NTNU
Norwegian University of
Science and Technology

The Application of Recurrent Neural Networks on Early Detection of Credit Default Risk

Andreas Owe

Master of Science in Informatics
Submission date: March 2018
Supervisor: Mads Nygård, IDI

Norwegian University of Science and Technology
Department of Computer Science

Abstract

Credit card companies lend money to borrowers on the presumption of the borrowers being able to pay back the borrowed amount plus a fee in the form of interest. There is an inherent risk in lending money: that the borrower will not be able to pay back the borrowed amount, and therefore default on the payment; causing the lender to lose their invested money.

This thesis aims to determine if we can use recurrent neural networks in order to identify defaulted customers based on a customers account data during their first 12 months with the bank. Several models have been developed in order to examine different properties of the recurrent neural network, and how they impact the models efficacy. While previous research has used neural networks to classify default, this thesis uses a type of network called a recurrent neural network; which specializes in finding meaning in temporal data.

We have found that by using a recurrent neural network, we can identify defaulted customers within their first 12-month relationship with the bank. The best model was able to accurately classify 90% of the defaulted customers, at the expense of mis-classifying some of the non-defaulted customers.

Sammendrag

Kredittkortbedrifter låner ut penger til låntagere under formodningen at låntageren vil betale tilbake den lånte summen pluss en avgift i form av renter. For bedrifter ligger det en risiko i å låne ut penger: at låntageren ikke betaler tilbake den lånte summen, og dermed misligholder lånet; noe som forårsaker at utlåneren taper sine investerte penger.

Denne oppgaven har som mål å fastlå om vi kan bruke 'recurrent neural networks' til å identifisere låntagere som misligholder lånet sitt basert på kundens kontodata i deres første 12 måneder med banken. Flere modeller har blitt utviklet for å utforske de ulike egenskapene til 'recurrent neural networks', og hvordan de påvirker modellenes egenskap til å klassifisere mislighold. Mens tidligere forskning har brukt neurale nettverk til å klassifisere mislighold, vil denne oppgaven bruke en type nettverk kalt 'recurrent neural networks'; som spesialiserer seg i å finne mening i data over tid.

Vi har funnet at ved å bruke 'recurrent neural networks', kan vi identifisere kunders mislighold innenfor deres første 12 måneder ved banken. Den beste modellen klarte å klassifisere 90% av kundene som misligholdt lånet, på bekostning av å feil-klassifisere enkelte kunder som ikke hadde misligholdt.

Acknowledgement

First, I would like to thank my supervisor Mads Nygård for his invaluable support and guidance on this thesis. His patience, encouragement, and insightful comments have been of utmost help to me throughout the last year.

Furthermore, I would like to thank Christian Meland at SpareBank1 for his guidance and the opportunity to write this thesis. Finally, I would like to thank Keith L. Downing for his guidance on neural networks.

A.O.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The introduction chapter will explain the motivation behind the thesis as well as give a brief introduction to the Norwegian credit market.

1.1 Motivation

The motivation for this thesis is to analyze credit card data to determine if a customer has defaulted on their payment (not paid back the borrowed amount). Credit card companies lend money to borrowers on the presumption of the borrowers being able to pay back the borrowed amount plus a fee in the form of interest. There is an inherent risk in lending money: that the borrower will not be able to pay back the borrowed amount, and therefore default on the payment; causing the lender to lose their invested money.

Lenders try to minimize their risk by thoroughly vetting their customers and checking for signs that might indicate that the customer will default. This has traditionally been done by creating complex mathematical models that considers a multitude of variables. With the rise of cheap and fast computing power, and progress in machine learning, it has now become feasible to train computers to detect signs that a customer will default. An accurate model to detect default that can be easily extended and maintained could be very valuable to lenders.

This thesis will explore the application of neural networks on detecting early signs of default. Neural networks is a machine learning technique used in classification and

prediction. Although early forms of neural networks have been around since 1943, the machine learning technique has in recent years had a resurgence in popularity and further development, and has shown promising results in areas such as speech recognition, vision, and bioinformatics.

To explore the feasibility and accuracy of using neural networks to detect signs of default, a recurrent neural network - a subtype of neural networks - will be trained on a data set of credit card data and subsequently analyzed. The problem will be defined as a classification problem, where we look at a new customers first 12 months of account data, and depending on if they have defaulted within this period, try to correctly classify them as being either a 'good' or 'bad' customer. **This thesis aims to conclude if neural networks could be efficient in detecting defaults, and furthermore to examine which properties of the neural network has an impact on its efficacy.**

1.2 The Norwegian Credit Card Market

The Norwegian credit card market consists of a multitude of actors, from larger banks such as DNB and SpareBank1, to smaller companies specializing in credit cards and consumer loans. During the last couple of years, the government of Norway, as well as experts in financial fields has expressed concerns of the rising use of credit cards and the rising debts that have followed.

Norway is a technologically advanced country, where financial institutions have been quick to adopt technology into their services, and where consumers are eager to use systems that makes life easier. In 2015, Norwegians used their credit- and debit-cards on average almost 400 times during the year, in contrast to Germany, another technologically advanced country, where citizens on average used their cards less than once a week. Cash is used in less than 5% of all transactions in Norway, compared to an average of around 15% in the Euro-area, or over 45% in the USA (NoregsBank, 2017).

Norway is on the forefront in the use of credit cards, and with the low costs of using them, many consumers own multiple cards. In 2016, there were almost 7 million credit cards in circulation among Norwegians, in a country with a population of just over 5.2 million people. Many credit cards can be owned without paying a yearly fee, making it easier for consumers to afford getting more than one card.

For the banks, the consumers use of debit- and credit-cards is in their interest, as it is cheaper and more profitable than handling large amounts of cash. In 2016, 64% of all transactions in Norway was done with cards, which netted financial institutions 4.7 billion kr in transaction fees alone.

Although the rising use of credit cards in Norway is profitable for financial institutions, the growing consumer debts have raised concerns that it's not sustainable, with fear that the growing amount of unsecured loans could have a negative effect on the Norwegian economy. This has caused Finanstilsynet to tighten regulations in an effort to slow the growth in consumer debt and defaults.

But for the financial institutions, tighter regulations are not necessarily a bad thing. Neither consumers, nor banks profit from consumers defaulting on their loans. Tighter regulations, better selection processes of who gets granted loans, as well as better follow-up of existing customers can all lead to larger profits for banks if it keeps the default rate low. This thesis aims to conclude if neural networks can be used by banks in order to identify defaults.

Chapter 2

Background

This chapter will examine the problem area through existing scientific literature. It starts off with an explanation of the problem area, before going into some common solutions to the problem area. The chapter rounds of with an analysis of some related works that are close in topic to this thesis.

2.1 Problem Area - Credit Scoring

Credit scoring is the methods and techniques used by lenders to forecast the financial risk of lending to consumers. By using historical data and information about the customer, the lender can use statistical and machine learning techniques to infer the risk of granting a new loan or to reassess the loans of an existing customer.

Before credit scoring became universally adopted, banks would rely on their employee's local knowledge and "gut feeling" when giving out loans. This meant that banks needed a local presence in the area where their customers where situated to get the best understanding of their customer's situation.

The explosive growth in the amount of credit cards and loans, combined with the centralization and merging of banks made the traditional method of "gut feeling assessment" of risk infeasible for most banks. A small-business loan could in the past take up to 12-1/2 hours to process (Mester, 1997). The sheer number of loans and long processing times meant that banks needed more efficient methods of estimating customer risk, while cen-

tralization caused banks to adopt methods that relied less on their employee's personal knowledge and more on quantifiable data.

One of the first instances of credit scoring came during the second world war when credit analysts were drafted into military service. The credit firms got their analysts to write down the rule of thumbs they used when evaluating customers; these rules could then be applied by less experienced analysts and is an example of an expert system (Thomas, 2000). As time went on, many statistical techniques were developed, such as probit models, linear probability models, and discriminant analysis.

2.1.1 Data Analysis & Pre-Processing

The methods of credit scoring are different in the assumption they make of the data, and in how they process it, but the results of the methods are inherently the same, an estimation of risk. There are two ways in which the credit risk problem can be modelled. Either as a regression (statistical) problem, or as a classification problem. In regression, we try to estimate the probability of a relationship between two or more variables. The output of a model using regression is a probability between 0 and 1. In classification, we try to infer what class a new data point belongs to. This classification can come with an accompanying probability of certainty.

The biggest difference between the two variations of modelling risk is that the statistical result needs a cut-off threshold to put customers into the two bins of "will be given a loan" and "will not be given a loan". The analyst or firm using the model must have a guideline on their risk tolerance, for example only accepting applicants whose credit score is over 0.80; giving them an 80% chance of paying back their loan. This could be a double-edged sword, as on the one hand it allows the firm to set their risk-tolerance, while on the other it puts us back to a "gut-feeling" decision of what is acceptable risk. A discrete result makes it easy to select who will be given a loan and who will not: give a loan to the customers that the model deems will "not default". This however means that one must trust the method.

Justifying a loan can in many cases be an important necessity, or even a requirement by law. Laws such as the American Equal Credit Opportunity Act (ECOA) prohibits lenders to discriminate their customers based on colour, race, religion, and sex when assessing

who should be given a loan. It is hard to justify a rejection of an application that uses "gut feeling" to assess an applicant, as the subjective nature of the method and the assessor's personal prejudices is bound to play a role in the outcome. By using a statistical method, one can omit data such as sex and race when developing the model, which in turn gives credibility to the objectivity of the model. However, there is a concern among researchers that there exist surrogate variables that captures data such as sex and race, and by using these variables, the models discriminate in these areas (Thomas, 2000).

The data used in the creation of credit score models plays a huge part in the model's predictive ability. As earlier mentioned, some data such as sex and race can in many cases by law not be used, even if they should show predictive ability.

2.1.2 Volume of Data

Credit firms gather lots of data on their customers and it is not uncommon to have databases consisting of more than 100,000 customers measured on over 100 different variables (Hand and Henley, 1997). Databases that contain past payment history can have even more variables. As is often the case, more data is usually better, but research has shown that even limited data can yield good results. A study by Sustersic et al. (2009) using neural networks and logistic regression on 21 variables (from an initial data set of 67 variables) managed to achieve a 79.3% average prediction accuracy on one of their neural network models.

Handling large volumes of data can be both expensive and complex, and even though more data is usually better, using a subset of the gathered data in the credit score model can often be good enough. One must take into the account the diminishing returns of adding more variables to the model. In data sets of over 100 variables it is common to find instances of high correlation between variables, in which case, it could be justifiable to include only a subset of the correlated variables.

One way to reduce the dimensionality of data is to use expert knowledge, experience, and judgement to get a feeling for which variables it makes sense to include. Even though using experience to assess data can be questioned, using it complementary to statistical and machine learning methods can make the model easier to justify. A system too complex that outperforms a simpler system could be unappealing to management if it can't be explained

why it includes certain data.

Dimensionality in data is one aspect important to credit scoring, another is volume. Having 100 different variables of data means nothing if we don't have the volume to find patterns between customers. When it comes to volume, more is almost always better.

To explain why more is almost always better we can look to the law of large numbers. The law of large numbers (LLN) is a statistical theorem that describe the result of performing the same experiment a large number of times. The theorem states that the average result of an experiment should be close to the expected value, when a large number of trials has been performed, and should move closer to the expected value as more trials are performed. The parallel to this for data sets is that we should expect patterns to emerge in our data set when increasing the size of our data. Although this might be true, we should keep in mind that correlation does not necessarily mean causation.

So how many customers should we have in our data set to be confident that our data set captures a representative amount of the population? This is a hard question to answer, and there is really no right answer, but the more customers we have, the more confident we can be that we capture the trends in the overall population.

Although we want to have a large number of customers in our data set, as we increase our data set, we can expect to increase the noise in the data as well. Although we can expect the noise to increase linearly with our data, more data volume means more noisy data to handle; increasing the time it takes to process our data. Noisy data can be a big problem and we generally want as little noise in our data as possible. We classify noise as data points that are either corrupted, distorted, or outliers in our data set. In real life data, there is bound to be some noise, so we need techniques to filter out the noisy data.

These techniques include cleaning up data that might be distorted, i.e. making sure data that means the same thing is identically represented. We need procedures for handling missing values; whether a missing value should get a default value, be inferred from other data, or removed is an issue that needs to be addressed. Outliers are data points that is drastically different from other data points, but might still be valid, but could skew the results. It is normal to remove outliers from data sets to keep the rest of the data as uniform as possible, thus increasing our prediction accuracy for the majority of the population.

2.1.3 Variety and Veracity of Data

The data that is gathered for credit scoring is usually very varied. We can have quantifiable data such as income, outstanding loans, and number of children. But we can also have qualitative data with categorical answers such as yes or no: marital status, car ownership, home ownership, etc. Choosing the categories of which to group customers is an important step in the creation of a credit model, or as Thomas (2000) said it, ”The art of credit scoring – choosing sensible categories”.

The easiest form of categorical data is binary; an answer to the question, “Do you own a car?” can yield two answers, yes or no. In a credit model, the two answers would be given different weights depending on how important they are to the overall risk. The same holds for multivariate categories with more than two possible answers.

The tricky part is how the answers to the questions should add up to the final credit score, and how we should model the risk within each category. If we plot default risk with age, we do not get a straight line, but instead a non-linear curve, which is a common phenomenon. This means that we cannot model the age variable linearly, but have to come up with an alternative representation. One way to model the default risk would be to model it as a more complex curve. Another option would be to group the age into different categories, where each category has homogenous risk within the category, but different risk from other categories.

2.1.4 The Importance of Time

Population drift explains the tendency of a population to evolve over time, so the distributions change, which is a problem for credit scoring. Credit scoring techniques assume a static population where the default risk of a borrower today, would be the same as a borrower two years from now. This is not necessarily the case when the demographics of a population change.

To combat population drift, one has to either have models that can be updated continuously with new data to take account of the drift, or one must periodically generate new models. For the latter, there is the problem of how often this should be done, and if it should be done before there are any signs of a drift or after. Then there is the problem of even detecting a population drift. This might be noticed by seeing a change in the default

rate, or in individual variables.

One question related to time is what is a suitable time horizon for prediction. The norm seems to be between 12 and 18 months (Thomas, 2000). Any less and the data has not had the time to stabilize. Having a longer time horizon opens up for population drift as explained above.

2.1.5 Benefits and Limitations

There are many benefits to using credit scoring. As stated earlier, it can give us improved objectivity in the loan giving process. Not only can we distance the human factor from the process, but omitting the use of discriminatory data in the model gives it greater credibility in that it treats each customer equally and fairly. Improved objectivity is one thing, but by using credit scoring we are able to explain the reasoning behind our decision of whether to grant a loan. This mainly applies to statistical methods where we can look at how the model distributes the weights of importance on each variable in the data. Neural networks operate in a black box environment, where we cannot look inside the model and see the reasoning behind its results; this is one of neural networks biggest drawbacks.

Other benefits of credit scoring are the time- and cost-saving factors. After the initial creation of the model, much of the work previously needed to be done by employees are now performed automatically by the model. One must only feed the data on the new applicant into the model, and after processing, we get a result back on if we should give out a loan. In a fully built out system with automatic information gathering, data handling, and reporting, the human work needed to be done is almost zero. This translates to big time- and cost-savings for the company, as they can employ fewer analysts, and the analysts they employ have more time to review special cases that might need extra considerations.

Although there are apparent benefits of credit scoring, it also comes with its limitations. The black box nature of neural networks means that we cannot use this type of model to explain the reasoning of the result it gives; we just have to “trust the method”. Not being able to give a reasoning behind the result is in itself not necessarily a bad thing, but it is a shortcoming of this particular technique.

A limitation that applies to all the techniques of credit scoring is that it generally has a hard time dealing with underrepresented groups. As credit scoring techniques rely on

the similarity between customers to infer results, it means that we will get better results on groups that contain a disproportionally larger number of people. The data on which the model is based, should be balanced around the variables it contains to capture a wide variety of groups and to ensure that the number of underrepresented groups or characteristics is kept to a minimum.

Similarly, the model should ideally also account for external factors. Including external factors in the model is a challenge, and is generally not done, at least not in most academic research. But only including data from customers during times of economic expansion could give contrasting results if the model is used during recessions, where the economic factors are different.

Credit scoring has been an integral part in the credit business since its adoption a half-century ago. Although the techniques and models used are changing and evolving, the principal is the same: having an effective and efficient technique to forecast the financial risk of lending.

2.2 Common Solutions to Problem Area

There are at least four categories of techniques for creating credit scores: subjective scoring, expert systems, statistical, and machine learning. Within these categories there are many different techniques with its own weaknesses and strengths, and although their methodology is different, the end goal of producing a predictive model is the same.

2.2.1 General Solutions

The general solutions are families of techniques that each can have widely different sub-techniques to solve a problem. Instead of going into each of these specific techniques, this section will explain the main ideas encompassing each family of techniques.

Subjective Scoring

Subjective scoring is the old way of assessing the creditworthiness of a borrower. There are no inherent rules or strategy, only the lenders subjective opinion of the borrower. In practice, a loan officer would consider the financial status of the borrower, the economic

situation in the area, and any other relevant information to each borrower on an individual case. This is both a laborious and unscientific process that has since been replaced with more efficient and structured techniques.

Expert Systems

Expert systems are a formalization and standardization of rules derived from the experience of managers and loan officers. These explicit rules can be used alone, or in combination with statistics and mathematics to create a scorecard (Constangioara, 2011). A decision tree is a simple expert system that uses experience to make decisions, not statistical analysis of the data. Statistical analysis can however be used to control the growth of the tree. Expert systems are usually used today in cases where extra consideration is needed.

Expert systems contain three main components: a knowledge base containing all the facts and rules, an inference engine that make conclusions based on the facts and rules, and lastly an interface for explaining the reasoning behind the conclusions to the users of the system (Rosenberg and Gleit, 1994). The rules in a well built out expert systems can easily number in the hundreds, where the rules can be modelled as conditional statements (if X, then Y).

The main limitation of expert systems is that they make their conclusions based on rules made by experts in their respective domain. There is no inherent learning in the system, as there is in machine learning techniques; this means that it is a laborious process of trial and error to optimize the model. Classification (decision) trees, is a machine learning technique that is based on the idea of expert systems, with the added benefit of automatically making its own rules and training its model to make as good a prediction as possible.

Regression

Regression is a family of statistical methods used to estimate the relationships among variables. Given a set of independent variables (variables used to make a prediction), regression is used to observe how these variables affect the dependent (outcome) variable. There are many different regression models and estimation techniques that may be used

depending on the desired outcome or on the type of data used.

One form of regression is linear regression, which is a linear approach for modelling the relationship between the dependent variable, and one or more independent variables. It gives good results if there is a linear relationship in the data. The assumption of a linear relationship gets harder to meet as the dimensionality of the data increases, and linear regression is therefore more suitable to data of a lower dimensionality.

The other two main types of regression are the probit model and the logistic regression. The probit model is a binary classification technique, where it is assumed that the sample can fall into one of two categories. The logistic regression is also a categorical technique that assumes a binary output, but a multivariate version of the technique also exists, which allows for multiple different output categories.

Neural Networks

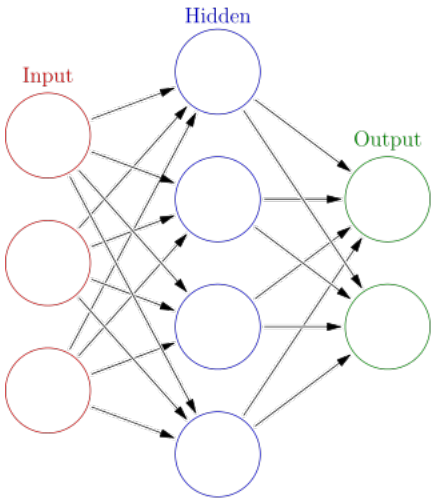


Figure 2.1: A Neural Network represented as a graph. Each circle represents a neuron, and the arrows denotes the relationship between the output of one neuron, and the input of another. Downloaded from: `https://en.wikipedia.org/wiki/Artificial_neural_network`

Neural networks (NN) is a machine learning technique inspired by the workings of the biological brain. It can be viewed as a graph consisting of neurons (nodes) and synapses (edges) as viewed in figure 2.1. The graph or network of neurons is generally organized in

layers, where each layer may perform a different function on the data. Receiving neurons can process input data and signal other neurons downstream from it.

Neurons may have a state that is changed depending on its internal function and the signal it receives. Neurons and synapses also have weights that varies as the neural networks learning proceeds, and which influences the strength of the signals that is sent from a neuron.

There are many different types of neural networks. From simple Feed Forward (FF) networks to more advanced Deep Convolutional Networks (DCN), the type of network is defined by the composition of the networks neurons and its different layers. A deep network is usually defined as a network consisting of at least two layers of hidden neurons (a hidden neuron is a neuron not defined as an input or output neuron, therefore what goes in and out of the neuron is hidden from the user). As the complexity and cardinality of the network increases, so does the training time of the network; there is a trade-off between model accuracy and training time that can quickly come out of hand.

2.2.2 Specific Solutions

The techniques presented in this section are more specific than the ones in the previous section. The reason for going more in depth on these techniques is their prominent status in both research and in industry.

Discriminant Analysis

Discriminant analysis is a statistical technique for classification, that is similar to logistic regression. The drawback of discriminant analysis is that it requires more assumptions about the data, and has more restrictions than logistic regression, and is for this reason less used. However, when assumptions are met, discriminant analysis is a powerful technique that works well on smaller data sets.

There are some main assumptions that has to be met before using discriminant analysis. The variables all have to be normally distributed, and have equal variance within each predictive group. The data also has to be independent, meaning that there is no correlation between one sample variable A, and another sample variable B. The last assumption can depending on the data set be hard to work around, but the first two can to a certain degree

be worked around by transforming the original data over to a normal form.

K-Nearest Neighbour

The k-nearest neighbour (k-NN) technique is a machine learning technique for grouping objects based on similarity. It partitions objects based on their similarity to the k nearest neighbours. k-NN belongs to the family of instance-based learning algorithms, which means that it does not require establishing a predictive model, but instead compares new problem instances to instances already seen during training.

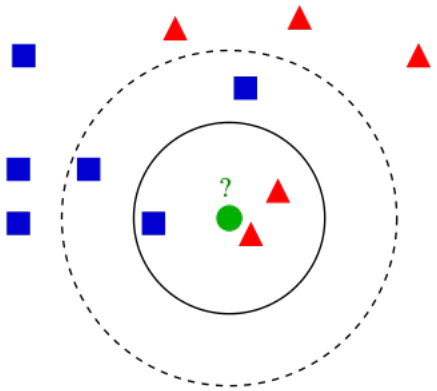


Figure 2.2: Example of k-NN classification. Downloaded from: `https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm`

Figure 2.2 shows an example situation of the k-NN classification problem. The green circle is our new observation which we want to classify. If $k = 3$ (we classify based on the three nearest samples), we can observe that of the three nearest samples (within the solid line circle), two out of three are red, as such our new observation gets classified as a red triangle. If we however set $k = 5$, we need to look at all the samples within the dashed line circle, where the three blue squares are in majority, which causes our new observation to be classified as a blue square. As just demonstrated, what we set the k value to, has a great impact on the classification outcome.

Disadvantages to this technique is that its highly sensitive to the distance function (how we define and calculate the distance between two samples), as well as the cardinality of the k-variable. Noisy data is especially damaging to this model, making k-NN most effective on data with clearly partitioned groups. The biggest limitation for k-NN is the curse of

dimensionality, which for k-NN means that an increase in the number of variables will result in an increase in similarity between data points; in turn reducing the dissimilarity between groups which is needed to properly partition the data set. This can be overcome by first reducing the cardinality of the data set, at the cost of having less variables used in the prediction.

Classification Trees

Classification trees (or decision trees, as they also are called) are structures where nodes denote tests of an attribute, edges the outcomes of these tests, and leaves being the resulting classes. Classification trees are good at displaying the non-linearity in data. After having built a classification tree, we can assign classes to new data points by feeding them into the tree, and traversing nodes based on the tests until we hit a leaf node. Classification trees are prone to overfitting data; meaning that if we are not careful with how we create our tree, we can end up with a tree that is excellent at predicting classes for our test data set, but may have poor performance on other data.

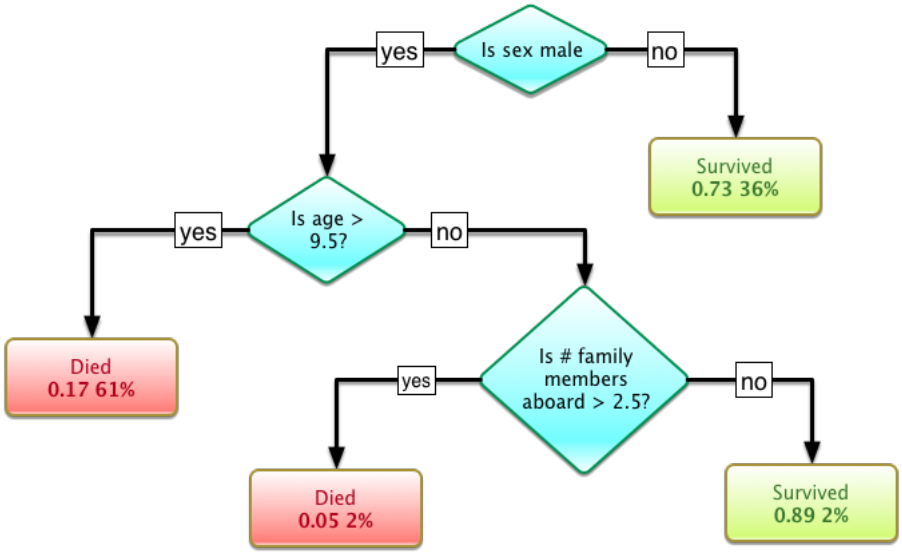


Figure 2.3: A classification tree showing the survival of the passengers on the Titanic. Downloaded from: `https://en.wikipedia.org/wiki/Decision_tree_learning`

Figure 2.3 shows a simple classification tree built from a data set containing informa-

tion on the Titanic passengers. The nodes (blue boxes) denote the tests used to classify a new observation, while the leaf nodes are the outcomes, in this case a binary feature with the possible values of *Died* and *Survived*. Traversing the tree, we are able to classify all new samples into either of the two possible outcomes. Although this is a functioning classification tree, more granularity in the tests would most likely result in better results; but we have to be careful not to add too many tests, as this might overfit the tree to our training data.

There are many techniques that can be used to improve classification trees. Pruning is the act of removing branches from the tree that hold little power in classifying data. This generalizes the tree, and reduces the chance of overfitting our data. Another machine learning technique that takes advantage of classification trees is the random forest. A random forest is an ensemble learning method where multiple classification trees are trained in parallel with different parameters. We can then take the mode or mean of the trees for creating our final model.

2.3 Analysis of Some Common Solutions

Both researchers and banks are continuously on a quest to find a better solution to the credit risk modelling problem; as the reward for even a modest increase in predictive accuracy could be very profitable. The invention of new techniques and the experimentation with parameters over the years does not seem to indicate that an optimal solution to the problem has been found.

However, one should not expect to find an optimal solution – as the problem space is too large – but rather a best approximation. How good this approximation can end up being is hard to say, but with correct classification percentages in literature usually ranging from a low 60 to a high 80 according to Abdou and Pointon (2011), one could expect to find better solutions.

So, what is the best technique for modelling credit risk? There is currently no technique that is universally regarded as the de facto standard of modelling risk. One can find research for almost every technique with results showing that this technique yields the best results. There is however a trend in existing literature as found in Yeh and Lien (2009) and West (2000), showing that neural networks consistently outperform or comes

close to other techniques. A study by Abdou and Pointon (2011), comparing results from published research, showed that in the six studies neural networks were a part of, it beat the other techniques four out of six times. This does not necessarily mean that neural networks are the best technique, but it highlights it as an important candidate for continued research.

This section will analyse regression, k-NN, and neural networks. These are all common but different techniques that can and is used for credit scoring. The other techniques described in 2.c although interesting in themselves, will not be analysed as most of their limitations and benefits can be encapsulated in the other techniques analysed in this section.

Some other noteworthy techniques that is not discussed in this thesis but has shown promise are: support vector machines (Zhou et al., 2010), extreme learning machines (Bequé and Lessmann, 2017), and hybrid approaches combining multiple techniques; such as neural networks and logistic regression (Li et al., 2016).

2.3.1 Regression

Regression is a well proven technique in prediction problems, being used in both simple and more advanced problem areas. The many types of regression are due to its assumptions about the data, and as with other techniques, it is important to choose the type of regression that best fits the data. The most well-known technique however is linear regression, which this section will be focused on.

The main limitation of linear regression is that it underperforms on complex data. Although credit data has been studied for a long time, it is still not widely understood why one customer might default and why another will not default. There are certain attributes that might be regarded as better indicators than others, but there is no consensus of a list of attributes that are must haves in a credit risk model.

Heteroscedasticity is a concept referring to when the variability of a variable is unequal across its range of values. The variability can be any statistical dispersion in the data, e.g. variance or standard deviation. When the variance among some variables is not finite, we say that we have heteroscedasticity. Another way to look at it is that we have diverging subsets within a variable. Figure 2.4 showcases this by plotting a person's income with

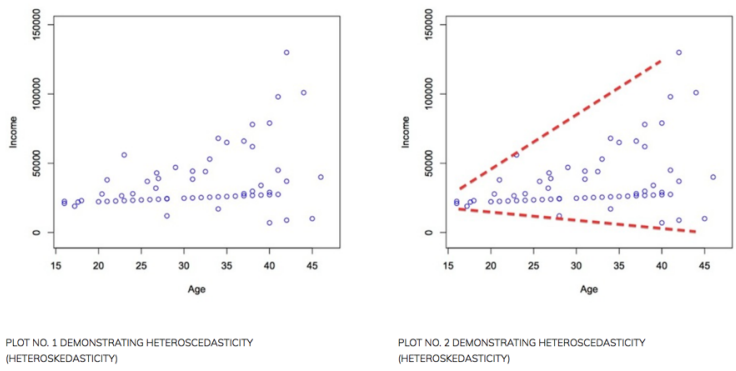


Figure 2.4: Two plots using hypothetical data explaining the concept of heteroscedasticity. Downloaded from: `http://www.statsmakemecry.com/smmctheblog/confusing-stats-terms-explained-heteroscedasticity-heteroske.html`

their age. As we come to the age of 20, we observe two subgroups appearing: one group continuing with a modest increase in income as they age, and another with a much larger increase in income.

The standard linear regression technique assumes that there is no heteroscedasticity, and non-linear techniques such as the logit and probit models can suffer from biases if used on heteroscedastic variables. This means that in theory we should be cautious of using regression on this type of data, but a study by Gelfand and Lockhart (2015) had some interesting findings. It found that linear regression in some cases beat many of the more modern techniques. This could either mean that the effect of heteroscedasticity and regressions assumption about it is not as severe as previously thought, or it could be the case that the data in the study could not properly encapsulate the concept of heteroscedasticity. In either case, heteroscedasticity is still a concept occurring in real world data and should be considered when working on and choosing a prediction technique.

Apart from heteroscedasticity, linear regression also has problems dealing with outliers, i.e. data points that falls far outside the rest of the data. The reason for this is that linear regression relies on calculating the mean of the data. This means that data points that has a significantly different value from the mean, has the effect of moving the mean in such a way that it can change the slope of the regression curve. This can be seen in figure 2.5, where the addition of the outlier changes the curve, and in turn the predictive

power of the model. This can be overcome by using other methods such as Bayesian linear regression which is more robust to outliers than the more standard method of 'sum of least squares'.

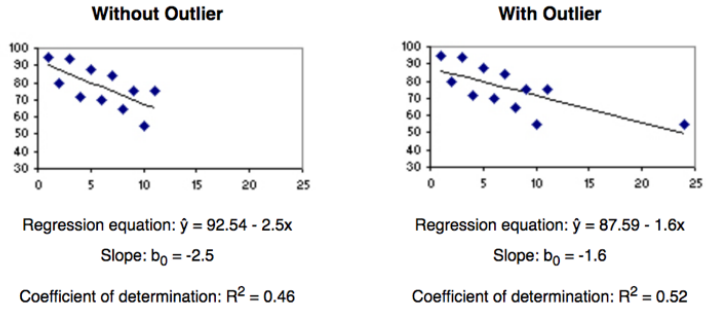


Figure 2.5: Showing how an outlier can change the slope of a regression curve. Downloaded from `http://stattrek.com/regression/influential-points.aspx?Tutorial=AP`

Lastly, linear regression has as the name implies an assumption of the data being linearly separable. A workaround is to try to transform the data to a linear form or use another type of regression such as logit, which assumes logarithmic data, or Poisson regression which assumes that the dependent variable has a Poisson distribution.

Even though regression, and especially linear regression has many limitations and makes many assumptions, it is still a good technique if the underlying assumptions holds. Regressions is a simple technique that can create good results, and its many variations allows it to be adapted to the data. This however requires a good understanding of the data and the attributes relationships.

2.3.2 K-Nearest Neighbour

The k-nearest neighbour (k-NN) technique is a non-parametric method, meaning that it makes no assumptions about the data. This is a useful property for more complex data as it loosens (not eliminates) our need to understand our underlying data, and allows us to focus more on fine-tuning the model. A study by Henley and Hand (1996) looked into the application of k-NN on credit scoring. When compared against linear and logistic regression, and decision trees; k-NN performed best out of the four techniques.

One of k-NN's largest limitations is its preference in having an equally distributed

number of dependent variables. In the case of credit scoring, this means an equal amount of good and bad customers. As an overwhelming number of customers are good customers that pay back what they owe, this is an especially bad limitation. The consequence of this is that the number of observations (customers) we can use is limited by the number of bad customers. This leaves us with the need to reduce the data set size substantially, which can have an impact on the predictive power of the model if we accidentally prune away customers that have an important predictive power.

The reason an unequal distribution of good and bad customers in k-NN is detrimental; is that the algorithm works by the concept of majority voting. The k nearest neighbours to our newest observation votes on which class the observation should be put into by doing a similarity comparison using a distance function (Euclidean and Manhattan distances are popular choices). A class with a disproportional large number of observation will have a statistical advantage of being chosen as the most similar observation, especially if k is large. This is worsened if the feature space of the data set is large, i.e. the data set has many attributes.

Another limitation of k-NN is that it suffers from the curse of dimensionality, which is why it is common to use dimensionality reduction techniques such as PCA (principal component analysis) on the data set prior to using k-NN. Credit card companies gather lots of data about their customers, which amounts to a large number of attributes that could be used in a credit scoring model. Choosing which attributes to include in a model, and which to exclude is not an easy task, and with k-NN being so sensitive to a large feature space, having this reduction as a necessary task is a major drawback of using k-NN for credit scoring.

As stated earlier, k-NN uses a distance function to calculate the similarity of two observations. The two largest considerations that needs to be taken into account is the choice of the distance function, and the format of the data. Many statistical and machine learning techniques can take advantage of having data on a scaled form. The main benefit of scaling our data to for example between 0 and 1, is that all attributes will be equally weighted (important) when using our distance function.

A case in which k-NN excels is if the data can be cleanly separated into clusters. This is most useful if the dependent variable has multiple possible values. For credit scoring this could be: good, medium, bad, extra consideration.

2.3.3 Neural Networks

Neural networks are also a non-parametric technique that is excellent at finding meaning in nonlinear data. As with regression, neural networks are a collective concept consisting of many subgroups. The different types of networks all have their strengths and weaknesses, but on a fundamental level shares many of the same benefits and limitations.

One limitation of neural networks is that they require considerable training time to be effective. The computational power and in turn the time it takes to train a neural network is usually much larger than other machine learning techniques. This has garnered neural networks criticism as not being applicable for real-world use. But with the ever-decreasing cost of computing power, and the continued research into more efficient ways of using neural networks, this criticism matters less and less. It is however important to keep in mind that neural networks are computationally expensive.

The cost of training a neural network depends on a multitude of factors. The number of attributes is for neural networks as with other techniques a limitation that can be overcome by reducing the feature space. Neural networks are also affected by the number of neurons that are put into the network, and by how many layers one chose to include.

Deep neural networks are networks with two or more hidden layers of neurons, and adding more layers can make the time it takes to train them significantly longer. It is therefore advised to use as few layers as one feels is needed. In theory, one hidden layer is enough for a neural network to approximate any measurable function (Hornik et al., 1989). But in practice, experimentation is usually needed to see if adding more layers makes an improvement to the model's accuracy.

Another disadvantage of neural networks is that they are in essence a black box. In the context of neural networks, this means that it is really hard to understand why a model works the way it does. As a neural network model trains on a data set, it continuously adjusts the weights of the different attributes, and uses these weights in the neurons to activate the neurons function. It is practically impossible to extract meaning from how the different weights and neurons contribute to the overall score, and as more layers and neurons are added, the complexity increases. Understanding how one neuron works is not enough as the problem primarily exist when we have multiple neurons that can depend on other neurons in a nonlinear way.

The black box nature of neural networks is maybe their largest problem. For many use-cases – credit scoring included – the ability to understand why a model works, and how it works is a major benefit. It allows the makers of the model to gain valuable insight into the problem area and understand the areas of their domain that makes the biggest impact. There is ongoing research into better understanding the theory behind neural networks and how they work, as well as building tools to get a glimpse into their inner workings (Yosinski et al., 2015). But as of yet, our understanding of how neural networks work internally is still limited.

2.3.4 What is the best method?

There is clearly no 'best' method that beats all other methods, as the different techniques each have their strengths and weaknesses. With all the techniques, the goal is to make an accurate model that approximates the problem area. But there will always be some limitation of a technique that has to be dealt with, and the question is which limitation is the easiest for the users of the model to live with.

With that being said the choice of technique usually boils down to the type of data. For smaller data sets with linearly separable groups, linear regression will almost always be a good choice, and there is usually no clear benefit to using a more computationally expensive technique such as neural networks. Likewise, convolutional deep neural networks have established itself as the leading technique on image recognition, outperforming simpler techniques.

It is in the area of complex problem areas with data sets consisting of a large number of attributes and often nonlinear data sets where the choice of technique matters the most. More research is needed to understand how statistical and machine learning techniques perform on larger and complex data sets and how we can further improve their accuracy. This thesis will attempt to further the research on the application of neural networks in credit scoring.

2.4 Related Works

Neural networks have existed for a long time, with the widely popular backpropagation algorithm having roots back to 1960. After falling out of favour for a period, neural networks have in the last two decades had a resurgence of interest due to the decreasing cost of computing power, and the discovery of neural networks predictive capability on image- and speech-recognition. There has been developed many different types of neural networks over the years. Two of the more common ones that encapsulate most sub-types are: feed-forward and recurrent; with each having numerous sub-types.

Feed-forward neural networks (FNN) are arguably the most common type of neural network, consisting of a network where information moves from the input nodes to the output nodes. The different sub-types depend on the configuration of the nodes and the type of activation function used in the nodes. Networks with two or more hidden layers of neurons (nodes) are usually called deep neural networks, while networks that use convolutional techniques (used in image recognition) are called convolutional neural networks.

Recurrent neural networks (RNN) like feed-forward networks deliver information from the input nodes to the output nodes, but can in addition retain information inside the neurons. This is done by introducing a memory cell to the neurons that remembers previously seen values. This property allows recurrent neural networks to process sequences of data; making RNNs suitable to use on data with a temporal property. There are also unsupervised versions of recurrent networks, such as the self-organizing map, which maps an input space to an output space without any interaction from the user; in essence, finding meaning in data without knowing what it is looking for.

2.4.1 Neural Nets Versus Conventional Techniques in Credit Scoring

A paper by Abdou et al. (2008), aimed to investigate the application of neural networks in credit scoring in the Egyptian banking sector. The paper focuses on two types of neural networks, probabilistic neural nets (PNN), and multi-layer feed-forward nets (MLFN), and compares them to three other credit scoring techniques: discriminant analysis, probit analysis, and logistic regression.

The different techniques were trained and evaluated on a dataset provided by an Egyptian bank. The data set consisted of 581 personal loans, where the number of bad loans

amounted to 25.5% of the total loans. There was a total of 20 independent variables in the data set, which of 12 were selected to be used in the models. The dependent variable took on two values, “1” if the loan was good, and “0” if the loan was defaulted on.

The results after comparing the different techniques against each other, showed a clear advantage to the neural networks. Only the neural networks achieved an overall classification rate of more than 90% correct classifications. The probabilistic neural network achieved the best score, with a correct classification rate of 96.21%. However, if looking to minimize the misclassification cost, instead of maximizing the correct classification rate, the MLFN model proved to be a better option than PNN.

The motivation behind the paper was to evaluate different types of neural networks and compare them to traditional credit scoring techniques. The paper recommends future studies to look at other advanced statistical scoring techniques, as well as integrating existing techniques. The paper also recommends using more data, and more variables to increase the accuracy of the models.

2.4.2 A Neural Network Approach for Credit Risk Evaluation

The paper “A Neural Network Approach for Credit Risk Evaluation” (Angelini et al., 2008), aimed to apply neural networks to credit risk assessment. They developed two neural network systems, one feed-forward neural network model, as well as a special purpose architecture. The models were evaluated on data related to Italian small-businesses.

The data set consisted of data on 76 small Italian businesses, with 11 features for each business. For each business, the data set had annual data over three years. There was a total of 48 “good” businesses, and 28 businesses classified as “bad”.

The feed-forward neural network was constructed as a network consisting of an input layer, two hidden layers, and an output layer consisting of a single output neuron as shown in Figure 2.6. The number of neurons in the hidden layers were varied to be between 25 and 33. This network trained using the backpropagation technique managed to get zero misclassifications on the training set, while only getting an error rate of 8.6% on the test set.

The special purpose architecture, or the ad-hoc network as the paper also calls it is a four-layer network where the input neurons are grouped in threes. Each group of three

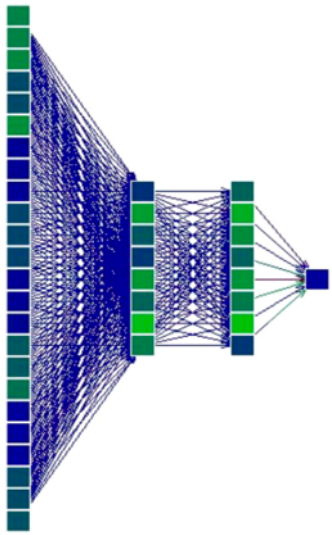


Figure 2.6: A four-layer feed-forward neural network. (Angelini et al., 2008)

neurons is then connected to a single neuron in the first hidden layer as shown in Figure 2.5. The first hidden layer is then connected to the second hidden layer, which again connects to the output neuron. This network also achieved good results, with no errors on the training set, and only a 4.3% error rate on the test set.

The conclusion of the paper is that the results can be considered as "state-of-the-art". The results are seemingly good, but when looking at the average of the total errors we see numbers that are more in line with other existing literature. The feed-forward network has an average error rate of between 11% and 14% depending on the configuration of the network, while the ad-hoc network has an average error rate of 7%.

What this shows, and which is brought forward in the conclusion of the paper is how important the data analysis and selection part of the process is. As real-world data is often incomplete and noisy, properly cleaning and pre-processing the data, as well as carefully choosing the right variables is maybe the most important part in the model building process.

For future work, the paper suggests focusing on procedural techniques for data analysis, parameter optimization, and processing, as well as testing the models on wider sets of data. They also mention looking at recurrent neural networks.

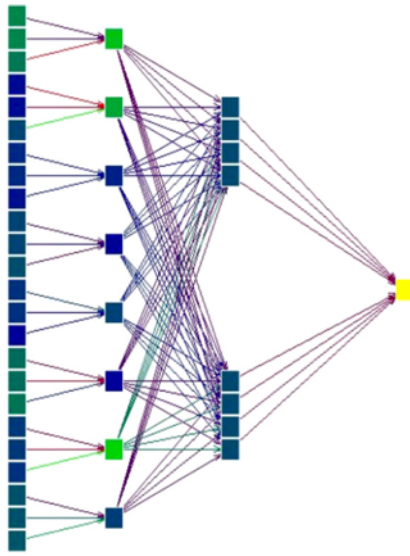


Figure 2.7: The ad-hoc neural network. (Angelini et al., 2008)

Chapter 3

Dataset

The data set used in this thesis is provided by SpareBank1, a consortium of Norwegian savings banks. The data spans from November 2013 to April 2017, and has data on more than 160,000 unique customers. The data consists of three separate data set files: transactions, aggregated data, and tax data. This chapter starts with an introduction of the data, before going into an analysis and discussion. Finally, there will be a discussion of how to define risk, which will be the dependent variable in the experiments presented later in the thesis.

3.1 The Data

This section will cover the three different data sets and give a rudimentary overview of each. Due to privacy and the sensitive nature of the data, all identifying information has been removed from the data set by SpareBank1.

3.1.1 Aggregated Data

The aggregated data consists of 2,199,272 rows of data spanning 99 features (columns). Each entry (row) consists of data related to a specific month and year. The features in the data set can be classified to be dynamic, static, or labelled as an identifier flag.

The static features are features that do not change, but adds additional information. Examples of such features are: account id, gender, account creation date, and which bank

the account was created in.

The identifier flags are binary flags which are set based on certain conditions. Examples include ”OverLimitFlag” which is set to one if the account has gone over its credit limit within the current period. Another example is the ”CollectionFlag” which signals that the account has defaulted within the current period.

The last type of feature is dynamic features which can change from period to period. Some features such as ”CreditLimitAmt” (credit limit amount) rarely changes, while other features such as ”MonthInDCA” (months the account have been in default) incrementally updates for each month the account is in default. There are also a variety of features that holds values on how much credit has been used, how much has been paid in fees, as well as other aggregated features related to payments.

3.1.2 Tax Data

The tax data set consists of data from 154 thousand accounts; it has 15 features that include data on both loan amounts and taxes. The loan features include amounts of different types of debt the customer currently has, such as student loans, car loans, and mortgage loans. The tax features cover how much the customer pays in taxes, and how much they earn.

3.1.3 Transaction Data

The transaction data set is the largest of the three with a total of 8.98 million rows spanning 33 features. Each row in the data set corresponds to one transaction, for which the features describe that transaction. The features encompass information such as the amount and currency used, different identifier flags, and additional descriptive information like type of store and country of said store.

3.2 Analysis of the Data

Of the three data sets, the aggregated data is arguably the most interesting one. It captures data on a granularity which is not too coarse, but also not too fine. The transaction data has good data if deeper understanding of each customer is needed, while the tax data is supplementary data which could be used in conjunction with the aggregated data. To limit

the scope of the thesis, only the aggregated data will be used, and sections 4.3 and 4.4 will go into details of how the data is used. This section will give a short analysis of the data found in the aggregated data set.

3.2.1 Characteristics

The data set with aggregated features consists of data spanning 41 months; the distribution of data points can be seen in figure 3.1. The distribution shows a sharp increase of data points up until around late 2015 where it stabilizes. The reason for this is that the dataset includes data for accounts in the 18 first months since the account was created. As more and more accounts are created, the cumulative amount of data points increase.

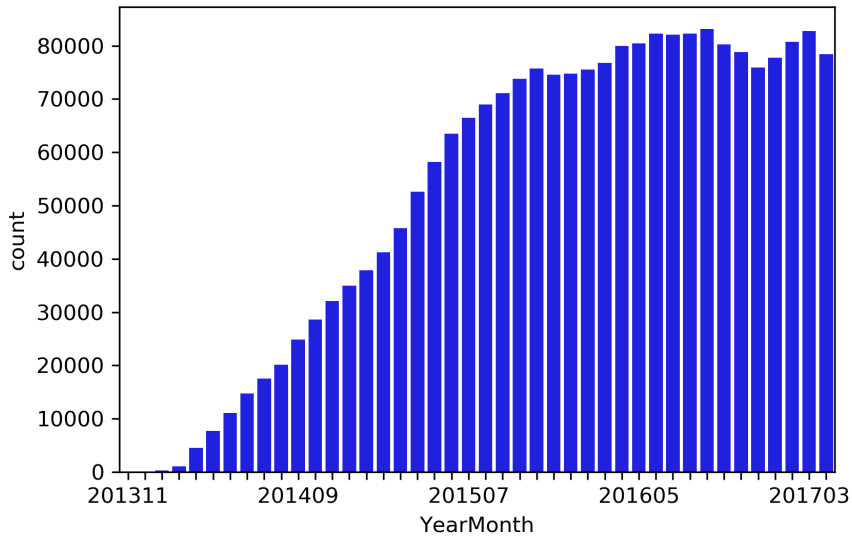


Figure 3.1: Distribution of the YearMonth variable

Figure 3.2 shows the distribution of data points by how long since their corresponding accounts where created. We can observe that the number data points steadily drop as the MonthsSinceAccountCreated variable increases. As we are modelling a time series, we preferably want complete series that include data for all months; if we don't have a complete series, we need to predict the missing values, which in turn adds unwanted uncertainty to our model. If we want to look at accounts in their first 12 months since

creation, the number of accounts we can use is limited by the number of accounts that has data in all of those months. This means that even if we have over 160,000 unique accounts, we only have about 110,000 accounts with data for 12 months after the accounts were created.

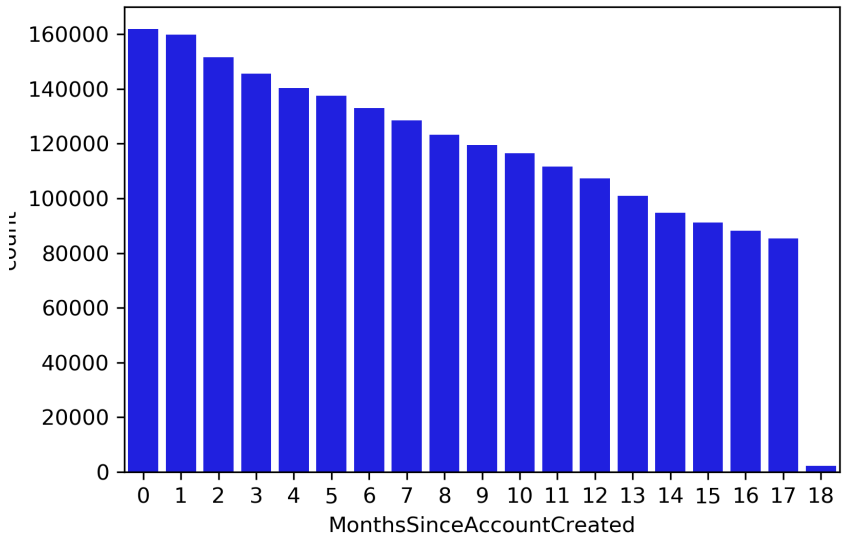


Figure 3.2: Distribution of the MonthsSinceAcconutCreated variable

The distribution of the age of the customers as shown in figure 3.3 shows the larger picture of which kind of people is represented in the data set. As the figure shows, there is a spike of customers with ages in the mid 20's, and there is a falling trend in the number of customers as the age increases. The age in itself is not particularly interesting, but plotting the customers age against the number of accounts that has defaulted as shown in figure 3.4 gives interesting information.

Figure 3.4 shows no disparity between the count of ages in the data set, and the number of defaults for each age group. The slope from the mid 40's age group and up is steeper for the defaults, which tells us that younger people is disproportionately more likely to default than older people.

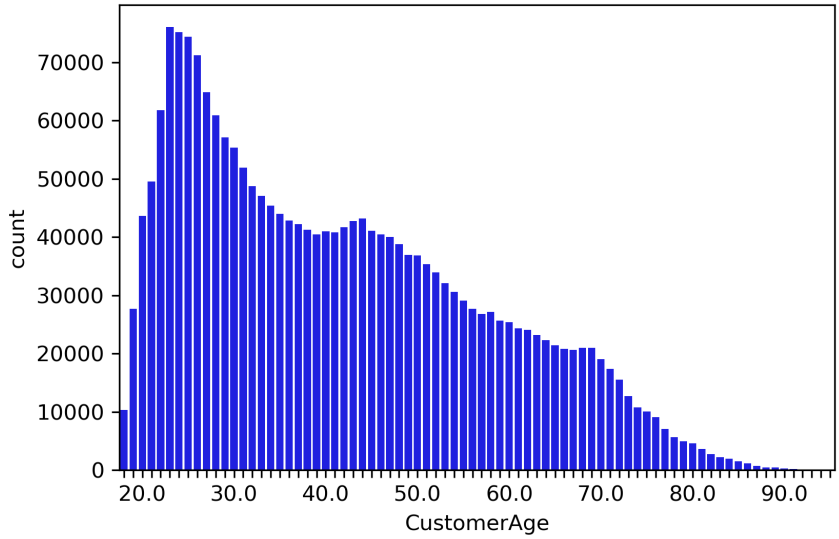


Figure 3.3: Distribution of the CustomerAge variable

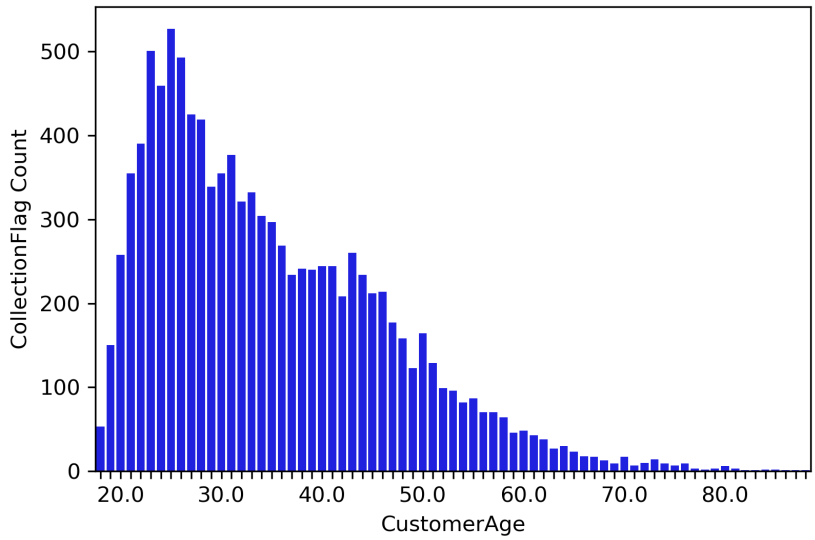


Figure 3.4: The number of defaults plotted against customers age

3.3 Feature Selection & Transformation

Feature selection is the act of selecting a subset of features from a data set for use in a model. There are multiple reasons for doing feature selection, some of the more important ones are: shorter training time of the model, simplification, reducing overfitting, improving accuracy, and avoiding the curse of dimensionality.

For some models, such as k-NN, reducing the dimensionality of the data is crucial when dealing with larger data sets, while neural networks does not suffer from the same negative effects. The curse of dimensionality as it is called, occurs due to the exponential growth of the search-space caused when increasing the number of features. The area of the search-space can be calculcated by n^m, where *n* is the number of samples in the data set, and *m* is the number of features. Adding a new sample to the data set will not affect the search-space much, but adding features will result in an explosive growth.

Two beneficial reasons for reducing dimensionality are: simplifying the model, and improving the training time. The number of features in a model is correlated with the running time, so if we want to have a shorter training time, one option is to reduce the number of features used in the model. Reducing the number of features also makes the model easier to interpret. Having hundreds of features makes it hard to understand what causes the model to perform the way it does, while limiting the number of features can make it easier to deconstruct the model and understand the impact of each feature. This is especially true for a neural network, which has no inherent practice of allowing us to look into the model and understand how each feature affects the result. The consequence of reducing the number of features is a possible loss of accuracy, as more data usually yields better results.

It is generally accepted that adding more data to a model is positive, but it is not always the case; some data may simply not have an effect on a model's predictive ability. Reducing the number of features in a data set does not necessarily mean removing data; another common method is to reduce features through feature transformation. Feature transformation is the act of creating new feature from already existing features by combining multiple features into one. This can either be done for the goal of dimensionality reduction, or for improving accuracy by creating a new feature whose predictive ability is higher than the individual features of which it is created from.

One use-case for feature transformation is creating features which is a ratio of other features. For a data set with financial information, the absolute values of earnings, debt, and other similar features may be of some value, and the model might be able to pick up that a high spending amount to credit limit ratio might be bad; but it can be easier for a model to understand the connection by making a new feature of spending to credit limit ratio.

Finally, feature selection can also help in reducing overfitting. Overfitting is when a model models the training data too well. This is a problem when using the model for new data, as the model has learned to look for specific values and noise that might not exist in the new data. Reducing the number of features is one way to avoid overfitting, as there is less data for the model to learn from.

Feature selection and transformation can be a powerful tool when constructing a machine learning model. It might not always be necessary, but it can often help in getting a better prediction accuracy. Feature selection is not an exact science, so understanding the problem, the data, and how the machine learning technique uses the data is crucial in constructing a good model.

For this thesis, we have kept the data set more or less intact. This has been done as the focus of the thesis is on the efficacy of the neural network on this type of data and problem, and not on optimizing the data set. We have removed features that are equivalent to the dependent variable, as well as features that have no predictive ability, such as *id* and *date* features. Furthermore, we have introduced three new features, that are ratios of existing features. A list of all included features can be found in appendix B.

3.4 Defining Credit Default Risk

In the context of credit scoring, risk can be defined in more than one way; it is therefore necessary to define what this thesis deems as risk. Risk is defined on Dictionary.com as ”exposure to the chance of injury or loss”. In credit scoring the injury or loss we are concerned with is the money lent to a customer. But we can choose to define risk in different ways, e.g.: as losing all the money, or the customer not being able to pay back the money for a certain period (but he might be able to pay it back later). This section will clarify exactly what this thesis defines as risk.

The simplest way to define risk is the customer not paying back the money, and the bank having to close their account. This is the worst-case scenario for the bank, possibly having lost their whole investment. However, before coming to this point, the bank would have sent default notices to the customer, demanding payment back. Many customers will at this point pay the fee accompanying the notice, as well as resume paying the monthly instalments. But there will also be some customer who will end up defaulting, which is to be expected.

Looking only at customers who defaulted and had their account closed is one possible solution, but it is both difficult and one could argue not the best way of doing it. It is a difficult task because the number of customers who have their accounts closed is such a low number out of the total number of customers. Gaining any insight into what differentiates the defaulting customers is difficult due to the class imbalance of 'good' and 'bad' customers. But the problem being difficult should not be a deterrent to trying to do it. However, not only is it difficult, but depending on what we want to achieve, looking only at defaulted customers with closed accounts might not be the right approach.

The problem with looking at only defaulted customers with closed accounts is that we might end up with predicting customers that are close to default, and for who it is too late to help getting back to being a good customer; which can pay back what they owe in monthly instalments. If we are able to detect customers early enough, then we have more of a possibility to take action in helping the customers, and in turn increase the banks' profits.

This is why **this thesis defines risk as having gone to default for up to three consecutive months**. We shall now go into the reasoning behind this choice. Firstly, choosing to define risk as having defaulted, but not having the account closed gives us an earlier detection chance of risky customers, as closed accounts have usually been in default for several months. Secondly, we have defined risk as being in default for up to three months, and not exactly three months. This is due to the low number of accounts that have been in default for exactly three months. In order to have more data to look at, we also include customers that have been in default for only two months or one month. The alternative would have been to include accounts with default for three months or more, but this would have left us with data on customers that are even more risky, or which have had their accounts closed.

Three months has been chosen to give an early enough detection chance and is com-

monly used as the standard definition for a 'bad' loan in other research (Henley and Hand (1996); Thomas (2000); Karim et al. (2010)).

It is important to clarify that what we have defined here is risky behaviour, and that the customers classified as 'bad' customers still might fulfill their obligations to the bank. What we define as risky behaviour will imposingly dictate the results of the thesis, and as such the results of this thesis could differentiate from the results of other research papers whose definition of *risk* might not be identical. In the third experiment of this thesis, we have looked at how changing the definition of risk affects the neural networks ability to correctly classify good and bad customers.

3.4.1 Risk in the Data Set

Having given an overview and definition of risk, we need to choose how to represent it based on the data in the data set. There are a number of features that capture the state of a customer's default in the data set. However, there is one feature which perfectly matches our definition of risk; the feature named 'MonthInDCA'. This feature records how many months the account has been in default. So, looking at accounts that have a maximum 'MonthInDCA' value of three will give us all the accounts that we have defined as being owned by risky customers. In the data set given by SpareBank1, the number of accounts who have been to default for up to three months is 2627.

Chapter 4

Methodology

This chapter will consist of the practical aspects of the thesis: the choice of framework, the transformation of the data set, and the implementation of the neural network.

4.1 Frameworks

There are many machine-learning frameworks that have neural network capabilities. Some of the most popular are Tensorflow, Theano, and Torch. There are also several other less known frameworks, as well as wrapper-frameworks such as Keras whose sole purpose is making other frameworks easier to use by abstracting away complexities.

4.1.1 Tensorflow

Tensorflow is a machine-learning framework developed by Google, and released in 2015. It has an API for various languages, but Python is recommended for stability reasons; it can be run on both CPU's and GPU's.

In Tensorflow, data is represented by multidimensional arrays which are called tensors. Computations on tensors is done by building a computation graph, where computation nodes take tensors as inputs, and output their results to other computation nodes or tensors.

Tensorflow's API include multiple different operators, layers, and functions. One of Tensorflow's strengths is the ability to both define computation graphs directly, or use

higher-level abstractions such as estimators, which simplifies the building of graphs by building the graph based on the input to the function.

4.1.2 Theano

Theano is another machine-learning framework, which originated from the University of Montreal. It is a Python library that has tight integration with NumPy, a Python library for working with large multi-dimensional arrays and matrices. As with Tensorflow, Theano also has the ability of using both the CPU and the GPU for doing computations.

Theano is generally more efficient than Tensorflow, but can be harder to use due to its lower-level language, with less abstractions. This can be overcome by using frameworks like Keras, which can use Theano as the backend, while making it easy to write computation graphs.

On 28. September 2017, the developers of Theano announced that further development will cease after version 1.0 is released. This was in part due to the emerging of other libraries, which has resulted in an abundance of frameworks similar to Theano.

4.1.3 Torch

Torch is a machine-learning and computation framework released in 2002. It uses a scripting language based on Lua and is used by major technology companies such as Facebook and IBM. As with most other leading frameworks, Torch can be used on both CPU's and GPU's.

Similar to Tensorflow, Torch uses tensors as the primary object, which can be fed into other packages to be performed operations on. Torch has a neural network package which gives good flexibility in how the network is constructed, as well as being easy to use.

4.1.4 Choosing a Framework

Most frameworks share the same functionality and can in turn construct the same neural networks. The major difference seems to be how computation graphs are constructed, i.e. how easy it is to use the framework. Frameworks that offer a higher-level approach of constructing computation graphs while keeping the ability to fine tune the models are

preferred for this thesis, as they make prototyping easy, while not sacrificing too much control over the finer details.

Another consideration is access to good documentation and support. When using a framework, it is not uncommon to come across difficulties and obscure errors reports. Access to good documentation and support allows for faster diagnosis of problems, which can be a major time saver when working on complex solutions.

This thesis will use Tensorflow with the integrated Keras wrapper for creating the neural networks. There are multiple reasons for choosing Tensorflow. It has established itself as one of the prominent machine-learning frameworks with thousands of users, which gives good access to both documentation and support. It is also backed by Google, a major player in the advancement of artificial intelligence and machine-learning, which lends credibility to the continued support of Tensorflow.

Another advantage of using Tensorflow is its ease of use. The Tensorflow package includes many higher-level abstractions, which makes it easy to construct neural networks fast. Keras, a popular higher-level abstraction, is included in the base package of Tensorflow. While it is possible to use these abstractions, Tensorflow also allows for detailed control of the computation graph by allowing users to construct the graph node by node.

Although most frameworks allow for the construction of similar computational graph, Tensorflow used trough the Keras package, gives a good trade-off of complexity and control, and is used in this thesis.

4.2 Data Transformation

There is no standard for how a data set should be structured, and as such data sets are often dissimilar in form. Frameworks and tools however, often needs their data input on a special form and structure, and depending on how the data is represented, and which data is included, the result from the framework/tool will differ. This section will describe why the data has been transformed and how it was done; going more in depth on some central concepts.

4.2.1 Data Representation

Data is usually represented in one of four ways: float, integer, string, boolean. Numbers (float, integer) can be specified even further depending on the size of the number being represented, e.g. we can specify a number as being int16 if it is a smaller number, or int32 if it is slightly larger. A neural network can only work with numbers, which means that any strings, or booleans that exist in our data must be converted to a float or integer representation. For booleans this is easy, we make an encoding where we define *false* as being 0, and *true* as being 1. For strings, we follow the same technique.

To encode a feature consisting of categorical data (strings), we make a mapping which replaces the string value with a number. If we have a feature for recording what phone a customer has which can take on the values: iPhone, Samsung, Nokia; we could map them to the integers 1, 2, and 3 respectively. The model would only see the integers, but as the developers of the model we know the interpretation of each integer. This method is simple, it keeps the meaning of the feature, and it saves us space by reducing a potentially long string down to a small integer. The drawback of this approach is that it introduces a ranking of the values. The model could observe an entry with the value 3, indicating that the customer has a Nokia, but the model would interpret this as being better (or worse, depending on our model) than if the person had a Samsung phone, because 3 is higher than 2. This is a problem; we did not intend to specify that a Nokia is better than a Samsung, only that the customer possesses a Nokia phone. Furthermore, this representation would imply that the average of an iPhone and a Nokia is a Samsung ((1+3)/2 = 2). A way to remedy this situation is to use another encoding scheme called one-hot encoding.

One-hot encoding works on categorical data where we know all possible values. It works by creating a new feature for each possible categorical value, where the new feature is represented as an integer with two possible values: 1 if the value is present, and 0 if the value is absent. Keeping with our example in the previous paragraph; instead of assigning Nokia to the value 3, we would create a new feature, *hasNokia* which we would assign the value 1 if the customer possessed a Nokia phone. Similarly, we would create new features for Samsung and iPhone. This eliminates the problem of ranking in our data, but has the drawback of needing one feature for each possible value. This is fine when the original feature only has a small set of possible values, but it has the potential to spiral out

of control if we need to encode multiple feature that each could take on a large number of values. In a real setting, we could end up with tens or hundreds of new features, where only a small subset of the features have a value of 1; wasting a large amount of space and in turn slowing down our model due to the additional data needing to be processed. Despite these drawbacks, one-hot encoding is still favoured for encoding string based information, and is used in this thesis to encode features with string values.

4.2.2 Missing Values

In larger data sets where manual review of the data is too cumbersome, there is a high chance of missing or corrupted values. This could be due to an error during the recording of the data, or it might be that the value being recorded did not exist for a particular case. Either way, missing values need to be fixed before a data set can be used, as models might stop working when encountering such a value. There are a few different ways to deal with missing data; where imputation and deletion are two common solutions.

Imputation is the act of replacing missing values with substituted values. Which substituted value to use depends a lot on the data set and the type of data being imputed. One solution is to use mean substitution, where the missing value is imputed to be the mean of the existing values. This has the benefit of not changing the mean of the data, but is problematic because it does not capture any correlations between the imputed feature and other features in the data set. Another solution is to use regression. With regression, other features are used to estimate the missing values, keeping correlation intact, but has the disadvantage of possibly overfitting the data to the regression line; removing any variance which would have occurred in natural data.

Deletion of missing data is another solution. The simplest approach is to delete any features including a missing value. This way we won't introduce any complications that arise in other methods, but the drawback is that we are removing a whole feature, which could have predictive power in our model. Another way to delete missing values is to remove the data points including missing values; this way we can still use the feature in our model, but reduce the number of data points. The problem with this approach is that it can easily introduce bias to the data set, as we are reducing the set of data points in a possibly asymmetrical way. In a worst-case scenario, we might end up completely

removing a distinct value from a feature because this value was correlated with missing values in another feature. **For this thesis, due to the large size of the data set, the strategy of removing missing values has been chosen.**

4.2.3 Class Imbalance

Class imbalance refers to when the distribution of a feature is skewed in such a way that one value occurs far more often, or far less than other values. There is no threshold for when a feature suffers from class imbalance, but for a binary feature, a 2:1 distribution would not be particularly unbalanced, while a 10:1 distribution could be considered unbalanced.

When talking about class imbalance, we usually refer to the class imbalance of the dependent variable; as a class imbalance in this variable has a direct consequence on the accuracy of the model. As we usually have numerous independent variables, a class imbalance in one or two of these variables will most likely have little effect on the predictive accuracy of the model. With that being said, some independent variables might be affected by class imbalance, but have strong correlation with the dependent variable, in which case, it would be beneficial to try to remedy the class imbalance problem.

There are multiple ways to attack the class imbalance problem, two popular ones are oversampling and undersampling; a hybrid approach is a third possibility. Oversampling is a method in where we want to increase the underrepresented class, by artificially inflating it. This is usually done by multiplying the existing samples. A more complex oversampling technique is to create new artificial data points of the underrepresented class. Undersampling is the opposite approach; instead of increasing the underrepresented class, we reduce the over-represented class by removing some samples at random, until we get the class balance we want. The hybrid approach combines these two techniques, removing some over-represented samples, while duplicating some underrepresented ones.

Both sampling-methods introduced above has their drawbacks, and the hybrid approach suffers from both drawbacks (although less impacted). By undersampling, we risk removing some of the samples which is more representative, reducing the predictive power of the model. Oversampling runs the risk of overfitting to the samples that gets duplicated, and might be less predictive when seeing new samples that deviates from those learned.

For this thesis, **undersampling has been chosen** as the preferred method. As we are randomly dropping over-represented samples from the data set - until we reach our desired balance - the risk of significantly reducing useful information is acceptably low. Other techniques would have had other limitations and benefits, and without a decisively superior choice, undersampling was chosen.

4.2.4 Normalization

Feature scaling is a normalization technique used to standardize the values of the features in a data set. Some machine learning techniques require the input to be normalized, while most others will generally benefit or at least not be impacted negatively. There is no inherent disadvantage to feature scaling, as even though we lose some information while scaling, we retain the relationship in the data, which is what we want. For example, instead of using a feature with *age* data of [10, 15, 20]; we could instead scale it down to [0, 0.5, 1], retaining the relative distances between the values, but losing the information that the first data point is half the value of the third point.

$$x^{'} = \frac{x - min(x)}{max(x) - min(x)}$$

Features are usually scaled to either [0, 1] or [-1, 1], using min-max scaling, or standardization (Z-score normalization). Min-max scaling is the simplest of the two techniques. The equation above for min-max scaling will rescale a feature to [0, 1].

$$x^{'} = \frac{x - \bar{x}}{\sigma}$$

Another approach is to use standardization with the above formula, where $\bar{x}$ is the mean of x, and σ is the standard deviation. By using standardization, we get a distribution where the mean is zero, and the standard deviation is one. This can have a positive impact when the data set consists of features on different scales.

For this thesis, **the chosen approach is to use min-max scaling** to get the distribution scaled to the interval [0, 1]; as this is often recommended when using neural networks. Standardization using z-score normalization could also be a viable choice, and could be explored with further research.

4.3 Framing the Problem

There are many different ways to define a problem. Some problems are constrained by the type of data we have, while for other problems, it makes no sense to define them in certain ways. Many classification problems consist of single observations which we want to classify. If we for example want to classify a flower, we might look at its color, the length and width of its petals, as well as several other characteristics which defines which type of flower it is. For the aforementioned type of classification problem, it will probably not be needed to look at how the flower changes over time; by adding a time-dimension to the data, as the other characteristics should be enough to differentiate different types of flowers. But it is possible to add a time-dimension, and this holds true for a lot of other problems.

Adding a time-series dimension increases the complexity of the problem, since we add a lot of additional data. When looking at the stock market, we could look at how different stocks relate to each other at a distinct point in time, but we are almost always more interested in looking at how a stock has performed over time. This thesis looks at how we can identify customers who has defaulted, and this problem too can be modeled discretely, by looking at a snapshot in time of each customer. But by adding a time-dimension (since we have this data), we might be able gain significantly more insight by looking at the behaviour of a customer over time.

A **time series** is a series of data points ordered on time, usually with an equal distance of time between each data point. Data points that are closer together is often regarded as being more related to each other than those further apart. Weather data can be modeled as a time series, and if we want to predict the temperature at 14:00, looking at the temperature at 13:00 would usually be regarded as a better predictor than looking at the temperature at 06:00; as the 13:00 data point is closer to 14:00. Apart from weather data; signal processing, financial data, seismological data, and even sequential data such as sentences and words, can be modelled as a time series. If we model a sentence as a time series, we can use machine learning to predict the most likely next word; a feature which is built into many smart-phones today.

This thesis frames the problem as a time-series problem, in the hope of gaining insight by investigating how a customers behaviour over time impacts their default rate.

4.4 Implementation

This section will deal with how the solution has been implemented. As explained in previous sections, the solution is modeled as a time series, implemented with the Tensorflow framework (with Keras) and is written in Python. This section will explain how the solution has been implemented along with the decisions that has been taken while implementing it.

4.4.1 Modelling the Time Series

For a standard backpropagation neural network, the input usually consists of a two-dimensional data set (similarly for many other statistical and machine learning techniques), which when viewed in for example Excel, has a column for each feature, and a row for each data point. The neural network is then fed the rows one at a time (or in batches), and uses the information it gets to train the model.

A time series can be modelled the same way as explained above, but the data will have to be ordered by time. So when the neural network is fed the data, it knows that successive data points is further away in time (if time is ordered by ascending). But this only works if we have a single sequence (one customer, one stock, etc...). If we have more sequences (as the data for this thesis has), we need to model and feed the neural network a 3-dimensional data structure. For our network, the data structure is on the pattern *[batch_size, timesteps, input_dim]*. The first dimension *batch_size*, refers to the number of sequences we have in the data. The second dimension *timesteps*, is as the name implies the number of time steps we have for each sequence, where each time step is one data point (row of data). The last dimension *input_dim* is the number of features we have in each data point. Together, these three dimensions make up the structure of the data we want when modelling a time series with multiple sequences.

The problem we are trying to solve is to identify 'bad' customers. We have previously defined a 'bad' customer as a customer who has received a default notice for three consecutive months; which is captured in the feature 'MonthsInDCA'. This feature has a value for each data point in the data set, but we want to look at a customers whole history and then predict if they are 'good' or 'bad'.

Figure 4.1 shows two different alternatives to how we can define the dependent vari-

able. Option **A** is how the raw data looks like, but we want to transform it to look like option **B**. The reasoning behind this is that if we train our neural network on the data structure that looks like **A**, we in fact learn our model how a 'good' or 'bad' month looks like for the customers (each data point in our data is one month). This could be valuable information, but it is not what we are after. Instead we want our model to look at the whole period (several months) and then determine how 'good' or 'bad' the customer is. This is achieved by having one dependent variable for each sequence in our data.

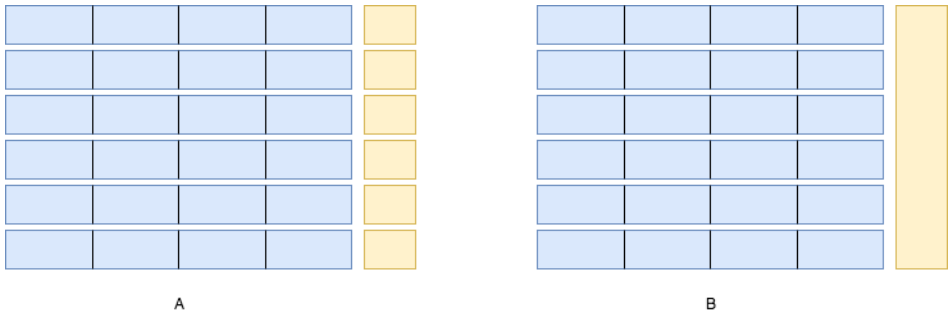


Figure 4.1: Figure showing six rows of data points (blue), with different alternatives for the dependent variable (yellow). **A**: One dependent variable for each data point. **B**: A single dependent variable for all data points.

There are many ways to order both the input and output of our neural network. Option **B** in 4.1 is what we call a *many-to-one* sequence (many inputs, one output); **which is the type of sequence-ordering used for the model in this thesis**. The many inputs refers to how we input multiple data points (one for each time step) into our model, while the one output is the dependent variable explaining if a customer is 'good' or 'bad'.

The number of dependent variables we need is equal to the number of sequences we have. So we store the dependent variables in an array, where the first element in the array holds the dependent variable for our first sequence, and so on. When validating our predictions, the value we get from the model is checked against the dependent variable in our array to check if we an accurate prediction.

4.4.2 Neural Network Architecture

Before we go into the details of how the neural network for this thesis is built, we will explain how a neural network functions. This section will go through the different compo-

nents of a neural network, and then put them together to explain how a simple feed-forward neural network works.

Feed-Forward Neural Network Architecture

A Feed-forward neural network (FNN) is built up of an input layer, an arbitrary number of hidden layers, and an output layer. The input layer holds the data points used in training the network. The hidden layers consists of neurons that take in data from previous neurons, and which is sent through an activation function; which transforms the inputs to a single new output value. Finally, an output layer presents our predictions. Figure 4.2 shows the flow of data through a FNN.

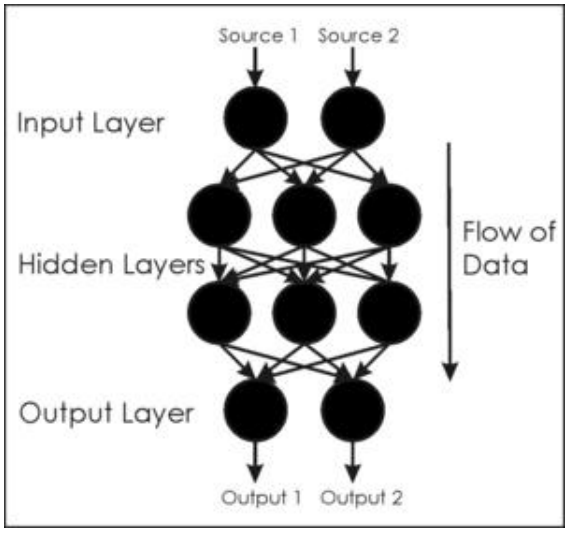


Figure 4.2: The flow of data through a FNN. Downloaded from `https://github.com/cazala/synaptic/wiki/Neural-Networks-101`

When we input a data point in the network, it propagates forward through the network until it reaches the output node which outputs the networks predicted value of the dependent variable. This predicted value is then compared to the actual value of the dependent variable for the data point with a loss function. The error value is then back-propagated backwards through the network in order to change the weights of each the neurons. This is usually done by using an optimization algorithm called stochastic gradient descent (SGD). SGD calculates the derivative of the error function in respect to the neurons weights so the

error decreases. The process we have just described is what is called the backpropagation algorithm, and is the most commonly used technique in training neural networks.

Stochastic Gradient Descent

Stochastic gradient descent (SGD) is an optimization algorithm for finding the minimum of a function, and is used in neural networks in order to optimize the weights of the neurons. The algorithm works on the premise that the fastest way to find the minimum of a function is to move in the negative direction of its gradient. Figure 4.3 shows the gradient descent algorithm at work. The algorithm attempts to find the global minimum through a ”zig-zagging” motion, and as it nears the minimum, each step it takes gets smaller and smaller. This reduction in step-length as it nears a minimum is one of the limitations of the algorithm, and a reason for why it gets increasingly harder to increase the accuracy of a neural network as it gets better at predictions.

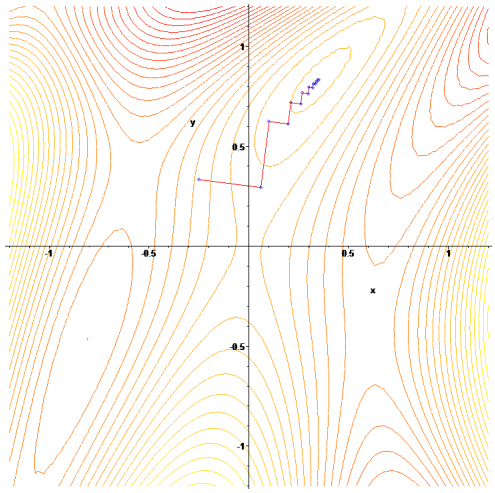


Figure 4.3: An example run of the gradient descent algorithm. Downloaded from `https://en.wikipedia.org/wiki/Gradient_descent`

The Loss Function

The loss function (or error function) is a measure of the discrepancy between the predicted value and the actual value of the dependent variable. The objective of the neural network is to reduce the value of the loss function through manipulating the weights. The lower

the loss gets, the better the neural network gets at accurately predicting the dependent variable. In this thesis, we have used **binary cross entropy** as the loss function; which is a loss function that measures the discrepancy of a binary variable. The formula for binary cross entropy is shown below, where $\hat{y}$ is the predicted value, and y is the actual value.

$$\mathcal{L} = -\frac{1}{n}\sum_{i=1}^{n}\left[y^{(i)}\log(\hat{y}^{(i)}) + (1-y^{(i)})\log(1-\hat{y}^{(i)})\right]$$

The Neuron

To properly understand how a neural network functions, we must understand its building block, the neuron. A neuron can be modelled as seen in figure 4.4. The neuron takes in a set of inputs, either from an input layer (consisting of input neurons), or from other neurons. Each input X_i is a single value, if the input comes from an input layer, each input *X* corresponds to the value of a feature in the data set. If the input is another regular neuron, the input will be the value that the previous neuron calculated.

Accompanying each input, there is a weight w_{ij}, where *i* is the input neuron, and *j* is the current neuron. The weight is used to transform the input-value into a weighted value which is used to minimize the loss function.

The weights are aggregated in a transfer function, which outputs a net input to the activation function. The activation function then transforms the input and outputs the result, which is sent on to another neuron to do the same process again, or to an output neuron which outputs the result to the user. The goal of the neurons - in binary classification - is to output an aggregated value that is close to either 0 or 1 (closest to the desired target).

The Activation Function

The activation function is the core of the neuron. It takes in an arbitrary number of scalars and outputs a different scalar, whose value depends on the function. What function we use can have a major impact on the accuracy of the neural network model. There is no formula to calculate the optimal activation function for our data set, which means that we need to find which function works best through trial and error.

The easiest activation function is the *step function* as seen in figure 4.5. It outputs 1 if the input is positive, and 0 if the input is negative. Another name for this type of activation

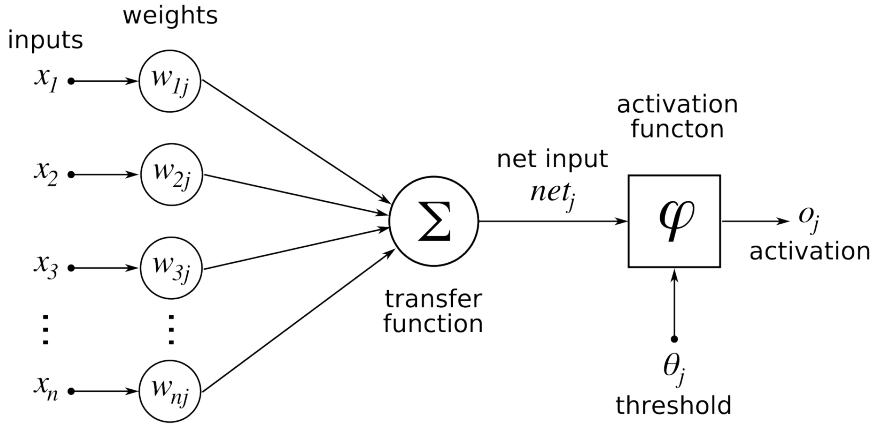


Figure 4.4: The inner workings of a neuron. Downloaded from `https://en.wikibooks.org/wiki/Artificial_Neural_Networks/Activation_Functions`

function is a threshold function. When the scalar is at a certain value, the neuron 'flips' and sends out a signal. This is a very simple function that works in binary classification problems, where we can only have two possible categories for the dependent variable. But for more complex problems with many possible categories, thresholds are much too simple. Even for binary classification, thresholds are too simple for most practical uses, as the granularity of only getting 1 or 0 is usually too low in order to properly model the problem space.

Another option is to use a *linear function*, this allows us to have multiple different classes for our dependent variable. But there are numerous problems with using a linear function. The most limiting problem is that a linear function will not introduce non-linearity into our neural network. Non-linearity is what makes neural networks great, by allowing them to approximate almost any function of interest given enough neurons Hornik (1991).

If we want to have a neural network with multiple layers, there is no point in using a linear activation function. This is due to how linearity works, no matter how many layers of linear functions we have, the output layer would still be a linear representation of the first layer. Which means that we might as well reduce down to only one layer of hidden neurons in order to reduce complexity. This is in contrast to non-linear functions, which when combined, can create interesting and fitting representations of our problem space.

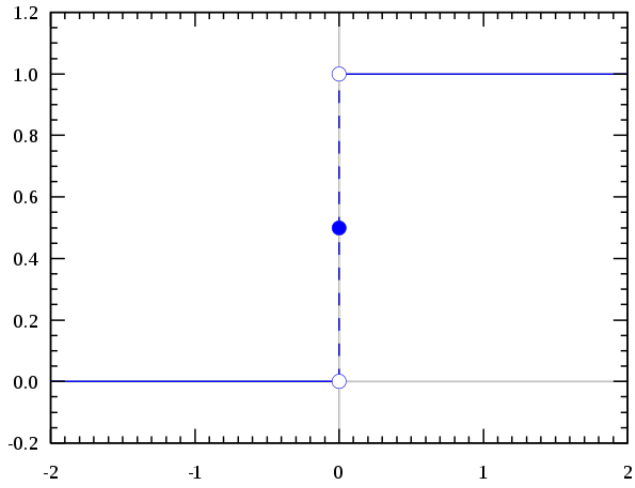


Figure 4.5: A step function which outputs 1 if the input is positive, and 0 if the input is negative. Downloaded from `https://en.wikipedia.org/wiki/Step_function`

Instead of a step function, or a linear function, we could use a *sigmoid* function, such as the non-linear logistic sigmoid function in figure 4.6. Non-linear functions such as the sigmoid have many advantages compared to a linear function; but as with any activation function, the sigmoid function also suffers from limitations.

To start with some of the benefits, the sigmoid is a bound function. This means its output is constrained to an interval; in the sigmoids case [0, 1]. The advantage of a bound function is that it is safe from the exploding gradient problem. In short, the exploding gradient problem is a phenomenon that can occur when the backpropagation algorithm computes the gradient of the loss function. If the activation function outputs increasingly large numbers, the gradient can grow so large that it substantially lowers performance, or in the worst case causes an integer overflow.

Another benefit is that due to the steep slope of the function between x = -2 and x = 2, the sigmoid tends to force output values to the extremes of the [0, 1] range; which makes for clearer predictions.

A limitation is that the sigmoid function suffers from the vanishing gradient problem. This is due to the near horizontal parts on either side of the function. The gradient on these horizontal parts can become so small that each update of the weights almost makes no difference to the accuracy of the model, requiring the neural network to go through

numerous iterations in order to improve itself.

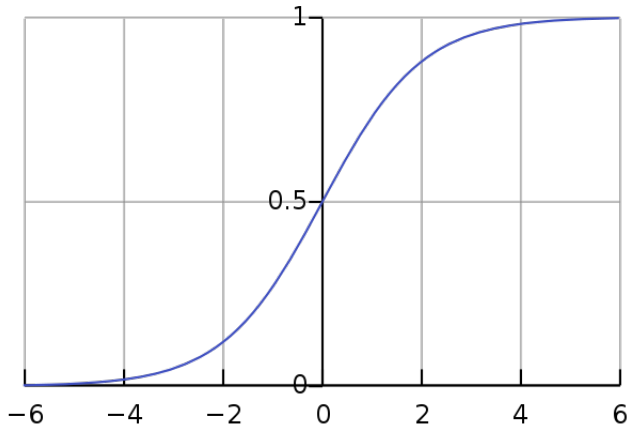


Figure 4.6: Logistic sigmoid function. Downloaded from `https://en.wikipedia.org/wiki/Logistic_function`

There are multiple activation functions, each with their strengths and weaknesses. Which activation function will work best depends on both the structure of our neural network, as well as the data set. As such, as explained in the opening paragraph; it is through trial and error in which we might find an activation function which satisfies our goals.

4.4.3 Recurrent Neural Networks

The type of neural network we are using in this thesis is called a *recurrent neural network* (RNN). A major benefit of RNNs are that they are much less constrained than traditional feed-forward neural networks (FNN). A FNN takes in a fixed-size vector as input, goes through a fixed number of computational steps, then return a fixed-size vector as output. A RNN on the other hand allows us to work on sequences of vectors. This opens up for more configurations in how we process our data as seen in figure 4.7. Furthermore, FNNs assume that all inputs and outputs are independent of each other, and for many problems this is fine, but when looking at temporal data which have an inherent relationship between successive data points, we need to use techniques such as RNNs if we want to capture this relationship.

What makes RNNs great is the extra feedback-loop in the neuron as seen in figure 4.8 which allows it to save its own state at time t and use this value at time $t+n$. It is this

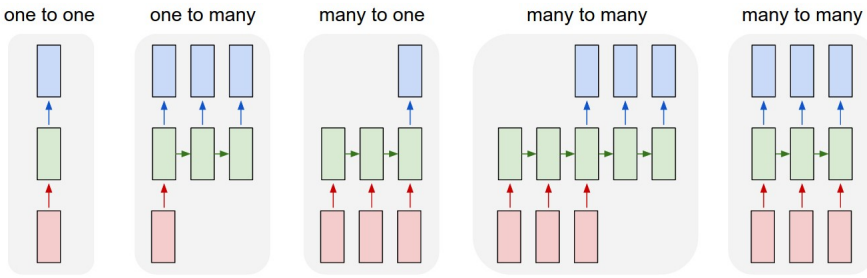


Figure 4.7: Each rectangle is a vector and arrows represent functions (e.g. matrix multiply). Input vectors are in red, output vectors are in blue and green vectors hold the RNN's state (more on this soon). From left to right: **(1)** Vanilla mode of processing without RNN, from fixed-sized input to fixed-sized output (e.g. image classification). **(2)** Sequence output (e.g. image captioning takes an image and outputs a sentence of words). **(3)** Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment). **(4)** Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French). **(5)** Synced sequence input and output (e.g. video classification where we wish to label each frame of the video). Notice that in every case are no pre-specified constraints on the lengths sequences because the recurrent transformation (green) is fixed and can be applied as many times as we like. (Figure and description downloaded from `http://karpathy.github.io/2015/05/21/rnn-effectiveness/`)

property that allows the RNN to capture the relationship between successive data points.

Long Short-Term Memory

The long short-term memory (LSTM) neuron is a type of recurrent neuron that has proven to improve RNNs capabilities. LSTMs limit the effect of the vanishing and exploding gradient problems, and are better at learning context-sensitive relationships in data. This is done by introducing a forget-gate inside the LSTM-neuron. This gate controls the information that the neuron remembers, allowing it to focus on the important parts, while discarding information that is of little use. Due to this interesting property, and the limited availability of research into the application of LSTM-neurons on credit scoring; **LSTM-neurons were chosen to be used in this thesis**.

4.4.4 The Network

The neural network implemented in this thesis is a recurrent neural network consisting of LSTM-neurons. It is constructed of three layers: the input layer, one hidden layer, and one

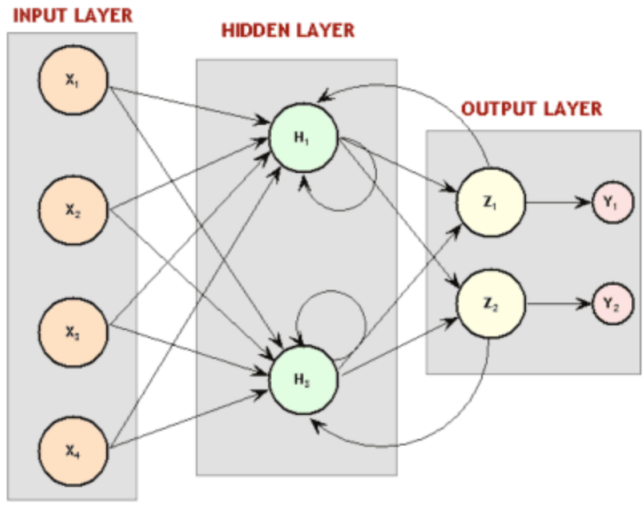


Figure 4.8: Recurrent Neural Network Architecture. Downloaded from `http://www.mattmoocar.me/blog/RNNCountryLyrics/`

output layer. The size of the input layer is proportional to the number of features in the data set (105).

The hidden layer consists of LSTM-neurons that take their inputs from the input-layer. The number of hidden-layer neurons for the first experiment is set to be equal to the number of features. Deciding on the number of neurons is mostly trial and error, but as a starting point; keeping it equal to the number of features gives us a baseline to work with. In the second experiment, we both increase and decrease the number of neurons in the hidden-layer in order to look at how this affects the models performance.

In between the hidden-layer and the output-layer, we have placed a **dropout**-layer. A dropout-layer sets a proportion of the input-values it receives to zero. The reason for doing this is to reduce the chance of the model overfitting to the training data. By setting some random values to zero, the model will always receive an incomplete picture of the data. This forces the model to continuously adapt, as the neurons not affected by the dropout has to step in and readjust their weights in order to improve the predictive capabilities of the model.

What fraction of neurons we need to drop at each iteration to give us the best performance needs to be determined trough testing, but a small dropout-rate will most likely have little effect, while a high rate could result in underfitting, i.e. the model does not have

enough data to accurately make a prediction. Through testing various rates, **a dropout-rate of 0.2 was chosen** (dropping 20% of neurons) for the network in this thesis, as this seemed to give a reasonable effect on the performance compared to other rates or no dropout.

Finally, we have the output-layer which consists of a single LSTM-neuron. This layer takes in all the values from the hidden layer and distills them down to a single prediction with a value between 0 and 1, where the threshold for being categorized as a default (value of 1) is 0.5.

Additional Properties

In addition to deciding how the neural network layers should be built up; we have to decide on the values of a number of properties which specifies how the network should behave. Some of these properties, such as the choice of activation function, optimizer, and number of neurons are adjusted in the different experiments in order to understand their effects on the performance of the model.

Other properties have been purposefully set to certain values through initial testing of the models. The number of properties we have to decide on would result in a possible combination of properties that this thesis considers too high to thoroughly test and document. Therefore, **only a subset of the properties have been chosen to undergo more thorough examination** as written about in chapter 5. The factor for choosing the properties used in the experiments was due to their respective impact on the performance of the model in the initial testing.

The remaining properties were as earlier stated set through initial testing, in conjunction with industry best practices, and recommendations from scientific literature.

Chapter 5

Experiments

This chapter will explain the experiments conducted. It will start with going through the test set up and the evaluation criteria. Afterwards it will present the results, before going into a discussion of the results.

5.1 Experiment set-up

The experiments have been conducted using the model implemented with Keras and Tensorflow as presented in section 4.4.4.

After the data clean-up - eliminating accounts with insufficient data (minimum 12 months of data after account creation) - we are left with 111,416 accounts; where 3.7% (4105) of the accounts are classified as 'bad' accounts. These accounts include dependent variable values exceeding the three months which we earlier defined as representing a 'bad' account. When only looking at the three months, we are left with only 2627 'bad' accounts. To compensate for this we have done an undersampling of the 'good' accounts as explained in section 4.2.3.

5.1.1 First Experiment

The first experiment includes all the cleaned data, and has been set up to investigate the effect the **activation function** and the **optimizer** have on the performance of the model.

The other properties of the neural network have been kept constant for all the models in the experiment.

The two activation functions that are tested are the **relu** and **tanh** functions. These are both non-linear in nature, and is widely regarded as being good options when modelling a neural network. For the optimizer, **rmsprop**, **adam**, and **adagrad** have been chosen.

Each activation function and optimizer has been tested in conjunction with each other, making up six models for this first experiments.

5.1.2 Second Experiment

In the second experiment, the best optimizer and the best activation function from the first experiment have been used. This experiment has been designed to investigate the efficacy of the model in correctly flagging bad accounts that have been in default for at most three months. For this experiment, we have a total of 69,001 accounts, where the 'bad' accounts amount to 3.67% of the total accounts.

For this experiment, we have varied the number of neurons used. The first two models have 10 and 50 neurons respectively, which is both lower than the number of features in the data set. The third model has 105 neurons, which is equal to the number of features in the data set. For the last model, we have doubled the amount of neurons to 210, in order to look at if an increase in neurons, more than the number of features, has any impact on the performance of the model.

5.1.3 Third Experiment

Finally, for the last experiment, we have taken the same combination of properties from the second experiment while testing how the model performs when we vary our definition of the dependent variable. We have earlier defined the dependent variable as being any account which has been in default for three consecutive months. In this experiment we vary this definition to look at being in default for one, two, three, and four consecutive months.

As the number of 'bad' accounts is varied in the four different models; undersampling has been done on the 'good' data in order to bring the percentage of 'bad' accounts as close to 3.67% as possible when testing the models. This was done to ensure that all the

models in the experiment were trained as identical as possible in all aspects except the varied definition of a 'bad' account.

5.2 Evaluation criteria

This section will explain how we evaluate performance of the neural network. There are many ways to evaluate the performance of a model, where the most simplistic is tallying up how many correct classifications the model makes. Even if this is a correct way to evaluate a model, it can hide how effective a model actually is; especially when dealing with class imbalance. A model which scores a 97% accuracy might seem good, but if we revealed that the number of 'bad' accounts makes up 3% of the data set, and that the model failed to classify any of these correctly; the 97% score is less impressive.

5.2.1 Confusion Matrix

A confusion matrix is a table layout for visualizing the performance of a statistical or machine learning method. It records the number of correct and incorrect classifications which the model has made. The layout for the confusion matrix used in this thesis is shown below.

Since this thesis deals with a binary classification problem, our dependent variable can take on two values, 1 and 0, where 1 is defined as the account being in default. True positives (**TP**) are correct classifications of accounts in default (dependent variable is 1), while true negatives (**TN**) are correct classifications of accounts that are not in default. False positives (**FP**) are what is known as *type 1 errors*. It is also known as a false hit, as we claim something which is in fact false. A false negative (**FN**) is what is known as a *type 2 error*, also called a miss. It is the opposite of a FP; we miss a prediction which should be classified as a default.

Confusion matrices are good at giving us better insight into a models accuracy; especially for problems with a class imbalance. Depending on our objective, we might want to limit the number of type 1 or type 2 errors. If it is acceptable to miss some predictions, but we want to limit the miss-classification of 'good' customers as 'bad', we would seek to reduce the number of type 1 errors on the expense of possibly increasing type 2 errors.

This also holds for the inverse scenario.

		Predicted class: 0	Predicted class: 1
Actual Class	0	True Negative	False Positive
	1	False Negative	True Positive

5.2.2 Accuracy

Accuracy is the measure of how many correct predictions we make. A higher score means we made more correct predictions, which means the model performed objectively well given its task. But as explained above, it can on class imbalanced data sets give us a wrong view of a models performance. It has been included in this evaluation as it still gives some insight into the models performance. The formula for accuracy is:

$$Accuracy = \frac{TP + FP}{TP + FP + FP + FN}$$

5.2.3 Precision & Recall

Precision and recall are two performance metrics which can be used in evaluating binary classification. They are both a measure of *relevance*.

Precision is the fraction of selected items which are relevant, or in other terms: how many of our default predictions are correct. This metric reveal the quality of our model in not mistakenly labelling 'good' accounts as 'bad' accounts. The formula for precision is given below.

$$Precision = \frac{TP}{TP + FP}$$

Recall on the other hand is the fraction of relevant items which are selected. This is a measure of quantity, i.e. how many 'bad' accounts are we able to identify. The formula for recall is given below.

$$Recall = \frac{TP}{TP + FN}$$

5.2.4 F-measure

The F-measure (F_1) is a measure of a models accuracy which takes both precision and recall into account. There are different variations of the measure, but for this thesis the traditional F-measure which is the harmonic mean of precision and recall has been used. There is a conundrum in evaluating whether precision or recall is the most important measure of a model. In some cases a higher precision at the cost of low recall might be favourable, and in other cases the opposite might hold true. But as the decision of which of these measures is the most important is highly subjective, **this thesis will use the F_1 score in order to rank the efficacy of the models**. The formula for F-measure is shown below.

$$F_1 = 2 \times \frac{precision \times recall}{precision + recall}$$

5.3 The Results

This section will present the results of the three experiments. The full tables of recorded information for each of the models can be found in appendix A.

5.3.1 First Experiment

For the first experiment, the dependent variable (MonthInDCA) was kept constant to include all possible values. The properties that were changed between the different models in the experiment were the **activation function**, and the **optimizer**. The results for the first experiment are found in table 5.1, and the confusion matrices for each of the models are shown in figure 5.1.

Table 5.1: First Experiment

Model ID #	Activation Function	Optimizer	Accuracy	Precision	Recall	F_1
1-1	relu	rmsprop	0.986	0.845	0.786	0.815
1-2	tanh	rmsprop	0.985	0.958	0.633	0.762
1-3	relu	adam	0.984	0.782	0.822	0.802
1-4	tanh	adam	0.985	0.875	0.705	0.780
1-5	relu	adagrad	0.985	0.784	0.823	0.805
1-6	tanh	adagrad	0.981	0.889	0.589	0.709

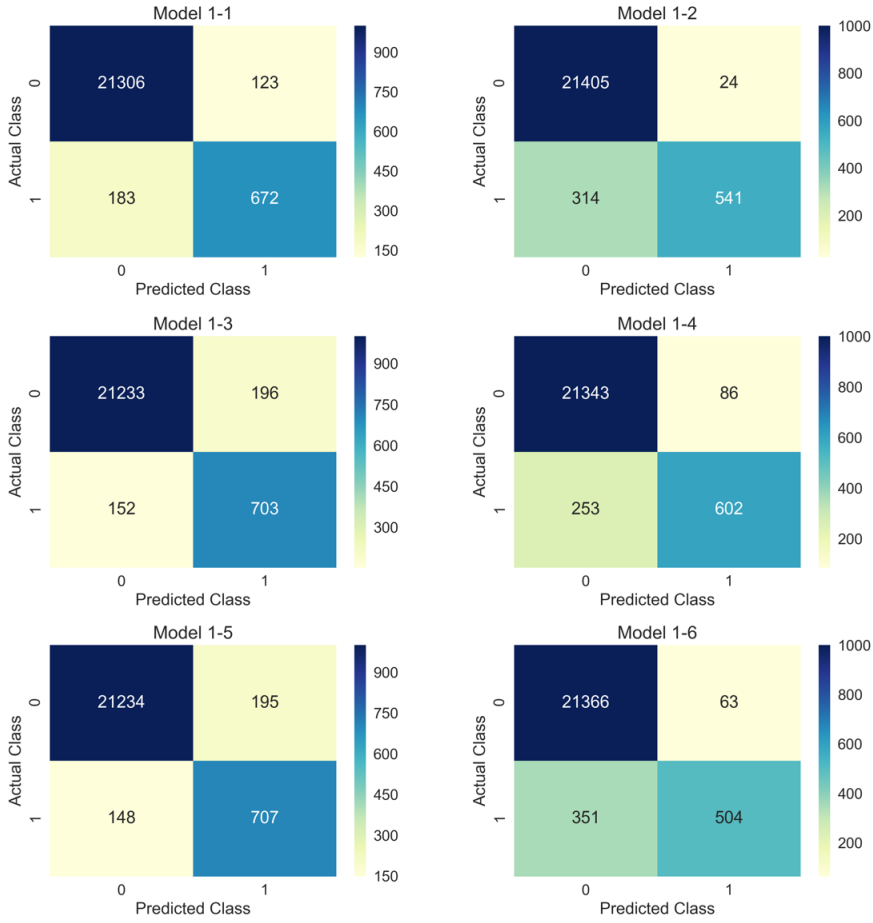


Figure 5.1: Confusion matrices of the models from the first experiment.

5.3.2 Second Experiment

In the second experiment, the dependent variable was set to three months, i.e. 'bad' accounts were defined as accounts which have been in default for up to three months. The adjusted variable in this second experiment was the number of neurons in the hidden layer. The model was tested with 10, 50, 105, and 210 neurons. All the models used the relu activation function, and the rmsprop optimizer. The results for the second experiment can be found in table 5.2, and the confusion matrices for each of the models are shown in figure 5.2.

Table 5.2: Second Experiment

Model ID #	Neurons	Accuracy	Precision	Recall	F_1
2-1	10	0.980	0.757	0.678	0.715
2-2	50	0.984	0.867	0.670	0.756
2-3	105	0.977	0.641	0.899	0.749
2-4	210	0.986	0.796	0.838	0.817

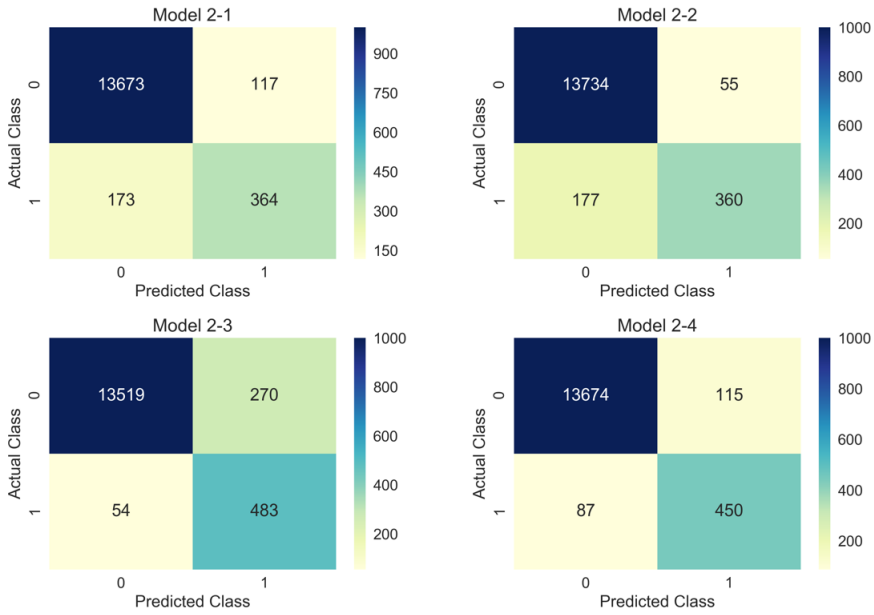


Figure 5.2: Confusion matrices of the models from the second experiment.

5.3.3 Third Experiment

The third and last experiment kept the settings from model #2-4 in the second experiment, while varying the definition of the dependent variable. The experiment was run four times, with the dependent variable taking on the values: 1, 2, 3, and 4. The results for the third experiment can be found in table 5.3, and the confusion matrices for each of the models are shown in figure 5.3.

Table 5.3: Third Experiment

Model ID #	MonthInDCA	Accuracy	Precision	Recall	F_1
3-1	4	0.986	0.777	0.887	0.829
3-2	3	0.986	0.796	0.838	0.817
3-3	2	0.985	0.900	0.682	0.776
3-4	1	0.980	0.759	0.676	0.715

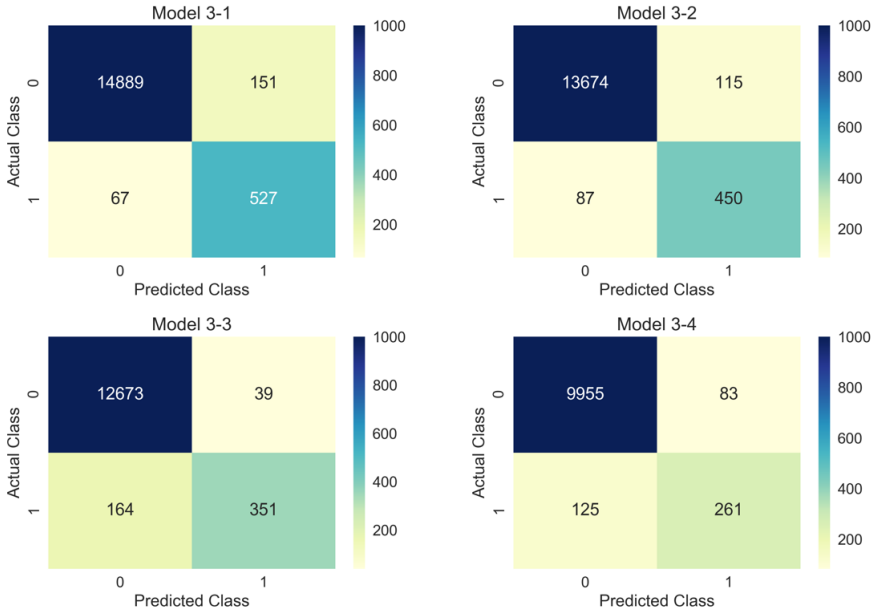


Figure 5.3: Confusion matrices of the models from the third experiment.

5.4 Discussion of the results

This section will go through a discussion of the results. There will be an analysis of the scores from the experiments, as well as an exploration into what the results might reveal of the models.

5.4.1 First Experiment

The first experiment which looked at the activation function and the optimizer used in the neural network reveals that the neural network can correctly classify both the 'good' and the 'bad' accounts to a certain degree. The best score was achieved with the *relu* activation

function, and the *rmsprop* optimizer, and achieved a F_1 score of 0.815. The lowest score, was achieved by the *tanh* and *adagrad* combination, which netted a F_1 score of 0.709.

If we look at model #1-1 in figure 5.1, we observe that it managed to correctly classify 672 samples as 'bad', while 183 'bad' accounts were misclassified as 'good' (false negatives). We furthermore observe that 123 'good' accounts were misclassified as 'bad' (false positives). If we compare this to the relu-adagrad combination in model #1-5, we can observe the opposite of the results in model #1-1. Here we have a higher count of false positives than false negatives, instead of the other way around. A higher count of false positives is also the case for model #1-3 (relu-adam). If we compare the FP counts to the FN counts of the models which used the tanh-function; we observe that all the models had a higher FN count than FP count.

A higher FP count than FN count shows a trend throughout the three experiments to generally yield a better F_1 score (when the F_1 score is over 0.8, all but one models have a higher FP-rate than FN-rate). The reason for this might be that the models with a high FP count is more amenable to classifying samples as 'bad'. On the other hand, models with a high FN score, is more careful in classifying samples as 'bad'.

Furthermore, another observation in this first experiment is that the relu-function beats the tanh-function in all of the models. We also observe that the tanh-function struggles with a recall-rate that is much lower than its precision-rate, therefore lowering its overall F_1 score. This shows us how important the choice of activation-function can be to the results. The tanh-function has however a much better precision score than the relu-function. So if the objective was to minimize the number of false positives, tanh would be the best choice, even if it scores lower at F_1.

Using the right activation-function for the data set has a noticeable impact on the efficacy of the model. The optimizer however, appears to have little to no effect in our experiment. For this thesis we have used the standard settings for the optimizers as defined in the Tensorflow and Keras frameworks. Using other settings for the optimizers than the default settings might give other results than those netted in this thesis.

5.4.2 Second Experiment

The second experiment looked at how varying the neurons impacted the results of the model. A total of four models were trained; varying the number of neurons between 10, 50, 105, and 210 neurons for the different models. Model #2-1 with only 10 neurons gave a F_1 score of 0.715, performing worst of the four. This may indicate that we had too few neurons in order for the model to properly learn the relationships between the 105 features used.

Model #2-2 and #2-3 both scored relatively close, with #2-2 despite having less than half the neurons of #2-3, actually achieving the higher score. The scores however are so close that the variation can most likely be attributed to chance. Since we randomly drop some of the neurons when training the model (due to the dropout layer), there is a certain degree of randomness to how well the models learn. Since the two models score close, there seems to be no advantage to the score in having 105 neurons over 50 neurons, or the other way around. Decreasing the number of neurons used can however be advantageous if the models are equally effective, as a lower neuron count decreases the time it takes to train the neural network.

A notable achievement in this experiment was that model #2-3 achieved the best recall rate of 0.90, when looking at all of the models. This means that only 10% of the 'bad' accounts were not identified. This accuracy however came at the cost of a low precision score of 0.64, which was the lowest of all the models. This suggests that the model over-fitted to the data, learning well to identify 'bad' customers, at the expense of allowing mis-classification of 'good' customers.

The final model (#2-4) used 210 neurons, and performed the best in this experiment, achieving a F_1 score of 0.817. Further increases in the number of neurons could possibly achieve better scores, but would likely offer diminishing returns. The downside of adding more neurons is the increased training time of the neural network.

5.4.3 Third Experiment

The third and last experiment was designed to investigate how varying the definition of the dependent variable impacted the results. By varying the definition of the dependent variable (MonthInDCA), we vary how 'bad' the customers we look at are. When only

looking at customers which have been in default for one month, we presume that they are better customers than those who are in default for longer periods of time. A longer period of being in default means a lower ability or willingness to paying back the owed money.

When we look at the results in table 5.3, we observe a correlation between the number of months the customer has been in default (MonthInDCA), and the F_1 score for the model. There is a notable difference between model #3-1, and #3-4. We can also observe that as we increase the number of months included, there seems to be a diminishing return to the results. Going from one to two months makes more difference on the score than going from three to four months.

It makes sense that it should be easier for the model to identify customers who stay longer in default, as they more likely have different behaviour than those who stay in default for one month, or those who never go to default. An interesting observation is that model #3-4 managed to score better than model #1-6, even though the model in #1-6 was trained on better data. The configuration of the neural networks parameters seems to have a significant impact on a models efficacy.

Chapter 6

Summary and Recommendations for Further Work

This final section includes a summary of the thesis, and concluding remarks on the results from section 5.3. Finally, there is a discussion on possible further research.

6.1 Summary & Conclusion

The motivation behind this thesis was to explore the efficacy of neural networks on detecting signs of default in credit card customers. This was done by developing a recurrent neural network, and training it on a data set given by Sparebank1. The recurrent neural network in this thesis was built up of LSTM-neurons (Long Short-Term Memory); a special type of neurons for use on sequential data. This particular type of neuron was used as it is supposed to be effective on temporal data by remembering important events, and forgetting less important events.

The problem was framed as a classification problem, where we looked to classify each customer as 'good' or 'bad' based on their account-activity during their first 12 months as a customer. A 'bad' customer was defined as someone being in default for three consecutive months during this 12-month period. For the third experiment of this thesis, the definition of a 'bad' customer was varied from one month and up to four months, in order to explore

how the neural networks results changed when varying the data.

The results obtained from the three experiments showed that the constructed model was able to achieve a F_1 score of 0.829 with the configuration of model #3-1. With this model, 527 out of the 594 'bad' customers were correctly identified, while 151 customers were misclassified as 'bad'. Although far from a perfect score, through further experimentation of different neural network parameters, and more thorough examination and transformation of the data set; a higher score should be presumed achievable.

The first experiment showed clear favour for relu as the best activation function when used in conjunction with this data set. The tanh function was however far more superior in its precision score, at the expense of a lower recall-rate, and in turn a lower F_1 score. The choice of optimizer seemed to have little to no effect on the final scores. Further validation should be done in order to determine if the choice of optimizer has a significant effect on a models results.

The second experiment looked at how the neurons affected the models efficacy. The findings here, were that a model with a low number of neurons (10) - when compared to features (105) in the data set - produced noticeable worse results than a model with a high number of neurons (210). The two models with 50 and 105 neurons performed similarly, and scored between the two outliers.

In the third and last experiment we saw a correlation between how long the customer had been in default and the F_1 score of the model. This is as expected, as a worse customer would be expected to deviate more from the norm, and therefore be easier for the model to identify. Maybe surprisingly is that when looking at customers who had only been in default for one month; the model was still able to achieve a recall rate of 0.676, meaning that 261 out of 386 customers were correctly classified as 'bad'.

Neural networks using LSTM-neurons have proved themselves as being able to correctly classify credit card default. Better understanding of the data set and the other parameters could have resulted in better results.

6.2 Recommendations for Further Work

For further work, looking at different pre-processing techniques used on the data set and how this affects the results could be interesting. A model is only as good as its underly-

ing data, so identifying what properties of a data set containing credit card data is most important to a models efficacy; could in turn yield more accurate models.

More work should be done on other parameters of neural networks and their effects. This thesis has showed that recurrent neural networks can be used in classifying defaults, but further research should be done to verify and test parameters not focused on in this thesis.

Finally, it would be interesting to look at how a recurrent neural network performs when the problem is defined as a forecasting problem; where instead of looking backwards in time and classifying accounts, it would try to look forward in time and predict if the accounts would default some time in the future.

Bibliography

Abdou, H., Pointon, J., and El-Masry, A. (2008). Neural nets versus conventional techniques in credit scoring in Egyptian banking. *Expert Systems with Applications*, 35(3):1275–1292.

Abdou, H. A. and Pointon, J. (2011). Credit scoring, statistical techniques and evaluation criteria : A review of the literature. *Intelligent Systems in Accounting, Finance and Management*, 18(2-3):59–88.

Angelini, E., di Tollo, G., and Roli, A. (2008). A neural network approach for credit risk evaluation. *The Quarterly Review of Economics and Finance*, 48(4):733–755.

Bequé, A. and Lessmann, S. (2017). Extreme Learning Machines for Credit Scoring: An Empirical Evaluation. *Expert Systems with Applications*, 86:42–53.

Constangioara, A. (2011). Consumer Credit Scoring. *Romanian Journal of Economic Forecasting*, 14(3):162–177.

Gelfand, S. J. and Lockhart, R. (2015). Understanding the Impact of Heteroscedasticity on the Predictive Ability of Modern Regression Methods by.

Hand, D. J. and Henley, W. E. (1997). Statistical Classification Methods in Consumer Credit Scoring: A Review. *Journal of the Royal Statistical Society. Series A (Statistics in Society)*, 160(3):523–541.

Henley, W. E. and Hand, D. J. (1996). A k-nearest-neighbour classifier for assessing consumer credit risk. *Journal of the Royal Statistical Society. Series D (The Statistician)*, 45(1):77–95.

Hornik, K. (1991). Approximation Capabilities of Muitilayer Feedforward Networks. 4(1989):251–257.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359 – 366.

Karim, M. Z. A., Chan, S.-G., and Hassan, S. (2010). Bank efficiency and non-performing loans: Evidence from malaysia and singapore. 2010:118–132.

Li, K., Niskanen, J., Kolehmainen, M., and Niskanen, M. (2016). Financial innovation: Credit default hybrid model for SME lending. *Expert Systems with Applications*, 61:343–355.

Mester, L. J. (1997). What Is the Point of Credit Scoring? *Business Review (Federal Reserve Bank of Philadelphia)*, (February 1997).

NoregsBank (2017). Noregs Bank Memo, Kunderetta betalingsformidling 2016. (2).

Rosenberg, E. and Gleit, A. (1994). Quantitative Methods in Credit Management: A Survey. *Operations Research*, 42(4):589–613.

Sustersic, M., Mramor, D., and Zupan, J. (2009). Consumer credit scoring models with limited data. *Expert Systems with Applications*, 36(3):4736–4744.

Thomas, L. C. (2000). A survey of credit and behavioural scoring: forecasting financial risk of lending to consumers. *International Journal of Forecasting*, 16(2):149–172.

West, D. (2000). Neural network credit scoring models. *Computers and Operations Research*, 27(11-12):1131–1152.

Yeh, I. C. and Lien, C. h. (2009). The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. *Expert Systems with Applications*, 36(2 PART 1):2473–2480.

Yosinski, J., Clune, J., Nguyen, A. M., Fuchs, T. J., and Lipson, H. (2015). Understanding neural networks through deep visualization. *CoRR*, abs/1506.06579.

Zhou, L., Lai, K. K., and Yu, L. (2010). Least squares support vector machines ensemble models for credit scoring. *Expert Systems with Applications*, 37(1):127–133.

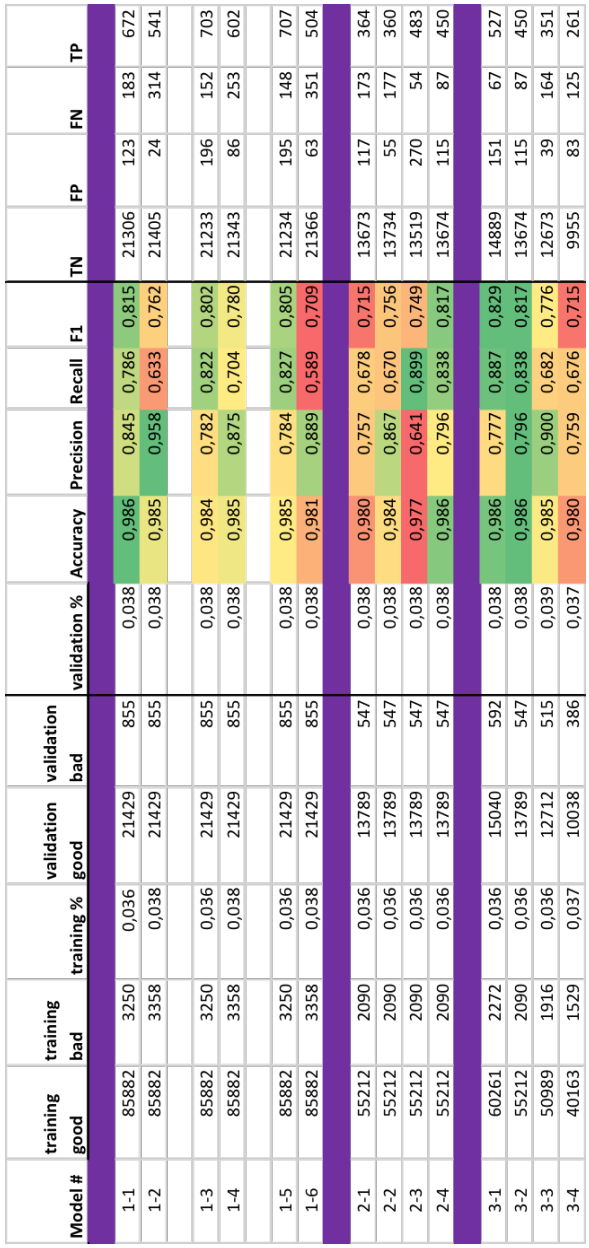
Appendix A

Parameters used in Experiments

This appendix contains the different parameters used in the neural network models for each experiment. Figure A.1 and figure A.2 can be found on the next two pages.

Model #	MonthInDCA (Dependent Variable)	Activation Function	Loss	Optimizer	Dropout	LSTM Neurons	Batch size	epochs	validation split	total accounts	good accounts	bad accounts	% bad
1-1	any	relu	binary_crossentropy	rmsprop	0,2	105	1024	10	0,2	111416	107311	4105	0,037
1-2	any	tanh	binary_crossentropy	rmsprop	0,2	105	1024	10	0,2	111416	107311	4105	0,037
1-3	any	relu	binary_crossentropy	adam	0,2	105	1024	10	0,2	111416	107311	4105	0,037
1-4	any	tanh	binary_crossentropy	adam	0,2	105	1024	10	0,2	111416	107311	4105	0,037
1-5	any	relu	binary_crossentropy	adagrad	0,2	105	1024	10	0,2	111416	107311	4105	0,037
1-6	any	tanh	binary_crossentropy	adagrad	0,2	105	1024	10	0,2	111416	107311	4105	0,037
2-1	3	relu	binary_crossentropy	rmsprop	0,2	10	256	10	0,2	71628	69001	2627	0,037
2-2	3	relu	binary_crossentropy	rmsprop	0,2	50	256	10	0,2	71628	69001	2627	0,037
2-3	3	relu	binary_crossentropy	rmsprop	0,2	105	256	10	0,2	71628	69001	2627	0,037
2-4	3	relu	binary_crossentropy	rmsprop	0,2	210	256	10	0,2	71628	69001	2627	0,037
3-1	4	relu	binary_crossentropy	rmsprop	0,2	210	256	10	0,2	78167	75301	2866	0,037
3-2	3	relu	binary_crossentropy	rmsprop	0,2	210	256	10	0,2	71628	69001	2627	0,037
3-3	2	relu	binary_crossentropy	rmsprop	0,2	210	256	10	0,2	66132	63701	2431	0,037
3-4	1	relu	binary_crossentropy	rmsprop	0,2	210	256	10	0,2	52116	50201	1915	0,037

Figure A.1: Neural network properties, part one



Model #	training good	training bad	training %	validation good	validation bad	validation %	Accuracy	Precision	Recall	F1	TN	FP	FN	TP
1-1	85882	3250	0,036	21429	855	0,038	0,986	0,845	0,786	0,815	21306	123	183	672
1-2	85882	3358	0,038	21429	855	0,038	0,985	0,958	0,633	0,762	21405	24	314	541
1-3	85882	3250	0,036	21429	855	0,038	0,984	0,782	0,822	0,802	21233	196	152	703
1-4	85882	3358	0,038	21429	855	0,038	0,985	0,875	0,704	0,780	21343	86	253	602
1-5	85882	3250	0,036	21429	855	0,038	0,985	0,784	0,827	0,805	21234	195	148	707
1-6	85882	3358	0,038	21429	855	0,038	0,981	0,889	0,589	0,709	21366	63	351	504
2-1	55212	2090	0,036	13789	547	0,038	0,980	0,757	0,678	0,715	13673	117	173	364
2-2	55212	2090	0,036	13789	547	0,038	0,984	0,867	0,670	0,756	13734	55	177	360
2-3	55212	2090	0,036	13789	547	0,038	0,977	0,641	0,899	0,749	13519	270	54	483
2-4	55212	2090	0,036	13789	547	0,038	0,986	0,796	0,838	0,817	13674	115	87	450
3-1	60261	2272	0,036	15040	592	0,038	0,986	0,777	0,887	0,829	14889	151	67	527
3-2	55212	2090	0,036	13789	547	0,038	0,986	0,796	0,838	0,817	13674	115	87	450
3-3	50989	1916	0,036	12712	515	0,039	0,985	0,900	0,682	0,776	12673	39	164	351
3-4	40163	1529	0,037	10038	386	0,037	0,980	0,759	0,676	0,715	9955	83	125	261

Figure A.2: Neural network properties, part two

Appendix B

List of Features Used

- AvgBalanceAmt
- BALANCE_AMT
- CustomerAge
- HAS_DIRECT_DEBIT_AGREEMENT_IND
- HAS_ESTATEMENT_AGREEMENT_IND
- MaxBalanceAmt
- MinBalanceAmt
- MonthsSinceAccountCreated
- OpenAccountFlag
- OVERDUE_AMT
- OverdueFlag
- OverLimitFlag
- INTEREST_EARNING_LENDING_AMT
- STATEMENT_DUE_DAY_OF_MONTH_NUM

- InterestPostedAmt
- ActiveAccountFlag
- CLOSING_BALANCE_AMT
- OPENING_BALANCE_AMT
- EFFECTIVE_PAYMENTS_AMT
- TOTAL_PAYMENTS_AMT
- MINIMUM_TO_PAY_AMT
- NewAccountFlag
- ClosedDuringPeriodFlag
- OverdraftFlag
- HasSupplementaryCard
- SumOf6ActiveTrxnFlags
- CreditLimitAmt
- CreditLimitIncreaseFlag
- CreditLimitDecreaseFlag
- CreditLimitChangeAmt
- SumOf6ActiveStmtFlags
- SumOf12ActiveStmtFlags
- PaymentsAmt
- PaymentsNum
- TurnoverAmt
- TurnoverNum

- TurnoverDomAmt
- TurnoverDomNum
- TurnoverIntAmt
- TurnoverIntNum
- PurchaseAmt
- PurchaseNum
- PurchaseDomAmt
- PurchaseDomNum
- PurchaseIntAmt
- PurchaseIntNum
- ElectronicPurchaseAmt
- ElectronicPurchaseNum
- ElectronicPurchaseDomAmt
- ElectronicPurchaseDomNum
- ElectronicPurchaseIntAmt
- ElectronicPurchaseIntNum
- CashAmt
- CashNum
- CashDomAmt
- CashDomNum
- CashIntAmt
- CashIntNum

- FundtransferAmt
- FundtransferNum
- TurnoverAtStart
- FundTransferAtStart
- PurchaseAtStart
- CashAtStart
- balance_change
- credit_used
- credit_used_last
- GENDER_NAME_Kvinne
- GENDER_NAME_Mann
- DISTRIBUTOR_NAME_BN Bank
- DISTRIBUTOR_NAME_SpareBank 1 BV
- DISTRIBUTOR_NAME_SpareBank 1 Gudbrandsdal
- DISTRIBUTOR_NAME_SpareBank 1 Hallingdal Valdres
- DISTRIBUTOR_NAME_SpareBank 1 Kredittkort
- DISTRIBUTOR_NAME_SpareBank 1 Lom og Skjåk
- DISTRIBUTOR_NAME_SpareBank 1 Modum
- DISTRIBUTOR_NAME_SpareBank 1 Nord-Norge
- DISTRIBUTOR_NAME_SpareBank 1 Nordvest
- DISTRIBUTOR_NAME_SpareBank 1 Nøtterøy-Tønsberg
- DISTRIBUTOR_NAME_SpareBank 1 Oslo Akershus

- DISTRIBUTOR_NAME_SpareBank 1 Ringerike Hadeland
- DISTRIBUTOR_NAME_SpareBank 1 SMN
- DISTRIBUTOR_NAME_SpareBank 1 SR-Bank
- DISTRIBUTOR_NAME_SpareBank 1 Søre Sunnmøre
- DISTRIBUTOR_NAME_SpareBank 1 Telemark
- DISTRIBUTOR_NAME_SpareBank 1 Østfold Akershus
- DISTRIBUTOR_NAME_Sparebanken Hedmark
- PRODUCT_NAME_BN BANK GOLD MC
- PRODUCT_NAME_INTERNE KONTI
- PRODUCT_NAME_LOfavør MasterCard
- PRODUCT_NAME_SB1 EXTRA MC
- PRODUCT_NAME_SH GOLD MC
- PRODUCT_NAME_SpareBank 1 MasterCard Gold
- PRODUCT_NAME_SpareBank 1 Visa Gold
- PRODUCT_NAME_Sparebank 1 Platinum MC
- SalesProductCategory_Boliglån
- SalesProductCategory_Kampanje
- SalesProductCategory_Ordinær
- SalesProductCategory_Platinum
- SalesProductCategory_Ung/Student
- ApplicationSalesChannel_Autentisert web
- ApplicationSalesChannel_Nettbank

- ApplicationSalesChannel_Open web
- ApplicationSalesChannel_Operatørkanal
- ApplicationSalesChannel_Responsside