multi-task learning formulation for survival analysis. This approach involves modifying survival problems into a multi-task formulation where the multi-task learning approach is leveraged to predict the survival distribution (Li et al., 2016).

Although the model is trained on a gene expression dataset and outperformed several states of the art models upon comparison, it is not a deep learning technique, and it cannot manage the case of complex survival data. For asset health management, a study that combines deep learning and survival analysis is conducted by Liao et al., (2016). In this paper, the authors designed three layers of the deep neural network to simultaneously extract features from the input data using the LSTM architecture, which was stacked by another layer through the mean pooling to learn the representative features before the final layer attached to predict asset health condition. After application using both small and large datasets, the model showed promising results. Similar to most techniques discussed above, this model is not only inappropriate for the case of recurrent events with competing risks but also a solid parametric assumption concerning the survival time is used.

Despite remarkable achievements of deep learning techniques over the traditional methods across disciplines, the field of survival analysis seems to lag in employing these techniques, especially in the case of complex survival data. Most contributions to survival analysis using deep learning techniques focus on the simple survival data structure, and very little literature is available for the case of recurrent events with competing risks. As specified earlier, in practice, we always encounter complex survival data. For recurrent events with competing risk survival problems, particular features may account for the perseverance of a risk-specific outcome. In modeling this type of problem, it is necessary to design a network that accounts for the complex correlational structure among the covariates and addresses the dependency among the risk-specific outcomes.

Therefore, we propose a multi-tasks approach that uses an external autoencoder (ExternalAE), the Risks Information Weights (RIW) as an attention mechanism, and the residuals connections to address the risk-specific outcomes for the case of recurrent events with competing risks. The multi-task approach is suggested due to its success in various applications for handling complex data

(Lee et al., (2018)). The proposed model has multifold advantages, including adequately accounting for the dependency structure among the risk-specific outcomes, denoising the data to improve efficiency, accounting for the complex correlational structure among the covariates, and complete freedom from traditional assumptions.

# 3. MATERIAL AND METHODS

## 3.1. Recurrent Events with Competing Risks Survival Problem

In real-life applications, for instance, in medicine, the data structure is more complex. Every patient has a unique set of characteristics that lead to different responses to a particular treatment at a given time. In practice, health practitioners commonly encounter situations where patients receive treatment due to multiple co-occurrence health issues that repeat over time. Treating patients under such conditions may result in complications. For instance, various breast cancer treatments may accelerate cardiovascular diseases (Lee et al., 2018). Further, data generated through survival studies on such groups of patients is always complex.

The simplest method to analyze survival data with competing risk outcomes dates back to the earlier works by Kalbfleisch and Prentice (1984), which is a conventional approach using the CPH model by considering the competing events as censored under the strong independent assumption (Zhang, 2019). This approach is unrealistic due to the complex correlational structure among the competing events, which leads to inappropriate Kaplan-Meier estimates.

One of the alternatives to the cause-censoring techniques to compute the cause-specific risk probabilities in case of competing risks is to consider the cumulative incidence function (CIF) (Gooley et al., 1999). The CIF suffices in modeling survival data with competing risks, and an extension to account for the case of recurrent events with competing risks is archivable (Gupta et al., 2019).

However, like the rest of the traditional approaches to survival analysis, strong assumptions concerning the stochastic nature of the survival time are applied. Therefore, there is a need for survival model practitioners to apply AI techniques that are free from traditional assumptions.

## 3.2 Representation of Recurrent Events with Competing Risks Survival Problem

Like in any survival analysis problem, each subject has three essential components: the survival time $T^i$; the covariates vector $X^i \in R^p$, which can either be static or time-variant; and the label

$I^i$, which is an indicator function for an event or censoring. In this study, we assume none-informative right-censored data where the actual patient's survival time is longer than the observed time due to reasons such as loss of follow-up. To define our complex survival problem, we consider a set of mutual exclusive competing risks

$$K \in R^{k+1} = \{\emptyset, 1,2,3, \dots, K\},$$

where an individual subject experiences only one event at a particular recurrent time $T^i = t^i$ and $\emptyset$ is censorship.

To allow multitasks learning, we discretize the continuous survival time to a finite set denoted as $\{T_1, T_2, T_3, \dots, T_{max}\}$, where $T_{max}$ is the maximum time horizon predetermined at the study's commencement. At any time-step $T^i = t^i$, we define an event experienced by an individual subject using an indicator function

$$I = \{1, if\ \ T^i \leq C^i\ 0, if\ \ T^i > C^i\ \ \text{, where } C^i \text{ is the observed time.}$$

Assuming we have $N$ data points, we present the dataset for our complex survival problem as $\{X_t^i, T_t^i, I_t^i, R_t^i\}_{i=1}^N$ where $X_t^i \in R^p$ is the vector of covariates for an individual $i$ at the recurrent time step $t$, $T_t^i \in R^{T_{max}}$, is an observed event/censorship time for an individual $i$ at the recurrent time step $t$, $I_t^i$ is an indicator function evaluated at the time $t$ for an individual $i$, and $R_t^i$ is the event type occurred at the time $t$ for the subject $i$.

### 3.3. Analysis of the Complex Survival Problem

As an alternative approach to avoid strictly independent assumptions on the competing risks, the Cumulative Incidence Function (CIF) can appropriately assign the cause-specific probabilities to the respective class (Fine & Gray, 1999). Therefore, in this study, the CIF will be an essential estimator for the probabilities of the cause-specific events under the presence of competing risks. Conditioned on the covariates, Fine and Gray (1999) defined the risks-specific CIF as follows:

Let $T\ and\ C$ be the event and censoring times, $K \in R^{k+1} = \{\emptyset, 1,2,3, \ldots, K\}$ is the set $K$ observable risks, and let $X \in R^{p}$ be the feature vector. Considering the right-censored survival data, we can observe the survival time as $[min(T, C), \Delta = I(T \leq C]$, where $I(.)$ is an indicator function.

Given the set of independent identically distributed samples of size $n$ with the records $\{T_i, \Delta_i, \Delta_i k_i, X_i\}$, conditioned on the covariate vector $X$, we define the CIF for a specified cause $K = k^*$ as

$$F_{k^*}(t, X) = P(T \leq t, k^*|X) = \sum_{t^*=0}{}^{t} P(T = t^*, k^*|X) \tag{3.1}$$

Let $\{W_t\} \in R^{l}$ be the sequence of real-valued vectors whose elements are such that: $\sum_i \frac{e^{w_{t_i}}}{\sum_j \ e^{w_{tj}}} = 1$, we define the weighted cumulative incidence function (WCIF) of an instance with risk $k^*$ at a particular recurrent time step $t^*$ as

$$F_{k^*,t^*}(T_t, U_t) = P(T_t \leq t^*, K = k^*|U_{t^*}, (T_t \geq t^*) \cup ((T_t \leq t^*) \cap K \neq k^*)) \tag{3.2}$$

which can be evaluated as

$$F_{k^*,t^*}(T_t, U_t) = \sum_{s_t=0}{}^{t^*} P(T_t = s_t, K = k^*|U_{t^*}, (T_t \geq t^*) \cup ((T_t \leq t^*) \cap K \neq k^*)) \tag{3.3}$$

where the quantity $(T_t \geq t^*) \cup ((T_t \leq t^*) \cap K \neq k^*)$ represents the risk set, $U_t = W_t \odot X_t$ and $\odot$ is the Hadamard product. Therefore, the risk-specific probability will be given by an Equation 3.3. For further details and derivation of the WCIF, see Section 6.

# 4. NEURAL NETWORKS FOR SURVIVAL ANALYSIS

Neural networks can be used in survival analysis to model the relationship between a set of input variables and the time until the event of interest occurs. These models are called survival neural networks, and they can be used to make predictions about the survival time of new patients or to identify important factors that influence survival. One popular type of survival neural network involves the CPH model where the risk score function is learned via a neural network. Learning the weights of the CPH model using a neural network allows the auto-modeling of complex relationships between the input variables and the survival time, and it also allows for the modeling of censored subjects (Alaa & van der Schaar, 2017).

An alternative technique to survival neural networks is deep survival analysis, which leverages the deep learning framework suitable for survival problems. Deep neural networks can be used for both survival and event prediction. Overall, neural networks can be useful for survival analysis because they can model complex relationships between input variables and survival time, and handle censoring, making predictions more accurate (Farragi & Simon, 1995).

## 4.1. The Feedforward Neural Networks

F2eedforward neural networks, also known as multilayer perceptrons (MLPs), are the building blocks of deep learning models. The MLPs are used to approximate any arbitrary non-linear function $\Omega_\theta : x \to y$ by learning the parameters $\theta \in \Theta$ through input-output mapping. They are called feedforward because information flows in one direction only, i.e., from the inputs $X$ to the outputs $Y$.

They are called networks because they are composed of several functions described with a directed acyclic graph, i.e., $\Omega_\theta(x) = \Omega_0(\Omega_1(\Omega_2 \dots (\Omega_{n-2}(\Omega_{n-1}(\Omega_n(x))))))$, where $n$ is the number of layers in the network and $\Omega_i$, is the $i^{th}$ layer (Goodfellow et al., 2016). In order to understand the mechanisms of the feedforward neural networks, it is better to start with the linear model $f_\theta : x \to y$ and generalize using the non-linear transformation kernel $\Psi_\theta$.

Consider the nonlinear function $\Psi_\theta: R \to R$; let $(W, b) \in \Theta$ be the network's weights where $W \in R^{n,m} and\ b \in R^m$. We can express the dense layer of the feedforward neural network as a non-linear transformation $f_\Theta: R^n \to R^m = \Psi(W^T x + b)$. The official name of the transformation function $\Psi$ is called the activation function.

To obtain the MLPs, we combine several dense layers using composite functions. Suppose $F_\Theta: R^n \to R^m$ is an MLP with $n$ layers; then it can be expressed as $F_\Theta(x) = f^0_{\Theta_0} \circ f^1_{\Theta_1} \circ f^2_{\Theta_2}, \ldots, , f^{n-1}_{\theta n-1} \circ f^n_{\Theta_n}$, where the number of layers $n$ determines the depth of the network. The intermediate layers, $(1,2,3, \ldots, n-1)$ are hidden layers, the final layer, $n$, is the output layer, and the layer $0$ is the input layer. Training of this network involves the optimization of parameters $(W, b)$ and hyperparameters turning such as the depth of the network $n$, number of neurons in the layer, and the type of activation function $\Psi$.

The architecture in Figure 4.1 is the simplest form of a feedforward neural network with a single hidden layer to solve the exclusive OR (XOR) problem. The classical XOR problem consists of two binary inputs, which evaluate "True" if the values are different and "False" when the values are similar (Brutzkus & Amir, 2019). The left-hand side graph consists of individual neurons linked together to present the mapping $f_\Theta: x \to y$, while the right-hand side is the compact form with the individual nodes representing a layer.

There is a wide choice of non-linear transformation functions ($\Psi$) available in the literature. We can select the activation function based on the task at hand, such as classification, regression, data generation, and many more. We select activation functions in the intermediate layers in favor of the model's convergence. Some of the most used activation functions in practice include the sigmoid (logistic) activation, the hyperbolic tangent ($tanh$) activation, the rectified linear unit (ReLU) activation, and its variants, and the softmax activation function.

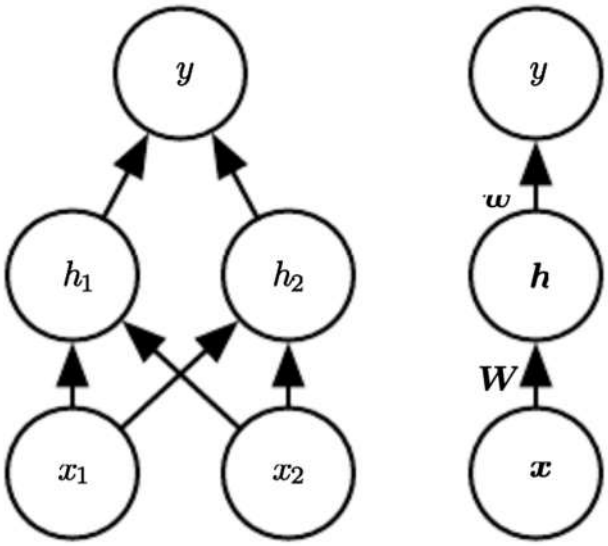



Figure 4.1. The feedforward neural network for the XOR problem

Considering the random vector $X \in R^n$, we have the following mathematical definitions concerning the activation function: We obtain the sigmoid activation as

$$f: R^n \to R^n; f(x) = \frac{e^x}{1+e^x}.$$

The hyperbolic tangent activation is given as

$$f: R^n \to R^n; f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

The rectified linear unit (ReLU) activation is given as

$$f: R^n \to R^n; f(x) = max(0, x) = x^+,$$

and the softmax activation is given as

$$f: R^n \to R^n; f(x) = \frac{e^{x_i}}{\sum_j \quad e^{x_j}}$$

### 4.2. Recurrent Neural Networks

For the past decade, artificial neural networks have been demonstrated to achieve magnificent performance over traditional approaches and other machine learning algorithms on tasks such as classification, image segmentation, regression, data generation, features extraction, data denoising, image inpainting, and many more. Most of these tasks are accomplished by using deep learning models with convolution layers, i.e., deep convolutional neural networks (CNN) and feedforward neural networks (MLPs) architectures. Although these networks can be applied in modeling temporal sequences, the overall performance remains substandard due to the complex correlation structure in sequential data (Erizmann, 2021).

Therefore, recurrent neural networks (RNN) are specifically designed to handle temporal correlation structure in sequential data (Rumelhart et al., 1986). The RNN helps to identify patterns in a sequence or temporal data such as texts, speech, video, time series, signals, genomics data, survival data, and many more (Manaswi, 2018).

The RNN has been successfully utilized across different disciplines to model temporal data, and remarkable achievements are seen in the field of Natural language processing (NLP), specifically machine translation, speech recognition, texts, and music generation, to mention a few (Lee *et al*., 2018).

The performance of the RNN model on sequential data is guided by its unique ability to store memory and allow parameter sharing across the time steps, which enables the model to generalize well in practice. The neurons of the RNN model can keep track of information using the state vector ($h_t$) at every step.

At each time step, the RNN neuron receives an input sequence ($X_t$) and the previous state vector ($h_{t-1}$) to compute the current state vector ($h_t$). Let $\Psi$ be the activation function, ($T_x, T_y$) be the lengths of input $\{X_t\}$ and output $\{Y_t\}$ sequences, respectively, for each layer in an RNN we can

compute the following quantities:$h_t = \Psi_1(W_x^T x_t + W_h^T h_{t-1} + b_1)$ and $y_t = \Psi_2\left(W_y^T h_t + b_2\right)$ where $h_t \in R^{T_x}$, $y_t \in \{Y_t\}_{t=1}^{T_y}$ and $x_t \in \{X_t\}_{t=1}^{T_x}$ are the hidden states, the output, and the input vectors at a time $t$. The parameters $\Theta = [W_x, W_h, b_1, b_2]$ are shared across all time steps.

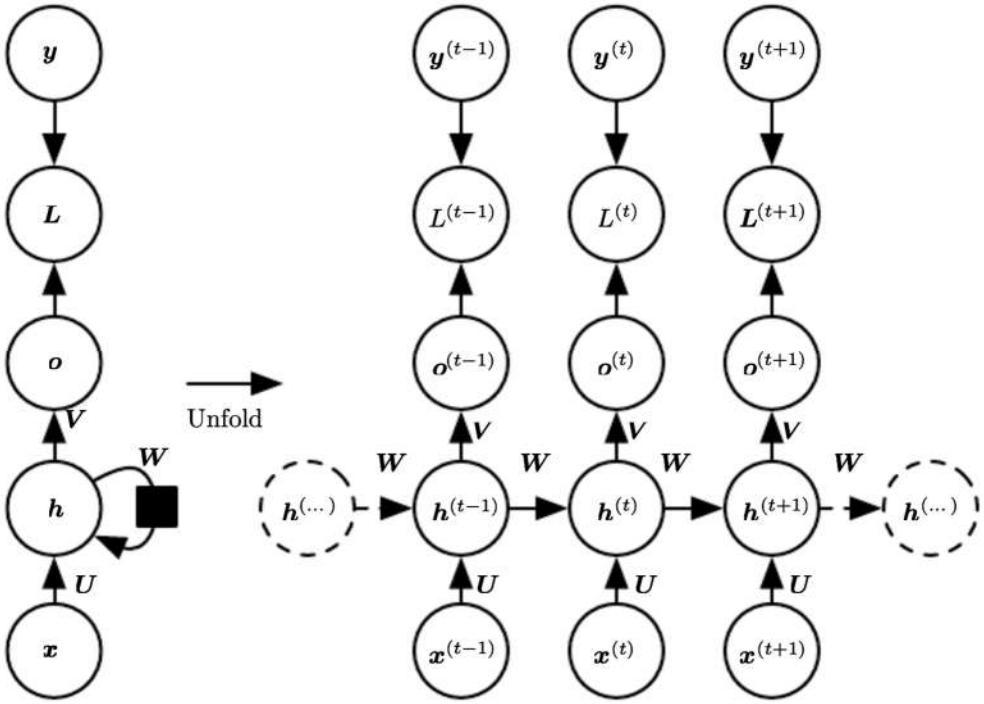



Figure 4.2. The recurrent neural network architecture

Figure 4.2 displays the computational graph of the RNN. The left-hand side graph is the compact form (the unrolled) with the cyclic arrow to represent re-currency. The right-hand side graph is a long version of the RNN architecture where the network is unrolled through time to become similar to MLPs. Notice that the weights ($\Theta = W$) are shared across all time steps. Here the activation function for the hidden states and output states are given by ($\Psi_1 = U$) and ($\Psi_2 = V$). Depending on the task the output vector ($y$) can be computed at every time step or suppressed until the end of the sequence.

There are various input-output structures of the RNN computational graphs conditioned on the task at hand. In the case of sequence-to-sequence models such as time series forecasting and

machine translation, both inputs and outputs are sequences (Erizmann, 2021). Further, the input can be a sequence, and the output can be a vector, like the case of sentiment analysis, to predict customer satisfaction or opinion towards a particular product.

Also, the vector-to-sequence input-output relationship can be observed for tasks such as language or music generation and image captioning. The general format of the input-output relationship can be achieved using the encoder-decoder architecture, where the sequences are allowed to take any shape.

In practice, we always encounter sequences with elements of different lengths. Either padding can address this problem according to the longest element in the sequence or truncation according to the shortest element in the sequence (Erizmann, 2021). Due to parameter redundancy and length of the sequence, the vanilla RNN suffers much from the problem of gradient decay or explosion during training. This is because the backpropagation (BPP) algorithm operates over the unrolled version of the RNN architecture, which led to the multiplication of gradients with similar magnitudes. If these gradients are less than one, we have a decaying gradient issue, and the network will stop training. On the other hand, if these gradients are higher than one, we encounter the problem of gradient explosion.

These issues are addressed using several techniques, including applying gated recurrent units (Goodfellow et al., 2016). The most prominent gated RNN architectures used in practice are the Long Short Term Memory (LSTM) and the Gated Recurrent Unit (GRU) (Ravanelli et al., 2018). The LSTM cell stores two state vectors associated with temporally and long-term memories to keep track of information flows across time steps in longer sequences.

At every step t, the LSTM cell outputs state vectors $C_t$ and $h_t$. Using state vectors, the LSTM can automatically ignore or preserve past information when updating the next stage of the network, which is achieved using several gates that create paths through time to enable a steady flow of derivatives.

At every time step, the LSTM cell conducts several operations across different gates, namely the input gate ($i_t$), the forget gate ($f_t$), and the output gate ($o_t$). These gates consist of simple feedforward sub-networks. The name "forget" is associated with the sigmoid activation function, which emits higher probability values to open the gate and low probability values to close the gate. The following computations are performed within the LSTM cell at each time step:

- The input ($X_t$) is scaled using the hyperbolic tangent ($tanh$) activation function as

$$Z_t = tanh(W_{zx}^T X_t + W_{zh}^T h_{t-1} + b_z).$$

- The computation at the forget gate($f_t$) is expressed as

$$f_t = \sigma\left(W_{xf}^T X_t + W_{hf}^T h_{t-1} + b_f\right).$$

- The computation at the input gate ($i_t$) is given as

$$i_t = \sigma(W_{xi}^T X_t + W_{hi}^T h_{t-1} + b_i).$$

- The computation at the output gate ($o_t$) is given as

$$o_t = \sigma(W_{xo}^T X_t + W_{ho}^T h_{t-1} + b_o).$$

- We obtain the state vectors ($C_t, h_t$) as

$$C_t = f_t \odot C_{t-1} + i_t \odot Z_t \quad \text{and} \quad h_t = o_t \odot tanh(C_t), \text{ respectively.}$$

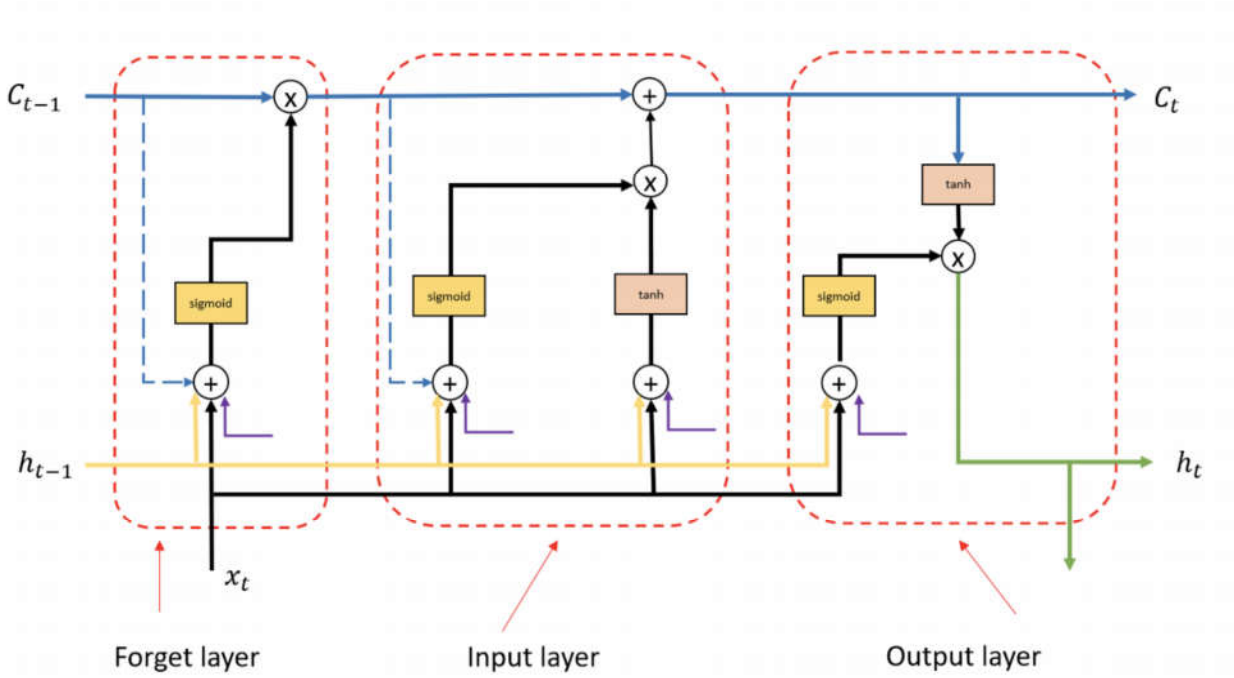



Figure 4.3. The LSTM cell architecture (*Long Short-term Memory - Wikipedia*, 2022)

Notice that every gate has a unique set of trainable parameters to be shared across the time steps given as

$$[W_{zx}, W_{zh}, W_{xf}, W_{hf}, W_{xi}, W_{hi}, W_{xo}, W_{ho}, b_z, b_f, b_i, b_o]$$

The state vectors are obtained using the Hadamard product of respective components. Also, the derivatives of the state vector i.e., $\left[\frac{\partial C_t}{\partial C_{t-1}} = f_t\right]$ do not directly depend on the network parameters which makes LSTM efficient during training (Goodfellow et al., 2016).

The complex architecture of the LSTM can be simplified by suppressing the number of computations. This is achieved by the introduction of the gated recurrent unit (GRU). The GRU cell controls the flow of gradients via the reset gate $(r_t)$ and the update gate $(Z_t)$ (Cho et al., 2014).

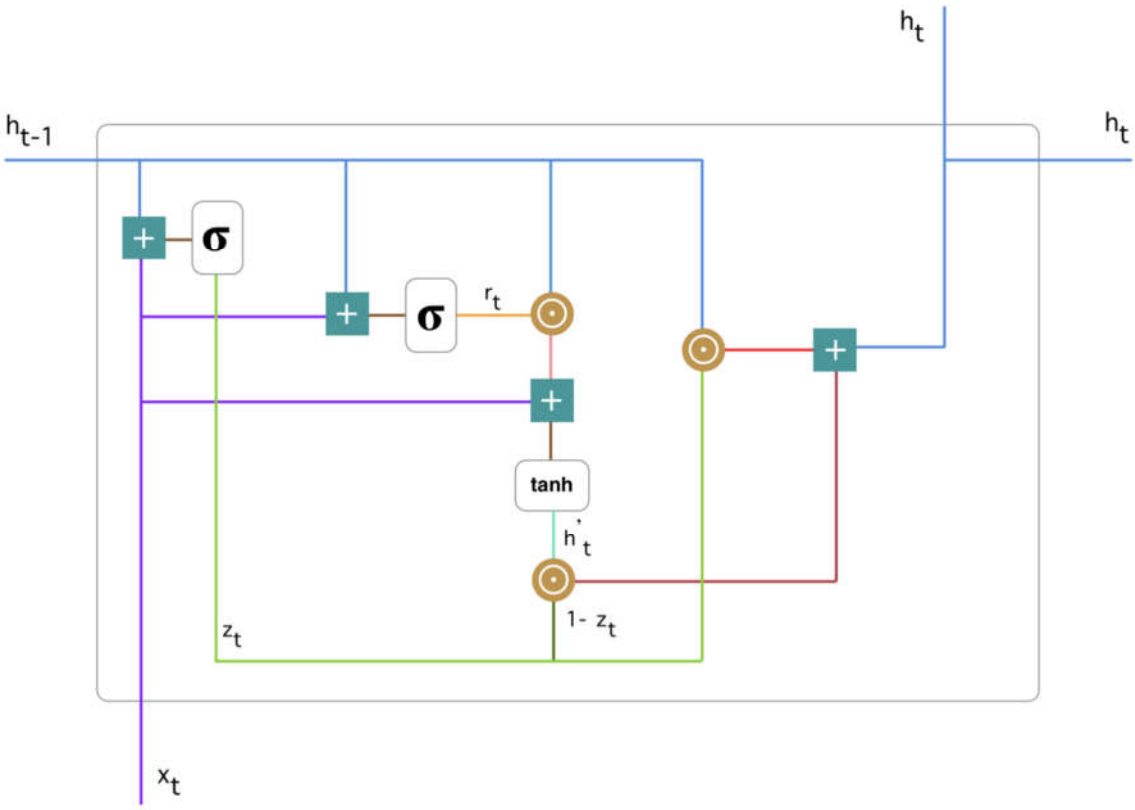



Figure 4.4. The GRU cell architecture (*Gated Recurrent Unit - Wikipedia*, 2016)

The GRU contains a single state vector that propagates information across the sequence. Using the non-linear transformation functions $\sigma$ and $tanh$. We carry out the following computations within the GRU cell:

- In the update gate, we have

$$Z_t = \sigma\left[W_{zx}^T X_t + W_{hz}^T h_{t-1} + b_z\right].$$

- We compute the reset gate as

$$r_t = \sigma[W_{rx}^T X_t + W_{hr}^T h_{t-1} + b_r].$$

- The state candidate is given as

$$C_t = tanh[W_{xc}^T X_t + W_{hc}^T (h_{t-1} \odot r_t) + b_c] .$$

- We obtain the output vector as

$$h_t = Z_t \odot h_{t-1} + (1 - Z_t) \odot C_t.$$

The update gate combines and filters information received from the input sequence at the current time with the output received from the previous GRU cell. In practice, the GRU provides almost equal performance to the LSTM. Due to the flexibility of these architectures, there are many variants of both GRU and LSTM in the literature (Erizmann, 2021).

### 4.3. Autoencoders (AE) networks

Autoencoders are a unique form of neural network architectures that are widely applied for various tasks, including feature extraction, dimensionality reduction, data reconstruction, and data generation (Bank et al., 2020). The great advantage of autoencoders is their ability to be embedded into the primary model and concurrently trained to accomplish the given task. Autoencoders have two network systems: the encoder (E) and the decoder (D). The encoder $E: R^n \rightarrow R^d, d \leq n$ maps the input vector of size $n$ into the code space of size $d$, while the decoder $D: R^d \rightarrow R^n$ maps the code space to decode the original information. The desired task is accomplished by optimizing the cost of the form $J_\theta = \eta(X, D \circ E(X))$ (Erizmann, 2021).

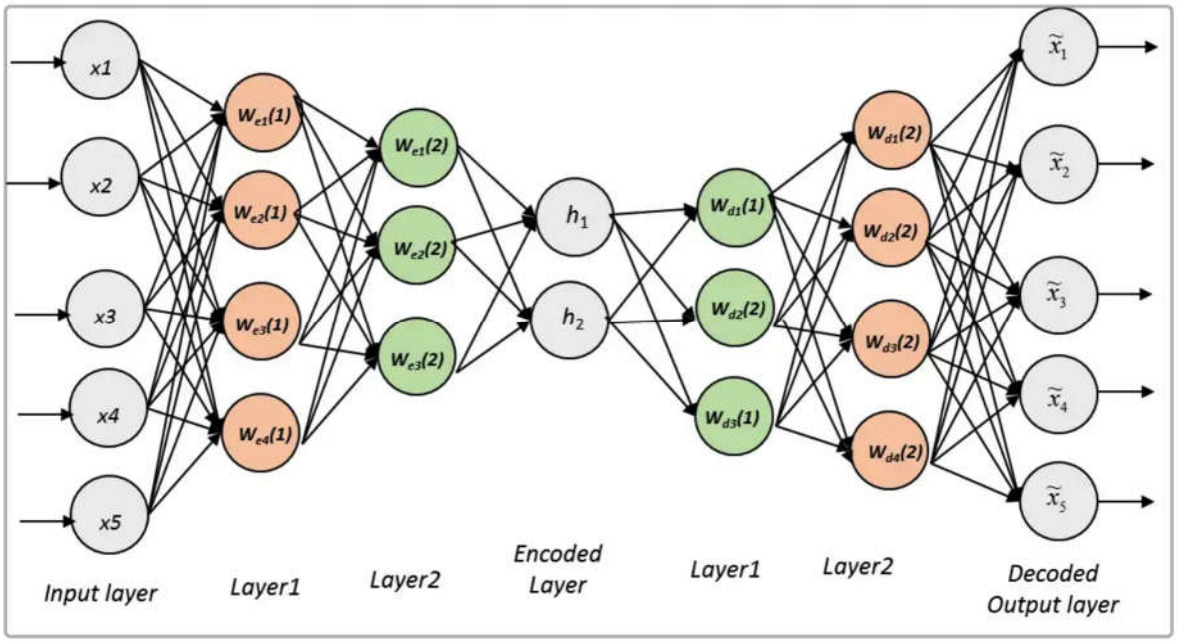



Figure 4.5 The autoencoder architecture (Jonnalagadda, 2018)

In Figure 4.5, the left-hand side of the encoded layer constitutes the encoder's network while the right-hand side is the decoder's network.

### 4.3.1. Feature selection with Autoencoders

Like in other machine learning models, deep learning models for survival analysis may suffer from performance deficits due to noise and redundant information in the dataset (Carl *et al*., 2019). Selecting the most relevant features is necessary to reduce computation complexity and gain efficiency in training machine learning models.

Several feature selection techniques are available in the literature, including filter and wrapper algorithms and the embedding approach. Filters and wrappers are independent of the primary model, while the embedding approach allows selection and training procedures to be parallel (Jundong et al., 2016).

In this study, we embed the autoencoder as an unsupervised feature selection algorithm into the primary network to facilitate feature selection. Given the unlabeled data matrix $X \in R^{(m,n)}$, $m$ is the number of samples, and $n$ is the feature size. The task is to select the set of representative features, say $(p \leq n)$, that are useful for training our machine learning model. This autoencoder feature selector leverages the simple autoencoder network with the MLPs architecture proposed by Wang et al., (2017). For the detail of this network see Section 5.

### 4.4. Attention Mechanism

The attention mechanism is a popular term in the deep learning arena, particularly for the family of encoder-decoder networks. By allowing the network to learn the alignments between inputs and outputs, significant performance is achieved for the task related to neural machine translation (NMT) (Bahdanau & Cho, 2014), speech recognition (Chorowski et al., 2015), image segmentation (Sinha & Dolz, 2020), image captioning (Huang et al., 2019), time series forecasting (Du et al., 2020) and many more.

There are several forms of attention mechanisms based on the alignment score function. To improve the performance of the basic encoder-decoder in NMT, the additive (soft) attention technique, which allows the network to align and translate, is introduced by Bahdanau & Cho, (2014).

To obtain the conditional probability of the target sequence $P(y_i|y_{i-1}, y_{i-2}\ldots, y_2, y_1, X) = g(y_{i-1}, s_i, c_i)$ for a machine translation problem, the authors introduced an attention mechanism that uses the context vector

$$C_i = \sum_j^{T_x} \alpha_{i,j} h_j$$

and the alignment model is given as

$$e_{i,j} = A(s_{i-1}, h_j)$$

where $A$ is the feed-forward neural network, $(h_j, s_i)$ is the hidden states of the RNN, $C_i$ is the context vector at $t_i$, $T_x$ is the length of the input sequence, $y_{t_i}$ is the target at $t_i$ and

$$\alpha_{i,j} = \frac{exp(e_{i,j})}{\sum_{k=1}^{T_x} exp(e_{i,k})}$$

are $P(y_i|x_j)$ used to extract information from the input sequence. The decoder network generates predictions based on this information by paying attention to the specific part of the input sequence. These weights are learned using the feed-forward neural network, which is trained with the rest of the system. Another form of attention mechanism is content-based (hard) attention with the alignment score function given as

$$g(s_t, h_i) = S_c(s_t, h_i) = cos(s_t, h_i),$$

which is the cosine similarity between the decoder and encoder hidden states (Mnih *et al*., 2014). Luong et al., (2015) modified Bhadanau's attention by computing the context vector at both global and local levels. They leverage the concepts of soft and hard attention to obtain what is termed as global and local attention. The global attention considers all the hidden states of the encoder to derive the context vector $(C)$.

The alignment model is

$$A_{T_x} = g(s_t, h_{t_x}) = \frac{exp\left(score(s_t, h_{T_x})\right)}{exp\left(score(s_t, h_{T_x}')\right)},$$

where $s_t$ is the decoder's hidden state, and $h_{T_x}$ is the encoder's hidden state. The score function $score(a, b)$ can be computed as $a^T b$, which is a dot product attention also given as $a^T W b$, where $W$ is the transformation matrix if vectors $a$ and $b$ are of different dimensions. The general attention is given as $v^T tanh(W[a \oplus b])$ which is similar to Bhadanau's attention where the alignment model learns $(v, W)$.

To reduce computation complexity, the local attention, which aligns only on the part of the encoder's hidden states per target, is considered. In this case, the network selects the alignment $a_t$ for each target, and the context vector $C_t$ is computed by considering the small differentiable window $[a_t - D, a_t + D] \in R^{2D+1}$ where $D$ is empirically selected.

The alignment score function for the local attention can be computed based on monotonic assumption or predicted using a sigmoid transformation. For more computational details, see Luong et al., (2015). More development on the attention mechanism include the scaled dot product attention, which leads to the transformers networks (Vaswani et al., 2017).

Transformers networks are completely based on the stacked self-attention (intra-attention) mechanism for both encoder and decoder networks. The scaled dot product attention is based on three quantities, namely query (Q), which is the vector of feature values; key (K), which represents a feature contribution; and value (V) which is the vector corresponding to each input element. Here the score function takes query and keys as inputs to produce probability vectors which determine the contribution of each input element to the target.

The weights are obtained as $\frac{([QK^T])}{d_k}$, where $d_k$ is a scale factor and [.] is the dot product. We can compute the scaled dot product attention as,

$$Attention(Q, K, V) = Softmax\left(\frac{(QK^T)}{d_k}V\right)$$

This is similar to the dot product attention in computational complexity when the scale factor is negligible. To allow the network to attend to the information at different subspace positions jointly, the multi-head attention can be leveraged (Vaswani et al., 2017). This is achieved by several linear projections on the key, query, and value vectors along their respective dimensions, which are learned differently.

Attention is computed in parallel on each projected vector. Outputs are combined and projected once more to get the final results. Multi-head attention is given as

$$MultiHead(Q, K, V) = [head_1 \oplus head_2 \oplus head_3 \oplus \ldots, \oplus head_h]W^0$$

and

$$head_i = Attention\left(QW_i^Q, KW_i^K, VW_i^V\right),$$

where

$$W_i^Q \in R^{(d_{model}, d_k)}, \quad W_i^K \in R^{(d_{model}, d_k)}, \quad W_i^V \in R^{(d_{model}, d_v)}, \text{ and } \quad W^0 \in R^{(hd_v, d_{model})}$$

are parametric matrices that are directly learned by the network. For our network, we consider the global attention by Luong et al., (2015), which is similar to additive (Bahdanau & Cho, 2014) attention. We consider the query, key, and value vectors as

$$Q = K = V = [C_t \oplus X_t] \tag{4.4.1}$$

for $X_t \in R^{T_x, d_h}$ and $C_t \in R^{T_x, 2*d_h}$, where $d_h$ is the hidden size of the RNN and $\oplus$ is the concatenation operation. The alignment vector is given as

$$A_t = g(X_t, C_t) = \frac{exp(score(X_t, C_t))}{exp\left(score(X_t, C_t{}')\right)},$$

where $score(.)$ is an MLPs.

# 5. THE PROPOSED NETWORK ARCHITECTURE

The proposed architecture consists of four components. The external autoencoder (ExternalAE), the shared sub-network, which is an LSTM block, the self-attention, and the risk-specific sub-networks, which are MLPs networks (Marthin & Tutkun, 2023). The central architecture is shown in Figure 5.1, while Figure 5.2 shows the autoencoder in detail.

Given the sequence of feature vectors $\{X_t\}_{t=1}^{l} \Longrightarrow X \in R^{n \times l}$ where $l$ is the sequence length, and $n$ is the feature size at time $t$. We partition the input vectors into categorical and numerical sets as $[\{X_t\}_1^l : \{X_t^{cat}\}_1^l ; \{X_t^{num}\}_1^l]$. The task is to select the set of representative features say $(p \leq n)$, that are useful for training our machine learning model.

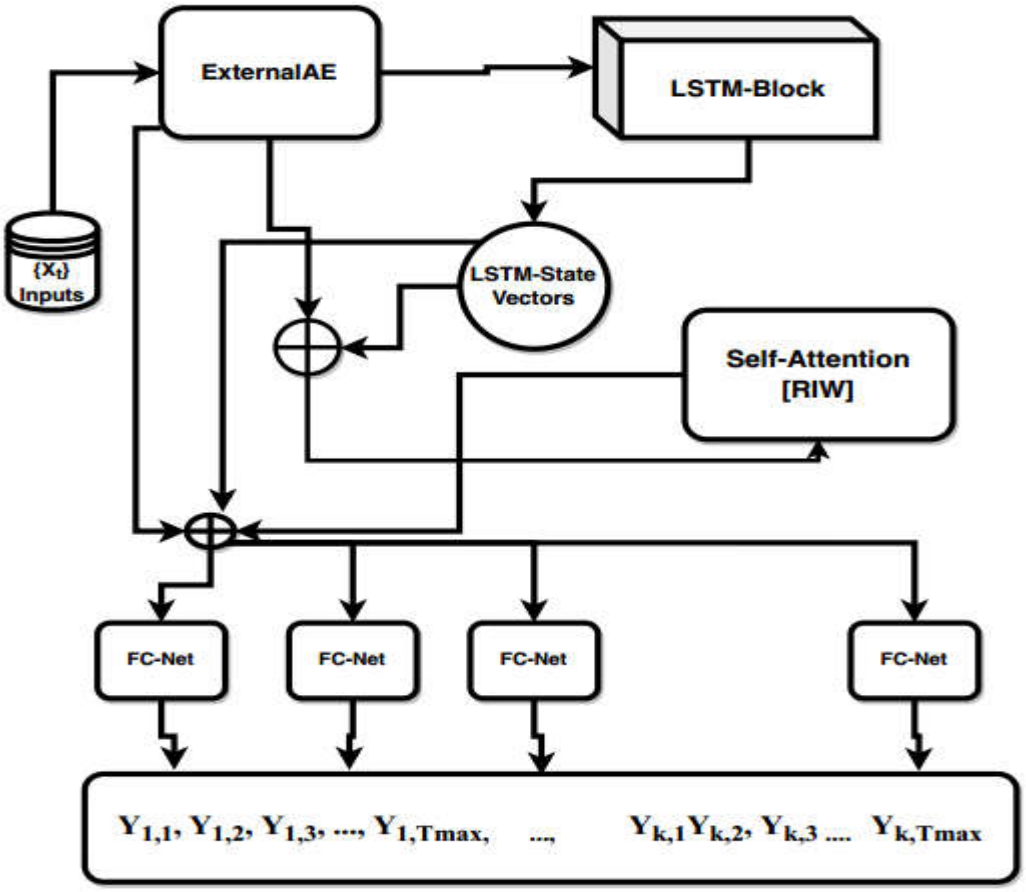



Figure 5.1. The general architecture

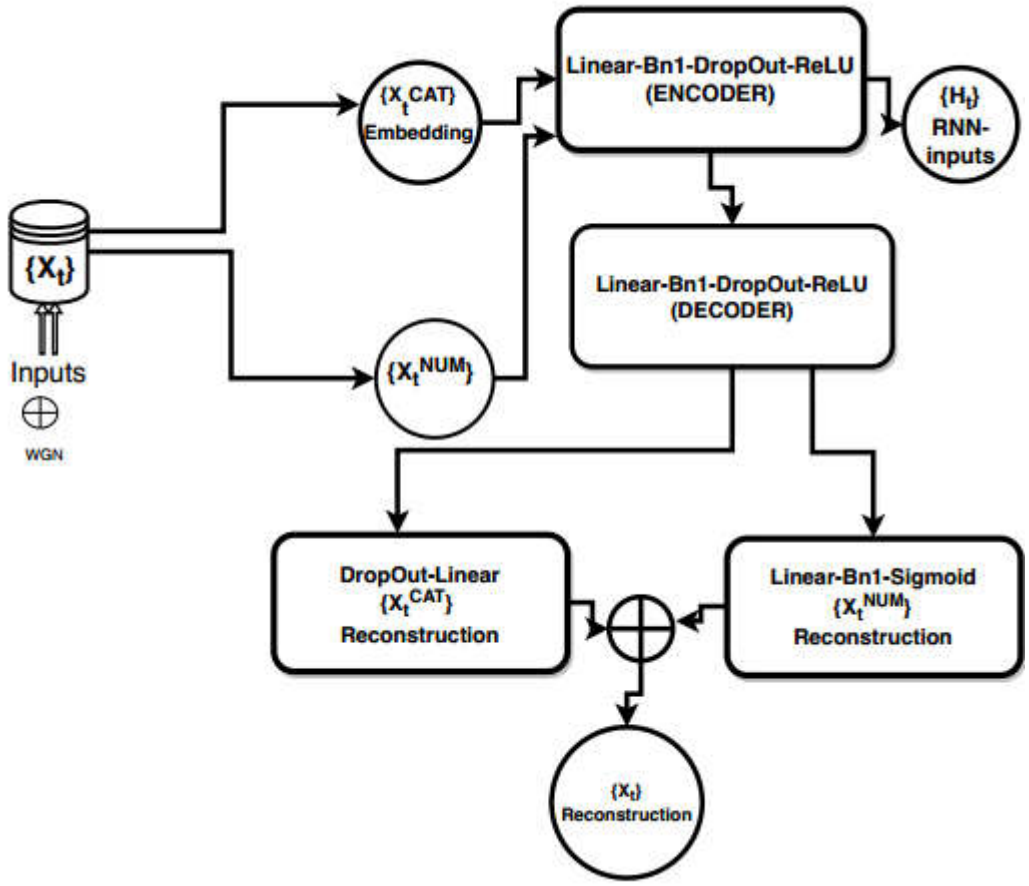



Figure 5.2. The external autoencoder (ExternalAE) architecture

The encoder network is computed as follows:

$$\{Z_t^{cat}\} = \{\eta_t\} + \{X_t^{cat}\} \Longrightarrow E_t^1 = f_1(Z_t^{cat}) = \delta_t^1(Z_t^{cat}, W_t^1) = H_t^{cat} \tag{5.1}$$

$$\{Z_t^{num}\} = \{\eta_t\} + \{X_t^{num}\} \Longrightarrow E_t^2 = f_2(Z_t^{num}) = \delta_t^2(Z_t^{num}, W_t^2) = H_t^{num} \tag{5.2}$$

The decoder network is given as

$$D_t^1 = g_1(f_1(Z_t^{cat})) = g_1(H_t^{cat}), D_t^2 = g_2(f_2(Z_t^{num})) = g_2(H_t^{num}) \tag{5.3}$$

which produces the reconstruction outputs as

$$\{\hat{X_t^{cat}}\} = D_t^1 \circ (E_t^1) = g_1(H_t^{cat}) = \delta_t^3(f_1(Z_t^{cat}), W_t^3) \tag{5.4}$$

$$\{\hat{X_t^{num}}\} = D_t^2 \circ (E_t^2) = g_2(H_t^{num}) = \delta_t^4(f_2(Z_t^{num}), W_t^4) \tag{5.5}$$

where $\{W_t^1, W_t^2, W_t^2, W_t^4 \in \theta\}$ are the weights of the encoder-decoder's MLPs given as $\delta_t^1, \delta_t^2, \delta_t^3, \delta_t^4$ respectively, and $\{\eta_t\}$ is the white Gaussian noise (WGN) randomly introduced during training to improve the reconstruction loss [14].

The RNN input is given as $\{\hat{X}_t\} = H_t^{cat} \oplus H_t^{num} \in R^{p \leq n}$, where $\oplus$ is the concatenation operator. In the architecture, we maintain the input notation $(X_t)$ to ease the notation.

The autoencoder network is trained by minimizing the loss function given as:

$$L_t^{ae} = L_t^{num} + L_t^{cat} \text{ , where}$$

$$L_t^{num} = \frac{1}{2m}\left|\left|X_t - \hat{X_t}\right|\right|_F^2 + \alpha\left|\left|W_t^2; W_t^4\right|\right|_{2,1} + \frac{\beta}{2}\sum_{i=1}{}^2 \left|\left|W_t^i\right|\right|_F^2 \tag{5.6}$$

$$L_t^{cat} = \left\{l_t^1, l_t^2, l_t^3, \ldots, l_t^q\right\}^T where l_t^i = -\omega_{c_t^i}\left[\frac{log(exp(x_t^i, y_t^i)}{\sum_{c_t=1}^{C_t}(exp(x_t^i, c_t))}\right] \tag{5.7}$$

where $m$ is, the batch size and $||.||_F$ is the Frobenius norm for matrices,

$\left|\left|W^*\right|\right|_{2,1} = \sum_{i=1}^{n}\sqrt{\sum_{j=1}^{h}(W_{i,j}^*)^2}$ is the sparsity term to filter redundant information and $\frac{\beta}{2}\sum_{i=1}^{2}\left|\left|W_t^i\right|\right|_F^2$ is the weight decaying factor to reduce overfitting and speed the convergence process. $L_t^{cat}$ is the weighted cross entropy loss where the weights $(\omega_{c_t^i})$ are proportional to the class scores of each categorical feature and $(x_t^i, y_t^i)$ are the ground truth and reconstructed values. The shared sub-network consists of two stacked layers of RNN with an LSTM architecture. All forms of RNN architecture, like vanilla RNN and the GRU, are used for experimentation.

The LSTM receives the hidden representation of the external autoencoder ($\{\hat{X_t}\}$) as input. The following computations are performed within an LSTM cell at each time step:

- The input $\hat{X_t}$ is scaled using the hyperbolic tangent activation function ($tanh$) as

$$Z_t = tanh\left(W_{zx}^T \hat{X}_t + W_{zh}^T h_{t-1} + b_z\right)$$

- The computation at the forget gate ($f_t$) is given as

$$f_t = \sigma\left(W_{xf}^T \hat{X}_t + W_{hf}^T h_{t-1} + b_f\right)$$

- The computation at the input gate ($i_t$) is given as

$$i_t = \sigma\left(W_{xi}^T \hat{X}_t + W_{hi}^T h_{t-1} + b_i\right)$$

- The computation at the output gate ($o_t$) is given as

$$o_t = \sigma\left(W_{xo}^T \hat{X}_t + W_{ho}^T h_{t-1} + b_o\right).$$

- The state vectors ($C_t, h_t$) are given as

$$C_t = f_t \odot C_{t-1} + i_t \odot Z_t,\ h_t = o_t \odot tanh(C_t) \text{ respectively.}$$

The input to the attention network is given as $\{V_t\} = \left[\hat{X}_t \oplus h_t\right] \Longrightarrow V \in R^{2p \times l}$. The RIW is given as

$$RIW = argmax_\theta g(V_t) = W_t \ni \sum_i \frac{e^{w_{t_i}}}{\sum_j e^{w_{tj}}} = 1, W_t \in R^l \tag{5.8}$$

where the score function $g_\theta(.)$ is simple $MLPs$ and $\sigma$ is the sigmoid function. The empirical $WCIF$ is given as

$$\hat{F}_{k^*,t^*}(T_t, X_t) = f_{\theta \in \Theta}\left([W_t \odot \hat{X}_t] \oplus [h_t \oplus \hat{X}_t\right) \tag{5.9}$$

where $\odot$ is the Hadamard product, $W_t$ is computed by Equation 5.8, $\hat{X}_t = H_t^{cat} \oplus H_t^{num}$, and $f_{\theta \in \Theta}$ is the $MLPs$ with the softmax outputs which is the final layer of our network.

# 6. THE LOSS FUNCTION

The loss function is an essential component of a training loop and plays a prominent role in the model's performance. In this study, we focus on minimizing the loss function that accounts explicitly for the complexity of recurrent events with competing risks under the presence of non-informative right censoring. Our loss function consists of several components.

The main component is the negative partial log-likelihood loss ($L_l$) which is specifically designed to accommodate the case of recurrent events with competing risks, the ranking loss ($L_r$), which act as a penalty factor for the miss-classification of the risks-specific outcomes, and the reconstruction loss ($L_a$) which is given in Equation 5.6 and Equation 5.7 above to train our feature selector. We obtain the total loss as $L_t = \beta_1 L_l + \beta_2 L_r + \beta_3 L_a$, where the network learns $[\beta_1, \beta_2, \beta_3]$ directly.

The single event (here described as simple) case survival problem, the CPH (Cox, 1972) model, can be expressed as $h_{\theta \in \Theta}(t|X) = h_0(t)e^{X^T\theta}$ where $h_0$ is the baseline hazard, $\theta \in \Theta$ are the weights/parameters, and $X$ is the vector of covariates. Ignoring the baseline hazard $h_0(t)$, we can achieve a consistent estimator $\hat{\theta}$ using the random sample $[T_i, \delta_i, X_i]$ where $T_i$ is an observed censoring/survival time for an individual, and $\delta_i$ is the indicator function with $\delta_i = 1$ for an event and $\delta_i = 0$ for censoring.

Let us consider the set of $K$ distinct events which occur according to their respective times; the risk set (risk-free individuals by time $t$ is given as $R_t = \{i: T_i \geq t\}$. For uncensored data, the conditional probability of the observed failures/events given the risk set constitutes an essential part of the likelihood function. At each distinct time $T_i$, we obtain the contribution to the likelihood function as

$$
\begin{aligned}
&L_i: \theta \in \Theta \\
&= P[i^{th} instance\ fails\ given\ that\ there\ is\ at\ least\ one\ instance\ in\ the\ risk\ set R_t] \\
&= \frac{P[instance\ i\ fails\ given\ that\ is\ at\ risk\ by\ time\ T]}{\sum_{l \in Rt} P[instance\ l\ fails\ given\ that\ is\ at\ risk\ by\ time\ T]}
\end{aligned} \tag{6.1}
$$

By intuition, this means the probability of an individual developing an event given that he/she was event-free up to that particular time. In terms of the hazard function, we can write Equation 6.1 as

$$L_i: \theta \in \Theta = \frac{h(T_i|X_i)=P[T_i \leq t_i|X_i]}{\sum_{l \in R(T_i)} h(T_i|X_l)} . \tag{6.2}$$

Following (Cox, 1972, 1975), under the CPH assumption, the likelihood function for the $K$ distinct events can be written as

$$L_{\theta \in \Theta} = \prod_{i=1}^{K} \left[ \frac{h_0(t_i) e^{X_i^T \theta}}{\sum_{l \in R(T_i)} h_0(t_l) e^{X_l^T \theta}} \right] = \prod_{i=1}^{K} \left[ \frac{e^{X_i^T \theta}}{\sum_{l \in R(T_i)} e^{X_l^T \theta}} \right]. \tag{6.3}$$

Since the survival data always contains some incomplete records due to censorship, we express the likelihood function in a form that accounts for this information. We cannot record any event for a censored instance, but partial information concerning survival time is available (in our study, we assume that censorship is observable).

For the case of simple survival data, the likelihood function for a right-censored instance corresponds to the survival function, i.e., $L_i: \theta \in \Theta = S_i(T_i)$. On the contrary, if an instance develops an event/failure at time $T_i$, we obtain the likelihood function assuming non-informative right censored data as $L_i: \theta \in \Theta = S_i(T_i|X_i) h_i(T_i|X_i)$.

Therefore, we can write the partial likelihood function for a non-informative right censored survival data can as

$$L_{\theta \in \Theta} = \prod_{i=1}^{n} [h_i(T_i|X_i)]^{\delta_i} [S_i(T_i)]. \tag{6.4}$$

We can also include the risk-set information in the likelihood function to have it as

$$L_{\theta \in \Theta} = \prod_{i=1}^{n} \left[ \frac{(h_i(T_i))}{\sum_{l \in R(T_i)} h_l(T_i)} \right]^{\delta_i} \left[ \sum_{l \in R(T_i)} h_l(T_i) \right]^{\delta_i} S_i(T_i) \tag{6.5}$$

where $\sum_{l \in R(T_i)} h_l(T_i)$ is the risk set. Since the first term of Equation 6.5 is the sufficient statistic for $\theta \in \Theta$, under the proportional hazard assumption we have the likelihood function given as

$L_{\theta \in \Theta} = \prod_{i=1}^{n} \left[ \frac{(h_i(T_i))}{\sum_{l \in R(T_i)} h_l(T_i)} \right]^{\delta_i}$, which simplifies to

$$L_{\theta \in \Theta} = \prod_{i=1}^{n} \left[ \frac{h_0(t_i) e^{X_i^T \theta}}{\sum_{l \in R(T_i)} h_0(t_i) e^{X_l^T \theta}} \right]^{\delta_i} = \prod_{i=1}^{n} \left[ \frac{e^{X_i^T \theta}}{\sum_{l \in R(T_i)} e^{X_l^T \theta}} \right]^{\delta_i} \tag{6.6}$$

where $\theta \in \Theta$ is the vector of parameters, $X$ is the covariates vector, $\delta_i$ is an indicator function for the event if $(\delta_i = 1)$ and censorship if $(\delta_i = 0)$, and $h_0(t_i)$ is the baseline hazard.

For the cause-specific events (events with competing risks), we are interested in the distribution of the time to failure due to a specified risk under other competing risks. Let us assume two competing failures $(k_1, k_2) \in K$ and their possible failure times $(T_1, T_2)$ respectively. In practice, for a particular instance, we usually record the $min[T_1, T_2]$ assuming the events are mutually exclusive. In this case, we can consider the crude death rate (hazard) or the cumulative incidence rate as the parameter of interest. We compute the crude death rate for a subject due to a particular risk $(K = k^*)$, given that an instance was alive past time $(t)$ as

$$h_{k^*}(t_i) = \lim_{\Delta_{t_i} \to 0} \left[ \frac{P(t_i < T_i \leq t_i + \Delta_{t_i}, K = k^* | T_i \geq t_i)}{\Delta_{t_i}} \right] \tag{6.7}$$

which is also known as instantaneous failure potential for the $i^{th}$ instance due to risk $k^*$ within an infinitesimal time change. We can obtain the cumulative incidence function (CIF) for a subject among the group of instances present in the risk set at that particular time point as $C_{k^*}(t) = P[T \leq t, K = k^* | R_k]$, where $R_k$ is the risk set, including the $k^{th}$ candidate. Considering all subjects in a dataset/batch, we have the relation $C_{total} + S_t = 1$ where $S_t$ is the chance that neither individual developed a risk over the entire study duration.

Fine and Gray (1999) considered the CIF as the sub-distribution function to include covariates in the model. We can express this sub-distribution hazard function as the ratio of the derivative of cause-specific probability to the survival distribution

$$h_{k^*}(t) = \lim_{\Delta_t \to 0} \frac{P[t<T\leq t+\Delta_t, K=k^*]}{\Delta_t} = \left[\frac{\frac{\partial C_{k^*}(t)}{\partial t}}{1-C_{k^*}(t)}\right] \tag{6.8}$$

Replacing the usual baseline hazard component with the cause-specific probability/sub-distribution hazard, we obtain the model suitable for competing-risk events as $h_{k^*}(t) = h_{0,k^*}(t)e^{X^T\theta}$ with $h_{0,k^*}$ in the form of Equation 6.7 (Fine & Gray, 1999). Consider the tuple $(t_i, \delta_i, k_i, x_i)_{1=1}^{n}$ where $t_i$ is an event or censorship time for the $i^{th}$ subject, $\delta_i$ is an indicator function, $k_i$ is a risk-specific event, and $x_i$ is a covariate vector. If we assume none-informative right censored survival problem with mutually exclusive events, we have the likelihood function for the $K$ competing risks written as

$$L_{\theta\in\Theta} = \prod_{i=1}^{n}\left[h_{k_i}(t_i,x_i)^{\delta_i}S_k(t_i,x_i)\right] = \prod_{i=1}^{n}\left[h_{k_i}(t_i,x_i)^{\delta_i}\prod_{k=1}^{K}e^{-H_k(t_i,x_i)}\right] \tag{6.9}$$

where $H_k(t_i, x_i)$ is the CIF for the $i^{th}$ subject. This means that the subject censored at time $t$ will have the survival information up to that point given as

$$S(t_i,x_i) = \prod_{k=1}^{K}e^{-H_k(t_i,x_i)}$$

otherwise the contribution to the likelihood function by that instance will be equivalent to the death density expressed in terms of survival and hazard functions as

$$h_{k_i}(t_i,x_i)S_{k_i}(t_i,x_i) = h_{k_i}(t_i,x_i)\prod_{k=1}^{K}e^{-}H_k(t_i,x_i) \tag{6.10}$$

which is equivalent to the joint likelihoods for individual risks.

In practice, we usually encounter a scenario where the event of interest is not necessarily death or disappearance from the study (Gupta *et al.,* 2019). Therefore, subjects are exposed to multiple risks which may repeat several times (recurrence) over the study's duration. For the case of a single recurrent event, several traditional approaches have been introduced in the literature.

The most straightforward approach is the counting processes which applies the usual CPH model with the assumption that subjects with more than one interval (occurrence) are treated independently as if they are different examples. The partial likelihood function is the product of individual functions at each ordered time point. When recurrent events are viewed differently at each time step, the marginal approach technique, similar to CIF, can be employed.

As the generalization to recurrent events with competing risks survival problem, we extend the CIF concept for the case of competing risk to contain recurrent events scenario. This is achieved by conditioning the risk-specific outcomes across the time steps. We can obtain the cause-specific probability for a specified event $K = k^*$ for a subject across multiple time stamps conditioned on the covariates and risk set as

$$C_{k^*}(t^*|x_t^*, k_t^*) = \sum_{t=0}^{t^*} P[T = t, K = k^*|R^*, X_{t^*}] \tag{6.11}$$

where $R^* = \{(i \in (T_i \geq t^*) \cup (T_i \leq t^* \cap (K \neq k^*)\}$ is the risk set which consists of survivors from the risk $k^*$ and those who developed other forms of events prior to time $t^*$, and $X_{t^*}$is the covariate vector at time $t^*$. This can further simplifies to

$$C_{k^*}(t^*|x_t^*, k_t^*) = \frac{\sum_{t=0}^{t^*} P[T=t, K=k^*|X_{t^*}]}{1-\sum_{K\neq\emptyset}\sum_{t=0}^{t^*} P[T=t, K=k^*|R^*, X_{t^*}]} \tag{6.12}$$

where the denominator of Equation 6.12 is the survival information for all uncensored subjects in the risk set by time $t^*$.

We can express the survival distribution for a specified subject concerning the particular risk as the compliment of the cause-specific probability for events only

$$S_{k^*}(t^*|x_t^*, k_t^*) = 1 - C_{k^*}(t^*|X_t^*, k_t^*) = 1 - \frac{\sum_{t=0}^{t^*} P[T=t, K=k^*|X_{t^*}]}{1-\sum_{K\neq\emptyset}\sum_{t=0}^{t^*} P[T=t, K=k^*|R^*, X_{t^*}]} \quad (6.13)$$

Following Equation 6.9, the likelihood function for the case of recurrent events with competing risks can be generalized. This is achieved by conditioning on the recurrence time with the maximum sequence length of $(T)$ using the sample of size $n$ as follows

$$L_{\theta\in\Theta} = \prod_{i=1}^{n}\sum_{t=0}^{T}\left[h_{k_i}(t_i, X_{t_i})\right]^{\delta_i} S_{k_i}(t_i, X_{t_i}) = \prod_{i=1}^{n}\sum_{t=0}^{T}\left[h_{k_i}(t_i, X_{t_i})\right]^{\delta_i}\prod_{k=1}^{K} e^{-H_{k_i(t_i, X_{t_i})}},$$

which simplifies to

$$L_{\theta\in\Theta} = \prod_{i=1}^{n}\prod_{k=1}^{K}\sum_{t=1}^{T}\left[h_{k_i}(t_i, X_{t_i})\right]^{\delta_i} e^{-H_{k_i(t_i, X_{t_i})}} \quad (6.14)$$

Since it is mathematically contented to work with the log-likelihood function, we can express Equation 6.14 as

$$l_{\theta\in\Theta} = \sum_{i=1}^{n}\sum_{k=1}^{K} log\left[\sum_{t=1}^{T}\left[h_{k_i}(t_i, X_{t_i})\right]^{\delta_i} e^{-H_{k_i(t_i, X_{t_i})}}\right],$$

which simplifies to

$$l_{\theta\in\Theta} = \sum_{i=1}^{n}\sum_{k=1}^{K} log\left[\sum_{t=1}^{T} \left[h_{k_i(t_i, X_{t_i}))}\right]^{\delta_i}\right] - \sum_{i=1}^{n}\sum_{k=1}^{K} H_{k_i(t_i, X_{t_i})} \quad (6.15)$$

Following similar steps, we define the weighted cause-specific probability for a particular event $(K = k^*)$ recorded at a given time stamp conditioned on the covariates and risk set of an individual as

$$C_{k^*}(t^*|u_t^*, k_t^*) = \sum_{t=0}^{t^*} P[T = t, K = k^*|R^*, U_{t^*}] \quad (6.16)$$

where $R^* = \{i \in (T_i \geq t^*) \cup (T_i \leq t^* \cap (K \neq k^*\}$ is the risk set, $U_{t^*} = W_{t^*} \odot X_{t^*}$ is the weighted covariate vector at the time $t^*$ with $W_t \in R^l$ is a real-valued vector whose elements are such that: $\sum_i \frac{e^{w_{t_i}}}{\sum_j e^{w_{tj}}} = 1$.

This simplifies further to gives

$$C_{k^*}(t^*|U_t^*, k_t^*) = \frac{\sum_{t=0}^{t^*} P[T=t, K=k^*|U_{t^*}]}{1-\sum_{K\neq\emptyset}\sum_{t=0}^{t^*} P[T=t, K=k^*|R^*, U_{t^*}]} \tag{6.17}$$

where the denominator of an Equation 6.17 is the survival information for all uncensored subjects in the risk set by time $t^*$. The weighted overall survival ($WOS$) for an instance concerning a particular event is the complement of the weighted cause-specific probability for events only. This can be written as

$$S_{k^*}(t^*|U_t^*, k_t^*) = 1 - C_{k^*}(t^*|U_t^*, k_t^*) = 1 - \frac{\sum_{t=0}^{t^*} P[T=t, K=k^*|U_{t^*}]}{1-\sum_{K\neq\emptyset}\sum_{t=0}^{t^*} P[T=t, K=k^*|R^*, U_{t^*}]} \tag{6.18}$$

Equations 6.17 and 6.18 are the quantities of interest for our study. Since we have yet to learn the nature of probability distributions for the survival time and the complex events interactions, the closed-form solutions cannot be evaluated analytically. Therefore, empirical estimates $\hat{C}_{k^*}(.)$ and $\hat{S}_{k^*}(.)$ are obtained using Equation 5.9.

Following Equation 6.14, the likelihood function with weighted covariates for the case of recurrent events with competing risks can be generalized. This is achieved by conditioning on the recurrence ($T$) using the sample of size $m$ as follows:

$$L_{\theta\in\Theta} = \prod_{i=1}^{m} \sum_{t=0}^{T} \left[h_{k_i}(t_i, U_{t_i})\right]^{\delta_i} S_{k_i}(t_i, U_{t_i}) =$$

$$\prod_{i=1}^{m} \sum_{t=0}^{T} \left[h_{k_i}(t_i, U_{t_i})\right]^{\delta_i} \prod_{k=1}^{K} e^{-H_{k_i(t_i, U_{t_i})}} \tag{6.19}$$

$$= \prod_{i=1}^{m} \prod_{k=1}^{K} \sum_{t=1}^{T} \left[h_{k_i}(t_i, U_{t_i})\right]^{\delta_i} e^{-H_{k_i(t_i, U_{t_i})}} \tag{6.20}$$

Since it is mathematically contented to work with the log-likelihood function, we can express Equation 6.20 as

$$l_{\theta\in\Theta} = \sum_{i=1}^{m} \sum_{k=1}^{K} log\left[\sum_{t=1}^{T} \left[h_{k_i}(t_i, U_{t_i})\right]^{\delta_i} e^{-H_{k_i(t_i, U_{t_i})}}\right] \tag{6.21}$$

which simplifies to

$$l_{\theta\in\Theta} = \sum_{i=1}^{m}\sum_{k=1}^{K} log\left[\sum_{t=1}^{T} \left[h_{k_i(t_i,U_{t_i}))}\right]^{\delta_i}\right] - \sum_{i=1}^{m}\sum_{k=1}^{K} H_{k_i(t_i,U_{t_i})} \tag{6.22}$$

Since in machine learning, we usually minimize the loss function, then equivalently, instead of maximizing the above partial log-likelihood, we reduce the negative partial log-likelihood given as

$$-l_{\theta\in\Theta} = -\left[\sum_{i=1}^{m}\sum_{k=1}^{K} log\left[\sum_{t=1}^{T} \left[h_{k_i(t_i,U_{t_i}))}\right]^{\delta_i}\right] - \sum_{i=1}^{m}\sum_{k=1}^{K} H_{k_i(t_i,U_{t_i})}\right] \tag{6.23}$$

which simplifies to

$$-l_{\theta\in\Theta} = L_l = -\sum_{i=1}^{m}\left[(\delta_i = 1) log\left[\frac{\sum_{t=0}^{t^*} P\left[T=t,K=k^*|U_{t^*},R^*\right]}{1-\sum_{k\neq\emptyset}\sum_{t=1}^{t^*} P[T=t,K=k^*|U_{t^*},R^*]}\right]\right] + [(\delta_i = 0) log[1 - \frac{\sum_{t=0}^{t^*} P\left[T=t,K=k^*|U_{t^*},R^*\right]}{1-\sum_{k\neq\emptyset}\sum_{t=1}^{t^*} P[T=t,K=k^*|U_{t^*},R^*]}]] \tag{6.24}$$

Equation 6.24 constitutes the loss function's main component ($L_l$) necessary to optimize our neural network.

### 6.1 The Ranking Loss ($L_r$)

The ranking loss is the miss-classification penalty necessary to regulate the network for wrong risk-specific event predictions. Ideally, the ranking loss is based on the comparison approach analogous to the concept of the concordance index (C-index) (Lee *et al.*, 2018). This component regulates the network by penalizing the incorrectly ordered pair of instances with a high risk (subjects who developed an event by time $t$) compared to those who remained event-free past time $t$. Using the C-index formulation.

We compute the $L_r$ loss by leveraging the risk-specific probabilities given by the estimated CIF in Equation 5.9. We consider the pared individuals $(i, j)$ as a matching pair for a specified risk $(K = k^*)$ at the time $t^*$ given that the subject $j$ has not developed the event until time $t^*$, i.e., $(t_j^* > t_i^*)$. Under this condition, we expect the estimated risk probabilities (CIF) to satisfy the rule

$$\hat{C}_{k^*}(t_i^*, |X_{t_i^*}) \geq \hat{C}_{k^*}(t_i^*, |X_{t_j^*}) \tag{6.25}$$

We compute the loss among the acceptable pairs of subjects as follows:

$$L_r = \sum_{t^*=1}^{T} E_{k^*,t^*} \sum_{i \neq j}^{n} G_{k_{i,j}^*} \Phi \left[ \hat{C}_{k^*}(t_i^*, |X_{t_i^*}), \hat{C}_{k^*}(t_i^*, |X_{t_j^*}) \right] \tag{6.26}$$

where $G_{k_{i,j}^*}$ is an indicator function of the form

$$G_{k_{i,j}^*} = \{1 \; if (t_j^* > t_i^*, K = k^*), 0 \; if (t_j^* \leq t_i^*, K = k^*) \tag{6.27}$$

$E_{k^*,t^*}$ is the hyper-parameter to trade off the ranking loss for the $k^{th}$ risk with other competing risks, and $\Phi(.)$ is a convex kernel. For our case, we compute the weighted ranking loss by conditioning on the weighted covariates. Under this condition we expect the estimated risk probabilities (WCIF) to satisfy the rule

$$\hat{C}_{k^*}(t_i^*, |U_{t_i^*}) \geq \hat{C}_{k^*}(t_i^*, |U_{t_j^*}).$$

For our network, we compute the $L_r$ loss with weighted covariates among the acceptable pairs of subjects is as

$$L_r = \sum_{t^*=1}^{T} \epsilon_{k^*,t^*} \sum_{i \neq j}^{m} G_{k_{i,j}^*} \Phi \left[ \hat{C}_{k^*}(t_i^*, |U_{t_i^*}), \hat{C}_{k^*}(t_i^*, |U_{t_j^*}) \right]. \tag{6.28}$$

For convenience we set $\epsilon_{k^*,t^*} = \epsilon$ across all time steps and $\Phi(a, b) = e^{-(a-b)}$ which is an exponential smoothing kernel. The reconstruction loss $(L_{ae})$ for the autoencoder is given as

$$L_{ae} = \sum_{t=1}^{T} [L_t^{num} + L_t^{cat}] \tag{6.29}$$

where $L_t^{num}$ and $L_t^{cat}$ are given in Equations 5.6 and 5.7.

### 6.2 Performance Metrics for Survival Models

Unlike other machine learning models, assessing the classifier's performance for the survival analysis task requires special techniques to account for the event times and their respective probabilities (Longato *et al.*, 2020). Traditional metrics, such as the area under the receiver operating characteristics (ROC) curve, fail to quantify the correlation between the predicted outcomes and observed survival times.

We address this issue by using the concordance index (C_index), which relies on the idea of a matched-mismatched (concordant-discordant) pair of observations. The C-index is a comprehensive metric that summarizes the relationship between event times and risk probabilities to aid model evaluation and selection (Wang *et al.*, 2019).

Consider the pair of random subjects $(i, j)$ with the risk scores $(R_i, R_j)$ and the survival times $(T_i, T_j)$. We compute the C-index for a single-event survival problem as

$$\text{C-index} = P\left[R_j > R_i | T_i > T_j\right] \tag{6.30}$$

This number expresses the relationship between the risk score of a subject and its respective event time. The above relation implies that the subject $j$ is at higher risk of encountering an event than the $i^{th}$ subject.

Therefore, in this manner, the higher values of the C-index imply the assignment of higher risk scores to subjects in the higher-risk category (Longato *et al.*, 2020). Harrell's estimator is the favored approach to estimate the C-index (Harrell *et al.*, 1982). Let $\hat{R_i}$ and $\hat{R_j}$ be the estimates of risk probabilities for the subjects $i$ and $j$ at times $(T_i, T_j)$, respectively.

We obtain Harrell's estimator for C-index with tied observations as the ratio between the number of matched pairs and the total comparable pairs as follows

$$\hat{C}_{index} = \frac{\sum_{i=1}^{n} \delta_i \sum_{i \neq j} \left[ I[T_j > T_i] + (1 - \delta_j)[T_j = T_i] \right] \left[ I(\hat{R}_j < \hat{R}_i) + \frac{1}{2}(\hat{R}_j = \hat{R}_i) \right]}{\sum_{i=1}^{n} \delta_i \sum_{i \neq j} [T_j > T_i]} \tag{6.31}$$

where $\delta$ is an event indicator, $I(.)$ is an indicator function to fetch the number of concordant pairs, and $n$ is the sample size. Without ties in the survival times and predicted risk probabilities, Equation 6.7 reduces to

$$\hat{C}_{index} = \frac{\sum_{i=1}^{n} \delta_i \sum_{i \neq j} I[T_j > T_i] I(\hat{R}_j < \hat{R}_i)}{\sum_{i=1}^{n} \delta_i \sum_{i \neq j} [T_j > T_i]} \tag{6.32}$$

It is easy to notice that if all the samples are correctly matched according to the predicted risks, i.e., the quantity $I(\hat{R}_j < \hat{R}_i)$ in the numerator gives 1 for all pairs, then the estimated C-index = 1, while if all pairs are discordant, then the estimated C-index = 0 and therefore, $0 \leq \hat{C}_{index} \leq 1$.

Also, the update of $\hat{C}_{index}$ stops when the last event is encountered (say at time $r$), and a clear indication of this event are necessary for practical application. The termination point of Harrell's $\hat{C}_{index}$ algorithm can be set to some lower value, say ($r^* < T_{max}$), to establish a split between the early ($T_{max} \leq r^*$) and later ($T_{max} \geq r^*$) stages of the data to guide the identification of hazardous individuals (Longato *et al.,* 2020). We can generalize Harrell's C-index algorithm to account for the complex survival data.

Consider the pair of subjects $(i, j)$ with the specified risk ($K = k^*$). Let $(\hat{R}_i, \hat{R}_j)$ be the predicted risk probabilities at time ($T = t^*$) for subjects $i$ and $j$, respectively. We define the risk-specific time-dependent concordant index as

$$C_{k^*}(t^*) = P\left[ (\hat{R}_i > \hat{R}_j) | (t^* | X_{j_t^*} > t^* | X_{i_t^*}), K = k^* \right]. \tag{6.33}$$

We can obtain the estimate of Equation 6.9 following Equation 6.7 as

$$\hat{C}_{k^*}(t^*) = \frac{\left[\sum_{i=1}^{n} \delta_i \sum_{i \neq j} G_{k^*_{i,j}} \left[t^*_j > t^*_i\right] G_{k^*_{i,j}} \left[\hat{R_i} > \hat{R_j}\right]\right]}{\sum_{i=1}^{n} \delta_i \sum_{i \neq j} G_{k^*_{i,j}} \left[t^*_j > t^*_i\right]} \tag{6.34}$$

The generalization of Harrell's C-index algorithm to account for the recurrent events with competing risks survival data is achieved by conditioning over the time stamps. For the recurrent events with competing risks scenario, it is essential to consider variation due to time-dependent covariates (Lee *et al.*, 2018).

Following Equations 6.7, 6.8, 6.9 and 6.10 we obtain the time-dependent concordant index with weighted covariates for our complex survival problem as follows:

- Consider the pair of subjects $(i, j)$ with the specified risk $(K = k^*)$.

- Let $(\hat{R_i}, \hat{R_j})$ be predicted $WCIF$ at the time $(T = t^*)$ for subjects $i$ and $j$ respectively. We define the weighted covariates' risk-specific time-dependent concordant index as

$$C_{k^*}(t^*) = P\left[(\hat{R_i} > \hat{R_j}) | (t^* | U_{j^*_t} > t^* | U_{i^*_t}), K = k^*\right]. \tag{6.35}$$

We estimate Equation 6.35 as

$$\hat{C}_{k^*}(t^*) = \frac{\left[\sum_{i=1}^{m} \delta_i \sum_{i \neq j} G_{k^*_{i,j}} \left[t^*_j > t^*_i\right] G_{k^*_{i,j}} \left[\hat{R_i} > \hat{R_j}\right]\right]}{\sum_{i=1}^{m} \delta_i \sum_{i \neq j} G_{k^*_{i,j}} \left[t^*_j > t^*_i\right]} \tag{6.36}$$

Using the estimate of the $WCIF$ from Equation 5.9 results to

$$\hat{C}_{k^*}(t^*) = \frac{\left[\sum_{i=1}^{m} \delta_i \sum_{i \neq j} G_{k^*_{i,j}} \left[\hat{C}_{k^*}(t^*_i | U_{t^*_i}) > \hat{C}_{k^*}(t^*_i | U_{t^*_j})\right]\right]}{\sum_{i=1}^{m} \delta_i \sum_{i \neq j} G_{k^*_{i,j}} \left[t^*_j > t^*_i\right]} \tag{6.37}$$

where $G_{k_{i,j}^*}$ is given by Equation 6.3, $\delta_i$ is an event indicator, and $m$ is the batch size.

Another popular metric used to evaluate the performance of survival models is the Brier score. The Brier score is a measure of the accuracy of probabilistic predictions (Weigel *et al.,* 2007). It is commonly used in the field of weather forecasting, but can also be applied to other types of probabilistic predictions, such as those made by machine learning models (Hu *et al.*, 2022).

The Brier score is calculated as the mean squared difference between the predicted probability and the actual outcome for a binary classification problem (Weigel *et al.,* 2007). The Brier score ranges from 0 to 1, with 0 representing a perfect score and 1 representing a completely incorrect score. The Brier score can also be extended to multi-class classification problems by averaging the score for each class.

The Brier score can be useful for comparing different probabilistic predictions, as it provides a single scalar value that summarizes the overall accuracy of the predictions. It can also be used to evaluate the performance of probabilistic classifiers. In addition, the Brier score can be decomposed into two terms: resolution and reliability. The resolution term measures the capability of the model to distinguish between events that are likely to happen and events that are unlikely to happen. The reliability term measures the consistency of the predicted probabilities with the observed frequencies (Hu *et al.,* 2022).

One of the main advantages of the Brier score is that it allows the evaluation of the performance of a probabilistic prediction in a way that can be directly related to the loss that would be incurred by using the predicted probabilities to make decisions.

For our study, we calculate the Brier score at every timestamp using the following formulation:

$$BR_t = \frac{1}{m}\sum_{i=1}^{m} I_t^i (\hat{S}_t^i - y_t^i)^2 \tag{6.38}$$

where $m$ is the batch size, $I_t^i$ is an indicator function for the subject $i$, $\hat{S}_t^i$ is the weighted survival prediction for the subject $i$, and $y_t^i$ is the actual outcome for the subject $i$.

Therefore, to assess and compare the performance of our network against the selected benchmark models, we employ Equations 6.13 and 6.14 as the performance measure.

# 7. TRAINING MACHINE LEARNING MODEL

Training refers to the process of optimizing the neural network parameters (Goodfellow *et al.,* 2016). Consider the neural network model ($M_\theta: X \to Y, \theta \in \Theta$) where $\{X, Y\} \in R^{(N, d_{model})}$, $\theta$ is trainable parameters belonging to parametric space $\Theta$, $N$ is the total number of subjects (batch size), and $d_{model}$ is the dimension of the feature vector.

Learning the parameters $\theta \in \Theta$ of the neural network involves the adjustment of these weights conditioned to the training data and patterns (Shang *et al*., 1996). Given the cost function $L_{\theta \in \Theta}$ and the training data $\{(X, Y): (x, y)\} \in R^{(N, d_{model})}$, we can empirically obtain the optimal weights by minimizing the cost function $L_{\theta \in \Theta}$.

Mathematically; we can express this operation as $\hat{\theta} = \underset{\theta \in \Theta}{argmin} L(M_\theta | (x, y))$. Since the parameters of the neural network exist in the high dimensional parametric space, obtaining the closed-form solution for the above optimization problem is not possible in practice. In addition, the nature of the data and the type of cost function increases the complexity in computation and interpretability, which makes the neural networks regarded as black-box models (Molnar *et al.,* 2020).

The commonly used procedure is an iterative technique that allows updating the parameters at every step. For the convex cost function $L_{\theta \in \Theta}$, we start by initializing the set of parameters to random values $\theta = \theta^0$ and subsequently obtaining the next solution as $\theta^i = \theta^{i-1} - \eta \quad_\theta L\left(M_{\theta_i} | (x^i, y^i)\right)$ where $\eta > 0$ is the learning rate parameter which determines the size of the step towards optimality and $\quad_\theta (.)$ is the partial derivative of the loss function with respect to the training parameters.

Following the function's gradient to the optimal point is called the steepest gradient descent since, after each step, the parametric vector descends towards the optimal space via the path with the highest decline in the cost function (Goodfellow *et al.*, 2016). The training data $\{(X, Y): (x, y) \in R^{(N, d_{model})}\}$ passes through the network $M_{\theta \in \Theta}$ at every iteration for the optimization process. This procedure is called the batch or deterministic gradient method and is achievable when the dataset is small enough to fit into the memory at once.

In practice, we always encounter big training datasets and complex loss functions, which makes the optimization process quite expensive with the computational complexity $\geq O(N)$. Since the gradients are an expectation, we can approximate by using a part of the dataset which are independent and uniformly randomly selected.

Therefore, instead of passing the entire training data, a mini-batch $\{x_1, x_2, x_3, \ldots, x_m\}$ is selected at every iteration to compute the gradients of the cost function. Here, $m < N$ can be any small number ranging from 1 to a few hundred (Goodfellow *et al.,* 2016).

By applying the mini-batch examples, the Monte Carlo estimate of the gradient is given as $g = \frac{1}{m} \nabla_{\theta \in \Theta} \sum_{i=1}^{m} \left[ L(M_\theta(x^i, y^i)) \right]$. When the mini-batch size is one, the procedure is called the stochastic gradient descent (SGD), and the computational complexity reduces to $O(m = 1)$. With the recent technological advancement, multiple machines can be used in parallel for more efficient training (Erizmann, 2021).

In practice, the SGD and its variants are the most used algorithms in finding the optimal solution to the neural network. In addition, if the training data is small, it is beneficial for the network to attend to it several times by setting the minimum number of epochs for better results. The number of epochs can be tuned along with other hyperparameters in the network. For more factors associated with the mini-batch size and training, see Chapter 8 of Goodfellow *et al.*, (2016).

To obtain the derivative of the loss function with respect to the model's parameters, i.e., $\nabla_{\theta \in \Theta} L(M_\theta, (x, y))$, we employ the backpropagation (BPP) algorithm. The BPP algorithm is the fastest method that employs the chain rule of differentiation to express how quickly the loss function changes as the network parameters (weight and bias) change, revealing the model's overall behavior (Nielsen, 2015).

The goal of the BPP algorithm is to compute the derivative of the cost function with respect to the weights and biases of the neural network model, i.e., $\nabla_{\theta \in \Theta} L(M_\theta, (x, y))$. In order to conduct this operation; we assume that the cost function $L(M_\theta, (x, y))$ can be expressed as the expectation over the cost of an individual example, i.e., $L = E_{(\theta_i \in \Theta_i)}[L_i]$ where $i$ is an example since BPP computes

the partial derivative per individual example and take the average overall training samples for the final result.

Also, the cost function is assumed to be represented as a function of the outputs of the neural network i.e., $L = L(M_\theta(x), y)$.

Nielsen (2015) summarizes the BPP algorithm into the following steps:

- Supply the input $X$ to the neural network $M_{\theta\in\Theta}$ and set its corresponding activation $a^1$,
- For each layer $1,2,3,\ldots,L$ of the neural network perform forward propagation by computing the quantities $Z^l = W^l a^{l-1} + b^l$ and $a^l = \sigma(Z^l)$ where $W, b, \sigma$ are the weights, biases, and activation functions at the layer $l$,
- Compute the output error $\delta^L = {}_a L \odot \sigma'(Z^L)$ where $\odot$ is the Hadamard (point-wise) multiplication of vectors,
- Back-propagate the error by computing $\left((W^{(l+1)^T}\delta^{(l+1)}\right) \odot \sigma'(Z^l)$ ,
- Obtain the gradients for the weights and biases as $\frac{\partial L}{\partial W^l_{j,k}} = a_k^{(l-1)}\delta_j^l$ and $\frac{\partial L}{\partial b_j^l} = \delta_j^l$ .

Training the recurrent neural network (RNN) requires unfolding the network through time. Although the RNN shares weights across time stamps, the network can quickly explode once unfolded since every time stamp constitutes a unique layer.

We apply the standard BPP algorithm to the unfolded network, and we call it Backpropagation Through Time (BPPT) which is quite expensive to compute since all intermediate states need to be known in advance (Erizmann, 2021). For more details and proof of the BPP algorithm, see Chapter 2 of Nielsen (2015).

We can apply various methods to accelerate the training of the RNN models, including teacher-forcing and curriculum learning. These techniques are specifically used for the RNN with the encoder-decoder structure whereby, during the decoding process, the network is supplied with the ground truth target instead of previous predictions (teacher-forcing) or randomly supplied with the ground truth predictions over a particular probability value (curriculum learning) to assist the next step of prediction (Goodfellow *et al.*, 2016).

At any time-step $t$, the hidden state of the RNN model is given as $h_t = \Gamma[W_x^T X_t + W_h^T h_{t-1} + b_t]$, which gives the partial derivative of the state vector at the time $t$ with respect to the previous state as $\frac{\partial h_t}{\partial h_{t-1}} = \Gamma'[W_x^T X_t + W_h^T h_{t-1} + b_t]W_h$. Since the derivatives used for BPPT depend on the product of the activation function ($\Gamma[.]$) and the network parameters at later layers, the RNN suffers from the problem of gradient decay or an explosion depending on the nature of the non-linearity function.

In addition, multiplication with the derivatives of identical weights may accelerate the problem due to parameter sharing, i.e., when the weights are less than one (gradient decay) or greater than one (gradient explosion). Several measures to address this issue are proposed in the literature, including truncated backpropagation through time, gradient clipping, the better choice of activation function, and the use of gated recurrent units such as LSTM and GRU (Erizmann, 2021).

# 8. THE BENCHMARK MODELS

For comparison purposes, we consider both traditional and deep learning models. We compare our approach with the Fine and Gray model (1999), which we extend to account for the case of a recurrent event with competing risks, the random survival forest (SRF) for competing risks (Ishwaran *et al.,* 2014), the DeepHit (Lee *et al.*, 2018), the DeepSurv (Katzman *et al.,* 2018), and CRESA (Gupta *et al.*, 2019). Below we give a brief description of each model.

## 8.1. A Proportional Hazards Model for the Subdistribution of a Competing Risk

The famous traditional approach to dealing with multiple events with competing risks in the right way is to apply the concept of the incidence function. To mitigate the strong assumptions and interpretability issues in modeling competing risks, Fine and Gray (1999) developed the semi-parametric proportional hazard model, which relies on the cumulative incidence function (CIF). Consider $K$ competing events labeled $(1,2,3,\dots,K)$ and a vector of covariates $X$. For a particular event, $K = k^*$, We obtain the sub-distribution hazard function at the time $T = t^*$ as

$$h_{k^*}(t) = \lim_{\delta_t^* \to 0} \frac{P(t^* \leq T \leq t^* + \delta_t^*, K = k^* | (T \geq t^*) \cup (T \leq t^* \cap K \neq k^*)}{\delta_t^*} \tag{8.1}$$

where the risk set $(T \geq t^*) \cup (T \leq t^* \cap K \neq k^*)$ includes all survivors from the risk of interest and other individuals who encountered other events different from $k^*$. For the sub-distribution baseline hazard $h_{0,k^*}(t)$, we express the risk function as $h_{k^*}(t^*) = h_{0,k^*}(t^*)e^{X^T\beta}$ where $\beta$ is the parameters vector and $X$ is the vector of covariates. The (CIF) for the specified risk $k^*$ at the time $t^*$ is given as $F_{k^*}(t^*|X) = P(T \leq t^*, K = k^*|X, (T \geq t^*) \cup (T \leq t^* \cap K \neq k^*)$.

We obtain the generalization for the case of recurrent events with competing risks as follows:

Let us assume $K$ events occur multiple times represented by the time steps $(1,2,3,\dots,R)$. We compute the CIF for an individual at a particular time step, say $t_r^*$, by conditioning on the previous time steps.

We compute the sub-distribution hazard function as

$$h_{k^*}(t_r^*) = \lim_{\delta_{t_r} \to 0} \frac{P\left(t_r^* \leq T_r \leq t_r^* + \delta_{t_r^*}, K = k^* | (T_r \geq t_r^*) \cup (T_r \leq t_r^* \cap K \neq k^*\right)}{\delta_{t_r}},$$

which simplifies to

$$h_{k^*}(t_r^*) = h_{k^*}(t_r^* | X_r) = h_{0,k^*}(t_r^*) e^{X_r^T \beta} \tag{8.2}$$

where $(r \in R)$.

We can now obtain the Fine-Gray CIF for the case of recurrent events with competing risks as

$$F_{k^*}(t_r^* | X_r) = P(T_r \leq t_r^*, K = k^* | X_r, (T_r \geq t_r^*) \cup (T_r \leq t_r^* \cap K \neq k^*) \tag{8.3}$$

Although uniformly consistent estimators of the predicted CIF for an individual with specified covariates exist, the solid proportional hazard assumptions and the inability to handle non-linear effects limit the performance of this model.

### 8.2. Random Survival Forest (RSF) for Competing Risks

Random survival forest (RSF) is the collection of decision trees specifically for right-censored survival data. In their first work, the authors proposed a tree ensemble algorithm that treats the survival problem as a classification problem, but instead of predicting a particular class or value at the terminal node, they predicted the quantity called ensemble mortality. From this setting, we obtain an extension to include the competing risk scenarios (Ishwaran et al., 2014).

The RSF differs from Breiman's random forest (RF) (Breiman, 2001) concerning the splitting rules. In RSF, the splitting rule incorporates survival time and the status of an event. The RSF algorithm consists of the following steps:

1. We select the ‘b’ bootstrap samples which include 63% of the data for training and the remaining 37% referred to as out-of-bag (OOB) samples for validation.
2. We develop the survival tree using each bootstrap sample ‘b’ by selecting some covariates at random to obtain a decision variable, and the splitting rule that maximizes the survival difference between the daughter nodes is selected.
3. The survival tree is grown until we reach a particular criterion, i.e., until the number of notable deaths is not less than a pre-specified positive value.
4. We compute the cumulative hazard function (CHF) at the terminal node for each tree and obtain the average across all trees (ensemble prediction).
5. The out-of-bag samples (OOB) are used for evaluation.

On the contrary, different splitting criteria to grow a competing risk survival tree is proposed. As a modification to address a survival problem with competing risks, the CIF of Fine and Gray (1999) is estimated at the terminal node.

The generalized log-rank test, which relies on the weighted difference of the cause-specific risks of the Nelson-Aalen estimates, is used as a splitting criterion to detect the variables related to the cause-specific hazards. To obtain the optimal CIF estimates, Gray's test (Gray, 1988) compares the cumulative incidence functions of the daughter's nodes to ensure the selection of variables directly affecting the CIF is used as a splitting rule.

Further, the composite rule combining the cause-specific splitting criteria across the event's type is used to obtain the CIF of all causes simultaneously and identify variables influencing the cause-specific events.

The outline of the RSF for the competing risk algorithm includes the following steps.

1. Obtain the ‘b’ bootstrap samples from the training data
2. Select a random sample of covariates from the bootstrap, and use the variable that maximizes the competing risks splitting rule as a decision variable to split the node into daughters nodes

3. Grow the tree to full size until the termination criteria are reached (until the terminal nodes have a minimum pre-specified number of unique candidates.
4. Risk-specific quantities of interest, such as CIF and cumulative hazards functions, are estimated at the end of each competing risk decision tree, and the average across all trees is obtained as ensemble estimates.

As our benchmark, we train the RSF for competing risks at every time step.

### 8.3. A Deep Learning Approach to Survival Analysis with Competing Risks (DeepHit)

DeepHit is an alternative to the traditional approach for survival prediction in case of multiple events with competing risks, leveraging the power of deep neural networks (Lee et al., 2018). DeepHit has a multitask learning architecture which allows the network to learn the joint distribution of survival times and risks directly.

The DeepHit model assumes a class of mutually exclusive competing events $K \geq 1$ with right-censorship denoted as $K = \emptyset$ discrete survival time $(t_1 = 0, t_2, t_3, \ldots, T_{max} = 100)$, which leads to the survival data expression as $D = \left\{(X^i, s^i, k^i)\right\}_{i=1}^{N}$ where $s = min(T, C)$ is the time at which an event or censorship is recorded for a subject.

The target is to predict the probability for a particular event $K = k^*$ that occurs at the time $T = s^*$ for the given subject with covariates $X$, which is given as $P(T = s^*, K = k^* | X, k^* \neq \emptyset)$. Figure 8.1 shows the DeepHit architecture.

The architecture consists of fully connected layers for both shared and risk-specific sub-networks. The shared sub-network maps the input vector $X$ to the hidden representation $f(X)$, which is then combined with the raw inputs using the residual connection to give $Z = \{f(X), X\}$, where $Z$ is the input to the risk-specific sub-networks which uses the shared softmax layer to learn the probability distribution of survival times for every risk.

The network utilizes the Fine and Gray (1999) CIF to assess the discrimination ability across competing events. For a particular risk say $K = k^*$ the CIF for an individual with covariate $X$ at the time $T = t^*$ is given as

$$F_{k^*}(t^*|X) = P(T \leq t^*, K = k^*|X)$$

which simplifies to $\sum_{t=0}^{t^*} P(T = t, K = k^*|X)$. This quantity is unknown and is estimated from the network using the softmax probabilities $y_{(k^*,t^*)}$ as $\hat{F}_{k^*}(t^*|X) = \sum_{t=0}^{t^*}\left(y_{k^*,t}^*\right)$.

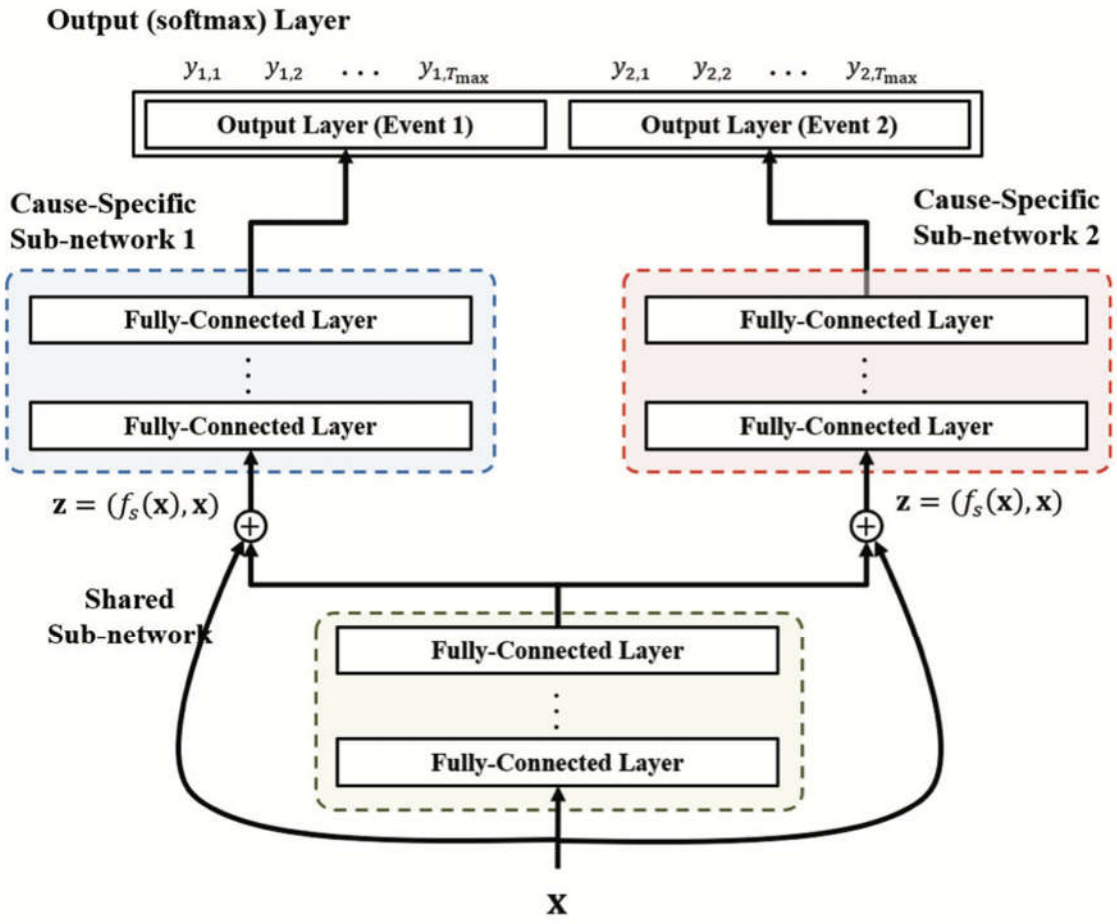



Figure 8.1. The DeepHit architecture

For training, the DeepHit utilizes the loss function which consists of the partial log-likelihood function for survival data with competing risks and the ranking loss as a miss-classification penalty. The total loss is given as

$$L = L_1 + L_2 = -\sum_{i=1}^{N}\left(1(k^i \neq \emptyset)log(y_{k^i,t^i}) + 1(k^i = \emptyset)log(\sum_{i=1}^{N}(1 - \hat{F}_k((t^i)|X^i)))\right) + \sum_{k=1}^{K}\alpha_k\sum_{i\neq j}A_{k,i,j}\eta(\hat{F}_k(t^i)|X^i), \hat{F}_k(t^i)|X^j)) \quad (8.4)$$

where $1(.)$ is an indicator function, the second term in the log-likelihood loss ($L_1$) covers survival information for the censored individuals, $A_{k,i,j}$ is an indicator function to capture the alignment of two individuals $(i, j)$ according to the specified risk ($K = k$), $\alpha_k$ is a trade-off parameter, and $\eta(.)$ is a convex kernel. As the measure of performance, the time-dependent concordance index is given as

$$C^t = P\left[\hat{F}_k(t^i|X^i) \geq \hat{F}_k(t^i|X^j)|t^i \leq t^j\right] = \frac{\sum A_{k,i,j} 1\left(\hat{F}_k(t^i|X^i) \geq \hat{F}_k(t^i|X^j)\right)}{\sum_{i \neq j} \quad A_{k,i,j}}. \tag{8.5}$$

We will train this model at multiple time points as the baseline for the case of multiple recurring events with competing risks.

### 8.4. Personalized Treatment Recommender System Using A Cox Proportional Hazards Deep Neural Network (Deepsurv)

In order to address the limitation of the CPH model, the DeepSurv is specifically designed to model the high-level treatment-covariate interactions to achieve personalized treatment recommendations (Katzman *et al.,* 2018).

In this study, the authors address the linearity assumption on the log risk function of the CPH model by using neural networks. The model is structured as follows: Consider the CPH model given as $f(t|X) = h_0(t)e^{\tau_x}$, where $\tau_x$ is the risk score to be estimated, and $h_0(t)$ is the baseline common to all subjects.

The Cox proportional hazard assumption estimates the risk score $\tau_x$ as a log-linear function of the covariates given as $\tau_x = e^{X^T\beta}$, where $\beta$ are the model's coefficients, and $X$ is the vector of covariates. Since, in practice, we encounter complex interactions between patient covariates and treatment concerning the outcome risks, the linearity assumption on the risk score function is not adequate.

We can replace the risk score function with the neural network's output, which is the modification of Faragi and Simon's (1995) method (Katzman *et al.*, 2018). The network architecture consists of

MLPs with a scaled exponential linear unit (SELU) activation. For regularization, they employed the dropout technique and $l_2$ norm, which is added as a component of the loss function.

The risk score function $\tau_x$ is now approximated by the output of the neural network $\hat{h}_{\theta\in\Theta}(X)$, which is parameterized by the new weights ($\theta \in \Theta$). We train this model by minimizing the loss function, which consists of the partial log-likelihood function with $l_2$ a penalty on the estimated parameters given as

$$l_\theta = -\frac{1}{N_{E=1}}\sum_{i:E_i=1}\left(\hat{h}_\theta(X_i) - log(\sum_{j\in R_i} e^{\hat{h}_\theta(X_j)})\right) + \delta\left|\left|\theta\right|\right|_2^2 \tag{8.6}$$

Notice that this loss explains the event cases only and the censorship information is ignored since the treatment recommender system applies to the available patients.

As the benchmark, we can extend this model to account for the recurrent event with competing risks. Considering a set of $K$ events that repeats R times for a given subject, we can express the hazard function for a specified event $k^*$ at a time $T_r$ as $h_{k^*r}(t_r|X_r, K_r = k^*) = h_{0k^*,r}e^{M_{\theta\in\Theta}(X_r,k^*)}$, where $M_{\theta\in\Theta}(.)$ is the output from the neural network at the time $r$ of risk $k^*$.

We can then express the likelihood function as

$$L_\theta = \prod_{i:E_{ri}=1}\left(\prod_{k\in K} \quad \frac{e^{M_{\theta\in\Theta}(X_{r_i},k)}}{\sum_{j\in R_i} e^{M_{\theta\in\Theta}(X_{r_j},k)}}\right) \tag{8.7}$$

where $M_{\theta\in\Theta}(.)$ is an estimated risk score function for risk $k$ at the time $r$.

As our benchmark, we train DeepSurv by minimizing the loss function of the form

$$L_\theta = -\frac{1}{N_{E=1}}\sum_{r\in R}\sum_{i:E_{ri}=1}\left[M_{\theta\in\Theta}(X_{r_i},k) - log(\sum_{j\in R_i} e^{M_{\theta\in\Theta}(X_{r_j},k)})\right] + \delta\left|\left|\theta\right|\right|_2^2 \tag{8.8}$$

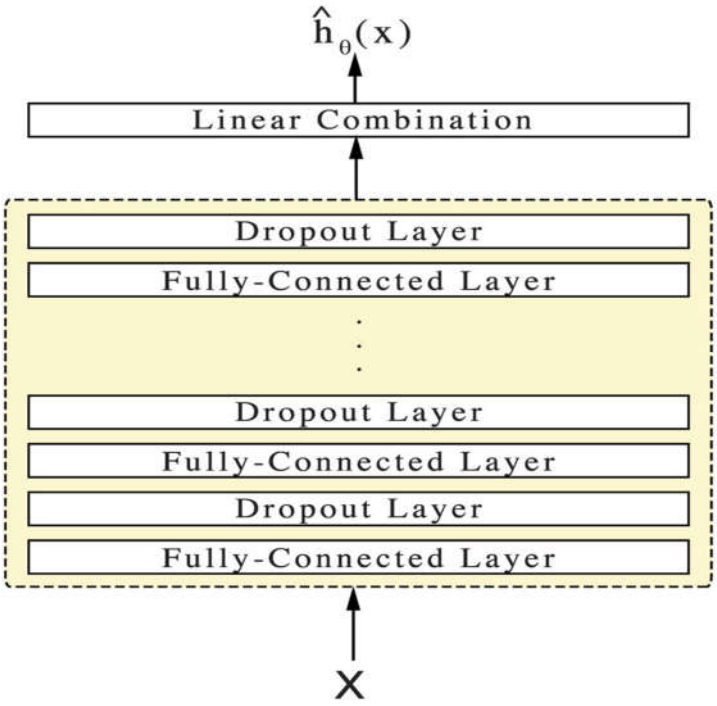



Figure 8.2. The DeepSurv architecture

### 8.5. A Deep Learning Approach to Competing Risks Recurrent Events Survival Analysis (CRESA)

CRESA is the deep learning algorithm for predicting survival time distribution and risk-specific outcomes for recurrent events with competing risk survival problems (Gupta et al.,2019). The network is designed to mitigate the strong assumptions of traditional approaches, such as constant hazards function and the stochastic nature of the data generation process. The authors utilized the CIF concept from Fine and Gray (1999) to predict the risk-specific outcomes in case of recurrent events with competing risks. For the baseline, both traditional and deep learning techniques are applied. The assessment is achieved based on simulation and real datasets. Figure 8.3 shows the architecture of the CRESA network.

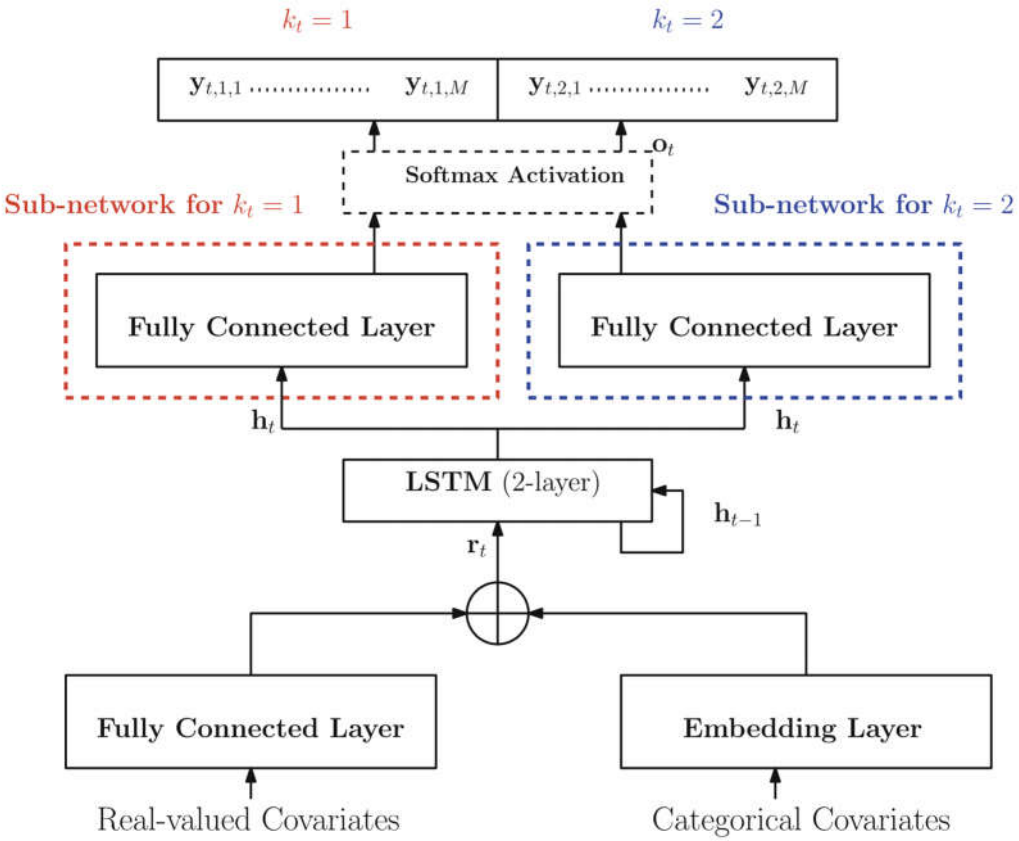



Figure 8.3. The CRESA network architecture

This architecture consists of two stacked LSTM layers, which receive the preprocessed survival data. The hidden states of the LSTM become the input to the risk-specific sub-networks, the MLPs. To obtain the risk-specific survival distribution, a shared softmax layer is employed. In case of a single risk, this network reduces to the simple LSTM.

The loss function consists of ranking loss and partial log-likelihood, specifically for recurrent events with competing risks. For a specified risk $K = k^*$ and covariate vector $X_t^*$, the cause-specific cumulative incidence function at or before time $s_{t^*}$ is given as

$$F_k^*(s_t^*|x_t^*) = P(S_t \leq s_t^*, K = k^*|X_{t^*}, (S_t \geq s_t^*) \cup (S_t \leq s_t^* \cap K \neq k^*)) \tag{8.9}$$

where $(S_t \geq s_t^*) \cup (S_t \leq s_t^* \cap K \neq k^*)$ is the risk set which consists of all individuals free of risk $k^*$ and those who developed any other competing events prior to C. The partial log-likelihood is given as

$$L_1 = \sum_{t=1}^{T} \sum_{i=1}^{N} \left[ (\delta_{t^i} = 1) log\left( y_{t^i}^i, s_{t^i}, k_t^i \right) + (\delta_{t^i} = 0) log\left( 1 - \sum_{k=1}^{K} \hat{F}_k(s_t^i | X_t^i) \right) \right] \quad (8.10)$$

where $\delta_i$ is an indicator function for censoring or event, and $\hat{F}_k(s_t^i | X_t^i)$ is predictions from the softmax layer of the network. The ranking loss is given as

$$L_2 = \sum_{t=1}^{T} \alpha_{k,t} \sum_{i \neq j} A_t^{k,i,j} \tau\left( \hat{F}_k(s_t^i | x_t^i), \hat{F}_k(s_t^i | x_t^j) \right) \quad (8.11)$$

where $\alpha_k$ is a trade-off parameter, $A_t^{k,i,j}$ is an indicator function given as

$$A_t^{k,i,j} = P\left( \hat{F}_k(s_t^i | x_t^i) \leq \hat{F}_k(s_t^i | x_t^j) | s_t^i \geq s_t^j \right),$$

and $\tau(.)$ is a convex function.

The ranking loss is a penalty factor if an individual is wrongly classified as possessing a particular risk. As the benchmark, this model can be obtained as a particular case of our network when we exclude the residual connections, an external auto-encoder, and the attention mechanism.

# 9. EXPERIMENTATION

In this study, we utilize both synthetic and real-life datasets to train and evaluate our model. We consider the case of two events for demonstration purposes, and generalization can be achieved analogously. We describe the data and problem formulation in the following section.

## 9.1 Simulation of Recurrent Events with Competing Risks

Generation of recurrent events with competing risks survival data involves a dynamic process that considers all possible forms of complexity, such as events dependency and heterogeneity among the subjects. In medical research, it is necessary to determine the correct time scale to reflect the problem under study. We employ the calendar time scale when simulating events that evolve in the long run, while the gap time scale, which depends on the inter-arrival distribution, is selected for events that evolve within a limited time (Penichoux et al.,2015).

Since it is unrealistic to assume a homogeneous stochastic process for the complex survival time, the Gompertz distribution can be applied to simulate data that reflects reality (Bender et al.,2005). To address the variation among the subjects in a dynamic survival setting, adding a random effect term specific to each individual is necessary. In practice, gamma frailty under the Gompertz distribution baseline hazard has been proven to effectively models a heterogeneous population (Tutkun & Marthin, 2021). In this study, we made the following assumptions for data generation.

Following the simulation done by (Lee et al., 2018) and (Gupta et al., 2019) on complex survival data, we also consider the set of multivariate distributed normal covariates

$$\{(x_1, x_2, x_3)\} \sim N(0, I_4).$$

For simplicity, we assume the case of two competing risks ($K = 2$). To reflect the complexity of the recurrent events with competing risks, we employ Gompertz distribution for the survival and the gap times, i.e., $T_{k,t} \sim Gompertz(\lambda_k, \alpha_k)$ and $w_k \sim Gompertz(\lambda_k, \alpha_k)$. To account for the events dependency, we express the parameters of the time generator distribution in terms of the

covariates, i.e., $[\lambda_1 = \lambda_2 = 1]$, $\left[\alpha_1 = \omega_1^T x_{1,t} + (\omega_3^T x_{3,t})^2, \alpha_2 = \omega_2^T x_{2,t} + (\omega_3^T x_{3,t})^2\right]$, where $\omega_1 = \omega_2 = \omega_3 = 10$.

We also include the gamma-distributed frailty term ($\delta \sim \Gamma(a, b)$) as a random effect to address the heterogeneity among the subjects. Therefore, for an individual subject, the survival times are distributed as

$$T_{1,t}^i \sim Gompertz(\lambda_1, \alpha_1)\ T_{2,t}^i \sim Gompertz(\lambda_2, \alpha_2)\ ,$$

where $\lambda_1, \lambda_2, \alpha_1, \alpha_2$ are defined above. A random variable $x \sim Gompertz(\alpha, \beta)$ if the probability density function *(pdf)* is given as

$$f(x) = \beta e^{\alpha x\left[e^{\frac{\beta}{\alpha}[1-e^{\alpha x}]}\right]},$$

where $\alpha$ is the shape parameter and $\beta$ is the scale parameter.

Following the work of Penichoux et al., (2015), the multiplicative proportional intensity using the gap time is given as

$$h_k(t|X_t) = h_{0,k}(t - T_{N(t-)})e^{X_t^T \beta_k},$$

where $(t - T_{N(t-)})$ is the gap time which represents the inter-arrival times between the recurrent events for risk $k$.

To account for the heterogeneity, we simulate the identical independently distributed $(iid)$ samples $(\delta_k)$ from the gamma distribution. We assume the gap time quantity follows the same distribution as the survival times, i.e., $Gompertz(a, b)$. Let the gap-time variable for the $k^{th}$ event

$$(w_k = (t_k - T_{N(t_k-)})) \sim Gompertz(a, b),$$

the cumulative distribution function for the $j^{th}$ step $(F_{k_j})$ is given as

$$F_{k_j} = 1 - e^{H_{0,k}(w_k)e^{\beta_k^T x_j}}$$

where $H_{0,k}$ is the CDF of the Gompertz model. Further, the random variable $y \sim \Gamma(a, b)$, if the probability density function is given as

$$f(y) = \frac{a^b y^{b-1} e^{-ay}}{\Gamma(b)}, (a, y) > 0.$$

Including this frailty term, we can write the proportional intensity function as

$$h_k(t|X_t) = h_{0,k}(t - T_{N(t-)})\delta_k e^{X_t^T \beta_k}$$

where $\delta_1 \sim \Gamma(a = 2, b = 3)$ and $\delta_2 \sim \Gamma(a = 0.5, b = 3)$. Since a random variable's cumulative distribution function (CDF) is a random variable with a uniform distribution, we have

$\left[F_{k_j} = 1 - e^{H_{0,k}(w_k)\delta_k e^{\beta^T X}}\right] \sim U[0,1]$, which implies

$$\left[1 - F_{k_j} = e^{H_{0,k}(w_k)\delta_k e^{\beta^T X}}\right] \sim U[0,1].$$

Therefore, we generate the survival process as

$$\{T_{k,t}\} = \frac{\delta_{k,t}}{\lambda_{k,t}}\left\{log\left[1 - \frac{\alpha_{k,t}}{\lambda_{k,t}}\frac{log(U)}{exp(\beta^T X}\right]\right\},$$

where $\beta_1 = \beta_2 = \beta_3 = \beta_4 = 1, U \sim U[0,1]$ and $\lambda_{k,t}, \alpha_{k,t}, \delta_{k,t}$ are defined above.

We then select the survival time $T_{k,t}^i = min(T_{1,t}^i, T_{2,t}^i)$ if $K_t^i \neq \emptyset$ and $T_{k,t}^i = C_{k,t}^i$ if $K_t^i = \emptyset$ where $C_{k,t}^i$ is the uniformly distributed censorship accounts the 45% of the data, i.e.,

$C_{k,t}^{i} = U[0, min(T_{1,t}^{i}, T_{2,t}^{i})]$. We employed Numpy and Pandas libraries in Python to simulate two processes of length five, each with a maximum time horizon of 30 and 30,000 subjects. We outline our data generation algorithm in Figure 9.1. Figure 9.2 gives the overall event's status distribution across the time steps and the survival time distribution with embedded covariates. We observe the uniform occurrence of the events for all time steps since we assumed common censorship with a rate of 45% to demonstrate the performance of our model for the data with steady dynamics.

To evaluate the performance of the ExternalAE, we introduced a total of 49 binary covariates to account for noise and redundancy information. This is accomplished following the work of Lee & Carlin (2012) and Rietschel *et al*.(2018), where, for each subject, we simulate the process $\{Z_t^i; t = 1,2,3,,4,5\} \in R^{49\times5}$, where $Z_t^i \sim Bernoulli(p_t)$ and $p_t \sim U[0,1]$.

**Algorithm 1:** Complex Survival Data Generator

N = 30000, T = 5, $\beta_1 = \beta_2 = \beta_3 = \beta_4 = 1$
$\lambda_{1,t} = \lambda_{2,t} = 1, \omega_1 = \omega_2 = \omega_3 = 10,$
$T_{c_1} = [\,], T_{c_2} = [\,], Data = [\,], Label = [\,]$
$\alpha_1 = \omega_1^T \mathbf{x}_{1,t} + (\omega_3^T \mathbf{x}_{3,t})^2, \alpha_2 = \omega_2^T \mathbf{x}_{2,t} + (\omega_3^T \mathbf{x}_{3,t})^2$

**for** *i in range N* **do**
  **for** *t in range T* **do**
    $X_{1,t} \sim N(0, I_4)$
    $X_{2,t} \sim N(0, I_4)$
    $\mathrm{X}_{3,t} \sim N(0, I_4)$
    **if** *k = 1* **then**
      // Simulating the first event process
      a = 2, b = 3
      $\delta_k \sim \Gamma(a, b)$
      $W_k \sim Gompertz(\lambda_{1,t}, \alpha_{1,t})$
      $U_1 \sim U[0,\ 1]$
      $\{T_{1,t}\} = \delta_k \left[\frac{1}{\lambda_{k,t}} \left\{log\left[1 - \frac{\alpha_{k,t}}{\lambda_{k,t}} \frac{log(U_1)}{exp(\beta^T X}\right]\right\}\right]$
    **else**
      // Simulating the second event process
      a = 0.5, b = 3
      $\delta_k \sim \Gamma(a, b)$
      $U_2 \sim U[0,\ 1]$
      $W_k \sim Gompertz(\lambda_{2,t}, \alpha_{2,t})$
      $\{T_{2,t}\} = \delta_k \left[\frac{1}{\lambda_{k,t}} \left\{log\left[1 - \frac{\alpha_{k,t}}{\lambda_{k,t}} \frac{log(U_2)}{exp(\beta^T X}\right]\right\}\right]$
    **if** $(T_{1,t}$ *or* $T_{2,t}) \leq 30$ **then**
      $\mathrm{T}_{c_1}.append(T_{1,t}),\ T_{c_2}.append(T_{2,t})$
      // Simulating censored process
    C = 0.45 * N,
    $C_t \sim U(0, min(T_{1,t}, T_{2,t}), size = C)$
    $[[C_t \oplus (ZeroVector) \in \mathcal{R}^{N-C}].Randomize] \in \mathcal{R}^N$
    // Creating the label variable
    **for** *c in range C* **do**
      **if** $(T_{1,t} < T_{2,t})$ *and* $C_t \neq 0$ **then**
        $Label.append(1)$
      *ElseIf*
      $(T_{1,t} > T_{2,t})$ *and* $C_t \neq 0$ $Label.append(2)$
      **else**
        $Label.append(0)$
      // Obtaining the survival time
      $S_t = Min[T_{1,t},\ T_{2,t}]$ *if* $C_t \neq 0$ *and* $S_t = C_t$ *otherwise*
      // Packing the data
    $Data.append\,[S_t,\ Label,\ X_{1,t},\ X_{2,t},\ X_{3,t}]$

Figure 9.1. Recurrent event with competing risks survival data generator algorithm

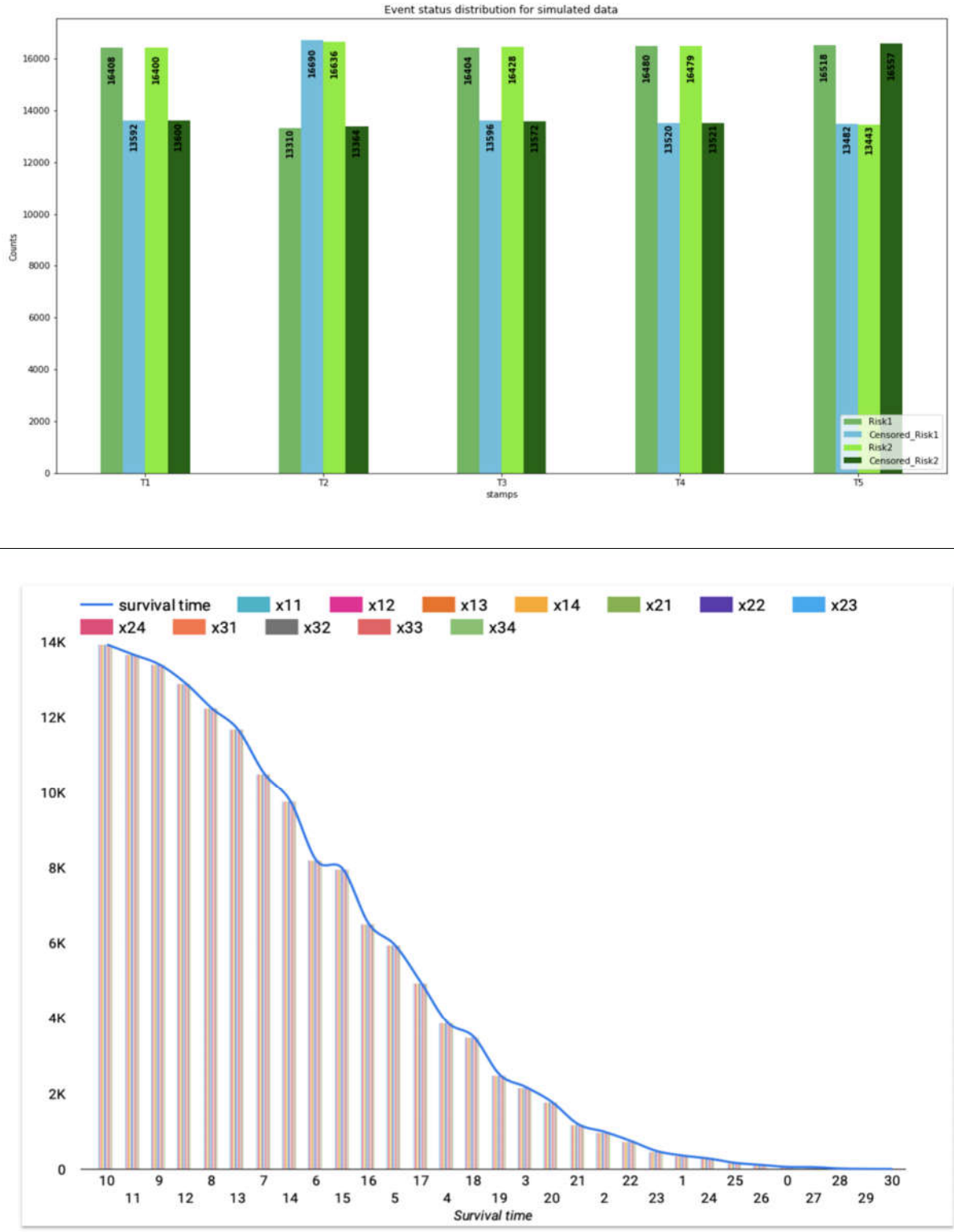



Figure 9.2. Survival and event status distribution across the time stamps

### 9.2. UNOS-OPTN (KIDPAN) Dataset

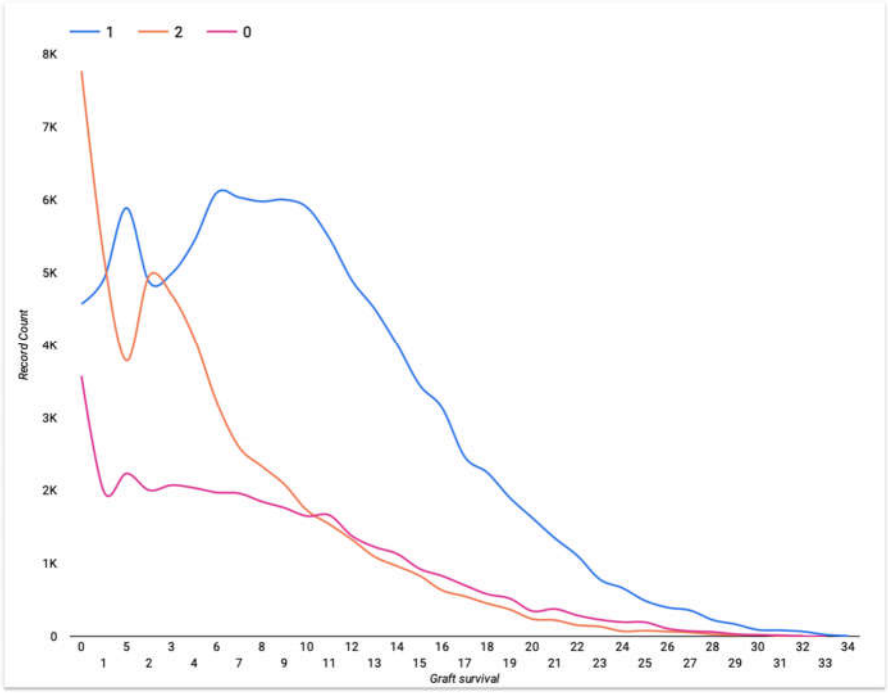
UNOS-OPTN is the Organ Procurement and Transplantation Network (OPTN), which is a national transplant system that is operated by the United Network for Organ Sharing (UNOS). The OPTN is responsible for the allocation of organs for transplantation in the United States and for maintaining a national database of transplant data. The UNOS-OPTN database contains information on organ donors, transplant recipients, and organs that have been transplanted in the United States. The data includes information on the characteristics of the donors and recipients, the organs that were transplanted, and the outcomes of the transplants (UNOS-KIDPAN, 2022).

The UNOS-OPTN database is often used for research purposes, such as studying trends in organ transplantation and identifying factors that may affect transplant outcomes. The database is available for download to researchers who have obtained the necessary approvals. The OPTN repository contains several datasets concerning various organ transplants collected from patients and donors since 1987. The UNOS-KIDPAN is a dataset that contains data on kidney and pancreas transplant recipients in the United States.

The dataset was developed by the United Network for Organ Sharing (UNOS) and is maintained by the Scientific Registry of Transplant Recipients (SRTR). The data includes information on the characteristics of the transplant recipients, the donors, and the transplanted kidneys and pancreas, as well as outcomes such as graft survival and patient survival. The UNOS-KIDPAN dataset is often used for research purposes, such as studying trends in kidney and pancreas transplantations and identifying factors that may affect transplant outcomes (UNOS-KIDPAN, 2022).

In this study, we employ the kidney-pancreas (KIDPAN) data as of July 2022 which consists of patients hospitalized recurrently due to the failure of a kidney (KGRFT) or pancreas graft (PGRFT) after receiving an organ transplant. Among 97,050 patients who experienced graft failure at most four times, 60% are right-censored. We considered the duration (in years) until the graft failure as the survival time, and an individual is right-censored when death occurs. We selected 100 features related to demographics, clinical tests, physical measurements, medication, and mortality information for both patients and donors. Figure 9.3 gives the events' status distribution across the time stamps, and the overall event distribution based on graft survival.

Figure 9.4 displays the relationship between antigen counts for both patients and donors and the overall graft survival. We can notice that the durability of the graft is improving with low antigen counts for both patients and donors,i.e., patients with low counts for the specified antigen types, who received the organ from corresponding donors also with low counts for the specified antigen types have higher graft survival. Figure 9.5 gives the summary distribution of graft survival based on immunosuppressive disorders statuses, i.e., diabetes and HIV serostatus. For further descriptions of the data concerning some covariates and graft survival-time relationships, see the Appendix 1.



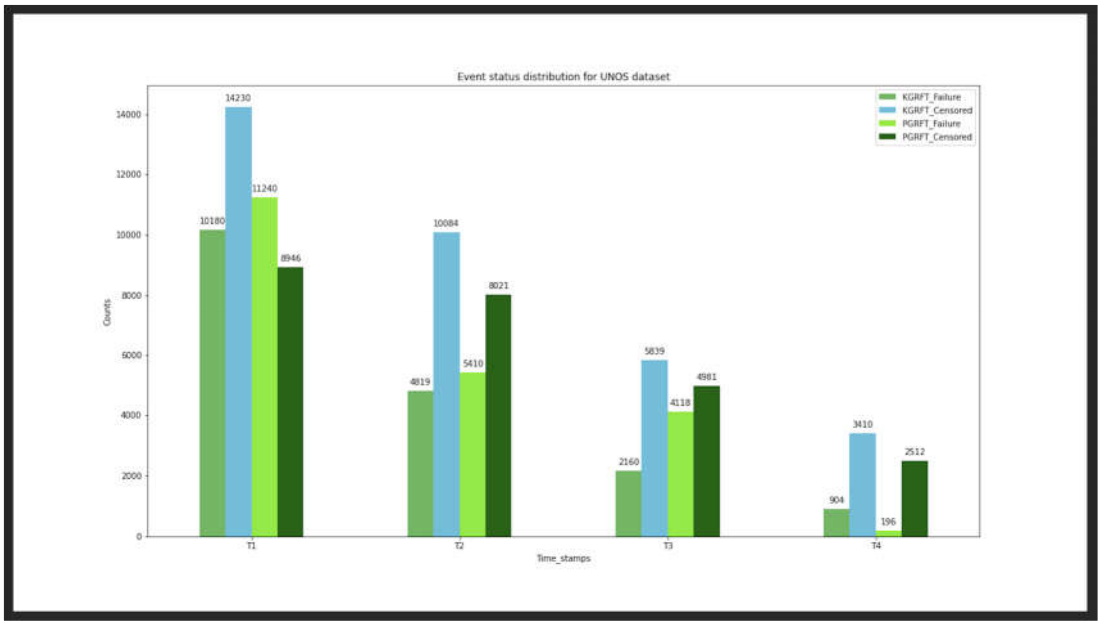




Figure 9.3. Graft survival and event status distribution across the time stamps

From Figure 9.3, we can track the overall trend of graft failure for the patients who received both kidney and pancreas transplants. The orange curve '2' represents pancreas transplant patients, the red curve '0' represents kidney transplant patients, and the blue curve '1' represents censored patients. Most patients experience failure at the early stage after the transplant. Graft failure due to a pancreas transplant is more frequent at the earlier stage compared to a kidney transplant. This is observed through the sharp slope of survival curve 2 (orange color). After year 10 most patients are readmitted due to kidney graft issues compared to pancreas graft failure. We also notice rapid fluctuation at the early stage after the pancreas transplant. There is a declining trend in the number of patients as shown in the bar plot, which makes the survival process unstable across time stamps. More patents are censored compared to events at every time step, which is the case since we have 60% of censorship.

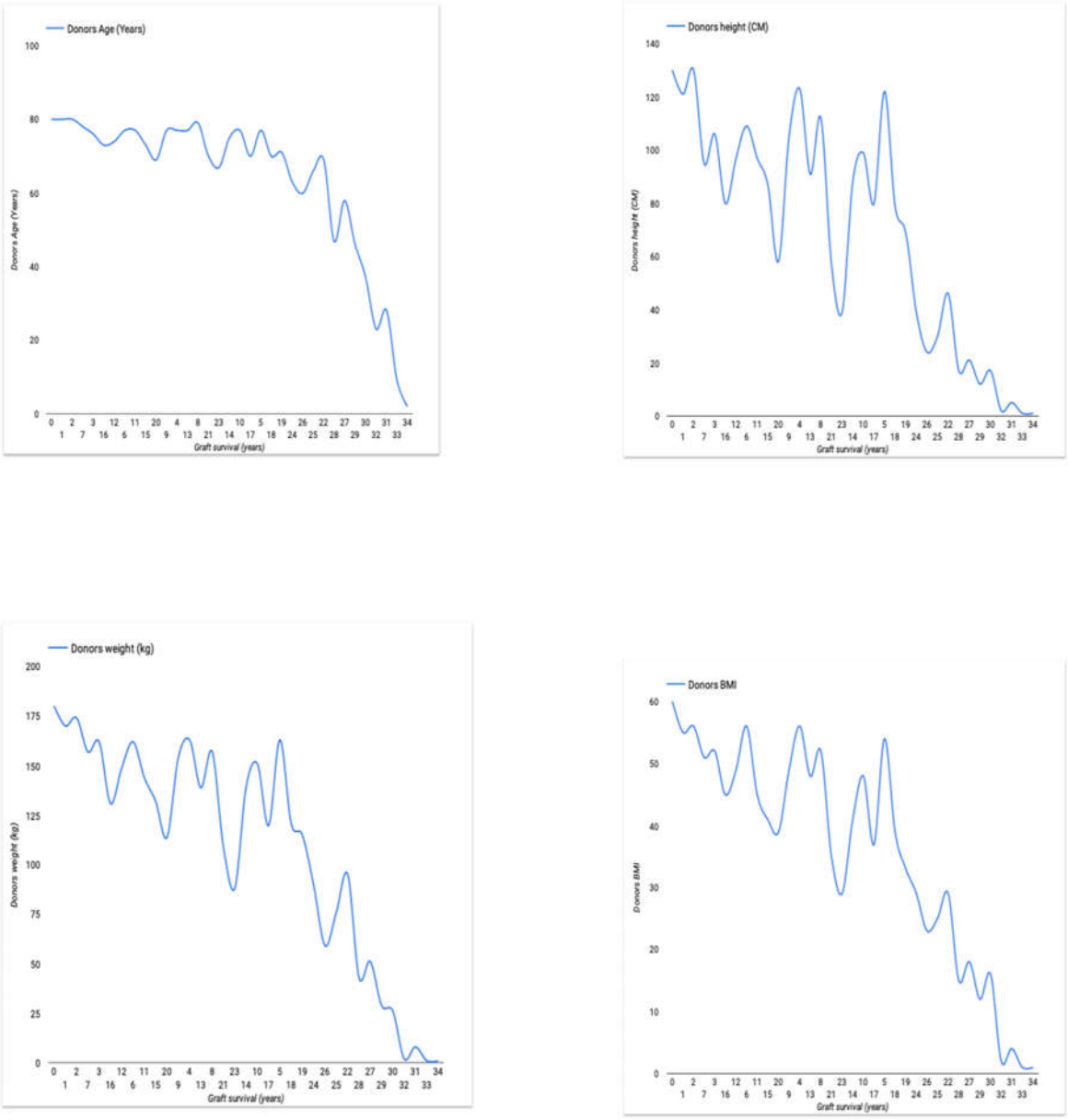
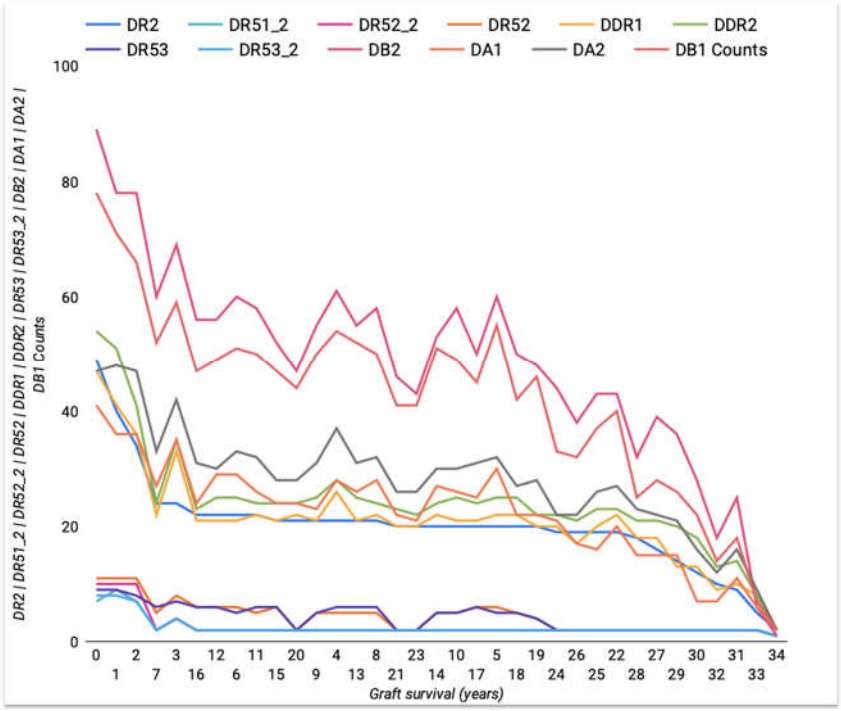



Figure 9.4. Graft survival based on donors' physical characteristics



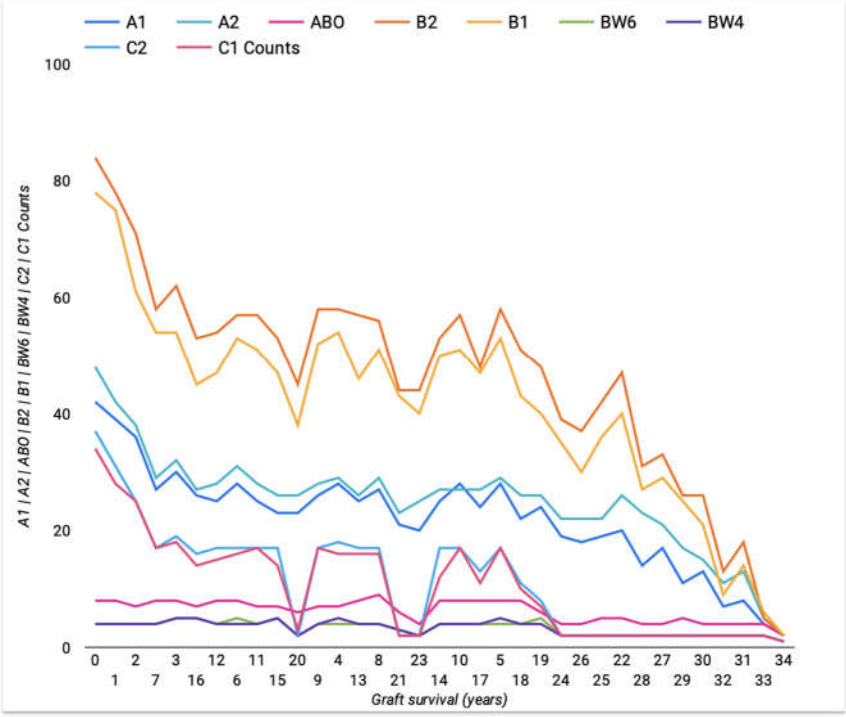
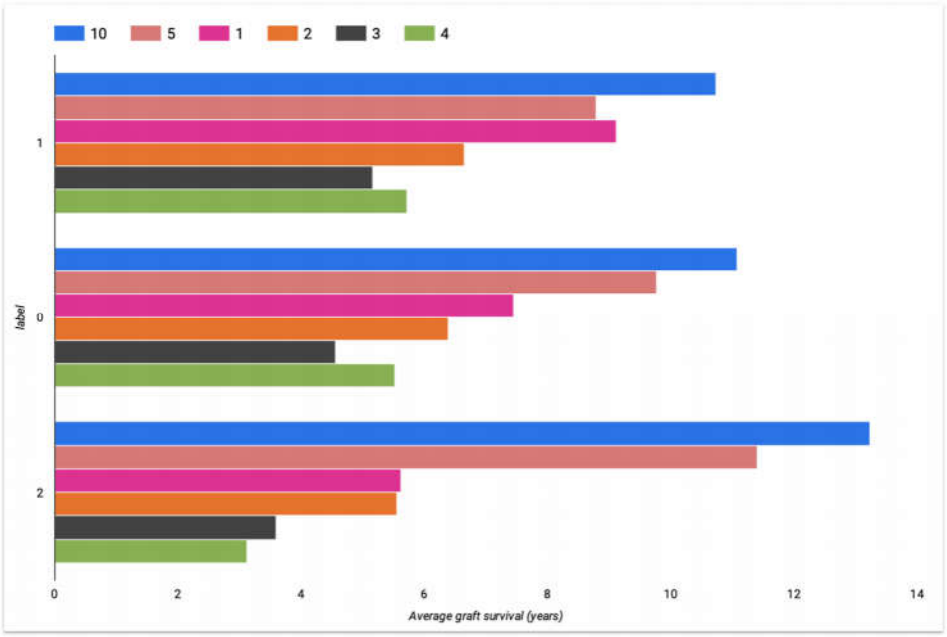




Figure 9.5. Graft survival based on patient's-donors antigen counts

Figure 9.4 gives the relationship between overall graft survival and the donor's physical characteristics. We select the donor's physical measurements such as weight in kilograms, height in centimeters, age in years, and BMI. We observe some interesting patterns concerning graft survival and the selected physical features. We can identify the inverse relationship between graft survival and the donor's physical measurements. Also, we notice the rapid fluctuation in graft survival concerning the selected physical characteristics.

We observe an interesting relationship between the selected antigen types and graft survival from Figure 9.5. We compute and plot the antigen counts for both donors and patients against graft survival. The top plot in Figure 9.5 is related to the selected antigen from donors, and the bottom plot is for patients. In both plots, we notice an improvement in graft performance with low levels of antigen counts. These features and the related ones may be good predictors for our black-box model.



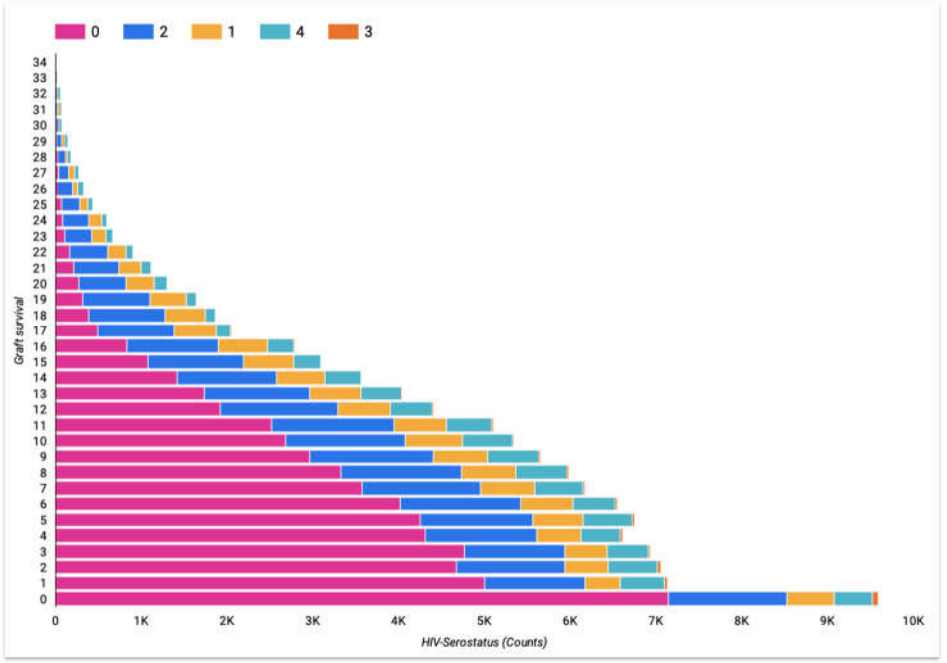




Figure 9.6. Graft survival based on patient's diabetes and HIV serostatus

The relationship between graft survival and patients' immune suppression statuses can be derived from Figure 9.6. The top diagram is the bar plot of event status against the average graft survival embedded with the patient's diabetes history. Diabetes history is interpreted as the number of years after diagnosis. There is an unusual outcome for patients with 10 years history of diabetes with graft survival. We would expect poor graft performance for this category of patients since diabetic individuals are at high risk for graft failure after transplant (Lin *et al*., 2021). On the other hand, the distribution of patients across the event status based on diabetes history and average graft survival seems to be fairly uniform.

The bottom subplot displays the relationship between the overall graft survival time and patients' HIV serostatus. From this plot, we can observe the general overview of graft performance taking into account the recipient's HIV serostatus. Among the patients who experience graft failure in the first 15 years after transplants, most have category '0' HIV serostatus followed by category '2', '1', and '4'. Patients with category '3' HIV serostatus constitute less proportion across the observation period. The overall graft survival declines with time since the transplant as indicated by more failures, especially after year 10.

### 9.3 The MIMIC-III Clinical Dataset

Medical Information Mart for Intensive Care (MIMIC-III) is a popular database that comprises anonymized clinical data for patients admitted to the Beth Israel Deaconess Medical Center in Boston, Massachusetts (Johnson et al., 2016). The dataset includes information on patient demographics, diagnoses, vital signs, medications, laboratory tests, and other types of clinical data.

The MIMIC-III dataset is often used for research purposes, such as studying trends in patient care and outcomes in the ICU and identifying factors that may affect patient outcomes. The MIMIC-III dataset is widely used in the field of healthcare and has been used in many research studies and publications. MIMIC-III database associated with both adults i.e., patients over 16 years old, and neonates (newborns) admitted to the critical care units over a specified duration.

The critical care units feature the Coronary Care Unit (CCU), Cardiac Surgery Recovery Unit (CSRU), Medical Intensive Care Unit (MICU), Surgical Intensive Care Unit (SICU), and Trauma

Surgical Intensive Care Unit (TSICU). Over 53,423 adult patients were admitted to the critical care units between 2001 and 2012, and 7838 neonates were admitted between 2001 and 2008 (Johnson *et al.,* 2016).

The MIMIC-III database consists of data collected during the normal routine of hospital operations i.e., data were downloaded from different sources including the hospital's EHR databases, archives from the critical care intensive system, and social security administration health master file (Johnson *et al.,* 2016).

The datasets comprise various classes including patient descriptions such as demographic information; interventions, which include procedures such as dialysis and imaging studies, laboratory measurements such as hematology and blood chemistry, medications, free textual data such as hospital discharge summaries, electrocardiogram reports, and progress notes, physiological information such as verified vital signs. Figure 9.7 shows the flow chart of the MIMIC-III database (Johnson *et al,* 2016).

The usage of the MIMIC-III dataset is waived from the patient's consent since there was no medical care impact and all sensitive health information related to the patients was de-identified. In this study, we employ the MIMIC-III data to predict the ICU stay duration (in days) for patients admitted multiple times to the ICU due to various health issues. For our study, we derived 46,520 patients recurrently admitted to the Intensive Care Unit (ICU) due to multiple risks. We focus on patients who were readmitted to the ICU at most five times due to several heart problems or other risks.

A subject is right censored when death occurs. Fifty-one features related to the patient's demographic records, mortality, laboratory test results and medication were derived. Out of 25,744 patients admitted to the ICU at most five times, 22,600 developed heart conditions or other risks, and 3,144 were censored. For simplicity, we generated two groups of subjects based on heart conditions and other health issues. We also introduced a total of 49 binary covariates to account for noise and redundancy information following the same procedure outlined in Section 9.1

Figure 9.8 displays a detailed summary of the event status across time stamps and patient distribution based on admission status, risks, and hospital death. Figure 9.9 shows that the patient's ICU readmission frequency is directly proportional to the length of stay in the hospital (LOS). In addition, we obtain the relationship between a patient's blood chemistry and length of stay in the ICU. Figure 9.9 shows patient distribution based on the top 25 heart issues, hospital death, and gender while Figure 9.10 displays event distribution based on patient demographic information, length of ICU and hospital stay, and mortality. In addition, we obtain the patient's heart-rate pattern based on blood pressure in Figure 9.11. For further descriptions of the data concerning some covariates-survival relationships see the Appendix 1.

Figure 9.7 displays the MIMIC-III data flow. Patient records are collected across different ICU centers and external sources into a single archive. For the protection of human subjects, the concept of privacy is contained by de-identification and date shifting before channeling the data into a public database. The users are allowed access to the database upon special request. For improvement, users can provide feedback, such as corrections, which is channeled back to the central archive.

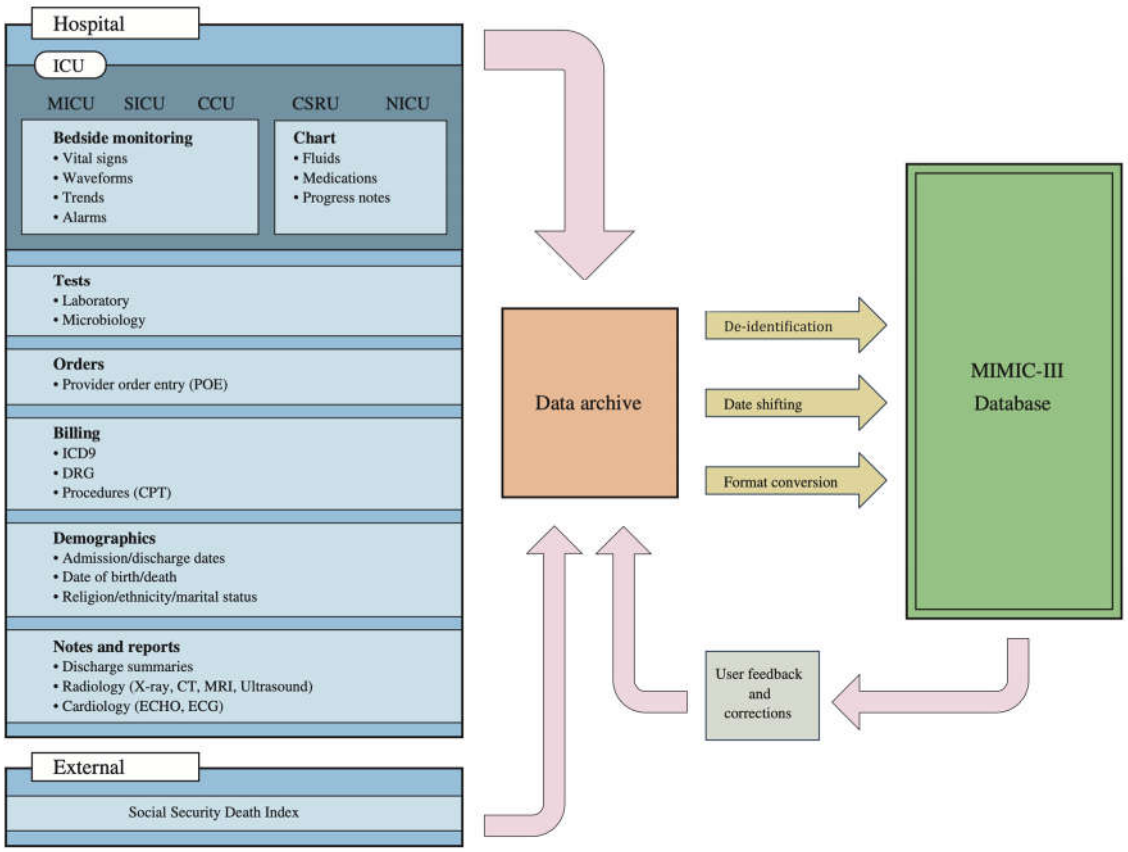
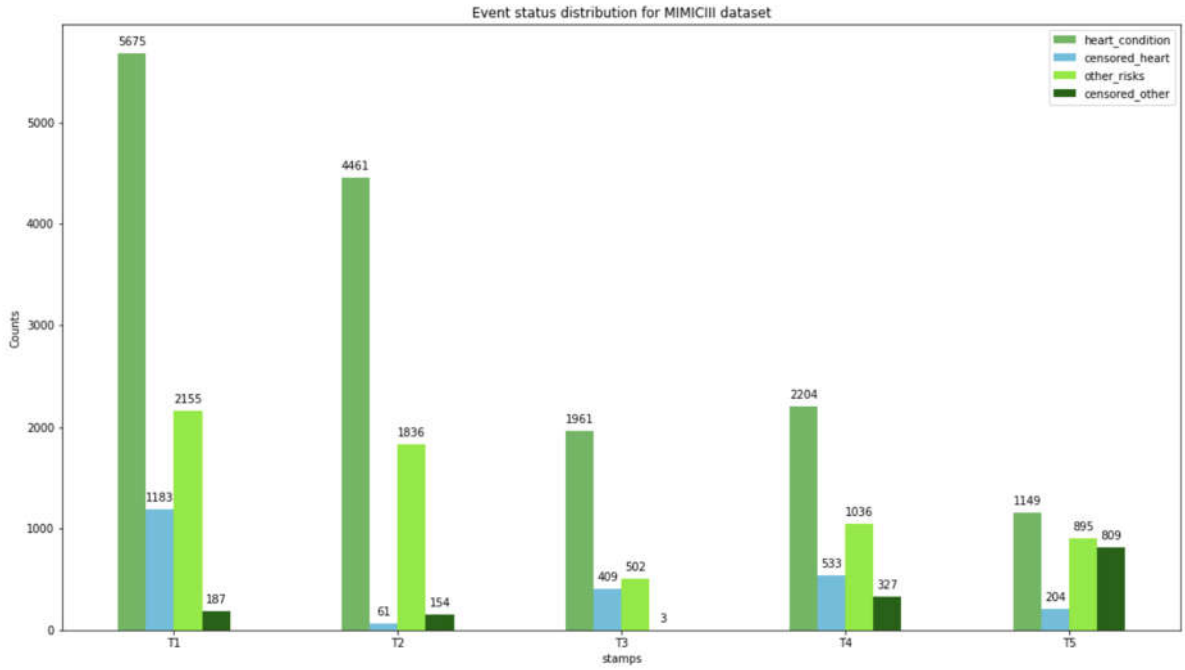



Figure 9.7. The MIMIC-III database flowchart



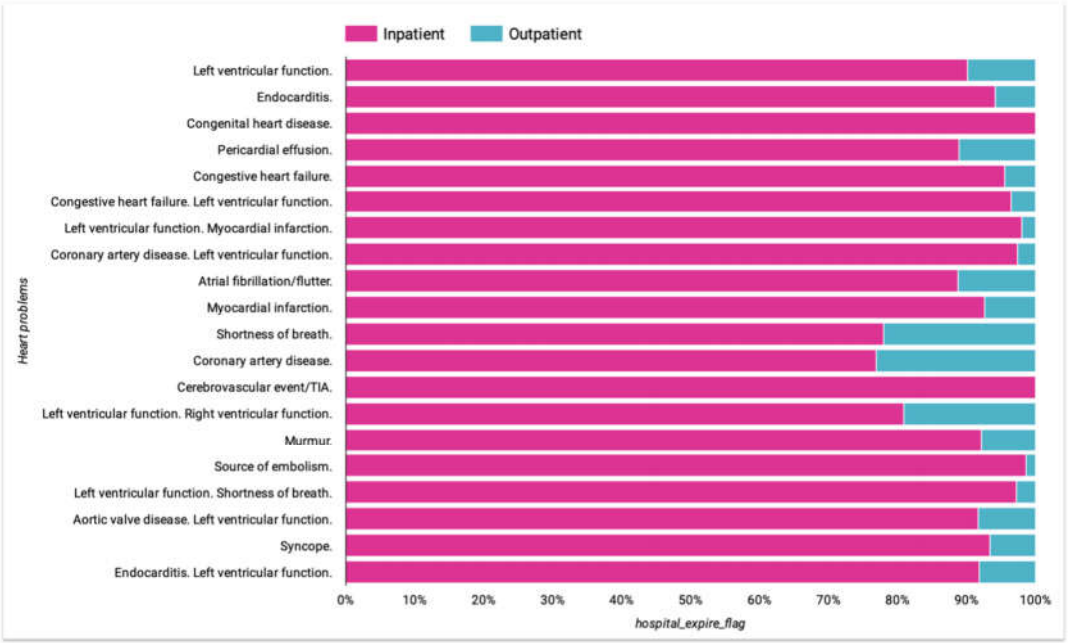
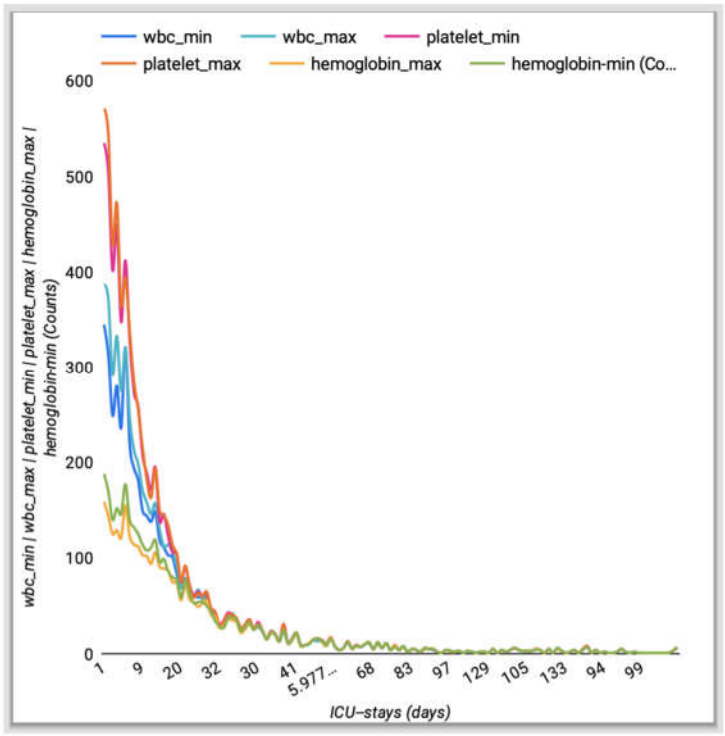




Figure 9.8. Distribution of heart conditions and overall event status across the time stamps

The top plot in Figure 9.8 shows the distribution of the event's status along the time steps. Since the classes are quite imbalanced within and between the time steps, we expect some challenges, especially when using classical approach predictive modeling. Most patients are admitted to the ICU due to heart conditions compared to other risks. The number of subjects declines with the frequency of ICU readmission. The bottom plot in Figure 9.8 gives the overall patient distribution based on hospital admission status, different heart issues, and hospital death. More than 80% of patients who suffered multiple heart problems are in-patients, while less proportion of the data is out-patient subjects.



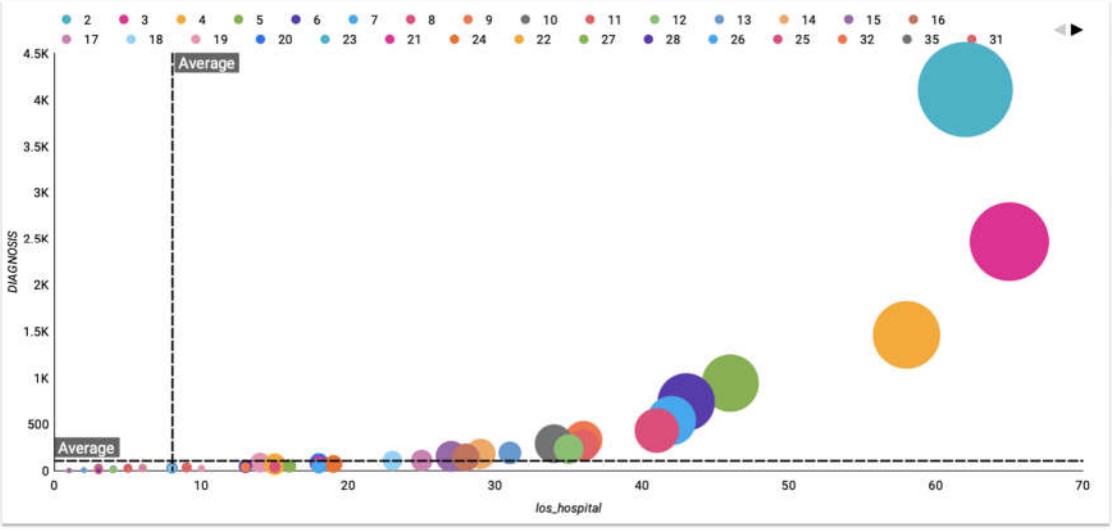
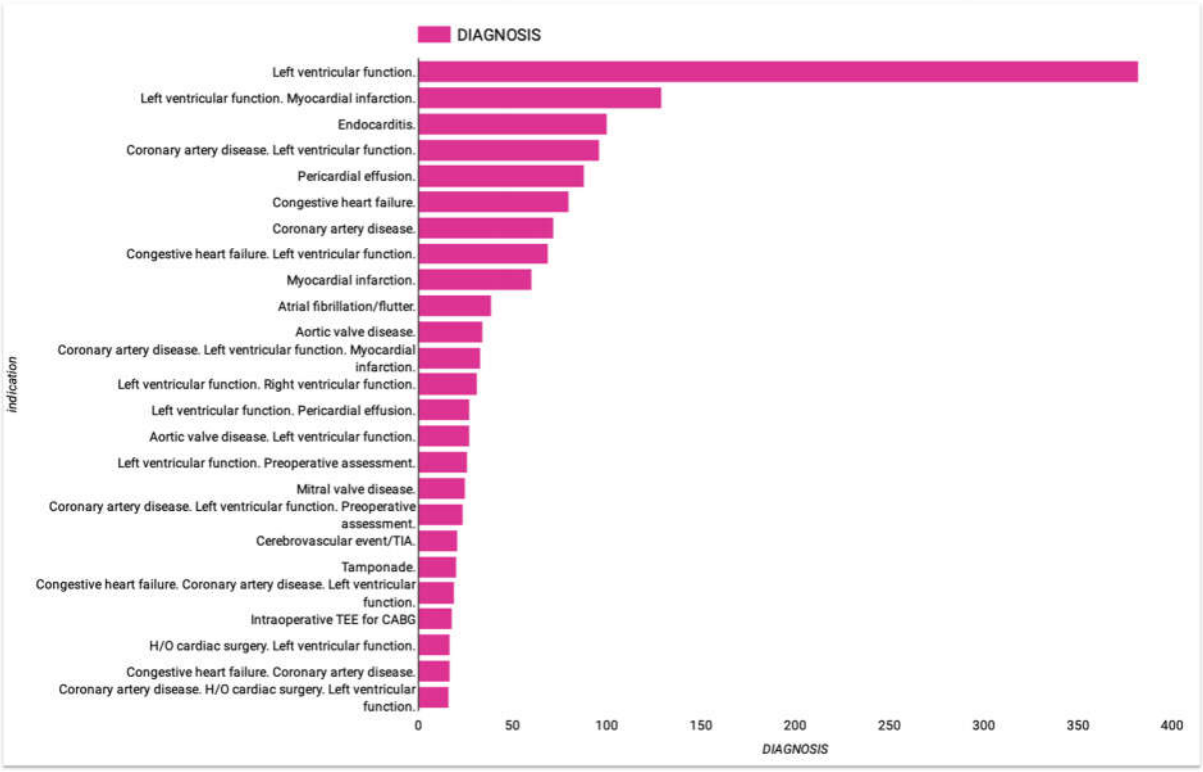




Figure 9.9. Patients distribution based on the ICU stay duration, LOS, and laboratory results

The top plot in Figure 9.9 displays patients' laboratory results in terms of cell counts concerning the number of days spent in the ICU. Except for patients who stay about five days in the ICU, patients with a large number of cell counts spent less time in the ICU compared to those with diminished cell counts. In our study, we consider patients who stayed at most 30 days in the ICU due to extremely missing covariates information.

The bottom plot in Figure 9.9 gives an overview of patient distribution based on the length of stays in the hospital and the ICU after the exclusion of censored individuals. The color of the bubbles represents the length of ICU stay, while the bubble’s size represents the hospital staying duration. The longer the patient stays at the hospital, the higher the frequency of ICU readmission.



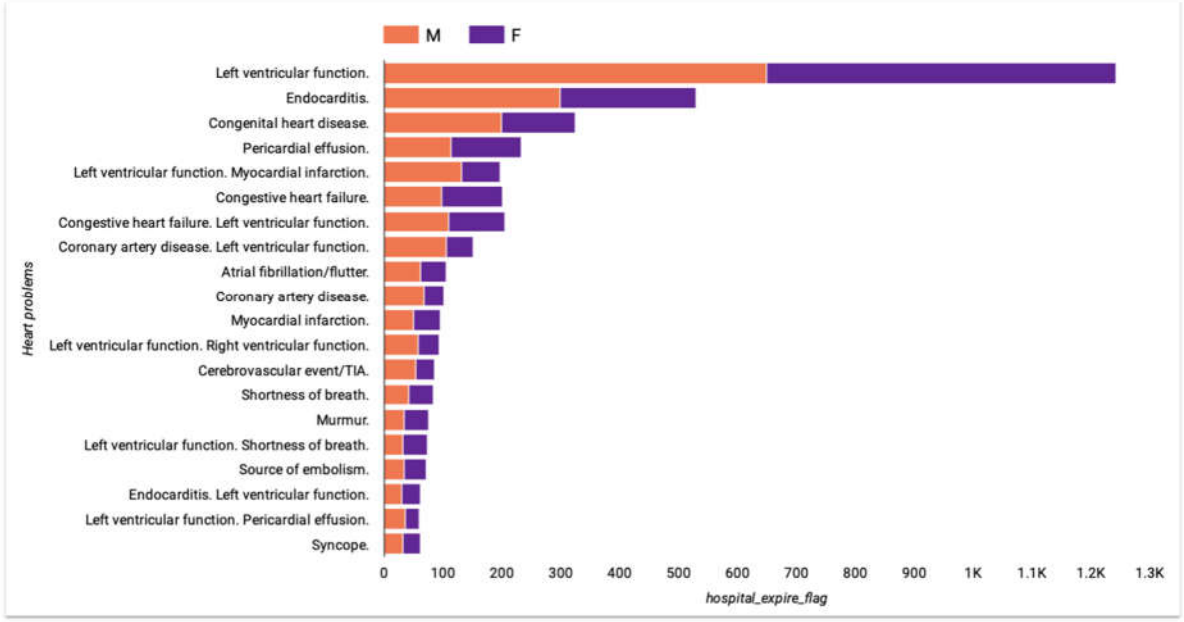




Figure 9.10. Patient distribution based on the top 25 heart issues, event status and gender

Figure 9.10 gives a detailed description of the patient's distribution based on different heart conditions concerning demographic factors. The top plot shows the overall distribution of patients over each category for events only. Most patients are admitted to the ICU due to left ventricular malfunction and myocardial infarction. We can also observe multiple heart issues in a patient, which occurred most likely after heart surgery. The bottom plot shows a detailed description by considering mortality and gender factors. The number of hospital deaths due to different heart issues is fairly distributed between male and female subjects.

| | | gender / los_hospital / admission_age / los_icu / hospital_expire_flag | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | M | | | | F | | | |
| DIAGNOS... | ethnicity_grouped | los_hospital | admission_age | los_icu | hospital_expire... | los_hospital | admission_age | los_icu | hospital_expire... |
| 1 | white | 2.7K | 82 | 2.7K | 2.7K | 1.6K | 82 | 1.6K | 1.6K |
| | unknown | 874 | 63 | 874 | 874 | 495 | 58 | 495 | 495 |
| | other | 104 | 46 | 104 | 104 | 55 | 34 | 55 | 55 |
| | native | 3 | 2 | 3 | 3 | 3 | 3 | 3 | 3 |
| | hispanic | 73 | 37 | 73 | 73 | 53 | 28 | 53 | 53 |
| | black | 215 | 56 | 215 | 215 | 332 | 68 | 332 | 332 |
| | asian | 50 | 27 | 50 | 50 | 45 | 27 | 45 | 45 |
| 0 | white | 6K | 83 | 6K | 6K | 5K | 86 | 5K | 5K |
| | unknown | 826 | 79 | 822 | 826 | 743 | 75 | 743 | 743 |
| | other | 319 | 68 | 318 | 319 | 228 | 53 | 228 | 228 |
| | native | 11 | 2 | 11 | 11 | 6 | 4 | 6 | 6 |
| | hispanic | 369 | 61 | 369 | 369 | 248 | 57 | 248 | 248 |
| | black | 798 | 73 | 798 | 798 | 1K | 75 | 1K | 1K |
| | asian | 413 | 57 | 413 | 413 | 318 | 56 | 318 | 318 |

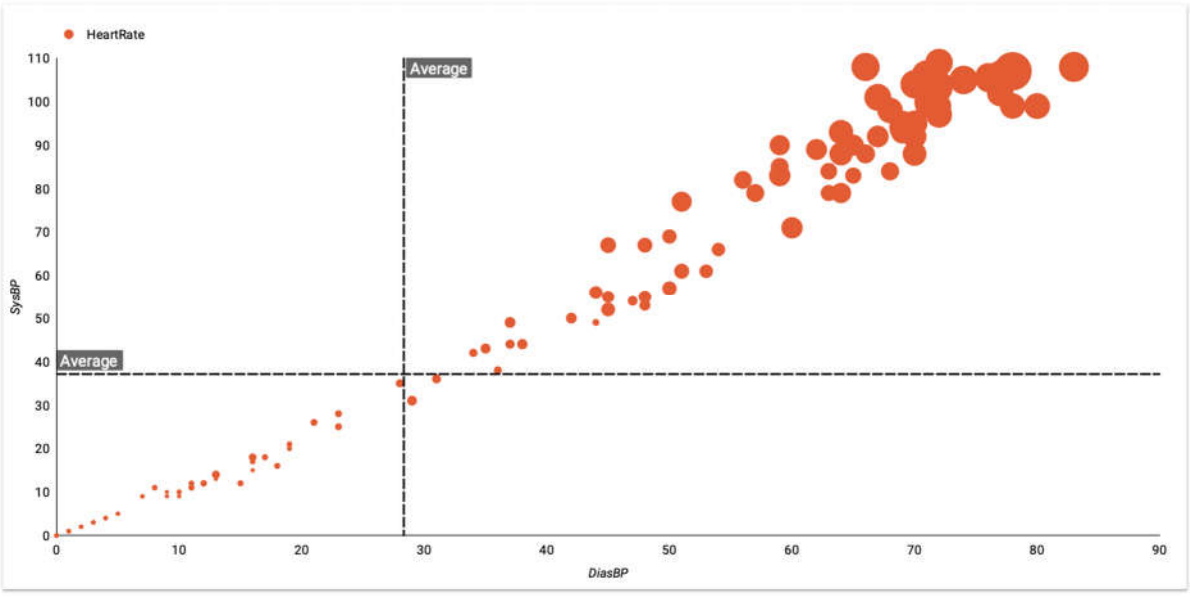



Figure 9.11. Patient distribution based on event status, blood pressure, ICU stay duration, LOS, and demographics features

The top diagram in Figure 9.11 is the pivot table which displays information concerning patients' ethnic group, age at the admission stage, length of stay in the ICU (LOS-ICU), length of stay in the hospital (LOS-hospital), hospital mortality, and event status. From the table, we can see that all patients in the hospital were at least admitted to the ICU. Hospital mortality is higher for male patients compared to female patients. Older patients have higher ICU readmission frequency and hospital mortality.

The bottom subplot shows the heart rate distribution based on the patient's blood pressure. A low heart rate is indicated by smaller dots, while large dots represent a high heart rate. From the plot, we can see that a low heart rate is associated with low blood pressure, while patients with high blood pressure have a faster heart rate.

### 9.4. Data Preparation for a Machine Learning Task

Data preparation is the process of cleaning, formatting, and organizing data in a way that makes it suitable for analysis and machine learning tasks. Data preparation involves a range of activities, such as gathering and collecting data from various sources, removing missing or corrupted data, formatting the data in a consistent way, and converting the data into a format that is suitable for the task at hand. Data preparation may also involve normalizing or standardizing the data, selecting and constructing relevant features from the raw data, and splitting the data into training and test sets (Talend, 2022).

Data preparation is an essential step in the machine-learning process, as it can significantly affect the efficiency and accuracy of the trained algorithm. In practice, we can categorize data into two main groups including structured and unstructured data. Structured data is well organized and specified information usually stored in a predefined format, while unstructured data involves a mixture of information that exists in its raw forms. Examples of structured data include the relational database, where data are formatted into precisely defined fields, while unstructured data are more qualitative rather than quantitative and include information such as texts, charts, images, or sensor data (Li *et al.,* 2022).

Structured data are easy to use and allow access to a wide variety of analytics tools, while unstructured data are complex to use and require experts on limited analytic tools. Since structured data are predefined, they are limited in use and storage options, while unstructured data are format free and can be accumulated at a faster rate (Talend, 2022). Data preprocessing is a crucial step in any data science project in order to draw essential insights and build relevant predictors or classifiers. In practice, machine learning algorithms require data to be in a particular format and shape.

For better and more efficient performance of an algorithm, we require data that is valid, clean, rich in features, and compatible to achieve the desired goal for a given task. Data preparation involves dealing with missing or incomplete information, improper data value format, extreme information which can either be outliers or anomalies, inconsistency and unstandardized categories, and sparse attributes that occur mainly due to fuzzy matching and feature engineering techniques.

Although data preprocessing procedures are unique to the data types, algorithm, and desired goal for the project there exist common steps for any machine learning project including data cleaning, feature selections, data augmentation, feature engineering, data transformation, and dimensionality reduction (Kernbach *et al.,* 2022).

In order to estimate the performance of a machine learning algorithm, data splitting is an essential routine. For proper generalization, an algorithm not only requires training on a sufficient dataset but also needs to perform well on unseen future data. Generally, for a machine learning project, it is advised to at least retain a small portion of data to assess the generalization of a learned algorithm. Before deployment of the machine learning algorithm, it is essential to follow the training, validation, and testing pipelines.

Since machine learning algorithms are data-driven techniques, especially during training, it is crucial to employ a proper data-splitting technique to maximize the overall performance and generalization of the learned algorithm. There are several techniques widely applied in practice to split the data into train, validation, and test sets. When we have enough data the commonly used

procedure involves the (70-30) or (80-20) or (90-10) percentage splits for the train and test sets, while some proportion of the training data may be retained for validation purposes.

The training dataset is used to train the machine learning algorithm. Usually, the performance of the model on the training data may be almost perfect, and it is not advisable to evaluate the machine learning algorithm relying on the training set because it may simply be a memorization. The validation dataset is used to evaluate the model concurrently during training. For optimization purposes, the validation data plays a crucial role in assisting the tuning of the model's hyper-parameters (Talend, 2022). We employ the testing dataset to evaluate the learned algorithm on the unseen examples before sending it to production.

In practice, for some domains, due to the nature of the problem, we may encounter a limited dataset. For such a scenario, we cannot afford to withhold the data following the above splitting technique because we would want to utilize every limited example available for training purposes. Possible solutions may involve restriction to machine learning algorithms that are not prone to overfitting such as support vector machine (SVM) or using cross-validation techniques. The common problem encountered in a machine learning project is overfitting or underfitting the learned algorithm (Kernbach & Staartjes , 2022).

Underfitting is a problem in machine learning that occurs when a model is too simple and is unable to capture the underlying structure of the data. This can lead to poor performance on the training data and poor generalization to new, unseen data (Kernbach & Staartjes, 2022). Underfitting is often caused by using a model that is too simple or by not providing the model with enough training data. It can also be caused by using poor quality or irrelevant features, or by not properly tuning the model's hyperparameters.

Generally, under this circumstance the performance of the model on the training data is unsatisfactory. To address underfitting, we may need to use a more complex model, provide the model with more training data, or use more relevant or higher-quality features. We may also need to tune the model's hyperparameters to find the best balance between complexity and fit the data.

It is necessary to leverage the dataset to perform hyperparameters' tuning using techniques such as cross-validation (Kernbach & Staartjes, 2022).

Overfitting is often caused by using a model that is too complex or by using a large number of features, relative to the amount of training data. It can also be caused by using irrelevant features' matrix, or by not properly tuning the model's hyperparameters. To address overfitting, we may need to use a simpler model, use fewer features, or use regularization techniques to constrain the model's complexity. We may also need to optimize the model's hyperparameters to find the best balance between the complexity and fit of the model. Figure 9.12 gives an overview of underfitting and overfitting a machine learning algorithm.

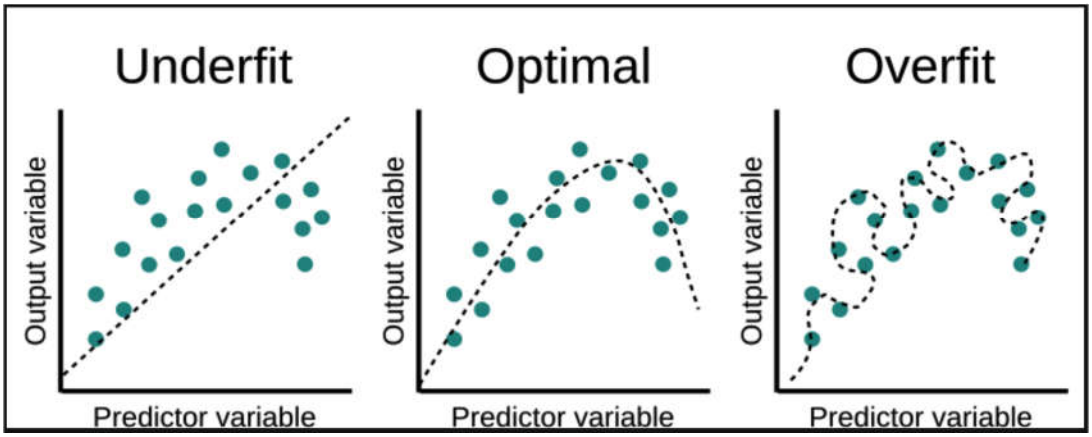



Figure 9.12. Overfit, underfit, and the optimal fit of a learning algorithm (Rathod, n.d)

The best way to deal with the overfitting and underfitting problems is by employing model selection techniques such as cross-validation. Cross-validation is a statistical concept used by machine learning practitioners to optimize the performance of the model, especially when we encounter limited data.

This technique involves partitioning the data into several groups referred to as folds. The idea behind cross-validation is to run the analysis on each fold iteratively and take the average to obtain the final results. There are several forms of cross-validation techniques including K-folds cross-

validation, stratified K-folds cross-validation, leave-n-out (LNO), and leave-one-out (LOO) cross-validation (Brownlee, 2022).

The following steps elaborate the idea behind the K-folds cross-validation: - Randomly, split the dataset into K subsets (samples also called folds), train the model on the first K-1 folds, and evaluate the performance of each model using the remaining fold, iteratively repeat the process and take the average performance across the folds (Brownlee, 2022).

This procedure may reduce bias in the resulting model since every example has a chance of appearing in the train and validation set. When we have a class imbalance issue for a classification task the K-folds cross-validation may not suffice. This is because some folds may contain data of similar class only which lead to bias in the final model.

To address this issue, we stratify the data accordingly to ensure the appropriate proportion of examples of each class within a fold. The K-folds cross-validation after stratification is called stratified K-folds cross-validation. Figure 9.13 displays the conceptualization of the K-folds cross-validation.

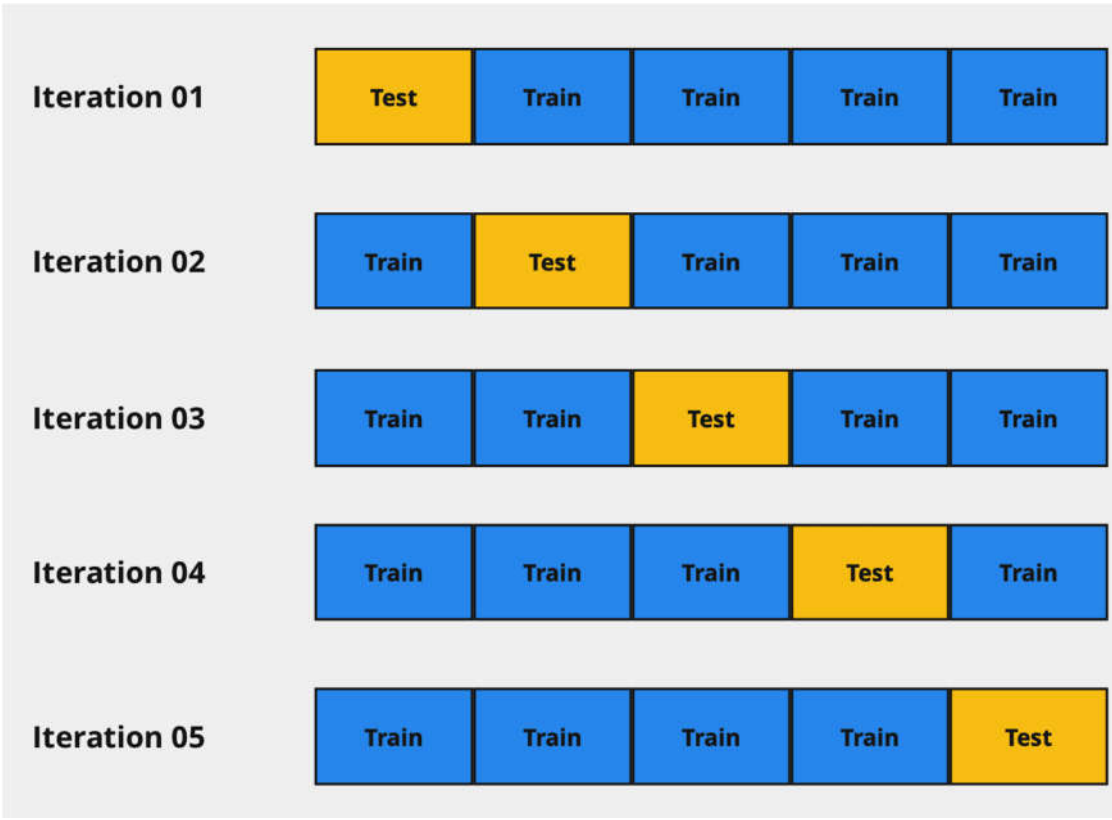



Figure 9.13. The K-folds cross-validation (*How to Implement K Fold Cross-Validation in Scikit-Learn*, n.d.)

The leave-n-out cross-validation (LNO) involves an iteratively random selection of 'n' examples from the dataset of size N. For all possible combinations of 'n' examples at every iteration, we train the model on 'N-n' data points and use the remaining 'n' to evaluate the model (Buintick et al., 2013).

The most simplified version of LNO is the leave-one-out (LOT) cross-validation where 'n' is set to one, which results in N number of combinations for the dataset of size N. Since our model is complex and we have enough data, we employ the hold-out data splitting technique by considering the ratio of 80:20 train-test split (Buintick et al., 2013).

### 9.5. Preparation of Survival Data for the RNN Model

Consider the sequence of length $\{X_t\}_{t=1}^{l}$ , where $l$ is the length of the sequence. The RNN model receives the input of shape $[B, l, P]$, where $B$ is the batch size (total number of examples supplied at the time $t$), $l$ is the length of the sequence, and $P$ is the number of features (Paszke *et al.*, 2019). Refers to our complex survival problem in Section 3.2 we have the dataset of form

$$\{X_t^i, T_t^i, I_t^i, R_t^i\}_{i=1}^{N}.$$

Using the sequence of covariates $\{X_t\}$, we are required to produce the survival distribution and the risks-specific outcomes as output. The sequence length for our case corresponds to the number of recurrent (time stamps). For cost computation, both $\{T_t \; and \; R_t\}$ are supplied with the aid of an indicator function $\{I_t\}$ as a mask.

In our study, we have the input of shape $[N = 30{,}000, l = 5, P = 61]$, for the case of the synthetic dataset, $[N = 25{,}744, l = 5, P = 100]$, for the case of the MIMIC-III clinical dataset, and $[N = 97{,}050, l = 4, P = 100]$, for the UNOS-OPTN dataset. We use the ratio of 80:20 to split the data into training and testing sets. Further, we keep 20% of the training data as a validation set.

Since this project is implemented in PyTorch with the PyTorchLightining framework, we utilized PyTorch Dataset and DataLoader classes to customize our data into the proper format. PyTorch's Dataset and DataLoader are utilities for loading and manipulating data in PyTorch.

A Dataset is a class that represents a dataset and defines how to access the data. It should inherit from the base 'torch.utils.data.Dataset' class and implement two methods: '__len__ 'and '__getitem__'. The '__len__' method should return the number of samples in the dataset, and the '__getitem__' method should return a sample at a given index.

A DataLoader is a utility that loads data in parallel from a dataset object. It takes in several arguments, including the dataset object, a batch size, and whether or not to shuffle the data. The

batch size determines the number of samples per batch, and shuffling the data means that the order of the samples will be randomized each time the data is loaded.

One of the main advantages of using DataLoader is that it allows loading data in parallel using multiple workers. In summary, a Dataset is a class that represents a dataset, it defines how to access the data, it should be inherited from the base 'torch.utils.data.Dataset' class and implement two methods '__len__' and '__getitem__'. DataLoader is a utility that loads data in parallel from a dataset object, it allows loading data in parallel using multiple workers which can speed up the data loading process, especially when working with large datasets (Paszke *et al.*, 2019).

# 10. TRAINING PROCEDURES

For experimentation, the dataset is preprocessed by scaling all numeric features and creating the embedding for categorical variables. Following the work of Rietschel *et al. (*2018) and Lee & Carlin(2012), we added a total of 49 synthetic binary covariates to the MIMIC-III clinical and the simulated datasets to demonstrate the power of the external autoencoder.

We appropriately organized the data and obtained a split of 80% to 20% for the training and testing sets. Further, we consider 20% of the training set as the validation set. We implement our model by exploring RNN, GRU, and LSTM architectures with two layers and a hidden size set as a hyperparameter ranging from (32 to 128).

We leveraged batch normalization and dropout layers as regularizers. The attention mechanism and risk-specific subnetworks consist of a simple fully connected feedforward network with a softmax output. The hidden size and the number of layers for the external autoencoder are hyperparameters optimized directly by the network.

We employ Adam optimizer and leverage the ReduceLROnPlateau (Paszke *et al.*, 2019) learning rate scheduler from Pytorch with a factor of 0.1, threshold of $e^{-3}$, and minimum decay of $e^{-5}$. We trained our model via backpropagation with the gradient clipping threshold of 1.5, and $(\epsilon_1, \epsilon_2)$ for the ranking, loss is set as hyperparameters.

We perform hyperparameter tuning through a simple exhaustive grid search (Bergstra *et al.,* 2012).

### 10.1. Batch Normalization and Dropout

Batch normalization and dropout are two techniques used in deep learning to improve the performance and stability of neural networks. Batch normalization is a technique that normalizes the activations of a layer in a neural network across a batch of data. This helps to reduce the internal covariate shift, which occurs when the distribution of the input data to a layer changes during training (Ioffe *et al.*, 2015; Srivastava *et al.,* 2014).

By normalizing the activations, batch normalization helps to stabilize the training process and make it more efficient. Dropout is a technique that randomly drops out a certain percentage of the weights in a layer during training.

This helps to prevent overfitting, which happens if the algorithm is too complex and starts to memorize the training data rather than generalize to new examples. Dropout acts as a form of regularization by forcing the algorithm to learn multiple independent manifolds, rather than relying on a few specific neurons.

Batch normalization is typically used after the activation function in a layer, and dropout is typically used after the linear transformation in a layer. Both techniques are typically used when training the network and are switched off during the evaluation and testing stages. In summary, Batch normalization standardizes the activations of a layer in a neural network across a batch of data, it helps to reduce the variances and stabilize the training process.

### 10.2. ReduceLROnPlateau

In machine learning, the learning rate is a hyperparameter that determines how quickly the model learns the underlying patterns in the training data. The learning rate is used during the optimization process of training a model, where the goal is to minimize the error between the predicted outcome and the actual output (Erizmann, 2021).

The learning rate is typically set at the beginning of training and can significantly affect the performance of the network. A common technique for selecting a learning rate is to try out different values and pick the best result on a validation set. In our study, we experimented with different learning rates by leveraging the ReduceLROnPlateau learning rate scheduler (Paszke *et al.*, 2019).

ReduceLROnPlateau is a method for dynamically reducing the learning rate in an algorithm during training. It is typically used as a callback function in the training process. The method monitors a chosen metric, such as the validation loss, and if the metric does not improve for a certain number

of epochs, the learning rate is reduced by a specified factor. This allows the model to converge faster and reach a better local minimum.

### 10.3. Hyperparameter Optimization

Hyperparameter tuning involves procedures of finding the best set of hyperparameters for an AI algorithm, in order to achieve the best results for the task at hand. Since we can not leverage the data to directly optimize the network hyperparameters, careful choice and assessment of their values are usually applied during the evaluation stage (Goodfellow *et al*., 2016).

There are several techniques for hyperparameter optimization, some of which include:

- Grid search: This is a hyperparameter tuning technique in machine learning where a set of hyperparameters for a given model is defined, and a search is conducted over all possible combinations of these hyperparameters (Erizmann, 2021).
- Random search: This method involves randomly sampling hyperparameter values from a predefined range (Lee *et al.*, 2018).
- Bayesian optimization: This is a probabilistic approach that uses a model of the objective function to guide the search for the optimal hyperparameters. It balances exploration and exploitation by using a probabilistic model to predict the expected improvement in the objective function for each set of hyperparameters (Lee *et al.*, 2018).
- Genetic algorithms: This algorithm begins by creating an initial population of random hyperparameters. Each set of hyperparameters is evaluated using a performance metric like accuracy or loss, and a fitness score is assigned to it. The algorithm then selects the fittest individuals from the population and performs genetic operations like crossover and mutation to create a new generation of potential solutions. This process is repeated for multiple generations until the algorithm converges to the optimal set of hyperparameters (Xiao *et al*., 2020).

These are some of the most used methods to optimize the hyperparameters of a model, however, other methods can also be used without limitations. As mentioned above, we employ the Grid search technique with a window of (32 to 128) for the hidden size of RNN, LSTM, and GRU, (1

to 3) for the number of layers of the external autoencoder, and (0 to 1) for the hyperparameters associated with the ranking loss ($L_r$) and autoencoder loss ($L_a$).

For simplicity, we set $\beta_1 = \beta_2 = \beta_3 = 1$ in $L_t = \beta_1 L_l + \beta_2 L_r + \beta_3 L_a$ .

We leverage the Test Tube library from PyTorch for hyperparameters tuning. Test Tube is a Python library for managing machine learning experiments. It can be used for hyperparameter tuning by specifying the hyperparameters to be tuned, the search space for each hyperparameter and a performance metric to evaluate the model.

TestTube also allows you to define the optimization algorithm to be used for tuning, such as grid search, random search, or Bayesian optimization. It also provides a way to track the experiments, save the results and load previous experiments. It also allows users to use different backends such as MongoDB, SQLAlchemy, etc to store the experiment results, which can be helpful if you have a large number of experiments.

Users can also specify the resources (such as GPU, CPU) that are required for the experiment to run, and TestTube will automatically match the resources to the available hardware. TestTube also provides a way to visualize the results of the experiments, such as plotting the performance metric over time or comparing the performance of different runs of the experiment (Paszke *et al.*, 2019).

### 10.4. Optimizer Selection

When training an AI system, the goal is to find the optimal set of model parameters that can accurately predict the outcomes. However, finding the optimal parameters is a complex and iterative procedure that requires adjusting the model parameters in a systematic way to improve its performance. Optimizers are used to automate this process by iteratively updating the network parameters based on the feedback received from the training data. They work by estimating the slopes of the loss function concerning the model parameters and then making the adjustment in the direction that reduces the loss (Ghorbani *et al.*, 2019).

In practice, some commonly used optimizers include:

- Stochastic Gradient Descent (SGD): SGD is an optimization algorithm commonly used in machine learning for training models that involve large amounts of data. It is a variant of the gradient descent algorithm that updates the model parameters based on a randomly selected subset of the training data, rather than the entire dataset (Goodfellow *et al.,* 2016).
- Adam: This is an extension of the gradient descent algorithm that includes adaptive learning rates for each weight in the neural network, allowing the learning rate to adapt to the characteristics of the gradients observed during training. This adaptive learning rate reduces the risk of getting stuck in local minima and accelerates the convergence of the algorithm (Kingma *et al.*, 2014).
- RMSprop: This optimizer also adapts the learning rate of each parameter based on historical gradient information, but it uses a moving average of the squared gradient instead of the gradient itself (Tieleman & Hinton, 2012).
- Adagrad: This optimizer also adapts the learning rate of each parameter, but it uses a different approach. It uses a per-parameter learning rate that is inversely proportional to the historical gradient information (Lydia *et al.*, 2019).
- L-BFGS: This is a Quasi-Newton method optimizer that computes the Hessian matrix of the loss function and uses it to update the model's parameters. It's computationally expensive but often converges faster than other optimizers.

It is a common procedure to try different optimizers and select the one that gives the best results in the validation set. For our study, we employed an Adam optimizer.

The training process for an AI model typically involves the following steps (Goodfellow *et al.,* 2016):

- Collect and prepare the data: This involves gathering a dataset that is representative of the problem you are trying to solve and cleaning or preprocessing the data to ensure it is in a format that can be used to train the model.
- Split the data into training and testing sets: The dataset is usually split into two sets, one for training the model and one for evaluating its performance.

- Define the model architecture: This step involves deciding on the specific type of model to use (e.g. a neural network) and defining its computational graph.
- Train the algorithm: During the training process, the model is exposed to a dataset containing input data and corresponding output values, and it uses this data to adjust its internal parameters to optimize its predictions or decisions. The goal of training is to find the optimal set of parameters that minimizes the difference between the model's predictions and the true output values for the training data.
- Evaluate the model: The trained model is then evaluated on the test data to measure its performance and identify any issues that need to be addressed.
- Fine-tune the model: This is an iterative process that involves changing the values of the hyperparameters, training the model, and evaluating its performance on a validation set. The hyperparameters that lead to the best performance on the validation set are then selected as the final settings for the model.
- Deploy the model: The deployment process involves several steps, including converting the trained model to a format that can be used by the deployment platform, optimizing the model for efficient inference on the target hardware, and setting up a scalable and secure infrastructure for serving the model.

Following the above steps, we trained our model with and without an external autoencoder. We selected the Fine-Gray model (Fine & Gray, 1999), the DeepHit (Lee *et al.,* 2018), the random survival forest (RSF) (Ishwaran *et al.,* 2014), CRESA (Gupta *et al.,* 2019), and DeepSurv (Katzman *et al.,* 2018) as benchmarks.

We trained the Fine-Gray and the RSF models for all time steps using the **cmprsk** (Gray & Gray, 2014) and **randomForestSRC** (Ishwaran et al., 2023) packages in *R*. We trained the DeepHit and CRESA following the work of (Gupta *et al.,* 2019).

### 10.5. Results and Discussion

Using the validation dataset, we perform 20 trials for the hyperparameters optimization. Table 10.1 shows the values of the optimal hyperparameters for the best model across the datasets and models.

Using the test dataset, we evaluate the performance of our model by computing the metric given in Equations 6.2.8 and 6.2.9 (Marthin & Tutkun, 2023).

We obtain the C-index per risk across the timestamps and the Brier scores across the time stamps. Table 10.1 displays the optimal values of hyperparameters obtained through the grid search. For CmpXRnnSurv_AE trained on MIMIC-III clinical dataset, the optimal hyperparameters are 64 units for the LSTM hidden size, two hidden layers for the external autoencoder, and 37 neurons for the autoencoders' hidden size.

The optimal ranking loss parameters $\epsilon_1 = \epsilon_2$ is 0.358, the optimal sparsity and decaying rate parameters for the autoencoders' loss are $\alpha = 0.0046$ and $\beta = 0.0925$, the best validation loss is 0.4193, and the learning rate is 0.0001. Without an external autoencoder (CmpXrnnSurv) trained on the MIMIC-III dataset has optimal hyperparameters of 0.001 for learning rate, 128 units for LSTM hidden size, 0.510 for ranking loss parameters, and best validation loss of 0.6905. We can draw analogous explanations across UNOS-OPTN(KIDPAN) and the synthetic datasets.

Table 10.1. Optimal Hyperparameters across datasets and models

| Optimal values of hyper-parameters for 20 trials | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **Dataset** | **Model** | **Learning rate** | **h_dim_LSTM** | **Layers_AE** | **h_dim_AE** | $\epsilon_1 = \epsilon_2$ | $\alpha$ | $\beta$ | **Best validation loss** |
| MIMIC-III clinical | **CmpXRnnSurv_AE** | $e^{-4}$ | 64 | 2 | 37 | 0.358 | 0.0046 | 0.0925 | 0.4193 |
| | CmpXRnnSurv | 0.001 | 128 | - | - | 0.510 | - | - | 0.6905 |
| UNOS-OPTN (KIDPAN) | **CmpXRnnSurv_AE** | 0.001 | 128 | 2 | 56 | 0.492 | 0.0115 | 0.0186 | 0.3027 |
| | CmpXRnnSurv | $e^{-5}$ | 128 | - | - | 0.453 | - | - | 0.5532 |
| Synthetic | **CmpXRnnSurv_AE** | 0.001 | 64 | 2 | 12 | 0.374 | 0.0081 | 0.0141 | 0.3825 |
| | CmpXRnnSurv | $e^{-4}$ | 128 | - | - | 0.546 | - | - | 0.7084 |

Table 10.2. Model selection based on architecture performance

| Synthetic dataset | | | |
|---|---|---|---|
| **Model** | **Best C_t score** | **Best B_t score** | **Rank** |
| CmpXRnnSurv-RNN | 0.81 | 0.6588 | 2 |
| CmpXRnnSurv_AE-RNN | 0.85 | 0.4816 | |
| CmpXRnnSurv-GRU | 0.80 | 0.6601 | 3 |
| CmpXRnnSurv_AE-GRU | 0.85 | 0.4798 | |
| CmpXRnnSurv-LSTM | 0.83 | 0.6582 | **1** |
| CmpXRnnSurv_AE-LSTM | **0.86** | **0.4794** | |
| **MIMIC-III-Clinical dataset** | | | |
| **Model** | **Best C_t score** | **Best B_t score** | **Rank** |
| CmpXRnnSurv-RNN | 0.87 | 0.4980 | 3 |
| CmpXRnnSurv_AE-RNN | 0.89 | 0.3199 | |
| CmpXRnnSurv-GRU | 0.86 | 0.4931 | 2 |
| CmpXRnnSurv_AE-GRU | 0.88 | 0.3206 | |
| CmpXRnnSurv-LSTM | 0.86 | 0.4914 | **1** |
| CmpXRnnSurv_AE-LSTM | **0.89** | **0.3197** | |
| **UNOS-OPTN(KIDPAN) Dataset** | | | |
| **Model** | **Best C_t score** | **Best B_t score** | **Rank** |
| CmpXRnnSurv-RNN | 0.87 | 0.4392 | 3 |
| CmpXRnnSurv_AE-RNN | 0.91 | 0.2055 | |
| CmpXRnnSurv-GRU | 0.89 | 0.4325 | 2 |
| CmpXRnnSurv_AE-GRU | 0.90 | 0.2069 | |
| CmpXRnnSurv-LSTM | 0.89 | 0.4311 | **1** |
| CmpXRnnSurv_AE-LSTM | **0.92** | **0.2036** | |

Table 10.2 displays the summary results of the model's evaluation on the testing dataset. We trained our network with and without an external autoencoder (CmpXRnnSurv_AE and

CmpXRnnSurv). For performance evaluation, we compute the weighted time-dependent concordance index and time-dependent Brier scores across the datasets.

From Table 10.2, CmpXRnnSurv_AE and CmpXRNNSurv have the best performance when an LSTM architecture is employed. CmpXRnnSurv_AE reaches the best score of 92% for the Ct index and predicts discrepancy of 20% from the ground truth outcomes for the UNOS-OPTN(KIDPAN) dataset. Across the datasets, except for the synthetic dataset, we observe the better performance of the GRU architecture.

Since our model under the LSTM architecture slightly provided better predictions, we present its summary results across the datasets and models in Tables 10.3 and 10.4 respectively. As seen in Tables 10.3 and 10.4, the CmpXRnnSurv-AE outperformed the benchmark models across all datasets except for the first time stamp (Marthin & Tutkun, 2023).

We can notice that the CmpXRnnSurv-AE attains the highest performance when trained on UNOS-OPTN (KIDPAN) dataset with a C-index up to 0.92 and a Brier score of 0.2036, while the RSF has the lowest performance with a C-index of 0.43 and a Brier score of 0.9330. These results were due to the noise invariant provided by the external autoencoder, information transfers through the residual connection, and the ability to handle the complex correlation structure by computing the WCIF.

Without an external autoencoder, our model (CmpXRnnSurv) still gives better results, especially on the MIMIC III and the synthetic datasets, while CRESA demonstrated competitive performance for the UNOS-OPTN(KIDPAN) dataset. This performance highlighted the ability of our model to handle data complexity by using the RIW to obtain the WCIF instead of the basic CIF. The Fine-Gray and the DeepHit models provide better performance only at the first time-step due to the inability to handle noise and complex correlational structure in the data. The RSF gives the worst performance across all datasets. In Appendices **1G, 1H,** and **1I** we provide the predicted WCIF and WOS for the selected subjects.

Table 10.3. The weighted time dependent concordance indices

| Synthetic dataset | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Risk 1 | | | | | Risk 2 | | | | |
| | Time stamps | | | | | | | | | |
| **Model** | **T1** | **T2** | **T3** | **T4** | **T5** | **T1** | **T2** | **T3** | **T4** | **T5** |
| **CRESA** | 0.53 | 0.65 | 0.71 | 0.77 | 0.59 | 0.63 | 0.79 | 0.72 | 0.69 | **0.75** |
| **FINE-GRAY** | **0.55** | 0.44 | 0.49 | 0.49 | 0.51 | 0.53 | 0.50 | 0.46 | 0.49 | 0.50 |
| **RSF** | 0.49 | 0.50 | 0.43 | 0.45 | 0.43 | 0.44 | 0.49 | 0.44 | 0.48 | 0.45 |
| **CmpXRnnSurv** | 0.52 | 0.68 | 0.74 | 0.80 | 0.71 | 0.61 | 0.83 | 0.76 | 0.70 | 0.72 |
| **CmpXRnnSurv_AE** | 0.54 | **0.73** | **0.77** | **0.83** | **0.75** | 0.64 | **0.86** | **0.80** | **0.73** | **0.75** |
| **DeepHit** | **0.55** | 0.50 | 0.57 | 0.61 | 0.49 | **0.66** | 0.55 | 0.51 | 0.48 | 0.50 |
| **DeepSurv** | **0.56** | 0.51 | 0.55 | 0.50 | 0.48 | **0.52** | 0.49 | 0.49 | 0.51 | 0.48 |
| | **MIMIC-III Clinical dataset** | | | | | | | | | |
| | **Heart problems** | | | | | **Other risks** | | | | |
| | T1 | T2 | T3 | T4 | T5 | T1 | T2 | T3 | T4 | T5 |
| **CRESA** | 0.53 | 0.79 | 0.74 | 0.85 | 0.79 | 0.55 | 0.76 | 0.85 | 0.81 | 0.78 |
| **FINE-GRAY** | **0.61** | 0.48 | 0.44 | 0.51 | 0.49 | **0.59** | 0.45 | 0.50 | 0.47 | 0.51 |
| **RSF** | 0.42 | 0.50 | 0.46 | 0.52 | 0.50 | 0.48 | 0.43 | 0.44 | 0.50 | 0.49 |
| **CmpXRnnSurv** | 0.57 | 0.76 | 0.75 | 0.83 | 0.81 | 0.54 | 0.75 | 0.86 | 0.79 | 0.80 |
| **CmpXRnnSurv_AE** | 0.59 | **0.81** | **0.79** | **0.88** | **0.83** | 0.57 | **0.82** | **0.89** | **0.83** | **0.83** |
| **DeepHit** | 0.51 | 0.59 | 0.55 | 0.78 | 0.61 | 0.52 | 0.60 | 0.57 | 0.71 | 0.54 |
| **DeepSurv** | 0.48 | 0.52 | 0.49 | 0.47 | 0.52 | 0.48 | 0.49 | 0.50 | 0.48 | 0.51 |
| | **UNOS-OPTN(KIDPAN) dataset** | | | | | | | | | |
| | **Kidney graft (KGRFT) Failure** | | | | | **Pancreas graft (PGRFT) Failure** | | | | |
| | T1 | T2 | T3 | T4 | | T1 | T2 | T3 | T4 | |
| **CRESA** | 0.64 | 0.75 | 0.86 | 0.81 | | 0.60 | 0.77 | 0.81 | 0.80 | |
| **FINE-GRAY** | 0.59 | 0.51 | 0.49 | 0.50 | | **0.68** | 0.56 | 0.44 | 0.50 | |
| **RSF** | 0.51 | 0.44 | 0.48 | 0.52 | | 0.49 | 0.55 | 0.47 | 0.43 | |
| **CmpXRnnSurv** | 0.62 | 0.78 | 0.85 | 0.89 | | 0.59 | 0.81 | 0.79 | 0.79 | |
| **CmpXRnnSurv_AE** | 0.65 | **0.81** | **0.90** | **0.92** | | 0.61 | **0.88** | **0.85** | **0.86** | |
| **DeepHit** | **0.69** | 0.54 | 0.61 | 0.58 | | 0.52 | 0.66 | 0.50 | 0.59 | |
| **DeepSurv** | **0.54** | 0.51 | 0.50 | 0.52 | | 0.50 | 0.52 | 0.49 | 0.53 | |

Table 10.4. The weighted time dependent Brier scores

| MIMIC-III Clinical dataset | | | | | |
|---|---|---|---|---|---|
| | Time stamps | | | | |
| **Model** | **T1** | **T2** | **T3** | **T4** | **T5** |
| CRESA | 0.5614 | 0.5041 | 0.5100 | 0.4995 | 0.5120 |
| FINE-GRAY | 0.5160 | 0.6983 | 0.7419 | 0.6857 | 0.7099 |
| RSF | 0.7831 | 0.8429 | **0.8690** | 0.8152 | 0.8355 |
| CmpXRnnSurv | 0.5316 | 0.5107 | 0.4998 | 0.5071 | 0.4914 |
| **CmpXRnnSurv_AE** | 0.4920 | **0.3197** | 0.3918 | 0.4081 | 0.3821 |
| DeepHit | 0.5964 | 0.7655 | 0.7155 | 0.7033 | 0.7813 |
| DeepSurv | 0.5941 | 0.6512 | 0.7005 | 0.6692 | 0.7109 |
| **Synthetic dataset** | | | | | |
| | T1 | T2 | T3 | T4 | T5 |
| CRESA | 0.7240 | 0.6714 | 0.7105 | 0.6680 | 0.7005 |
| FINE-GRAY | 0.7819 | 0.8106 | 0.7918 | 0.8144 | 0.8396 |
| RSF | **0.9330** | 0.8731 | 0.9188 | 0.9075 | 0.9241 |
| CmpXRnnSurv | 0.7341 | 0.6582 | 0.6673 | 0.6930 | 0.6855 |
| **CmpXRnnSurv_AE** | 0.6011 | 0.4921 | 0.4857 | 0.5009 | **0.4794** |
| DeepHit | 0.7921 | 0.8029 | 0.8009 | 0.7990 | 0.8105 |
| DeepSurv | 0.8011 | 0.7816 | 0.8192 | 0.8300 | 0.7935 |
| **UNOS-OPTN (KIDPAN) dataset** | | | | | |
| | T1 | T2 | T3 | T4 | |
| CRESA | 0.5102 | 0.4831 | 0.5001 | | 0.4463 |
| FINE-GRAY | 0.7835 | 0.8016 | 0.7355 | | 0.7681 |
| RSF | 0.8213 | 0.7741 | **0.8339** | | 0.7140 |
| CmpXRnnSurv | 0.5210 | 0.4829 | 0.5080 | | 0.4311 |
| **CmpXRnnSurv_AE** | 0.4277 | **0.2036** | 0.2981 | | 0.3095 |
| DeepHit | 0.6284 | 0.7051 | 0.6925 | | 0.7360 |
| DeepSurv | 0.6144 | 0.6519 | 0.7102 | | 0.6796 |

# 11. CONCLUSION AND FUTURE OUTLOOK

We developed a novel deep learning model termed CmpXRnnSurv-AE for the complex survival problem. To the best of the author's knowledge, this is the second work on a deep-learning approach to recurrent events with competing risks. Using the external autoencoder, we addressed the noise invariant problem in the complex survival data.

By computing the weighted cumulative incidence function (WCIF), we managed the complex correlation structure in recurrent events with competing risk survival data. Since deep learning models are overparameterized and deterministic, the point estimates provided are unreliable for the predictions' uncertainty measurements.

In addition, the RNN is subjective to temporal inconsistency. Temporal inconsistency in RNNs refers to the problem that the internal state of the RNN may not accurately reflect the information present in the input sequence up to the current time step, due to the fact that the state is updated as the sequence is processed. This can make it difficult for the RNN to accurately process long sequences or sequences with complex temporal dependencies.

Historical Consistent Neural Networks (HCNN) is a type of neural network architecture that addresses the problem of temporal inconsistency in RNNs. In HCNN, the internal state of the network is updated only at specific time steps, determined by the input, rather than at every time step. This allows the network to maintain a consistent internal state that accurately reflects the information present in the input sequence up to the current time step, making it better suited to process long sequences or sequences with complex temporal dependencies (Rockefeller *et al.,* 2022).

Bayesian approach is widely used in Machine Learning, especially in probabilistic models and in modeling uncertain and complex systems. The Bayesian framework is a mathematical framework for reasoning under uncertainty. It provides a way to reason about the probability of different hypotheses given some observed data. In the Bayesian framework, all unknown quantities

(parameters and/or hypotheses) are treated as random variables, and probability distributions are assigned to them.

The key idea of Bayesian inference is to update the probability distribution of the unknown quantities, given the observed data, using Bayes' theorem. This process is called Bayesian updating or Bayesian inference. In the Bayesian framework, the prior knowledge is encoded into prior probability distributions, which combined with the likelihood of the data, lead to posterior probability distributions. The posterior probability distributions are then used to make predictions or inferences about the unknown quantities (He *et al.,* 2023).

In future work, we propose to apply the Historical Consistent Neural Networks (HCNN) in the Bayesian framework to quantify uncertainties in the model's predictions for the case of complex survival problems.

# REFERENCES


Alaa, A.M., van der Schaar, M., Deep multi-task Gaussian processes for survival analysis with competing risks, Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS 2017), (**2017**) 2326–2334.

Bahdanau, D., and Cho, K., Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv: 1409.0473, **2014**.

Bank, D., Koenigstein, N. and Giryes, R., Autoencoders, arXiv preprint\newline arXiv:2003.05991, **2020**.

Bender, R., Augustin, T., and Blettner, M., Generating survival times to simulate Cox proportional hazards models. Statistics in Medicine, 24(11) **(2005)** 1713-1723.

Bergstra, J., & Bengio, Y., Random search for hyper-parameter optimization. Journal of Machine Learning Research, 13(2) (**2012) 281-305**.

Breiman, L. Random forests, Machine learning, 45(1) **(2001)** 5-32.

Brownlee, J., Data Preparation, Machine Learning Mastery (2020, June 30), https://machinelearningmastery.com/what-is-data-preparation-in-machine-learning, (Last access: **August 18, 2022**)

Brutzkus, Alon, and Amir Globerson. Why do larger models generalize better? A theoretical perspective via the XOR problem, International Conference on Machine Learning PMLR 97 **(2019)** 822-830.

Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., and Varoquaux, G. API design for machine learning software: experiences from the scikit-learn project, arXiv preprint arXiv:1309.0238, **2013.**

Cho, K., Van Merriënboer, B., Bahdanau, D. and Bengio, Y., On the properties of neural machine translation: Encoder-decoder approaches, arXiv preprint arXiv:1409.1259, **2014.**

Chorowski, J. K., Bahdanau, D., Serdyuk, D., Cho, K. and Bengio, Y. Attention-based models for speech recognition. Advances in Neural Information Processing Systems, 28, **2015**.

Coccia, M., Deep learning technology for improving cancer care in society: new directions in cancer imaging driven by artificial intelligence, Technol. Soc., 60 (**2020)** 1-11.

Cox, D. R. Regression models and life□tables. Journal of the Royal Statistical Society: Series B (Methodological), 34(2) **1972,** 187-202.

Cox, D.R., Partial likelihood, Biometrika, 62(2) **(1975)** 269-276.

Du, S., Li, T., Yang, Y. and Horng, S. J., Multivariate time series forecasting via attention-based encoder–decoder framework. Neurocomputing, 388 **(2020)** 269-279.

Erizmann, D., Long-Term Time Series Forecasting and Uncertainty Estimation with Bayesian Neural Networks, Master's thesis, The University of Bremen, **2021.**

Faraggi, D., Simon, R., A neural network model for survival data, Statistics in Medicine, 14(1) **(1995)** 73-82.

Fine, J. P., Gray, R. J., A proportional hazards model for the sub-distribution of a competing risk, J. Am. Stat. Assoc., 94(446) **(1999)** 496–509.

Gated recurrent unit - Wikipedia. (2016, May 18). Gated Recurrent Unit - Wikipedia. https://en.wikipedia.org/wiki/Gated_recurrent_unit, (Last access date: April 2, 2023).

Ghorbani, B., Krishnan, S. and Xiao, Y., An investigation into neural net optimization via hessian eigenvalue density, Proceedings of the 36th International Conference on Machine Learning PMLR, 97 (**2019**) 2232-2241.

Giunchiglia E., Nemchenko A., van der Schaar M., RNN-SURV: A Deep recurrent model for survival analysis. In: Kůrková V., Manolopoulos Y., Hammer B., Iliadis L., Maglogiannis I. (eds) Artificial Neural Networks and Machine Learning - ICANN 2018 - 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part III, volume 11141 of Lecture Notes in Computer Science, Springer (**2018**) 23–32.

Goodfellow, I., Bengio, Y. and Courville, A. Deep learning (Vol. 1). Cambridge, MA, USA: MIT Press, **2016**.

Gooley, T.A., Leisenring, W., Crowley, J., Storer, B.E., Estimation of failure probabilities in the presence of competing risks: new representations of old estimators, Statistics in Medicine, 18 **(1999)** 695-706.

Gray, R.J., A class of K-sample tests for comparing the cumulative incidence of a competing risk, Ann. Stat., 16 **(1988)** 1141–1154.

Gray, B., & Gray, M. B., Package 'cmprsk'. Subdistribution analysis of competing risks, R package version, 2 **(2014)** 2-7.

Gupta G., Sunder V., Prasad R. and Shroff G., CRESA: A Deep learning approach to competing risks, recurrent event survival analysis. In: Yang Q., Zhou ZH., Gong Z., Zhang ML., Huang SJ. (eds) Advances in Knowledge Discovery and Data Mining. PAKDD 2019. Lecture Notes in Computer Science, vol 11440. Springer, Cham. https://doi.org/10.1007/978-3-030-16145-3-9, **2019**.

Harrell, F. E., Califf, R. M., Pryor, D. B., Lee, K. L. and Rosati, R. A., Evaluating the yield of medical tests, Jama, 247(18) **(1982)** 2543-2546.

He, W., Li, Q., Ma, Y., Niu, Z., Pei, J. and Zhang, Y. Machine learning in nuclear physics at low and intermediate energies, *arXiv preprint arXiv:2301.06396*, **2023**.

Hu, Y., Yang, T., Zhang, J., Wang, X., Cui, X., Chen, N., ... & Zou, J., Dynamic prediction of mechanical thrombectomy outcome for acute ischemic stroke patients using machine learning. Brain Sciences, 12(7) **(2022)** 938.

Huang, S., Wang, D., Wu, X., and Tang, A., DSANet: Dual self-attention network for multivariate time series forecasting, Proceedings of the 28th ACM international conference on information and knowledge management, **2019**, 2129-2132.

Ishwaran, H., Gerds, T. A., Kogalur, U. B., Moore, R. D., Gange, S. J. and Lau, B. M., Random survival forests for competing risks, Biostatistics, 15(4) **(2014)** 757-773.

Ishwaran, H., Kogalur, U. B., & Kogalur, M. U. B. (2023). Package 'randomForestSRC'. breast, 6(1).

Ioffe, S., Szegedy, C., Batch normalization: Accelerating deep network training by reducing internal covariate shift, International Conference on Machine Learning PMLR 97, **2015**, 448-456.

Johnson, A.E., Pollard, T.J., Shen, L., Lehman, L.W., Feng, M., Ghassemi, M., Moody, B., Szolovits, P., Celi, L.A., Mark, R.G., MIMIC-III, a freely accessible critical care database. Sci Data 3, 160035 **(2016).** https://doi.org/10.1038/sdata.2016.35.

Jonnalagadda, V. K. (2018, December 6). Sparse, Stacked and Variational Autoencoder. Medium.https://medium.com/@venkatakrishna.jonnalagadda/sparse-stacked-and-variational-autoencoder-efe5bfe73b64. (Last access date: August 5, 2023).

Kaplan, E.L., Meier, P., Nonparametric Estimation from Incomplete Observations. Journal of the American Statistical Association, 53 **(1958)** 457-481.

Katzman, J. L., Shaham, U., Cloninger, A., Bates, J., Jiang, T. and Kluger, Y., DeepSurv: personalized treatment recommender system using a Cox proportional hazards deep neural network, BMC Medical Research Methodology, 18(1) **(2018)** 1-12.

Kernbach, J.M., Staartjes, V.E., Foundations of Machine Learning-Based Clinical Prediction Modeling: Part II—Generalization and Overfitting. In: Staartjes, V.E., Regli, L., Serra, C. (eds) Machine Learning in Clinical Neuroscience, Acta Neurochirurgica Supplement, vol 134. Springer, Cham, **2022**.

Kingma, D. P., Ba, J., Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980,* **2014**.

Lee, C., Zame, W. R., Yoon, J., van der Schaar, M., DeepHit: a deep learning approach to survival analysis with competing risks. In 32nd Association for the Advancement of Artificial Intelligence (AAAI) Conference, **2018**.

Lee, C., Yoon, J., Schaar, M. v.d., Dynamic-DeepHit: A Deep learning approach for dynamic survival analysis with competing risks based on longitudinal data, IEEE Transactions on Biomedical Engineering, 67 (1) **(2020)** 122-133.

Lee, K. J., & Carlin, J. B., Recovery of information from multiple imputation: a simulation study. Emerging Themes in Epidemiology, 9 **(2012)** 1-10.

Li, J., Hu, X., Wu, L. and Liu, H., Robust unsupervised feature selection on networked data, Proceedings of the 2016 SIAM International Conference on Data Mining. Society for Industrial and Applied Mathematics, **2016**.

Li, Y., Wang, J., Ye, J., Reddy, C.K., A multi-task learning formulation for survival analysis, Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ser. KDD '16. ACM, **2016**, 1715–1724. [Online]. Available: http://doi.acm.org/10.1145/2939672.2939857.

Li, I., Pan, J., Goldwasser, J., Verma, N., Wong, W. P., Nuzumlalı, M. Y. and Radev, D., Neural natural language processing for unstructured data in electronic health records: A review, Computer Science Review 46 100511 **(2022)**.

Liao, L., Ahn, H., Combining deep learning and survival analysis for asset health management, International Journal of Prognostics and Health Management, Special Issue Big Data and Analytics, 7 **(2016)** 1-7.

Lin H, Yan J, Yuan L, Qi B, Zhang Z, Zhang W, Ma A, Ding F., Impact of diabetes mellitus developing after kidney transplantation on patient mortality and graft survival: a meta-analysis of adjusted data, Diabetol Metab Syndr, 13(1) **2021** 126.

Lydia, A., Francis, S. Adagrad—an optimizer for stochastic gradient descent, Int. J. Inf. Comput. Sci, *6*(5) **(2019)** 566-568.

Long short-term memory - Wikipedia. (2022, March 1). Long Short-term Memory - Wikipedia. https://en.wikipedia.org/wiki/Long_short-term_memory. (Last access time: April 12, 2022).

Longato, E., Vettoretti, M. and Di Camillo, B. A practical perspective on the concordance index for the evaluation and selection of prognostic time-to-event models, Journal of Biomedical Informatics 108 103496 **(2020)**.

Luck, M., Sylvain, T., Cardinal, H., Lodi, A. and Bengio, Y., Deep learning for patient-specific kidney graft survival analysis, arXiv preprint arXiv:1705.10245, **2017**.

Luong, M. T., Pham, H., & Manning, C. D., Effective approaches to attention-based neural machine translation. arXiv preprint arXiv:1508.04025, **2015**.

Marthin, P., & Ata Tutkun, N., Recurrent neural network for complex survival problems, Journal of Statistical Computation and Simulation, DOI: 10.1080/00949655.2023.2176504, **2023** (Published online: 15 Feb 2023).

Mnih, Volodymyr, Nicolas Heess, and Alex Graves. Recurrent models of visual attention, Advances in Neural Information Processing Systems, 27 **(2014).**

Manaswi N.K., RNN and LSTM. In: Deep Learning with Applications Using Python. Apress, Berkeley, CA, **2018**.

Molnar, C., Casalicchio, G. and Bischl, B. (2021, February). Interpretable machine learning–a brief history, state-of-the-art and challenges. In ECML PKDD 2020 Workshops: Workshops of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD 2020): SoGood 2020, PDFL 2020, MLCS 2020, NFMCP 2020, DINA 2020, EDML 2020, XKDD 2020 and INRA 2020, Ghent, Belgium, September 14–18, **2020**, Proceedings (pp. 417-431). Cham: Springer International Publishing.

Nielsen, M.A., Neural Networks and Deep Learning. Vol. 25. San Francisco, CA, USA: Determination Press, **2015**.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G. and Chintala, S., PyTorch: An Imperative Style, High-Performance Deep Learning Library, Advances in Neural Information Processing Systems 32 **(2019)** 8024–8035. Curran Associates, Inc. Retrieved from http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Pénichoux, J., Moreau, T. and Latouche, A., Simulating recurrent events that mimic actual data: a review of the literature with emphasis on event-dependence, arXiv preprint arXiv:1503.05798, **2015**.

Prentice, R., Kalbfleisch, J., Peterson, A., Flournoy, N., Farewell, V. and Breslow, N., The analysis of failure times in the presence of competing risks, Biometrics 34(4) **(1978)** 541-554.

Prentice, R. L., Vollmer, W. M., & Kalbfleisch, J. D., On the use of case series to identify disease risk factors, Biometrics, **(1984)** 445-458.

Rathod, J. (n.d.). Underfitting, Overfitting, and Regularization. Jash Rathod. https://jashrathod.github.io/2021-09-30-underfitting-overfitting-and-regularization.

Ravanelli, M., Brakel, P., Omologo, M. and Bengio, Y., Light Gated Recurrent Units for Speech Recognition, in IEEE Transactions on Emerging Topics in Computational Intelligence 2(2) **(2018)** 92-102.

Rietschel, C., Yoon, J. and van der Schaar, M., Feature selection for survival analysis with competing risks using deep learning, arXiv preprint arXiv:1811.09317, **2018**.

Rockefeller, R., Bah, B., Marivate, V. and Zimmermann, H. G. Improving the predictive power of historical consistent neural networks, Engineering Proceedings 18(1) 36 **(2022).**

Rumelhart, D. E., Hinton, G. E. and Williams, R. J., Learning representations by back-propagating errors, Nature, 323(6088) **(1986)** 533-536.

Shang, Y., & Wah, B. W., Global optimization for neural network training, Computer, 29(3) **(1996)** 45-54.

Sinha, A., Dolz, J., Multi-scale self-guided attention for medical image segmentation, IEEE Journal of Biomedical and Health Informatics, 25(1) **(2020)** 121-130.

Singer, J. D., Willett, J. B., Survival Analysis, in J. A. Schinka & W. F. Velicer (Eds.), Handbook of psychology: Research Methods in Psychology, 2 **(2003)** 555–580.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R., Dropout: a simple way to prevent neural networks from overfitting, The Journal of Machine Learning Research, 15(1) **(2014)** 1929-1958.

Talend, Data Integrity and Governance, (2022, April 17), https://www.talend.com/resources/eliminating-barriers-to-data-excellence/, (Last Access: August 18, 2022)

Tieleman, T., Hinton, G., Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude, COURSERA: Neural networks for machine learning, 4(2) **(2012)** 26-31.

Tutkun Ata, N., Marthin, P. A comparative study with bootstrap resampling technique to uncover behavior of unconditional hazards and survival functions for gamma and inverse Gaussian frailty models, Math Sci 15 **(2021)** 99–109.

UNOS-KIDPAN dataset 2022, Organ Procurement and Transplantation Network (OPTN)

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., and Polosukhin, I. Attention is all you need. Advances in neural information processing systems, 30, 2017.

Wang, S., Zhengming Ding, and Yun Fu, Feature selection guided auto-encoder, Proceedings of the AAAI Conference on Artificial Intelligence, 31(1) **2017**.

Wang, P., Li, Y., Reddy, C.K., Machine learning for survival analysis: A survey, ACM Comput. Surv., 51(6), Article no: 110 **(February 2019)**, 36 pages.

Wang, P., Shi, T., Reddy, C.K., Tensor-based temporal multi-task survival analysis, IEEE Transactions on Knowledge and Data Engineering, doi: 10.1109/TKDE.2020.2967700, **2020**.

Weigel, A. P., Liniger, M. A., and Appenzeller, C., The discrete Brier and ranked probability skill scores, Monthly Weather Review, 135(1) **(2007)** 118-124.

Xiao, X., Yan, M., Basodi, S., Ji, C., and Pan, Y., Efficient hyperparameter optimization in deep learning using a variable length genetic algorithm. *arXiv preprint arXiv:2006.12703,* **2020**.

Yao, J., Zhu, X., Zhu, F., & Huang, J., Deep correlational learning for survival prediction from multi-modality data, International conference on medical image computing and computer-assisted intervention (pp. 406-414). Springer, Cham, **2017**.

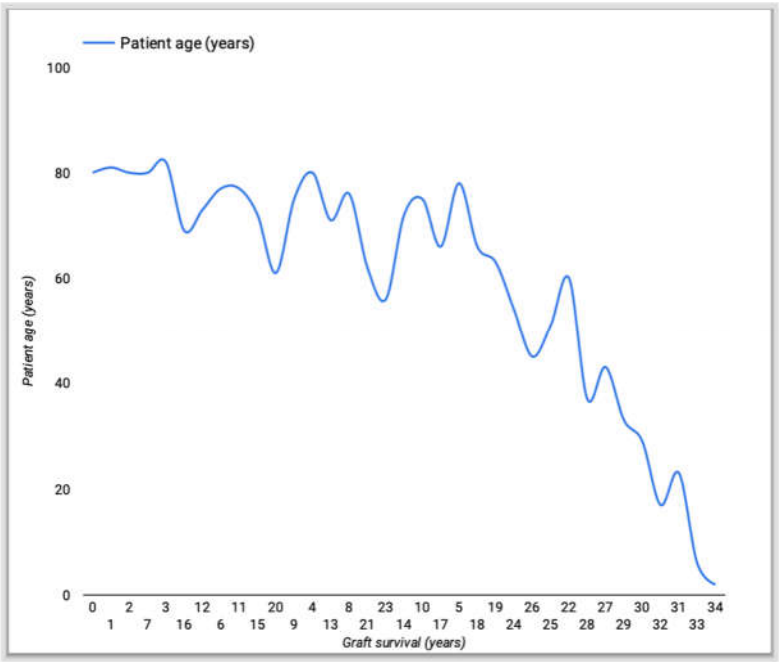
Yessou, H., Analysis of Deep Learning Loss Functions for Multi-Label Remote Sensing Image Classification, Master's thesis, The Polytechnic University of Milan, **2020**.

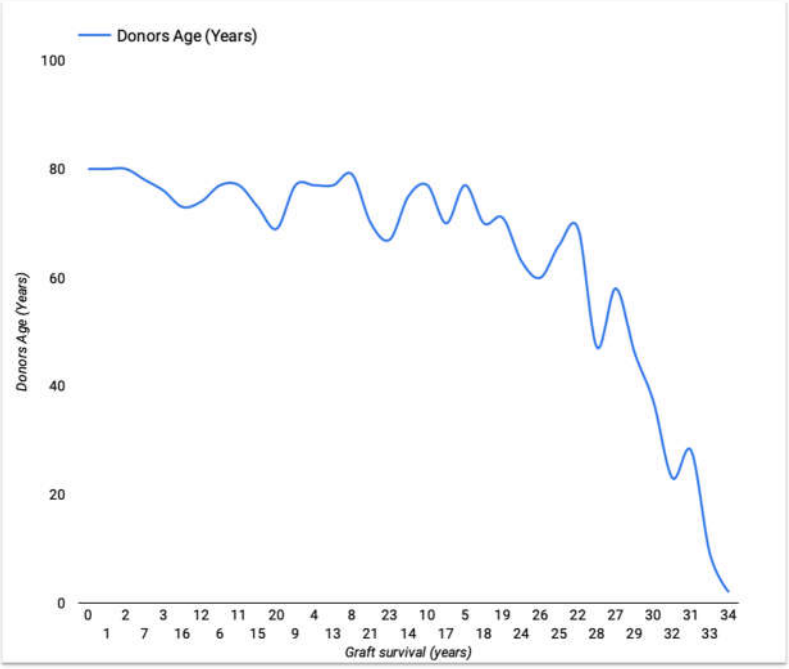
Zhu, X., Yao, J., Huang, J., Deep convolutional neural network for survival analysis with pathological images, 2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), Shenzhen **(2016)** 544-547, doi: 10.1109/BIBM.2016.7822579.

# APPENDICES

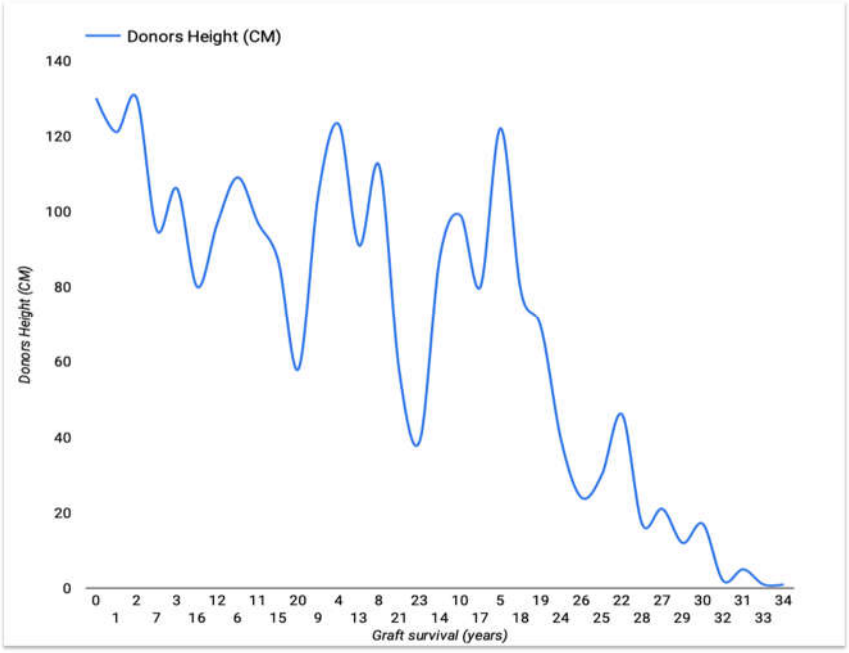
## APPENDIX 1 - Descriptions of the data concerning some covariates-survival relationships

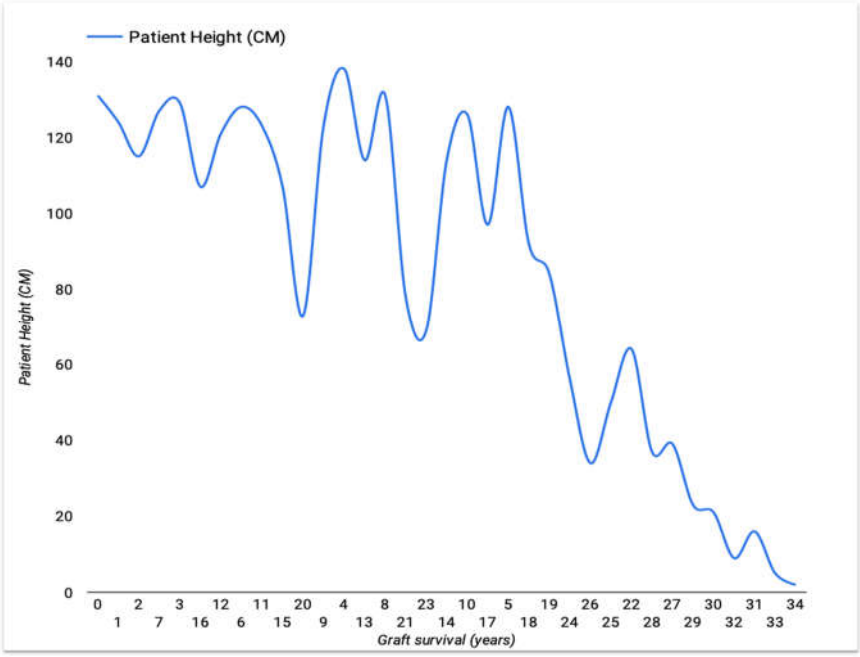
Appendix 1A: Graft survival based on patient's-donors age



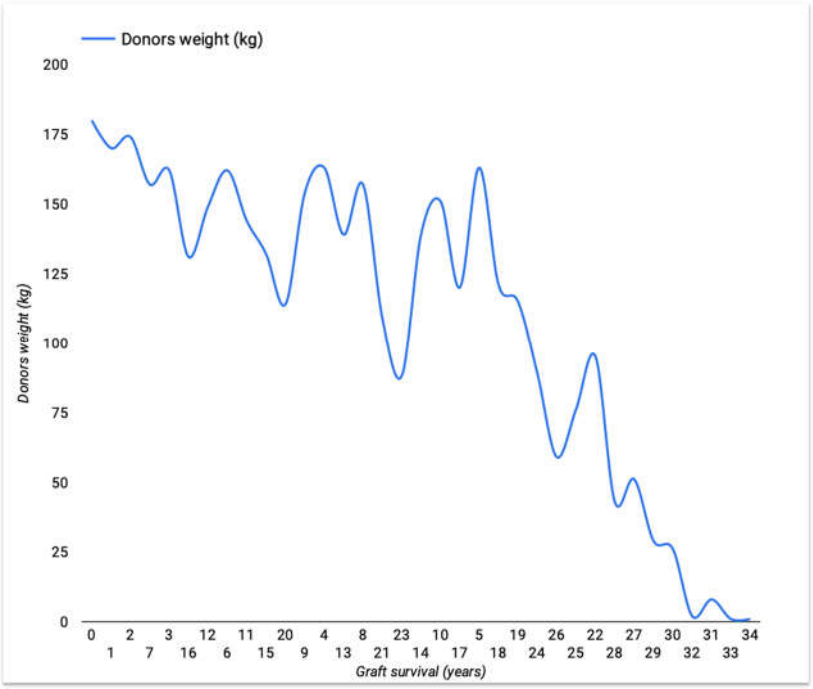



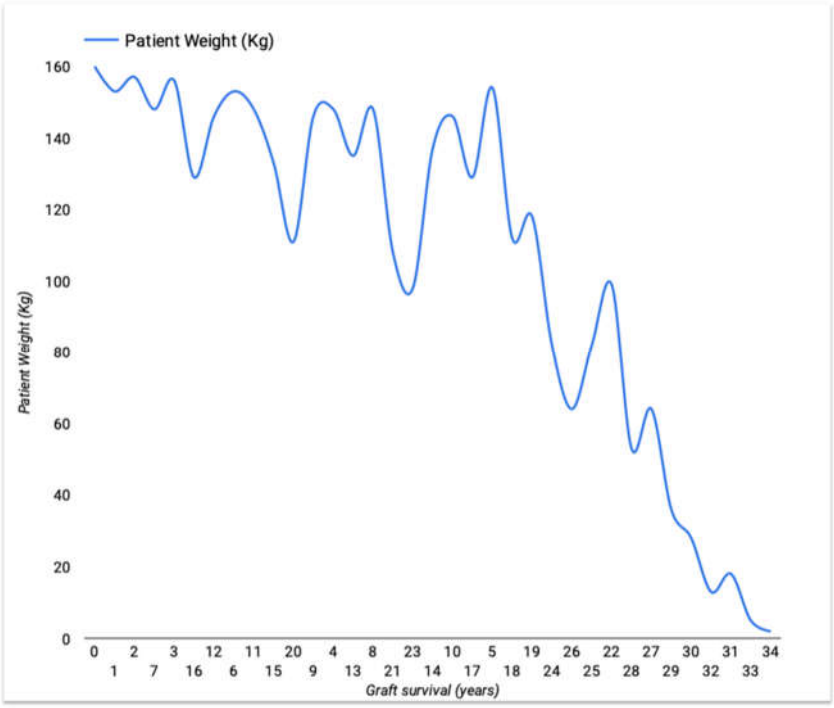
Appendix 1B. Graft survival based on patient's-donors height



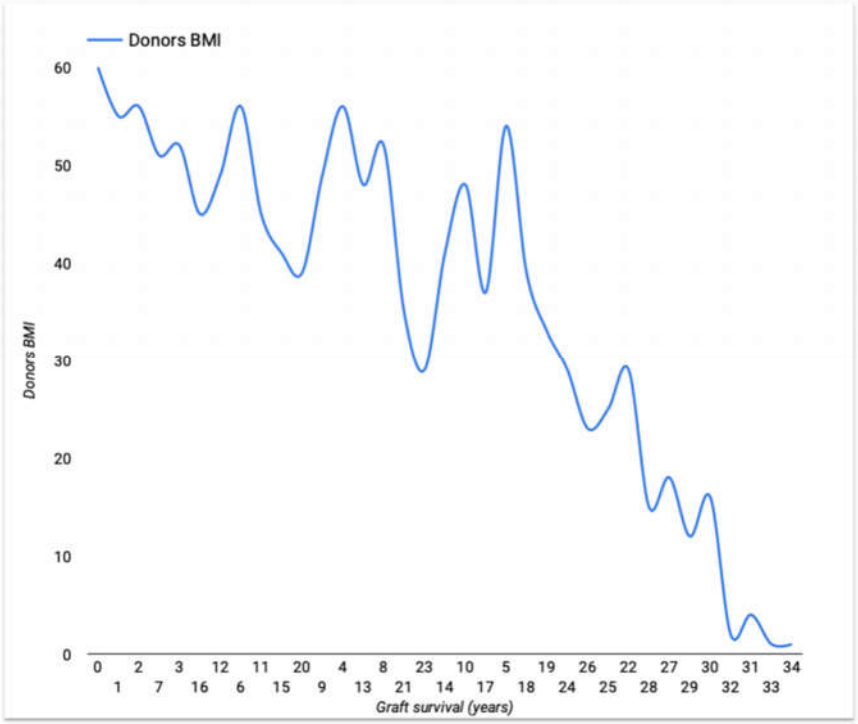



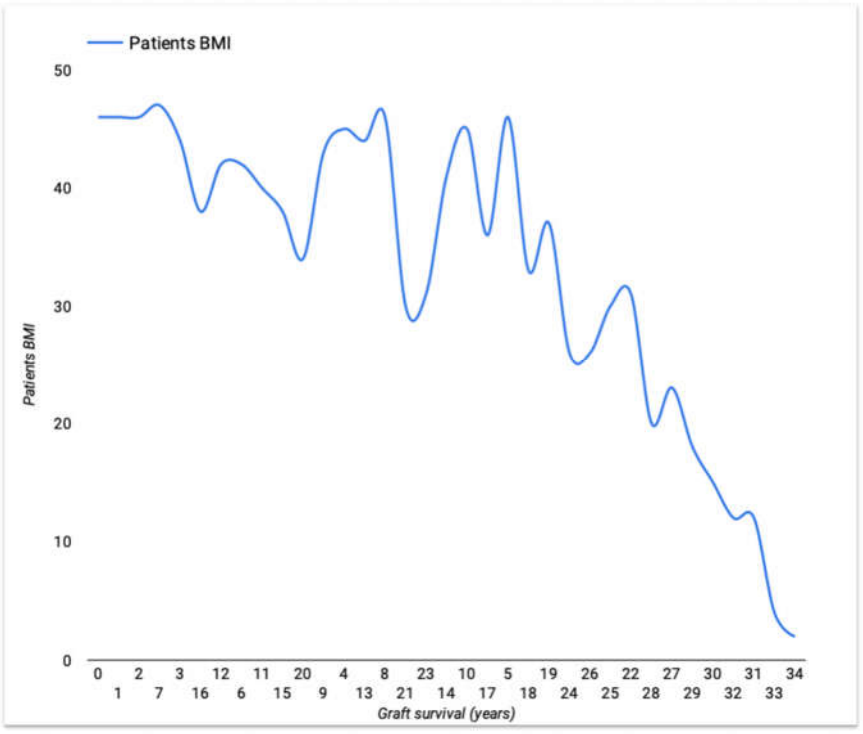
Appendix 1C. Graft survival based on patient's-donors weight



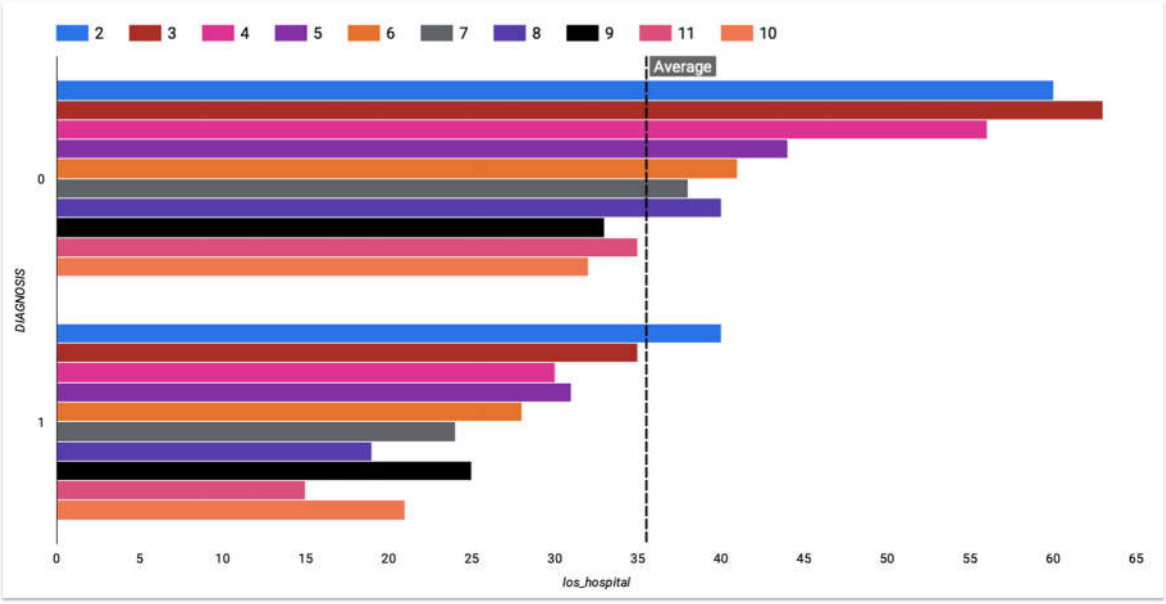



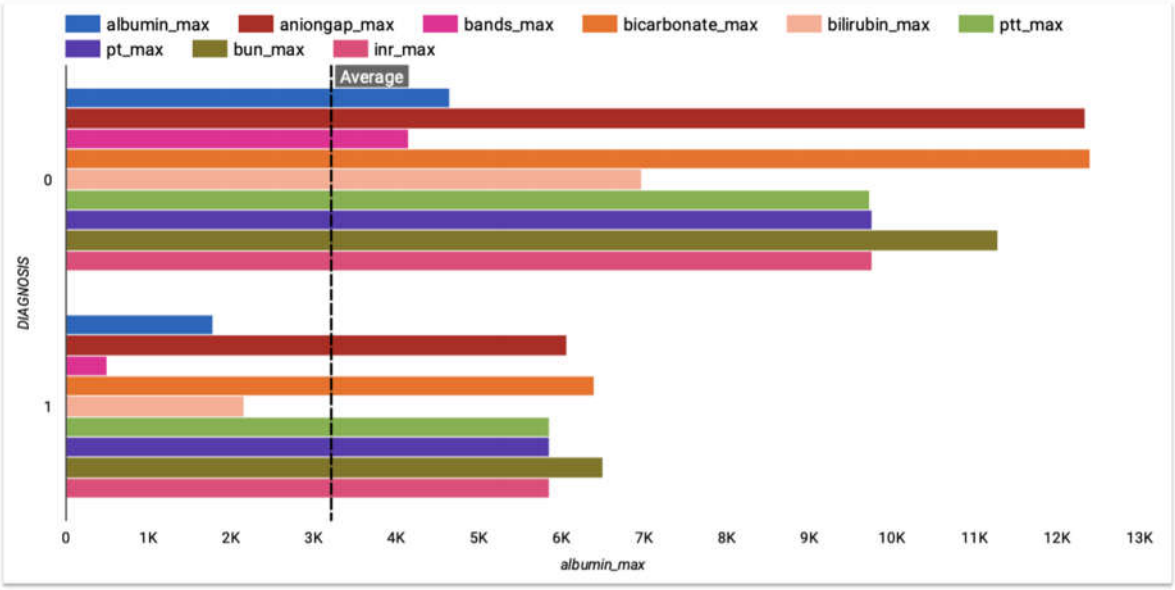
Appendix 1D. Graft survival based on patient's-donors BMI



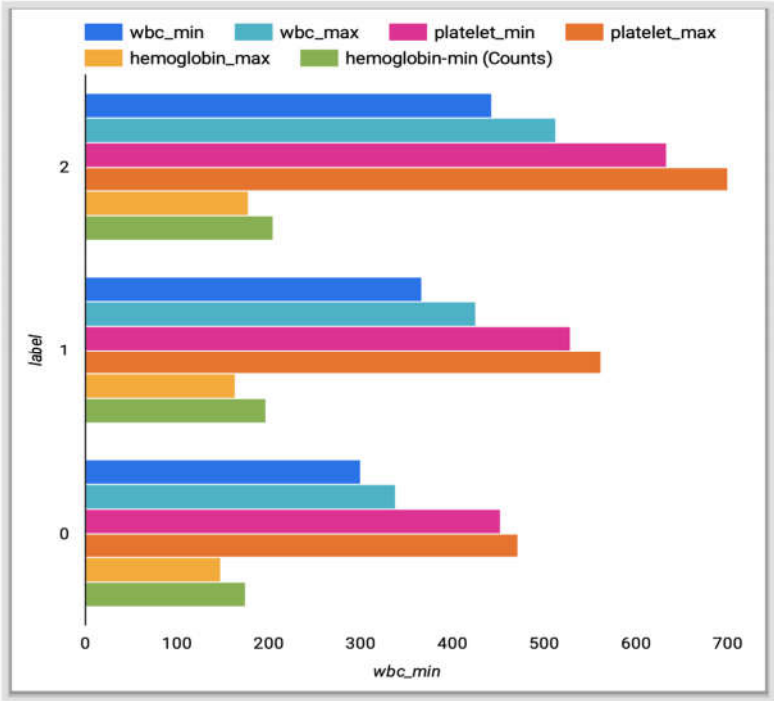



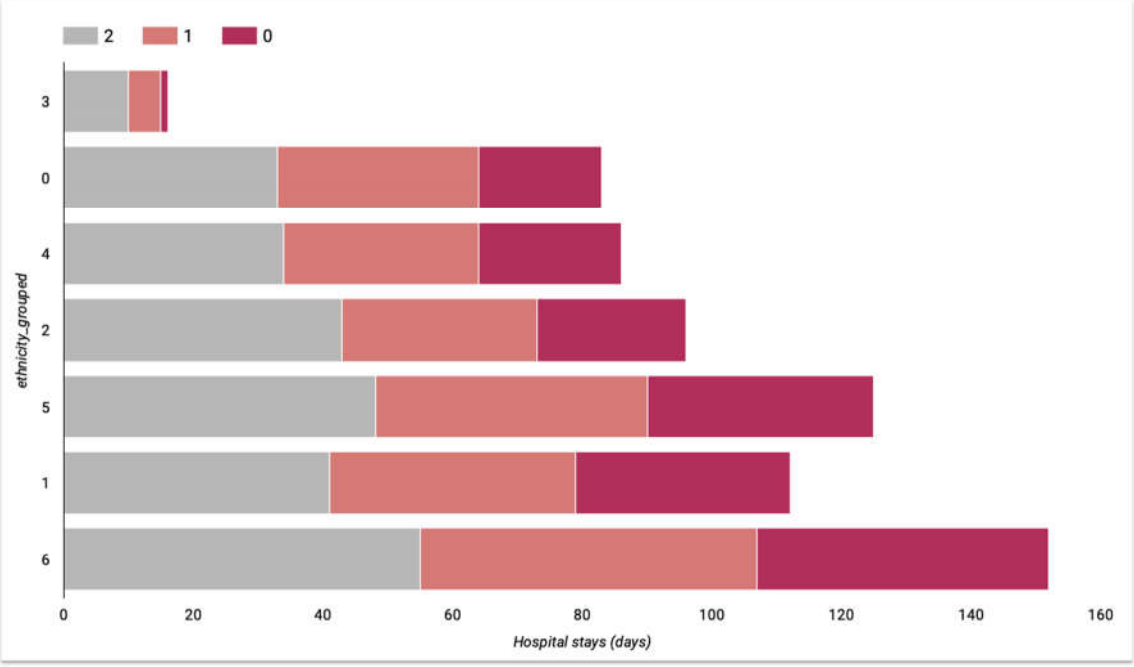
Appendix 1E. Patient distribution based on event's status, LOS, ICU-stay's duration and laboratory measurements



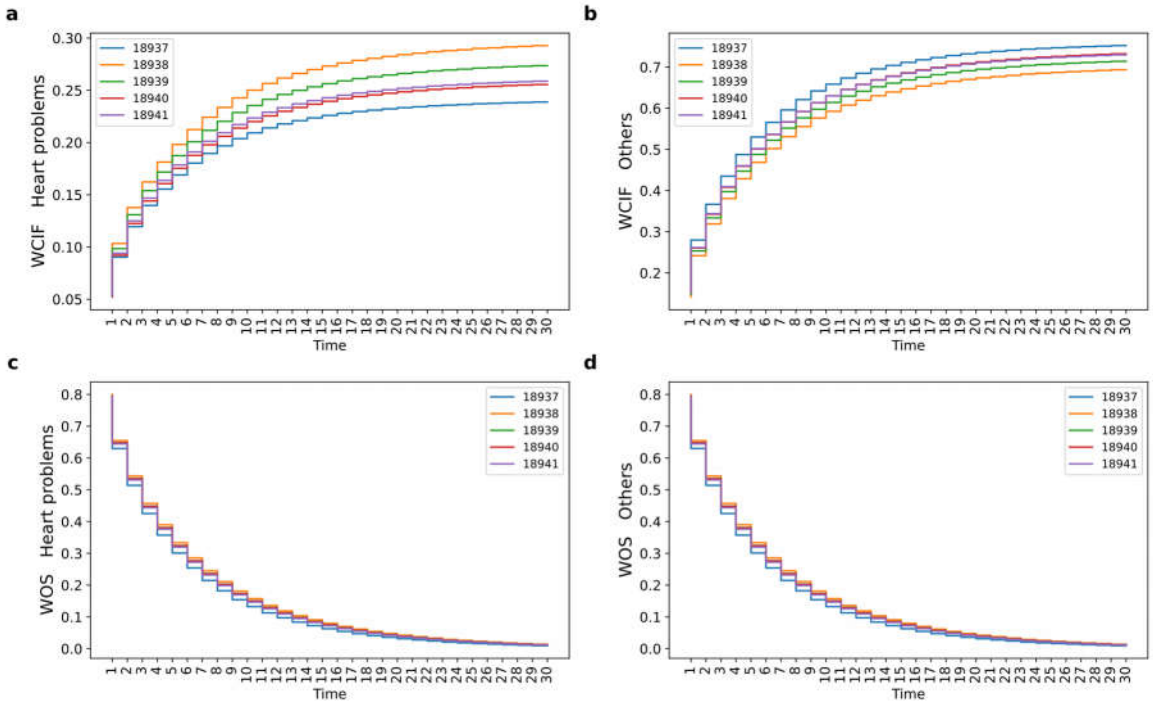



Appendix 1F. Patients distribution based on ethnicity, LOS, laboratory results and event's status

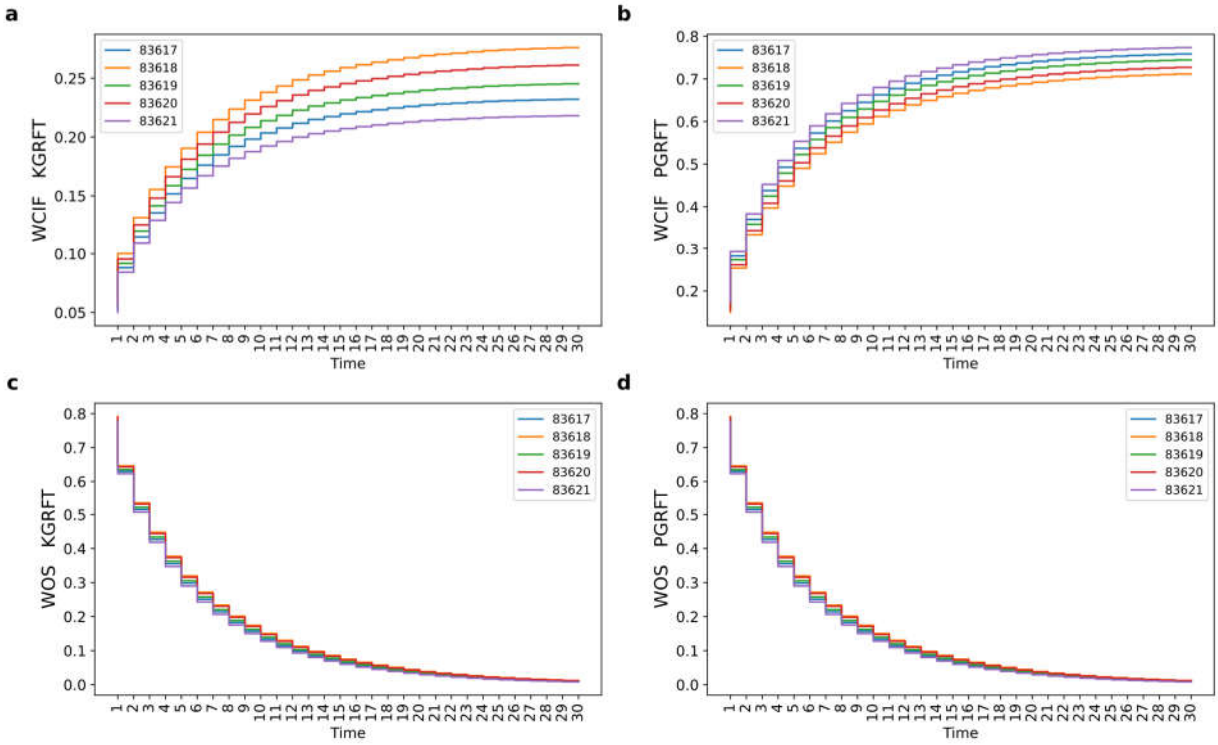


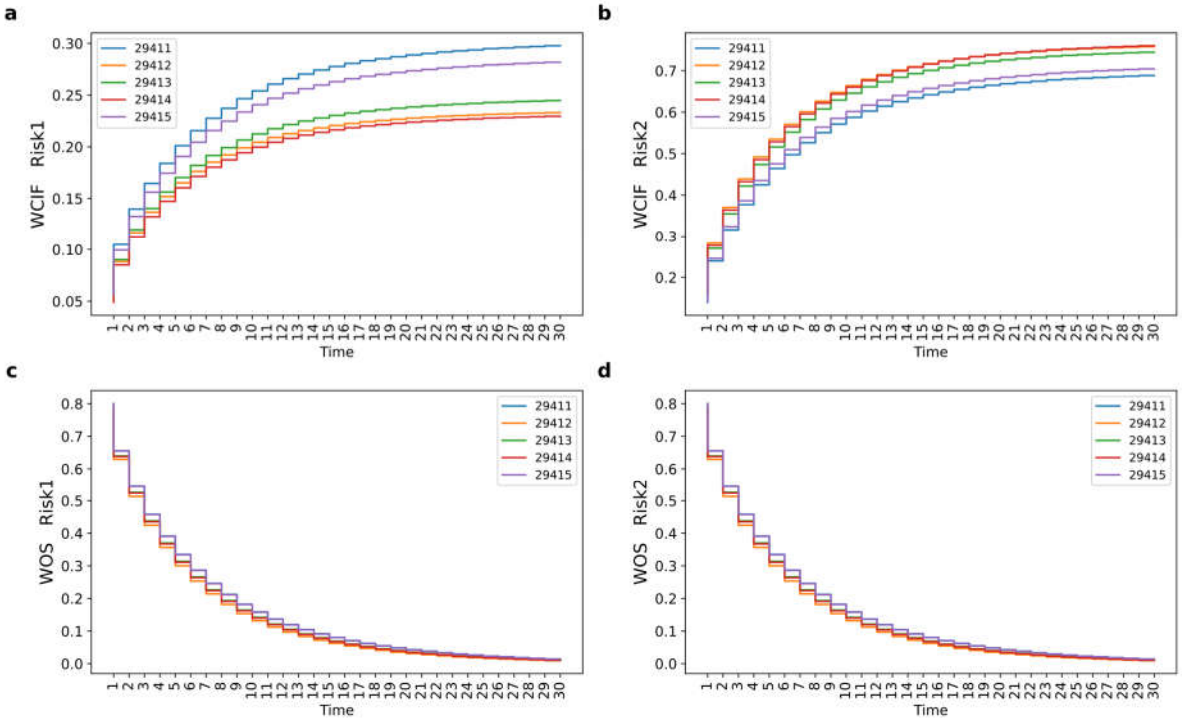
Appendix 1G. WCIF and WOS predictions on the MIMIC-III dataset for the selected subjects.



Appendix 1H. WCIF and WOS predictions on the UNOS-OPTN(KIDPAN) dataset for the selected subjects.



Appendix 1I. WCIF and WOS predictions on the synthetic dataset for the selected subjects



## APPENDIX 2 – Publications Produced From The Thesis

- **Pius Marthin**, N. Ata Tutkun (2023) Recurrent neural network for complex survival problems, Journal of Statistical Computation and Simulation, DOI: 10.1080/00949655.2023.2176504 (Online published: February 15, 2023)

- **Marthin, P**., Ata Tutkun, N., "Recurrent neural network for complex survival problems", 23th National and 6th International Biostatistics Conference, Ankara/Türkiye, 26-29 October 2022. (Oral Presentation)