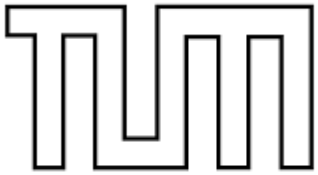
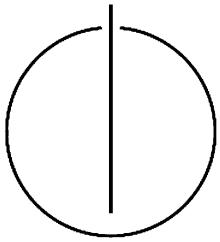


Department of Informatics
Technical University of Munich

Thesis in postgraduate studies of Informatics

Modeling of Mortgage Loan Prepayment Risk with Machine Learning

Shlomo Amar



Modeling of Mortgage Loan Prepayment Risk with Machine Learning

Thesis in postgraduate studies of Informatics

November 2020

Author:
Shlomo Amar

Advisor:
Prof. Dr. Stephan Günnemann

Abstract

A mortgage loan embeds the option to prepay part or the full amount of the loan before its maturity. This option is referred to as mortgage prepayment, and it causes a risk to the bank lending the mortgage loan due to the loss of future interest payments and the creation of complications in the refinancing strategies. Given the typical size of a mortgage portfolio within the balance sheet of a bank, estimating the prepayment rate is therefore vital. There are two kinds of prepayment models available: the optimal prepayment models, which consider prepayment as a consequence of rational behaviour, and exogenous models which also take into account borrower's specifics, loan characteristics and macroeconomic factors. In this thesis, we focus on the second kind of techniques and investigate the applicability of machine learning and specifically of artificial neural network models, which are then applied to predict the prepayment rate on pools of mortgage loans. The estimators expose the highly nonlinear nature of the relationships between input variables and borrowers' prepayment behaviour. To improve model interpretability, we conduct a sensitivity analysis to determine how input variables affect the output of the model.

Contents

1. Introduction

A *mortgage* is the most common way to obtain a large amount of money in a short time to cover a significant expense, for instance, for purchasing a property. A mortgage is a legal agreement between two parties, the lender (or *mortgagee*) which is usually a bank or other financial institution, and the borrower (or *mortgagor*) which is the client, a private person or a company, who asks for the loan. In exchange for lending the money, the mortgagee receives interest payments from the mortgagor. Since the bank can sell the property purchased with the mortgage loan if the client fails to make his payments, the conditions of a mortgage loan are usually better than those of other types of client loans.

Concerning mortgage loans, a predominant part of the risk for the lender lies in the possibility of the client's default, as became unfortunately clear during the 2008 credit crisis. However, there is another aspect that poses a risk. Clients can pay back a part or the full amount of the loan earlier than discussed in the contract. These kinds of unexpected payments are called *prepayments*. Since the lender makes money on loans by receiving interest, prepayments will decrease the profitability of mortgage contracts. Especially when considering that clients are more likely to prepay when current market interest rates are lower than the contractual interest rate on the mortgage agreement. However, a simple loss of future interest payments is not the only risk that the prepayment option brings to the lender; prepayments expose the bank to liquidity risk and interest rate risk: First, a mortgage portfolio has to be refinanced. So, *liquidity risk* happens when the duration of the funding mismatches the duration of the mortgage. An incorrect estimation of prepayments leads to the risk of under or over funding. Additionally, a mismatch in interest rates paid and received leads to *interest rate risk*. It arises because fixed-rate mortgages are usually hedged against interest rates changes using interest rate swaps (IRS), in which the bank receives the floating rate and pays the fixed rate. Thus, a prepayment could make the bank pay a fixed rate on the IRS, which is higher than the fixed-rate received from the new mortgage (since borrowers often use prepayments to refinance new mortgages with lower interest rates). To cover itself against prepayment risk, the lending bank must have the ability to predict the prepayment rate for years ahead with an eye to implementing some effective hedging strategies against this risk.

There are two primary methodologies to predict when borrowers are going to prepay their loans. The first methodology includes *optional theoretical models* and takes into account the financial aspects only. Therefore, borrowers will prepay according to these models only when it is economically convenient for them, and in particular, when the current market interest rate is lower than the one paid on their mortgage. However, people do not always follow a rational justification when they consider to prepay their mortgage. Therefore, models of this kind are not able to fully explain the general prepayment behaviour. The latter methodology consists of *exogenous models* which are able to take into

account both financial and non-financial related information. These models are based on data analysis. To calculate the prepayment risk and to protect against it, one must be able to correctly estimate the prepayment rates for clients or groups of clients that show similar behaviour. A data set with information related to mortgages, to clients and even to macroeconomic factors is needed to initiate such a model.

Many banks estimate the prepayment ratio based on (multinomial) logistic regression models, taking into account many variables that may influence prepayment decisions. We will take the exogenous model approach a step further by using more complex data analysis tools. Using these tools allow us achieving higher performance due to their capability of learning complex nonlinear relationships between the input features and the output target. This thesis aims, therefore, to find an alternative method and to explore the application of artificial neural networks in the estimation of prepayment risk for mortgage loans. Neural networks are effectively used in many fields, including image and speech recognition, fraud detection and many more. Neural networks are sets of algorithms, broadly simulate the human brain, and are designed to recognize patterns. We, therefore, hope that neural networks are better at capturing the irrational behaviour of clients than traditional methods.

The remainder of this thesis is structured as follows. Chapter 2 provides a general background of the mortgage market, including various aspects of mortgage prepayment, which are needed to understand the rest of the work. The motivation for the choice of our approach is also discussed. Chapter 3 deals with the required theoretical background of artificial neural networks and of feedforward networks and their components in particular. Chapter 4 describes how the data are collected and preprocessed. It should be mentioned that the implementation of this work is done in Python using Numpy and Pandas libraries for large-scale data manipulation. Chapter 5 contains the practical implementation of the models and how the final models are selected. For building and training neural networks, we utilize the Keras API on top of TensorFlow library. The second part of this chapter discusses the outputs of the models and presents the performance results of in-time and out-of-time forecasts. In Chapter 6, we conclude the findings of our work and give hints to follow for future research on this topic.

2. Mortgages and Prepayments

2.1 General background

As already introduced, a mortgage is a loan issued by a bank or other financial institution, and the borrower typically uses it to fund the purchase of a real estate. The bank holds this property to secure the payments of the loan, and whenever the borrower fails to pay off the loan, the bank can sell the property in an attempt to reduce its losses and offset the loan. To that end, the size of a mortgage loan can be relatively large, and the interest rate paid is relatively low compared to other types of loans. The maturity of a mortgage is often set to 10 up to 30 years.

The loan size is called the *principal* or the *notional* of the loan. The borrower regularly pays interest until the maturity of the mortgage loan, at which the principal has to be fully repaid. The interest rate on the mortgage loan can be variable or fixed. While a variable interest rate is frequently adapted according to the current market rate, a fixed interest rate will remain unchanged. Most of the borrowers select a fixed interest rate for a period which varies from the first one year up to the entire duration of the loan.

When selecting a mortgage, the client may choose the way to repay the principal of the mortgage loan. The two most common types of mortgages in this regard are annuity and linear mortgage loans. In an *annuity loan*, the borrower pays a fixed amount, occurring every month, until the total principal of the loan has been repaid. The annuity amount is fixed, but its composition changes. In the beginning, most of the amount is made up by the interest and just a small amount is dedicated to the principal repayment. The higher the remaining principal is, the higher is the interest amount. Therefore, as time passes, the interest part gradually becomes smaller, but the principal repayment part increases. In a *linear loan*, the borrower repays a fixed amount of the principal every month, and also interest payments based on the remaining principal are made by the borrowers. It means that a linear mortgage has higher initial principal payments, and therefore it reduces the actual debt faster than an annuity. The less popular repayment schedule is a *bullet loan*. In this case, the client only pays the interest during the running time of the loan and repay the full principal amount at the end of the contract. Borrowers often used bullet loans in the past due to the high tax incentives given by the state, such as savings mortgages in the Netherlands.

As aforementioned, mortgage loan borrowers usually can repay part or the full amount of the remaining principal before the due date, with or without extra costs. From the lending bank perspective, prepayments make the duration of their mortgage portfolio stochastic and create complications in the refinancing strategies.

This research utilizes the single-family loan-level data set provided by Freddie Mac, the Federal Home Loan Mortgage Corporation [10]. Freddie Mac is a public government-

sponsored agency that was established in 1970 to enlarge the secondary mortgage market in the US. Together with the Federal National Mortgage Association (Fannie Mae), Freddie Mac buys mortgages on the secondary market from lenders such as banks. Freddie Mac and Fannie Mae either hold these mortgages or package them into mortgage-backed securities (MBS) which are then sold to investors on the open market. This secondary mortgage market increases the liquidity available for mortgage lending and additional home purchases, and it helps to make credit equally available to all borrowers [11]. The Federal Housing Finance Agency (FHFA) authorized Fannie Mae and Freddie Mac to publish their loan-level performance data, aiming to improve transparency and help investors building accurate credit performance models. The most common mortgage type within the Freddie Mac single-family loan portfolio is the fixed-rate annuity mortgage loan with a maturity of thirty years. Likewise, this category is the most popular mortgage product in the US, and it represents over 75% of all home loans.

2.2 Prepayment models and general drivers

As already pointed out, the models proposed in the literature can broadly be categorized into optimal prepayment models and exogenous prepayment models. The former class contains models that consider prepayments to happen as a consequence of entirely rational behaviour of borrowers. These models address prepayments as an option assumed to be exercised optimally. A prepayment option is an option which "reflects the difference between the value of the outstanding loan repayments at the interest rate at the time of prepayment for the remaining term of the loan, minus the amount of the loan then outstanding" [34]. Rational borrowers should exercise the prepayment option only if the above-stated difference is positive, and else, the option is worthless since it would be more appropriate to put money in a savings account rather than prepay the mortgage.

People, however, do not always act economically rationally. Studies have shown that prepayment models based on pure financial variables would either underestimate or overestimate the prepayments under particular circumstances [5]. In other words, a substantial stochastic component related to behavioural uncertainty can be observed. Studies have reacted by modeling this stochastic component in addition to the 'optimal' element, followed with the development of exogenous models using different explanatory variables to represent prepayments. Exogenous models aim to explain the relationship between observed prepayment rates and a set of explanatory variables through statistical analysis and data mining. The influence on prepayment rates can be estimated on an aggregated level and even on a loan-level if such data are available. Including borrower characteristics in these models can explain behavioural heterogeneity. However, if data on individual borrowers are unavailable, this behavioural diversity can be modelled by classifying individual borrowers into different behavioural types [7]. There are many reasons why borrowers decide to make prepayments on their mortgage loan. In reality, many variables influence borrowers' decision to prepay or not. These factors may be related to

- Borrower characteristics: age, income, creditworthiness, employment status, marital status.
- Loan characteristics: loan amount, loan age, mortgage interest rate, prepayment penalty, geographical location, property type.
- Macroeconomic data: housing prices, market mortgage rates, month of the year.

Interest incentive is one of the most important prepayment determinants. When the interest rate for a new mortgage contract would be lower than the individual contractual one, the borrower has a financial incentive to pay off his mortgage and to get a new one. For this reason, Interest incentive is often called *refinancing incentive*. The higher the remaining principal to be paid is, the stronger is the refinancing incentive.

The *seasoning* effect is another possible driver of prepayment, and it is related to the loan age. Prepayments often exhibit S-shaped relation with loan age: In fact, prepayments rarely take place after loan origination, then they gradually increase until they reach a constant level near the maturity of the loan. This phenomenon represents the probability of selling the house as the loan age increases [21].

Trends in house prices give insight into the activity in the housing and mortgage markets. In durations of house price growth, housing sales and mortgage initiations may increase, because profit will be made when the property is sold. Prepayments due to home relocation are more frequent in these periods. Conversely, home sales tend to decrease during periods of price depreciation due to missing profit and intensified by risk-averse home buyers that would hesitate to enter the market [33].

The month of the year may influence the prepayment rates. This effect is called *seasonality*, and it captures the strong seasonality on the housing market. Significantly more houses are sold during the summer months and early fall. Selling a home (or *housing turnover*) will usually trigger a full prepayment [22]. Besides, partial prepayments in December and January are more likely to happen, due to Christmas bonuses paid in December or due to possible tax benefits of loan prepayments paid before the end of the year.

A prepayment can occur with or without a penalty. To control prepayment behaviour, financial institutions in many countries often set prepayment limits in the mortgage contract. Prepayments that exceed these limits are charged with penalty payments. The presence of a prepayment penalty may influence the prepayment behaviour in such a way that the gain as mentioned earlier from refinancing incentive may not be large enough to cover the penalty applied in case of prepayment. In Germany, it is usually possible to prepay up to 10% of the original loan size every year without any penalty. In contrast, standard residential mortgages in the US offer full prepayment flexibility: The entire principal outstanding can be paid off at any time without a prepayment penalty. However, borrowers can choose a penalty charge in their mortgage to the advantage of reduced interest rates. These contracts provide a benefit to the borrowers for accepting the penalty.

2.3 The proposed approach

Prepayment modeling is essential to balance sheet planning and risk analysis for banks. It is also among the most complex areas of financial modeling, because of the highly nonlinear and interactive nature of the underlying risk factors, as well as due to possible regime changes in credit availability and borrower behaviour. Moreover, prepayments due to housing turnover and prepayments due to refinancing incentives may have different risk factor sensitivities. For instance, refinancing incentive tends to increase with loan size, while housing turnover often decreases with loan size [44]. Besides, there are other reasons for partial prepayments. Therefore, many sub-models would be required, each with its risk factor dependencies, specified and estimated separately. However, our data do not include the reason for prepayments. It makes the separation of those models impossible.

As touched upon earlier, there are two main approaches for prepayment modeling, one that uses optional theoretical models and one that uses exogenous models. The two currently most popular statistical frameworks for exogenous models are survival analysis [24] and logistic regression. In survival analysis, we are interested in finding the time until prepayment events occur, and hence, the probability distribution for the duration of the mortgage. Logistic regression techniques is widely used by banks to predict whether mortgages will prepay or not. In recent years, approaches using machine learning algorithms such as decision trees and artificial neural networks are also considered [36]. Due to the complexity and nonlinearity in borrowers' behaviour, we choose artificial neural networks as a suitable machine learning approach for modeling and predicting prepayments.

Many research studies have formulated the prediction of prepayments as a classification problem. However, even though classification models can predict the probability of prepayment events, they are not able to learn whether a prepayment event happens or not and the prepayment size at the same time. Therefore, studies based on classification approach and construct the prepayment rates by dividing the number of predictions in a particular prepayment class by the total number of data points. Assuming that all prepayment observations are full prepayments, this approach would be sufficient to forecast the prepayment rate, since the rate would correspond to the proportion of 'ones' in the target vector of the predicted prepayment class [32]. However, in reality, most of the prepayment events are partial prepayments, and even though the model would be extended by incorporation partial prepayments into a multi-class problem, a classification model still could not assess the size of partial prepayments. To overcome the latter issue, Saito in [32] applied a correction term to the 'partial prepayments' class by multiplying the class probability with the averaged observed prepayment rate. We decide to use a different kind of approach from those already introduced and to formulate the prepayment model as a regression problem predicting prepayment rates in one go.

We aim to forecast the prepayment rate of a portfolio of mortgages, but the information that we have is related to individual mortgages. It means that we can approach the problem by looking at prepayments on loan-level, then aggregate the results. We can also look at portfolio level seeking for the average prepayment rate of each portfolio instead of the values for each loan. In the context of prepayment risk, banks are usually interested in prepayments on portfolio level rather than on loan-level. The goal is, therefore, to place greater emphasis on the portfolio level. In addition to that, most individual client information is not available in our data set. It is therefore difficult to have an insight into the prepayment behaviour of every single borrower. Our previous attempt on loan-level did not lead to satisfying results. Given the typically large size of mortgage portfolios, an aggregate-level analysis would provide a reasonable approximation to the results of a loan-level study. A common form of aggregation used by banks and other financial institution in the context of prepayment modeling is the conversion of individual loans with similar characteristics into a single averaged loan. We follow this approach which will be described in Section 4.2.

Therefore, the goal of this work is to create a prepayment model based on artificial neural networks solving a nonlinear regression problem and to estimate the prepayment rate for different groups of mortgage loans according to their characteristics.

3. Artificial Neural Networks

This chapter provides an introduction into artificial neural networks and specifically into the concept of feedforward networks. The Universal Approximation Theorem [6, 20] is the theoretical foundation of why we can use artificial neural networks to approximate any continuous function that describes a sophisticated relationship between input and output. In general, there are two distinct parts concerning the information flown through the network: the forward pass and the backward pass. In Section 3.1 we will explain how the input is forward-propagated through the network to produce the output. In Section 3.2 we will introduce several optimization techniques and the backpropagation algorithm to update the weights of the network after each training cycle. We will shortly discuss the choice of different cost and activation functions as well as the importance of weight initialization to increase the performance of the network (Sections 3.3 and 3.4). Few methods to reduce overfitting are explained in Section 3.5. In the remainder of this chapter, we will introduce the group lasso used for feature selection and the partial dependence plot, a tool that aims to improve the interpretability of neural network and other black-box models. In this chapter, we will apply a slightly modified version of the notation used by Bishop [1].

3.1 Introduction

A neural network is used to approximate a generally unknown and sophisticated function $f : \mathbb{R}^{d_0} \to \mathbb{R}^{d_L}, (x_1, ..., x_{d_0}) \mapsto (t_1, ..., t_{d_L})$, where d_0 is the input dimension and d_L the output dimension. In this fashion, the goal is to create a neural network that takes an input vector $\mathbf{x} = (x_1, ..., x_{d_0})$ and generates an output vector $\mathbf{y} = (y_1, ..., y_{d_L})$ which provides a good approximation of the target vector $\mathbf{t} = (t_1, ..., t_{d_L})$. A neural network consists of layers of nodes, also called *neurons*. A typical network consists of L layers with one input layer, several hidden layers and one output layer. Figure 3.1 gives a graphic illustration of a neural network. Every single neuron applies an affine transformation of the input it receives using weights. The *weighted input* is denoted by a. Each weighted input a is then passed through a differentiable, nonlinear *activation function* $h(\cdot)$ to generate the output of this neuron.

An input node i takes input x_i and sends it as output to all nodes in the first hidden layer. Each hidden node in the first layer produces the weighted input. After the activations in the first hidden layer, the process is repeated as the information flows to the next hidden layer; thus, the output of one layer is the input for the next layer. This is why networks of this kind are called *feedforward* neural networks. Generally, the activation of

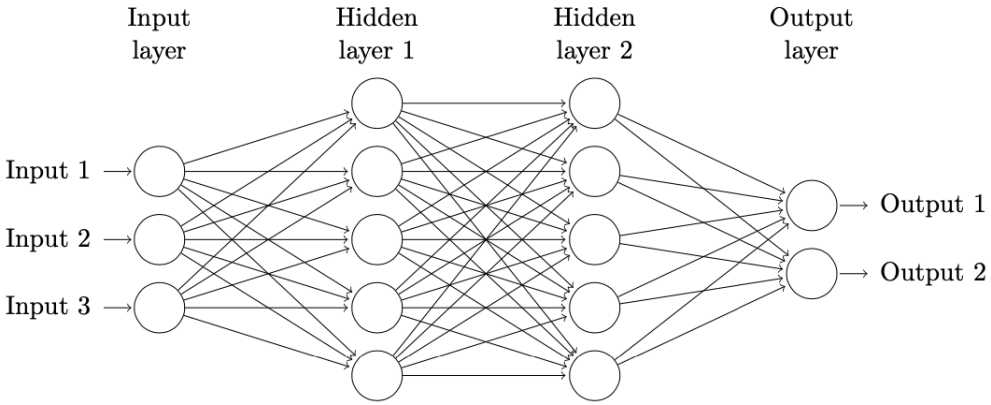


Figure 3.1: Schematic representation of a particular fully connected neural network

neuron j in layer l is specified by

$$z_j^{(l)} = h(a_j^{(l)}) = h\Big(\sum_{i=1}^{d_{l-1}} w_{ji}^{(l)} z_i^{(l-1)} + b_j^{(l)}\Big), \tag{3.1}$$

where constants $w_{ji}^{(l)}$ are the weights of hidden neuron j for input i, and the constant $b_j^{(l)}$ is the bias of hidden neuron j. The activation vector of layer l is equivalently expressed with

$$\mathbf{z}^{(l)} = h(\mathbf{a}^{(l)}) = h\Big(\mathbf{W}^{(l)}\mathbf{z}^{(l-1)} + \mathbf{b}^{(l)}\Big),$$

where the activation is applied element-wise. The output of the network is, therefore, the activation vector of the output layer L, and it is expressed by $\mathbf{y} = h(\mathbf{a}^{(L)})$ with the components $y_j = h(a_j^{(L)})$. It becomes clear that the output $\mathbf{y}$ of the network is a function of the input $\mathbf{x}$ and all weights and biases of the network. After weights and biases are initialized, they are updated to produce a more accurate output vector $\mathbf{y}$ which should be closer to the target vector $\mathbf{t}$. This process is done by selecting a proper cost function to quantify the approximation error, and minimizing the cost function by updating the weights and biases according to a selected optimization method.

The purpose of a neural network is to model a complicated function given a sample of observations, and to apply this function on unseen data points. Training such a network requires a lot of data. The available data is often split into a training, a validation and a test set. The training data is used to optimize the weights and biases, the *learnable parameters* of the network. The validation data is then used to signal overfitting and to choose suitable hyperparameters such as the number of layers. Hyperparameters are model parameters other than weights and biases. Finally, the performance of the network is tested on the test data.

3.2 Training neural networks

This section outlines the steps that are involved during the training of feedforward neural networks. After the input has been forward-propagated through the network to produce the output (Section 3.1), we need to evaluate in each training stage how well does the

output $\mathbf{y}$ approximate the target values $\mathbf{t}$. It is what the cost function of the network does. Commonly used cost functions are introduced in Subsection 3.2.1.

Updating the weights and biases of the network is done according to the selected optimization method. In Subsection 3.2.2 we will highlight several optimization techniques that may find the optimal values of the weights and biases given the input and target values. Training of neural networks is normally done by gradient descent optimization algorithms. Additionally, Adam optimizer is introduced.

In order to perform the update of the weights and biases, the error must be backwards-propagated through all network nodes from the last to the first layer. The backpropagation algorithm is the method for updating neural networks in the backward pass, and it will be introduced in Subsection 3.2.3.

3.2.1 Cost functions

The *cost function* (or *error function*) is a function of the weights and biases given the input of training data set $\mathcal{X} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$ and the corresponding actual targets $\mathcal{T} = \{\mathbf{t}_1, ..., \mathbf{t}_N\}$, where N is the number of training data points. Cost functions often correspond to the sum of the *loss functions* of each training data point,

$$C(\mathbf{W}, \mathbf{b} \mid \mathcal{X}, \mathcal{T}) = \sum_{n=1}^{N} C_n(\mathbf{W}, \mathbf{b}|\ \mathbf{x}_n, \mathbf{t}_n), \tag{3.2}$$

where C_n is the loss of the n'th data point. Selecting the loss function is highly dependent on the learning task we are dealing with. In regression problems, the most common choice is the *mean squared error* (MSE) function,

$$C_n(\mathbf{W}, \mathbf{b} \mid \mathbf{x}_n, \mathbf{t}_n) = \frac{1}{2}\|\mathbf{y}_n - \mathbf{t}_n\|^2, \tag{3.3}$$

where $\|\cdot\|$ denotes the Euclidean norm and $\mathbf{y}_n(\mathbf{W}, \mathbf{b}|\mathbf{x}_n)$ is the predicted target. For classification problems, *cross-entropy loss* (or *log loss*) is the most commonly used loss function, and it is defined as

$$C_n(\mathbf{W}, \mathbf{b} \mid \mathbf{x}_n, \mathbf{t}_n) = -\sum_{i=1}^{d_L} t_i \log(y_i), \tag{3.4}$$

where y_i and t_i are the elements of the output and the target vector, respectively. Simulations on classification problems indicate [35] that using the cross-entropy error outperforms the MSE and even speed-up training and improve generalization to unseen data.

3.2.2 Optimization techniques

Gradient descent

The iterative method *gradient descent* is one of the most well-known optimization algorithms. This iterative method requires a differentiable cost function and a starting point of the algorithm. Each iteration takes a step opposite to the direction of the gradient. In the context of feedforward neural networks, the learnable parameters are set to their initial values in the first iteration, and the input is pushed through the network to generate

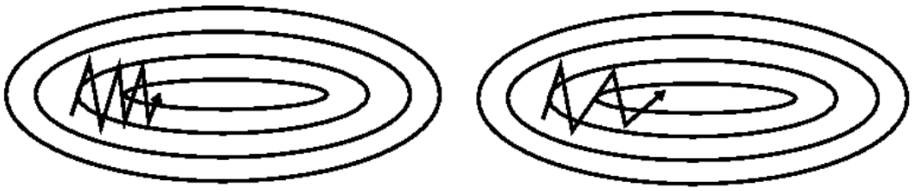
the output, which is then evaluated and compared with the target values by applying the cost function. The minimum of the cost function is obtained by iteratively modifying the learnable parameters in the direction of the steepest descent. It is expressed by

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \eta \nabla_{\boldsymbol{\theta}} C^{(t)},$$

where θ is the set of learnable parameters, t is the iteration counter, η is the step size, called the *learning rate* of the algorithms and $\nabla_{\boldsymbol{\theta}} C$ denotes the gradient of the cost function. With small learning rates, it will take longer to minimize the cost function, and we will have a higher chance of converging to a local minimum. Conversely, high learning rates would lead the algorithm to not converging or even diverging with a higher probability. Choosing a reasonable learning rate is therefore vital, but is not an easy task. Various adaptive learning rate techniques, where the rate is reduced over time, are approaches to mitigate these problems.

In order to find the gradient of the cost function, it is necessary to calculate the sum of gradients over all training data points. Thus, the learnable parameters are only updated once, after evaluating the network on all training data. Operating on large datasets would therefore cause the calculation to be computationally expensive, resulting in very slow learning. We use consequently the *mini-batch* gradient descent technique, which splits the training data into so-called mini-batches of smaller size. The learnable parameters are updated based on the gradient of the mini-batch data points. One full cycle over the dataset through all the mini-batches is known as an *epoch*. In this way, we can achieve several updates with only one epoch. Note that it is necessary to randomly shuffle the training data set after each epoch to prevent from getting stuck in cycles and also to avoid the algorithm from accidentally selecting unrepresentative mini-batches. By doing this, we refer to the approach known as *stochastic gradient descent* (SGD), in which the cost function is evaluated on a randomly selected subset of the training data.

SGD algorithm might perform well, given a suitable mini-batch size and learning rate value. But there are conditions for which the learning progress can be slow. For instance, when the algorithm has to pass through a narrow valley with steep, rocky walls. As illustrated in Figure 3.2a, the algorithm might oscillate up and down the walls instead of moving along the bottom towards the local minimum [31]. A possible solution to this problem is to add a *momentum* term [30]. Instead of only updating the learnable parameters based on the current gradient, a fraction of the update vector from the previous step is added, so that an exponential moving average of the gradients is computed. In this way, the momentum of the movement towards the local minimum is weighted in the optimization, and it helps accelerate the gradients in the right direction, thus, leading to faster convergence (Figure 3.2b).



(a) SGD without momentum (b) SGD with momentum

Figure 3.2: Path of update progress [31]

Another method that tries to dampen the oscillations, but in a different way than momentum is *RMSProp*, which stands for Root Mean Square Propagation [18]. RMSProp is an unpublished adaptive learning rate method that adapts the learning rate for each of the learnable parameters.

Adam optimizer

An optimization method that has become widely used is *Adam* or Adaptive Moment optimization algorithms, introduced by Kingma and Ba in 2016 [23]. Adam can be looked at as a combination of RMSprop and SGD with momentum. It calculates the squared gradients to scale the learning rate similar to the RMSprop, and it takes advantage of the momentum by using an exponential moving average of the gradient instead of the gradient itself. The Adam algorithm calculates in the first step the exponential moving average of the gradient and of the gradient squared,

$$m^{(t+1)} \leftarrow \beta_1 m^{(t)} + (1-\beta_1)\boldsymbol{\nabla}_{\boldsymbol{\theta}} C^{(t)},$$

$$v^{(t+1)} \leftarrow \beta_2 v^{(t)} + (1-\beta_2)\boldsymbol{\nabla}_{\boldsymbol{\theta}} C^{(t)} \odot \boldsymbol{\nabla}_{\boldsymbol{\theta}} C^{(t)},$$

where the hyperparameters β_1 and β_2 are the exponential decay rates of the moving averages, $\odot$ is the element-wise multiplication and the initial values $m^{(0)}$ and $v^{(0)}$ are set to zero. Kingma and Ba [23] realized that m_t and v_t are biased towards zero. Therefore, they proposed bias corrections for $t > 0$,

$$\hat{m}^{(t+1)} = \frac{m^{(t+1)}}{1-\beta_1}, \quad \hat{v}^{(t+1)} = \frac{v^{(t+1)}}{1-\beta_2}.$$

In second step, the learnable parameters are updated according to

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \eta \frac{\hat{m}^{(t+1)}}{\sqrt{\hat{v}^{(t+1)}} + \epsilon}, \tag{3.5}$$

where η is the learning rate, and ϵ is a small scalar used to prevent a division by zero. The designer of Adam suggest the default hyperparameter values $\eta = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. It was proofed in [23] that Adam algorithm converges if the cost function is convex and has bounded gradients. However, cost functions are not convex in most neural network applications. Nevertheless, empirical results show [23] that the Adam algorithm outperforms other optimization methods even for non-convex cost functions. Adam optimizer has therefore become popular by the neural network community, and it is also used in this work.

3.2.3 Backpropagation algorithm

The backpropagation algorithm is the method used in the backward pass, and it allows the error to propagate back through the network, from the last to the first layer. The objective of the algorithm is to compute the partial derivative of the cost function with respect to each of the learnable parameters. Those derivatives will be finally used in the selected optimization algorithm. Recall from equation (3.2) that the cost function is basically the

sum of the loss function over all training data points. Considering N data points, the derivative of the cost function with respect to a weight $w_{ji}^{(l)}$ or a bias $b_j^{(l)}$ is given by

$$\frac{\partial C}{\partial w_{ji}^{(l)}} = \sum_{n=1}^{N} \frac{\partial C_n}{\partial w_{ji}^{(l)}}, \qquad \frac{\partial C}{\partial b_j^{(l)}} = \sum_{n=1}^{N} \frac{\partial C_n}{\partial b_j^{(l)}}. \tag{3.6}$$

We also recall from equation (3.1) the relation of nodes from three consecutive network layers, $z_i^{(l-1)}$, $z_j^{(l)}$, $z_k^{(l+1)}$, with

$$z_j^{(l)} = h(a_j^{(l)}) = h\Big(\sum_i w_{ji}^{(l)} z_i^{(l-1)} + b_j^{(l)}\Big),$$

$$z_k^{(l+1)} = h(a_k^{(l+1)}) = h\Big(\sum_j w_{kj}^{(l+1)} z_j^{(l)} + b_k^{(l+1)}\Big).$$

We will first introduce a useful notation δ [1], that is the error of node j in layer l, and it is defined as the partial derivative of the loss function with respect to the weighted input $a_j^{(l)}$ of that node. For notational simplicity, the subscript n in the δ is dropped. We can express the error in the output layer L as

$$\delta_k^{(L)} = \frac{\partial C_n}{\partial a_k^{(L)}} = \frac{\partial C_n}{\partial y_j} \frac{\partial y_k}{\partial a_k^{(L)}} = \frac{\partial C_n}{\partial y_k} h'(a_k^{(L)}). \tag{3.7}$$

To find the error in the other layers, we apply the multivariate chain rule and we get

$$\delta_j^{(l)} = \frac{\partial C_n}{\partial a_j^{(l)}} = \sum_k \frac{\partial C_n}{\partial a_k^{(l+1)}} \frac{\partial a_k^{(l+1)}}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial a_j^{(l)}} = h'(a_j^{(l)}) \sum_k w_{kj}^{(l+1)} \delta_k^{(l+1)}. \tag{3.8}$$

Knowing the expressions for the errors, we can move backwards by applying the chain rule to finally obtain the derivatives of the loss functions with respect to the weights and the biases,

$$\frac{\partial C_n}{\partial w_{ji}^{(l)}} = \frac{\partial C_n}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial w_{ji}^{(l)}} = \delta_j^{(l)} a_i^{(l-1)}, \qquad \frac{\partial C_n}{\partial b_j^{(l)}} = \frac{\partial C_n}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial b_j^{(l)}} = \delta_j^{(l)}. \tag{3.9}$$

The backpropagation procedure can be summarized as follows: The input data is first forward-propagated through the network by calculating the weighted inputs and the activations of all hidden and output nodes. Given the loss function, the errors of all output nodes are evaluated using equation (3.7). Then, the errors of every hidden node in the network are backpropagated using equation (3.8). Finally, the required derivatives are obtained using equation (3.9) and summed up according to equation (3.6) to find the derivatives of the cost function with respect to every weight and bias in the network. After obtaining these derivatives, we can apply any of the optimization algorithm discussed in Subsection 3.2.2.

As shown in equation (3.8), the error δ is a product of the weights, the derivative of the activation function and the δ's of other layers. It becomes clear that the derivatives of the cost function with respect to the learnable parameters expressed in equation (3.9) is the product of many derivatives of these activation functions. Since the update of weights and biases in each iteration step depends on the size of the gradient, small derivatives

of the activation functions would lead to a small change in the weights and biases, and the network may hardly learn anything in each iteration step. This effect is often called *learning slowdown*, and it is obviously stronger for earlier layer nodes in deeper neural networks since the gradient tends to get smaller as we move backwards through the hidden layers. In other words, nodes in the earlier layers learn much slower than nodes in later layers. This phenomenon is known as the *vanishing gradient* problem, and as a result, learning will slow down through those layers. Learning slowdown effect might be reduced by an intelligent choice of the activation function that will be discussed in Section 3.3. Additionally, weights can reasonably be initialized, so that the network will start learning in regions where the derivatives of the activation functions are large (Section 3.4).

3.3 Activation and output functions

The choice of activation and output functions will be shortly discussed in this section.

Sigmoid

The standard logistic function is historically one of the most commonly used activation functions. In this circumstance, this function is known as the *sigmoid* function. An early formulation of the Universal Approximation Theory [6] was based on the sigmoid function. Sigmoid is a real, differentiable, non-decreasing function, approaching 0 in the negative and 1 in the positive limit. The sigmoid function is denoted by σ, and it is defined by

$$\sigma(a) = \frac{1}{1 + \exp(-a)}.$$

This function, however, has two considerable drawbacks: Sigmoid suffers from the learning slowdown effect as a consequence of the vanishing gradient phenomenon [43]. If the absolute value of the weighted input is big, learning is very difficult because the gradients are nearly zero at the tails. We then consider the nodes to be 'saturated' or 'dead'. The second problem is due to the non-zero-centred characteristic of the sigmoid activation function. Nodes in later layers can receive input values that are either all positive or all negative. This fluctuation can establish 'zig-zag' dynamics, switching from positive to negative values between each layer during optimisation [43].

Hyperbolic tangent

Another popular activation function is the *hyperbolic tangent*, and it is defined as

$$\tanh(a) = \frac{\exp(a) - \exp(-a)}{\exp(a) + \exp(-a)}.$$

The *tanh* activation function is defined on the interval $(-1, 1)$ and it is, in fact, a shifted and scaled version of the sigmoid function, $\tanh(a) = 2\sigma(2a) - 1$. Similar to sigmoid, the *tanh* function also suffers from the vanishing gradient problem, as its gradients tend towards zero at the tails. The benefit of *tanh* over sigmoid is that its activations are cantered around zero, so that negative and positive inputs will be strongly mapped in the negative and positive direction [14].

Rectified Linear Unit

The so-called *rectified linear unit* (ReLU) is a function that aims to solve the vanishing gradient problem. The ReLU was introduced as an activation function by Krizhevsky, Sutskever and Hinton in 2012 [25], and it is defined as

$$f(a) = \max\{0, a\}.$$

The ReLU is currently the most prominent activation function, as it does not suffer directly from the vanishing gradient problem due to the constant gradient for positive input values. Additionally, this simple threshold rule makes the ReLU computationally highly efficient. The downside of the ReLU is that it may lead to many nodes with a zero activation, as a consequence of negative input values. Such a node is considered to be 'dead', once it produces a zero output on most of the training data, and it not able to recover because of the zero gradients of the ReLU.

Leaky ReLU and Exponential Linear Unit

The *leaky ReLU* (LReLU) was presented in 2013 [28] as an attempt to reduce the effect of dying neurons. For $\alpha \leq 1$ the Leaky ReLU is defined as

$$f(a) = \max\{\alpha \cdot a, a\},$$

and it allows a small gradient α (typically set between 0.01 and 0.3) in the domain, where the ReLU outputs zero. Another recent effort to enhance the ReLU is the *exponential linear unit* (ELU) [4]

$$f(a) = \begin{cases} a & \text{if } a > 0 \\ \alpha(\exp(a) - 1) & \text{if } a \leq 0 \end{cases}.$$

The ELU as well allows for negative values of the activation. But in contrast to Leaky ReLU, the ELU saturates for large negative values, which can positively influence noise-robustness [4].

Output functions

The *output function* of a neural network is the activation function of its output layer. The selection of a proper output function is subject to the application the network is designed for. The most common choice for regression problems is the *identity* output function $y_j = a_j^{(L)}$ in combination with the MSE cost function, and for classification problems this is the *softmax* output function in combination with the cross-entropy cost function. The softmax output function of the j'th output neuron is defined as

$$y_j = \frac{\exp(a_j^{(L)})}{\sum_{j=1}^{d_L} \exp(a_j^{(L)})}.$$

Since the softmax of all output nodes sum up to one, the softmax function generates a probability distribution by its design, and y_j can be interpreted as the probability of belonging to the class j. For binary classification problems, using sigmoid as an output function is equivalent of using the softmax function. An in-depth discussion of output functions and their combination with the choice of the cost function can be found in Bishop [1].

3.4 Weight initialization

Since a neural network learns by adapting its weights and biases, the initialization of these parameters is crucial. The initial values can influence both the speed and the optimality of convergence. An inconvenient initialization may contribute to the vanishing or exploding gradient problem, and this would lead to a very slow learning process or no convergence of the algorithm. Clearly, we need the learning signal to flow appropriately through the network in both directions, in the forward pass when making predictions, and in the reverse direction when backpropagating the gradients. In other words, we neither want the learning signals to 'die out' nor to explode and saturate.

By all means, the choice of the activation functions plays an important role in the efficiency of the initialization method. For symmetric activation function such as logistic sigmoid and hyperbolic tangent, Glorot and Bengio [13] introduced the *Xavier initialization* method in 2010. To let the signal flow properly through the network, they specified two conditions. Glorot and Bengio argued that the variance of each layer's outputs needs to be equal to the variance of that layer's inputs. Additionally, the gradients should have equal variance before and after flowing through every layer in the backward pass. These two constraints can only be satisfied simultaneously if the number of input and output connections are equal ($d_{l-1} = d_l$) for every layer. Since, in the general case, d_{l-1} and d_l are not identical, the average of them is suggested. When the weights are drawn from a Gaussian distribution with a zero mean, Glorot and Bengio proposed to set the variance to

$$Var[\mathbf{W}] = \frac{2}{d_{l-1} + d_l}.$$

When sampling from a uniform distribution, this is translated to the sampling the interval $[-r, r]$, where

$$r = \sqrt{\frac{6}{d_{l-1} + d_l}}.$$

The experimental results of Xavier Initialization were shown with the hyperbolic tangent activation function. It turns out that Xavier initialization is not quite optimal for ReLU functions. For non-symmetric functions like the ReLU, He et al. [17] introduced a more robust weight initialization method, which is often referred to *He initialization*. This method applies the same idea of Xavier Initialization but to ReLU functions. Specifically, the Gaussian standard deviation or the interval variable of the uniform distribution is multiplied by $\sqrt{2}$.

3.5 Reduction of overfitting

As illustrated in Figure 3.1, each neuron in every layer is connected to every other neuron in the previous layer. In even a relatively small network, there are a large number of trainable parameters. It leads to the risk of overfitting [3] when the amount of training data is small. Under those circumstances, the network might memorize a huge number of training examples and learn noise in the data instead of just learn the underlying patterns. Such an overfitted network will perform very well on the training data while fitting poorly on unseen data. In other words, the model is not able to generalize well from observed data to unseen data. The easiest way to withstand overfitting when it does occur is to collect

more training data or to cut down the number of trainable parameters, thus, to reduce the number of layers and hidden neurons. However, collecting more data is often not possible, and smaller networks will sometimes fail to learn complex input-output relationships. In such situations, there are many other methods to reduce or to indicate overfitting. This section contains the methods used or considered in this work.

Separation of training, validation and test data

The easiest way to signal overfitting is probably to split the available data set. The best practice is to use around two-thirds of the initial data set for training and to split the remaining data in half; one part is used as the validation data and the other part as test data. The network is then trained on the training data, and it will be run on the validation data after each epoch. Only in the end, after the final network architecture has been selected, the network is run on the test data. Since the network did not train on the validation data, the error on this set would probably be bigger than the error on the training data. If this difference is too large, the model is possibly overfitting the training data. Another signal that overfitting occurs is when the error on the training data decreases but the error on the validation data starts to increase.

Early stopping

The *Early stopping* approach is closely related to the method of splitting the data into training, test and validation sets. Early stopping literally means stop training when there are signals that overfitting might happen. As a consequence, the network will stop training when it has learned the majority of the generic features, but before it starts learning noise or anomalies in the training data. In practice, we set a large number of training epochs, and we stop training when generalization error increases.

Weight regularization

A different approach to withstand overfitting is *weight regularization* or *weight decay*. It was shown [37] that large weights are more frequently found in networks where overfitting appears. Weight regularization method adds an extra penalty term to the cost function, forcing larger weights to have a higher loss. Since the training algorithm aims to minimize the cost function, adding this term will force the weights to be smaller. As a consequence of smaller weights, the network output will change less when the input values change. In this way, fluctuations due to noise in the training data will have a smaller effect on the output, and the risk of overfitting will be reduced. A common regularization term is the squared $L2$ norm of the weights, and it called $L2$ *regularisation*. The extended cost function becomes

$$C = C_0 + \frac{\lambda}{2} \sum_l \left\| \mathbf{W}^{(l)} \right\|_2^2,$$

where C_0 is the original and unregularized cost function from equation (3.2), $\|\cdot\|_2$ is the $L2$ norm, and λ is called the regularization parameter. Weight regularization is often used together with other methods discussed in this section.

Dropout

Dropout is a regularisation method that was first introduced by Hinton et al. [19]. The fundamental idea is to randomly drop neurons and their connections from the network during training. By ignoring a random fraction of neurons, each network update during training is performed with a different view of the network. Dropout prevents overfitting by providing a practical way of combining many different neural network architectures in parallel.

3.6 Group lasso feature selection

Regularization is typically associated with the prevention of overfitting. In Section 3.5, we shortly introduced the weight decay as a regularization method, in which the weights are penalized for their magnitude. The $L2$ regularization is considered as the most commonly used form of weight decay, Another type is the $L1$ or *lasso regularization*. It put a restriction on the sum of the $L1$ norms of the weights to the loss function [16]. In fact, *lasso* stands for Least Absolute Shrinkage and Selection Operator, and it is a type of linear regression analysis method that performs the $L1$ regularization [38]. Lasso regularization leads to sparsity in the solution, forcing weights of less important nodes to have zero values. Concerning the input nodes, this is the same as ranking the input variables. It allows us to use lasso regularization as a feature selection tool, filtering out less important variables in a naive and an easy to implement way.

Feature selection is a critical step for machine learning projects. It can help improve accuracy, stability, and runtime, and avoid overfitting as well. There are many filter, wrapper and embedded feature selection techniques used for machine learning models [26]. We decide to utilize lasso regularization as an embedded feature selection method in this work. When using lasso regularization for feature selection, we apply the weight penalty only to the weights of the first layer $\mathbf{W}^{(1)}$. The cost function becomes

$$C = C_0 + \lambda \left\| \mathbf{W}^{(1)} \right\|_1 = C_0 + \lambda \sum_i \sum_j |w_{ij}^{(1)}|,$$

where C_0 is the unregularized cost function from equation (3.2), $\|\cdot\|_1$ is the $L1$ norm, and λ is the regularization parameter which controls the degree of sparsity.

In a neural network, many weights act on one input variable. To 'zero-out' its value, we have to 'zero-out' all its corresponding weights at the same time. The standard lasso regularization introduced above is not able to achieve that. Moreover, since we represent each categorical input feature as a collection of binary variables, standard lasso regularization can not provide information for the combined importance of a categorical feature. Therefore, we need to ensure that all of the binary variables that correspond to a single categorical feature get 'zeroed-out' or kept together. A useful method to deal with a group of weights is the *group lasso* approach introduced by Yuan and Lin [42]. Group lasso lets us combine weights that are associated with one feature. To achieve this goal, we define groups of weights: All outgoing weights from one input node are put together. Additionally, weights of all input nodes that represent one categorical feature are grouped together as well.

To ensure that the same degree of penalty is applied to large and small groups, the regularization parameter is scaled by the group size. Let $\mathbf{W}_g^{(1)}$ denotes the weight sub-

matrix of a given group g, let p_g be its size, we modify the cost function such that

$$C = C_0 + \lambda \sum_g \sqrt{p_g} \left\| \mathbf{W}_g^{(1)} \right\|_2 . \tag{3.10}$$

Confusingly, the group lasso proposed by Yuan and Lin uses the $L2$ norm like in the $L2$ regularization, but standard lasso uses the $L1$ norm. Nevertheless, group lasso does not do the same as $L2$ regularization does. Finally, we can calculate the feature magnitudes $\left\| \mathbf{W}_g^{(1)} \right\|_1$ to rank input features according to their importance in descending order.

3.7 Partial dependence plot

Problematically, black-box models such as neural networks offer limited interpretability. There are a variety of algorithm-specific techniques that aim to improve the interpretability of these models. A convenient tool is Friedman's *partial dependence plot* (PDP) [12]. It plots the change in the average predicted value as the specified features vary over their marginal distribution. The partial dependence function is formally defined as [15]

$$f_S = \mathbb{E}_{\mathbf{x}_S}[f(\mathbf{x}_S, \mathbf{x}_C)] = \int f(\mathbf{x}_S, \mathbf{x}_C) d\mathsf{P}\mathbf{x}_C,$$

where S is the subset of model features for which the partial dependence function should be plotted, C is the complement set of S, and $\mathbf{x}_S$ and $\mathbf{x}_C$ are the corresponding feature values. The function f_S gives the expected value of the neural network model f when $\mathbf{x}_S$ is fixed, and $\mathbf{x}_C$ varies over its marginal distribution $d\mathsf{P}\mathbf{x}_C$. In reality, the true model f and the marginal distribution $d\mathsf{P}\mathbf{x}_C$ are unknown. Therefore, f_S is estimated by the average observations with

$$\hat{f}_S = \frac{1}{N} \sum_{i=1}^{N} \hat{f}(\mathbf{x}_S, \mathbf{x}_{C_i}),$$

where $\mathbf{x}_{C_i}$ are the observed values of the complement feature set S in the training set. In this way, we can give an insight into how neural networks act and the relationship between the feature set S and the predicted outcome.

4. Data collection and engineering

4.1 Loan level data

4.1.1 Raw data and data cleaning

In this chapter, we will introduce the data used in this work and discuss the methods by which we process and engineer new features. The final dataset used in the models is constructed from multiple sources. The Freddie Mac mortgage loans data is the primary dataset, supplemented by macroeconomic data from Freddie Mac and US Bureau of Labor Statistics.

The Single-Family Loan-Level data set provided by Freddie Mac [10] contains mortgage loans that originated in the US from January 2000 to December 2018. The data set mainly consists of fixed-rate annuity mortgage loans with a maturity of thirty years. The data itself is divided into origination records and performance updates over the lifetime of the mortgages. Origination records accommodate static information taken at the time of loan origination, including many individual loan variables such as original balance, original interest rate and loan-to-value ratio, as well as some limited information on borrower characteristics such as FICO score and debt-to-income ratio. Monthly performance is reported for each mortgage loan, including dynamic data like current unpaid principal balance and an indicator whether the mortgage has paid off or fully prepaid. The data set contains 950,000 mortgage loans in total. Each loan is tracked monthly from origination date until mortgage termination or maturity with a cut-off date of June 2019. Followed over time, this leads to approximately 63 million monthly performance records. Both origination and performance record are joint to monthly observation data set through loan identifier. For this work, each monthly observation of an individual loan constitutes a sample point.

We complement the loan-level data with local and national macroeconomic factors which may influence loan prepayment decision. Housing price index values are obtained from Freddie Mac [8], and unemployment rates are sourced from the US Bureau of Labor Statistics [39]. Both house price index and unemployment rate are monthly data observation, available per US state. Monthly data of market mortgage rate (US average of 30 year fixed rates for conforming home purchase mortgages) are also collected from Freddie Mac [9]. Note that we removed data points related to mortgage properties which are located in one of offshore US territories, as macroeconomic data are not available for these states.

A small fraction of loans in the dataset have missing data as a result of reporting errors or incomplete information provided by the borrower. Missing key features is most probably a result of reporting errors. Therefore, we expect that any data point should have at least original balance, original interest rate, FICO score and loan-to-value ratio. Samples missing one of these feature values are taken off from the data set. Missing data

for other features is more likely to be a result of the inability of borrowers to provide the data at mortgage origination, for instance, the debt-to-income ratio, which is usually not available for subprime borrowers [36]. Luckily, the proportions of missing data are below 3%. Therefore, we could afford to impute the missing data without any loss of valuable information. Data imputation is a simple and popular approach, and it involves using statistical methods to estimate a feature value from those values that are present. We obtain the median values for numerical features and the mode values for categorical variables, and then, we replace all missing values with the calculated statistics.

To generate the target variable (Subsection 4.1.3), we calculate the actual monthly payments from the different of current unpaid principal balance from two consecutive months. Therefore, we insist not to have gaps in the monthly performance data, and we remove mortgages for which we encountered missing performance updates. After cleaning the data as described above, roughly 47 million monthly data points are remaining.

4.1.2 Input variables

There are numerous input variables, which are related to clients' specifics, mortgage characteristics or macroeconomics factors. We first list the variables for mortgage i and time t that are in the loan data sets, including

- loan_size_t: Original principal balance of the mortgage.
- current_UPB_{it}: Current unpaid principal balance. It is often called current outstanding balance.
- $\text{interest\_rate}_{it}$: Current client's interest rate.
- loan_age_{it}: Number of months passed since mortgage origination.
- FICO_i: Credit score, between 301 and 850. It indicates the creditworthiness of the borrower. In general, it gives the likelihood an individual will timely repay his future obligations.
- LTV_i: Original loan-to-value ratio defined by the original principal divided by the purchase price of the mortgaged property.
- DTI_i: Original debt-to-income ratio defined as the sum of the borrower's monthly debt payments divided by his gross monthly income.
- loan_purpose_i: Indicates if the mortgage is a Refinanced mortgage (R) or a Purchase mortgage (P).
- occupancy_owner_i: Indicates whether the mortgage is Owner-occupied (O), Investment property (I) or a Second home (S).

Note that we use a one-hot encoding format using binary vectors to represent categorical data. To complement the variables from loan data sets, we generate several features that are found to have the most predictive power in the literature and as discussed in Section 2.2. We first provide an overview followed by the specification of the generated features. We construct

- $\text{interest\_incentive}_{it}$: The difference between the current mortgage market rate and the existing client's interest rate.
- $\text{rolling\_incentive}_{it}$: 24-months moving average of interest incentive.

- SATO_i: Spread-at-origination is the difference between the mortgage market rate and the original client's interest rate.
- $\text{unemployment\_rate}_{it}$: Seasonally adjusted unemployment level per US state.
- HPI_change_{it}: Appreciation of home-price-index level per US state between mortgage origination and reporting date.
- month_t: Two variables representing the *sin* and *cos* transformation of the calendar months.
- region_i: Indicates the US regions South, West, Mid-West, North-East, in which the property securing the mortgage is located.
- pre_crisis_t: Indicates whether the data observation falls in the period either after or before the credit crisis of 2008.

The refinancing or interest incentive is one of the most important determinants for prepayment. Refinancing is expected if the individual mortgage rate currently paid is lower than the rate available in the market. Therefore, we define the interest incentive as the difference between the current mortgage market rate and the existing client's interest rate,

$$interest\_incentive_{it} = interest\_rate_{it} - market\_rate_t.$$

If such an interest incentive exists for a longer duration, people are more likely to prepay their loans. Therefore, we generate also the 24-months moving average of interest incentive,

$$rolling\_incentive_{it} = \frac{1}{24} \sum_{s=t}^{t-23} interest\_incentive_{is}.$$

Spread-at-origination (SATO) is an additional indication that represents the credit quality of the borrower after the FICO score. High SATO value is equivalent to lower credit quality. Therefore, higher SATO loans tend to prepay slower due to latent credit issues that may lead to complications during the refinancing of the loans [41]. SATO is defined as the difference between the mortgage market rate and the original client's interest rate, both at mortgage origination date,

$$SATO_i = interest\_rate_{i,\,t_0} - market\_rate_{t_0}.$$

State-level macroeconomic risk drivers are linked to the mortgage data set through the US state of the mortgaged property. We add the seasonally adjusted unemployment rate and obtain the change of the house-price-index (HPI) from mortgage origination to the current reporting date. The HPI change is constructed as

$$HPI\_change_t = \frac{HPI_t}{HPI_{t_0}} - 1.$$

We also consider the seasonality effects, and more precisely, the calendar months. Traditionally, we would convert any categorical feature into binary vectors representation. In this case, the calendar months would be projected into a 12-dimensional space. Due to the cyclical nature of this variable, we reduce the dimensionality of the feature space by transforming a calendar month i into a *sine-cos* representation,

$$\sin(i) = 2\pi \cdot \frac{i}{12}, \quad \cos(i) = 2\pi \cdot \frac{i}{12},$$

where $i = 1, ...12$, January is assigned to 1, February to 2 and so on. The *sine-cos* transformation makes a smooth transition and lets the model know that months at the end of a year are similar to months at the beginning of the next year [27]. The geographic effect on prepayment intensity could be significant. Therefore, we group the US states into four regions West, Mid-West, North-East and South, according to the classification of US Census Bureau [40].

The last variable that we are taking into account is a binary variable splitting the data into two buckets, pre-crisis and post-crisis of 2008. We will shortly explain the reason for this variable by describing the regime change concerning the mortgage market: The period between 2002 and 2006 is known as the housing boom years in the US. The housing market was overflowed with effortless loans, which were also easy to refinance. During the refinance wave in 2003, the prepayment rate reached its highest level (and it will be illustrated in Figure 4.1). Later, in 2007–2008, the Great Recession changed the scene of the mortgage business. Since 2008, a remarkable number of homeowners were not able to repay their mortgage payments. For reducing the risk of default, the regulatory authority established a substantial amount of standards, reconsidering all aspects of the mortgage industry. After the 2008 financial crisis, loan refinancing became heavily suppressed, especially for borrowers with weak creditworthiness. This binary variable would, therefore, let the model incorporate the credit crisis of 2008.

4.1.3 Prepay amounts

The target variable used in this work is a numerical variable indicating the monthly prepayment rate. The prepayment rate expresses the monthly prepay amount as a fraction of the current outstanding balance. Since the target model is to estimate the pool-level prepayment rate, we will first present how the loan-level prepay amount is obtained. Later in Section 4.2, we will discuss how the final pool-level prepayment rate is generated from the loan-level prepay amount. The monthly prepaid amount PRP_t is defined as the sum of all actual principal payments that exceeding the expected principal cash flow under the contractual payment scheme,

$$PRP_t = PP_t^{(act)} - PP_t^{(exp)}. \tag{4.1}$$

The actual principal payment $PP^{(act)}$ can be identified from the data set by means of the current unpaid principal balance P_t for each observation. P_t reflects the mortgage ending balance as reported by the servicer for the corresponding monthly period. The monthly actual principal payment is basically the value difference from two consecutive months

$$PP_t^{(act)} = P_{t-1} - P_t.$$

As introduced in Section 2, the monthly annuity payment AP is composed of interest payment IP and principal payment PP. Therefore, the monthly expected contractual principal repayment for an annuity mortgage loan is implicitly derived by subtracting the monthly interest payment from the annuity payment. The annuity payment AP is constant over the lifetime of the mortgage and is calculated as

$$AP = \frac{r \cdot P_0}{1 - (1+r)^{-n}},$$

where P_0 denotes the original principal balance, r is the monthly mortgage rate, and n is the original term of the loan in months. The monthly interest payment is based on the current outstanding balance at the beginning of the month P_{t-1} and it is simply expressed by $IP_t = r \cdot P_{t-1}$. The monthly expected principal payment is therefore given by

$$PP_t^{(exp)} = \frac{r \cdot P_0}{1 - (1+r)^{-n}} - r \cdot P_{t-1}.$$

Note that there are many mortgages found in the data set, for which new terms (such as new values for r and n) were renegotiated during contracts' lifetime. Therefore, the monthly expected contractual cash flows are updated accordingly. At last, and as expressed in equation (4.1), we obtain the prepay amount for each month, expressed as the difference between the actual and contractual principal payments.

4.2 Pool level data

4.2.1 Construction of mortgage pools

According to the proposed approach (Section 2.3), we construct portfolios of mortgages from the loan data set. Pools are created by grouping single loans according to similar characteristics. Our approach is comparable to the process of grouping loans and issuing mortgage-backed securities by government-sponsored enterprises such as Fannie Mae and Freddie Mac. Specifically, we combine mortgage loans according to the origination year, US region, loan interest rate, loan original balance and FICO score. The specifications for interest rate, loan balance and FICO score are constructed by combining the values into bins. In this way, we desirably expect mortgage pools to comprise loans which should have similar prepayment characteristics. An example is a pool of loans originated in 2003, with loan sizes of less than \$80,000, interest rate between 3,5% and 4%, borrowers' FICO score less than 730 and the property securing the mortgage is located in Mid-West of the US.

Independent variables

For each mortgage pools, we generate the average of each loan-level input variable (Section 4.1). The averages are all weighted according to the current outstanding balance of the underlying loans. It is done by multiplying the feature value by the current outstanding balance and summing them all, and then, this total is divided by the total current outstanding balance of the pool. In this manner, the one-hot encoders of the categorical variables are transformed into percentage values of the total remaining pool balance. The input variables for our neural network model are listed in Table 4.1.

Target variable

From loan-level prepay amounts in equation (4.1) it is possible to obtain the target variable, which corresponds to the prepayment rate. This quantity indicates the monthly prepayment rate of a pool of mortgages and takes the name of *single monthly mortality* (SMM). In fact, the SMM represents the weighted average of all individual prepayment rates within that pool, and it is defined as the amount prepaid during the month divided

ID	Variable	Description
1	current_UPB$_{wa}$	weighted average current loan unpaid balance
2	current_UPB$_{sd}$	standard deviation current loan unpaid balance
3	SATO$_{wa}$	weighted average spread at origination
4	DTI$_{wa}$	weighted average debt-to-income
5	LTV$_{wa}$	weighted average loan-to-value
6	FICO$_{wa}$	weighted average credit score
7	unemployment$_{wa}$	weighted average unemployment rate
8	loan_size$_{wa}$	weighted average original loan size
9	loan_size$_{sd}$	standard deviation original loan size
10	interest_rate$_{wa}$	weighted average client's interest rate
11	interest_inventive$_{wa}$	weighted average interest inventive
12	rolling_insentive$_{wa}$	weighted average 24-months-rolling interest inventive
13	HPI_change$_{wa}$	weighted average house price index change
14	loan_age$_{wa}$	weighted average loan age
15	pre_crisis	0: after October 2019, 1: before October 2019
16.a	month$_{sin}$	*sin* representation of calendar month
16.b	month$_{cos}$	*cos* representation of calendar month
17.a	purpose_Ref$_{pct}$	percentage of loan purpose Refinanced loans
17.b	purpose_Pur$_{pct}$	percentage of loan purpose Purchased loans
18.a	occupancy_Own$_{pct}$	percentage of occupancy type Owner occupied loans
18.b	occupancy_Inv$_{pct}$	percentage of occupancy type Investor loans
18.c	occupancy_Sec$_{pct}$	percentage of occupancy type Second Home loans
19.a	region_South$_{pct}$	percentage of South region loans
19.b	region_West$_{pct}$	percentage of West region loans
19.c	region_Mid_West$_{pct}$	percentage of Mid-West region loans
19.d	region_North_East$_{pct}$	percentage of North-East region loans

Table 4.1: Model input variables

by the current outstanding balance at the beginning of that month,

$$SMM_t = \frac{\sum_i PRP_{it}}{\sum_i P_{i,t-1}}, \tag{4.2}$$

where PRP_{it} denotes the prepay amount from a particular loan i in month t and $P_{i,t-1}$ indicates the corresponding current outstanding balance at the beginning of that month (or 'starting balance'). Basically, it is like considering the pool as a massive mortgage with different starting balances every month. In fact, like interest rates, the prepayment rate is more often expressed in an annualized rate rather than as a monthly rate. The annual quantity is called *constant prepayment rate* (CPR), and the higher it is, the faster mortgagors are expected to repay their loans. The monthly SMM can be converted and annualized in terms of CPR by the following relation:

$$CPR = 1 - (1 - SMM)^{12}. \tag{4.3}$$

The SMM is the target variable of our regression model; the CPR, however, is more often used by banks and other mortgagees to quantify the prepayment risk. It indicates the

constant percentage of mortgage pools that assumed to be prepaid every year, and it can be seen as the prepayment speed of a mortgage pool.

Figure 4.1 plots the average CPR for each calendar month over the period from January 2000 to June 2019. The annualized repayment rate appears to show periods of high and low prepayment rates. From June 2001 until December 2003 the CPR is the highest, and it decreases to its minimum level in 2008. This decrease can be linked to the collapse of the US economy during the credit crisis of 2008. While prepayment was gradually increased over the period 2010 to 2014, it did not reach its pre-crisis level. The sharp decline at the beginning of 2013 and the downward trend from 2016 on can be explained by the increases of national mortgage rates and the resulting inverse refinancing incentives during these years.

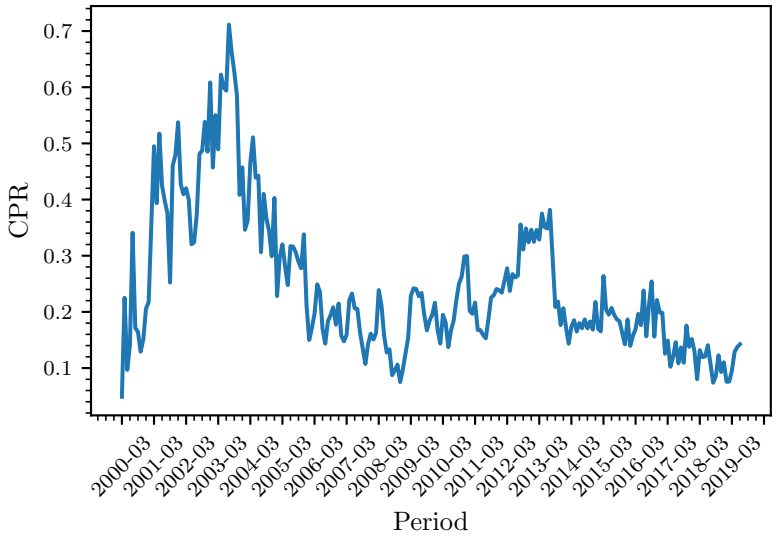
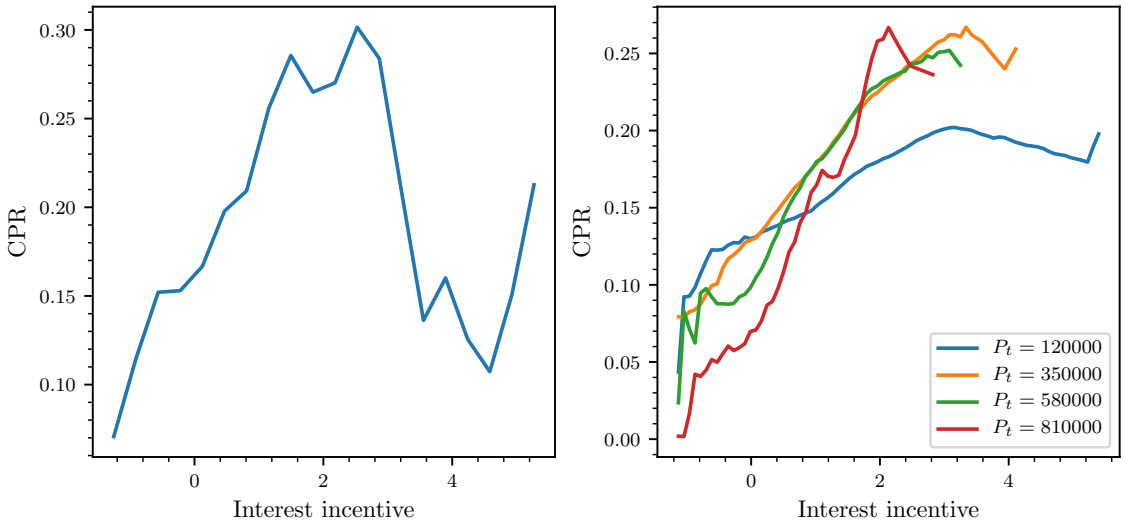


Figure 4.1: Time series of CPR

4.2.2 Nonlinear interaction effects

In this subsection, we analyse the response of prepayment rate to a few input variables to see whether prepayment behaviour is consistent with economic intuitions. We highlight the nonlinear relationships between prepayments and input variables, as well as the interaction effects of some input variables on prepayments. Note that we plot the CPR and not the SMM, as CPR is more expressive. We first examine the relationship that exists between prepayments and the two economically most significant prepayment drivers. The first variable is the interest incentive which is defined as the difference between the rate on the mortgage and the current market rate. As shown in Figure 4.2a, it appears that interest incentive does not influence the CPR in a linear increasing manner. Taking into account the fact that the gain from refinancing is higher when the remaining balance is higher, the likelihood of prepayment is strongly depended on the outstanding balance. Figure 4.2b shows how prepayments jointly respond to interest incentives and outstanding balances. It is visible in this figure that incentive has a positive influence on prepayments. Additionally, the CPR curves are steeper, and the prepayment speeds are generally higher for larger loan-size pools because the fixed portion of the refinancing costs often reduces the

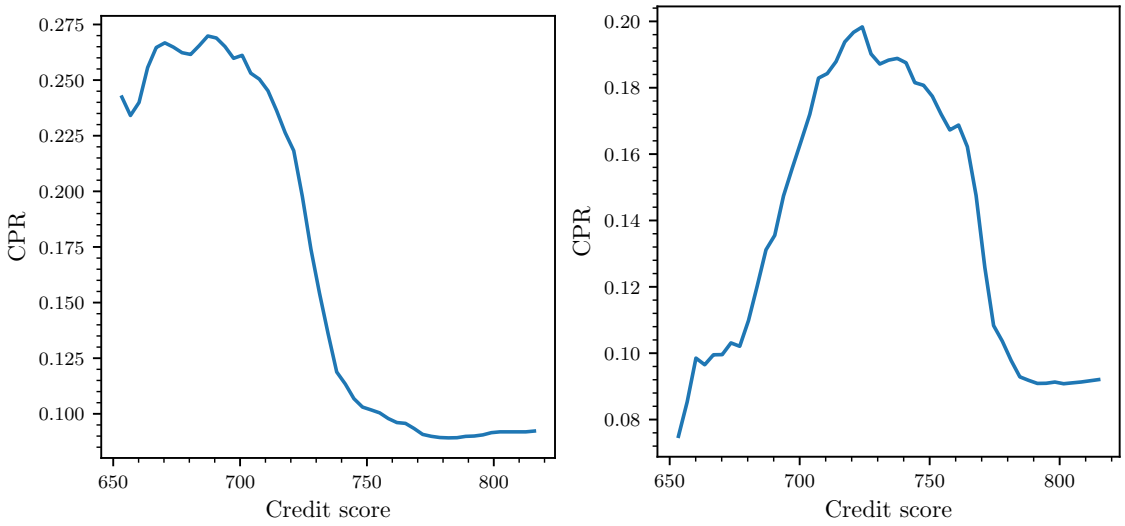
economic benefits of refinancing when the outstanding balance is relatively small. When interest incentive is negative, the motivation not to prepay is higher for larger loan-size pools, which is also in agreement with our observation.



(a) Interest incentive (b) Interest incentive and balance

Figure 4.2: Effects of interest incentives and loan balance

The next variables we analyse are the credit score and the pre-crisis indicator that reflects the regime change as discussed in Section 4.1. Note the contrast between the pre-crisis and post-crisis prepayment behaviour. As shown in Figure 4.3a, prepayment was



(a) Credit score prior to crisis (b) Credit score post crisis

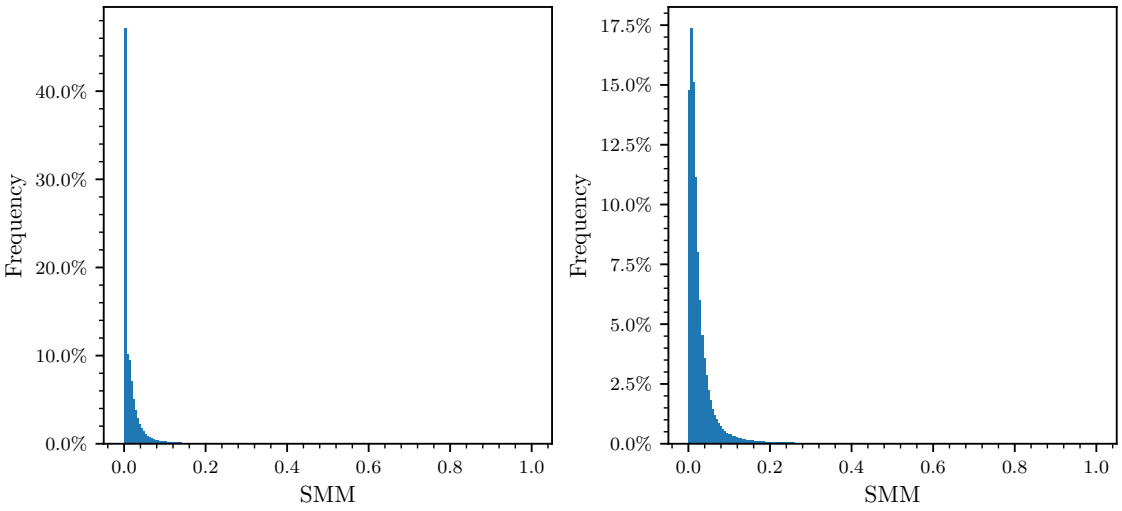
Figure 4.3: Effects of credit score and financial crisis of 2008

on average higher for borrowers with low creditworthiness in the pre-crisis period, and it was argued [44] that this was caused by massive house sells and the motivation to gain from house price appreciation. Low-scored borrowers had a stronger refinancing incentive

due to their relatively high mortgage rate. However, as shown in Figure 4.3b, borrowers that tend to be less creditworthy were slower to prepay after the crisis due to very tight mortgage underwriting standards.

4.2.3 Analysis of target variable

In equation 4.2 we define the SMM and consider it as the target variable of our prepayment problem. From the histogram in Figure 4.4a we can see that the observed SMM exhibits two features: A substantial proportion of the values are zero, and the remainder has a skewed distribution. The outcome containing true zeros may actually result from two separate processes: A process that is associated with the decision to prepay at all, and a process that determines the SMM value once it was decided to prepay. A possible approach as suggested by many researchers is using a so-called *two-factor* model [29], and in particular, a classification model for separating zero from non-zero observations, and a regression model to estimate the height of the SMM for non-zero observations. The two-factor model can be seen as a way of extending the regression model to better cope with the type of data we analyze in this work.



(a) including zero observations

(b) without zero observations

Figure 4.4: Frequency distribution of the target variable

Therefore, in addition to the initially suggested regression model, we develop a two-factor model based on two separate neural networks. The first factor is the one that assesses whether a particular observation is going to prepay or not. The values for this factor are obtained employing a binary classification, in which we assign the samples to the 'prepayment' class. Precisely, we do not focus on the final predicted class, but directly on the output of the model, which can be interpreted as the probability of belonging to the 'prepayment' class. The second factor assesses the height of the SMM. The values for this factor are obtained by solving the nonlinear regression problem. To conclude the final expected SMM rate, the two models are then combined by multiplying the estimated magnitude from the regression model with the probability from the classification model.

Supplementary, the skewed target suggests that we should also analyse the model fit. In general, a nonlinear regression model assumes that the residuals or the error terms of

the prediction $(y_i - t_i)$ are simply random fluctuations around the regression curve, and it is expected that the uncertainty in the prediction does not increase as the value of the prediction increases. This is the assumption of a homogeneous variance or homoscedasticity. Heteroscedasticity, on the other hand, is when the variance of the noise around the regression curve is not the same for all values of the predictor. When there is a sign of heteroscedasticity, it is often proposed to convert the target variable using one of the power transform techniques such as Box-Cox transformation [2].

One of the most common tools used to analyse heteroscedasticity is a *residual plot*, where residuals are plotted on the y-axis versus the fitted target on the x-axis. When the condition of homoscedasticity is satisfied, the residuals are randomly and uniformly scattered around the horizontal line at level 0. Figure 4.5 illustrates the residual plot from the pool-level data using the regression model, which will be briefly described in Section 5.2. Note that the upward-sloping edge presented in the residual plot is a result of zero-one bounded values of the target and the predictor. Nevertheless, there is no sign of heteroscedasticity in the regression, suggesting not to transform the target variable.

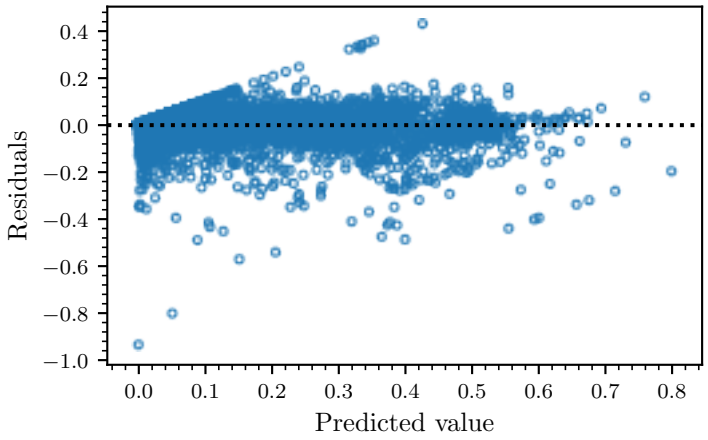


Figure 4.5: Residual plot

5. Model Selection and results

The construction of mortgage pools explained in Subsection 4.2.1 leads to a creation of roughly 1.1 million pool-level data points. We first consider observation from the reporting years 2000–2015. This data is split into a training, validation and test sets. Around 60% is selected as training data. 20% is used as validation data to tune the hyperparameters and to monitor possible overfitting. The remaining 20% of the data is left untouched until the model has been trained, and then it is used to test the final model. In addition to the test data set, we want to evaluate the predictive capabilities of the models on unseen out-of-time sample data. This data sample is a later data set than that, on which the model is fitted. Therefore, we keep all 2015–2018 data points to forecast the SMM outside of the estimation period, and we call this data collection the *forecasting set*.

Next steps are the selection of the input features and the values for the model hyperparameters. Obviously, the choice of features may have an impact on the optimal values of the hyperparameters, In contrast, the selected hyperparameters may affect the optimal choice of features. The best solution is probably to perform feature selection and hyperparameter tune 'at the same time'. However, due to computational restrictions, we could not afford to evaluate all the possible combinations by running a nested search of hyperparameters tune and feature selection. Therefore, we roughly run a hyperparameter search first, making sure not to assign extremely bad values. After that, we perform feature selection, with hyperparameters that are maybe not 100% optimized but at least not terrible either. Finally, we optimize the hyperparameters and model architectures with the features selected in the previous step.

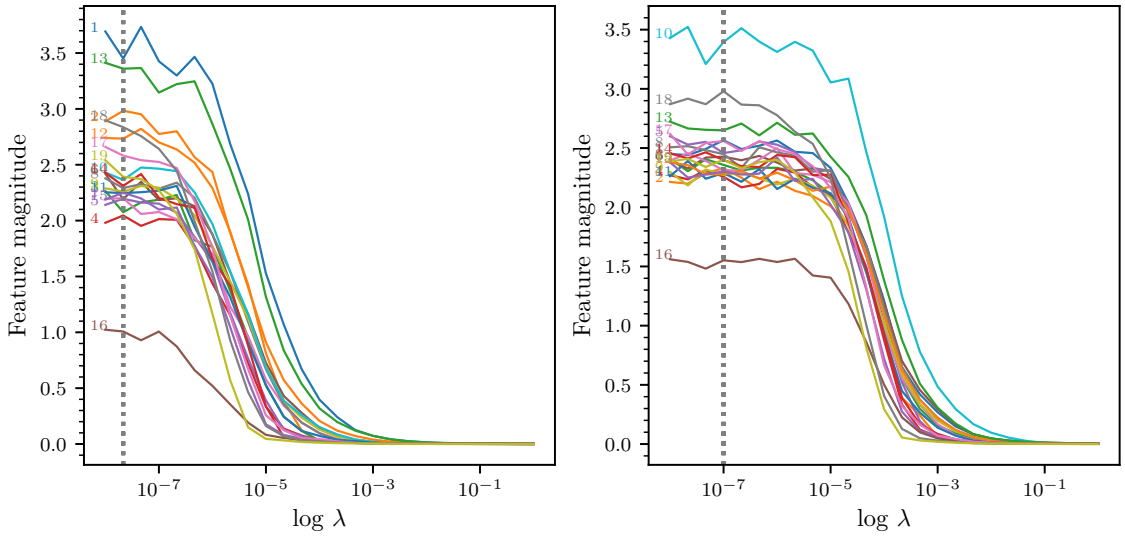
All techniques were evaluated using *K-fold cross-validation*, where K is set to 5. Generally, in K-fold cross-validation, the data is divided into K subsets. Each time, one of these subsets is used as the validation set, and the other $K - 1$ subsets are put together to form a training set. Model output is then averaged over all K trials. In this way, we combat the risk of losing important patterns in the data, increasing the robustness of our model.

5.1 Feature selection

Usually, a neural network does not require a sophisticated feature selection method. Instead, the network can choose the proper nonlinear forms of attributes and interactions. In fact, a property of neural networks is that they should be able to perform variable selection automatically within the network when there are sufficient hidden neurons and data. Explicitly, a neuron in the first layer tends not to be activated in the long run and will output zero value, if the corresponding feature is not meaningful for the prediction process. Nevertheless, in order to avoid overfitting and to keep the model computationally

less expensive, we perform the variable selection according to group lasso regularization as proposed in Section 3.6. We apply the selection process for both the regression and the classification model, separately.

The selection procedure is performed according to the following steps: We start the selection by training the models with all features that are listed in Table 4.1 and apply the group lasso method according to equation (3.10). For each input node in the networks, we include a regularization term pushing the entire row of outgoing weights to be zero simultaneously. We also decide to group all input nodes that are originally generated from categorical variables, and in particular, *sine-cos* representation of the month as well as the percentage of US regions, loan purpose and occupancy owner. Although they are not one-hot encoded, all members of each particular group should be either included or excluded together. We train the models with all variables and with a varying regularization parameter. Figure 5.1 shows the feature magnitudes as a function of the regularization



(a) Regression model

(b) Classification model

Figure 5.1: Magnitudes of input layer weights according to group lasso

parameter. The dotted grey lines indicate for each model the optimum point which is obtained as follows: Increasing regularization made first a small reduction of the validation error, and the optimum is reached when the validation error has started to increase. We compute the feature ranking at the optimal regularization parameter with respect to the validation error.

Next, we remove less important variables, one or two variables at the time, and repeat the procedure described above. We validate the ranking making sure that the validation error decreases when important variables are removed. We stop removing variables when the validation error at the optimal point has started to increase. The variables that are removed following this procedure are shown in Table 5.1.

5.2 Model architecture and hyperparameters

Neural networks have several hyperparameters whose values need to be tuned. The standard approach is to cross-validate these hyperparameters using the validation set. We

Regression model		**Classification model**	
ID	Variable	ID	Variable
16.a	month_{sin}	16.a	month_{sin}
16.b	month_{cos}	16.b	month_{cos}
4	DTI_{wa}	4	DTI_{wa}
3	SATO_{wa}	7	unemployment_{wa}
		2	current_UPB_{sd}

Table 5.1: Features removed during group lasso selection

train the networks with different hyperparameters on the training set and compare the error using the outcome of the cost function on the validation set. We start with the cross-validation of the model architecture, that is, the number of network layers and the number of neurons per layer. The more layers and more neurons a network has, the higher is the capability of this network to fit complex relationships. However, with more complexity, there is also a higher probability of overfitting. We also cross-validate the value of the learning rate, the batch size and the activation functions via a grid search. For each grid point, we train a neural network and then choose the hyperparameters at the grid point with the lowest validation error.

In the classification model, we set up the cross-entropy cost function in combination with the sigmoid output function. For the regression model, we minimize the mean squared error cost function incorporated with the sigmoid output function, because the target variable is bounded between 0 and 1. We noticed that using the sigmoid output function is a better choice for our regression problem since it squeezes the output into the interval $(0, 1)$.

Adam optimizer is a popular method in the neural network society; therefore, we chose to work with this optimizer. Cross-validation of the exponential decay rates of the Adam optimizer shows that the optimization is best with the proposed default values (Subsection 3.2.2). All decisions involve a lot of experiments. Surprisingly, there are no signs of overfitting in the optimum architecture. Therefore, weight regularization and dropout techniques are not applied to the models.

Hidden layers $\times$ **nodes**	**Batch size**	**Learning rate**	**Activation function**	**Regression loss** ($\times 10^{-4}$)	**Classification loss** ($\times 10^{-1}$)
1 x 80	256	0.001	LReLU $\alpha = 0.1$	3.489	2.0921
1 x 120	1024	0.01	LReLU $\alpha = 0.2$	3.821	2.0842
2 x 40	512	0.001	LReLU $\alpha = 0.1$	3.771	2.0710
2 x 80	256	0.001	LReLU $\alpha = 0.2$	3.362	**2.0337**
2 x 80	1024	0.001	LReLU $\alpha = 0.2$	3.615	2.0653
4 x 40	1024	0.001	LReLU $\alpha = 0.2$	3.655	2.0911
4 x 80	256	0.001	LReLU $\alpha = 0.2$	3.098	2.0440
4 x 80	256	0.001	ReLU	2.899	2.0461
4 x 80	256	0.001	LReLU $\alpha = 0.1$	**2.888**	2.0348
4 x 120	256	0.001	ReLU	2.932	2.0585
5 x 80	256	0.001	LReLU $\alpha = 0.1$	3.533	2.0655

Table 5.2: Results on the validation set for different setting

We test different combinations of hyperparameters for both neural network models. We then take the set of values with the lowest error achieved on the validation set. Some

of these settings, together with the respective validation error, are visible in Table 5.2. We highlight the combination that reveals to be the best in bold font.

5.3 Performance results

In this section, we proceed to show the results obtained in the prediction of the prepayment rate. We start with the performance of the models involved. Note that the prediction from the two-factor model is obtained by multiplying the results from both regression and classification models. To support our conclusion and to highlight the advantage of the nonlinear multi-layered models we developed, a baseline linear regression model based on a single-layered network is created. In particular, we set up a simple neural network with an input layer, no hidden layers and a sigmoid as output function which compresses the output into the interval $(0, 1)$. Feature selection and hyperparameters tune are not performed on this baseline model.

To evaluate the classification model, we choose the Accuracy metrics defined as the number of correctly classified data points divided by the number of all data points. The metric we use for other models is the *mean absolute error* (MAE), and it is calculated as the average of the absolute differences between the actual and the predicted values, $\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - t_i|$, where y_i and t_i are the predicted and the target values of the i'th test point, respectively. From an interpretation standpoint, MAE is preferred over the MSE we used for the loss function of the regression model in equation (3.3). In Table 5.3, the results for the performance metrics are displayed. We observed that the two-factor

Model	**Metric**	**Test set**	**Forecasting set**
classification	Accuracy	0.909022	0.887044
baseline	MAE	0.018601	0.011886
regression	MAE	0.004540	0.005681
two-factor	MAE	0.004295	0.005334

Table 5.3: Performance results

model performs slightly better than the regression model on both test and forecasting sets. Both models show a significantly improved performance over the baseline model. The baseline regression model has the highest MAE value, which indicates that there exist very complex nonlinear relationships in the data, and they cannot be well-predicted using a simple regression model. Note that we use the term 'simple regression' rather than 'linear regression', as the sigmoid function is used in the output of the baseline model. Since a small improvement is made by adding the results from the classification model to the regression model, we continue presenting other results using the two-factor model approach, and we draw the results from the baseline model for the sake of comparison.

Keep in mind that the classification model is an auxiliary component of the two-factor approach. In this work, thus, it is not considered as a stand-alone model for the prediction of the prepayment rate. In fact, all other models aim to forecast the SMM rate according to equation (4.2). Nevertheless, we proceed to show the results of the annualized prepayment rate, the CPR, by applying equation (4.3) to our estimations. Similar to the time series analysis in Figure 4.1, the predicted CPR values from the same month are clustered together and then averaged. A plot for the estimated CPR can be observed in Figure 5.2. The graph is divided into two parts by a vertical dotted grey line. The left side

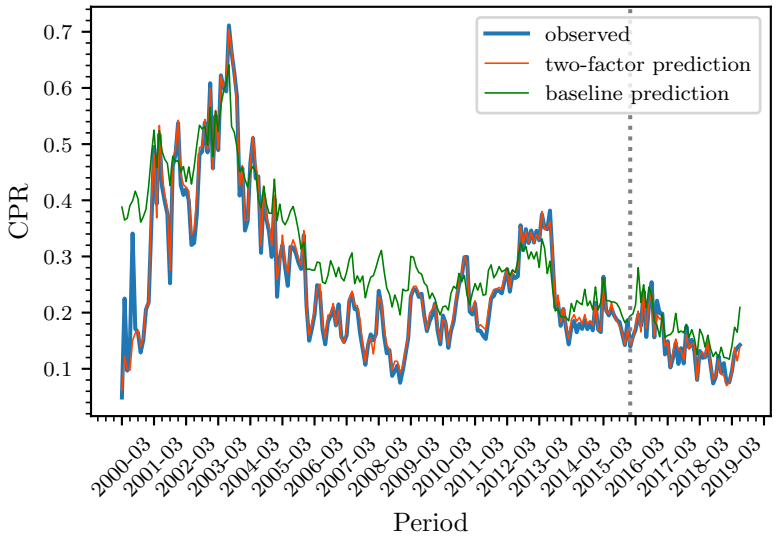
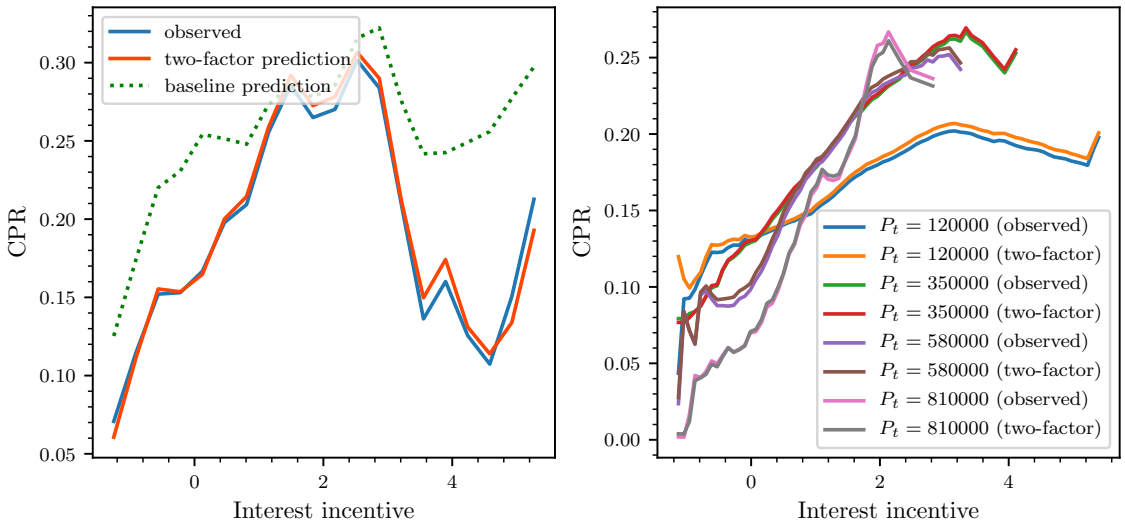


Figure 5.2: Model performance over time

shows the model performance on the test set, while on the right side, the results for the forecasting set are displayed. It shows that the two-factor model achieves better outcome than the baseline model, and it is able to accurately forecast the monthly averaged actual CPR with a very good in-time and out-of-time performance.

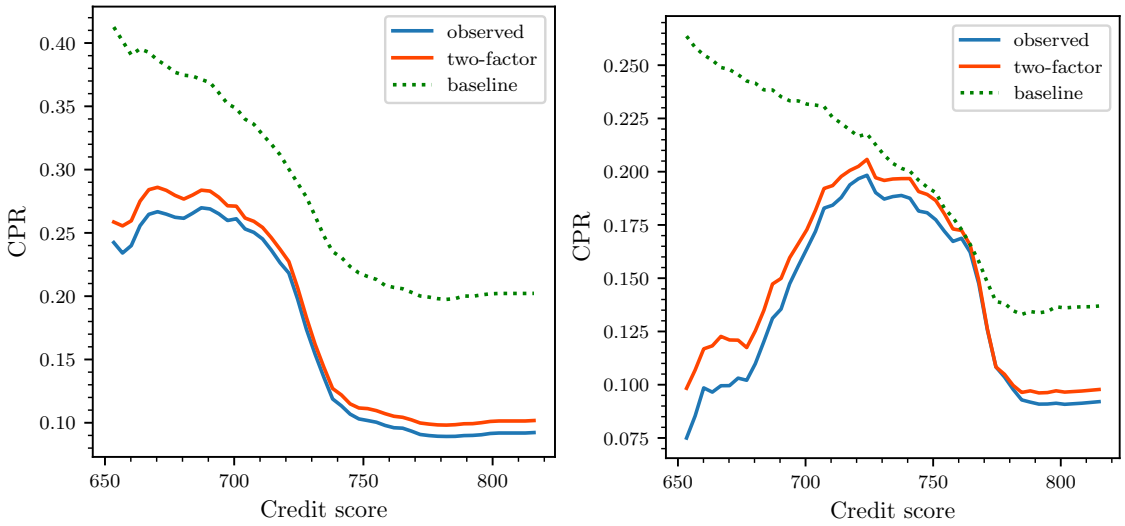
In order to enhance the transparency of our two-factors model, we track the prediction error of the CPR with respect to some risk factors. We plot the results similar to the plots we perform in Figures 4.2 and 4.3. We draw the predicted CPR curves versus the actual CPRs in Figure 5.3 and 5.4, and confirm that the two-factor model approach is well-performed. Note that the CPR curves from the baseline model are not displayed in Figure 5.3b to avoid an unreadable graph.



(a) Interest incentive

(b) Interest incentive and balance

Figure 5.3: CPR error tracking against interest incentive and balance



(a) Credit score prior to crisis

(b) Credit score post crisis

Figure 5.4: CPR error tracking against credit score and pre-crisis

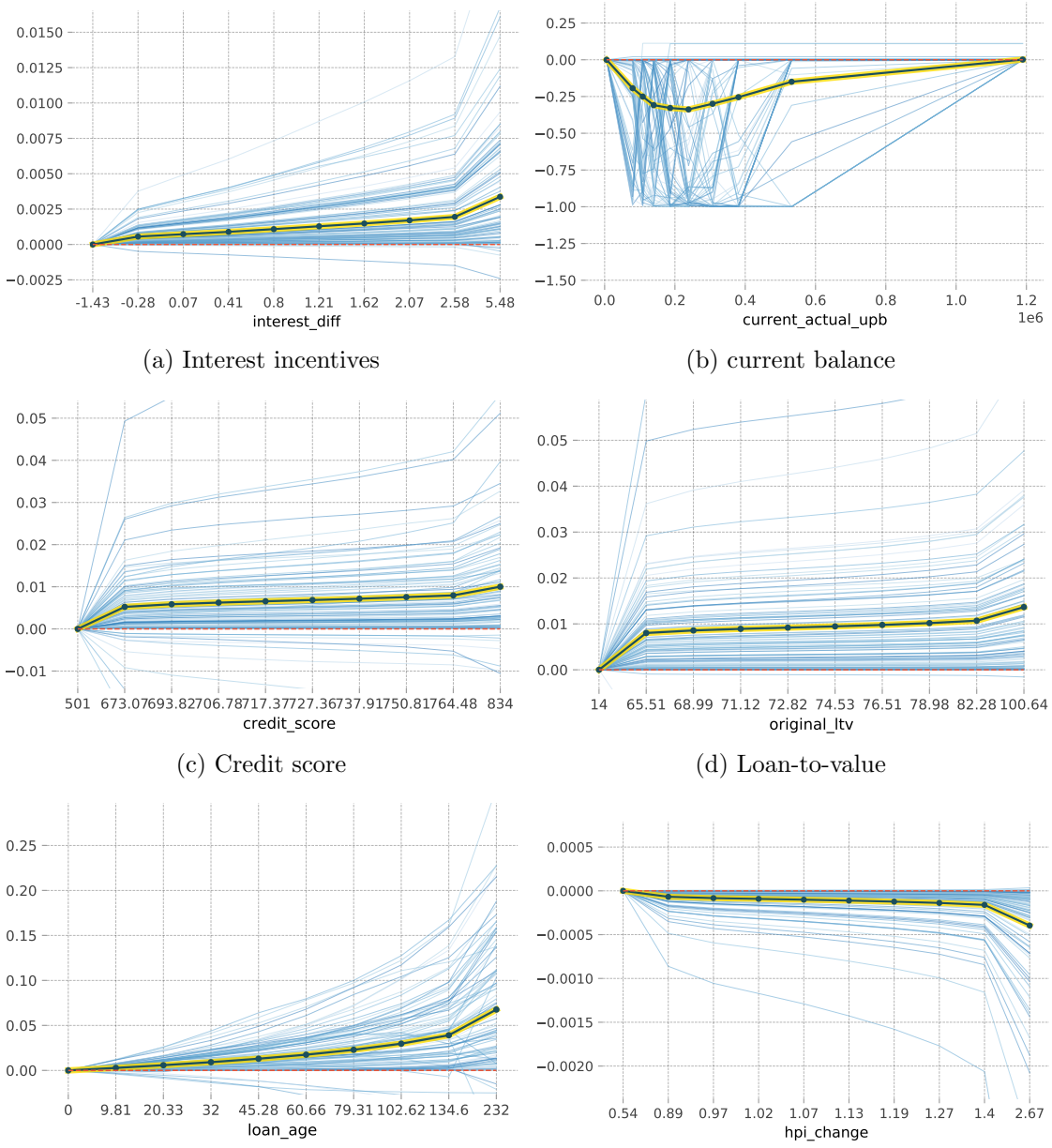
The two-factors model approach achieves a generally marginally small error tracking, and it is consistent with the economic intuitions discussed in Subsection 4.2.2. Again, much better results are realized from the two-factor model rather than from the baseline model.

5.4 Sensitivity analysis

This section gives practical insight into how neural networks act and in particular, how the input variables influence the output of the models. For linear models, this question is answered by looking at the model coefficients. One way to answer this question in the black-box world is through a partial dependence plot (PDP). As shortly introduced in Section 3.7, a PDP shows the marginal effect that one or more features have on the predicted outcome of the model. The PDP shows the average effect, but it does not focus on specific instances. Therefore, we also include the *individual conditional expectation* (ICE) plot [15] which visualizes the dependence of the prediction on its feature(s) for each instance separately, plotting one line per observation. The advantage of ICE over PDP is that it can uncover heterogeneous relationships.

In Figure 5.5, we plot the dependencies in the prediction process of the regression model for a couple of variables. The values of the input variable are on the x-axis, and the variation in model output is on the y-axis (therefore, negative values are also shown). The thicker blue-yellow line is the PDP, and it represents the overall relation between output and input by averaging the other thinner lines which represent the ICE, the behaviour of Individual observations. Note that we avoid plotting the lines for all data points since it would result in an unreadable graph. It appears that interest incentive, credit score, loan-to-value and loan age are, in general, positively correlated with the output of the regression model. However, the broad influence of HPI change on model output is negative. For low actual balance, increasing this quantity will cause a reduction in prepayments, but after a certain amount, the impact will change in the opposite direction.

The plots we present display one variable at a time. The impact on model output is valid only if the variable of interest does not interact strongly with other model variables. Partial dependence plots can be extended to check for interactions among specific model variables. By visualizing the partial relationship between the predicted response and one or more features, PDP illuminates the darkness of neural network models, so that such models can be interpretable.



(a) Interest incentives

(b) current balance

(c) Credit score

(d) Loan-to-value

(e) Loan age

(f) HPI change

Figure 5.5: Individual conditional expectation and partial dependence plots

6. Conclusion and outlook

This thesis investigates the prepayment behaviour of mortgage borrowers using the mortgage portfolio of Freddie Mac for mortgages originated across the US between 2000 and 2018. The aim has been to develop a prepayment model that is able to forecast the prepayment rate of mortgages portfolios and it also able to incorporate the nonlinear influence of many prepayment drivers on the economically irrational behaviour of borrowers. In particular, we have decided to focus on nonlinear models based on artificial neural networks, formulating the prepayment analysis as a regression problem. We have also analyzed the distribution of the target variable and proposed to enhance the regression model. It is the two-factor model framework in which two different aspects are considered: the first one determines whether a mortgage will prepay or not, the second one assesses the magnitude of prepayment. The two-factor model approach is based on a combination of classification and regression neural network models.

From the results in Section 5.3, it has become clear that both multi-layered neural network models we developed, the regression model and the two-factor model approach, have overperformed the baseline model, and they are able to successfully capture the underlying relations between the explanatory variables and prepayments. It does confirm the applicability of neural networks in prepayment modeling. The estimation of the prepayment rate has been accurate, and this is promising. We have achieved slightly better results from the two-factor model approach, comparing the results from the regression model. The prediction of the SMM has been improved by approximately 0.03% on average for each month. It is equivalent to the improvement of the CPR prediction by 0.4%. Such a small improvement could seem meaningless, but we have to consider that we are improving absolute errors which are expressed in percentages; therefore, this improvement is significant. Let us consider 100 billion as a reasonable size of the mortgage portfolio of a bank. 0.4% difference in the CPR is translated into the improvement of the prediction error by 400 million prepaid amount. Such a quantity does make an impact on the hedging strategy of the bank when mitigating the prepayment risk. Because a massive amount of money is involved, it is essential to keep improving the model and keep training it on large and recent datasets.

Unfortunately, we have not got access to real detailed borrowers‘ specific data. The actual relations between the explanatory variables and prepayments are likely to be a lot more complex than those we inspected. In consequence, our models have been all constructed at pool-level using summary statistics for characteristics of the underlying loans. Although an aggregate-level analysis can provide a reasonable approximation to the results of the loan-level study, the full heterogeneity of borrowers' behaviour is difficult to capture with an aggregate-level model. Moreover, it would be difficult to port the results of a pool-level analysis to the prediction of loan-level prepayments. In case detailed

borrowers' specific data are available, banks may think moving toward a loan-by-loan approach to evaluate mortgage prepayment at loan-level as well.

Note that we have used the observed input variables to achieve the out-of-time prediction of prepayment rates. To run a 'full' forecast in a production environment, one should be able to progress the mortgage portfolio through time and forecast all input variables, including macroeconomic factors. Forecasting macroeconomic variables are beyond the scope of this thesis. However, there are models available to predict the house-price-index and unemployment rates. Market mortgage rates are often derived from the swap rates plus a specific spread. Swap rates can be evolved using an interest rate model to specified the stochastic dynamics of the interest rate.

When working with neural networks in financial risk modeling, we need to consider the dynamics in the financial markets and in client behaviour which may change from year to year. Therefore, the models need to be regularly retrained as new data becomes available, and recent data should be added to the training set. Regulatory changes may cause data to be irrelevant for the model, and thus, they can be removed from the training set. To be able to train neural network models well, large amounts of data are necessary. In the case of mortgage prepayments, this is not a restriction, because massive amounts of prepayment data can usually be collected and used for training.

Finally, we want to comment on the so-call the 'black box' property of neural networks. Due to the highly nonlinear transformation of input data and the large amounts of trainable parameters, it is difficult to track the impact of changing one of the input variables, Hence, the network can not provide banks with general and simple rules for their decision-making. Undoubtedly, banks could analyze the sensitivity of the network output and visualize the joint impact of different explanatory variables. Still, in the high dimensional prepayment world, this is most probably not insightful. Perhaps, the best way to convince banks and regulators is to show the performance of the network on test data sets.

Due to the theoretical capability of artificial neural networks to approximate very sophisticated functions, they have the potential to generate better prepayment predictors than the models currently used by banks and other financial institutions. Because of that, we advise to continuing the research in their applications. Neural network-based approaches have the potential to add value to prepayment risk modeling and the complex field of behavioural finance.

Bibliography

[1] C. M. Bishop. *Pattern recognition and machine learning*. Springer-Verlag, 2006.

[2] G. E. Box and D. R. Cox. An analysis of transformations. *Journal of the Royal Statistical Society: Series B (Methodological)*, 26(2):211–243, 1964.

[3] R. Caruana, S. Lawrence, and C. L. Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In *Advances in neural information processing systems*, pages 402–408, 2001.

[4] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[5] M. Consalvi and G. S. di Freca. Measuring prepayment risk: an application to unicredit family financing. Technical report, Working Paper Series, UniCredit&Universities, 2010.

[6] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[7] Y. Deng, J. M. Quigley, and R. Van Order. Mortgage terminations, heterogeneity and the exercise of mortgage options. *Econometrica*, 68(2):275–307, 2000.

[8] *Freddie Mac House Price Index*. Federal Home Loan Mortgage Corporation, . http://www.freddiemac.com/research/indices/house-price-index.page.

[9] *Freddie Mac Primary Mortgage Market Survey*. Federal Home Loan Mortgage Corporation, . http://www.freddiemac.com/pmms/pmms_archives.html.

[10] *Single family loan-level dataset*. Federal Home Loan Mortgage Corporation, . http://www.freddiemac.com/research/datasets/sf_loanlevel_dataset.page.

[11] *Fannie Mae and Freddie Mac*. Federal Housing Finance Agency, . https://www.fhfa.gov/SupervisionRegulation/FannieMaeandFreddieMac/Pages/About-Fannie-Mae---Freddie-Mac.aspx.

[12] J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[13] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[14] F. Godin, J. Degrave, J. Dambre, and W. De Neve. Dual rectified linear units (drelus): a replacement for tanh activation functions in quasi-recurrent neural networks. *Pattern Recognition Letters*, 116:8–14, 2018.

[15] A. Goldstein, A. Kapelner, J. Bleich, and E. Pitkin. Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation. *Journal of Computational and Graphical Statistics*, 24(1):44–65, 2015.

[16] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3:1157–1182, 2003.

[17] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[18] G. Hinton, N. Srivastava, and K. Swersky. Neural networks for machine learning, lecture 6a: overview of mini batch gradient descent. http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

[19] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[20] K. Hornik, M. Stinchcombe, H. White, et al. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[21] J. P. A. M. Jacobs, R. H. Koning, and E. Sterken. Modelling prepayment risk. *Dept. Economics, University of Groningen*, 2005.

[22] P. Kang and S. A. Zenios. Complete prepayment models for mortgage-backed securities. *Management Science*, 38(11):1665–1685, 1992.

[23] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980v9*, 2017.

[24] D. G. Kleinbaum and M. Klein. *Survival analysis*. Springer-Verlag, 2010.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[26] V. Kumar and S. Minz. Feature selection: a literature review. *Smart Computing Review*, 4(3):211–229, 2014.

[27] L. London. *Encoding cyclical continuous features - 24-hour time*. https://https://ianlondon.github.io/blog/encoding-cyclical-features-24hour-time/.

[28] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *International Conference on Machine Learning*, 2013.

[29] M. K. Olsen and J. L. Schafer. A two-part random-effects model for semicontinuous longitudinal data. *Journal of the American Statistical Association*, 96(454):730–745, 2001.

[30] N. Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

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

[32] T. Saito. Mortgage prepayment rate estimation with machine learning. Master's thesis, Delft University of Technology, 2018.

[33] J. Schultz. The size of the affordable mortgage market: 2015-2017 enterprise single-family housing goals. *Federal Housing Finance Agency*, 2014.

[34] M. Sherris. Pricing and hedging loan prepayment risk. *Transactions of society of actuaries*, 1994.

[35] P. Y. Simard, D. Steinkraus, J. C. Platt, et al. Best practices for convolutional neural networks applied to visual document analysis. In *Proceedings of the Seventh International Conference on Document Analysis and Recognition*, pages 958–962, 2003.

[36] J. Sirignano, A. Sadhwani, and K. Giesecke. Deep learning for mortgage risk. *arXiv preprint arXiv:1607.02470*, 2016.

[37] T. G. Slatton. A comparison of dropout and weight decay for regularizing deep neural networks. 2014.

[38] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B*, 58(1):267–288, 1996.

[39] *State Employment and Unemployment*. U.S. Bureau of Labor Statistics. https://www.bls.gov/web/laus.supp.toc.htm.

[40] *Census Regions and Divisions of the United States*. US Census Bureau. https://www2.census.gov/geo/pdfs/maps-data/maps/reference/us_regdiv.pdf.

[41] Y. Yu. Msci agency fixed rate refinance prepayment model. Technical report, MSCI inc., 2018.

[42] M. Yuan and Y. Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, 2006.

[43] C. Zhang and P. C. Woodland. Parameterised sigmoid and relu hidden activation functions for dnn acoustic modelling. In *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[44] J. Zhang, F. Teng, S. Lin, et al. Agency mbs prepayment model using neural networks. *The Journal of Structured Finance*, 24(4):17–33, 2019.