Håkon Andersen Holte

Survival Analysis with Deep Recurrent Neural Network Model for Prediction of Credit Card Defaults

Master's thesis in Applied Physics and Mathematics
Supervisor: John Sølve Tyssedal
June 2021

Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

Håkon Andersen Holte

Survival Analysis with Deep Recurrent Neural Network Model for Prediction of Credit Card Defaults

Master's thesis in Applied Physics and Mathematics
Supervisor: John Sølve Tyssedal
June 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

NTNU
Norwegian University of
Science and Technology

Preface

This thesis was written in the spring of 2021 and is the final assignment of my Masters of Science degree at the Norwegian University of Science and Technology (NTNU). It is thus the concluding work of my specialization in industrial mathematics, which has made up the final three years of the five year study program Applied Physics and Mathematics. Specifically, my specialization is in statistics and machine learning, which are both central concepts in this thesis. The thesis was written in cooperation with SpareBank 1 Kreditt, and submitted to the Department of Mathematical Sciences in June of 2021 for evaluation. For anyone interested in reading the thesis, a background in mathematics is a great advantage, and some knowledge of statistics, machine learning and deep learning is also helpful. To produce the results presented in this thesis, the programming language Python was used, and the code can be accessed on GitHub via the link `https://github.com/HaakonHolte/Masters_Thesis_Code`. Any of the code available here is free to use. Be aware, however, that the code is written with the specific dataset used in the thesis in mind, and this will not be made available due to confidentiality.

If you do not care about what I'd classify as my "personal statement" regarding this thesis, you should skip this paragraph and head straight to the acknowledgements below. I was first introduced to the realm of credit scoring by SpareBank 1 Kreditt in 2020, when I interned there. When presented with results from the work of Gunnarson et al. (2019), and the claim that deep learning did not provide much value beyond other methods in credit scoring, I remember saying "Well then they just haven't found the right neural network yet". A statement which I, admittedly, did not have the expertise nor experience to make. Although this thesis was mainly an experiment to see how a full longitudinal model would perform in credit scoring, I did have a small hope that I would prove myself right through it. Unfortunately, as you will learn from the abstract, this thesis did not introduce the model that would revolutionize the credit scoring world. Although this did not come as a surprise (I do not in any way fancy myself an expert on deep learning or survival analysis), it is always a bit dismaying when the outcome is not as splendid as one had a shimmer of hope for, regardless of how optimistic this may have been. Despite this, I stand by my choice of method, and I am very thankful to both NTNU and SpareBank 1 Kreditt for getting to work within such an interesting and exciting field. I truly feel that I have learned a lot, and gained invaluable experience working with models that I simply find to be extremely cool. And hopefully, one day, someone will come along and prove me right on my initial sentiment regarding deep learning and credit scoring.

I would like to acknowledge several people who have helped and supported me throughout my work with this thesis. First and foremost I would like to thank my supervisor at the Department of Mathematical Sciences, John Sølve Tyssedal, for his unwavering support, patience and academic input, as well as for many good conversations about all sorts of topics including statistics, pandemics and punch cards. I would also like to extend my gratitude to Jens Morten Nilsen, who was my supervisor at SpareBank 1 Kreditt, for his invaluable domain knowledge and readiness to help and answer questions. I would further like to thank SpareBank 1 Kreditt for letting me use their data. I also feel like I would be remiss if I did not pay some homage to the many brilliant scientists whose shoulders I have stood on while working on this thesis, and whose works I refer to throughout it. Finally, I would like to thank my family (danke sehr, Jódl) and friends for their moral support, and my girlfriend Jenny for her admirable patience when listening to the ramblings of an angry mathematician over why his validation error doesn't decrease. You all play vital parts in why this thesis exists.

Trondheim, 20.06.2021
Håkon Andersen Holte

Abstract

In this thesis, a deep neural network model combined with concepts from survival analysis is applied to sequential credit card data with the purpose of predicting time to default for the credit card customers. Personal financial problems, societal impact and loss of revenue for credit card institutions are all factors that motivate the exploration of machine learning in the field of credit scoring. The thesis was written in cooperation with SpareBank 1 Kreditt, who provided the dataset used in the thesis. This consisted of around 11300 credit card accounts, with daily observations on transaction data. An exploratory data analysis was performed in order to get familiarized with the dataset. A discrete time model was introduced, and a neural network with an LSTM structure was used to obtain predictions about the time to default of the credit card customers. The results obtained were mixed, with the predictions themselves being quite inaccurate, but reasonable according to evaluation criteria currently used by SpareBank 1 Kreditt. The performance on the models was on par with models of lower complexity that are currently being used. Several simplifications were made and several stones left unturned throughout the thesis, meaning there are several topics one could consider worthy of further research. In particular, the way in which the predictions are obtained include an unknown functional relationship which should be more thoroughly explored.

Sammendrag

I denne oppgaven brukes en dyp nevral nettverksmodell kombinert med konsepter fra levetidsanalyse på sekvensielle kredittkortdata med det formål å predikere tid til mislighold for kredittkortkunder. Personlige økonomiske problemer, samfunnsmessige konsekvenser og tap av inntekter for kredittkortinstitusjoner er alle faktorer som motiverer utforskningen av maskinlæring innen kreditt-scoring. Oppgaven ble skrevet i samarbeid med SpareBank 1 Kreditt, som sto for datasettet som ble brukt i oppgaven. Dette besto av rundt 11300 kredittkortkontoer, med daglige observasjoner av transaksjonsdata. En dataanalyse ble utført for å bli kjent med datasettet. En diskret tidsmodell ble introdusert, og et nevralt nettverk med en LSTM-struktur ble brukt for å oppnå prediksjoner om kredittkortkundenes tid til mislighold. Resultatene som ble oppnådd var blandede, og prediksjonene i seg selv var ganske unøyaktige, men rimelige i henhold til evalueringskriterier som for tiden brukes av SpareBank 1 Kreditt. Modellenes prestasjoner var på nivå med modeller med lavere kompleksitet som for tiden er i bruk. Flere forenklinger ble gjort gjennom oppgaven, noe som betyr at det er flere områder som kan være verdt å undersøke videre. Disse inkluderer spesielt et ukjent funksjonalt fohold som er sentralt i måten prediksjonene oppnås på.

Contents

Figures

Tables

Chapter 1

Introduction

Credit cards are a very popular means of payment in Norway. In July 2020, SpareBank 1 SR-Bank reported that 3.2 million Norwegians have credit cards, consumer loans or other unsecured debt (SpareBank 1 SR-Bank, 2020). This is based on data from the Norwegian debt register, Gjeldsregisteret. The number has decreased somewhat since then, and is now around 3.18 million (Gjeldsregisteret, 2020). The many advantages and the freedom that often come with paying with a credit card explain why the payment method is so widely used. However, these advantages do not come without a risk. As mentioned, more than 3 million Norwegians have unsecured debt, and 2/3 of the people who have gotten a consumer loan at some point in their life, still have consumer loans. Furthermore, 1/3 of people with consumer loans are struggling to pay down their loans (Gemini, 2020). According to E24, 250 000 Norwegians above the age of 18 have payment remarks due to late payments (E24, 2020). The damage of such personal monetary problems are felt both personally by the people experiencing them, by society and by the credit institutions. Machine learning presents itself as a possible aid in reducing the significance and number of occurrences of these problems. As we are (hopefully) emerging from the Covid-19 situation, we are seeing an increase in the use of credit cards (Finansavisen, 2021), as well as, very recently, an increase in consumer debt (NRK, 2021). This comes after a time of around a year where both the use of credit cards and the consumer debt has been steadily decreasing (Finans Norge, 2020). This development further motivates the study of potential machine learning algorithms for use in credit scoring.

This section begins by introducing the basics of how credit cards work, before moving on to presenting the concept of credit card defaults. Emphasis will be on the definition of default as used by the credit card company SpareBank 1 Kreditt, whose data are used for training the models suggested in this thesis. Following this is an outline of the problem at hand, as well as a short overview and discussion of the history of machine learning within the field of credit scoring.

1.1 Credit Cards

A credit card is a card issued by financial institutions that allows its owner to make purchases, cash withdrawals and bank transactions with money they lend from the institution. Both parties enter into an agreement, in which the credit card recipient agrees to pay back the money he or she lends from the financial institution. The financial institution, on the other hand, promises to lend the customer money up to a certain amount, known as the credit card limit, and often provides the customer with access to benefits through insurances or bonus programs. The financial institutions make money on the credit card business through a range of fees, as well as from interest on the money that their customers lend and do not pay back in full by a given due date.

When deciding whether or not to grant a potential customer a credit card, financial institutions face a decision which can often be difficult. On the one hand, not being able to pay ones credit card debt does not only materialize in great personal burden, but also in financial losses for credit institutions. On the other hand, rejecting a customer from receiving a credit card could result in the loss of potential revenue. Therefore it is vital for these institutions to constantly make good and informed decisions, not only on which individuals to grant credit cards, but also on what credit limit to set, and what other measures to take in order to ensure the continued, but not exaggerated, credit card use of their customers. It is in making these informed decisions that credit scoring models come into play, and we will have a look at examples of these and how they have previously been applied. Before doing this, however, an introduction of the concept of credit card defaults is merited.

1.2 Credit card Defaults

A credit card default occurs when a credit card owner has become severely delinquent on his or her credit card payments (The Balance, 2020). The term is often used by banks, credit card companies and other financial institutions to describe the event in which a customer is unable to repay his or her debt in a certain amount of time. This definition as well as the amount of time until a customer is considered to have defaulted, varies between the many relevant institutions. One common definition is the one used in the Basel II regulatory framework: a credit card account is considered defaulted if it is 180 days past due or has been partially or fully charged off (Qi, 2009). As this thesis is written in cooperation with SpareBank 1 Kreditt, their definition of default will be used throughout. It states that any customer that fails to repay a minimum amount on his or her debt by day 120 after the due date, is considered to have defaulted.

In Figure 1.1 is shown the process of a customer going from healthy to severely delinquent. Key events that occur during the process are indicated by marks along the time axis. Each mark is accompanied by the name of the event in question, along with the number of days after due date the event occurs. On statement date the customer receives a billing statement, which states the total amount that the customer owes the company, and the due date for payment. If the customer does not manage to pay the minimum amount before due date, it will receive a statement date dunning on the next statement date. If the amount is still not paid by the next due date, a due date dunning is ushered. Failure to pay this dunning before it is due, results in a collection advice and accompanying due date. If the customer is still unable or unwilling to pay, the account goes to debt collection. Should it remain in debt collection for 60 days, the customer is declared delinquent, and the account is said to have defaulted.

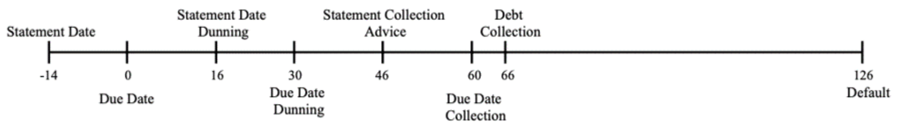

Figure 1.1: Visualization of the process a customer goes through from statement date until default.

1.3 Problem Outline

As defaults incur a significant loss of income for credit card companies, it is desirable to develop statistical models able to quantify the likelihood that a customer will experience a default. This quantity is often known as a *credit score*. Such models are beneficial to credit card companies in that they allow them to control the risk in their portfolio through measures aimed at customers who seem to be at risk of defaulting. Furthermore, such models allow the institutions to tailor their counselling to fit the customer's needs, and keep them in the loop about how they seem to be doing financially. Thus, these models do not only benefit the credit institutions, but the customers as well.

One obvious, and quite common, way to obtain credit scores is to directly predict the probability of default, often abbreviated 'PD', for customers. This can be accomplished through the use of machine learning methods, for instance regression trees, logistic regression or random forests. The last two are quite popular, and ensemble methods such as random forests are known to perform quite well for this task (Gunnarson et al., 2019). However, as was pointed out in Banasik et al. (1999), one could argue that knowing how long time will pass before a customer defaults, might be more valuable than knowing the probability that a customer will default at a certain point in time. This inspires the approach taken in this thesis: given a customer and customer data such as their age, gender and expenditure, predict the number of days until this customer defaults. This problem is then known from the domain of statistical survival analysis. A plethora of methods, both parametric, semi-parametric and non-parametric are then available, many of which have been applied to the specific problem of credit card defaults on real-life datasets (Dirick et al., 2016).

Before proceeding with the thesis and looking at the dataset to be used, some history and motivation for the use of survival analysis in the context of credit risk modelling is warranted, as it is already well documented. One early example of its use is Nahrain (1992), where the idea of using survival analysis in credit card scoring was introduced, and where AFT models were used. Some years later, Banasik et al. (1999) asked the question of *when*, rather than *if*, a customer will default, and applied the Cox proportional hazard method to answer this question, arguing that the results obtained were competitive with that of the widely used logistic regression. Among more recent works are Zhang and Thomas (2012), where linear regression and survival analysis are compared in predicting loss given default (LGD), and Dirick et al. (2014), where mixture cure models are applied to credit loan data.

The approach to the problem of predicting time to default in this thesis will be through the use of deep learning, and more precisely, through recurrent neural networks (RNNs). Neural networks have been very popular over the last decades, showing excellent predictive power in cases where large amounts of data are available. RNNs are well-suited for the problem at hand because they are able to handle sequential data very well. Their construction allows them to save information from former iterations and consider this information when working with the next input. In this thesis, a very popular type of RNNs, called long-short term memory units (LSTMs), will be used. The reason for this will be more closely explained in the theory chapter, but to make a long story short, they are able to handle information loss over time better than the standard RNNs, and they also handle other problems which often arise in RNNs.

Chapter 2

Dataset

The dataset being used in this thesis is provided by SpareBank 1 Kreditt, and consists of profile and transaction data for several of the company's customers. It contains daily observations on 70 variables for each account, with around 11300 unique accounts in the dataset. There are no observations registered during weekends. The observations are done over a period of approximately 26 months, starting in July of 2018. As default events are relatively rare in the credit card business, the dataset is significantly imbalanced, with around 14% of the customers experiencing a default event during the time period in question. The customers are not chosen entirely at random; SpareBank 1 Kreditt calculates an initial score for their customers, known as an application score. A correlation between low application score and higher default rate has been observed, and thus the dataset is designed to contain customers with a low application score. In the first part of this chapter, the most important variables of the dataset are presented and discussed. In the last part, consequences of the imbalance in the dataset are addressed.

2.1 Variables and Response

As mentioned, there are 70 variables present in the data set. The most important variables and variable types are outlined here, whereas a full list of the variables is provided in Table A.1 in Appendix A. This table also indicates what variables are used in the data analysis and in the models trained in this thesis.

There are mainly two variables that are of interest as response variables in the dataset: 'DC2Ind' and 'RemaningLifetime'. The variable 'DC2Ind' is an indicator variable stating whether the customer in question defaults during the experiment, and is equal to 1 if this is the case, and 0 else. 'RemaningLifetime' states, for accounts with DC2Ind=1, the number of days remaining until default occurs. For accounts with DC2Ind=0, it simply states the remaining days until censoring or the end of the experiment. For the task of simple binary classification or estimation of probability of default, the variable 'DC2Ind' is a clear choice for response variable, typically to enter into some sort of cross-entropy loss function, which will be discussed later in the thesis. However, when trying to predict the time until default occurs, it is useful to include the ’RemaningLifetime’ variable, and somehow incorporate this into a loss function which is suited for this type of predictions. As we will see, this is a bit more of an intricate matter, partly due to the variable only providing complete information for the accounts that actually do default; for the ones that do not, we only know that they had not yet defaulted by the time the experiment was ended.

The remaining variables will be used as potential covariates in the models that are developed. They can be split into two main categories: customer profile information and customer transaction data. The first category includes variables such as the customer’s unique account number, the age of the customer, how many months since the account was created and what credit card product the customer uses. There are 15 variables in this category, 11 of them categorical and 4 of them numerical. Several of these will not be included in the models or data analysis, as they are non-informative in the sense that they do not contain any information about the customers’ behaviour. The account number and the date at which the account was created are examples of such variables.

The second category includes variables such as account balance, amount that is overdue, number of cash withdrawals and spending on different types of products and services. This category includes 53 variables, 4 of which are categorical, the rest are numerical. Some of the variables in this category are cumulative, and there is some variation as to how these behave. For instance, the variable ’BALANCE_AMT’ will remain constant until the customer makes use of their credit card, i.e. makes a purchase, transaction or withdrawal, or pays down some or all of their credit card balance. Meanwhile, the variable ’OVERDUE_AMT’ will only change whenever a credit card down payment is due, in which case it will increase, decrease or remain constant depending on how much of the balance is paid by the customer. Other variables provide only the single-day expenditure within certain categories, so there may be no pattern between their values from one day to the next. Among these categories are international cash withdrawals, airline expenditure, retail expenditure and fees.

The dataset is created in such a way that all subjects have observations from the day the account in question is opened to the end of the experiment, or until censoring. This applies regardless of whether the subjects experience a default event or not. Due to this, the ’RemaningLifetime’ variable has negative entries for any subject that defaults during the experiment. These negative entries start occurring after the subject defaults. The information provided in these entries is naturally of no use, as the subject has already defaulted, and is thus no longer a customer of the bank, and cannot use their credit card. For this reason, these negative entries are removed from the dataset, so that customers that default do not have observations after they experience the default.

2.2 Imbalance of Dataset

Regarding the response variable 'DC2Ind', there is a clear imbalance in the dataset, in that there are significantly fewer observations of customers that end up defaulting than customers that do not. Imbalanced datasets are known to give rise to a number of challenges when applying machine learning methods (Kuhn and Johnson, 2013). Table 2.1 gives an overview of the numbers and percentages of defaulters and non-defaulters in the dataset.

	Number	Percentage
Defaults	1597	14.1%
Non-defaults	9709	85.9%

Table 2.1: Number and percentage of customers that do/do not experience default during the experiment.

There are several ways to attempt to remedy these problems, such as random over- and undersampling (Brownlee, 2020a). However, this thesis will not make use of such methods.

When dealing with imbalanced datasets, one has to be aware of the consequences the imbalance has on performance metrics. Consider for instance the quite common metric of accuracy, that is,

$$R_A(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n}\sum_{i=1}^{n} I_{y_i=\hat{y}_i}, \tag{2.1}$$

where

$$I_{y_i=\hat{y}_i} = \begin{cases} 1, & \text{if} y_i = \hat{y}_i \\ 0, & \text{if} y_i \neq \hat{y}_i \end{cases} \tag{2.2}$$

is the indicator function. Assume the dataset in question consists of $n = 1000$ samples, 900 of which belong to class 0, and 100 of which belong to class 1. Consider a classifier which is highly biased towards the majority class, and assume it correctly classifies 890 of the samples from class 0, and 5 of the samples from class 1. The reported accuracy of the model would then be 0.895, which would give the indication that the model is performing fairly well. However, the model only correctly classifies 5% of the samples from class 1. In the setting of defaults, this could for instance indicate that around 95% of the customers that are at risk of default will not be detected. When the case is such that the minority class is the class we are mainly interested in, it is clear that a different metric, one which addresses the low true positive rate of the classifier, is required. Two such metrics are the balanced accuracy and Matthews correlation coefficient, which are presented here, based on Brodersen et al. (2010) and Chicco and Jurman (2019), respectively. The balanced accuracy is quite simply the arithmetic mean of sensitivity and specificity,

$$R_{BA}(\mathbf{y},\hat{\mathbf{y}}) = \frac{1}{2}\left(\frac{TP}{TP+FN} + \frac{TN}{FP+TN}\right). \tag{2.3}$$

Here, TP denotes the number of true positive observations, TN the number of true negatives, FP the number of false positives and FN the number of false negatives. The rationale behind this formulation is that the accuracy is calculated within each class, and then the average of these is reported as the overall performance of the model. Considering again the above example, the balanced accuracy score of that classifier would be 0.52. This naturally needs to be seen in connection with the specificity and sensitivity, as otherwise there is no way of knowing whether the model performs well on one class and poorly on the other, or if it performs similarly to random guessing for both classes.
Matthews correlation coefficient is given by

$$R_{MCC}(\mathbf{y},\hat{\mathbf{y}}) = \frac{TP\cdot TN - FP\cdot FN}{\sqrt{(TP+FP)\cdot(TP+FN)\cdot(TN+FP)\cdot(TN+FN)}}. \tag{2.4}$$

This metric ranges from -1 to 1, where -1 is the "worst" value, corresponding to misclassification of every single sample, and 1 is the "best" value, corresponding to perfect classification of all samples. A value of 0 corresponds to random guessing. In the above example, the classifier in question obtains a Matthews correlation coefficient of 0.096. Matthews correlation coefficient has been reported to be a reliable measure for machine learning on imbalanced datasets, as it gives a good score only if the model obtains good results in all four confusion matrix categories, that is, a reasonable amount of true positives, true negatives, false positives and false negatives (Chicco and Jurman, 2019). In this thesis, it will be used as the main metric for evaluating classification models.

Table 2.2 shows how many defaults that occur within seven selected time intervals. The first default takes place after 78 days, and the last one takes place after 684 days. Notice in particular the number of defaulters between day 100 and 150, which is higher than the number of defaulters in any of the other intervals.

Days	77-100	100-150	150-200	200-300	300-400	400-500	>500
Defaults	194	439	157	323	216	171	97

Table 2.2: Number of defaults that occur at selected time intervals.

As can be seen from Table 2.3, half of the customers that experience a default do so in the first 204 days of their customer relationship, and more than 75% default within the first year. This indicates that the model needs to be evaluated using data from early in the customer relationship, as to make sure it is able to make sound predictions also for customers who default relatively early.

Statistic	Days
Mean	240
Std	141
25%	113
50%	204
75%	336

Table 2.3: Some summary statistics of the survival times of uncensored observations. The percentages indicate the number of days at which a certain percentage of the uncensored observations have experienced a default event.

Chapter 3

Theory

The purpose of this chapter is to give an introduction to the theory relevant to the methods used in the thesis. The first section gives an overview of some important concepts in the field of survival analysis, and introduces the notion of a discrete time model, which will be utilized later. The second section focuses on neural networks, and provides an overview and walk-through of their key concepts. In particular, this is where the idea of recurrent neural networks and LSTMs, which are integral to this thesis, will be introduced. Throughout this chapter, we will try to stay true to a notation where regular lowercase letters such as l, m and n are scalar quantities, boldface lowercase letters such as $\mathbf{b}, \mathbf{h}$ and $\mathbf{x}$ are vectors, and boldface uppercase letters such as $\mathbf{A}, \mathbf{C}$ and $\mathbf{W}$ are matrices.

3.1 Survival Analysis

The general objective of survival analysis is to model the time until the occurrence of some event. This time is often referred to as the failure time, survival time, lifetime or simply event time. Letting T denote the stochastic variable for survival time and assuming for the current time being that this is continuous, it can be characterized in terms of its probability density function $f(t)$ and cumulative density function $F(t) = P(T \leq t)$, for $t \geq 0$. Assume in this section that we have a dataset consisting of N observations on the form $(z_i, y_i, x_{i1}, x_{i2}, \ldots, x_{id})$, where z_i denotes the observed survival time, y_i is a binary censoring variable indicating whether the subject experiences an event during the time of the experiment, and x_{ij} is the value of covariate j for subject i, where $j \in \{1, 2, \ldots, D\}$, meaning D covariates are present.

3.1.1 Survival Function and Hazard Rate

The survival function and hazard rate are important concepts in the field of survival analysis. They are two fundamental quantities which play essential roles in the field (Kleinbaum and Klein, 2005). The survival function can be thought of as a means through which one can express the cumulative density function of T,

$$S(t) = 1 - F(t) = P(T > t). \tag{3.1}$$

It gives the probability that an event has not yet occurred at time t, or equivalently when considering death events, the probability that the subject is still alive at time t. It is assumed that all subjects live past $t = 0$ and that the cumulative probability of an event increases towards 1 as t increases, that is, $S(0) = 1$, $S(t)$ is monotonically decreasing and $\lim_{t\to\infty} S(t) = 0$. These assumptions are quite reasonable from both a practical perspective and from assumptions that are usually made about the cumulative density function, namely that it is a monotonically increasing function which approaches the value 1 in the right limit. To also address the practical perspective, we would naturally expect that if we observe a subject over time, the probability that the subject experiences an event during this time would increase if the time interval over which the subject is observed becomes longer.

Besides the probability that a subject will still be alive at any time t, an interesting quantity is the instantaneous risk or rate of an event. This quantity is known as the hazard function, and it is defined by,

$$h(t) = \lim_{\Delta t \to 0} \frac{P(t \leq T < t + \Delta t | T \geq t)}{\Delta t} = \frac{f(t)}{S(t)}. \tag{3.2}$$

As it is put in Kleinbaum and Klein (2005), the hazard function gives us the instantaneous potential per unit time for the event to occur, given that it has not yet occurred by time t. In contrast to the survival function, no particular assumptions are made about the hazard rate, and its shape could be largely dependent on the problem at hand.

The hazard and survival functions are connected through

$$S(t) = e^{-\int_0^t h(u)du} = e^{-H(t)}, \tag{3.3}$$

where $H(t) = \int_0^t h(u)du$ is known as the cumulative hazard function.

3.1.2 Censoring

Given that survival data is gathered on a total of N subjects from an experiment starting at time $t_0 := 0$ and ending at time t_f, it is possible that not all subjects experience an event before the experiment has ended. Furthermore, subjects may exit the experiment before it has ended, also without having experienced an event. This motivates the definition of a *censored* observation: an observation whose value is incomplete due to random factors for the subject (Hosmer et al., 2008). Observations of this this type are said to be *right censored*. It is assumed that censoring is independent of the event in question, and of the censoring and survival status of other subjects. What this means is basically that the knowledge of the event's existence or potential to occur does not affect the subject's decision to stay in or leave the experiment, and that the subject is not influenced by other subjects' death, survival or leaving the experiment. As is understood from the definition, for a censored observation we do not know the true event time. All we know is that the subject had not experienced an event at the time it left the experiment or the experiment was concluded. The importance of this becomes clear when considering what loss function to minimize in the model training routine, as many loss functions use the ground truth directly in order to evaluate the model performance.

3.1.3 Discrete Time Model

In many situations, including the one explored in this thesis, time is viewed in discrete intervals. This follows naturally from the finite precision of time observations. Time is then separated into slices, where each slice represents one time unit. This unit can correspond to any duration of time, e.g. a day, a week, a month etc. Slice i is then denoted by t_i, giving us a series of slices $t_0 < t_1 < ... < t_L$, where $L \in \mathbb{N}$. The extension of the survival function to the discrete case is straightforward,

$$S(t_l) = P(T > t_l) = \sum_{i>l} P(T \in t_i), \tag{3.4}$$

where $P(T \in t_i)$ gives the probability that an event occurs at time t_i, i.e. on the ith day or during the ith week, for instance. This probability can be rewritten in terms of the discrete survival function as

$$p_l = P(T \in t_l) = S(t_{l-1}) - S(t_l). \tag{3.5}$$

Using this and Equation 3.2, the hazard rate in the discrete case can be defined as the conditional probability that an event occurs at time t_l given that it had not occurred by time t_{l-1},

$$h_l = P(T \in t_l | T > t_{l-1}) = \frac{P(T \in t_l)}{P(T > t_{l-1})} = \frac{p_l}{S(t_{l-1})}. \tag{3.6}$$

Note that the hazard function is now defined as a probability, and thus it takes on values between 0 and 1. Also note that if we increase the number of time slices, the difference $t_l - t_{l-1} = \Delta t$ becomes smaller and approaches zero in the limit, resulting in our original definition of the hazard rate in the continuous case.

3.1.4 Performance Metrics for Survival Analysis Models

When evaluating the performance of a survival analysis model, one usually looks at two aspects: discriminatory abilities and predictive abilities, where the latter is also known as *calibration*. Discrimination refers to a models ability to separate subjects with different survival times (Harrel et al., 1996). Roughly speaking, a model has good discriminatory abilities if it in general predicts a shorter remaining lifetime the shorter the true remaining lifetime of the subject, and a longer remaining lifetime the longer the true remaining lifetime of the subject. A common metric for assessing these abilities is the *concordance index*, or *C-index*, and to understand this metric, we need to introduce the concept of *concordance*. Let z_r denote the true survival time of a subject r, and let $\hat{z}_r$ denote the predicted survival time of the same subject. A pair of subjects (r,s) is said to be concordant if it holds that if $z_r < z_s$, then $\hat{z}_r < \hat{z}_s$. In other words, if the observed survival time of subject r is smaller than that of subject s, then the model has to predict a shorter survival time for subject r than for subject s for the pair to be concordant (Harrel et al., 1996). What the C-index expresses is the percentage of concordant subject pairs. It is defined by

$$C = \frac{CPa + 0.5 \cdot TPa}{PPa}, \tag{3.7}$$

where Pa is short for "pairs". Here, CPa denotes the amount of concordant pairs of subjects, TPa denotes the amount of tied pairs (pairs where the predicted survival times are equal) and PPa denotes the total amount of possible pairs. For a pair of subjects to be possible, at least one of them must have experienced an event. That is, if a dataset consists of two subjects that experience an event and three that don't, then there are 7 possible pairings that include at least one of the subjects that experienced an event, and thus $PPa = 7$ in this case.

The predictive abilities, or calibration, of a model refer to the quality of the different predictions relative to the value they are trying to predict. Simply put, a model has better predictive abilities the closer the predictions are to the true values. Given a subject r with lifetime z_r, a model exhibits strong predictive abilities if $|z_r - \hat{z}_r|$ is small relative to the number z_r. When considering predictive abilities, it is important to recall that there only exists a complete ground truth for the uncensored observations. For the censored observations it is only known that the true lifetime is longer than the one observed, and thus it can be difficult to find meaningful metrics for these observations. For now, we omit the censored observations and focus on metrics that can be applied to the uncensored ones. Assume there are M uncensored observations in the considered set. One performance metric which immediately comes to mind is the mean absolute error, defined by

$$\text{MAE} = \frac{1}{M}\sum_{i=1}^{M} |z_i - \hat{z}_i|, \tag{3.8}$$

where z_i is the observed remaining lifetime and $\hat{z}_i$ is the predicted remaining lifetime. The mean absolute error gives an indication of how far from the truth the predictions are on average, and it can work well if the values that z can have are contained in a relatively small interval. Consider that we have five observations with remaining lifetimes 10, 13, 17, 20 and 150 days, and assume the model has predicted remaining lifetimes of 5, 7, 8, 10 and 100 days. This gives an MAE of 16, which is a misleading number for several reasons. First of all, only one of the absolute errors is actually larger than 16. This could lead to the model getting disregarded as too imprecise, when actually, the first four predictions are quite close to their targets, and could, depending on the context, be acceptable predictions. Secondly, the number gives no indication of whether our model in general predicts too small or too large values. Thirdly, it does not take into account the size of the error relative to the true values. In the example, the predicted value of 100 is 50 away from its target, and is thus the prediction that draws the MAE up to a high value. However, relative to its target, it is actually the best of the five predictions. This brings us to the next candidate performance metric, the mean relative absolute error,

$$\text{MRAE} = \frac{1}{M}\sum_{i=1}^{M}\frac{|z_i - \hat{z}_i|}{z_i}. \tag{3.9}$$

The mean relative absolute error gives an indication of how large the error in the prediction is relative to the target value. If it is equal to 0, the prediction is perfect, and the further from 0 it is, the further the prediction is from the true value. If it is equal to 1, the prediction is twice as large as the true value (or equal to 0, however in this case something is likely to be wrong with the model). In our example, we would get an MRAE of 0.46, which more clearly expresses the fact that we have four predictions that deviate from the true value by around half of this value. An advantage of MRAE over MAE is that it will be less affected by the presence of outliers.

A drawback of both performance metrics introduced thus far is that, when applied naively to an entire dataset where the truth value varies over a large range, they do not explain in which regions the model performs well or badly. An MRAE of 0.5 could result from the model predicting values whose deviation from the true value is generally 0.5 times the true value, or it could result from a model predicting progressively worse for increasing remaining lifetime. In the case of prediction of credit card default times, the latter phenomenon is likely to occur.

One metric that is suggested to use for survival data is the Brier score. A major advantage of the Brier score is that it can be adapted to also include censored observations. For right-censored data, Graf et al. (1999) proposed to use a Brier score weighted by the inverse probability of censoring. Now letting t_i denote the survival time for subject i, this Brier score can be expressed as

$$\text{BS}(t) = \frac{1}{N}\sum_{i=1}^{N}\left[\frac{\hat{S}(t|\mathbf{x}_i)^2 I_{t_i \leq t, y_i = 1}}{\hat{S}_{C*}(t_i - |\mathbf{x}_i)} + \frac{(1-\hat{S}(t|\mathbf{x}_i))^2 I_{t_i > t}}{\hat{S}_{C*}(t|\mathbf{x}_i)}\right], \tag{3.10}$$

where $\hat{S}(t)$ is an estimate of the survival function at time t, $\hat{S}_{C*}(t)$ is an estimate for the censoring probability at time t and t_i- is the time right before time t_i. Note that some of the notation here is borrowed from Kvamme (2019), who also provides some discussion around the Brier score as well as other evaluation metrics for survival analysis. Unfortunately, as will be explained in Chapter 4, the Brier score does not quite seem to suit our model.

3.2 Neural Networks

Artificial Neural Networks (ANNs) constitute a branch of methods within machine learning which have enjoyed an immense growth in popularity over the last decades. Advances with regard to these methods have been made within several fields, particularly in image recognition, natural language processing and computer vision, where the so-called multi-layered versions of ANNs have greatly outperformed state-of-the-art methods. Neural networks draw their inspiration from biological neural networks that exist in the bodies of many living creatures, maybe most interestingly in humans.

In this and the following section, the basic principles of deep (and shallow) neural networks are introduced and explained to some degree. This particular section focuses on the central components in a neural network, and how these can be arranged into what is known as different *architectures*. It holds a greater focus on architectures for what is known as Recurrent Neural Networks (RNNs), as these are the ones used to obtain the results in the thesis. As this thesis is more result-oriented and does not base itself upon any particularly new theoretical results, the concepts will be explained quite briefly, and much of the rigorous mathematical derivations and foundations will merely be omitted. One of the primary references used in the chpater is Goodfellow et al. (2016), which has good explanations of many deep learning methods. The relevant theoretical foundations are also covered here. For those interested in further reading on machine learning and its foundations, the books by Hastie et al. (2008) and Shalev-Shwartz and Ben-David (2014) are proposed as reading material.

3.2.1 Feedforward Neural Networks and the Multilayer Perceptron

One of the first modern neural networks that were introduced was the *Rosenblatt perceptron* (Rosenblatt, 1958). It was inspired by neurons in the human body, and in particular by the idea that a neuron could be modelled using a threshold for when it would fire (McCulloch and Pitts, 1943). Later, the idea of the *multi-layer perceptron* (MLP) was introduced, essentially placing several perceptrons after one another in a layered structure. As explained in Du and Swamy (2013), MLPs are what is called *universal approximators* for non-linear functions, meaning they can approximate any non-linear function to an arbitrary degree of precision. The multi-layer perceptron is a class of what is called *feedforward artificial neural networks*. These networks have a structure which is logical and quite easy to grasp, and they make for a good introduction to the components that are central to any neural network, as well the most common mathematical operations that take place in neural networks.

A neural network can be said to have three essential components: weights, biases and non-linear functions. What the network actually does is calculate weighted sums of its inputs, shift the results and apply non-linear transformations. One common representation of neural networks is in the form of a graph, as can be seen in Figure 3.1. The graph consists of nodes and edges, where nodes represent computational units and edges represent the paths along which information flows. Usually, each node is defined by the composition of a non-linear function, known as an *activation function*, with an affine-linear transformation as

$$f(\mathbf{x};\mathbf{w},b) = \sigma\Big(\sum_{i=1}^{d} w_i x_i + b\Big) = \sigma(\mathbf{w}^T\mathbf{x} + b). \tag{3.11}$$

Here, $\mathbf{x} \in \mathbb{R}^d$ is an input vector, $\mathbf{w} \in \mathbb{R}^d$ is a weight vector and $b \in \mathbb{R}$ is a bias term. The activation function σ is a non-linear function which is applied element-wise to its input. Common activation functions are the sigmoid, hyperbolic tangent and ReLU functions, which are defined by

$$\text{sigmoid}(x) = \frac{1}{1+e^{-x}} \tag{3.12}$$

$$\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1} \tag{3.13}$$

$$\text{ReLU}(x) = \max(0, x) \tag{3.14}$$

One or more nodes together make up one *layer* of the neural network, where each node performs an operation similar to that in Equation 3.11, only with different sets of weights. Note that the summation need not be over all $i \in \{1,2,...,d\}$; it is easy to imagine that a node would only incorporate certain parts of the input vector, denoted by an index set $\mathcal{I} \subseteq \{1,2,...,d\}$, and thus have a weight vector $\mathbf{w} \in \mathbb{R}^{|\mathcal{I}|}$, where $|\mathcal{I}|$ denotes the number of elements in the index set $\mathcal{I}$. For simplicity, this section will assume that all layers are *fully connected*, i.e. all nodes in one layer receive input from all nodes in the previous one. The output from a single layer can then be expressed as

$$\mathbf{f}(\mathbf{x};\mathbf{W},\mathbf{b}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{3.15}$$

where now, $\mathbf{W} \in \mathbb{R}^{n\times d}$ denotes the weight matrix (we accept a slight deviation from our standard notation here) and $\mathbf{b} \in \mathbb{R}^n$ denotes the bias vector. Note that what is being described here is a layer with n neurons, with each neuron receiving an input in the form of a vector of dimension d. Defining the vector-valued function $\mathbf{g}_l(\mathbf{x};\mathbf{W},\mathbf{b}) = \mathbf{W}^l\mathbf{x} + \mathbf{b}^l$, where $\mathbf{W}^l$ is the weight matrix and $\mathbf{b}^l$ is the bias vector for the lth layer, the final output of a neural network with L layers is a composition of operations,

$$\mathbf{f}(\mathbf{x};\mathbf{W},\mathbf{b}) = (\sigma_o \circ \mathbf{g}_L \circ \sigma \circ \mathbf{g}_{L-1} \circ \ldots \circ \sigma \circ \mathbf{g}_1)(\mathbf{x};\mathbf{W},\mathbf{b}). \tag{3.16}$$

Here, $\mathbf{W} = (\mathbf{W}^1, \mathbf{W}^2, ..., \mathbf{W}^L)$ is a vector containing the L weight matrices, and $\mathbf{b} = (\mathbf{b}^1, \mathbf{b}^2, ..., \mathbf{b}^L)$ is a vector with the L bias vectors as columns. The activation function of the last layer, which is the output layer, is denoted σ_o, as this activation function may be different from the ones used in the other layers. Of course, one

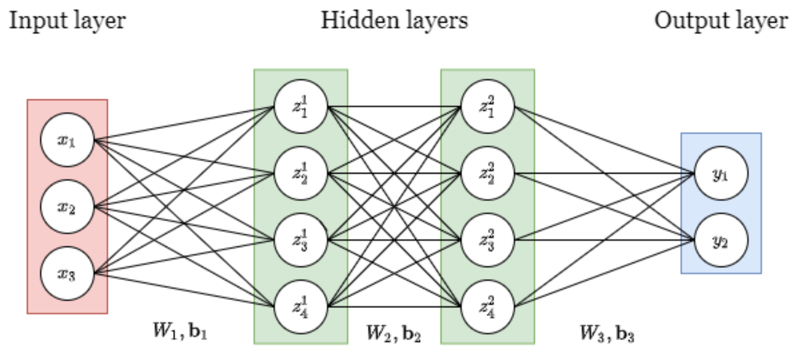

Figure 3.1: Illustration of a simple fully connected ANN. $\mathbf{W}^i$ and $\mathbf{b}^i$, $i \in \{1, 2, 3\}$, denote the weights and biases, respectively, of the hidden and output layers.

could also have different activation functions between different layers, however to keep it simple it is for the moment assumed that all layers except possibly the output layer have the same activation function.

A simple fully connected neural network is illustrated in Figure 3.1. The values in the hidden and output layers are computed through an affine-linear transformation followed by a non-linear transformation. For instance, the values in the first hidden layers are computed through $z_i^1 = \sigma((\mathbf{w}_i^1)^T \mathbf{x} + b_i^1)$. The subscript gives the bias or vector of weights specific to hidden node i. The attentive reader may have noticed that the words "node" and "neuron" seem to be used interchangeably here, and to make it clear, we are referring to the exact same thing when using either of these words.

So in short, what a neural network does is to implement a composition of affine-linear transformations and non-linear activation functions. The inclusion of non-linearity through the activation functions gives neural networks increased flexibility, precisely in that it allows the network to capture non-linear effects between the variables. A composition consisting only of affine-linear transformations would just be a new affine-linear transformation, as can easily be seen by composing, say, the function $\mathbf{h}_1(\mathbf{x}) = \mathbf{U}\mathbf{x} + \mathbf{a}$ with the function $\mathbf{h}_2(\mathbf{x}) = \mathbf{V}\mathbf{x} + \mathbf{b}$,

$$(\mathbf{h}_1 \circ \mathbf{h}_2)(\mathbf{x}) = \mathbf{h}_1(\mathbf{h}_2(\mathbf{x})) = \mathbf{U}(\mathbf{V}\mathbf{x} + \mathbf{b}) + \mathbf{a} = \mathbf{U}\mathbf{V}\mathbf{x} + \mathbf{U}\mathbf{b} + \mathbf{a} = \mathbf{W}\mathbf{x} + \mathbf{c},$$

defining $\mathbf{U}\mathbf{V} = \mathbf{W}$ and $\mathbf{U}\mathbf{b} + \mathbf{a} = \mathbf{c}$. Thus, without activation functions, neural networks would simply implement a regular linear regression. Activation functions also play another role in neural networks, which is the reason why they are called activation functions in the first place. They mimic the effect of action potentials in biological networks, in that they make neurons "fire" upon sufficient activation (Barnett and Larkman, 2007). In practice, this is done by designing the activation functions in such a way that they output larger values when their input is larger, i.e., they are monotonically increasing (or at least non-decreasing) functions. One useful function for getting an intuition of how this works is the sigmoid function, defined as

$$\sigma_s(x) = \frac{1}{1+e^{-x}}. \tag{3.17}$$

The sigmoid function ranges from zero to one and is monotonically increasing. In other words: the larger the value of the input, the closer the output value is to one. This can be interpreted as the neuron (or node) being "activated" when the output from the affine-linear transformation is large, and it being "turned off" when this value is small. This suggests a neat effect where a neural network can train its weights in such a way that the most relevant neurons achieve the highest degree of activation upon any input.

3.2.2 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are neural network architectures that are constructed to be able to effectively handle sequential data, that is, data with observations of the same variable at different times. In this type of data, the way in which a variable changes over time might be of interest, not only the value of the variable itself. RNNs attempt to capture such effects by "remembering" data from earlier in the sequence when dealing with data later in the sequence. Interestingly, whereas fully connected, feed-forward neural networks are universal approximators of non-linear functions, recurrent neural networks are universal approximators of dynamical systems (Du and Swamy, 2013), which gives an indication of the vast range of tasks that RNNs can be appropriate for. For instance, they have been highly successful in tasks such as speech recognition, image captioning and natural language processing (Graves et al. (2013), Wang et al. (2016), Yin et al. (2017)).

To talk about RNNs and how they treat data at different time steps, we have to introduce the idea of a network's *state*, or *cell state*. A cell is simply a computational unit with an associated set of weights. The cell state is then the output of a given cell at any time. The key to capturing time effects lies in what is called *feedback connections*, which is a way of feeding the previous state of the network into its current state. Before the previous state is fed into the current one, it undergoes a transformation using a separate set of weights which is associated with the feedback connection. The result of this transformation is known as the *hidden state*. Henceforth we will denote the cell state of cell t in layer m by $\mathbf{A}_t^m$, and the hidden state based on this cell state by $\mathbf{h}_t^m$.

The idea of an RNN is visualized in Figure 3.2, where the network has been unrolled in time. That means that the feedback connections are indicated by the arrows going sideways, which might seem a bit confusing at first, but which can be understood as them feeding "old" information forwards in time. The network in the figure has L layers and goes over n time steps. This can be interpreted as the network having n cells in each layer. Cells in the same layer share the same two sets of weights: $\mathbf{W}^{l_i}, i \in \{1, 2, ..., L\}$, and $\mathbf{W}^h$. This parameter sharing is essential to the model's ability to generalize information from different time steps (Goodfellow et al., 2016). Note that the biases $\mathbf{b}$ have been left out for visualization purposes.

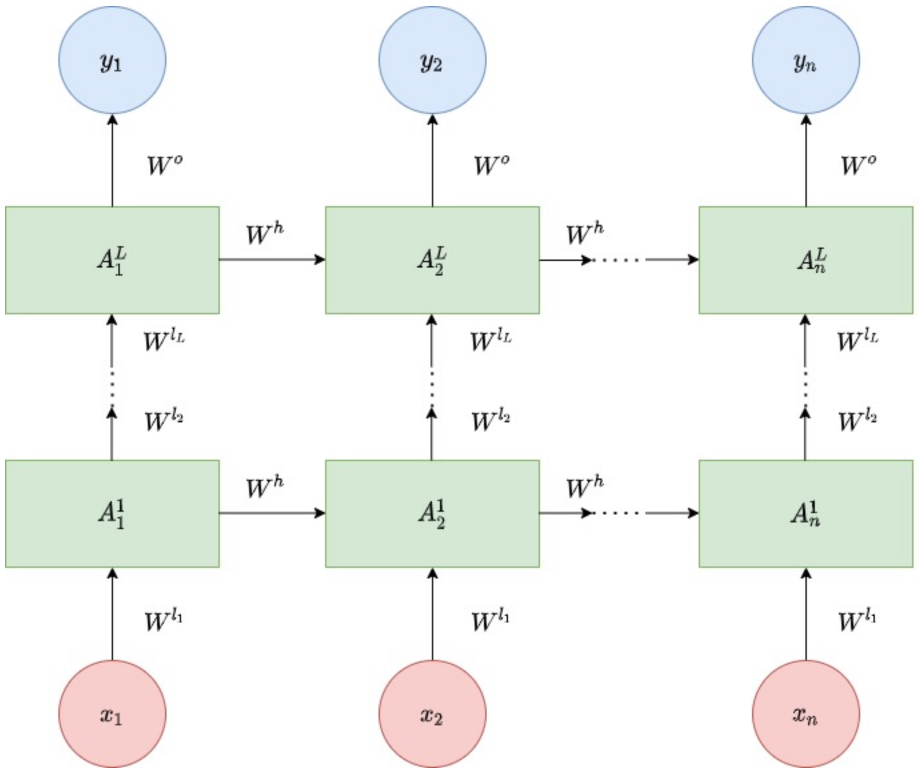

Figure 3.2: Illustration of a layered RNN. The arrows indicate the flow of information in the forward pass. For visualization purposes, the biases **b** have been left out.

For RNN cell t in layer m, the output computed is

$$\mathbf{A}_t^m = \tanh(\mathbf{W}^{l_m}\mathbf{A}_t^{m-1} + \mathbf{W}^h\mathbf{A}_{t-1}^m), \tag{3.18}$$

and the hidden state is given by

$$\mathbf{h}_t^m = \mathbf{W}^h\mathbf{A}_t^m. \tag{3.19}$$

Note that it is often usual to denote the output from regular RNNs by **h**, and talk only about the hidden state, rather than a hidden state and a cell state. In this section we have deviated slightly from this notation, which does not change the number of sets of weights or what operations are performed during the forward (or backward) pass. Now, some confusion might arise here regarding the information flow in the RNN, so we will take a moment to try and clear up some of this confusion. The cell state $\mathbf{A}_t^m$ is the value computed by the network in cell t, or at time step t, and in layer m. It is computed based on inputs "coming from below" and "coming from the left" in Figure 3.2. The input coming from below is either the vector of model covariates $\mathbf{x}$ (if $m = 1$) or a scaled version of the previous cell state, $\mathbf{W}^{l_m}\mathbf{A}_t^{m-1}$ (if $m > 1$), where $\mathbf{A}_t^{m-1}$ is the previous cell state. The input coming from the left is the hidden state $\mathbf{h}_{t-1}^m = \mathbf{W}^h\mathbf{A}_{t-1}^m$. After the cell state $\mathbf{A}_t^m$ has been computed as in Equation 3.18, it is passed out of the cell in two directions: upwards and to the right. The output going upwards is scaled by the weights $\mathbf{W}^{l_{m+1}}$, and the output going to the right is scaled by $\mathbf{W}^h$. After scaling, the value going to the right is what we have defined as the hidden state $\mathbf{h}_t^m$ in this section. For $t = 1$, i.e. for the first time step, the hidden state is initialized in some pre-determined way. Often, it is simply set to zero.

3.2.3 Long Short-term Memory Networks

Long short-term memory units (LSTMs) were first introduced by Hochreiter and Schmidhuber (1997). They are a type of RNNs that are designed to handle two problems occurring in RNNs: the problems of vanishing gradients and long-term dependencies. The latter is a phenomenon where information can get lost during the many iterations of an RNN. The main consequence of this is that if an observation made at time t might be of importance together with an observation made at a later time $t + s$, the network will not capture this importance because it has "forgotten" the information from time t during the intermediate time s. The LSTM architecture amends this problem by having a kind of gated structure, where each gate has a certain task. These tasks will now be explored in some more detail. When working with LSTMs, it is common to denote the hidden state at time t by $\mathbf{h}_t$, as we did for RNNs in the previous section, and the cell state at time t by $\mathbf{C}_t$. We follow this notation in this section.

One example of how an LSTM cell is often structured is shown in Figure 3.3. The equations behind the different quantities are given below. For the sake of simplicity, the superscript indicating the layer is dropped. This type of LSTM cell uses one of the most common architectures for this type of cell, and it is made up of three gates: the forget gate (output $\mathbf{f}_t$), the input gate (output $\mathbf{i}_t$) and the output gate (output $\mathbf{o}_t$). All the three gates get the same input, which is the previous hidden state $\mathbf{h}_{t-1}$ and the data $\mathbf{x}_t$. The role of the forget gate is to decide what parts of the previous cell state to discard and what parts to keep. Its output is determined by a sigmoid function, and so it is between 0 and 1. Thus, when performing the elementwise multiplication $\mathbf{C}_{t-1} \odot \mathbf{f}_t$, parts of the previous cell state $\mathbf{C}_{t-1}$ are kept to a larger degree the closer its corresponding element in $\mathbf{f}_t$ is to 1. The input gate has a quite similar role, as it decides what parts of the cell state will be updated. Just like for the forget gate, this is done through the use of a sigmoid function, followed by an elementwise multiplication with the cell state update vector $\mathbf{g}_t$. The resulting vector is then added to the previous cell state to create the current cell state, $\mathbf{C}_t$. A tanh activation function is then applied to the cell state before it is passed through the output gate. Finally, the role of the output gate is to decide what parts of the updated cell state to output from the cell. Again, this is done through the use of elementwise multiplication with the output from a sigmoid function. For an LSTM network with multiple layers, $\mathbf{x}_t$ is simply replaced with the output from the corresponding cell in the previous layer, i. e. with $\mathbf{h}_t$. The equations for calculating the outputs at different points in the LSTM cell are given below. In these equations, we let $\mathbf{V}_t$ denote the weights used to scale the input coming from below in Figure 3.3, and $\mathbf{W}_t$ denote the weights used to scale the input coming from the left. The superscripts simply follow the symbol given to the different functions.

$$\mathbf{f}_t = \sigma\left(\mathbf{b}_t^f + \mathbf{V}_t^f \mathbf{x}_t + \mathbf{W}_t^f \mathbf{h}_{t-1}\right) \tag{3.20}$$

$$\mathbf{i}_t = \sigma\left(\mathbf{b}_t^i + \mathbf{V}_t^i \mathbf{x}_t + \mathbf{W}_t^i \mathbf{h}_{t-1}\right) \tag{3.21}$$

$$\mathbf{g}_t = \tanh\left(\mathbf{b}_t^g + \mathbf{V}_t^g \mathbf{x}_t + \mathbf{W}_t^g \mathbf{h}_{t-1}\right) \tag{3.22}$$

$$\mathbf{C}_t = \mathbf{C}_{t-1} \odot \mathbf{f}_t + \mathbf{i}_t \odot \mathbf{g}_t \tag{3.23}$$

$$\mathbf{o}_t = \sigma\left(\mathbf{b}_t^o + \mathbf{V}_t^o \mathbf{x}_t + \mathbf{W}_t^o \mathbf{h}_{t-1}\right) \tag{3.24}$$

$$\mathbf{h}_t = \tanh\left(\mathbf{C}_t \odot \mathbf{o}_t\right) \tag{3.25}$$

Here, the quantity denoted by $\tilde{C}_t$ in the figure is given the name g_t for simplicity and readability. Note that the subscript t indicates the time step, and is not meant to be an index. The operator $\odot$ denotes elementwise multiplication.

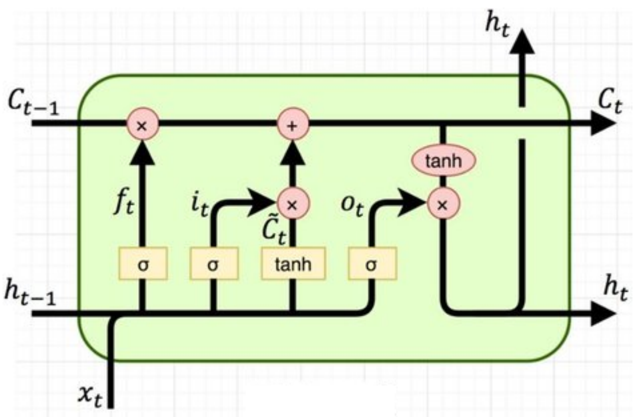

Figure 3.3: Illustration of an LSTM cell. The arrows indicate the flow of information in the forward pass.

3.3 Training a Neural Network

When constructing a neural network in order to perform a task, as with all other machine learning methods, the network needs to be trained. The performance of the network will depend heavily on the training routine; how the data is pre-processed, what loss function is chosen and what optimization algorithm is used, to mention a few elements. There is also the problem of hyperparameter tuning, which is an entire field of study in and of itself. This section explores the most important concepts of the neural network training procedure. It looks at the basics of the forward and backward pass, which are the main ingredients of training a neural network, as well as some other concepts that are often applied to make the training faster or better.

3.3.1 The Purpose of Training and the Problem of Overfitting

Clearly, the main objective of training a neural network is to make it as skilled as possible at solving a task, whether it is recognizing specific figures in an image, classifying a tumor or predicting time to default for credit card customers. As with most machine learning algorithms, this is done through the minimization of a loss function $\mathcal{L}$ which is determined beforehand, and which is somehow connected to the task the network is set to perform. Thus, the main objective of a neural network training procedure is simply to minimize some function. Easy, right? Well yes, but actually no. First of all, this minimization is usually very difficult, and has to be performed numerically on a huge parameter space. Secondly, there is the problem of overfitting, a very well known issue in machine learning. Overfitting occurs when a model "memorizes" the training data, rather than learning from it. There is no point in training a model if it does not generalize, i.e. if it is not able to make reasonable predictions on new data. In the case of neural networks, overfitting often occurs when the model is trained for too long. Fortunately, there exists several methods for avoiding overfitting, one of which is very simple and intuitive, and which is a key ingredient to training a neural network. Let's take a quick look at the general training procedure and how this gives rise to the method advertised.

The procedure of training a neural network is usually an iterative process. First, the data at hand is divided into three parts: training, validation and test sets. The data from the training set is then repeatedly passed through the network in what is known as the *forward pass*. Each forward pass is followed by a *backward pass*, which is the procedure that allows the model to actually learn and improve its predictions. Each pass through the whole training set is known as an *epoch*, and at the end of each epoch, a loss value is calculated. If the loss value after one epoch is lower than it was after the previous one, this is a sign that the model has made progress in its learning. Therefore, to be able to ensure that the model is actually learning, these losses are plotted, resulting in what is called *loss curves*. Loss curves having a decreasing tendency is thus a sign that the training is going properly. Now, overfitting usually occurs when the model has "seen" the training data too many times, and mostly starts memorizing the training samples. Naturally, the model will then achieve ever better performance on the training set, meaning the training loss will keep decreasing. However, what we want is for the model to

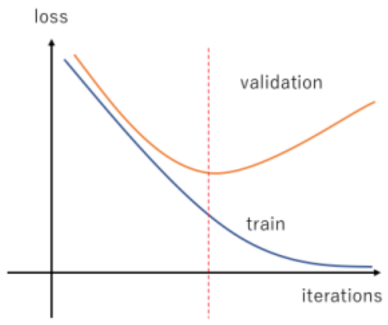

Figure 3.4: Example of loss curves. The blue curve indicates the training loss, and the orange curve indicates the validation loss. The dashed red line intercepts the validation loss curve at its lowest value.

perform optimally on unseen data. This is where the importance of the validation set becomes clear. In addition to tracking the loss on the training set, we could also track the loss on the validation set, and use this as an indicator for when the model starts overfitting. The model does not learn anything from the validation data, as no backward pass is performed during validation, so there is no danger of the model overfitting to these observations. The strategy is then to monitor the validation loss, and as long as it decreases, the model is getting better at generalizing to unseen data. When the model starts overfitting, its performance on unseen data will start to decrease, and the validation loss will start increasing. Thus, if we cut off training just as the validation loss starts to increase, we avoid overfitting and get a model that (hopefully) generalizes well to unseen data. This process is visualized in Figure 3.4, where the dashed red line indicates the time during training where the validation loss attains its minimum.

Usually when training a neural network, one will manually set a number of epochs for the model to go through. From the reasoning above it is clear that the minimum value for the validation loss may be obtained before the network has gone through all of its planned epochs. The process of stopping training earlier than initially planned has its own, very apt name. It is called *early stopping*, and can be implemented in a number of ways, but the simplest one is to stop training when there has been no improvement in the validation loss for a certain number of iterations. Another way is to set some minimum required improvement $\Delta\mathcal{L}$ and stop training if no such improvement has occurred over a certain number of iterations.

3.3.2 Data Preparation for Neural Networks

The first task of any machine learning procedure is to process and engineer the data in a way that makes it possible for the model to operate on and learn from it. In the case of neural networks, this often involves scaling the data, either through normalizing it to lie within some range, or to standardize it. This is particularly important when dealing with multiple explanatory variables that have different scales or value ranges. Scaling has also been shown to promote model performance when the range of the variable in question is large, e.g. from 0 up to 1000s (Brownlee, 2019b). Some neural networks require their input data to be scaled in a certain way, while others simply benefit from it. The main contributions of scaling is to make the parameter estimates more stable, make training and convergence faster and avoid hurting the learning process due to different scales in the variables.

3.3.3 Weight Initialization

When starting to train a neural network, its weights need to be initialized with some starting values in order to do the first forward pass. This can be done in multiple ways and has a large impact on training. Intuitively one would, if possible, initialize the weights in such a way that they would lie in the vicinity of the global minimum of the objective function. However, when dealing with neural networks, the parameter space is often extremely large, and the objective quite complex, so that this information is usually not available. It can also happen that the algorithm does not terminate in a local or global minimum, or even that no global minimum exists (Goodfellow et al., 2016). So why care about weight initialization? Because the weights are important for a reason different than just explaining where on the objective surface the algorithm's current value is. Neural networks are trained using gradient-based optimization, and the weights are involved in this process. For instance, initializing all weights as the same value will lead to all hidden layers computing the same function. This leads to a lack of what is known as *symmetry breaking*, which is necessary for the two layers to learn different aspects of the dataset, which is the main motivation for using multi-layered networks. In other words, the layers need to have different initial parameters. This leads to the idea of initializing with random values, usually through the use of a zero-centered Gaussian or uniform distribution. A further consideration is then what variance to use in the distribution one is sampling from. A too small variance means the weights are largely centered around 0, which can lead to the outputs growing smaller and smaller throughout the layers of the network. This eventually leads to the gradient of layer, say, j, wrt the weights of layer $j—1$ becoming small, thus ruining the gradient signal through the chain rule. A too large variance can lead to weights being so large that activation functions such as the sigmoid and tanh *saturate*, meaning they output many values that are outside of the main active area of their gradients. By active area is meant the part or parts of the function where the gradient is significantly different from zero. For the sigmoid function defined in Equation 3.17, for instance, the active area of the gradient would be a small area centered around $x = 0$, extending equally far on both sides to around $x = \pm 2$ or $x = \pm 4$. There is no strict definition of what the gradient value needs to be for the argument to be considered to be in the active area, however by looking at the sigmoid curve, one can get an impression of what is meant by the expression. Saturation often leads to small gradients, and thus what is known as the *vanishing gradient problem*, where gradients become smaller and smaller in value, approaching zero, and negatively affecting the model's ability to perform reasonable parameter updates. The choice of variance in the weight sampling distribution is therefore important in order to avoid this problem. One usual approach to decide what this variance should be is to use so-called Xavier initialization. Xavier initialization seeks to enforce

$$\mathrm{Var}(\mathbf{z}) = \mathrm{Var}(\mathbf{x}), \tag{3.26}$$

where $\mathbf{z}$ is the output from a layer taking $\mathbf{x}$ as its input. In other words, it seeks to ensure that the variance of the output from one each layer is equal to the variance of the input to this layer. This amounts to initializing the weights of layer i from a Gaussian or uniform distribution with zero mean and variance $1/\sqrt{n_i}$, where n_i is the number of input neurons in layer i. What Xavier initialization accomplishes is maintaining the gradient signal throughout the network, preventing vanishing or exploding gradients in the deeper layers of the network, as explained in Glorot and Bengio (2010). In this article, the uniform distribution is used as sampling distribution.

3.3.4 Forward Pass

The forward pass is the most basic part of the neural network training procedure. During the forward pass, input is fed into the network and passed through the layers. Computations are performed at each node, and the resulting values are then summed up, before an activation function is applied, as explained earlier in this chapter. This acts as the output from one layer, and is then passed on to the next layer. This is repeated until the output layer is reached, in which the final outputs of the model are computed. These outputs are then compared to the ground truth through the computation of the loss function value, usually simply referred to as the loss. Very often, additional concepts such as batch normalization and dropout are included during training in order to avoid overfitting or to avoid problems such as vanishing and exploding gradients. These concepts often behave differently when training and testing the model, and some of them are explored later in Section 3.3.7.

3.3.5 Backpropagation and the Role of Gradients

After the forward pass has been performed, the model outputs have been produced and the loss value has been computed, the loss value needs to be used in order to "inform" the network about how well its predictions compare to the true values. The network, in turn, needs to update its parameters in a way that allows it to make better predictions. This is done by computing the gradients of the loss function with respect to the different parameters in the network. That is, we are interested in computing

$$\frac{d\mathcal{L}}{dw_{ijk}}, \quad \text{for } i \in \{1,2,...,L\},\ j \in \{1,2,...,n_i\},\ k \in \{1,2,...,d\}. \tag{3.27}$$

Here, the index i gives the layer of the weight, j gives the node in layer i the weight is associated with and k gives the index of the input that the weight is multiplied with. For networks with many layers, this might seem like a daunting task at first. However, fortunately, the gradients can be computed easily through the chain rule, which states that:

$$\frac{df(g(x))}{dx} = \frac{dg(x)}{dx} \cdot \frac{df(g(x))}{dg(x)}, \tag{3.28}$$

i.e. the derivative of a composite function $f(g(x))$ with respect to the argument x can be factorized in terms of the derivative of f with respect to g and the derivative of g with respect to x. Applying this to Equation 3.27, we obtain

$$\frac{d\mathcal{L}}{dw_{ijk}} = \frac{d\mathcal{L}}{dz_m} \cdot \frac{dz_m}{dz_{m-1}} \cdot .... \cdot \frac{dz_2}{dz_1} \cdot \frac{dz_1}{dw_{ijk}}, \tag{3.29}$$

where z_l for $l \in \{1,2,...,m\}$ are the intermediate states that are computed in the network. This is a crucial part of the backward pass through a neural network, and is called *backpropagation* of gradients, since the gradients are computed starting at the final layer of the network, and then passed backwards in order to compute other gradients. For more material about backpropagation, see e.g. Goodfellow et al. (2016).

3.3.6 Optimization and Stochastic Gradient Descent

The goal of optimization in machine learning models is usually to update the model parameters, so that the model makes better predictions. Given some loss function $\mathcal{L}$, a ground truth value y and a model output $\hat{y}$, the objective of optimization is to find the parameter values that solve the minimization problem

$$\min_{\mathbf{W},\mathbf{b}} \mathcal{L}(\hat{y}, y). \tag{3.30}$$

Many models, including neural networks make use of gradient-based optimization. The objective then is to try and find parameters such that the gradient of the loss function is zero, i.e. such that $\nabla\mathcal{L} = 0$. Unfortunately, due to the large parameter space we are often dealing with, as well as the complexity of our model, it is usually not possible to find an analytical solution to the optimization problem 3.30. Thus, numerical optimization is used instead. One of the simplest gradient-based numerical optimization procedures, which is very often used when training neural networks, is the gradient descent algorithm. It implements the iteration

$$\theta_k = \theta_{k-1} - \gamma\nabla_\theta\mathcal{L}(\hat{y}, y). \tag{3.31}$$

Here, θ_k denotes the model parameters at step k of the algorithm, γ is known as the learning rate and ∇_θ denotes the gradient with respect to the parameters θ. The idea of gradient descent is that by changing the parameter values such that one goes in the opposite direction of the loss function gradient, the loss will decrease as steeply as possible. The learning rate determines how far to go in this direction before performing the parameter update, and it is in fact an extremely important hyperparameter in all neural networks. By setting the loss rate too large, an update step might overshoot and end up far beyond the minimum the algorithm is headed towards. By setting it too small, training will take a long time. We will come back to this parameter in Subsection 3.3.8. Like most other optimization algorithms, gradient descent is usually iterated until convergence, which is often defined by the condition $||\theta_k - \theta_{k-1}|| < \epsilon$ for some chosen value of ϵ. Another common stopping criterion is to repeat the algorithm for a certain number of iterations before it is terminated.

When the training set is very large, which is often the case when neural networks are used, computing the gradient becomes very expensive. This problem gives rise to *stochastic gradient descent*, which involves computing the gradient based on only a smaller subset of the training samples. This allows for more frequent parameter updates, which again leads to faster convergence. The foundations of stochastic gradient descent come from empirical risk minimization, which states that the expected average loss on any subset of the training set is equal to the expected average loss on the entire training set (Goodfellow et al., 2016). Using this property, the network can be trained much more efficiently, with parameter updates based on only a few training samples, rather than the entire dataset. The amount of samples used to compute the gradient at each iteration is known as the *batch size*, and usually consists of somewhere between one and a few hundred samples. The batch size is one of the many hyperparameters of a neural network, and the chosen batch size may have an effect on training. In general, larger batch sizes will slow down training, but give better gradient approximations. Smaller batch sizes mean the training is sped up, but the gradient approximations are worse. Finding the optimal batch size is therefore of great interest to the programmer, and will vary from network to network and problem to problem.

3.3.7 Additional Training Procedures, Dropout

As mentioned in Subseciton 3.3.4, additional procedures are often included in the training of neural networks. These procedures are included with the purpose of reducing training time, preventing overfitting or enhancing the efficiency of the training procedure in other ways. Early stopping, which was mentioned in Subsection 3.3.1, is one example of such a procedure, and contributes to avoiding overfitting. Other examples of additional training procedures are batch normalization, weight decay and dropout. We will take a closer look at the last one, as it is an interesting method which is quite easy to understand.

Dropout in neural networks has as purpose to regularize the neural network, or in other words, to reduce the chance of overfitting (Srivastava et al., 2014). The way it works is that nodes are randomly dropped from training at each iteration, meaning that they are essentially not part of the training procedure for that iteration. As a result, the training is actually done on many different architectures, rather than on one architecture alone. The reason why this is an advantage is that combining several models nearly always improves performance, and is in and of itself a type of regularization. The way dropout is implemented is that, during training, each node is dropped out with a set probability p. As mentioned, the dropped nodes are then excluded from the forward and backward pass during the current iteration. When the neural network is to be validated or tested, no nodes are dropped out, but the weights of the nodes are scaled down. The scaling factor is decided by the probability that the node in question would be retained during training (Srivastava et al., 2014). Notice that dropout introduces another hyperparameter: the *dropout probability*, p.

3.3.8 Hyperparameters

Depending on what architecture, optimization routine and loss function is used, there may be a very large amount of hyperparameters present when training a neural network. Some actually concern the architecture itself, such as the number of hidden layers or the number of nodes in each layer. Even the activation function can be viewed as a hyperparameter. One of the most important parameters is the learning rate (Goodfellow et al., 2016). As explained in Section 3.3.6, the learning rate determines how large a step is taken in the gradient direction when this has been computed. Finding the best possible learning rate is extremely helpful, as this will contribute to reduce training time as well as improve the loss. The learning rate is in a slightly unique position, in that one can get an impression of whether it is too large or too small through loss curves (Brownlee (2019a)). If the learning rate is too large, the loss curve will often have a sharp drop and then plateau after quite few iterations. If it is too small, the loss curve will decrease very slowly. Thus, loss curves can be of great help when tuning the learning rate.

Deciding on hyperparameter values can be a significant challenge. Setting the values manually requires insight into how the different parameters affect model performance, as well as into how they interact. Even with extensive knowledge of the hyperparameters and their effects, manually finding optimal values in the huge parameter space can be difficult. This motivates the use of more structured and comprehensible hyperparameter search methods, of which there are several types to choose from. Two simple ones are grid search and random search, which are both described in Goodfellow et al. (2016). Unfortunately, these types of searches are often very time consuming procedures, as optimal hyperparameter values are not necessarily independent of other hyperparameters. Because of this, not only all possible hyperparameter values, but also all possible hyperparameter combinations, must be considered. For illustration purposes, assume we have three hyperparameters we want to optimize, and we start with five candidate values for each hyperparameter. Rather than training the model 15 times, five for each hyperparameter, we would have to train the model $5^3 = 125$ times. It is easy to see how this procedure scales and becomes infeasible very quickly. Grid search also requires a grid to be set, meaning the programmer should have some knowledge of what parts of the parameter space are worth searching. More advanced methods exist within the field known as hyperparameter optimization. Examples are Bayesian Optimization (Shariari et al., 2015) and Genetic Algorithms (Lambora et al., 2019). Design of Experiments (DoE) and Response Surface Methodology are also potential methods for use in hyperparameter tuning (Weihs et al., 2006).

Chapter 4

Method

The aim of this chapter is to introduce the method that will be used in order to get results in this thesis. The discrete time model for survival analysis has already been introduced in the theory section, and this method builds on that model. The method is inspired by the one presented and used in Ren et al. (2018), and this chapter will largely introduce the method in the same way it was introduced in that article. The chapter will also introduce different models that will be trained, and discussion on how these will be evaluated.

4.1 Dataset and Notation

Before introducing the method, some explanation of the outlook of the dataset is needed. We assume that there are N unique customers in the dataset, and that customer i for $i \in \{1, 2, ..., N\}$ has N_i observations on D variables. We further assume that observation k for customer i, where $k \in \{1, 2, ..., N_i\}$, is on the form $(z_i^k, y_i, x_{i1}^k, x_{i2}^k, \dots, x_{id}^k)$, where z_i^k is the true remaining lifetime of the customer at observation k, y_i is a binary variable indicating whether the subject experiences an event during the time of the experiment, and x_{ij}^k is the value of covariate j for subject i at time k, where $j \in \{1, 2, \dots, D\}$. The vector of covariates for customer i at time k is then denoted by $\mathbf{x}_i^k$. Further, denote the matrix whose rows are covariate vectors at times ranging from t_1 up to t_{N_i} by $\mathbf{X}_i$, with $\mathbf{X}_i \in \mathbb{R}^{N_i x D}$. This means that these matrices may be of varying size, depending on how many observations exist for the different customers.

4.2 Recurrent Model and Survival Analysis

The core idea of this method is to use a recurrent neural network to predict hazard rates and survival functions for the different customers at different times. The way this is done is by viewing the output of cell l of the network as the hazard rate at time t_l. That is, we estimate the hazard rate of customer i at time t_l, denoted by

h_i^l, through

$$h_i^l = f_{\mathbf{\Theta}}(\mathbf{x}_i^l, t_l | \mathbf{h}_i^{l-1}). \tag{4.1}$$

Here, we let $\mathbf{\Theta}$ denote the model parameters, and thus $f_{\mathbf{\Theta}}$ denotes the function that the network with these parameters implements, and $\mathbf{h}_i^{l-1}$ denotes the hidden state of the network for customer i at time t_{l-1}. Take care not to confuse the hazard rate h with the hidden state $\mathbf{h}$ here. The former will exclusively be used as a scalar throughout the chapter, and the latter exclusively as a vector. Using the chain rule of probability, the survival function can be expressed through these estimated hazard rates:

$$\begin{aligned}
S_i(t_l|\mathbf{X}_i, \mathbf{\Theta}) &= P(T_i > t_l|\mathbf{X}_i, \mathbf{\Theta}) \\
&= P(T_i \notin t_1, T_i \notin t_2, ..., T_i \notin t_l|\mathbf{X}_i, \mathbf{\Theta}) \\
&= P(T_i \notin t_l|T_i \notin t_1, T_i \notin t_2, ..., T_i \notin t_{l-1}, \mathbf{X}_i, \mathbf{\Theta}) \\
&\cdot P(T_i \notin t_{l-1}|T_i \notin t_1, T_i \notin t_2, ..., T_i \notin t_{l-2}, \mathbf{X}_i, \mathbf{\Theta}) \\
&\cdot ... \cdot P(T_i \notin t_2|T_i \notin t_1, \mathbf{X}_i, \mathbf{\Theta}) \cdot P(T_i \notin t_1|\mathbf{X}_i, \mathbf{\Theta}) \\
&= \prod_{j=1}^{l}(1 - P(T_i = t_j|T_i > t_{j-1})) \\
&= \prod_{j=1}^{l}(1 - h_i^j)
\end{aligned} \tag{4.2}$$

Thus, what is obtained from the model really is the predicted hazard rate for customer i at each observed time $t_l, l \in \{1, ..., N_i\}$. Note that this is not an actual prediction of the remaining lifetime of the subject, nor is it a prediction of a future hazard rate; it simply estimates the hazard rate at a time already observed, meaning at a time at which we know the subject had not yet experienced an event. The hope is then that these hazard rates can somehow be used to make predictions about the remaining lifetime of subjects. In the spirit of the C-index introduced in Chapter 3: we want larger predicted hazard rates for the subjects that have a shorter true remaining lifetime. This is where the loss functions come into play.

4.3 Loss Functions

As is proposed in Ren et al. (2018), we will be using multiple loss functions in order to train our model. The necessity for this arises from the presence of censored observations, which makes it difficult to provide direct supervision on the observed survival times, as mentioned in Section 3.1. To deal with this, two loss functions will be used, and they will be combined to a total loss through a convex combination,

$$\mathcal{L}_{Tot} = \alpha\mathcal{L}_1 + (1-\alpha)\mathcal{L}_2, \tag{4.3}$$

where $\alpha \in (0, 1)$ determines which loss function is attributed the most weight.

The objective of our training algorithm is then to solve the minimization problem

$$\min_{\mathbf{\Theta}} \mathcal{L}_{Tot}(\hat{\xi}, \xi), \tag{4.4}$$

where ξ is the truth value which we want to achieve, and $\hat{\xi}$ is the predicted value. This perhaps overly generalized formulation might seem unnecessary, but it is used because several of the loss functions introduced in this section do not directly supervise over the remaining lifetime. Instead, they might have the goal of pushing a certain quantity to a specific value, e.g. pushing the survival function towards zero or one. The two loss functions $\mathcal{L}_1$ and $\mathcal{L}_2$ will have different purposes: $\mathcal{L}_1$ will focus on the uncensored observations, while $\mathcal{L}_2$ will focus on both censored and uncensored observations, with the hope that its inclusion will enhance the model's ability to discriminate between defaulters and non-defaulters.

For the loss function $\mathcal{L}_2$ we will use the binary cross entropy, defined by

$$\mathcal{L}_{BCE} = -\sum_{i=1}^{N}[c_i \ln S(t|\mathbf{X}_i, \mathbf{\Theta}) + (1-c_i)\ln(1-S(t|\mathbf{X}_i, \mathbf{\Theta}))], \tag{4.5}$$

where c_i is a binary censoring variable indicating whether the observation is censored or not, i.e.

$$c_i = \begin{cases} 1 & \text{if observation } i \text{ is censored} \\ 0 & \text{if observation } i \text{ is uncensored} \end{cases} \tag{4.6}$$

Observe that minimization of 4.5 corresponds to minimizing the survival function at time t for uncensored observations, and maximizing it for censored ones. Through Equation 4.2 it can be seen that this corresponds to maximizing and minimizing the hazard rates of the censored and uncensored observations, respectively. This intuitively makes sense, as we in general want the model to predict higher hazard rates for customers at risk of defaulting.

As discussed in Chapter 2, the dataset being used suffers from a significant class imbalance, making machine learning models biased towards the majority class when making predictions. One way to combat this issue is by weighting the loss related to observations from the minority class heavier than that related to observations from the majority class. This way, errors in predictions related to the minority class are viewed as more severe by the model, and thus more emphasis is put on these observations in the optimization procedure. Therefore we introduce the weighted binary cross entropy loss function,

$$\mathcal{L}_{BCEW} = -\sum_{i=1}^{N} v_i[c_i \ln S(t|\mathbf{X}_i, \mathbf{\Theta}) + (1-c_i)\ln(1-S(t|\mathbf{X}_i, \mathbf{\Theta}))], \tag{4.7}$$

where v_i is the weight applied to observation i, and where

$$v_i = \begin{cases} \beta & \text{if } c_i = 1 \\ 1-\beta & \text{if } c_i = 0 \end{cases} \tag{4.8}$$

where $\beta \in (0,1)$.

Naturally, we also want to utilize the more complete information that lies in the observations where a default actually occurs. Considering that the default time is known, one way to do this is to maximize the likelihood that an event does occur at the observed event time, which amounts to maximizing the hazard rate at this time. This can be done through the minimization of the negative log-likelihood. The negative log-likelihood loss function is given by

$$\begin{aligned}\mathcal{L}_{NLL} &= -\ln \prod_{i=1}^{N} P(T_i = t_{N_i} | \mathbf{X}_i, \mathbf{\Theta})^{1-c_i} \\ &= -\ln \prod_{i=1}^{N} \left[h_i^{N_i} \prod_{j<N_i} (1-h_i^j) \right]^{(1-c_i)} \\ &= -\sum_{i=1}^{N} (1-c_i) \left[\ln(h_i^{N_i}) + \sum_{j<N_i} \ln(1-h_i^j) \right]\end{aligned} \tag{4.9}$$

The probability in the first line is raised to the power of $1-c_i$ in order to allow for the loss function to also be used for censored observations, for which the loss function then takes the value 0. As mentioned, this loss function does not directly involve the actual observed time to default. What it aims to do is maximize the hazard rate at time N_i, while minimizing those at earlier times. Therefore, when making survival time predictions based on for instance 100 days of customer data, it is not clear how one would go from the predicted hazard rates to an actual prediction of time to default. Indeed, the predicted hazard rates may be of questionable value in the first place, since the predictions will be based on a limited time span, and no default event will have been observed during this time. Thus, if the model is trained to simply recognize how a default event looks, its predictions on incomplete time series might be of little value. One could hope that, somehow, training the model in the way advertised will lead to the model generally predicting larger hazard rates for customers that default than for customers that do not, and larger hazard rates for customers that default within less time after being observed.

In an attempt to supervise more directly over the survival times, a third loss function is introduced, namely the mean squared error for uncensored observations, or MSE for short. It is defined by

$$\mathcal{L}_{MSE} = \frac{1}{\sum_{i=1}^{N}(1-c_i)} \sum_{i=1}^{N} (1-c_i)(z_i - \hat{z}_i)^2, \tag{4.10}$$

where z_i is the true remaining lifetime for observation i and $\hat{z}_i$ is the predicted remaining lifetime for the same observation. As the model outputs hazard rates with values between 0 and 1, a transformation of either the true event time or the predicted hazard rate is necessary for this loss function to work. The approach taken in this thesis is to scale the true survival times to be numbers between 0 and 1, with 0 being the smallest observed survival time, and 1 being some upper bound on the predicted survival times. Immediately a drawback is clear from this: the model will not be able to predict survival times smaller than the smallest observed survival time or larger than whatever bound is set. Fortunately, this is not a large problem in practice. If the smallest observed survival time is close enough to 0, any prediction of remaining lifetime around this value will simply be interpreted as a customer in dire need of assistance or guidance. It is impossible to know whether the customer will default in one day, two days or a week, and what we are really interested in is that the customer is at very high risk of defaulting within short time. One could also omit this problem entirely by simply stating that a remaining lifetime of 1 day corresponds to the value 0. In the case of the upper bound, we are not that interested in customers that aren't expected to default in a long time, for instance in the next two years. Therefore, one could simply choose a remaining lifetime above which one is not really concerned for the customer at the moment, to correspond to the value 1. This approach is further supported by the significant uncertainty that is inherent in credit card default predictions; there is no way of knowing exactly what is going to happen over any time span, and this uncertainty grows ever larger as the time span in question increases. A prediction of a scaled remaining lifetime of 1 can simply be interpreted as the customer currently not being at risk of defaulting at any time in the near future. Using this approach to scale the true value of remaining lifetime, we would scale the values to lie between 0 and 1 through

$$\zeta = \frac{z-a}{b-a}, \tag{4.11}$$

where z is the true lifetime, a is the minimum and b is the maximum lifetime, which are as in the reasoning above. Note that the value of ζ increases as the true remaining lifetime increases, which is the opposite of what we would expect from the hazard rate. Therefore it would make more sense to compare this value to one minus the hazard rate, as this quantity has the same expected behaviour as ζ. Thus the MSE would be given by

$$\mathcal{L}_{MSE} = \frac{1}{\sum_{i=1}^{N}(1-c_i)} \sum_{i=1}^{N}(1-c_i)(\zeta_i - (1-h_i))^2. \tag{4.12}$$

The notation $\hat{z}$ is then reserved for the actual predicted remaining lifetime, which would somehow be obtained from the predicted hazard rate. This could either be done through assuming some functional relationship between the two, or through using a kind of "look-up table", which would be based on hazard rate predictions and true survival times in the training set.

Another method one could use is to directly assume a functional relationship between hazard rates and remaining lifetime, and transform the predicted hazard rates into predicted remaining lifetimes *before* applying the loss function. Naively, it seems reasonable to assume a relationship on the form

$$z = \frac{\eta}{h}, \tag{4.13}$$

where z is the remaining lifetime, and h is the hazard rate. The parameter η would then be a parameter which would have to be tuned or in some way learned. Note that this is a very simple and quite limiting functional form which might not reflect the true relationship between the remaining lifetime and the hazard rates, and which is purely based on the idea that a larger hazard rate should correspond to a shorter time to default. Note that in this case, due to the fact that we are also applying the binary cross-entropy loss, we will essentially be calculating the loss on two different predicted values; on the predicted hazard rates directly and on the predicted remaining lifetimes. Of course, the decision on what functional relationship to assume could also be aided by the use of plots, which could give an indication of how the hazard rates and remaining lifetimes are connected. The next section will look into how the remaining lifetime will be obtained from the hazard rates in this thesis.

4.4 Predicting Time to Default

Since the output from the model is a predicted hazard rate, it is desirable to find some way to transform this into a predicted remaining lifetime, as this is ultimately what we are interested in. One way to do this is to follow an approach already introduced, namely assuming some functional relationship $z = g(h; \boldsymbol{\psi})$, where the parameters $\boldsymbol{\psi}$ can be learned. The difficulty of this approach lies in choosing the correct functional relationship. By utilizing the assumption that a higher hazard rate should strictly be associated with a shorter remaining lifetime, one could arrive at a naive relationship such as the one in Equation 4.13. A more refined approach could consist in comparing plots of the predicted hazard rates versus the true remaining lifetime with a range of candidate functions, and see which appears to be the best fit.

A different approach, which is the one taken in this thesis, is to compare the predicted hazard rates to the true remaining lifetimes in the training set, and create some prediction "rules" based on this. This can be done through grouping observations with remaining lifetimes that lie inside some interval $R = [a, b]$. Doing this for all uncensored observations gives a number M of intervals, $R_1 = [a_1, b_1]$, $R_2 = [a_2, b_2]$, ..., $R_M = [a_M, b_M]$, where the choice of the number M will depend on the problem at hand. In our case it is reasonable to let these intervals be disjoint, and have each interval start where the previous ended, so that we get $R_1 = [a_1, a_2)$, $R_2 = [a_2, a_3)$, ..., $R_M = [a_{M-1}, a_M]$. Note that we do not require that these intervals be of equal length. After grouping the observations in this way, we consider the predicted hazard rates of the observations within each group. The idea is then that, for a customer with predicted hazard rate h_r, the predicted time to default should be inside the interval in which the observations have an average predicted hazard rate closest to h_r. That is, we want to predict $\hat{z}_r \in R_s$, where s is the index of the interval satisfying the set criterion. The predicted value $\hat{z}_r$ could then simply be the mean value of the interval, $(a_{s-1}+a_s)/2$, or one could use an internal functional relationship that seems to fit well within each interval. In this thesis, we will simply be using the mean.

The way this transformation is performed in practice is that M intervals, or bins, are formed, and then for each bin i, a bin of predicted hazard rates $Q_i = [q_{i-1}, q_i)$ is defined. It is important to note here that these bins are decided based on the predicted hazard rates of the observations that are placed in the different lifetime bins R_i, however no exact method of how to set the bins will be used. To inform the decision of how to make up the bins Q_i, the hazard rates of the observations within the different bins R_i are used. This is done through the calculation of statistics such as the mean, median and standard deviation of these hazard rates for each bin R_i. Predicted remaining lifetime for customer r with predicted hazard rate h_r is then given by

$$\hat{z}_r = \frac{a_{i-1}+a_i}{2} \quad \text{if} \quad h_r \in Q_i. \tag{4.14}$$

That is, if h_r falls into the hazard rate bin Q_i, then the mean value of the lifetime bin R_i is predicted as the remaining lifetime of the customer with predicted hazard rate h_r.

This approach has its advantages. First of all, it is easy to implement and does not require any additional parameters to be trained. Secondly, it avoids the restrictions that assuming a functional relationship comes with, and it allows us to treat different intervals in different ways. For instance, the variance in the predicted hazard rate for certain intervals might be larger than for others, and thus we might be interested in a more refined transformation in these intervals, whereas we are happy to simply predict the mean value for the intervals with lower variance. Furthermore, it combines nicely with the functional approach, as one can assume different functional relationships within each interval, and thus potentially get a better approximation. Note that, due to time limitations, this flexibility will not be taken advantage of to any large degree in this thesis, and is mostly mentioned so that the reader is aware of it. Allowing the intervals to have different lengths also allows for more and smaller intervals where the predicted hazard rate changes quickly, and fewer and longer intervals where it changes slowly.

One downside of the approach, particularly when always predicting the mean value of the interval in question, is that internal differences among the observations in the interval are ignored. In particular, observations with predicted hazard rates in opposite ends of the Q_i-interval will have the same predicted lifetime, although it would certainly seem as though we would expect a shorter lifetime for the observation with the larger hazard rate. Another problem is that if q_i is the upper limit in interval Q_i and the lower limit in Q_{i+1}, then observations with predicted hazard rates $h+\epsilon$ and $h-\epsilon$ where ϵ is close to 0, might get very different predicted remaining lifetimes, despite having very similar predicted hazard rates. Increasing the amount of intervals would remedy these problems, however this would mean there are fewer observations in each interval, and thus the grounds on which the intervals are chosen will be weaker, from a data perspective. As will become clear in the result section, the dataset used in this thesis is quite difficult, and thus predicted hazard rates will vary greatly for customers with similar lifetimes. Placing customers within bins therefore seems a more trustworthy method than putting ones full trust in the exact value predicted. Furthermore, we are not necessarily interested in the precise number of days to default (although having this information would naturally be awesome), and we are mainly interested in approximately how long it will be before a customer defaults. Therefore, as long as the intervals are sufficiently well created, we do not really care that much about internal differences.

In order to use this method, the predicted hazard rates of the validation and test sets might benefit from a scaling after the training is complete. The reason for this is that the information used for training and the information used for prediction is different. Furthermore, there may be differences in the distribution of the sets, particularly because some observations are removed from the validation and test sets, and not from the train set. The precise scaling to use is not necessarily clear in this case, although it does at first seem reasonable to scale the predictions so that they have the same mean value. However, as mentioned, the distributions might vary, and so the optimal scaling might be a different one. Several scalings might therefore need to be explored in order to find one which gives reasonable results. To avoid overfitting to the test data, the scaling will be determined using the validation set, and then applied to the test set to report the final performance of the model.

As a final note to this section, it is pointed out that the method advertised, particularly in the form it is implemented in practice in this thesis, is quite ad-hoc and in no major way mathematically rigorous. This is a consequence of the method being largely based on visual inspection of plots of the predicted hazard rates against remaining lifetimes, as well as of inspection of summary statistics for the hazard rates in the different bins of remaining lifetime. All in all, the method is prone to a large amount of subjectivity, something that will become apparent in Chapter 6, where the results are presented.

4.5 Models and Datasets

As discussed in Chapter 2, it is desirable to detect potential defaulters as early as possible. By day 100, around 12% of the customers that default have already experienced their default event, and by day 113, this number is up to 25%. Therefore we want to base our models on datasets consisting of observations from relatively short time periods from the beginning of the customer relationships. Of course, by making this time period too small, we will be discarding large amounts of information that may be useful. Also, based on empirical experience from SpareBank1 Kreditt, customers might be hesitant to utilize their credit card at the start of the customer relationship, and thus the data from this time alone might give a distorted image of customers' true credit card activity. It is decided to train the models on datasets based on two different time periods, both going from the start of the customer relationship. The first dataset contains observations up to day 70, and the second one contains observations up to day 100. For the second dataset, customers that have defaulted before day 100 are removed, as the predictions will be interpreted as being the predicted remaining lifetime from day 100. By day 70, no customers have defaulted yet, so no customers need to be excluded from this dataset. For the training of all models, a train-validation-test split is performed on the dataset in question. The split will put 60% of observations in the train set, 20% in the validation set and 20% in the test set. This sort of split is quite common for training neural networks in general (Brownlee, 2020b).

Before being passed into the neural network, the dataset went through some preprocessing. Each variable was internally standardized to have zero mean and unit variance, as standardization is a common means of data pre-processing, both for neural networks and other machine learning methods, as mentioned in Chapter 3. This standardization was performed separately for each day of the customer relationships. Thus, each variable would have a slightly different distribution for each observation time.

As mentioned earlier in this chapter, two loss functions are combined to make up the final loss function through a convex combination, as presented in Equation 4.3. The model suggested by (Ren et al., 2018) uses the BCE and NLL loss, Equations 4.5 and 4.9, respectively. The first model used in this thesis will also use these loss functions, however with the slight modification of including class-based weights in the binary cross entropy function. This is done in order to combat the imbalance of the dataset. An additional model will be trained using weighted BCE along with MSE (Equation 4.12). As mentioned earlier, this is due to the original model's perceived inability to supervise directly over the actual remaining lifetime. Due to time constraints, this model will only be trained on the dataset with 100 observations. It is important to note that the data used during training will be quite different for the two models. When using the NLL loss, complete time series will be used, as this is necessary to maximize the hazard rate at the actual time of default. When using MSE loss, incomplete time series will have to be used, since if complete time series were used, every customer in the training set that defaults would have a ground truth of 0 days of remaining lifetime. Therefore, the same amount of data that is available in the validation set will be available in the training set when using the MSE loss. That is, for evaluation on the dataset with 100 days of observations, the training set will also consist of 100 days of observations for each customer.

4.6 Means of Evaluation

As mentioned in Chapter 3, the concordance index, or C-index, if you don't have much time, is a useful tool for assessing a model's discriminatory abilities. In Harrel et al. (1996) it is pointed out that if the discriminatory abilities of the model are good, then better calibration can be obtained without hurting discrimination. However, if the model has poor discriminatory abilities, then no amount of calibration can correct this. Therefore it can be considered reasonable to prioritize discriminatory abilities above predictive abilities, and thus the C-index will be used as the main evaluation metric in this thesis. Considering our choice of method for transforming predicted hazard rates into predicted lifetimes, it is reasonable to calculate the C-index from the predicted hazard rates rather than from the predicted lifetimes. This has the added benefit of allowing for easier supervision over the C-index during training of the model. As higher hazard rates should mean a shorter remaining lifetime, the value $1-h$ is used to compute the C-index. That way, a higher value of $1-h$ is associated with a longer remaining lifetime, and the C-index can be calculated as normal.

As was discussed in Chapter 3, finding appropriate metrics to evaluate how well the models are calibrated can be challenging. We mentioned the mean absolute error and the mean relative absolute error, and the problems with using these metrics. The Brier score in Equation 3.10 was also mentioned as a candidate, and at first glance it seemed like it could be a useful metric to include. There are some problems, however, the main one being that the Brier score, or at least the one introduced in Equation 3.10, assumes that all observations have survival estimates beyond their true survival times. This is not true in our case, as we base our validation on a certain number of days of data, and then want to predict remaining lifetime based on that information. Subjects that default before the time the predictions are made, are removed from the validation set. Thus, there would be no subjects satisfying the conditions of the first indicator function, namely that $t_i \leq t$ and $y_i = 1$, where t is the time at which the predictions are made, and t_i is the time of the last observation for customer i. Another drawback is that an estimate of the censoring distribution is required for weighting. This can be difficult to estimate, as is also mentioned in Kvamme (2019). Because of the difficulty of finding appropriate metrics for evaluating how well the model is calibrated, we will limit ourselves to reporting the mean relative absolute error and provide residual plots of the predicted lifetimes. Hopefully, this will at least give some ground for comparing models to each other, and decide on which one is better calibrated.

As a way of evaluating the performance of their models, SpareBank 1 Kreditt are interested in how well the predictions hold up for the coming year. That is, does the model in general predict a lifetime of more than one year for subjects that do not default within the next year? And does it predict a lifetime shorter than a year for those that do? This metric is implemented by simply providing all customers with an additional label,

$$d_i^{365} = \begin{cases} 1 & \text{if } z_i \geq 365 \\ 0 & \text{if } z_i < 365 \end{cases}$$

and similarly for the predicted lifetimes,

$$\hat{d}_i^{365} = \begin{cases} 1 & \text{if } \hat{z}_i \geq 365 \\ 0 & \text{if } \hat{z}_i < 365 \end{cases}$$

This approach allows to view the problem as binary classification; observations with a lifetime of less than a year are classified to class 0, and those with a lifetime larger than a year are classified to class 1. As a consequence, metrics such as balanced accuracy and Matthews correlation coefficient, introduced in Chapter 2, can easily be computed to asses the predictions. For the defaulters this classification is straightforward. For the non-defaulters, however, some distinctions need to be made. For non-defaulters that are observed for more than a year after the time of prediction, it is also straightforward, as the model has then predicted correctly if it predicts a survival time of more than a year. For non-defaulters that are not observed for a full year after the time of prediction, however, we do not know for sure that the true lifetime will be larger than a year. Thus, these customers are excluded from this classification approach. In order to still get an impression of how well the model performs for these customers, we will report the percentage of these customers for which the model predicts a lifetime larger than or equal to the observed lifetime. This percentage will be denoted by $q_{\hat{z} \geq z}$.

4.7 Hyperparameters

As mentioned in Chapter 3, hyperparameter tuning is a central but often time-consuming part of constructing a neural network model. Due to the large amount of time required to do proper searches, hyperparameter tuning will be performed only to a very small degree in this thesis. As far as possible, default hyperparameters will be used, and learning curves will be used to assess the learning rate. Different values will be tried for some hyperparameters to see whether performance increases or decreases. Only one hyperparameter will be varied at a time, but no grid search-type structure will be applied. That is, if changing the value of one hyperparameter, call it $\mathcal{H}_1$, from value a_1 to a_2 leads to increased performance, and then changing the value of another hyperparameter $\mathcal{H}_2$ from value b_1 to b_2 further increases performance, then the combination (a_1, b_2) will not be tried. In other words, potential interactions between hyperparameters will not be explored. The absence of a properly structured hyperparameter tuning procedure will be discussed further in Chapter 7.

Chapter 5

Data Analysis

An initial analysis of the dataset can give valuable insight into how it is structured and what variables may be of importance. For models where the learned parameters are easily interpreted, it can also serve as a sanity check for the values of the parameters, e. g. whether a larger value of the variable in question is associated with a certain change in the response variable. In the case of LSTMs and neural networks in general, evaluating parameter values and determining what effect they have on the predictions made by the network can be difficult, due to the highly complex structure of the model. Therefore, the goal of this section will be to obtain insights into and get familiarized with the dataset. We will have a look at how both categorical and numerical variables affect the two response variables mentioned in Chapter 2, and the analysis will be split into two parts: static analysis and time series analysis. For variable definitions, please refer to Table A.1 in Appendix A.

5.1 Static Analysis

By static analysis is meant looking at variables at only one certain point in time. This can be useful for variables that don't change, or change in a predictable way, throughout the experiment, such as a customer's gender, age or time to default from the beginning of their customer relationship. This section will look at customers at the beginning of their customer relationship, and analyse a handful of variables through the use of histograms.

As there are mainly two response variables of interest in the dataset, one could be interested in analyzing how the different explanatory variables correlate with both. In the case of a categorical variable, it can be interesting to see how the numbers of defaulters and non-defaulters are spread across the different categories. A simple way to express this is through a histogram. A histogram is a simple plot showing the amount of subjects that fall into each category of some variable, such as the number of males and females, or the number of subjects that belong to different geographical areas. By first splitting the subjects into those that do experience an event, i.e. 'DC2Ind'=1, and those that don't, i.e. those with those with 'DC2Ind'=0, one can then create one histogram for each group for the same explanatory variable. The degree of variation in how these histograms look can then be used to get an initial impression of whether the variable in question is helpful in explaining differences between the two groups.

The Figures 5.1, 5.2 and 5.3 show histograms for the three binary variables 'GENDER_NAME', 'HAS_DIRECT_DEBIT_AGREEMENT_IND' and 'HAS_ESTATEMENT_AGREEMENT_IND', respectively. From Figure 5.1 it is clear that males seem more likely to experience a default event than women. Of the 4379 women in the dataset, 423 experience a default event, giving a default rate of around 0.0966. For the men, the rate is 0.168, with 1161 of the 6913 males in the dataset defaulting during the experiment. Thus we would expect this variable to be of some importance to our model. From Figure 5.2, we first of all observe that there is a large difference in the number of people that do have a direct debit agreement with the bank and the number of people that don't. Only 986 customers have a direct debit agreement, and of these, 48 experience a default event during the experiment, giving a rate of 0.0487. There are 10306 customers that do not have a direct debit agreement, 1536 of whom experience a default. This gives the higher rate of 0.149. Finally, Figure 5.3 shows a large difference in the two histograms. Of the 8654 customers that do have an e-statement agreement with the bank, 556 customers, corresponding to a percentage 0.0642, experience a default event. On the other hand, among the 2638 that do not have such an agreement, 1028 experience a default event, giving a rate of 0.390. This is the only variable of the three that has more defaulters in the minority category than the majority category. It certainly seems that this variable might be helpful in our model.

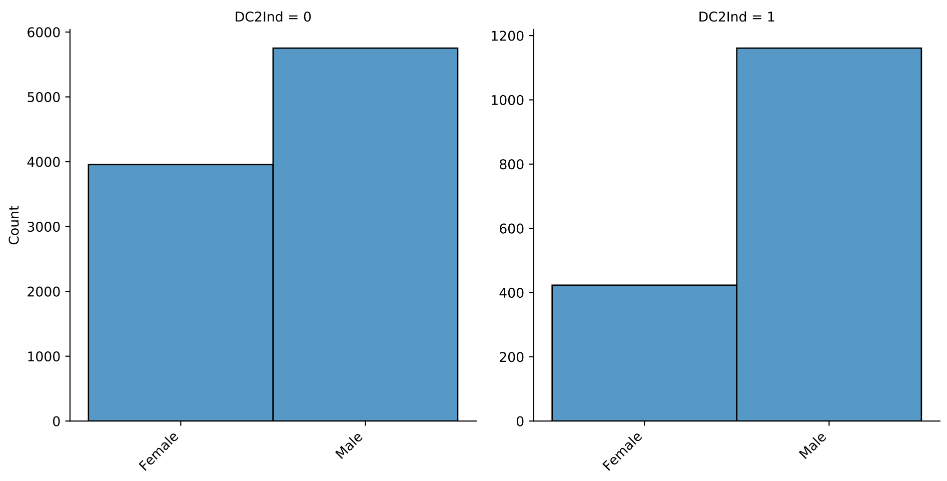

Figure 5.1: Histogram showing the number of males and females that default (right) and do not default (left) during the experiment.

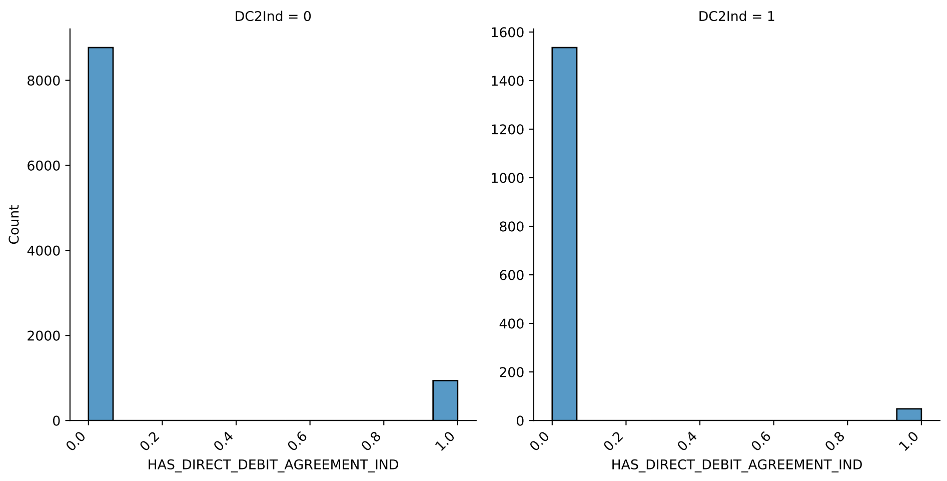

Figure 5.2: Histogram showing the number of subjects with and without direct debit agreement that default (right) and do not default (left) during the experiment.

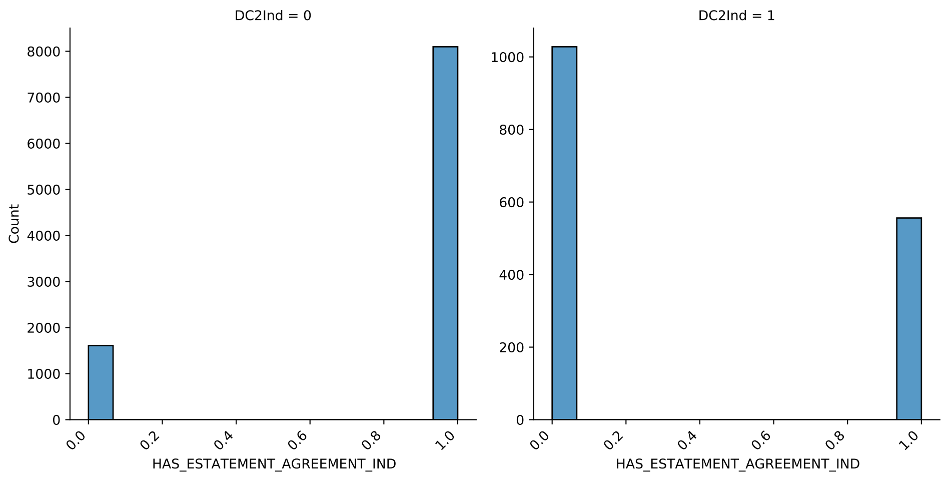

Figure 5.3: Histogram showing the number of subjects with and without e-statement agreement that default (right) and do not default (left) during the experiment.

We now take a look at some histograms for the variable 'CustomerAge', shown in Figure 5.4. Observe that there is an over representation of younger people in the dataset, and that around 6100 customers are less than 30 years old. The differences between the two histograms are not particularly large. It looks like the histogram of the defaulters is slightly heavier in the left part, indicating that younger customers are more likely to default than older ones. However, when looking at the data, it turns out that the difference is small. Looking at the actual numbers, it turns out that among the defaulters, around 51% are below 30 years old, and among the non-defaulters, this number is around 55%. Customers below 25 years make up around 23% of the defaulters, and around 27% of the non-defaulters. The very youngest customers appear to be supporting the initial hypothesis, as customers below 20 years old make up around 11.3% of defaulters, and around 10.5% of non-defaulters, however clearly the difference here is very small. All in all it does not look like the age of the customers in this dataset will have too much impact on their default rate.

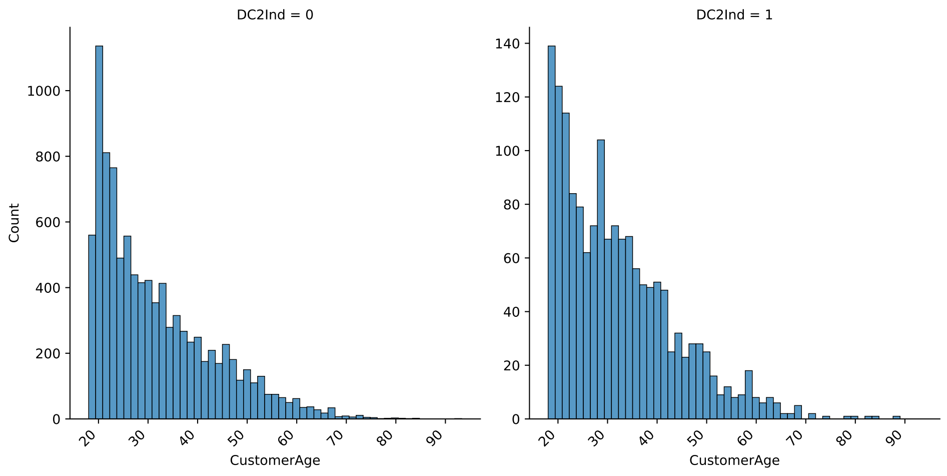

Figure 5.4: Histogram showing the age of subjects that default (right) and do not default (left) during the experiment.

Lastly in this static analysis we will have a look at the distribution of lifetimes for the uncensored observations, i.e. for the defaulters. The distribution is visualized in two ways: through a histogram and through what is called a kernel density estimate. A kernel density estimate, or KDE for short, is a continuous estimate of the probability density function of a variable. The estimate is based on some chosen kernel, which in this case is the Gaussian kernel (Kim and Scott, 2012). The visualizations are shown in Figure 5.5. Both the histogram and the KDE indicate that a large amount of customers default before day 150 of their customer relationship. To be more precise, close to 40% of the customers that default, do so before day 150. From day 150 we still see that less customers default the longer the customer relationship has been. After one year, nearly 80% of the defaults have occurred. These observations may indicate that customer relationships in general stabilize over time, and that many of the customers in the dataset that do default, exhibit "at-risk" behaviour from early on in their customer relationship.

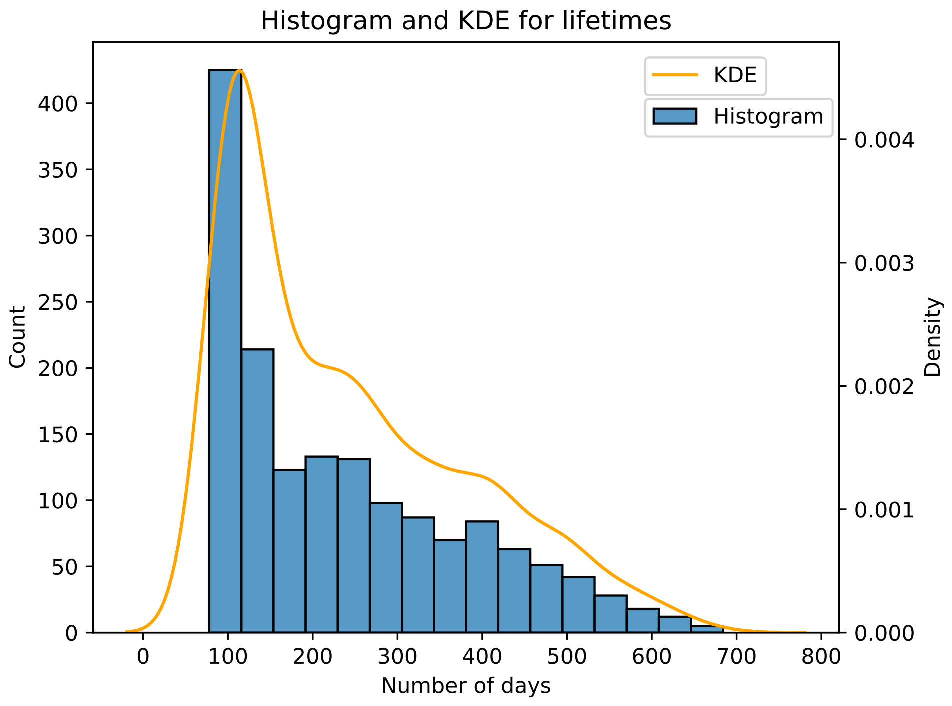

Figure 5.5: Histogram and kernel density estimate showing the remaining lifetime of uncensored subjects in the experiment. The left y-axis belongs to the histogram, and the right y-axis belongs to the KDE.

5.2 Time Series Analysis

The longitudinal data analysis is concerned with looking at entire time series rather than just summary statistics. Such an analysis can give useful insights into how different variables change over time, and how the change for defaulters compares to the change for non-defaulters. This section will mainly present plots trying to reveal such differences.

There are several variables in the dataset where the change over time might be of interest. Recall that there are around 28 numerical variables associated with the credit card activity of customers. Previous aggregated data analysis has shown that, based on kernel density estimates, the variables 'OVERDUE_AMT', 'IEL_AMT' and 'CASH_BALANCE_AMT' in particular are of interest in regard to revealing differences between defaulters and non-defaulters (Holte, 2021). In addition, the variables 'UtilizationL3', 'UtilizationL12' and 'AvgRevBalL3onL12' are expected to give interesting information about the customers. These variables will be explored in the first subsection of this section. Former analysis indicates that the variables giving a customer's spending in different areas do not explain much of the differences between defaulters and non-defaulters, and thus they have been viewed as less important variables. In the second subsection they will be revisited to see if their development over time can be of use.

Before looking at the time series plots, a short explanation of how these were created is warranted. For each variable, the values are collected in a two dimensional data frame $\mathcal{D}$, where each row $\mathcal{D}_{i:}$ corresponds to the values associated with one customer. Each column $\mathcal{D}_{:j}$ corresponds to the values of the variable for all customers on the jth day of their customer relationship. The element $\mathcal{D}_{ij}$ then corresponds to the value of the variable on the jth day of the customer relationship of customer i. The plots are then created by plotting each of the time series $\mathcal{D}_{i:}$ for $i \in \{1, 2, ..., N\}$ in grey, and computing summary statistics such as the mean and median values based on $\mathcal{D}_{:j}$ for all "living" customers at each time $t \in \{1, 2, ..., t_{\text{Max}}\}$, where t_{Max} is the longest observed time to default or censoring. The quantiles are computed in the same way as the mean and median, and they are visualized through two lines, giving upper and lower bounds within which a certain percentage of observations lie. These lines will be denoted by Q followed by the quantile they represent, which will either be 80 or 90. The y-axes have been scaled to include as many complete time series as possible, while still allowing for visual inspection of the summary statistics. Therefore some particularly large values have been cut.

Note that as time progresses, more and more customers disappear from the experiment, and as a consequence the statistics computed at later times have a smaller data foundation than the ones computed at early times. Thus, the statistics for late times should not be paid too much attention too, and as we will see, they will often exhibit a somewhat strange behaviour. Refer to Table 2.2 from Chapter 2 for an indication of how many customers default at different time intervals. Table 5.1 shows the number of customers that have not defaulted or been censored by different numbers of days into the customer relationship. However, due to the fact that no observations are registered during the weekends or holidays, each customer will have fewer observations than the number of days they are observed. As an example, the customers with the longest reported observation time have an observation time of 788 days, however the largest amount of observations for any single customer is 546. Therefore, what the time series really represent is the value at each time of observation, not at every single day of the customer relationship. The x-axis in the plots presented in this section should therefore not be viewed as showing day 1, day 2, day 3 and so on of the customer relationships, but rather as showing observation 1, observation 2, observation 3 and so on. This will still provide the insight we seek to gain with this data analysis. Table 5.2 shows how many customers have not defaulted or been censored by different amounts of observations, and can thus be used as a reference for the data foundations for the statistics computed at different times.

Day	Customers remaining
0	11292
100	11098
200	10505
300	9309
400	7643
500	5858
600	3771
700	1525
787	30

Table 5.1: Overview of the amount of customers that have not defaulted or been censored by different times of the experiment.

x value	Customers remaining
0	11292
50	11292
100	10694
150	10444
200	9497
300	7142
400	4163
500	1092
547	30

Table 5.2: Overview of the amount of customers that have not defaulted or been censored by different amounts of observations during the experiment.

5.2.1 General Transaction Variables

We start off by looking at some variables that former analysis on an aggregated dataset has indicated may be of interest. Firstly, we look at the time series plot of the variable 'BALANCE_AMT', which can be found in Figure 5.6. Note that this variable may in fact have negative values, which indicate a positive balance, i.e. that the customer is owed money by the credit card institution. What is revealed by the 'BALANCE_AMT' plot is that, unsurprisingly, non-defaulters spend less money through the use of credit cards than defaulters on average. At first glance, this gives the indication that people that spend more money through the use of credit cards are more likely to default than those that don't, which clearly is not a strange idea. However, this does not take into account the fact that many of the non-defaulters are inactive users, i.e. they own a credit card but rarely use it. These users contribute to lowering the mean value of the variable for the non-defaulters, and are likely to distort the image one would get from looking at only customers that are actually active. Setting a sort of activity threshold and then creating the same plot using only the customers that end up above this threshold would probably give a more correct and informative picture. The customers that almost never use their credit card are highly unlikely to default anyway, so removing them from the analysis would most likely not result in any significant loss of information. This goes for all variables analysed in this section.

We now go on to look at the plots of the variables 'CASH_BALANCE_AMT', 'OVERDUE_AMT' and 'IEL_AMT', which can be found in Figures 5.6, 5.7 and 5.8. 5.9. First of all, note that the lower quantiles, i.e. the 10% quantile and the 5% quantile, are in general very small, and mostly equal or close to zero for both defaulters and non-defaulters for all three variables. This further emphasizes the significant amount of inactive customers. Also note that this is not the case for 'BALANCE_AMT' for the defaulters. Observe that, for the defaulters, the way the mean value changes over time is quite similar for the three variables. This change is also quite similar to that in 'BALANCE_AMT'. There is a steep increase over the first ca. 50 observations, before there is a slight drop, followed by a slow increase towards observation 400. The variables 'BALANCE_AMT', 'CASH_BALANCE_AMT' and 'IEL_AMT' also have medians that behaves very similarly. These similarities are not very surprising, as the four variables mentioned are highly correlated with each other, something that was found in my project thesis (Holte, 2021), and which can easily be understood from the definition of the variables. An increase in 'CASH_BALANCE_AMT' leads to an increase in 'BALANCE_AMT', which again leads to an increase in 'OVERDUE_AMT' and 'IEL_AMT' if the balance is not paid down in time. Comparing the time series of the defaulters to that of the non-defaulters, we again see the expected pattern that the defaulters in general spend more than the non-defaulters, and that they on average owe a larger sum on which they pay interest. They also in general have a much larger sum that is overdue. Another interesting observation for all the four variables mentioned is that the median is almost always smaller than the mean, both for the defaulters and the

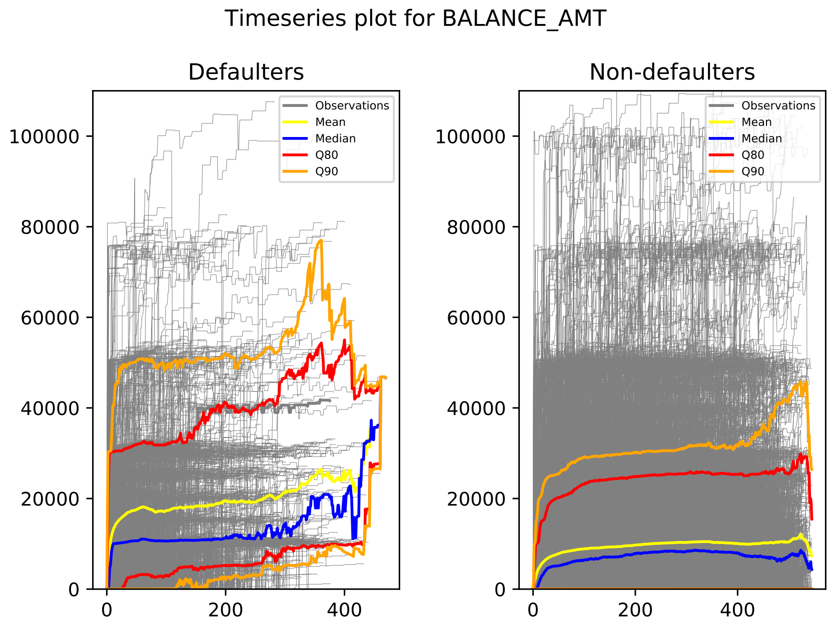

Figure 5.6: Time series plot for 'BALANCE_AMT' variable. The x-axis gives the number of days into the customer relationships, and the y-axis gives the value of the variable in Norwegian kroner (NOK). The pairs of lines Q80 and Q90 give bounds within which 80 and 90% of the observed values are, respectively.

non-defaulters. This might be due to the presence of very large values which inflate the mean, and it suggests that the distributions of the variables have quite heavy right tails.

The next time series to be analysed are the ones for the variables 'UtilizationL3', 'UtilizationL12' and 'AvgRevBalL3onL12'. The plots are shown in Figures 5.10, 5.11 and 5.12. For the first two, we again observe that both the mean and median values are larger for the defaulters than for the non-defaulters, indicating that defaulters on average spend a larger amount of their credit limit than non-defaulters. For 'UtilizationL3', we see that the median value for the defaulters is around 1 for essentially all observation times. This indicates that around half of the defaulters have maxed out their credit cards the last three months, or even borrowed beyond their credit limit. Both the median and the mean show a step function-like increase around observation 60 for the defaulters, and a less steep increase around the same time for the non-defaulters. This time is around where the first three months of the customer relationships are over, and thus the first time at which the average over the last three months can be properly reported. This observation emphasizes the more cautious nature of the non-defaulters, as they appear more hesitant to max out their credit cards during the beginning of their ownership. Again, however, inactive customers contribute to the deflation of the statistics. Note that the means and medians for the 'UtilizationL12' variable

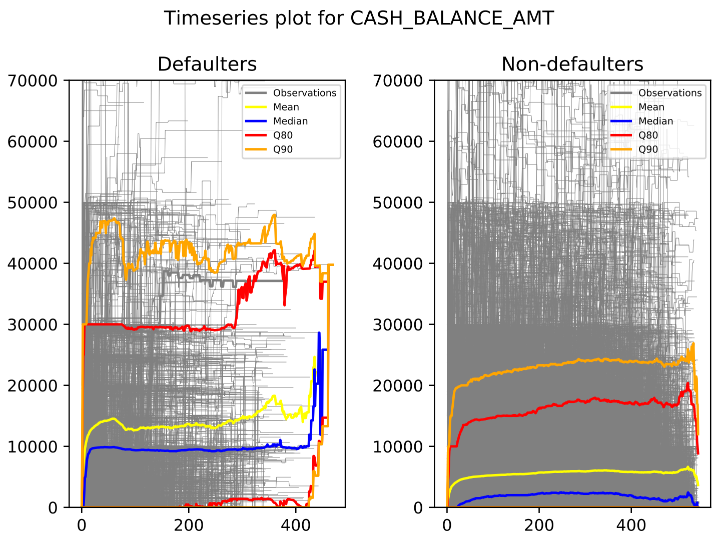

Figure 5.7: Time series plot for 'CASH_BALANCE_AMT' variable. The x-axis gives the number of days into the customer relationships, and the y-axis gives the value of the variable in Norwegian kroner (NOK). The pairs of lines Q80 and Q90 give bounds within which 80 and 90% of the observed values are, respectively.

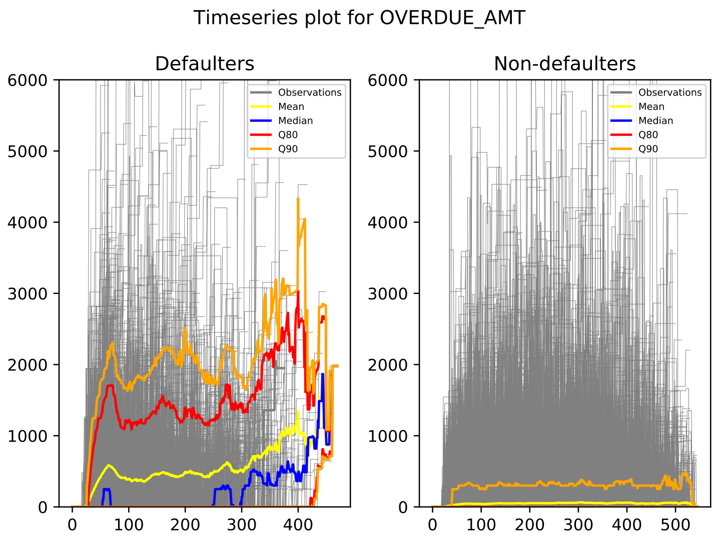

Figure 5.8: Time series plot for 'OVERDUE_AMT' variable. The x-axis gives the number of days into the customer relationships, and the y-axis gives the value of the variable in Norwegian kroner (NOK). The pairs of lines Q80 and Q90 give bounds within which 80 and 90% of the observed values are, respectively.

start increasing almost from the start of each customer relationship and increase steadily towards 1, which indicates that the two variables are reported differently. Also observe that for the two 'Utilization' variables, the median is larger than the

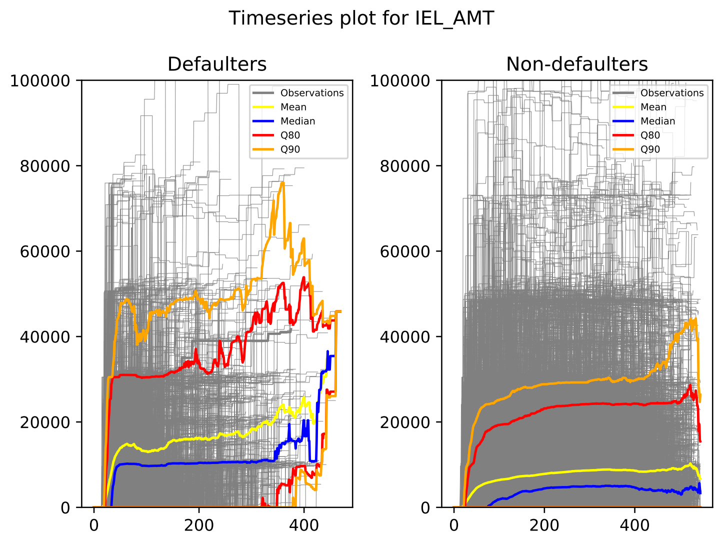

Figure 5.9: Time series plot for 'IEL_AMT' variable. The x-axis gives the number of days into the customer relationships, and the y-axis gives the value of the variable in Norwegian kroner (NOK). The pairs of lines Q80 and Q90 give bounds within which 80 and 90% of the observed values are, respectively.

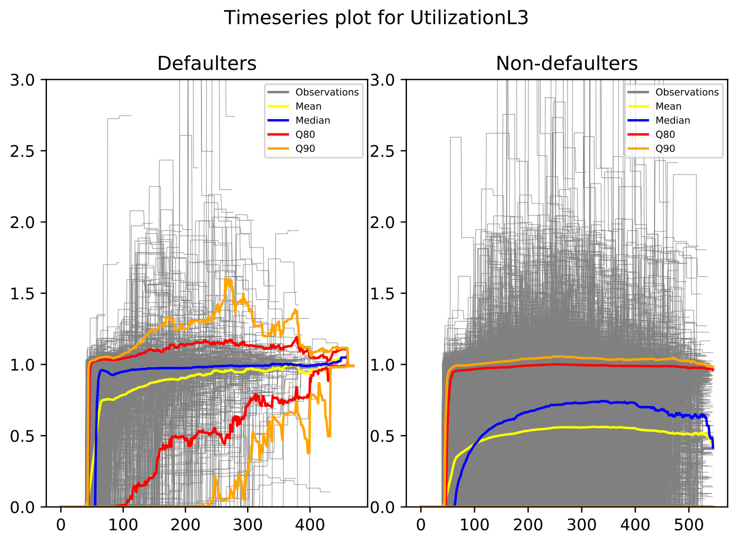

Figure 5.10: Time series plot for 'UtilizationL3' variable. The x-axis gives the number of days into the customer relationships, and the y-axis gives the value of the variable. The pairs of lines Q80 and Q90 give bounds within which 80 and 90% of the observed values are, respectively.

mean value for most of the time, indicating a heavier left tail in the distributions. The plot of 'AvgRevBalL3onL12' shows a quite similar behaviour for defaulters and non-defaulters, as the median is around the same value and changes in about

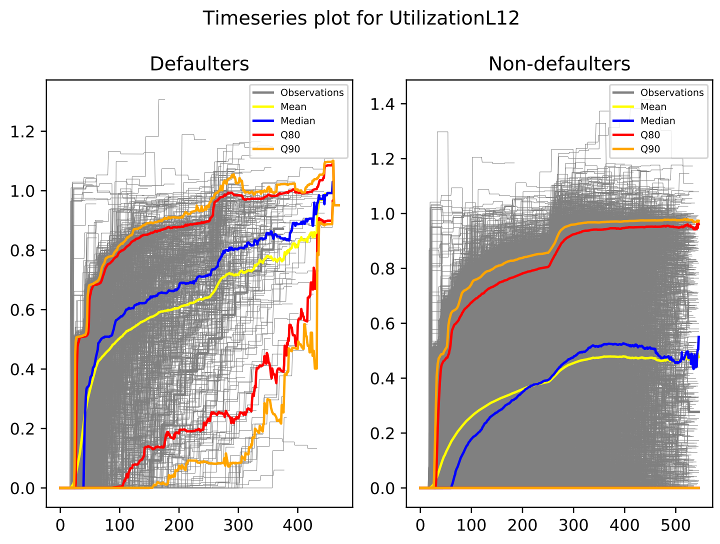

Figure 5.11: Time series plot for 'UtilizationL12' variable. The x-axis gives the number of days into the customer relationships, and the y-axis gives the value of the variable. The pairs of lines Q80 and Q90 give bounds within which 80 and 90% of the observed values are, respectively.

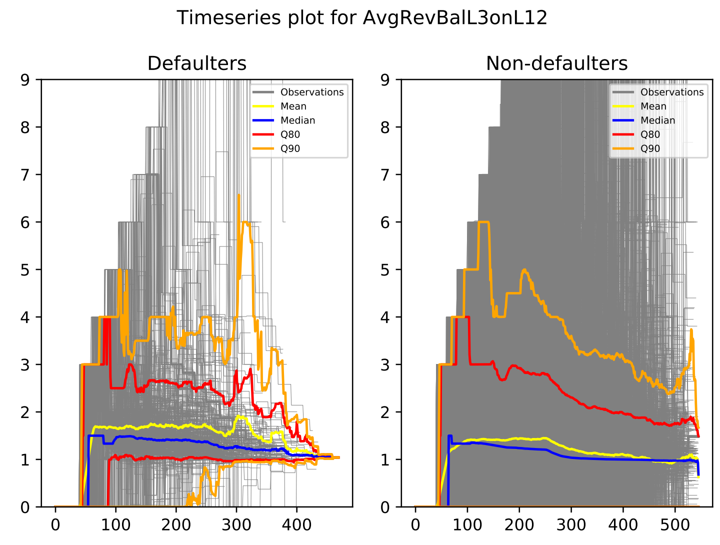

Figure 5.12: Time series plot for 'AvgRevBalL3onL12' variable. The x-axis gives the number of days into the customer relationships, and the y-axis gives the value of the variable. The pairs of lines Q80 and Q90 give bounds within which 80 and 90% of the observed values are, respectively.

the same way. The mean values are also quite similar, and both the medians and means do not vary much in time. This indicates that the average revolving balance remains close to constant over time, with a tendency to decrease. We also see the time series of several customers following a stair-like development, which indicates that their revolving balance increases every month. These might be customers who only pay back the lowest possible amount each month, and then proceed to increase their debt by further using their credit card. Notice that for both groups, the mean is larger than the median most of the time, and it deviates more for the defaulters than the non-defaulters, indicating heavier tails in the distribution for this group.

5.2.2 Category-specific Transaction Variables

As mentioned initially, previous analysis of aggregated versions of the dataset used in this thesis indicated that customers' spending in different categories would not be of much help in separating the defaulters from the non-defaulters. As a full longitudinal data analysis is still warranted for these variables in order to see if this hypothesis remains, we will now take a look at time series plots for several category-specific spending variables. We begin with taking a look at the time series for the variable 'SumQuasiCashL3M', which are shown in Figure 5.13. Former analysis has indicated that this might be the most interesting of the category-specific spending variables. From the plot it is observed that, again, defaulters on average spend more within this category than non-defaulters, but only up to around observation 300, where there is a marked drop in the expenditure of the defaulters. It is worth to note that, by observation 300, only around 190 defaulters remain, whereas almost 7000 non-defaulters remain. An interesting observation is that the mean value for the non-defaulters actually crosses the upper 80-percentage bound. Still, the mean value here is very low, and it appears as though the reason that this occurs is that there are very many customers that spend very little or no money within the category, meaning the 80-percentage upper bound ends up quite close to zero. This is in fact the typical story for these variables. The median, for instance, is always zero for both groups.

We now take a look at some of the other categories. We will only be looking at some selected variables, as looking at all of them is both time consuming and, frankly, a bit boring. The variables shown are the ones that the author found the most interesting. The variables shown are 'SumFOOD_STORES_WAREHOUSEL3M', 'SumINTERIOR_FURNISHINGSL3M', 'SumOTHER_RETAILL3M', 'SumOTHER_SERVICESL3M' and 'SumRESTAURANTS_BARSL3M', and these are shown in Figures 5.14, 5.15, 5.16, 5.17 and 5.18. Firstly, observe that, in contrast to 'SumQuasiCashL3M', it is the non-defaulters that on average spend the most within these categories. The mean values all behave in a similar manner, seeing a steep increase quite early in the customer relationships, and then decreasing slowly from around observation 60 or 70. As mentioned in the discussion of 'SumQuasiCashL3M', the median is always zero, due to the majority of customers spending little to no money within each category. We also observe that the upper 80-percentage bound creeps below the mean value for 'SumIN-

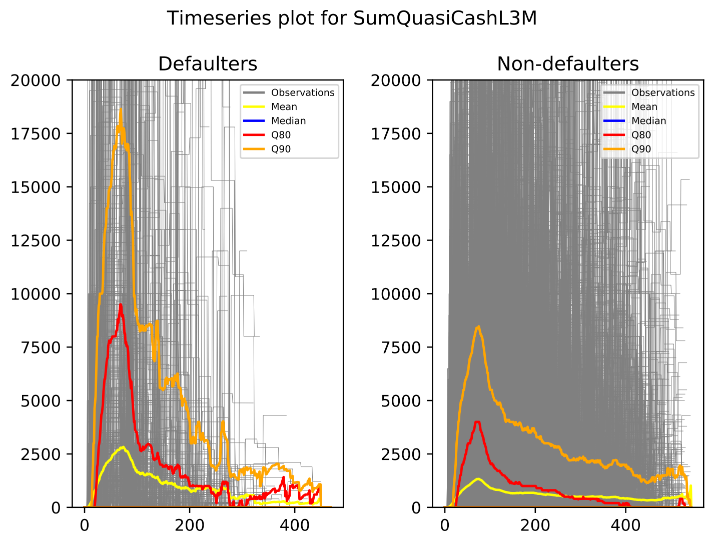

Figure 5.13: Time series plot for 'SumQuasiCashL3M' variable. The x-axis gives the number of days into the customer relationships, and the y-axis gives the value of the variable in Norwegian kroner (NOK). The pairs of lines Q80 and Q90 give bounds within which 80 and 90% of the observed values are, respectively.

TERIOR_FURNISHINGSL3M' and 'SumOTHER_SERVICESL3M', further indicating the absence of spending with a large majority of customers.

There are some variables remaining whose time series have not, and will not, be explored. The reason for this is mostly that the plots are either considered of little interest by the author, meaning that there is either few differences between the time series for the defaulters and the non-defaulters, or that there are very few non-zero observations of the variables. As a comment on this, and also as a final note to this chapter, I wish to re-iterate the impact that inactive customers have on the time series plots. In practice, these customers are not very interesting, as they very rarely experience defaults and also are not customers that institutions earn the most money from (low expenditure means loans can more easily be paid back in full by the due date, and interest earnings are low due to the small sums borrowed). It would therefore be interesting to explore how the plots would look with inactive customers excluded from the analysis. The reason this has not been done is that the time period for writing this thesis is limited, and as was mentioned in the beginning of this chapter, the data analysis is done mostly to give insight into the dataset, and is not expected to be of any help with regard to setting up the model or analyzing the results. Another reason is that it is not straightforward to define a threshold for inactivity. Removing all customers with 'BALANCE_AMT' equal to zero for all observations could be a place to start, however it would be reasonable to extend the concept of inactivity to also include customers with very low spending over time.

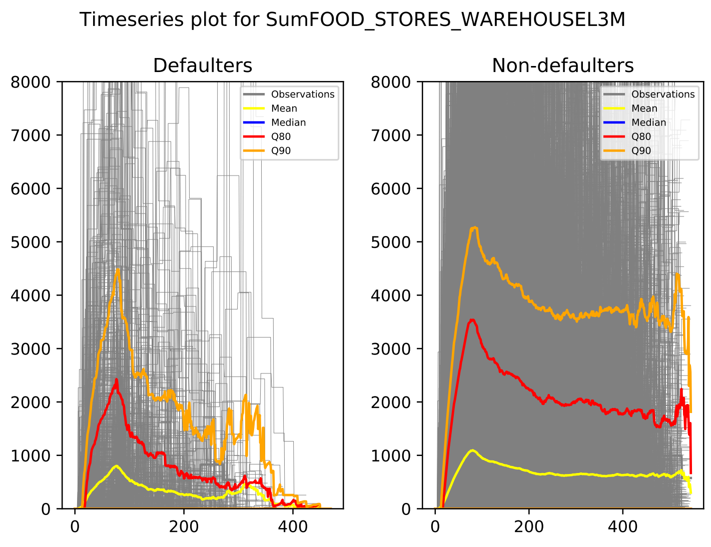

Figure 5.14: Time series plot for 'SumFOOD_STORES_WAREHOUSEL3M' variable. The x-axis gives the number of days into the customer relationships, and the y-axis gives the value of the variable in Norwegian kroner (NOK). The pairs of lines Q80 and Q90 give bounds within which 80 and 90% of the observed values are, respectively.

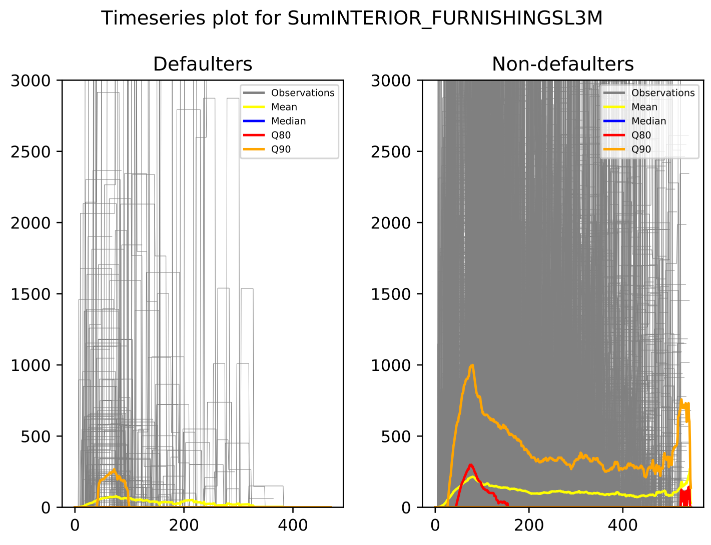

Figure 5.15: Time series plot for 'SumINTERIOR_FURNISHINGSL3M' variable. The x-axis gives the number of days into the customer relationships, and the y-axis gives the value of the variable in Norwegian kroner (NOK). The pairs of lines Q80 and Q90 give bounds within which 80 and 90% of the observed values are, respectively.

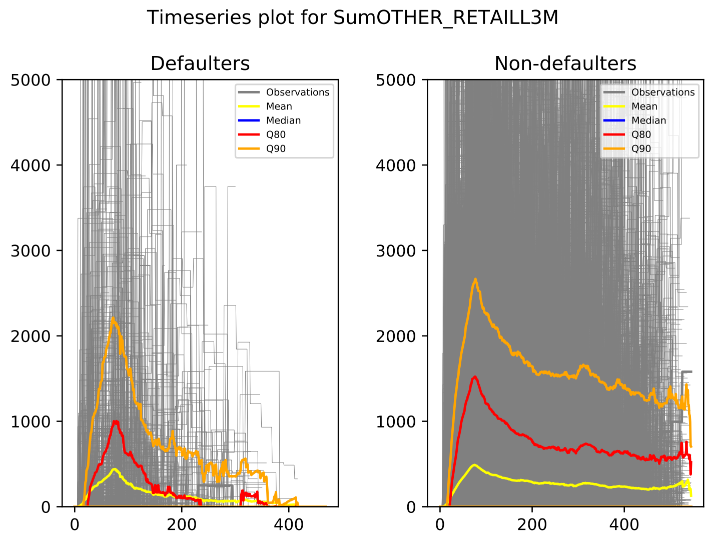

Figure 5.16: Time series plot for 'SumOTHER_RETAILL3M' variable. The x-axis gives the number of days into the customer relationships, and the y-axis gives the value of the variable in Norwegian kroner (NOK). The pairs of lines Q80 and Q90 give bounds within which 80 and 90% of the observed values are, respectively.

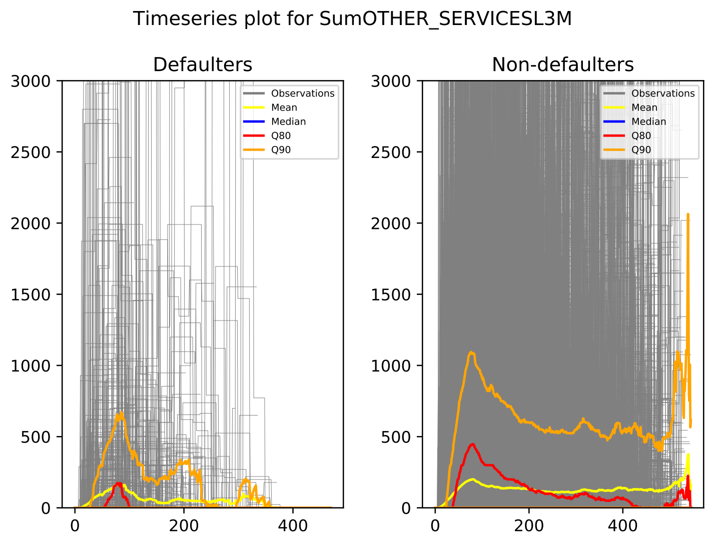

Figure 5.17: Time series plot for 'SumOTHER_SERVICESL3M' variable. The x-axis gives the number of days into the customer relationships, and the y-axis gives the value of the variable in Norwegian kroner (NOK). The pairs of lines Q80 and Q90 give bounds within which 80 and 90% of the observed values are, respectively.

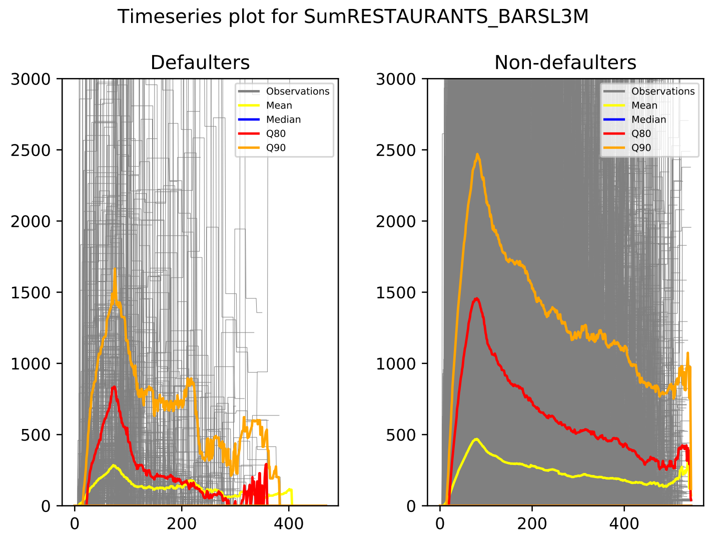

Figure 5.18: Time series plot for 'SumRESTAURANTS_BARSL3M' variable. The x-axis gives the number of days into the customer relationships, and the y-axis gives the value of the variable in Norwegian kroner (NOK). The pairs of lines Q80 and Q90 give bounds within which 80 and 90% of the observed values are, respectively.

Chapter 6

Analysis of Results

In this chapter, the results produced by the models introduced in Chapter 4 are presented and analyzed. The chapter is divided into two sections, where the first section covers the results from the models trained with negative log-likelihood and weighted binary cross entropy loss. The second section was meant to present the results from the models trained with mean squared error and weighted binary cross entropy loss, however, as is briefly explained in the section, results for these models were not obtained. The variables included in the trained models can be found in Table A.1 in Appendix A. The code used to obtain the results is available at `https://github.com/HaakonHolte/Masters_Thesis_Code`.

6.1 NLL and WBCE Loss

Firstly, we present the results obtained by using the combination of loss functions as proposed by Ren et al. (2018), namely the negative log-likelihood and binary cross entropy, where in addition a weighting is applied to the latter in an attempt to counteract the imbalance of the dataset. The loss functions are combined through a convex combination as in Equation 4.3, and the hyperparameter α, which will be used later in this section, refers to the α in this equation. Models were trained using several hyperparameter combinations, a full overview of which can be found in Table B.2 in Appendix B. Based on the achieved C-index scores, one model was selected and further analysed. This was done separately for the datasets of 100 days and 70 days of information. Thus, in this section, the results from two models will be presented. For simplicity, we will write NLL for the negative log-likelihood loss, and WBCE for the weighted binary cross entropy loss.

100 days

The hyperparameter values for the chosen model for the dataset based on 100 days of information are shown in Table 6.1. This model achieved a C-index of

Hyperparameter type	Hyperparameter	Value
Network	Hidden size	120
	Layers	4
	Dropout probability	0.5
Dataloader	Batch size	256
	Train-val-test percentages	72-14-14
Optimizer	Learning rate	0.01
	Momentum	0.5
Loss	Loss weight α	0.6
	BCE weight β	0.2

Table 6.1: Hyperparameter values for the model using NLL and WBCE loss on dataset with 100 days.

0.72 on the validation set, and so in terms of concordance it showed the best performance among the models trained on this dataset. The C-index achieved on the test set was 0.67. The model was trained using 6774 customers in the training set, 1338 customers in the validation set and 1323 observations in the test set. The discrepancy between the validation and test set comes from the train-validation-test split occurring prior to the removal of customers that default before 100 days have passed. Recall that such customers are only removed from the validation and test sets. Unfortunately, an error in the code was discovered after the results had been obtained, and at a time where it was too late to retrain the models. This error is lead to the percentage of customers in the validation and test sets being lower than what was intended, and as is expressed in Table 6.1, the validation and test sets make up 14% each of the customers, rather than 20%, which was the percentage originally intended.

The hazard rate predicted at day 100, or at the day of the final observation for customers not making it 100 days without defaulting, was the value analysed for each customer. The analysis showed that the mean of the predicted values on the train set was lower than the mean of the predicted values on the validation and test sets. Simultaneously, the median predicted value on the train set was larger than the one on the other two sets. This information is shown in Table 6.2. Scaling, as explained in Section 4.4, was now explored using the validation set. Using only the ratio of means for scaling lead to the predicted hazard rates in the validation set being smaller than what was already the case. This lead to very low predicted hazard rates for many customers, which is expected, as the median of the validation set was lower than that of the train set. In order to avoid this, another scaling method was attempted, which also incorporated the median value, and which was intended to work as a compromise between the mean and the median. The predictions were scaled by the average of the ratio of means and the ratio of medians. Letting $\mathcal{S}, \mathcal{T}$ and $\mathcal{V}$ denote the train, test and validation sets, respectively, the predicted hazard rates of the validation and test sets were scaled

	Train set	Validation set	Test set
Mean	0.0026	0.0030	0.0033
Median	$4.7809 \cdot 10^{-5}$	$2.8886 \cdot 10^{-5}$	$2.8566 \cdot 10^{-5}$
Standard deviation	0.0095	0.012	0.012

Table 6.2: Some summary statistics for train, validation and test set for the model using NLL and WBCE loss on dataset with 100 days.

Bins	Defaulters within bins
0-15	251
15-30	72
30-60	75
60-90	62
90-120	58
120-180	121
180-270	125
270-365	103
>365	89

Table 6.3: Bins for remaining lifetime, and how many defaulters from the train set are in each bin for dataset with 100 days.

by

$$s_{\mathcal{V}} = \frac{r^{\mathcal{S},\mathcal{V}}_{\text{Mean}} + r^{\mathcal{S},\mathcal{V}}_{\text{Median}}}{2}, \tag{6.1}$$

and

$$s_{\mathcal{T}} = \frac{r^{\mathcal{S},\mathcal{T}}_{\text{Mean}} + r^{\mathcal{S},\mathcal{T}}_{\text{Median}}}{2}, \tag{6.2}$$

respectively. Here, $r^{\mathcal{S},\mathcal{V}}_{\text{Mean}}$ is the mean of the predicted hazard rates on the train set divided by the mean of the predicted hazard rates on the validation set, and $r^{\mathcal{S},\mathcal{V}}_{\text{Median}}$ is the median of the predicted hazard rates on the train set divided by the median of the predicted hazard rates on the validation set. The same applies for $r^{\mathcal{S},\mathcal{T}}_{\text{Mean}}$ and $r^{\mathcal{S},\mathcal{T}}_{\text{Median}}$ for the test set.

After scaling the predicted hazard rates for the validation and test set, it must be decided what hazard rates would be associated with what remaining lifetime. The method used for determining this was described and discussed in Chapter 4. It was decided to use nine bins for the remaining lifetimes. These are shown in Table 6.3. Customers in the train set were then organized into these nine bins, based on their actual remaining lifetime after the first 100 days of their customer relationship.

Bin	Min	Max	Mean	Median	Std
0-15	3.7228e-04	1.0289e-01	3.3793e-02	3.1641e-02	1.8366e-02
15-30	1.3078e-04	6.2131e-02	1.2507e-02	1.1056e-02	1.1954e-02
30-60	2.3987e-05	6.7192e-02	5.8669e-03	8.8635e-05	1.4864e-02
60-90	1.9996e-05	2.8487e-02	2.1367e-03	5.4575e-05	6.6314e-03
90-120	2.2899e-05	9.4585e-02	8.4511e-03	8.1838e-05	1.8393e-02
120-180	1.6674e-05	5.7334e-02	8.9480e-03	1.5607e-04	1.5522e-02
180-270	1.8806e-05	5.6375e-02	3.3165e-03	4.9729e-05	9.3730e-03
270-365	1.6901e-05	6.6322e-02	3.3909e-03	5.3051e-05	1.1139e-02
>365	1.7722e-05	5.4422e-02	3.1289e-03	6.4396e-05	9.3418e-03

Table 6.4: Summary statistics for the different bins for remaining lifetime for the model using NLL and WBCE loss on dataset with 100 days.

In order to reveal what predicted hazard rates were typical for the customers in the different bins, summary statistics within each bin were calculated. These are shown in Table 6.4. The median and mean values for all bins were plotted against average remaining lifetime for the different bins in order to visualize what the connection between the hazard rate and remaining lifetime might look like. The plots are shown in Figures 6.1 and 6.2. Note that the median values for the first two bins '0-15' and '15-30' are removed for visualization purposes, due to them being significantly larger than the remaining median values. As can be seen from the plots, there is some unexpected behaviour for the mean and median hazard rates for the different groups. In particular, observe that the hazard rate drops quickly in the beginning, and then proceeds to increase over a certain time interval before decreasing again. This is counter-intuitive to our idea that the hazard rate should be monotonically decreasing as function of remaining lifetime.

After computing statistics for hazard rates within the different bins, the decision rules of which predicted hazard rate should correspond to what predicted remaining lifetime need to be set. This is done by inspecting the statistics and then defining bins for the hazard rates, within which the predicted remaining lifetime will be the same. Three competing sets of bins were made, one based on mean values, one based on median values and one which was made as an attempt at a compromise between the mean and the median. As the set based on this compromise seemed to achieve the best performance, this is the one that it was decided to use, and the one which will be presented here. The bins are presented in Table 6.5. In the code implementation, in order to have a specific number to work with, predictions in the group '>365' are set to 370, and so this will be the value used in plots and when computing evaluation metrics.

Now that the bins are defined, along with the remaining lifetime to be predicted within each bin, the final results on the validation and test sets can be analysed. For completeness, we also include the results on the train set. It is, however, the results on the test set that should be viewed as the final evaluation of the model. Firstly,

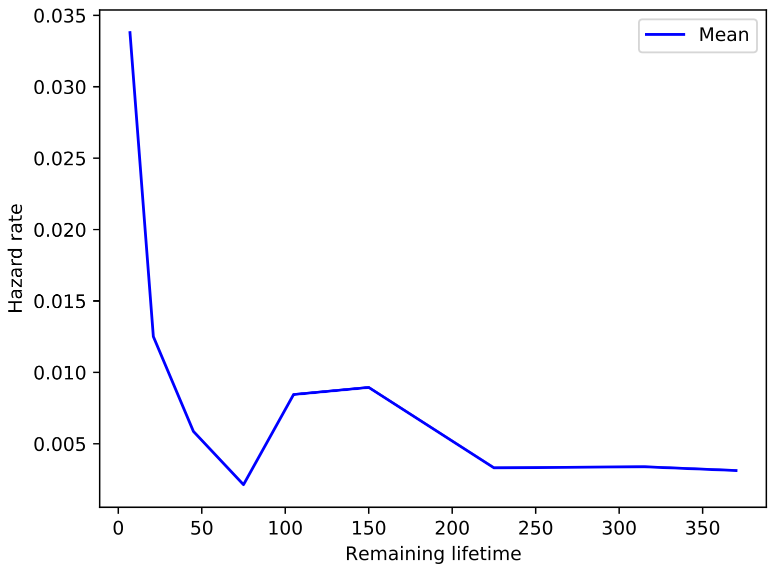

Figure 6.1: Plot of mean values for the bins of hazard rates against remaining lifetime for the model using NLL and WBCE loss on dataset with 100 days.

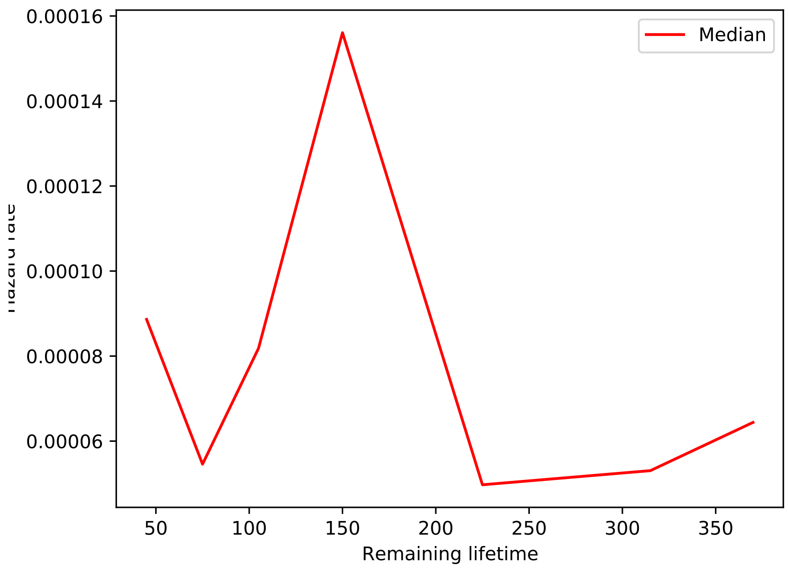

Figure 6.2: Plot of median values for the bins of hazard rates against remaining lifetime for the model using NLL and WBCE loss on dataset with 100 days. The value for bins ’0-15’ and ’15-30’ are excluded from the plot.

Bin	Predicted lifetime
$0 - 3.1 \cdot 10^{-4}$	>365
$3.1 \cdot 10^{-4} - 3.5 \cdot 10^{-4}$	315
$3.5 \cdot 10^{-4} - 4.1 \cdot 10^{-4}$	225
$4.1 \cdot 10^{-4} - 4.9 \cdot 10^{-4}$	150
$4.9 \cdot 10^{-4} - 5.8 \cdot 10^{-4}$	105
$5.8 \cdot 10^{-4} - 7.0 \cdot 10^{-4}$	75
$7.0 \cdot 10^{-4} - 8.5 \cdot 10^{-4}$	45
$8.5 \cdot 10^{-5} - 0.0125$	21
> 0.0125	7

Table 6.5: Hazard rate bins and the corresponding predicted remaining lifetime for observations within these bins for the model with NLL and WBCE loss on dataset with 100 days.

residual plots are shown in Figures 6.3, 6.4 and 6.5. Optimally, all points in these plots would lie on the line $y = 0$. As is clear from the plots, this is not the case, and many of the predictions seem to miss their target. Looking at the MRAE values, which are shown in Table 6.6, these indicate that the model on average predicts a lifetime which is larger than what is observed. Note that both the residual plots and MRAE values only give indications of the model performance on the customers that default, as these are the only ones with an actual observed lifetime.

Set	MRAE
Train	1.93
Validation	1.54
Test	1.69

Table 6.6: MRAE values of train, validation and test set for the model with NLL and WBCE loss on dataset with 100 days.

As mentioned in Chapter 4, SpareBank 1 Kreditt often evaluates their models by looking at how the predictions hold up for the coming year. This is done by checking whether the model predicts that the remaining lifetime will be less than one year for customers that do predict within one year, and similarly for those that do not default within one year. The approach was explained in Section 4.6. Balanced accuracy (BACC), Matthews correlation coefficient (MCC) and the quantity $q_{\hat{z} \geq z}$ for train, validation and test set are reported in Table 6.8. The amounts of defaulters, non-defaulters observed more than one year after time of prediction and non-defaulters observed less than one year after time of prediction in train, validation and test set are shown in Table 6.7.

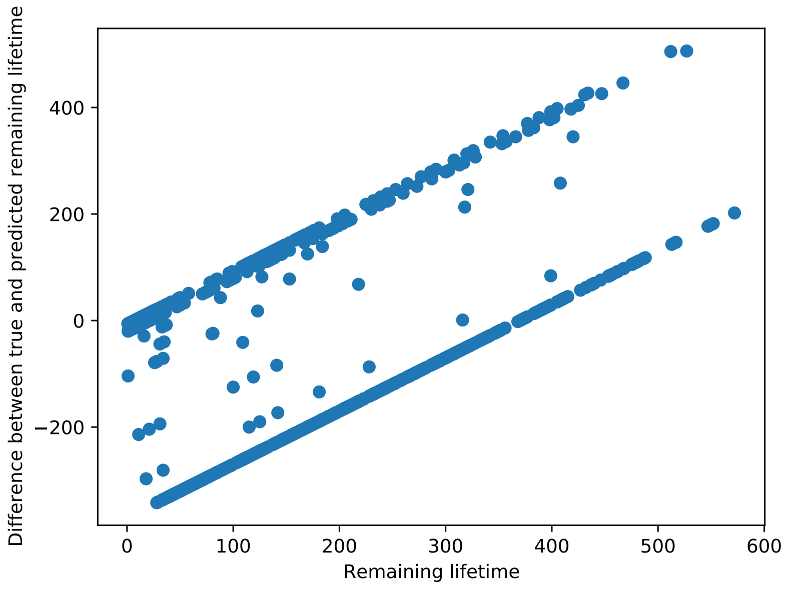

Figure 6.3: Plot of residuals $z - \hat{z}$ against remaining lifetime for defaulters in the train set for the model with NLL and WBCE loss on dataset with 100 days.

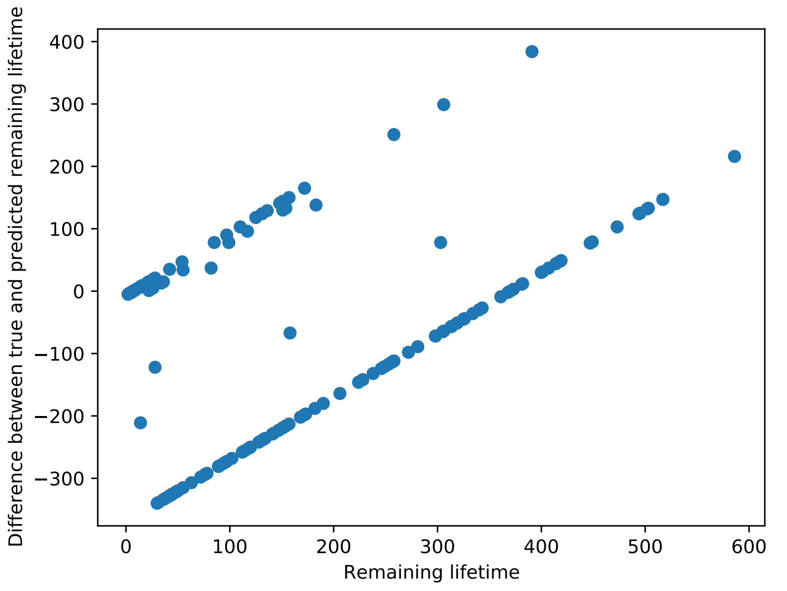

Figure 6.4: Plot of residuals $z - \hat{z}$ against remaining lifetime for defaulters in the validation set for the model with NLL and WBCE loss on dataset with 100 days.

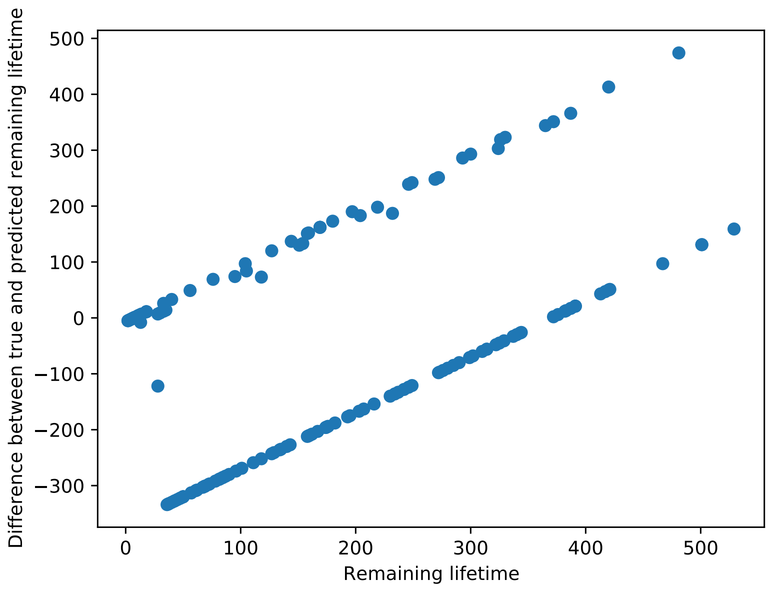

Figure 6.5: Plot of residuals $z-\hat{z}$ against remaining lifetime for defaulters in the test set for the model with NLL and WBCE loss on dataset with 100 days.

	Train	Validation	Test
Defaulters	956	154	163
Non-defaulters >365	3880	784	747
Non-defaulters <365	1938	400	413

Table 6.7: Amount of defaulters, non-defaulters observed more than one year after time of prediction and non-defaulters observed less than one year after time of prediction in train, validation and test set, for the model with NLL and WBCE loss on dataset with 100 days.

70 days

We now proceed to present the results obtained using the model with NLL and WBCE loss based on 70 days of information. By 70 days, no customers have defaulted yet, so no customers need to be removed. The best model based on this dataset achieved a concordance index of 0.58 on the validation set, which is significantly lower than that obtained by the best model based on the dataset with 100 days of information. The C-index achieved on the test set was 0.61, which is actually higher than what was achieved on the validation set. The hyperparameter values of the model are shown in Table 6.9. Observe that the values are very similar to the values for the model on the other dataset. In fact, all the values except for the learning rate, which is now lower, are the same.

	Train	Validation	Test
BACC	0.79	0.78	0.73
MCC	0.50	0.44	0.36
$q_{\hat{z} \geq z}$	0.92	0.92	0.89

Table 6.8: Values of evaluation metrics for binary classification of default within one year for the model with NLL and WBCE loss on dataset with 100 days.

Hyperparameter type	Hyperparameter	Value
Network	Hidden size	120
	Layers	4
	Dropout probability	0.5
Dataloader	Batch size	256
	Train-val-test percentages	72-14-14
Optimizer	Learning rate	0.0075
	Momentum	0.5
Loss	Loss weight α	0.6
	BCE weight β	0.2

Table 6.9: Hyperparameter values for the model using NLL and WBCE loss on dataset with 70 days.

As with the previous dataset, the last predicted hazard rates were used for the analysis. The mean, median and standard deviation of these predictions for train, validation and test set are shown in Table 6.10. For this model, both the mean and median are larger for the train set than for the validation and test set. The rescaling scheme defined by Equations 6.1 and 6.2 is used for this model as well, with the mean and median values given in Table 6.10 making up the ratios.

	Train set	Validation set	Test set
Mean	0.0014	$9 \cdot 10^{-4}$	0.0013
Median	$3 \cdot 10^{-4}$	$2 \cdot 10^{-4}$	$1 \cdot 10^{-4}$
Standard deviation	0.0055	0.0046	0.060

Table 6.10: Some summary statistics for train, validation and test set for the model using NLL and WBCE loss on dataset with 70 days.

The same nine bins as defined in Table 6.3 were used for this model. The amount of defaulters within each bin has naturally changed, and the new numbers are shown in Table 6.11. As previously, customers in the train set were organized into these nine bins based on their true remaining lifetime, after which statistics regarding the predicted hazard rates were computed for each bin. These statistics are shown in Table 6.12. Again, mean and median values were plotted against remaining lifetime for visualization purposes. These plots are shown in Figures 6.6 and 6.7. Note that, as previously, the median values for the first two bins '0-15' and '15-30' are removed for visualization purposes, due to them being significantly larger than the remaining median values. The same behaviour as for the previous model is observed in these plots as well, namely that both the mean and median values drop very quickly initially, and then proceed to increase for a while before decreasing again.

Bins	Defaulters within bins
0-15	10
15-30	103
30-60	203
60-90	86
90-120	70
120-180	119
180-270	129
270-365	114
>365	109

Table 6.11: Bins for remaining lifetime, and how many defaulters from the train set are in each bin for dataset with 70 days.

Bin	Min	Max	Mean	Median	Std
0-15	6.8296e-03	9.6421e-02	3.3771e-02	1.9867e-02	3.0633e-02
15-30	3.4022e-04	8.0639e-02	2.4426e-02	2.1911e-02	1.8876e-02
30-60	6.6758e-05	6.1265e-02	3.8980e-03	7.8624e-04	7.5363e-03
60-90	8.2139e-05	4.2990e-02	1.4009e-03	2.8589e-04	5.2481e-03
90-120	5.8468e-05	3.5430e-02	1.8243e-03	2.9805e-04	5.2092e-03
120-180	6.2170e-05	6.0851e-02	3.9590e-03	3.9251e-04	1.0278e-02
180-270	4.3427e-05	6.1619e-02	2.6822e-03	2.8284e-04	8.5480e-03
270-365	5.2041e-05	3.5069e-02	1.9626e-03	2.7689e-04	5.6127e-03
>365	7.9596e-05	3.9809e-02	1.4045e-03	2.9251e-04	4.5815e-03

Table 6.12: Summary statistics for the different bins for remaining lifetime for the model using NLL and WBCE loss on dataset with 70 days.

The decision rules for what predicted hazard rates are assoicated with what remaining lifetimes are presented in Table 6.13. They are different from the ones in the previous model, but the same method has been used for obtaining them. Again, the predictions in the group '>365' are set to 370 for practical reasons.

We are now prepared to report the final results for the model based on 70 days of information. Starting with the residual plots, shown in Figures 6.8, 6.9 and 6.10, much of the same tendencies as for the previous model are present. It appears that the model based on 70 days in general predicts more large values than the one based on 100 days. This can be seen by observing that there appears to be

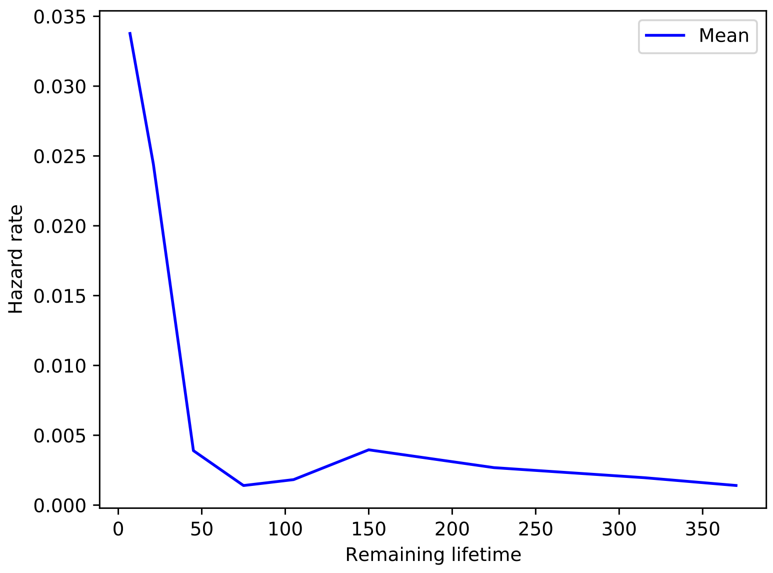

Figure 6.6: Plot of mean values for the bins of hazard rates against remaining lifetime for the model using NLL and WBCE loss on dataset with 70 days.

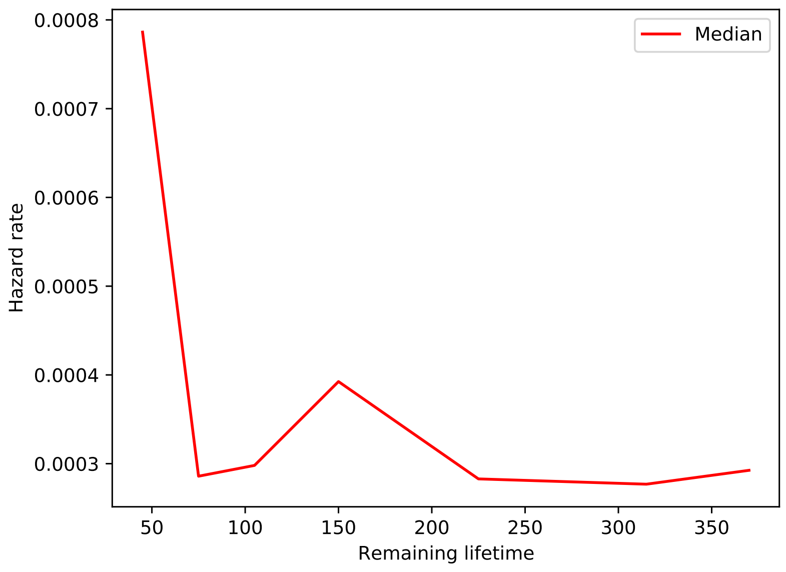

Figure 6.7: Plot of median values for the bins of hazard rates against remaining lifetime for the model using NLL and WBCE loss on dataset with 100 days. The value for bins ’0-15’ and ’15-30’ are excluded from the plot.

Bin	Predicted lifetime
$0-6.0\cdot10^{-4}$	>365
$6.0\cdot10^{-4}-1.0\cdot10^{-3}$	315
$1.0\cdot10^{-3}-1.5\cdot10^{-3}$	225
$1.5\cdot10^{-3}-2.2\cdot10^{-3}$	150
$2.2\cdot10^{-3}-3.0\cdot10^{-3}$	105
$3.0\cdot10^{-3}-4.0\cdot10^{-3}$	75
$4.0\cdot10^{-3}-5.2\cdot10^{-3}$	45
$5.2\cdot10^{-3}-0.022$	21
>0.022	7

Table 6.13: Hazard rate bins and the corresponding predicted remaining lifetime for observations within these bins for the model with NLL and WBCE loss on dataset with 70 days.

a larger density of dots along the bottom line of the plots. Looking at the MRAE values, which are shown in Table 6.14, these indicate that the model on average predicts a lifetime which is larger than what is observed. The reported values are comparable to the ones obtained for the previous model.

Set	MRAE
Train	1.93
Validation	1.85
Test	1.63

Table 6.14: MRAE values of train, validation and test set for the model with NLL and WBCE loss on dataset with 70 day.

Next, we consider how well the model performed with respect to binary classification of default one year ahead in time. Table 6.15 shows the amount of defaulters, non-defaulters observed more than one year after time of prediction and non-defaulters observed less than one year after time of prediction that are present in the train, validation and test set. The different evaluation metric values are shown in Table 6.16.On the test set, the results obtained are actually better than the once obtained for the previous model. However, considering that the scores on both the train and validation sets are worse than those on the test set, and considering that this was also the case for the MRAE and C-index, we come to suspect that the split simply favoured the test set in this particular case.

The results obtained from the trained models were mixed. For the model based on 100 days of information, the concordance index of 0.67 on the test set is not particularly high. However, it does indicate that the model works and does learn some reasonable decision rules. The residual plots indicate that the model unfortunately misses with more than 100 days on most of the predicted lifetimes. The

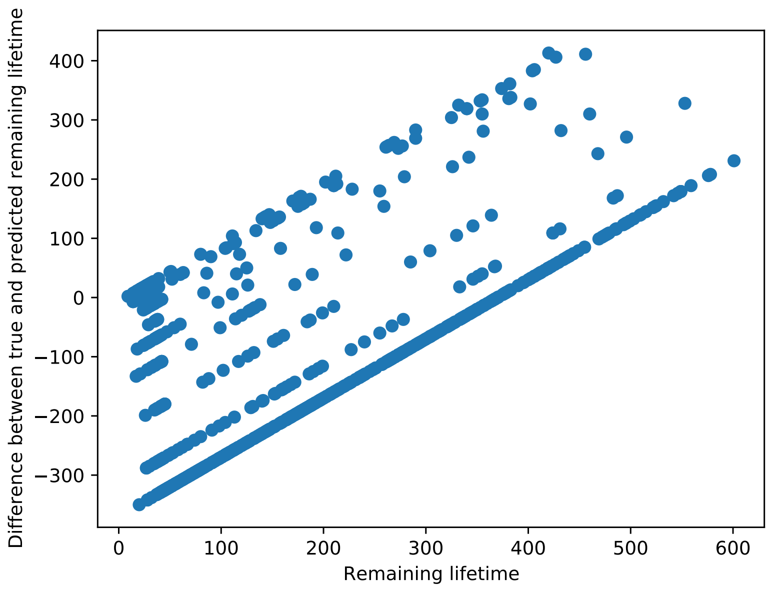

Figure 6.8: Plot of residuals $z - \hat{z}$ against remaining lifetime for defaulters in the train set for the model with NLL and WBCE loss on dataset with 70 days.

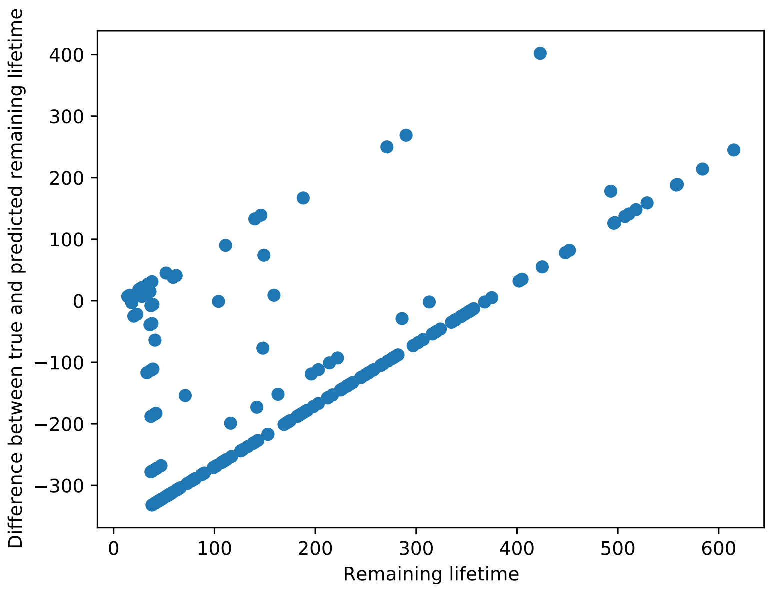

Figure 6.9: Plot of residuals $z - \hat{z}$ against remaining lifetime for defaulters in the validation set for the model with NLL and WBCE loss on dataset with 70 days.

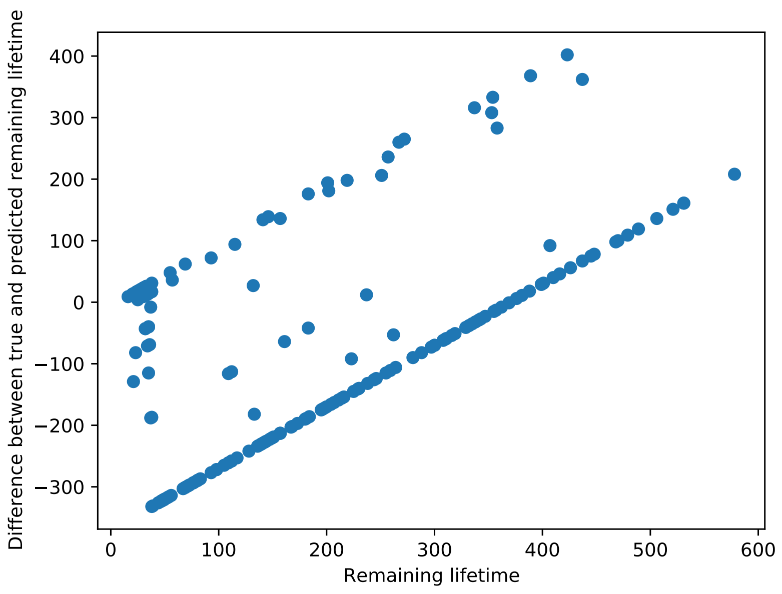

Figure 6.10: Plot of residuals $z - \hat{z}$ against remaining lifetime for defaulters in the test set for the model with NLL and WBCE loss on dataset with 70 days.

	Train	Validation	Test
Defaulters	943	199	195
Non-defaulters >365	4128	832	837
Non-defaulters <365	1703	323	322

Table 6.15: Amount of defaulters, non-defaulters observed more than one year after time of prediction and non-defaulters observed less than one year after time of prediction in train, validation and test set, for the model with NLL and WBCE loss on dataset with 70 days.

	Train	Validation	Test
BACC	0.74	0.70	0.74
MCC	0.30	0.33	0.41
$q_{\hat{z} \geq z}$	0.93	0.95	0.94

Table 6.16: Values of evaluation metrics for binary classification of default within one year for the model with NLL and WBCE loss on dataset with 70 days.

MRAE value of 1.69 on the test set is, surprisingly, lower than the value on the train set. It indicates that the model often predicts values that are either close to zero or more than twice as large as the true lifetime. Finally, the balanced accuracy and Matthews correlation coefficient indicate that the model works fine for binary classification for default within the next year, achieving scores around what is the current standard for machine learning models at SpareBank 1 Kreditt. For the model based on 70 days of information, the results are similar. Looking at the MRAE values, we see that the value on the test set is actually slightly lower than that for the model based on 100 days. It is quite similar however, and again shows that the model predictions are mostly off target, and often the predicted values are more than twice as large as the true ones. Again, the results for the the binary classification task were decent, however slightly worse than those obtained in the model based on 100 days. Further discussion of the results presented in this chapter, as well as of the methods used to achieve them, will be presented in the next chapter.

6.2 MSE and WBCE Loss

This section was supposed to present the results obtained by the models trained using mean squared error and weighted binary cross entropy as loss functions. Unfortunately, a problem arose in the code, and this problem remained unresolved for the remaining duration of the work on the thesis. Thus, no results were obtained using these loss functions. As the MSE loss would provide a more direct supervision over the actual remaining lifetimes, it is very unfortunate that the effects this could have on the predicted lifetimes could not be explored. No further discussion regarding this particular loss function in the frames of our recurrent, discrete time model will be provided, as this will merely serve as speculation without any results to back it up.

Chapter 7

Discussion

In this chapter we discuss several aspects of the methods used and the results obtained in this thesis, as well as the problem itself. Simplifications and assumptions that have been made along the way are explored and discussed. Before initiating the discussion it is worth pointing out that this thesis has been produced over a limited time span. Many simplifications are due to this, and there are many aspects of the different approaches taken throughout the thesis that have not been explored. As a result, potential for improvement and further research will also be discussed in this chapter.

To summarize the results, both models seem to do relatively well in the case of binary classification, in that the results are on par with what is the current standard. However, in genera, the models predict lifetimes that are quite a lot larger than the true lifetimes, and struggle to differentiate defaulters from non-defaulters. Considering the imbalance and incompleteness of the dataset, as well as earlier results (e.g. Holte, 2021) and experience from the industry, these results are not surprising. Comparing the results to those achieved in Holte (2021), it is interesting to note that a full longitudinal model does not perform much better than a model based on aggregated data. Naturally, the results are largely influenced by the method for determining the relationship between predicted hazard rates and remaining lifetime, which was quite experimental and not very rigorous. It would be interesting to see how the results may have turned out if this relationship was more deeply explored. Before leaving the results to discuss other aspects of the thesis, it is worth noting that any comparison of the two models needs to be viewed in light of the method used to choose the bins for the predicted hazard rates and its lack of mathematical rigor. Differences may therefore in part be ascribed to differences in "how well" the bins were chosen.

Thus far, we have not provided any justification for the choice of bins for the remaining days in Chapter 6. The bins were equal for both datasets, and are shown in Figure 6.3. The rationale between choosing the bins in this way was that, in general, we are more interested in more refined predictions for customers that are predicted to default within short time. This explains why the first two bins cover only 15 days each, while the next three cover 30 days each and the next bins cover 60, 90 and 95 days, respectively. Looking at the plots of the hazard rate statistics against the remaining lifetimes, we observe that the predicted hazard rates seem to drop significantly for the first three bins, which is behaviour that was expected, and which further motivates the use of smaller bins for small values of remaining lifetime. The final bin covers all remaining lifetimes larger than one year. This decision was based on the predictions becoming unreliable when the subjects have a long remaining lifetime, and on the fact that uncertainty starts to play a major role.

No analysis of feature importance was done in this thesis. The main reason for this was time constraints, however the decision rules of neural networks are also notoriously difficult to interpret, which is actually one of the main criticisms of them. The SHAP framework, introduced by Scott M. Lundberg only four years ago, provides a game theoretic approach to explain the output of any machine learning model (Lundberg and Lee, 2017, Lundberg, Scott M., 2021). It could certainly be interesting to explore feature importance in the models trained in this thesis using this framework. A feature importance analysis would be informative in the sense that it could give insight into how the model makes decisions, and which variables it considers to be important when making predictions. Furthermore, comparing the importance of the different features to the observations from the exploratory data analysis in Chapter 5 could be interesting, and reveal if the neural network may have uncovered deeper patterns than were recognized in that analysis.

The question of what evaluation metrics to use is essential to any machine learning method. All metrics have their advantages and drawbacks, and it might not always be clear what metric is the most trustworthy in association with the problem at hand. In this thesis, it was decided to use the concordance index as one of the main metrics, as it provides an indication of the discriminatory abilities of the model, which are arguably the most important abilities for a survival model. However, none of the loss functions used were designed specifically to maximize the concordance index. Using loss functions that can be more easily associated with the concordance index could therefore be a suggestion for future research. Another option is to try and find performance metrics that are more directly optimized by the chosen loss functions. Other performance metrics were considered, and in the end the mean relative absolute error was used, mainly due to the lack of a better metric for assessing how well the model was calibrated. The Brier Score was considered, however it was deemed unsuitable for our model, due to survival estimates exceeding the day of default being inaccessible. Using some sort of Brier Score that is adapted more specifically to our model could be of interest.

The dataset that has been used in this thesis presents several challenges. Firstly, there is the imbalance. As only around 13% of the customers experience a default during the time they are observed, and machine learning models tend to be biased towards the majority class, our neural network may have had a hard time trying to learn what characterizes a customer at risk of defaulting. This also leads to the network generally predicting hazard rates that are very low, leading to predicted lifetimes that are very high. Combined with the fact that many defaulters and non-defaulters exhibit very similar behaviour, the problem of predicting who will default, and at what time, becomes complicated. Another major challenge is that, due to privacy, the customer information that companies can gather and use is limited. It might therefore be (and it does indeed seem to be the case) that the useful information the dataset provides us with is limited, and that the information that really determines how much time will pass before a customer defaults is not available to us. Furthermore, even if we were allowed to record absolutely all the information that we wanted about customers, we still would not be able to foresee things like accidents, personal tragedies and other events that may affect a person's finances. This means that any prediction about the far future would have to be viewed with caution.

No large amount of data preparation was done in this thesis. The data was standardized to have zero mean and unit variance, and some observations were removed from the dataset, but no further feature engineering was performed. Furthermore, only one type of scaling was attempted. Scaling the data differently, for example normalizing features to have values between zero and one, could lead to different results. Some variables in the dataset were also disregarded, either because they had the same information as other variables, or because they were on a format which was not readily usable in the model. Spending more time extracting information from these variables could certainly be of use.

The decision to use a recurrent neural network in this thesis was mainly based on its adequacy for dealing with sequential data. Further, the LSTM was chosen in order to better be able to retain information from previous time steps. Whether these models are well suited to the different data types and combination of data types in the dataset was not explored. The non-parametric, discrete time model for predicting hazard rates was chosen based on the work of Ren et al. (2018), and no material outside of the article itself was used to inform the decision of using this model. This is partly because there is no large amount of material presenting models that suit our problem as well as the model proposed in Ren et al. (2018).

The loss functions originally proposed in Ren et al. (2018) seemed to lead to good results on the datasets treated in that article, however none of them provided direct supervision over the actual remaining lifetimes. This meant that some relationship between predicted hazard rates and remaining lifetime needed to be assumed, something that brought a new and unforeseen aspect into the model. As mentioned in Chapter 4, one could assume a functional relationship, for instance something similar in spirit to Equation 4.13. To aid in deciding what functional relationship to use, one could utilize plots and maybe some regression techniques. The decision to not assume any functional relationship here was made partly due to time limitations, and partly to avoid imposing restrictions that would follow from assuming a functional relationship. The method that was chosen was one that was easy to implement and had lots of room for changes and adaption. It was quite simple, but could easily be made more complex by assuming functional relationships within the different bins that were created, or through creating additional bins. The method seemed appropriate when considering the fact that the predicted hazard rates were seen to be quite inaccurate, and considering that any prediction could only be trusted to a certain degree due to unforeseeable events. Furthermore, even though an accurate prediction of time to default is of course desirable, a pinpoint indicating approximately how much time remains before a customer defaults is mainly what can be hoped for. Still, having some more strict decision rules for how to best create these bins could be beneficial, as no particular set of rules was followed when they were created, something that leaves a lot of room for subjectivity. If one wanted to completely avoid the need to find a relationship between hazard rates and remaining lifetimes, one could have the model directly predict remaining lifetime instead. This would require the use of different loss functions that supervised directly over remaining lifetime, or over scaled versions of the remaining lifetime. Another alternative would be to create an additional neural network on top of the already existing one, and have this network try to learn the relationship between hazard rate and remaining lifetime. This approach could certainly be of interest, as it would allow for the functional relationship to be learned rather than assumed. It would also allow for different degrees of complexity through the decision of network architecture.

As was mentioned initially in Chapter 6, the amount of hyperparameter tuning performed in this thesis was quite small. Several values were tried for several parameters, and the goal was mainly to see if a large improvement could be achieved by changing these in certain directions. Granted, an improvement in the C-index was seen as a result of this experimentation, with the validation concordance increasing from 0.60 in the first model to 0.72 in a later one for one of the datasets. This illustrates that hyperparameters have a significant impact on the model one is training, and motivates hyperparameter searches. Using more organized methods such as grid search, response surface methodology or other more advanced methods could give further insight and most likely result in even better hyperparameter values.

Considering that the models trained did not in effect surpass existing, simpler models in terms of predictive performance, the current value that deep learning adds to the problem of credit scoring has to be seen as limited according to this thesis. Furthermore, the fact that a model based on sequential data performs comparably to a model based on summary statistics indicates that either there is little more information to be gained from the full time series than from the summary statistics, or the approach taken does not harness this information in a good way. The many simplifications and unexplored opportunities mentioned in this chapter motivate further research to try and unite survival analysis and deep learning to achieve strong and reliable results for credit card data. Using other neural network models, looking more into proper loss functions and hyperparameter tuning, as well as dedicating time to find proper evaluation metrics, are all topics that, at least in the opinion of the writer, deserve attention and further research.

Summary

In this thesis, a deep neural network model combined with concepts from survival analysis is used to predict time to default for credit card customers. The dataset used consists of sequential data for customers of the credit card institution SpareBank 1 Kreditt. An exploratory data analysis is performed to get familiarized with the dataset, exploring both how unchanging and changing variables affect the two main groups of customers: those that default, and those that don't. A discrete time model is then formulated based on central concepts from survival analysis. An LSTM structure is used for the network architecture, and models are trained based on data from two different time periods: from day 0 to day 70 of customer relationships, and from day 0 to day 100. The output from the neural network is transformed from a number between 0 and 1 to a whole number, which is then used as the predicted amount of days until default. Several hyperparameter combinations are tried in order to increase performance. Two of the trained models and the results they achieved are presented, and the results and their implications are discussed.

The main objective of this thesis is to try a full longitudinal approach to a problem which SpareBank 1 Kreditt has mainly treated using models based on summary statistics. It is also an attempt at combining deep learning and survival analysis to make predictions based on credit card data. The results are comparable to the current standard, and do not directly motivate the use of such models in the industry, due to their complexity. However, as several simplifications are made and several paths left unexplored, there is an argument for further research into the topic. In particular, the relationship between the output of the network and the expected time to default should be more thoroughly explored. More effort should also be put into hyperparameter tuning, choice of loss functions and choice of evaluation metrics.

Bibliography

Banasik, J., J. N. Crook and L. C. Thomas (1999). 'Not if but when will borrowers default'. In:

Barnett, M. and P. Larkman (July 2007). 'The action potential'. In: *Practical neurology* 7, pp. 192–7.

Brodersen, K. H., C. S. Ong, K. E. Stephan and J. M. Buhmann (2010). 'The balanced accuracy and its posterior distribution'. In:

Brownlee, J. (2019a). *How to Configure the Learning Rate When Training Deep Learning Neural Networks*. URL: https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/ (visited on 18/06/2021).

Brownlee, J. (2019b). *How to use Data Scaling Improve Deep Learning Model Stability and Performance*. URL: https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/ (visited on 18/06/2021).

Brownlee, J. (2020a). *Imbalanced Classification with Python*.

Brownlee, J. (2020b). *Train-Test Split for Evaluating Machine Learning Algorithms*. URL: https://machinelearningmastery.com/train-test-split-for-evaluating-machine-learning-algorithms/ (visited on 19/06/2021).

Chicco, D. and G. Jurman (2019). 'The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation'. In: *BMC Genomics*.

Dirick, L., G. Claeskens and B. Baesens (2014). 'An Akaike information criterion for multiple event mixture cure models'. In:

Dirick, L., G. Claeskens and B. Baesens (2016). 'Time to default in credit scoring using survival analysis: a benchmark study'. In: *Journal of the Operational Research Society*.

Du, K.-L. and M. N. S. Swamy (2013). *Neural Networks and Statistical Learning*.

E24 (2020). *Nordmenn har 57 milliarder i misligholdt gjeld*. URL: https://e24.no/privatoekonomi/i/GaP8zJ/nordmenn-har-57-milliarder-i-misligholdt-gjeld (visited on 02/06/2021).

Finans Norge (2020). *Kredittkortbruken fortsetter nedover*. URL: https://www.finansnorge.no/aktuelt/nyheter/2020/10/kredittkortbruken-fortsetter-nedover/ (visited on 02/06/2021).

Finansavisen (2021). *Nordmenns kredittkortbruk øker*. URL: https://finansavisen.no/nyheter/personlig-okonomi/2021/04/06/7651895/nordmenns-kredittkortbruk-oker (visited on 02/06/2021).

Gemini (2020). *Norske husholdninger ligger i verdenstoppen i privat gjeld*. URL: https://gemini.no/2020/11/norske-husholdninger-ligger-i-verdenstoppen-i-privat-gjeld/ (visited on 02/06/2021).

Gjeldsregisteret (2020). *Nøkkeltall*. URL: https://www.gjeldsregisteret.com/pages/nokkeltall (visited on 02/06/2021).

Glorot, X. and Y. Bengio (2010). 'Understanding the difficulty of training deep feedforward neural networks'. In:

Goodfellow, I., Y. Bengio and A. Courville (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press.

Graf, E., C. Schmoor, W. Sauerbrei and M. Schumacher (1999). 'Assessment and comparison of prognostic classification schemes for survival data'. In: *Statistics in Medicine* 18.17-18, pp. 2529–2545. DOI: https://doi.org/10.1002/(SICI)1097-0258(19990915/30)18:17/18<2529::AID-SIM274>3.0.CO;2-5. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/%28SICI%291097-0258%2819990915/30%2918%3A17/18%3C2529%3A%3AAID-SIM274%3E3.0.CO%3B2-5.

Graves, A., A.-r. Mohamed and G. Hinton (2013). 'Speech Recognition with Deep Recurrent Neural Networks'. In:

Gunnarson, B. R., S. vanden Broucke, B. Baesens, M. Óskarsdóttir, J. De Weerdt and W. Lemahieu (2019). 'Deep Learning in Credit Scoring: Do or Don't'. In:

Harrel, F. E., K. L. Lee and D. B. Mark (1996). 'Multivariable Prognostic Models: Issues in Developing Models, Evaluating Assumptions and Adequacy, and Measuring ans Reducing Errors'. In: *Statistics in medicine*.

Hastie, T., R. Tibshirani and J. Friedman (2008). *The Elements of Statistical Learning*. Springer.

Hochreiter, S. and J. Schmidhuber (1997). 'Long Short-Term Memory'. In:

Holte, H. (2021). 'Survival Analysis for Prediction of Time to Default for Credit Card Customers'. In:

Hosmer, D. W., S. Lemeshow and S. May (2008). *Applied Suvival Analysis*. John Wiley & Sons, Inc.

Kim, J. and C. D. Scott (2012). 'Robust Kernel Density Estimation'. In: *Journal of Machine Learning Research*.

Kleinbaum, D. G. and M. Klein (2005). *Survival Analysis*. Springer.

Kuhn, M. and K. Johnson (2013). *Applied Predictive Modelling*. Springer.

Kvamme, H. (2019). 'Time-to-Event Prediction with Neural Networks'. In:

Lambora, A., K. Gupta and K. Chopra (2019). 'Genetic Algorithm- A Literature Review'. In:

Lundberg, S. M. and S.-I. Lee (2017). 'A Unified Approach to Interpreting Model Predictions'. In:

Lundberg, Scott M. (2021). *SHAP Documentation*. URL: https://shap.readthedocs.io/en/latest/index.html (visited on 20/06/2021).

McCulloch, W. S. and W. Pitts (1943). ‘A Logical Calculus of the Ideas Immanent in Nervous Activity’. In:

Nahrain, B. (1992). ‘Survival analysis and the credit granting decision’. In:

NRK (2021). *Økt kredittkortbruk i Norge*. URL: `https://www.nrk.no/nyheter/okt-kredittkortbruk-i-norge-1.15517815` (visited on 02/06/2021).

Qi, M. (2009). ‘Exposure at Default of Unsecured Credit Cards’. In:

Ren, K., J. Qin, L. Zheng, Z. Yang, W. Zhang, L. Qiu and Y. Yu (2018). ‘Deep Recurrent Survival Analysis’. In:

Rosenblatt, F. (1958). ‘The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain’. In:

Shalev-Shwartz, S. and S. Ben-David (2014). *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press.

Shariari, B., K. Swersky, Z. Wang, R. P. Adams and N. de Freitas (2015). ‘Taking the Human Out of the Loop: A Review of Bayesian Optimization’. In:

SpareBank 1 SR-Bank (2020). *Nordmenn har kvittet seg med en halv million lån*. URL: `https://www.sparebank1.no/nb/sr-bank/om-oss/nyheter/nordmenn-har-kvittet-seg-med-en-halv-million-lan.html` (visited on 02/06/2021).

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov (2014). ‘Dropout: A Simple Way to Prevent Neural Networks from Overfitting’. In: *Journal of Machine Learning Research* 15.56, pp. 1929–1958. URL: `http://jmlr.org/papers/v15/srivastava14a.html` (visited on 18/06/2021).

The Balance (2020). *What You Can Do About Credit Card Default*. URL: `https://www.thebalance.com/what-is-credit-card-default-960209#:~:text=Credit%20card%20default%20happens%20when,for%20other%20credit%2Dbased%20services.` (visited on 05/04/2021).

Wang, Y., Y. Yang, J. Mao, Z. Huang, H. Chang and W. Xu (2016). ‘CNN-RNN: A Unified Framework for Multi-label Image Classification’. In:

Weihs, C., K. Luebke and I. Czogiel (Feb. 2006). ‘Response Surface Methodology for Optimizing Hyper Parameters’. In: DOI: `10.17877/DE290R-14252`.

Yin, W., K. Kann, M. Yu and H. Schütze (2017). ‘Comparative Study of CNN and RNN for Natural Language Processing’. In:

Zhang, J. and L. C. Thomas (2012). ‘Comparisons of linear regression and survival analysis using single and mixture distributions approaches in modelling LGD’. In:

Appendix A

Variables in dataset

Variable	Description
RK_ACCOUNT_ID	Internal account Id
BK_ACCOUNT_ID	Internal account Id
AccountCreatedDateId	Date of account (card) created (YYYYMMDD)
ApplicationScore	Score of application
PRODUCT_NAME	Name of (card) product
CREDIT_LIMIT_AMT	Credit limit on card [kr]
GEN_BK_ACCOUNT_STATUS_CD	Account Staus code
PostalCodeFirst2	First 2 numbers in postal code
HAS_DIRECT_DEBIT_AGREEMENT_IND	Indicator, direct debit agreement selected ("avtalegiro")
HAS_ESTATEMENT_AGREEMENT_IND	Indicator, e-statement selected ("e-faktura")
CustomerAge	Customer's age in years
GENDER_NAME	Gender
DISTRIBUTOR_NAME	Bank Name
CashBackStatus	CashBackAccount's status
MonthsSinceAccountCreated	Account's age in months
AccountBalanceDateId	Date of account data (YYYYMMDD) (DD)
prevPeriodId	End of month before collection opened (YYYYMMDD), EOMB
BALANCE_AMT	Balance at DD
IEL_AMT	Interest earning balance at DD
CASH_BALANCE_AMT	Cash balance at DD
OVERDUE_AMT	The amount overdue at DD
DomCashNum	Number of domestic cash withdrawals at DD
DomCashSum	Sum of domestic cash withdrawals DD
DomPurchaseNum	Number of domestic purchases DD
DomPurchaseSum	Sum of domestic purchases DD
IntCashNum	Number of international cash withdrawals DD
IntCashSum	Sum intenational cash withdrawals DD

IntPurchaseNum	Number of internationsal purchases DD
IntPurchaSum	Sum of international purchases DD
Transfernum	Number of cash transfers DD
Transfersum	Sum of cash transfers DD
FeeNum	Number of fees DD
FeeSum	Sum of fees DD
SumAirlineL12M	Sum of transactions in given class last 12 months (EOMB)
SumELECTRIC_APPLIANCEL12M	Sum of transactions in given class last 12 months (EOMB)
SumFOOD_STORES_WAREHOUSEL12M	Sum of transactions in given class last 12 months (EOMB)
SumHOTEL_MOTELL12M	Sum of transactions in given class last 12 months (EOMB)
SumHARDWAREL12M	Sum of transactions in given class last 12 months (EOMB)
SumINTERIOR_FURNISHINGSL12M	Sum of transactions in given class last 12 months (EOMB)
SumOTHER_RETAILL12M	Sum of transactions in given class last 12 months (EOMB)
SumOTHER_SERVICESL12M	Sum of transactions in given class last 12 months (EOMB)
SumOTHER_TRANSPORTL12M	Sum of transactions in given class last 12 months (EOMB)
SumRECREATIONL12M	Sum of transactions in given class last 12 months (EOMB)
SumRESTAURANTS_BARSL12M	Sum of transactions in given class last 12 months (EOMB)
SumSPORTING_TOY_STORESL12M	Sum of transactions in given class last 12 months (EOMB)
SumTRAVEL_AGENCIESL12M	Sum of transactions in given class last 12 months (EOMB)
SumVEHICLESL12M	Sum of transactions in given class last 12 months (EOMB)
SumQuasiCashL12M	Sum of transactions in given class last 12 months (EOMB)
SumAirlineL3M	Sum of transactions in given class last 3 months (EOMB)
SumELECTRIC_APPLIANCEL3M	Sum of transactions in given class last 3 months (EOMB)
SumFOOD_STORES_WAREHOUSEL3M	Sum of transactions in given class last 3 months (EOMB)
SumHOTEL_MOTELL3M	Sum of transactions in given class last 3 months (EOMB)

SumHARDWAREL3M	Sum of transactions in given class last 3 months (EOMB)
SumINTERIOR_FURNISHINGSL3M	Sum of transactions in given class last 3 months (EOMB)
SumOTHER_RETAILL3M	Sum of transactions in given class last 3 months (EOMB)
SumOTHER_SERVICESL3M	Sum of transactions in given class last 3 months (EOMB)
SumOTHER_TRANSPORTL3M	Sum of transactions in given class last 3 months (EOMB)
SumRECREATIONL3M	Sum of transactions in given class last 3 months (EOMB)
SumRESTAURANTS_BARSL3M	Sum of transactions in given class last 3 months (EOMB)
SumSPORTING_TOY_STORESL3M	Sum of transactions in given class last 3 months (EOMB)
SumTRAVEL_AGENCIESL3M	Sum of transactions in given class last 3 months (EOMB)
SumVEHICLESL3M	Sum of transactions in given class last 3 months (EOMB)
SumQuasiCashL3M	Sum of transactions in given class last 3 months (EOMB)
Segment9Name	Segment (see separate documentation)
Segment23Name	Segment (see separate documentation)
UtilizationL12	average revolving balance last 12 months divided by average credit limit last 12 months
UtilizationL3	average revolving balance last 3 months divided by average credit limit last 12 months
AvgRevBalL3onL12	Avergage revolving balance last 3 months divided by average revolving balance last 12 months
DC2Ind	Indicator of default (lifetime event indicator)
RemaningLifetime	Remaining life DD (days)

Table A.1: Variables included in the original dataset, and their descriptions. Variables in boldface are included in the data analysis and the trained models.

Appendix B

Trained models

Hyperparameter name	Abbreviated name (used in Table B.2)
Number of layers	nl
Hidden size	hs
Batch size	bs
Learning rate	lr
Loss weight	lw
BCE weight	bw

Table B.1: Abbreviations for hyperparameters.

Loss functions	Days	Hyperparams	Val conc	Comment
NLL, WBCE	100	nl: 3 hs: 100 bs: 256 lr: 0.01 lw: 0.6 bw: 0.2	0.60	Concordance not computed properly
NLL, WBCE	100	nl: 3 hs: 100 bs: 128 lr: 0.01 lw: 0.6 bw: 0.2	0.57	Concordance not computed properly
NLL, WBCE	100	nl: 3 hs: 120 bs: 256 lr: 0.01 lw: 0.6 bw: 0.2	0.71	
NLL, WBCE	**100**	**nl: 4**	**0.72**	Best results for 100 days

		hs: 120 bs: 256 lr: 0.01 lw: 0.6 bw: 0.2		
NLL, WBCE	100	nl: 4 hs: 120 bs: 256 lr: 0.01 lw: 0.3 bw: 0.2	0.68	
NLL, WBCE	100	nl: 5 hs: 120 bs: 256 lr: 0.01 lw: 0.6 bw: 0.2	0.66	
NLL, WBCE	100	nl: 4 hs: 66 bs: 256 lr: 0.01 lw: 0.6 bw: 0.2	0.63	
NLL, WBCE	100	nl: 4 hs: 120 bs: 256 lr: 0.01 lw: 0.6 bw: 0.3	0.62	
NLL, WBCE	100	nl: 4 hs: 120 bs: 256 lr: 0.005 lw: 0.6 bw: 0.2	0.66	
NLL, WBCE	70	nl: 3 hs: 100 bs: 256 lr: 0.005 lw: 0.6 bw: 0.2	0.57	
NLL, WBCE	**70**	**nl: 4** hs: 120 bs: 256	**0.58**	Best results for 70 days

		lr: 0.0075 lw: 0.6 bw: 0.2		
NLL, WBCE	70	nl: 4 hs: 120 bs: 256 lr: 0.01 lw: 0.6 bw: 0.2	0.54	
NLL, WBCE	70	nl: 4 hs: 100 bs: 256 lr: 0.0075 lw: 0.6 bw: 0.1	0.56	
NLL, WBCE	70	nl: 4 hs: 120 bs: 256 lr: 0.0075 lw: 0.7 bw: 0.1	0.56	
NLL, WBCE	70	nl: 4 hs: 140 bs: 256 lr: 0.0075 lw: 0.6 bw: 0.1	0.56	

Table B.2: Overview of hyperparameters and validation concordance of the models trained. The models that performed best within their category have the first line written in bold text.

Håkon Andersen Holte

Survival Analysis with Deep Recurrent Neural Network Model for Prediction of Credit Card Defaults

NTNU

Norwegian University of
Science and Technology