PROGNOSTICS IN MANUFACTURING SYSTEMS USING DEEPSURV

A Thesis

by

NAGA VENKATA SAIDILEEP KORLAPATI

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2020

Major Subject: Industrial Engineering

ABSTRACT

Prognostics in manufacturing systems help forecast the future state using the data logs from the integrated sensors. Prognostics can be especially helpful in the current plant floor automation scenario where IoT devices are being utilized to the best possible extent. These devices generate large volumes of data that could provide value to the business when used in conjunction with analytic tools and allow businesses to take preventive measures before a breakdown could halt the plant machinery. Survival Analysis could be useful to learn the trend and forecast the event propagation using the data logs from these devices. This thesis attempts to apply modern machine learning techniques to learn the fault propagation trend from historical timestamps of the data and flag any impending breakdowns to take preventive action. DeepSurv, a python module intended for the treatment recommender setting in survival analysis, is used as a prognostic model for the manufacturing plant use case. The method is applied to the data collected from a 25-machine manufacturing plant floor. The results show that the model can forecast the future state of a machine using the historical patterns of the data logs with a Brier score of less than 0.15.

DEDICATION

To my parents,

Korlapati Narasimha Murthy and Korlapati Usha Sri,

and my extended family.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor Dr. Satish T. Bukkapatnam for the continuous support of my research, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis.

Besides my advisor, I would like to thank the rest of my thesis committee: Dr. Edward R Jones, and Dr. Hrayer Aprahamian, for their insightful comments.

My sincere thanks also go to Dr. Bharani Nagarathnam, and Dr. Bimal Nepal, for offering me the graduate assistantship.

Thanks also to my friends and colleagues and the department faculty and staff, making my time at Texas A&M University a great experience.

Finally, thanks to my family, near and far, for their unconditional love and support that got me through this work.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a thesis committee consisting of Professor Satish T. Bukkapatnam and Professor Hrayer Aprahamian of the Department of Industrial Engineering and Professor Edward R Jones of the Department of Statistics.

The data analyzed for Section 4 was provided by the research group of Professor Satish T. Bukkapatnam, with Ms. Afrin Kahkashan and Mr. Darpit Dave.

All other work conducted for the thesis was completed by the student independently.

Funding Sources

Graduate study was partly supported by a graduate assistantship from the Department of Engineering Technology and Industrial Distribution, and I would like to acknowledge the financial support for the research provided by the National Science Foundation.

NOMENCLATURE

PHM	Prognostics and Health Management
IoT	Internet of Things
CPH	Cox Proportional Hazards
PDF	Probability Density Function
CDF	Cumulative Distribution Function
KM	Kaplan-Meier
MLP	Multilayer Perceptron
RNN	Recurrent Neural Network
CNN	Convolutional Neural Network
GAN	Generative Adversarial Network
DFFN	Deep Feed Forward Neural Network
LM	Levenberg-Marquardt
SGD	Stochastic Gradient Descent
MBRSF	Manufacturing System-wide Balanced Random Survival Forest

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 Background

Mechanical components and machinery in a complex manufacturing system are subjected to degradation over time, which can lead to malfunctioning, and increased maintenance costs. One possible way for reliable operation is by monitoring the condition of the equipment, their breakdowns, and performing diagnostics and prognostics to layout the maintenance strategies. This process is often called prognostics and health management (PHM) [1].

There are two widely used categories of diagnostic and prognostic tools determined based on the process of monitoring data: data-driven, and model-based. The knowledge of monitoring, diagnostics, and prognostics is one of the fundamental areas to be familiar with, for its practical usage in continuous maintenance. With technological advancements like the Internet of Things (IoT) and the shift towards Industry 4.0, there has been a rapid increase in terms of the velocity, veracity, and volume of data generated from the sensors integrated with the machines. The availability of the data paves the way for a better understanding of machinery, mechanical components, and remaining useful life, thereby optimizing the operational impact of the machinery. This can also help in system-level monitoring rather than component-level monitoring, as well as in preventive maintenance with the proper usage of the model-based prognostic tools [2].

Nevertheless, significant challenges are associated with the model-based prognostic tools. One of which is the assumptions underlying the degradation of the mechanical component during the model development. One such model that has been used widely to determine the nature of the degradation process is Cox proportional hazards (CPH) [3]. This is often treated as the gold-standard for survival analysis. The recent advancements in the machine learning, especially the development of deep learning, relaxed the need to make some of these assumptions. Models developed from the CPH approach, however, suffer from the drawbacks of many other assumptions and are discussed in detail in Section 2.

Also, some critical technological challenges are associated with PHM and some of those pertinent to survival modeling for manufacturing systems are addressed below..

In the form of either historical time stamps or real-time, data act as the base for the entire PHM system. The raw data collected from sensors on harnessing and feeding into a model can help make actionable decisions on impending breakdowns in a manufacturing facility. However, the main limitation is addressing the uncertainties involved in the process. Even though model-based prognostic tools employ robust mathematical framework to track the degradation of a component, they cannot handle uncertainties that the modeling error, data quality, or randomness in future breakdowns causes [4]. Also, the models developed for component-level monitoring, and prognosis may not be ideal for generalization at the system-level.

In this thesis, we employ a deep learning based model, Deepsurv for breakdown prognostics.

1.2 Motivation and Objective

With machines getting more robust and the systems getting even more complicated by integrating new technology over time, the true potential of deep learning-based prognostic tools could come in handy for utilizing the vast amount of data and the resources available to harness the data. When deployed into production shop floors, the models can help make strategic plans for preventive maintenance by observing the current status of the machine or the component, thereby preventing the impending breakdowns. Notably, the use of such prognostic models can save about 50% in the form of maintenance costs [5].

This thesis gives a brief overview of Deepsurv that is developed from a CPH approach but harnesses the potential of deep learning. The underlying assumptions of the CPH model still hold for Deepsurv except that the latter model can embed the capability to model the non-linear distribution of the data.

Even though Deepsurv was developed for analysis in medical care, its true potential can be extended beyond the treatment recommender system. The main objective here is to provide an alternative form of survival analysis method that can employ deep learning, unlike the Random Survival Forest method, which has severe memory limitations. The aim of this work is presented below with the highlights of Deepsurv overcoming some of the shortcomings in the survival analysis field. The aims are to:

1. Make risk predictions that could help make decisions about the future status of the machine or the mechanical component and

2. Provide a viable alternative that could be production-ready to take preventive measures assessing the risk at the component-level.

1.3 Outline

The remainder of the thesis is organized as follows:

Section 2 presents a literature review of survival analysis and artificial neural networks. The first part of the section provides a brief overview of the statistical concepts of survival analysis, common survival models, and the estimation of baseline hazard. Artificial neural networks are introduced in the latter part, and various survival models developed using artificial neural networks are discussed.

Section 3 has an overview of Deepsurv, a deep learning-based model.

Section 4 presents the analysis of a case study. The data collected from integrated sensors in a manufacturing plant are analyzed using Deepsurv, and the results from the analysis are discussed in detail.

Section 5 summarizes the thesis and presents the limitations of the model and suggestions for future work.

2. LITERATURE REVIEW

2.1 A Brief Overview of Survival Analysis

The flexibility in defining an event in time-to-event analysis has given rise to the development of various statistical methods across different fields. The collection of methods to explain why a specific event of interest occurs defines survival analysis on a broader horizon. Survival Analysis in economics is often referred to as the duration analysis or hazard analysis or transition analysis, and in engineering, it is often called failure-time analysis or reliability analysis. The following sections define some basic notations in survival analysis to better understand the statistical concepts and review some early methods developed in this field to analyze time-to-event data. The later part of this section reviews some growing popular methods with the rapid advancements in machine learning, especially deep learning, to handle a large amount of data by effectively handling complexity.

In most cases of survival analysis applications, the variable of interest is the time at which a specified event of interest might happen in the future (Harrell Jr, 2018). Therefore, it is crucial to carefully define the survival time represented by a random variable T and censoring associated with the information on the start and endpoints of a study to model the underlying distribution effectively. The distribution of T, a non-negative random variable corresponding to the subject's survival time, is characterized by its probability density function (PDF) denoted by $f(t)$ and cumulative density function (CDF) denoted by $F(t)$. This characterization of T helps define some important terms, namely the survival function and hazard function, which helps determine the overall state of the sample/subject under consideration.

2.1.1 Survival function

Let T be a random variable representing the time to "failure" or the time to "loss" or the censored time. The survival function returns the probability of survival beyond time t and is expressed as

$$S(t) = Pr(T > t) = 1 - F(t) = \int_t^\infty f(u)du \tag{2.1}$$

2.1.2 Hazard function

The hazard function is defined as the ratio of the conditional probability of an event happening in the interval $[t, t + \delta t)$, given the event has not occurred until time t to that of the infinitesimal time frame δt. It usually describes the instantaneous rate of failure, rather than the probability of the event happening in the short interval of time. The hazard function is denoted by $\lambda(t)$ and is mathematically expressed as

$$\lambda(t) = \lim_{\delta t \to 0+} \frac{Pr(t \leq T < t + \delta t \mid t \leq T)}{\delta t} = \frac{f(t)}{S(t)} \tag{2.2}$$

In other words, the hazard function can be viewed as the ratio of the density of events at t to that of the survival probability at t.

From Equation 2.1, it can be seen that the derivative of $S(t)$ is $-f(t)$, and hence Equation 2.2 can be written as

$$\lambda(t) = -\frac{d}{dt} \log S(t) \tag{2.3}$$

On applying the boundary condition $S(0) = 1$, Equation 2.3 can then be written in the form

$$S(t) = \exp\,(-\int_0^t \lambda(u)du) = \exp(-\Lambda(t)) \tag{2.4}$$

$$\Lambda(t) = \int_0^t \lambda(u)du \tag{2.5}$$

where $\Lambda(t)$ in Equation 2.5 denotes the cumulative hazard function.

Thus, clearly a one-to-one correspondence exists between the functions, i.e., knowing one of either hazard function or the survival function, the other function can be obtained.

2.1.3 Censoring

Censoring is the most common consideration in Survival Analysis. Missing data in general statistical parlance is referred to as censoring in survival analysis. For instance, when a population is being observed for a particular event of interest, the observations are censored if the information of survival times is incomplete. Censoring can be classified into different categories based on the time at which the study begins and terminates, both of which make an integral part of survival time, as defined earlier. Censoring is explained by the censoring mechanisms and is broadly classified into two classes, namely,

1. Point censoring occurs when the exact time at which the event has happened is known. Point censoring can be further classified into two types, namely,

   (a) Right censoring: In a medical scenario, when individuals under study were lost to follow-up or when the individuals did not experience the event of interest at study termination, the individuals (or observations, commonly) are defined to be right-censored. This is the most common form of censoring observed, and almost all the methods (statistical or machine-learning) developed can handle right-censored data.

   For example, observations A and B in Figure 2.1 show an example of right-censored observations. Observation A experienced the event of interest after the study termination, and this phenomenon is called end-of-study censoring. On the other hand, observation B was lost-to-follow-up during the study, and this is often referred to as lost-to-follow-up censoring. In both cases, the observations are classified as right-censored but differ in the way the censored cases are handled during the analysis. In most cases of right-censored data, especially in the event of end-of-study censoring, there is no harm in an independent assumption for handling censored data.

   Independent assumption in censoring described by Leung et al. in [6] can be stated as, "time of entry to the study is independent of the risk period." This can also be stated as a risk at time t does not alter by having the information of censoring before any time

t. Leung et al.[6] also states that except for type I and type II censoring designs, the censoring mechanisms in most cases of right censoring are unknown, and as a result, some need exists to make certain assumptions before using the statistical methods to handle special cases like the lost-to-follow-up censoring data.

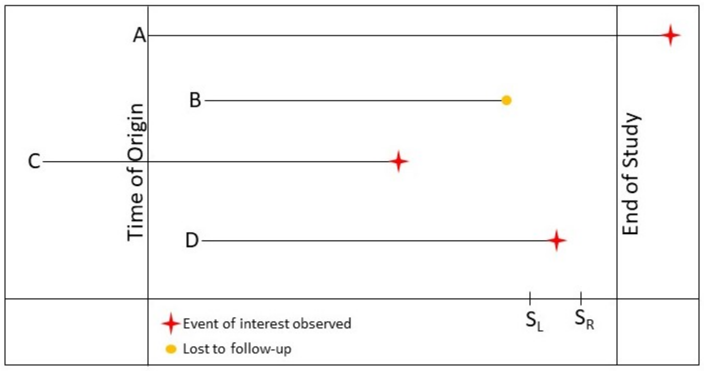


Figure 2.1: Illustration of right, left, and interval censoring

(b) Left censoring: In a medical scenario, when an individual has already experienced an event of interest before enrollment into the study, then that individual is said to be left-censored.

A special case of left censoring is the left truncation when an individual under study is at risk before enrollment into the study. Observation C in Figure 2.1 is one example of left truncation as the individual is at risk even before entering the study.

2. Interval censoring, a censoring mechanism encountered when individuals under study are observed periodically for the event of interest. Observation D in Figure 2.1 shows an interval-censored case, where the event is known to happen with certainty in a period $(S_L, S_R]$, except that the exact time at which the event has happened is unknown. One interesting aspect of this type of censoring is that theoretically it includes both the cases of point censoring; Hence, it is integral for handling the interval-censored observations.

With the knowledge of various censoring mechanisms, it is evident that censoring time gives more information than the fact that it exceeds the survival time, and, in such a scenario, the analysis conducted by omitting the censored data is biased as the valuable information of survival times from the censored observations is not taken into account. Four approaches have been discussed in [6] for the analysis of censored data and are listed as follows:

1. Complete data analysis, where the censored observations are completely ignored, and the analysis is conducted on the uncensored data, leading to a biased outcome.

2. The imputation approach can be further classified into left-point and right-point imputation. Left-point imputation assumes that all the censored cases fail right after the censoring time resulting in underestimating the survival probabilities. On the other hand, right-point imputation assumes that the censored cases never fail resulting in the overestimation of survival probabilities.

3. Analysis with dichotomized data considers a fixed period during the study, and analysis is conducted for the incidence of occurrence versus non-occurrence. The survival time is disregarded in this case. This method has severe limitations and is acceptable under the following conditions:

   (a) When the risk of failure is low, i.e., the risk-periods are longer.

   (b) When the aim is to study the covariates to prevent the event rather than to prolong the survival time.

4. The likelihood-based approach is, by far, the most effective estimation approach that adjusts depending on the censoring of individual observations.

2.1.4 Common survival models

Preliminary work in survival analysis mainly focused on the population as a whole rather than the factors affecting the population. Kaplan and Meier [7] conducted the first systematic study on estimating survival curve. Cox in 1972 [3] was among the first to incorporate covariates into modelling its effect on the underlying degradation process. Many attempts have been made [8], [9], [10] aimed at using neural networks to ease some of the assumptions of the Cox Proportional Hazards (CPH) model and capture the degradation process. In a major advance in 2008, Ishwaran et al. [11] reported the usage of random forests for survival analysis. Later, several studies, for instance [12], [13], [14], [15], [16], [17], [18], and more recently [19], conducted an analysis using deep neural networks, further alleviating the assumptions of CPH and presenting novel robust methods that are capable of estimating the baseline hazard function.

The most common survival models are as follows:

2.1.4.1 Kaplan-Meier estimator

Kaplan and Meier [7] provided a non-parametric method for estimation of the survival curve from incomplete observations in 1958. The method aims to plot the survival curve of different cohorts at population level and one limitation is that it does not incorporate individual covariates to estimate the risk predictions.

Let d_i be the number of events at time t_i, and let $t_{(1)} < t_{(2)} < ... < t_{(m)}$ denote the ordered distinct event times and let r_i denote the number of subjects still at risk at time t_i. Then the Kaplan-Meier's maximum likelihood estimator of the survival function is given by

$$\hat{S}(t) = \prod_{i:t_i<t} (1 - \frac{d_i}{r_i}) \tag{2.6}$$

The ratio $\frac{d_i}{r_i}$ estimates the conditional probability of subjects' survival past time t_i, and the conditional probability of surviving until time t_i is given by the compliment of $\frac{d_i}{r_i}$, i.e. $1 - \frac{d_i}{r_i}$. The intuition behind the estimation of $\hat{S}(t)$ is that the survival probability at time t is obtained by multiplying the conditional probabilities for all the event times up to time t.

The survival curve on using KM estimator is estimated only at the event times, and censoring is not considered, although the risk set r_i changes with respect to censored observations.

2.1.4.2 Cox proportional hazards model

Cox [3] incorporated covariates into modelling the degradation process considering the effects of different covariates in 1972. This method of estimating survival times or the time to failure is commonly referred to as proportional hazards model due to the underlying assumption that the ratio of two hazards at any time are proportional in nature.

The CPH model in general form is outlined as follows

$$\lambda(t|X) = \lambda_0(t)\exp(X\beta) \tag{2.7}$$

where, $\lambda(t|X)$ is a product of two non-negative terms, namely $\lambda_0(t)$, a baseline hazard function and $\exp(X\beta)$ generalized to include any risk function $\exp(h(X))$[3], referred to as risk score. This model can often be regarded as a multiplicative model where the effect of predictor variables act to multiply the baseline hazard.

An unknown p $\times$ 1 regression vector β, is estimated using partial likelihood estimation (PLE) and the β vector is tuned while training to optimize the Cox partial likelihood represented by $L_c(\beta)$ and given by the equation

$$L_c(\beta) = \prod_{i:E_i=1} \frac{\exp(\hat{h}_\beta(x_i))}{\sum\limits_{j\in R(T_i)} \exp(\hat{h}_\beta(x_j))} \tag{2.8}$$

where the values of T_i, E_i, and x_i are the respective event time, event indicator, and baseline data for the i^{th} observation.

The partial likelihood is the product of the probability at each event time T_i that the event has occurred to individual i, given the set of individuals still at risk at time T_i.

The negative log partial likelihood can then be written as

$$l_c(\beta) = -\sum_{i:E_i=1}(X_i\beta - \log\sum_{j\in R(T_i)}\exp(X_j\beta)) \tag{2.9}$$

The loss function $l_c(\beta)$ is minimized while training the model by tuning the weights and the model performance is assessed using concordance index (C-index) as the metric. The respective β coefficients obtained represent the increase in the log-hazard ratio for one unit increase of a predictor variable x_i, assuming the rest of the predictor variables are held constant. Similarly, $\exp(\beta_i)$ represent the hazard ratio for one unit increase of x_i.

The Cox method of estimating the hazard function is often regarded as semi-parametric because that it involves the parameters, $\lambda_0(t)$, baseline hazard which is an unknown function that determines the hazard at state space $X = 0$, and $\exp(h(X))$, estimated using parametric methods.

There are some basic assumptions outlined by Cox in using this method and Kumar, Klefsjö in [20] outlined them as follows:

1. "The times to failures are independent and identically distributed".
2. "All the influential covariates are included in the model".
3. "The ratio of any two hazard rates are constant with respect to time".

Often times, survival function is to be estimated to understand the survival distribution of population. To estimate the survival function, the cumulative hazard is estimated and then using Equation 2.4, survival function at any time point is estimated. Nelson-Aalen estimator [21] [22], and Breslow's estimator [23] are two popular choices to estimate the cumulative hazard function.

2.1.5 Baseline hazard estimation

Breslow's estimator [23] is used for estimation of the survival analysis in the later part of this thesis. If the observed event times contain no tied events, the cumulative baseline hazard estimate following Breslow's method [23] is given by

$$\hat{\Lambda}_0(t) = \sum_{t_i \leq t} \hat{\lambda}_0(t_i) \tag{2.10a}$$

$$= \sum_{t_i \leq t} \frac{1}{\sum_{j \in R_i} \exp(\hat{h}_\beta(x_j))} \tag{2.10b}$$

where R_i is the risk set, i.e., set of subjects still at risk at the event time t_i, and $\hat{\Lambda}_0(t)$ is considered to be the non-parametric maximum likelihood estimator of the cumulative baseline hazard function [24].

From equation 2.4, the baseline survival estimate can be estimated as follows,

$$\hat{S}_{(0,BR)}(t) = \exp(-\hat{\Lambda}_0(t)) \tag{2.11}$$

which gives the baseline estimate of the survival function at distinct event times.

For a subject with covariate vector $X = x^*$, the survival function at any time t can be estimated as [25]

$$\hat{S}(t \mid X = x^*) = \hat{S}_{(0,BR)}(t)^{\exp(\hat{h}_\beta(x^*))}. \tag{2.12}$$

2.2 Deep Learning

2.2.1 Artificial neural networks

2.2.1.1 Artificial neuron

The most fundamental unit of the artificial neural network is called an artificial neuron. McCulloch and Pitts in 1943 [26] proposed the simplified model of a neuron, as shown in Figure 2.2.

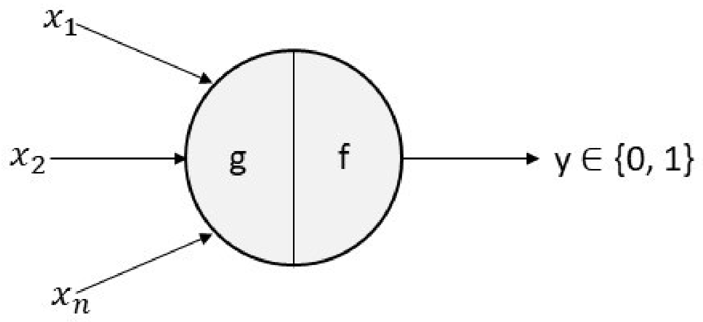


Figure 2.2: McCulloch Pitts neuron

MuCulloch Pits neuron takes in binary input and outputs a binary result where function g aggregates the inputs, i.e. $\{g(x) = \sum_{i=i}^{n} x_i\}$, and function f takes a decision based on $g(x)$ and a preset threshold θ. The function can be mathematically represented as

$$y = f(g(x)) = \begin{cases} 1 & \text{if } g(x) \geq \theta \\ 0 & \text{if } g(x) < \theta \end{cases} \tag{2.13}$$

Later in 1958, Frank Rosenblatt proposed the so-called classic perceptron algorithm [27] which alleviates feeding binary inputs to the neuron, and can take real values as input and outputs a binary result. Rosenblatt introduced weights $\{w_1, w_2, \ldots, w_n; w_i \in \Re\}$, expressing the relative importance of inputs to an output. The output of the Rosenblatt perceptron is mathematically expressed as

$$y = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} w_i * x_i \geq \theta \\ 0 & \text{if } \sum_{i=1}^{n} w_i * x_i < \theta \end{cases} \tag{2.14}$$

The threshold θ can be set to an arbitrary value to generalize the above model of representing an artificial neuron, and the weights can then be allowed to learn by following a mechanism. For this reason, bias $b = -\theta$ is introduced, and the mathematical representation for this generalization is as follows

$$y = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} w_i * x_i + b \geq 0 \\ 0 & \text{if } \sum_{i=1}^{n} w_i * x_i + b < 0 \end{cases} \tag{2.15a}$$

$$= \begin{cases} 1 & \text{if } \sum_{i=0}^{n} w_i * x_i \geq 0 \\ 0 & \text{if } \sum_{i=0}^{n} w_i * x_i < 0 \end{cases} \tag{2.15b}$$

where $x_0 = 1$ is called a bias neuron, and the weight associated with the bias neuron is $-\theta$. The advantage of using bias is twofold, as a bias neuron and to assess the model for bias vs. variance trade-off. Figure 2.3 illustrates a simple perceptron with bias and weights associated with each of the associated inputs.

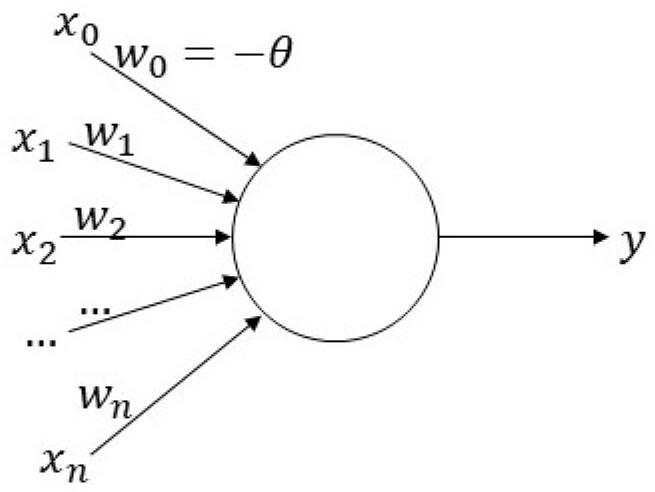


Figure 2.3: Rosenblatt perceptron

The output value of y can be changed based on the execution of an activation function, which triggers whether the neuron can fire. Several commonly-used activation functions, such as the step function (used to fire a neuron in McCulloch Pitts neuron), sigmoid function, tanh function, and ReLu, are in use depending on the problem.

Below is the list of the changes in the output y with use of different activation functions:

1. Sigmoid function:

$$y = \sigma(z) \tag{2.16a}$$

$$= \sigma(w \cdot x + b) \tag{2.16b}$$

$$= \frac{1}{1 + \exp(-z)} \tag{2.16c}$$

2. ReLu:

$$y = max(0, z) \tag{2.17}$$

3. Tanh function:

$$
\begin{aligned}
y &= tanh(z) && (2.18a)\\
&= \frac{2}{1+\exp(-2z)} - 1 && (2.18b)\\
&= 2\sigma(2z) - 1 && (2.18c)
\end{aligned}
$$

2.2.1.2 Artificial neural networks

Most of the results using neural networks are achieved by using a network of perceptrons, usually called architecture and not just with one perceptron. Multilayer perceptrons (MLP) are one particular case of artificial neural networks consisting of more than one perceptron. MLPs consist of an input layer, an output layer, and an arbitrary number of hidden layers. The hidden layers are an integral part of the architecture, as this is where the non-linear transformations of the input data take place. The ability to learn the highly complex data increases with the number of hidden layers at the expense of computational time, i.e., the network takes a very long time to train to adjust the weights of the network to get the desired output. A multilayer perceptron with four or more layers, like the one shown in Figure 2.4 [28], is called a deep neural network. A simple illustration of the deep neural network is shown below,

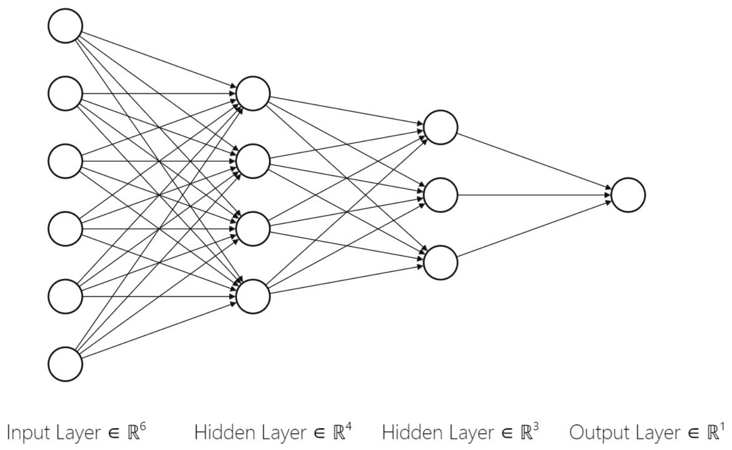


Figure 2.4: Representation of deep neural network

where the leftmost layer in the network is called the input layer, and the rightmost layer is called the output layer. The two layers in the middle are called the hidden layers.

There are different types of neural network architectures in use today such as a recurrent neural network (RNN), convolutional neural network (CNN), generative adversarial network (GAN) and the most common of all is the deep feedforward neural network (DFFN) where the output from one layer is passed as input to the next layer. DFF neural networks are fully connected in general, which means that the information from the input layer is passed through the hidden layers to the output layer employing various activation functions.

Every node in the network is arbitrarily assigned a weight during the initialization of the network. These weights are later tuned while training the network using the most widely used back propagation algorithm [9], which is discussed in the next section.

2.2.1.3 Training artificial neural networks

The mode of training the network depends on the type of problem under consideration, i.e., supervised or unsupervised. The network is trained, in case of supervised learning, to minimize the prediction error, or in case of unsupervised learning, to compress the input information. Emphasis is made on supervised learning in this section.

In supervised learning, the network is given with the input data and the labels associated with it. The aim of training the network is to update the weights of the network and to find the right balance between bias and variance of the model, i.e., to update the weights and biases so that the network then makes predictions as close as the target values. The process of updating the weights and biases is accomplished by learning algorithms.

The learning of the network is quantified in terms of deviation of the predictions from expected output with a cost function (C) i.e., the cost for a poorly learned network is higher and this is an indication for the weights and biases to be adjusted. Also, the choice of selection of the cost function depends on whether it is a regression or a classification problem. For example, squared error loss or multi-class cross-entropy loss is used as cost function respectively, for the above two cases. A simple illustration of the network with bias representation in the hidden layers is shown in Figure 2.5 [28]. The topmost node in input layer, and each hidden layer is a bias node. The inclusion of a bias node increases the flexibility of the model.

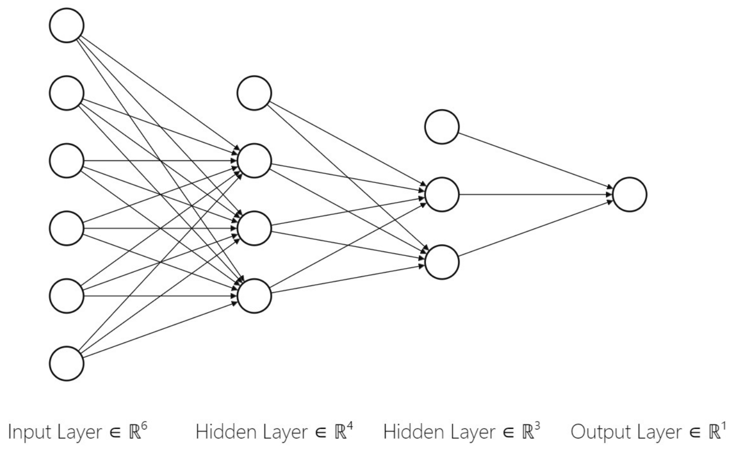


Figure 2.5: Representation of deep neural network with bias

The initial loss is calculated during the forward pass and the learning process involves computing the gradients of the cost function during back propagation at each iteration and then minimizing the cost function on the updated weights (w') and biases(b') i.e.,

$$w' = w - \eta \frac{\delta C}{\delta w} \tag{2.19a}$$

$$b' = b - \eta \frac{\delta C}{\delta b} \tag{2.19b}$$

where η is the learning rate, which could be controlled to vary the speed at which the network learns and updates the parameters. Setting a very small learning rate could lead to slow convergence, while a large learning rate can hinder convergence and causes the cost function to fluctuate.

Several supervised learning algorithms applied to the network employ optimizers to update weights and baises, some of which as explained by Meireles et al. in [29] are as follows:

1. Early Supervised Learning Algorithms:

While these algorithms are widely employed on a single neuron with varied input size, they are not particularly beneficial in solving MLPs.

   (a) Perceptron learning

   (b) LMS or Widrow-Hoff learning

   (c) Grossberg learning

2. Gradient Descent Methods:

   When dealing with a collection of perceptrons to handle huge data, the learning process also becomes complex, resulting in increasing time complexity with the amount of data. For this reason, there is a trade-off between the measure of model performance vs the time for learning the parameters, and as a result, gradient descent methods are broadly categorized into three variants [30], namely,

   (a) Batch gradient descent

      Let weights and biases collectively be represented by θ, then θ in batch gradient descent is updated as follows:

      $$\theta = \theta - \eta \cdot \nabla_\theta C(\theta) \tag{2.20}$$

      The gradient of the cost function w.r.t weights and biases is computed for the entire dataset at each iteration and as a result, the time complexity of batch gradient descent is huge. Thus, for a relatively huge dataset, the batch gradient descent can be very slow and can also cause redundancy because of the gradients recomputed for similar examples at each iteration.

   (b) Stochastic gradient descent

      Let x^i represent the i^{th} training sample, the stochastic gradient descent computes the gradient at each training sample, and simultaneously updates the parameters. Hence, the parameters are updated as follows:

$$\theta = \theta - \eta \cdot \nabla_\theta C(\theta; x^i) \tag{2.21}$$

As the gradient is computed at every training sample and the updates are performed simultaneously, stochastic gradient descent is prone to high variance and can cause the cost function to fluctuate heavily. This fluctuation in the cost function can sometimes lead to a better local optimum. Hence, with careful setting of the learning rate, this method can eventually converge to a local or global optimum like the batch gradient descent.

(c) Mini-batch gradient descent

When the number of training samples are large, computing gradients of the cost function for each input, as is performed in the batch gradient descent, can take very long time and thus make the learning very slow. Conversely, mini-batch gradient descent can be used to speed up learning by only computing the gradient of the cost function at randomly selected sub-sample from the training input.

Let a sub-sample of m training inputs (say $x_i; i = 1, \ldots, m$) be taken from the training data of size n, the parameter updates in mini-batch gradient descent is as follows:

$$\theta = \theta - \eta \cdot \nabla_\theta C(\theta; x_i) \tag{2.22}$$

The weights and biases of the network are randomly initialized, which acts as a starting point for random initialization of learning algorithms. Some of the several optimization techniques employed while training the network using gradient descent methods are broadly classified as follows:

1. First-Order Gradient Methods:

   (a) Backpropagation

   (b) Backpropagation with momentum

2. Second-Order Gradient Methods:

    (a) Newton method

    (b) Gauss-Newton method

    (c) Levenberg-Marquardt (LM) method

Of all the supervised learning methods listed above, gradient descent methods have proven to be very successful in training the neural networks.

2.2.2 Artificial neural network approach to survival analysis

The development of deep learning and improved hardware architecture showed promising results in applying the neural networks to survival analysis. This helped researchers implement a broad variety of network architectures to recognize patterns in the data and achieve state-of-the-art results. This section briefly discusses some of the deep learning techniques in survival analysis and the key distinctions of the techniques in achieving the desired output.

Ranganath et al. [12] in 2016, Alaa and Schaar [14] in 2017 proposed network architectures based on Bayesian framework. Ranganath et al. [12] introduced a hierarchical generative model called the deep survival analysis using the deep exponential families to capture the dependencies between the covariates and the failure time. Alaa and Schaar [14] used a deep multi-task Gaussian process to capture complex interactions between covariates and cause-specific survival times, without any parametric assumptions on the hazard rates.

Later in 2017, Katzman et al. [13] introduced Deepsurv, a deep neural network extension of the CPH model to effectively model complex interactions between the covariates and risk of failure. Section 3 of this document provides a detailed explanation of this method and its use in manufacturing systems is discussed in Section 4.

Giunchiglia et al. [17] in 2018 and Ren et al. [18] in 2019 proposed alternative approaches using recurrent neural networks. Giunchiglia et al. [17] proposed a fully parametric survival model to compute the risk score and the survival function of each subject for personalized survival analysis called RNN-SURV. On the other hand, Ren et al. [18] proposed a model called Deep

Recurrent Survival Analysis that combines deep learning to model the conditional probability of each event for each sample and survival analysis that handles censoring.

3. DEEPSURV

3.1 Overview of Deepsurv

The inability of the CPH model to capture the complex relationships among covariates in non-linear risk surfaces and the inability of the single hidden layer neural networks [9] to outperform CPH is the primary motivation behind employing the deep learning techniques to model the complex interactions between the covariates and the risk of failure. Deepsurv, a non-linear extension to the Cox proportional hazards model, is developed from the drawbacks of the CPH model and the Faraggi-Simon network. Deepsurv is a multi-layer perceptron and employs deep architecture to model the complex association among the covariates without any prior assumptions on the underlying risk surface.

Generally, covariates data (X), event indicator (e), and event time (t) make up survival data. Deepsurv is a feed forward neural network that takes input as an n-dimensional array of the covariates. The network can be configured with any arbitrary number of hidden layers, and the input data is propagated through these layers. Each hidden layer is set to have a non-linear activation function and can be configured to include a dropout layer, and a batch normalization layer. The output layer of the network is a single node with linear activation function, and the output is the estimated log partial hazard ($\hat{h}_\theta(x)$).

Deepsurv employs a fully connected deep neural network architecture that uses average negative log-likelihood as loss function. The loss function of the Deepsurv model is derived from Cox partial likelihood and Faraggi Simon network. In regard to the general training of the CPH model, the weights β are tuned to optimize the Cox partial likelihood, given by $L_c(\beta)$.

$$L_c(\beta) = \prod_{i:E_i=1} \frac{\exp(\hat{h}_\beta(x_i))}{\sum\limits_{j \in R(T_i)} \exp(\hat{h}_\beta(x_j))} \tag{3.1}$$

Faraggi-Simon's network, a preliminary attempt of nonlinear extension of the CPH model, optimizes weights of the network using a modified version of $L_c(\beta)$ ($\hat{h}_\beta(x)$ replaced by the output of the network $\hat{h}_\theta(x)$) coupled with l_2 regularization. The "penalized likelihood function" of Faraggi-Simon's network [9] is thus given by

$$L_c(\theta) + \lambda \sum_{k=1}^{p} \theta_k^2 \tag{3.2}$$

where $L_c(\theta)$ is the modified Cox partial likelihood parameterized by the weights of the Faraggi-Simon's network, λ is the l_2 regularization parameter, and p is the number of predictor variables (X: $n \times p$ data matrix).

Similar to Faraggi-Simon's network, Deepsurv employs the penalized negative average log-likelihood function given by the equation:

$$\{-\frac{1}{N} \sum_{i \in D} (\hat{h}_\theta(x_i) - log(\sum_{j \in R_i} e^{\hat{h}_\theta(x_j)}))\} + \lambda \theta_k^2 \tag{3.3}$$

where D is the set of subjects with observed event ($e = 1$), N is the total number of subjects with event observed, and $\hat{h}_\theta(x)$, output of the network, is the estimated log partial hazard.

The advantage of Deepsurv lies in the usage of modern techniques in deep learning such as dropout [31], weight decay regularization, batch normalization [32], learning rate scheduling [33], gradient clipping and some of the advanced optimizers used as part of the gradient descent methods. These modern optimizers or the update functions are readily available with the Lasagne python library [34], built on top of Theano [35], which Deepsurv uses for constructing the fully connected deep neural network. Some of the update functions that can be used with Deepsurv are listed in Table 3.1.

Update Function	**Description**
sgd	Stochastic Gradient Descent updates
momentum [36]	Stochastic Gradient Descent updates with momentum
nesterov_momentum [37]	Stochastic Gradient Descent updates with Nesterov momentum
adagrad[38]	Adagrad updates
rmsprop [39]	RMSprop updates
adadelta [40]	Adadelta updates
adam [41]	Adam updates
adamax [41]	Adamax updates
amsgrad [41]	AMSGrad updates

Table 3.1: List of update functions

While maintaining the constant hazard ratio assumption from the CPH model, Deepsurv also assumes the data is void of any tied events while training and measures are to be taken while dealing with the tied events. Methods like Breslow approximation or the Efron approximation are generally employed to handle the ties. Nonetheless, Deepsurv is capable of learning the complex relationship among the covariates. The model's predictive capability is validated against the metric widely used in survival analysis called the concordance-index (c-index). The implementation details of this method on a manufacturing plant floor data are discussed in Section 4.

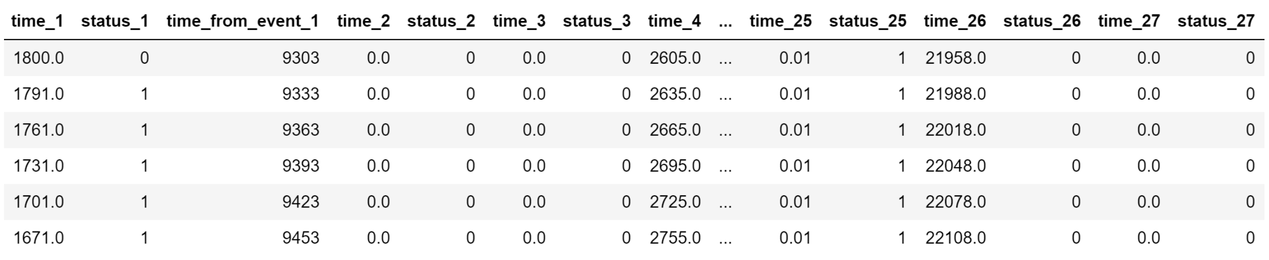
4. CASE STUDY AND ANALYSIS

The objective is to apply Deepsurv to derive a data-driven prognostic model to assess the risk of any impending breakdowns of machinery and mechanical components in a manufacturing system. For this purpose, a manufacturing plant floor consisting of 25 machines integrated with sensors is considered for this analysis, and Deepsurv is used to train and validate the analysis. The results validated at the machine level show that the model can accurately estimate the risk of failure 75$\%$ of the time. The analysis also shows that this type of data-driven model can be used for preventive maintenance.

4.1 Overview of the Dataset

The data collected from the sensors integrated with the machines, consisting of timestamps and fault codes for a breakdown, is processed to a general survival analysis data format (X: covariates data, e: event indicator, and t: event time).

A dataset for each machine is generated from the fault codes and time stamps. The dataset contains the columns indicating event times and indicators, and for each target machine, a column named '$time\_from\_event$' provides information regarding the target machine while training the model. Figure 4.1 shows the overview of a dataset of machine 1.



time_1	status_1	time_from_event_1	time_2	status_2	time_3	status_3	time_4	...	time_25	status_25	time_26	status_26	time_27	status_27
1800.0	0	9303	0.0	0	0.0	0	2605.0	...	0.01	1	21958.0	0	0.0	0
1791.0	1	9333	0.0	0	0.0	0	2635.0	...	0.01	1	21988.0	0	0.0	0
1761.0	1	9363	0.0	0	0.0	0	2665.0	...	0.01	1	22018.0	0	0.0	0
1731.0	1	9393	0.0	0	0.0	0	2695.0	...	0.01	1	22048.0	0	0.0	0
1701.0	1	9423	0.0	0	0.0	0	2725.0	...	0.01	1	22078.0	0	0.0	0
1671.0	1	9453	0.0	0	0.0	0	2755.0	...	0.01	1	22108.0	0	0.0	0

Figure 4.1: Overview of machine 1 dataset

The columns '$time\_1$' and '$status\_1$' are treated as target variables. The column '$time\_1$'

indicates the time till the next event, and any time beyond 1800 sec is treated as censored. For instance, where an event is observed, say 1791 from Figure 4.1, '$time\_1 = 1791$' represents the time until the next event. The corresponding columns in the row represent the state of other machines at this particular timestamp.

The column '$status\_1$' acts as an indicator, and the value equals one if an event is observed and 0 if either the instance is censored, or no event is observed.

The column '$time\_from\_event\_1$' gives information about the time elapsed since the last restoration from a breakdown. Likewise, for '$time\_1 = 1791$', it indicates that the last event was observed 9333 seconds ago, and the next event is likely to happen in 1791 seconds. The inclusion of this column in the covariates makes the model less biased towards the target machine, as the valuable information about the target machine is passed onto the model in the training phase.

The columns from '$time\_2$' to '$status\_27$' are treated as covariates, and these columns provide the information of time elapsed since the last restoration of other machines and corresponding status, respectively. Considering '$time\_1 = 1791$' again, at this timestamp, the '$time\_2$' column indicates that an event has not been observed yet, and hence the '$status\_2$' of machine 2 is 0. In this case, the state of machine 2 is likely to change in the future after 1791 seconds. If not, then the time gets accumulated, and this kind of behavior is observed in the case of '$time\_26$.'

Although the overview of this dataset is presented when machine one is treated as a target machine, the terminology holds for the data of other machines too. The corresponding target variables and covariates vary with the target machine.

4.2 Implementation Details

A fully connected deep neural network is considered for the analysis. The network consists of two hidden layers of 40 nodes each and an output layer of one node. The size of the input layer is determined by the number of observations passed while training the network. The source code for implementing the Deepsurv model is available in Appendix A.

4.2.1 Hyperparameter tuning

Features or covariates are first normalized, and the data is processed to a survival data format $\{X, e, \text{and } t\}$ before feeding the data for model training. The hyperparameters of the model, namely learning rate, learning rate decay, and momentum, are tuned by conducting a series of experiments to obtain optimum results. Also, different update functions or optimizers are tested for optimum results. Table 4.1 gives an overview of the sequential steps taken to optimize the hyperparameters. The data is shuffled, and the first step in the series is started by selecting the first 1000 observations for training. The sample size is gradually increased, and the model's performance is assessed based on the concordance index. Deepsurv uses the lifelines package to calculate the c-index between two series of event times, one series being the actual events times and the other being the estimated partial hazard. The best result is obtained from the last experiment listed in Table 4.1, and the c-index of the model on validation and test data is above 0.78, and hyperparameters from this experiment are considered for further analysis. The optimal hyperparameters for the model are listed in Table 4.2.

4.2.2 Model training

The model is trained on 70% of the data, validated on 20% of the data using the optimal hyperparameters from Table 4.2. The remaining 10% is left for testing the trained model's efficiency. The training data is shuffled to ensure randomness in the data, and as many as 500k observations covering a wide range of possible instances from the original dataset are provided for training.

A penalized negative log-likelihood loss function is used while training the model, as explained in section 3.1. A large value of l_2 regularization coefficient (15.0) is selected to penalize large weights. Figure 4.2 explains the reduction of the loss during training and the model's performance validated at every 250 epochs using the concordance index as an evaluation metric. As shown in Figure 4.2, although the decrease in loss is significantly smaller, there is a substantial increase in the model's accuracy beyond 250 epochs. However, due to time complexity, the model is trained only for 2000 epochs.

Sample size	Hyperparameters			Optimizer	Concordance index	
	learning_rate	**lr_decay**	**momentum**		**validation**	**test**
1000	1.00E-04	0.002	0.85	adam	0.679	0.431
1000	1.00E-04	0.0002	0.85	adamax	0.676	0.446
1000	1.00E-04	0.0002	0.85	amsgrad	0.664	0.494
1000	1.00E-04	0.0002	0.85	adadelta	0.465	0.417
1000	1.00E-04	0.0002	0.85	rmsprop	0.67	0.448
1000	1.00E-04	0.0002	0.85	adagrad	0.481	0.458
1000	1.00E-04	0.0002	0.85	nesterov_momentum	0.577	0.459
1000	1.00E-04	0.0002	0.85	momentum	0.594	0.505
1000	1.00E-04	0.0002	0.85	sgd	0.571	0.435
1000	1.00E-04	0.0005	0.8	momentum	0.567	0.464
1000	1.00E-04	0.0005	0.9	momentum	0.574	0.506
1000	1.00E-04	0.0004	0.9	momentum	0.572	0.454
1000	1.00E-04	0.001	0.9	momentum	0.621	0.542
10000	1.00E-04	0.001	0.9	momentum	0.619	0.5
10000	1.00E-04	0.001	0.9	amsgrad	0.673	0.557
1000	1.00E-02	0.001	0.9	amsgrad	0.622	0.499
1000	1.00E-03	0.001	0.9	amsgrad	0.644	0.537
10000	1.00E-03	0.001	0.9	amsgrad	0.73	0.62
10000	1.00E-03	0.001	1.0	amsgrad	0.724	0.629
10000	1.00E-03	0.001	1.0	momentum	0.632	0.489
10000	1.00E-02	0.001	1.0	amsgrad	0.736	0.632
10000	1.00E-02	0.01	1.1	amsgrad	0.732	0.627
10000	1.00E-02	0.015	1.0	amsgrad	0.729	0.632
10000	1.00E-02	0.025	1.0	amsgrad	0.734	0.634
586194	1.00E-02	0.025	1.0	amsgrad	0.78	0.781

Table 4.1: Hyperparameter optimization

Hyperparamater	**Value**
L2_reg	15.0
batch_norm	True
dropout	0.1
hidden_layers_sizes	[40,40]
learning_rate	1.00E-02
lr_decay	0.025
momentum	1.0
standardize	False

Table 4.2: Optimal hyperparameters

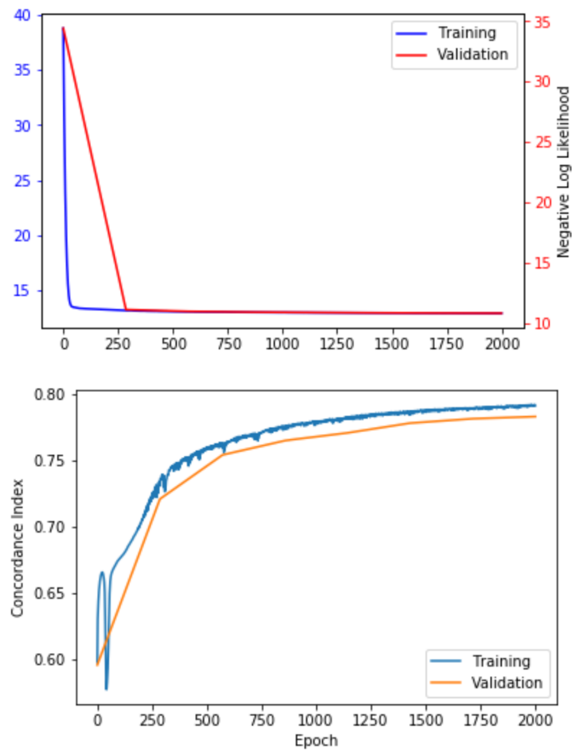


Figure 4.2: Training curve of Deepsurv model

The model's output, log partial hazard $(h_\theta(x))$, is used to estimate the baseline hazard function and the baseline survival function using Breslow's method, as described in Section 2.1.5. Both the cumulative hazard function $(\Lambda(t))$ and survival function $S(t|x)$ are estimated from the baseline hazard function using the equations 2.10 and 2.12, respectively. The following section explains the results and analysis of data collected from five different machines.

4.3 Results

The distribution of the estimated log partial hazard for censored and uncensored observations is shown in Figure 4.3. The boxplot shows that almost 75% of the censored observations (status=0) in the test data have log partial-hazard less than or equal to 0.84. On the other hand, for the observations where an event is observed, the model can estimate for almost 75% of the cases, a log partial hazard value greater than 0.747.

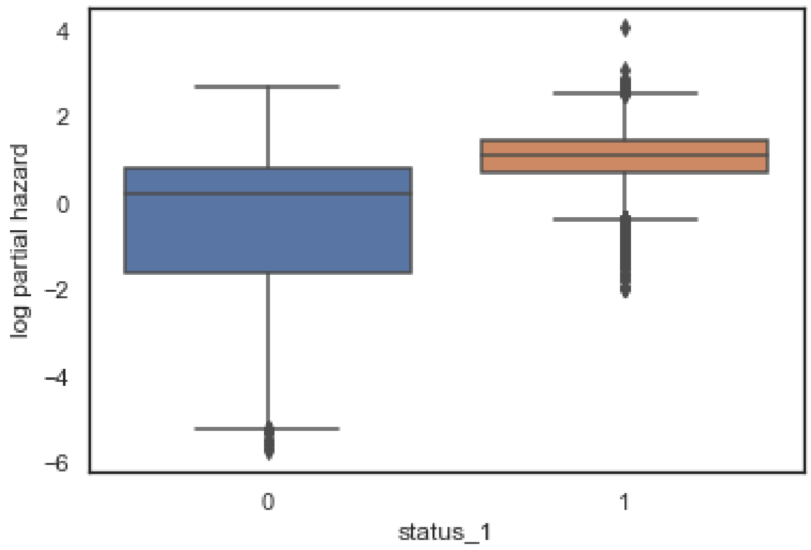


Figure 4.3: Distribution of estimated log partial hazard

The model output (log partial hazard) is in line with what is expected in case of the censored observations, and in case an event is observed. The model can correctly classify an event with a higher log partial hazard and shows an increased risk with an increase in log partial-hazard.

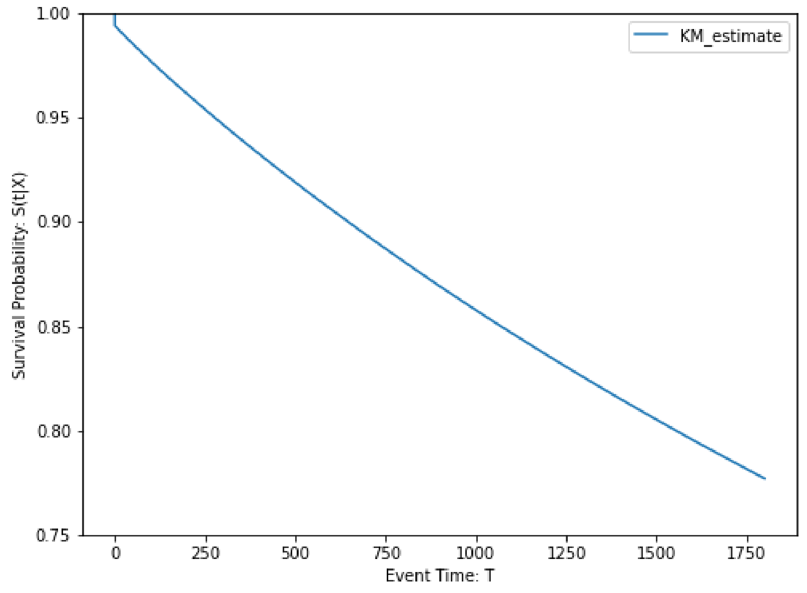


Figure 4.4: Survival probability estimates using Kaplan Meier estimator

To further demonstrate the Deepsurv model's effectiveness, survival probability estimates computed using log partial-hazard (output of the model), and Breslow's estimator are compared to the survival probability estimates from the Kaplan-Meier estimator, which does not take into account the effect of covariates. The survival function plot in Figure 4.4 indicates the Kaplan-Meier estimates of the survival function at distinct event times. The survival function plot in Figure 4.5 indicates the survival function estimates from the Deepsurv model. Here, the Kaplan Meier estimation of survival probability is considered as a baseline, and the Breslow's method estimates are compared against the baseline Kaplan-Meier estimates.

Breslow's estimation of survival probability appeared to produce much better results than Kaplan-Meier estimation, and consideration of the effect of covariates while Breslow's estimation decreased the overall survival probability at distinct event times. This is especially helpful in a manufacturing plant scenario where the breakdown of one machine impacts the status of other machines, and the inclusion of the information from various machines into the model can help in better estimation of the survival probability.

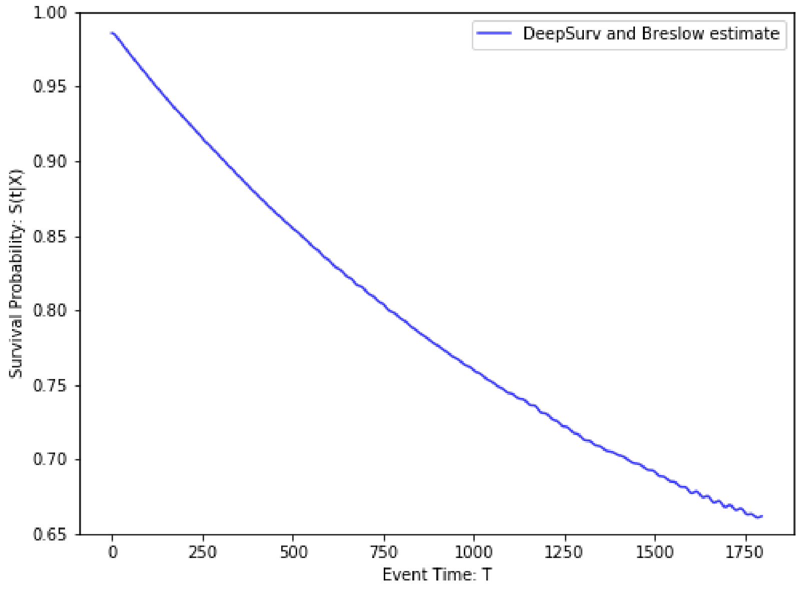


Figure 4.5: Survival probability estimates using log partial hazard and Breslow estimator

The model is also assessed on the data collected from five machines and the analysis is summarized in the Figures 4.6 and 4.7. The low Brier scores (< 0.25) of the model shown in Figure 4.6 at different timestamps on all the machines indicate the superior predictability at all possible event times, and the overall performance of the model at all times is shown in Figure 4.7. The model's reliable ranking of the survival times is shown in Figure 4.8. As the Deepsurv model results outperformed the survival probability estimation from known-best methods, further analysis is conducted to plot the hazardogram to demonstrate the change in survival probability when an impending breakdown is evident to happen.

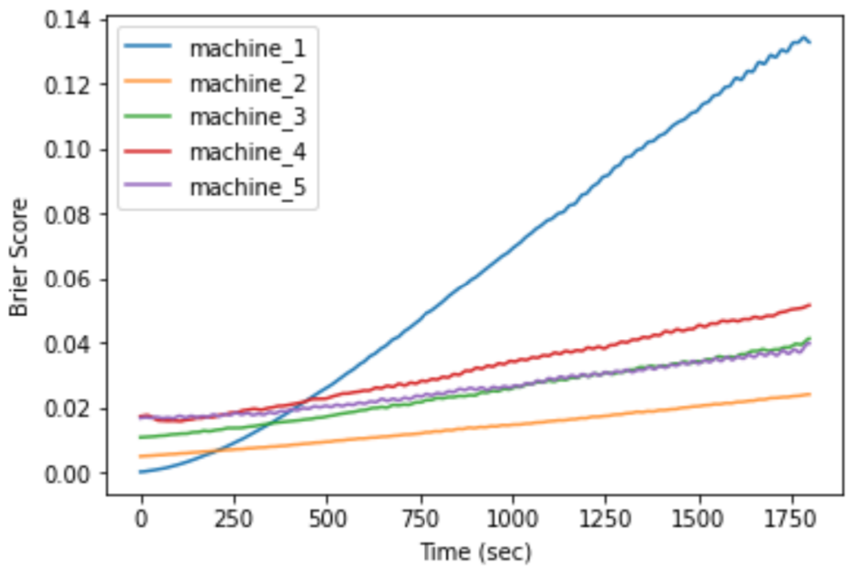


Figure 4.6: Brier score of Deepsurv model

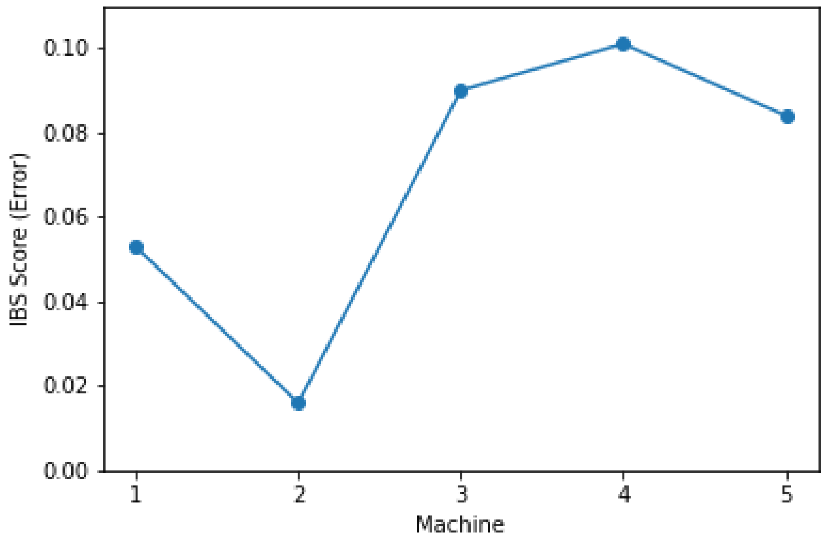


Figure 4.7: Integrated Brier score of Deepsurv model

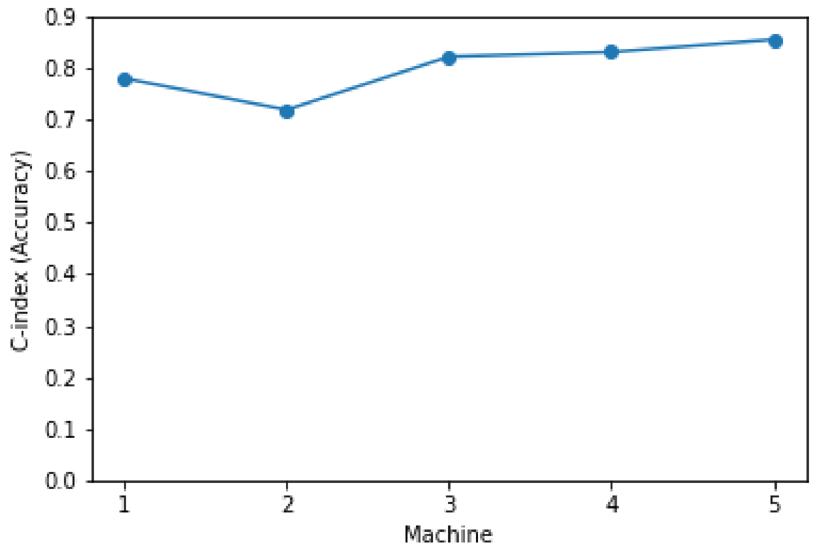


Figure 4.8: Accuracy of Deepsurv model

4.3.1 Hazardogram

A hazardogram, essentially a color plot is used to flag any impending breakdown based on the estimated survival probability for an episode. The prediction horizon considered is up to 1800 seconds. The plot explains the variation of survival probability over the prediction horizon, and the variability is explained for different episodes.

The hazardogram of machine 1 data is shown in Figure 4.9, and the code to plot the hazardogram is provided in Appendix A.

The hazardogram plots for all of the 5 machines training and test data are included in Appendix B.

The preliminary results from the hazardogram show that the model can capture some of the impending breakdowns at the early stages of the fault propagation. Here, the fault propagation is measured in terms of the survival probability shifting from a safe level to a critical level. The threshold for the safe level is assumed to be 0.5, i.e., at any point, when the survival probability shifts below 0.5, the instance is categorized as critical.

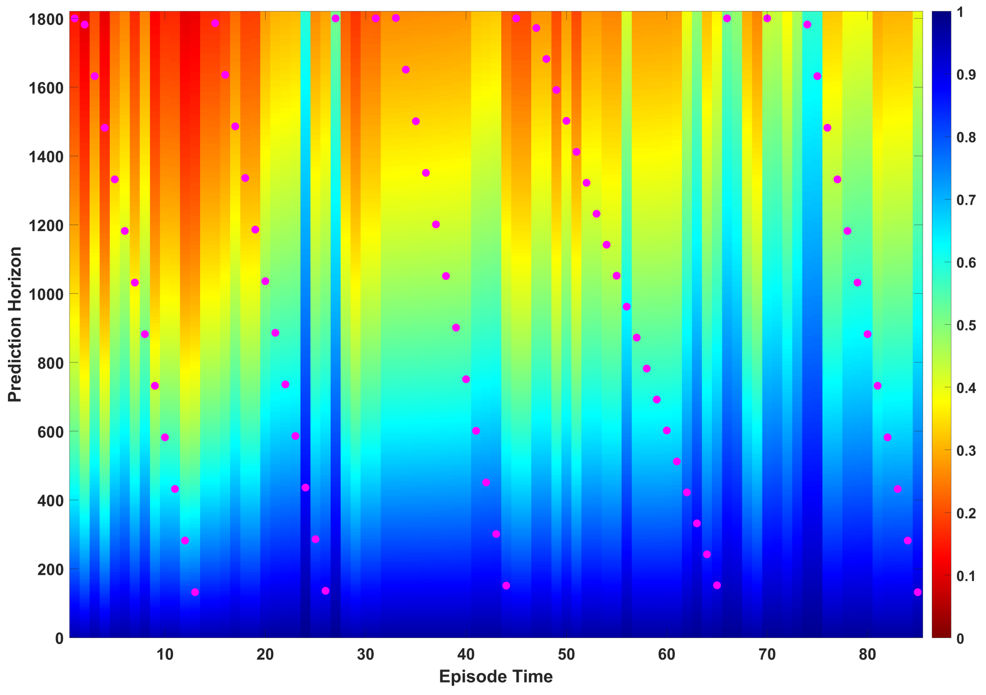


Figure 4.9: Hazardogram

The dots overlaid on top of the color plot are the actual event times of that particular episode. It is evident from the hazardogram that the Deepsurv model can flag an impending breakdown at early stages in most cases when the survival probability drops below the set threshold of 0.5. This leads us to the conclusion, that the Deepsurv can be used as a prognostic model to detect the impending breakdowns from the historical timestamps at early stages of fault propagation.

5. SUMMARY

5.1 Summary and Conclusions

In an attempt to analyze the fault propagation in manufacturing systems using modern-day prognostic tools in conjunction with survival analysis, Deepsurv, a non-linear Cox proportional hazards model, is used to forecast the state of a system to prevent any impending breakdowns. An industrial plant floor with machinery integrated with IoT devices is considered for this case study.

The work consists of processing the IoT data logs into a survival data format, hyperparameter tuning, and analyzing the risk predictions. The survival probability of a sample is estimated using Breslow's estimator, and the shift in the survival probability below a threshold is used to flag any impending breakdowns through hazardogram.

The Deepsurv model trained on data from 5 machines shows that the model can learn the fault propagation trend with a Brier score of less than 0.2 at all possible event times. This is also validated with the high c-index on the data from corresponding machines. The shift in the survival probability observed in the hazardogram suggests the model's ability to flag any breakdowns in the future at least 1800sec ahead. This provides a viable option to use the data logs from IoT devices and predict the future state of a machine through historical patterns in the data.

5.2 Limitations

Although Deepsurv performed better with the assumption of no-ties while learning the fault propagation trend, the analysis could be even better when advanced methods of handling tied event times are employed. Deepsurv can generalize well only when a large sample is provided while training, and this is a bottleneck for handling tied event times with some better-performing methods like the Exact method as the computations are nearly infeasible.

5.3 Discussion

Earlier in the thesis, the Cox proportional hazards model and Kaplan-Meier estimator are discussed. However, the survival probability estimates of the Deepsurv model are compared only against Kaplan-Meier survival probability estimates. The motivation to compare the Deepsurv estimates with that of Kaplan-Meier estimates goes back to the analysis presented in [5]. Here, a new technique called manufacturing system-wide balanced random survival forest (MBRSF) is developed, and its performance is compared against a variant of Cox proportional hazards model, a balanced Cox proportional hazards model as well as Kaplan-Meier model. For all these models, Kaplan Meier served as the baseline and hence to maintain consistency, the same baseline is used to compare the Deepsurv model.

Initially, the Deepsurv model network is chosen to have 5000 input nodes in the first hidden layer and 2500 in the second hidden layer, but the network's ability to generalize turned out to be poor, and the computations have become complicated with such an extensive network. Multiple network architectures with hidden layer nodes in the range of 25 to 40 are experimented with to test for the model's ability to generalize, and the best output on a sample data is achieved with the nodes listed in Table 4.2. The hazardogram plots for the training and test data of all five machines are presented and discussed to address any overfitting/underfitting of the model in Appendix B.

5.4 Future Work

A more advanced way of handling tied event times, which, when employed, can improve the identification of the fault propagation trends. Simultaneously, the network of the Deepsurv model can be experimented with, and a more stable release of deep learning frameworks like PyTorch or Tensorflow can be used instead of Theano, which has significantly less user community when writing this thesis.

REFERENCES

[1] F. Tao, M. Zhang, Y. Liu, and A. Nee, "Digital twin driven prognostics and health management for complex equipment," *CIRP Annals*, vol. 67, no. 1, pp. 169–172, 2018.

[2] R. Roy, R. Stark, K. Tracht, S. Takata, and M. Mori, "Continuous maintenance and the future–foundations and technological challenges," *CIRP Annals*, vol. 65, no. 2, pp. 667–688, 2016.

[3] D. R. Cox, "Regression models and life-tables," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 34, no. 2, pp. 187–202, 1972.

[4] R. Gao, L. Wang, R. Teti, D. Dornfeld, S. Kumara, M. Mori, and M. Helu, "Cloud-enabled prognosis for manufacturing," *CIRP Annals*, vol. 64, no. 2, pp. 749–772, 2015.

[5] S. T. Bukkapatnam, K. Afrin, D. Dave, and S. R. Kumara, "Machine learning and ai for long-term fault prognosis in complex manufacturing systems," *CIRP Annals*, vol. 68, no. 1, pp. 459–462, 2019.

[6] K.-M. Leung, R. M. Elashoff, and A. A. Afifi, "Censoring issues in survival analysis," *Annual Review of Public Health*, vol. 18, no. 1, p. 83–104, 1997.

[7] E. L. Kaplan and P. Meier, "Nonparametric estimation from incomplete observations," *Journal of the American Statistical Association*, vol. 53, no. 282, pp. 457–481, 1958.

[8] K. Liestbl, P. K. Andersen, and U. Andersen, "Survival analysis and neural nets," *Statistics in Medicine*, vol. 13, no. 12, pp. 1189–1200, 1994.

[9] D. Faraggi and R. Simon, "A neural network model for survival data," *Statistics in Medicine*, vol. 14, no. 1, pp. 73–82, 1995.

[10] W. N. Street, "A neural network model for prognostic prediction.," in *ICML*, pp. 540–546, 1998.

[11] H. Ishwaran, U. B. Kogalur, E. H. Blackstone, M. S. Lauer, *et al.*, "Random survival forests," *The Annals of Applied Statistics*, vol. 2, no. 3, pp. 841–860, 2008.

[12] R. Ranganath, A. Perotte, N. Elhadad, and D. Blei, “Deep survival analysis,” *arXiv preprint arXiv:1608.02158*, 2016.

[13] J. L. Katzman, U. Shaham, A. Cloninger, J. Bates, T. Jiang, and Y. Kluger, “Deepsurv: Personalized treatment recommender system using a cox proportional hazards deep neural network,” *BMC medical research methodology*, vol. 18, no. 1, p. 24, 2018.

[14] A. M. Alaa and M. van der Schaar, “Deep multi-task gaussian processes for survival analysis with competing risks,” in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 2326–2334, 2017.

[15] C. Lee, W. R. Zame, J. Yoon, and M. van der Schaar, “Deephit: A deep learning approach to survival analysis with competing risks,” in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[16] A. M. Alaa and M. van der Schaar, “Bayesian inference of individualized treatment effects using multi-task gaussian processes,” in *Advances in Neural Information Processing Systems*, pp. 3424–3432, 2017.

[17] E. Giunchiglia, A. Nemchenko, and M. van der Schaar, “Rnn-surv: A deep recurrent model for survival analysis,” in *International Conference on Artificial Neural Networks*, pp. 23–32, Springer, 2018.

[18] K. Ren, J. Qin, L. Zheng, Z. Yang, W. Zhang, L. Qiu, and Y. Yu, “Deep recurrent survival analysis,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 4798–4805, 2019.

[19] C. Lee, J. Yoon, and M. Van Der Schaar, “Dynamic-deephit: A deep learning approach for dynamic survival analysis with competing risks based on longitudinal data,” *IEEE Transactions on Biomedical Engineering*, vol. 67, no. 1, pp. 122–133, 2019.

[20] D. Kumar and B. Klefsjö, “Proportional hazards model: a review,” *Reliability Engineering & System Safety*, vol. 44, no. 2, pp. 177–188, 1994.

[21] W. Nelson, “Theory and applications of hazard plotting for censored failure data,” *Technometrics*, vol. 14, no. 4, pp. 945–966, 1972.

[22] O. Aalen, “Nonparametric inference for a family of counting processes,” *The Annals of Statistics*, pp. 701–726, 1978.

[23] N. Breslow, “Covariance analysis of censored survival data,” *Biometrics*, pp. 89–99, 1974.

[24] A. A. Tsiatis, “A large sample study of cox’s regression model,” *The Annals of Statistics*, vol. 9, no. 1, p. 93–108, 1981.

[25] F. Xia, J. Ning, and X. Huang, “Empirical comparison of the breslow estimator and the kalbfleisch prentice estimator for survival functions,” *Journal of Biometrics & Biostatistics*, vol. 9, no. 2, 2018.

[26] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[27] F. Rosenblatt, “The perceptron: a probabilistic model for information storage and organization in the brain.,” *Psychological Review*, vol. 65, no. 6, p. 386, 1958.

[28] A. LeNail, “Nn-svg: Publication-ready neural network architecture schematics,” *Journal of Open Source Software*, vol. 4, no. 33, p. 747, 2019.

[29] M. R. Meireles, P. E. Almeida, and M. G. Simões, “A comprehensive review for industrial applicability of artificial neural networks,” *IEEE Transactions on Industrial Electronics*, vol. 50, no. 3, pp. 585–601, 2003.

[30] S. Ruder, “An overview of gradient descent optimization algorithms,” *arXiv preprint arXiv:1609.04747*, 2016.

[31] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[32] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[33] A. Senior, G. Heigold, M. Ranzato, and K. Yang, "An empirical study of learning rates in deep neural networks for speech recognition," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 6724–6728, IEEE, 2013.

[34] S. Dieleman, J. Schlüter, C. Raffel, E. Olson, S. K. Sønderby, D. Nouri, *et al.*, "Lasagne: First release.," Aug. 2015.

[35] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv e-prints*, vol. abs/1605.02688, May 2016.

[36] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural Networks*, vol. 12, no. 1, pp. 145–151, 1999.

[37] Y. Nesterov, "Gradient methods for minimizing composite functions," *Mathematical Programming*, vol. 140, no. 1, pp. 125–161, 2013.

[38] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization.," *Journal of Machine Learning Research*, vol. 12, no. 7, 2011.

[39] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural Networks for Machine Learning*, vol. 4, no. 2, pp. 26–31, 2012.

[40] M. D. Zeiler, "Adadelta: An adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[41] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

APPENDIX A

SOURCE CODE

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import brier_score_loss
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import deep_surv
from deepsurv_logger import DeepSurvLogger, TensorboardLogger
import utils
import viz
import lasagne

# Define helper functions for baseline estimation using Breslow's method.
def baseline_hazard(label_e, label_t, par_haz):
    ind_df = pd.DataFrame({"E": label_e, "T": label_t, "h": par_haz})
    # group rows by distinct event times(T)
    summed_over_durations = ind_df.groupby("T")[["h", "E"]].sum()
    summed_over_durations["h"] = summed_over_durations["h"].loc[::-1].cumsum()
    # Breslow's estimation of baseline hazard
    base_haz = pd.DataFrame(
        summed_over_durations["E"] / summed_over_durations["h"], columns=["
    base_haz"]
    )
    return base_haz

def baseline_cumulative_hazard(label_e, label_t, par_haz):
    return baseline_hazard(label_e, label_t, par_haz).cumsum()

def baseline_survival_function(label_e, label_t, par_haz):
    base_cum_haz = baseline_cumulative_hazard(label_e, label_t, par_haz)
    return np.exp(-base_cum_haz)

# Define helper function to convert pandas dataframe to deepsurv input.
def dataframe_to_deepsurv_ds(df, event_col = 'Event', time_col = 'Time'):
    # Extract the event and time columns as numpy arrays
    e = df[event_col].values.astype(np.int32)
    t = df[time_col].values.astype(np.float32)

    # Extract the patient's covariates as a numpy array
    x_df = df.drop([event_col, time_col], axis = 1)
    x = x_df.values.astype(np.float32)

    # Return the deepsurv dataframe
    return {
        'x' : x,
        'e' : e,
        't' : t
    }
# Read the data of machine2
df_m2 = pd.read_csv('final_df_m2.csv')
# Standardize the input parameters
scaler = MinMaxScaler(feature_range=(0,1))
normalized_df = scaler.fit_transform(df_m2)
df_normalized = pd.DataFrame(data=normalized_df, columns=df_m2.columns)
df_normalized['time_2'] = df_m2.time_2.values
df_normalized['status_2'] = df_m2.status_2.values
df_m2 = df_normalized
# Define a new column to store the index values
index = [i for i in range(1, df_m2.shape[0]+1)]
```

```
df_m2['index'] = index
# Split the dataset into training, validation, and test data
df_train, df_test = train_test_split(df_m2, test_size = 0.15, random_state=99,
     shuffle=True)
df_train, df_valid = train_test_split(df_train, test_size = 0.18, random_state
    =99, shuffle=True)
# Store the index values of respective datasets separately and drop the index
    column
train_index = df_train['index']
valid_index = df_valid['index']
test_index = df_test['index']
df_train = df_train.drop('index', axis=1)
df_valid = df_valid.drop('index', axis=1)
df_test = df_test.drop('index', axis=1)
df_m2 = df_m2.drop('index', axis=1)
# Convert the pandas dataframe to deepsurv input
df_m2_ds = dataframe_to_deepsurv_ds(df_m2,
                                     event_col = 'status_2',
                                     time_col= 'time_2')
train_data_ds = dataframe_to_deepsurv_ds(df_train,
                                          event_col = 'status_2',
                                          time_col= 'time_2')
valid_data_ds = dataframe_to_deepsurv_ds(df_valid,
                                          event_col = 'status_2',
                                          time_col= 'time_2')
test_data_ds = dataframe_to_deepsurv_ds(df_test,
                                          event_col = 'status_2',
                                          time_col = 'time_2')
# Define the optimal hyperparameters of the model
hyperparams = {
    'L2_reg': 15.0,
    'batch_norm': True,
    'dropout': 0.1,
    'hidden_layers_sizes': [40, 40],
    'learning_rate': 1e-02,
    'lr_decay': 0.025,
    'momentum': 1.0,
    'n_in': train_data_ds['x'].shape[1],
    'standardize': False
}
# Instantiate the deepsurv model instance
model = deep_surv.DeepSurv(**hyperparams)
experiment_name = 'machine2_data'
logdir = './logs/tensorboard/'
logger = TensorboardLogger(experiment_name, logdir=logdir)
update_fn=lasagne.updates.amsgrad
n_epochs = 2000
# Train the deepsurv model
metrics = model.train(train_data_ds,
                      valid_data_ds,
                      n_epochs=n_epochs,
                      logger=logger,
                      update_fn=update_fn,
                      validation_frequency = 250)
# Print the final metrics
print('\n Train C-Index:', metrics['c-index'][-1])
print('\n Valid C-Index:', metrics['valid_c-index'][-1])
# Plot the training / validation curves
viz.plot_log(metrics)
# Print the concordance index on the dataset
print("C-index on training data: {}".format(float(model.get_concordance_index
    (**train_data_ds))))
print("C-index on validation data: {}".format(float(model.
    get_concordance_index(**valid_data_ds))))
print("C-index on testing data: {}".format(float(model.get_concordance_index
    (**test_data_ds))))
```

```
print("C-index on the whole data: {}".format(float(model.get_concordance_index
    (**df_m2_ds))))
# Predict the log partial hazard and save the predictions to a .csv file
np.savetxt("risk_pred_test_m2.csv",
           model.predict_risk(test_data_ds['x']),
           delimiter=",")
np.savetxt("risk_pred_valid_m2.csv",
           model.predict_risk(valid_data_ds['x']),
           delimiter=",")
np.savetxt("risk_pred_train_m2.csv",
           model.predict_risk(train_data_ds['x']),
           delimiter=",")
np.savetxt("risk_pred_df_m2.csv",
           model.predict_risk(df_m2_ds['x']),
           delimiter=",")
# Load the risk predictions
risk_train = pd.read_csv('risk_pred_train_m2.csv', header=None)
risk_valid = pd.read_csv('risk_pred_valid_m2.csv', header=None)
risk_test = pd.read_csv('risk_pred_test_m2.csv', header=None)
risk_df = pd.read_csv('risk_pred_df_m2.csv', header=None)
# Append the log partial hazard and partial hazard columns to the dataset
df_train['log_haz'] = risk_train[0].values
df_train['par_haz'] = np.exp(df_train.log_haz)
df_train['index'] = train_index

df_valid['log_haz'] = risk_valid[0].values
df_valid['par_haz'] = np.exp(df_valid.log_haz)
df_valid['index'] = valid_index

df_test['log_haz'] = risk_test[0].values
df_test['par_haz'] = np.exp(df_test.log_haz)
df_test['index'] = test_index

```

```python
df_m2['log_haz'] = risk_df[0].values
df_m2['par_haz'] = np.exp(df_m2.log_haz)
# Save the updated dataframes to a .csv file
df_train.to_csv('train_m2.csv', index=False, header=True)
df_valid.to_csv('valid_m2.csv', index=False, header=True)
df_test.to_csv('test_m2.csv', index=False, header=True)
df_m2.to_csv('df_m2.csv', index=False, header=True)
# Estimate the baseline hazard estimates of the data using Breslow's method
base_df = pd.DataFrame()
base_df['time'] = df_m2.time_2.sort_values().unique()
base_df = base_df.set_index('time', drop = False)
base_df['base_haz'] = baseline_hazard(df_m2.status_2,
                                      df_m2.time_2,
                                      df_m2.par_haz)
base_df['base_cum_haz'] = baseline_cumulative_hazard(df_m2.status_2,
                                                     df_m2.time_2,
                                                     df_m2.par_haz)
base_df['base_surv'] = baseline_survival_function(df_m2.status_2,
                                                  df_m2.time_2,
                                                  df_m2.par_haz)
haz = dict(zip(base_df.time, base_df.base_haz))
cum_haz = dict(zip(base_df.time, base_df.base_cum_haz))
surv = dict(zip(base_df.time, base_df.base_surv))
df_m2['base_haz'] = df_m2['time_2'].map(haz)
df_m2['cum_base_haz'] = df_m2['time_2'].map(cum_haz)
df_m2['base_surv'] = df_m2['time_2'].map(surv)
df_m2['haz'] = df_m2.base_haz * df_m2.par_haz
df_m2['surv'] = df_m2.base_surv ** df_m2.par_haz
# Calculate the Brier score of the model
brier_score = brier_score_loss(df_m2[df_m2.time_2!=1800].status_2,
                               df_m2[df_m2.time_2!=1800].surv,
                               pos_label=1)
round(brier_score, 3)
```

```python
# Prepare the data to plot hazardogram
survp_m2 = pd.DataFrame()
survp_m2['time'] = df_m2.iloc[:10000].time_2
survp_m2['par_haz'] = df_m2.iloc[:10000].par_haz
for i in survp_m2.time.sort_values().unique():
    survp_m2[i] = base_df.base_surv[i] ** survp_m2.par_haz
actual_time_m2 = survp_m2.time.values
survp_time_m2 = survp_m2.time.sort_values().unique()
survp_m2 = survp_m2.drop(['time', 'par_haz'], axis=1)
# Save the data for hazardogram in .csv files
survp_m2.to_csv('survp_m2.csv',
                index=False,
                header=False)
pd.Series(survp_time_m2).to_csv('survp_time_m2.csv',
                                index=False,
                                header=False)
pd.Series(actual_time_m2).to_csv('survp_actualtime_m2.csv',
                                 index=False,
                                 header=False)
# Plot the hazardogram
survp_arr = survp.T.values
color_map = plt.cm.get_cmap('jet')
rev_color_map = color_map.reversed()
fig, ax = plt.subplots(figsize=(16,10))
g = sns.heatmap(survp_arr, cmap = rev_color_map)
g.set_ylabel('Prediction Horizon')
g.set_xlabel('Episode Time')
y_count = np.array(range(0, len(survp.columns.to_list()), 20))
g.set_yticks(y_count)
g.set_yticklabels([survp.columns[i] for i in y_count])
x_count = np.array(range(0, survp.shape[0], 20))
g.set_xticks(x_count)
g.set_xticklabels(['{:d}'.format(i+1) for i in x_count])
```

```
g.invert_yaxis()
plt.scatter(range(survp.shape[0]),actual_time,  color='k')
plt.show()
```

APPENDIX B

HAZARDOGRAM PLOTS OF TRAINING AND TEST DATA

The hazardogram plots of training and test data of five different machines are plotted to measure the model's efficacy and check for any overfitting cases.

In all the cases described below, the model's predictability is checked for higher risk values and lower risk values, i.e., the edge cases. To achieve this, the training data and test data are sorted according to the estimated log partial hazard, and the top 100 and the bottom 20 records are considered for plotting the hazardogram.

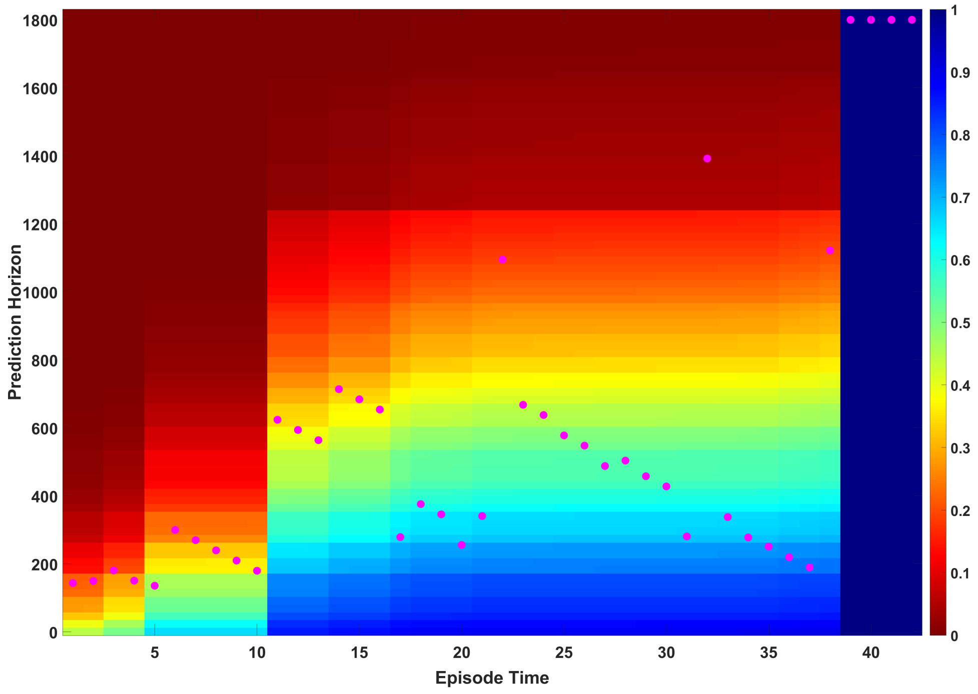


Figure B.1: Hazardogram: Training data of machine 1

Figure B.1 and Figure B.2 show the hazardogram plots of the training data and test data of machine 1, respectively. The integrated Brier score of Deepsurv model at all available event times of machine 1 data shown in Figure 4.7 indicates the model's ability to classify the events reasonably well. This is reflected in Figure B.1, where most of the episodes with an event in the time range of 150 - 400 are correctly flagged based on the survival probability estimates. Additionally, the episodes with low values of hazard tend to have a survival probability greater than 0.9. In a nutshell, the Deepsurv model is able to flag the impending breakdowns reasonably well on the training data of machine 1. The hazardogram on test data of machine 1 shown in Figure B.2 explains the relatively poor ability to flag breakdowns for prediction horizons of over 1200 sec. This is because the Brier score of the Deepsurv model on machine 1 tends to increase as the prediction horizon reaches the censoring time as observed in Figure 4.6. Apart from those episodes with higher prediction horizons, the Deepsurv model on machine 1 test data performed well on the records that fall in the mid-range of the prediction horizon.

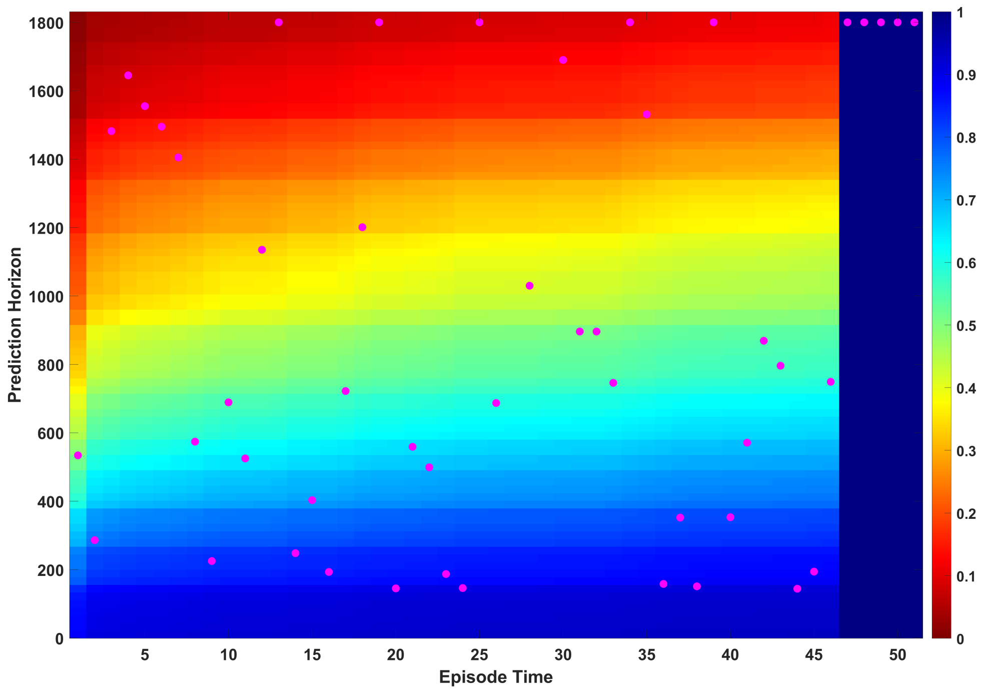


Figure B.2: Hazardogram: Test data of machine 1

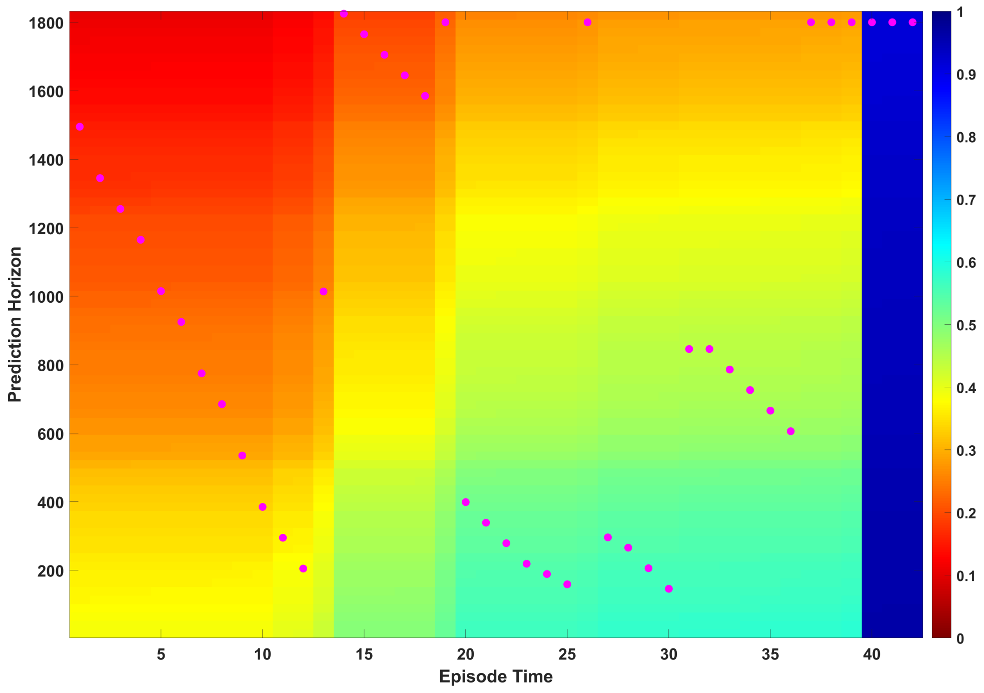


Figure B.3: Hazardogram: Training data of machine 2

Figure B.3 and Figure B.4 explain the hazardogram plots on machine 2 training data and test data, respectively. The relatively low concordance index of the Deepsurv model on machine 2 data shown in Figure 4.8 explains the imperfect ability to order the event times based on log hazard predictions. This is also seen in the case of training data where the episodes that have a higher prediction horizon tend to have a higher risk at the beginning of Figure B.3. This is also why some of the episodes in the range 15 - 30 with a high prediction horizon have a higher risk of breakdown. Simultaneously, the model can flag the events perfectly well that have a prediction horizon in the range 400 - 1000. As seen on the test data in Figure B.4, the episodes that fall in this range are flagged early in their fault propagation, and the least risky episodes tend to have a survival probability of greater than 0.9.

Figure B.5 and Figure B.6 explain the hazardogram plots on machine 3 training and test data, respectively. The Deepsurv model predictions on machine 3 data show similarities with machine 2

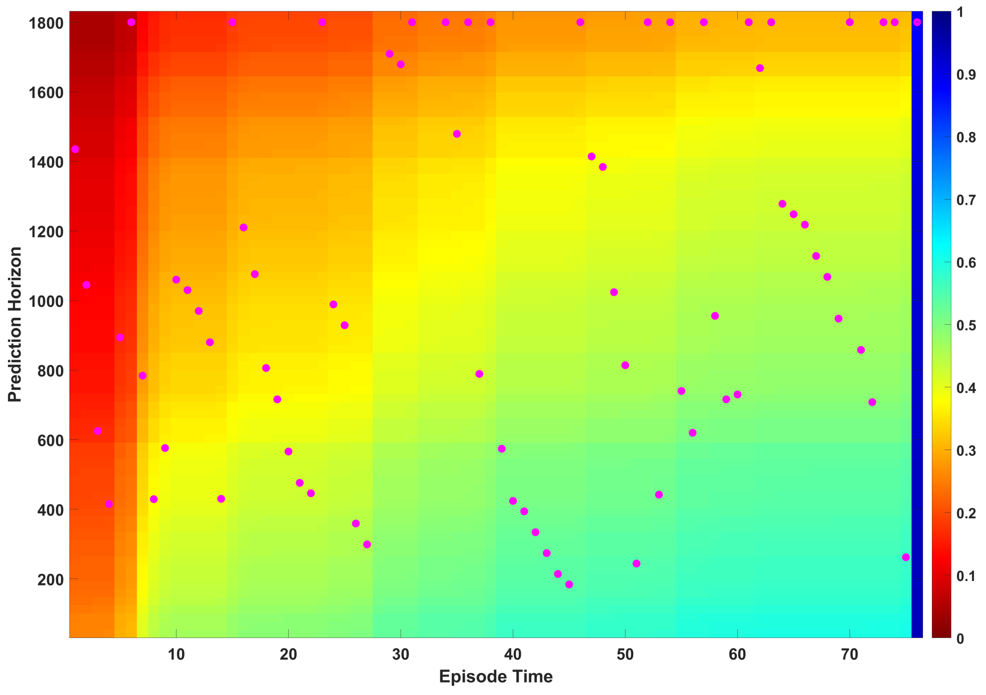


Figure B.4: Hazardogram: Test data of machine 2

data, i.e., most mid-range prediction horizon episodes are flagged reasonably well with a threshold set at 0.5. However, the Deepsurv model trained on machine 3 data tends to be inaccurate for episodes with a higher prediction horizon. This could be explained with the IBS score shown in Figure 4.7. Also, the model's Brier score tends to increase with a greater prediction horizon, as shown in Figure 4.6, which is in line with the poor behavior of the model observed in both Figure B.5 and Figure B.6. Similar to other machine cases explained above, the model estimates the survival probability greater than 0.9 for those episodes that have low-risk values.

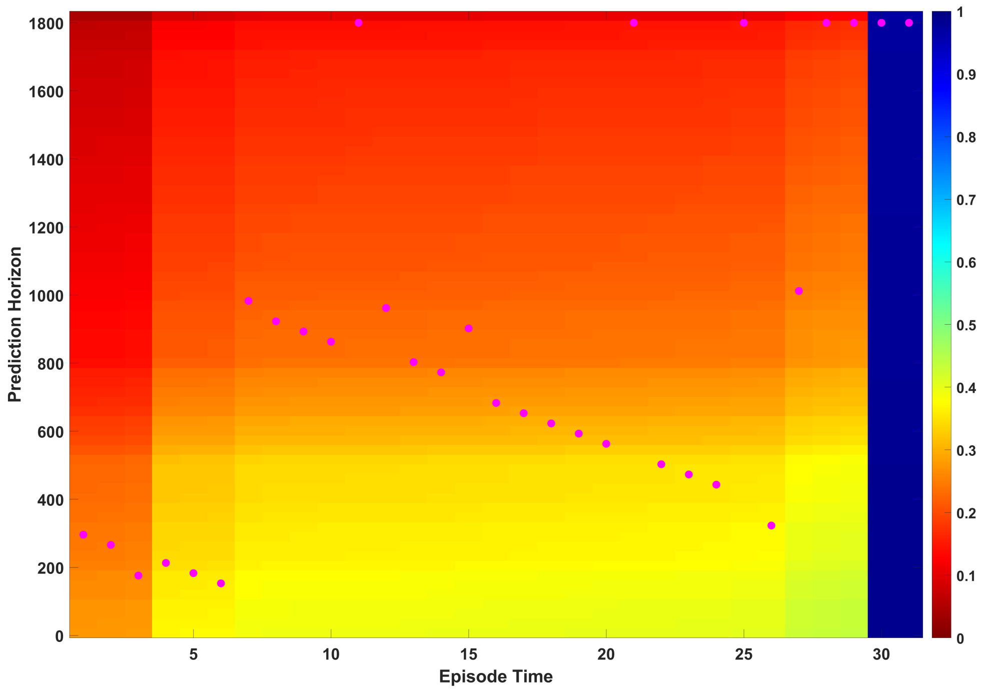


Figure B.5: Hazardogram: Training data of machine 3

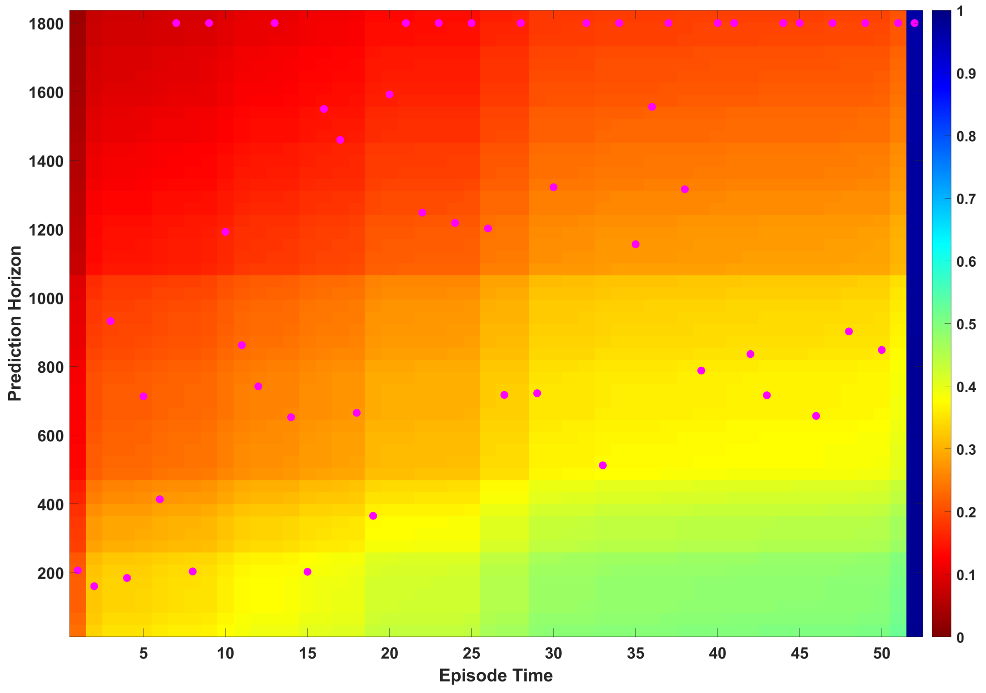


Figure B.6: Hazardogram: Test data of machine 3

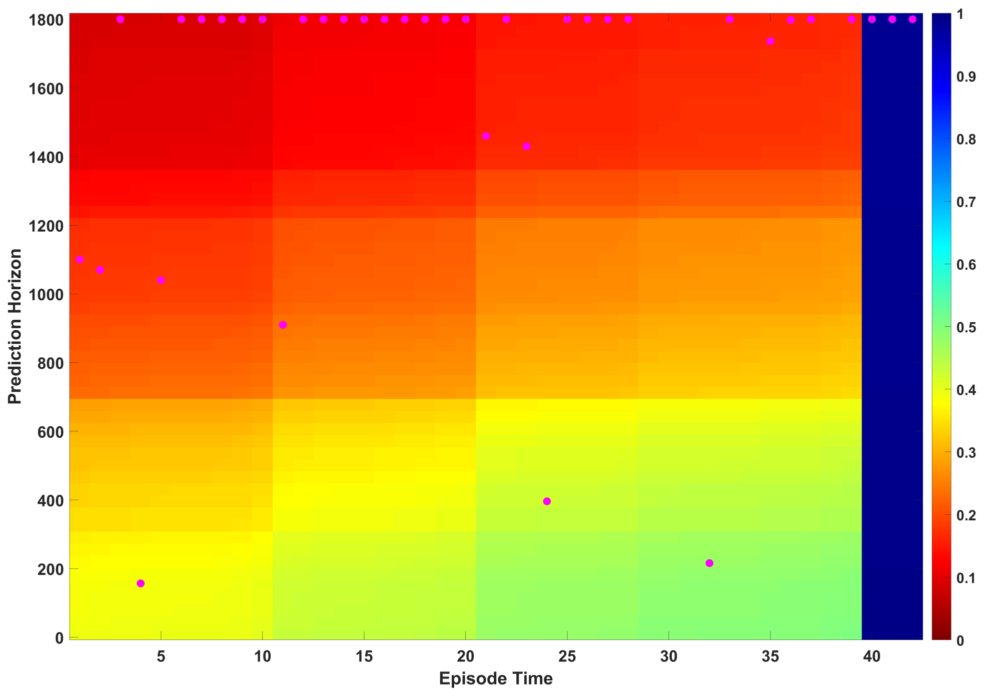


Figure B.7: Hazardogram: Training data of machine 4

Figure B.7 and Figure B.8 explain the hazardogram plots on machine 4 training and test data, respectively. The integrated Brier score of the Deepsurv model on machine 4 is the highest among all the machines, and this is reflected in the survival probability estimates at different episode times in Figure B.7 and Figure B.8. Also, the high Brier score at distinct event times, as shown in Figure 4.6, explains why some of the episodes with a prediction horizon greater than 1000 are incorrectly flagged. Also, both the training and the test data contain most episodes with a prediction horizon of 1800 sec, and the model is observed to generalize very poorly in such a case. In fact, an accurate model should classify episodes with a prediction horizon near 1800sec with a survival probability of greater than 0.8. This behavior is not observed in the Deepsurv model trained on machine 4 data, even though the trained model has a concordance index of 0.8.

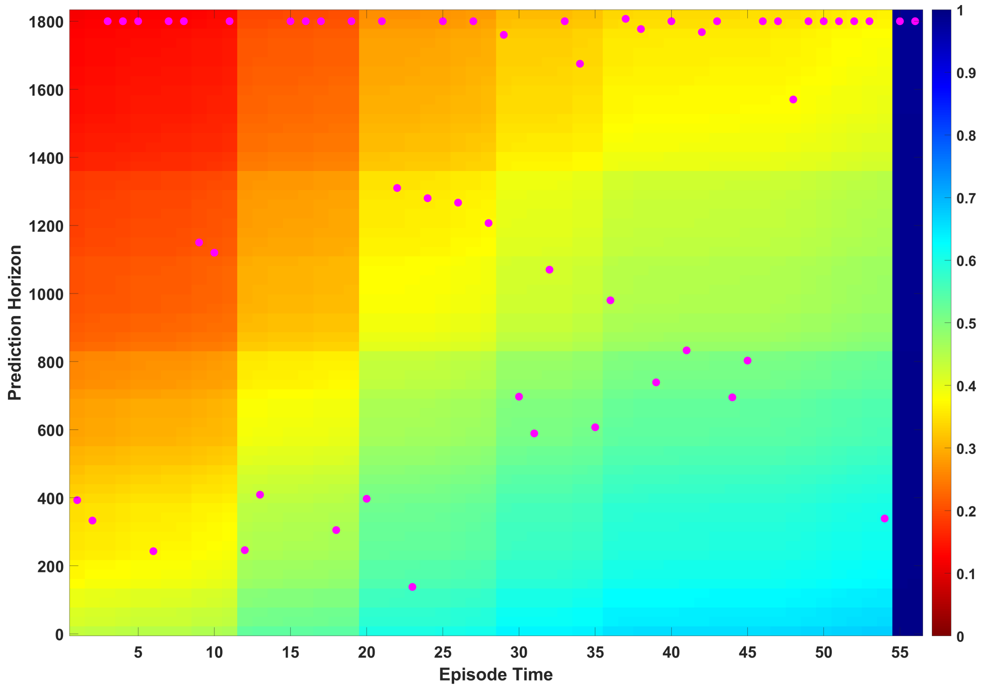


Figure B.8: Hazardogram: Test data of machine 4

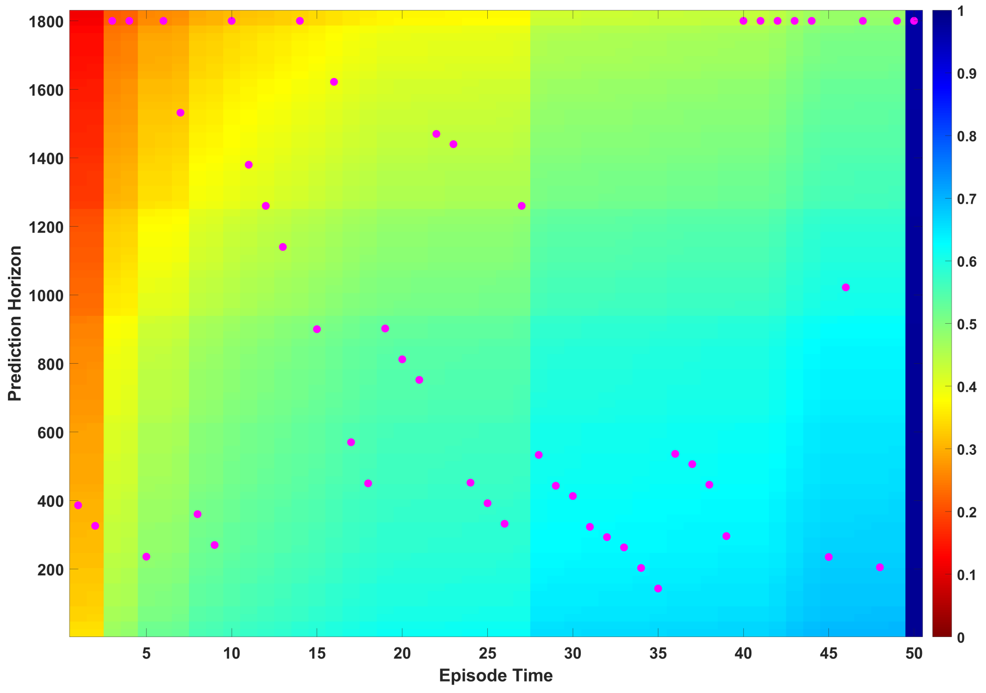


Figure B.9: Hazardogram: Training data of machine 5

Figure B.9 and Figure B.10 show the hazardogram plots of machine 5 training and test data, respectively. The highest concordance index on the data is achieved from the model trained on machine 5. However, the model's ability to correctly order the event times is not observed in the test data hazardogram plot. Also, the relatively high integrated Brier score shown in Figure 4.7 suggests that the model is not able to classify the episode times with a prediction horizon of above 1600 sec. This is a possible case where the model is overfitting on the training data, and the poor performance of the model on test data suffices the statement.

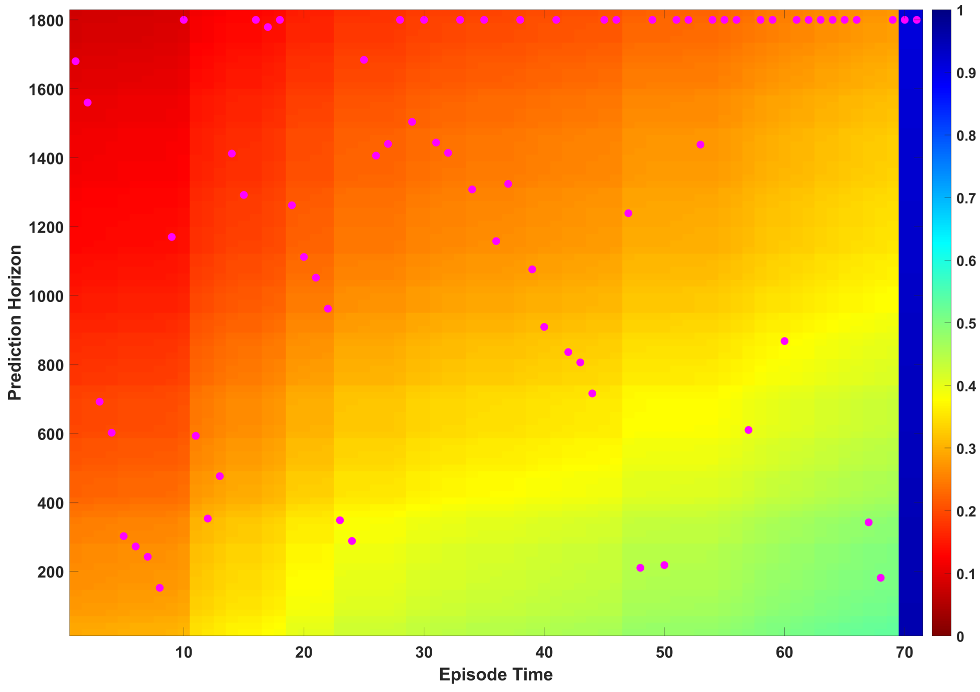


Figure B.10: Hazardogram: Test data of machine 5

Breslow's method of baseline estimation in this analysis does not take tied event times into account. Also, the Deepsurv model is built with the assumption of no ties in the data. In summary, the results from the hazardogram would have come out as expected when the tied event times are accounted for in the analysis.