A Deep Learning Prediction Model for Mortgage Default

Michael J. Cooper

A Thesis presented for the degree of
Masters of Engineering

Department of Engineering
University of Bristol
England

May 2018

A Deep Learning Prediction Model for Mortgage Default

Michael J. Cooper

Submitted for the degree of Masters of Engineering
May 2018

Abstract

Artificial Neural Networks (ANNs) have the ability to find non-linear correlations between sparse input data and desired outputs. This makes them well-suited to classification problems such as Mortgage Default Prediction.

This thesis explores how deep learning can be used to classify mortgages into Default or Fully Paid Loans. It compares the Recall, Sensitivity and AUC scores of different deep neural network architectures against a baseline linear classifier. This work examines the effect of different resampling, regularisation and cost-sensitive learning methods on the neural networks classification performance. The research uses the Freddie Mac Single-Family Loans dataset, which is used to create novel geographically based features on a per loan basis. The model is trained on an imbalanced dataset which includes using 15.3 million unique loans with 326 million performance updates, achieving an AUC score of 0.984.

Declaration

The work in this thesis is based on research carried out at the Department of Engineering, University of Bristol. No part of this thesis has been submitted elsewhere for any other degree or qualification and it is all my own work unless referenced to the contrary in the text.

References.
All references are hyper-linked to the bibliography.

Executive Summary

The purpose of this thesis is to explore the effectiveness of deep learning on Mortgage Default Prediction. The 2007-09 financial crisis led many independent examinations on both large and medium-sized banks. The overriding conclusions from these examinations were that banks must improve their credit risk management, specifically outlining the expectation to have adequate early warning systems in place to identify high-risk loans ahead of time. These conclusions layout the main motivation for this thesis.

We defined the Mortgage Default classification problem as the following: given a set of features for a mortgage at a specific point in time, output a prediction probability indicating whether the mortgage was more likely to be Fully Paid or Default at the end of the loan period. We used the Freddie Mac Single-Family Loans dataset which contains fully amortising 15, 20, and 30-year fixed-rate mortgages, covering approximately 60% of all mortgages originated in the U.S. from January 1999 to January 2017. Each loan has monthly performance updates which include information such as Current Unpaid Balance, Status[1], Loan Age, etc...

In order to assign a classification label, we only considered loans that had finished and therefore had a final state. After cleaning the data, we had 15.2m Fully Paid and 85k Default loans with approximately 326m performance updates. Using the conclusions from previous related work, we introduced new features to the data based on geographical location and individual loan performance, as well as adding economic data such as Housing Price Index, Unemployment rates, and National Interest rates. We then introduced the same set of features but which only consider the recent past, i.e. the last 12 months. We found that these recent past features are more effective when applied to economic and geographical based features and less so when applied to individual loan performance features.

Our results section compares a variety of optimisation methods; these include

[1]The status of a loan is defined in this work as either current, 30, 60, 90+ days delinquent, Foreclosed, REO (Real-Estate Owned) or Fully Paid.

resampling, normalisation, regularisation and cost-sensitive learning. We also evaluated performance under different feature selections and model architectures, where all comparisons are evaluated using 5-fold cross-validation. The model achieves an AUC score of 0.984 under the optimum choice of parameters and techniques. The following paragraphs will cover a subset of these methods and techniques in more detail, outlining the result achieved and conclusions drawn.

As observed from the difference between the number of Fully Paid and Default loans, there is a significant class imbalance. We approach this problem using two methods; a weighted loss function and resampling techniques. We weight the loss function such that, for each class, the output of the loss function is proportional to the inverse class ratio. We found that this alone did not produce the desired results, the Recall (rate of correctly classified Default instances) value was still significantly lower than desired at 0.413, with an AUC of 0.702. We observe that when no re-sampling method is applied, irrespective of the weighted class ratios, the model performs poorly. We hypothesise that because the minority class instances are processed so infrequently, it makes the decision regions very small relative to the overall vector space and therefore it is difficult for the network to converge optimally.

We therefore applied different resampling methods in order to correct the class-imbalance and then evaluated their effectiveness. We compared the Synthetic Minority Over-Sampling Technique (SMOTE) with a basic random under-sampling method, at a variety of class ratios. We found that SMOTE does not outperform random under-sampling at any class ratio, with the optimal resampling method being random under-sampling at a class ratio of 15:85 (Default : Fully Paid).

We tested several different model architectures at varying depths and found that the optimum consists of 2 hidden layers each with 100 nodes. Across all architectures, we find that two hidden layers perform better than either one or five, and five perform better than one. These results suggest that adding more than one hidden layer allows the model to learn more complex non-linear relationships, directly resulting in higher predictive power and performance. The linear model (0 hidden layers) has the lowest AUC and Recall values. This result suggests there exist complex non-linear relationships in the data that cannot be separated using linear regression.

We evaluated the model's performance under a variety of different feature sets. As previously stated, the optimum AUC score of the model was 0.984. This was achieved by evaluating the prediction output for each monthly performance update across the lifetime of a loan. Most interestingly, we decided to remove all monthly performance updates, with the exception of the first (the origin point of the loan). We then trained the model using only this subset of data where only a single labelled input vector exists per loan, and this achieved an AUC score of 0.707. Additionally, we trained the model with and without the additional features we added to the data set (as discussed previously); without this the model achieves an AUC score of 0.893, and with, it achieved the optimum value of 0.984.

In conclusion, we confirmed that a deep neural network (0.984 AUC) significantly outperforms a basic linear regression model (0.799 AUC), therefore validating the performance benefits of neural network models in this domain. If we compare our work to similar research as published by Bagherpour [2017] and Deng [2016], both papers implement models that attempt to classify loans that are 'Paying' versus 'Default' (our work looks at 'Default' versus 'Fully Paid' and does not consider active loans) using a logistic regression model. They achieved AUC scores of 0.860 and 0.968 respectively. Our results compare favourably to these. However, we do note that there is a difference in the exact classification objective between our paper and theirs.

We showed that the addition of novel features improved the classification performance increasing the AUC score from 0.893 to 0.984. As far as we are aware, many of the additional features introduced have not been previously researched in academic literature. We believe that this is due to limitations in data sources up to now, and also the computational requirement to process data on this scale. We, therefore, believe this provides a new and useful contribution to the field.

Future work could include an investigation into specific sub-classes of loans. For example, it would be interesting to know how well a neural network would perform using only Non-Performing Loans (NPLs)[2]. Similarly, it would be interesting to build separate models for different clusters within the data, for example, a predictive model for each state within the USA.

[2]Loans are Non-Performing if they are 90+ days delinquent

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Predicting mortgage delinquency[1] is a non-trivial problem due to a large number of variables that affect the outcome of a loan (discussed in section 1.2). Typically, for an individual to first hold a mortgage, some background check must have been completed whereby a suitable candidate is accepted, and an unsuitable candidate is rejected. This process should, in theory, constrain the pool of accepted candidates making the challenge of predicting the outcome more difficult.

We propose the use of machine learning methods to address this problem. They are typically well suited to finding non-linear relationships, the nature of which were otherwise not considered by the mortgage lender at both the time of issue and during the lifetime of the loan. This work focuses on Deep Neural Networks, as they are known for their ability to learn intermediate features between high dimensional data and high-level classification.

The purpose of this thesis is to implement a neural network model which can effectively differentiate between loans that are likely and unlikely to default. The models will be trained using the Freddie Mac Single-Family dataset from 1996 to 2016. This work compares a variety of deep learning methods; these include resampling, normalisation, regularisation and cost-sensitive learning. We also evaluate performance under different feature selections and model architectures, where all

[1]Delinquency is the term used when a borrower has failed to make scheduled repayments as defined in the loan documents.

comparisons are evaluated using 5-fold cross-validation. The specific classification task is to predict whether a mortgage, with n features, is more likely to be Fully Paid (i.e. the borrower pays back in full) or Default (i.e. the borrower defaults on the loan).

1.1 Motivation

The 2007-09 financial crisis has been described as the worst financial event since the Wall Street Crash in the early twentieth century which led to the Great Depression. There is overwhelming evidence which suggests that poorly executed risk management was one of the leading factors that led to this downfall; see Crouhy [2011], Ional et al. [2011], Flood et al. [2011]. For example, Goldman Sachs adjusted its positions in mortgage-backed securities[2] (MBS) in 2006, which differentiated themselves from the rest of the market, and is one explanation as to why they avoided the substantial losses suffered by Bear Stearns, Merrill Lynch, Lehman Brothers and others; see Jorion.

Since then, there appears to have been a significant investment by banks and other asset management institutions into risk management in general, as explained by Young [2013]. However, after many independent examinations on both large and medium-sized banks, there is still a concern. One such assessment conducted by De Nederlandsche Bank in 2015 concluded that "banks must improve their mortgage portfolio credit risk management". Specifically, they expect banks to have adequate early warning systems in place to identify and monitor high-risk loans ahead of time. The type of model we are proposing in this thesis could, as part of a larger decision process, contribute to such an early warning system.

If we look at consumer level spending in general capacity, the US Bureau of Labor Statistics states that in Q4 2017, consumer spending accounted for about 65% of U.S. Gross Domestic Product (GDP). It also reports that there is $3.8 trillion

[2]A mortgage-backed security (MBS) is type of asset-backed security that is secured by a set of mortgages

of outstanding consumer credit and in the last quarter of 2017, revolving credit[3] accounted for over 26% ($1.03 trillion). A 0.5% increase in identifying risky loans could prevent loss of over $5 billion. The continuous social and economic changes in our environment significantly affect consumer spending; see Parker et al. [2011]. This puts increasing importance on building accurate models to limit financial risk on behalf of the mortgage vendor.

The societal motivation for this research has been outlined above; we will now introduce the technical motivations in more detail. In recent years, there has been a significant increase in Deep Learning research; Gartner, Inc. predicts that, by 2019, deep learning neural networks will underpin the systems that performance best in three areas relevant to the financial sector: demand, fraud and failure predictions. The escalation in the field of deep learning has mainly been due to a number of aspects; improved availability of high-quality data, increased computational processing capacity (Graphical Processing Units (GPUs) specifically), improved mathematical formulas and more easily accessible frameworks such as Googles Tensorflow. These improvements have allowed data scientists to add significantly more layers within a network than previously possible, this addition has lead to the impressive results we see today; see Schmidhuber [2015].

We believe the societal and technical motivations are significant, providing a high incentive to explore these two domains in more detail.

1.2 Related Literature

The application of Machine Learning, specifically Deep Learning, in the financial domain is a relatively new concept, and as such, there exists only a small selection of related literature. However, there is an abundance of previous research into mortgage default in general. This section is therefore split into two parts; General, which will focus on research that analyses factors that affect mortgage default as well

[3]A line of credit where the customer can repeatedly borrow up to certain limits as long as the repayments are made on time.

as prepayment[4], and Machine Learning, which focuses on machine learning research specifically used to predict mortgage default.

1.2.1 General

In 1969, von Furstenberg released a paper that quantitatively analysed the impact of loan-to-value ratio, income and loan age on mortgage default rates. With the use of simple regression models, he concluded that there was significant correction between these variables and mortgage default rates. Subsequently, Vandell (1978), Webb (1982), Campbell and Dietrich (1983), and others all published research examining additional variables and their impact on mortgage default rates.

In 1998, Ambrose and Capone analysed a subset of Federal Housing Administration (FDA) loans that had defaulted, finding that Loan-to-Value ratio is a significant indicator of whether a delinquent loan will be either reinstated, sold or foreclosed. Similarly, Capozza and Thomson (2006) analysed a subset of sub-prime mortgages that had defaulted. Using a multinomial logistic regression model they found that loans with higher interest rates were less likely to enter the state, Real Estate Owned (REO), whereas loans with fixed interest rates and higher Loan-to-Value ratios had higher foreclosure probability.

In 2005, Danis and Pennington-Cross analysed a subset of fixed-rate mortgages which originated between 1996 and 2003. Using a multinomial logit model they found that loans with a higher FICO (credit scores) and longer periods of delinquency were associated with a higher probability of prepayment when compared to foreclosure. Whereas they found that negative equity[5] was associated with a higher probability of foreclosure when compared to prepayment. Following this research, Danis and Pennington-Cross released another paper in 2008 which analysed a larger sample (5,000 subprime mortgages) of the dataset over the same period. They found that loans with a higher FICO (credit scores) and longer periods of delinquency were

[4]Mortgage prepayment is where the borrower repays the mortgage before the end of the loan term.

[5]Negative equity occurs when the value of an asset used to secure the loan is less than the current unpaid balance.

less likely to foreclosure, but, in contrast to their last publication, they were also less likely to prepay. They also found that housing price volatility was a significant indicator of foreclosure, whereas unemployment rates were not correlated; see Danis and Pennington-Cross [2008].

Voicu et al. [2012] analysed mortgages that originated between 2004 and 2006 at a national level across the United States. After supplementing the mortgage data with economic data from the U.S. Bureau of Labor Statistics, they found that "default outcomes are affected by local economic and housing market conditions, the amount of equity in the home, and the states legal environment".

Chana et al. [2014] published a paper detailing their findings on mortgages that originated between 2003 and 2008 in New York City. In an empirical study, they found that loans with higher current balances, larger Loan-to-Value ratios and steeper declines in FICO score, are more likely to default or be refinanced. They also found an association between neighbouring characteristics and higher default rates. These areas include; districts where house price depreciation is greater than 10 percent in the past year and locations that see a sharp increase in the rate of foreclosure within the last six months.

1.2.2 Machine Learning

More recently, Sirignano et al. [2016] published research examining the effectiveness of a deep neural network at predicting the status[6] of a loan. They used a licensed dataset from CoreLogic with mortgages that originated from 1995 to 2014 covering roughly 70% of all mortgages originated in the US. Their results show the significance of local economic factors and state unemployment rates for explaining borrowers behaviour as well as improving the model's performance.

The research that is most aligned with this thesis has been published by Bagherpour [2017] and Deng [2016], both papers implement models that attempt to classify loans that are 'Paying' versus 'Default' (our work looks at 'Default' versus 'Fully

[6]The status of a loan is defined in this work as either current, 30, 60, 90+ days delinquent, Foreclosed, REO (Real-Estate Owned) or Fully Paid

Paid' and does not consider active loans, explained in more detail in section 4.2.1). They analysed the effectiveness of different Machine Learning methods (Support Vector Machines (SVM), Logistic Regression and Random Forrest (RF)) at predicting mortgage default on loans from the Fannie Mac 30-year fixed rate mortgage dataset.

Bagherpour [2017] compares the performance of a Random Forest model with a Support Vector Machine and Logistic Regression model. The logistic regression model performs best on the 'Paying' versus 'Default' classification task achieving an Area Under the Curve (AUC)[7] score of 0.860. The paper states that it uses the SMOTE upsampling method (discussed in Section 2.2.1) on both the training and testing datasets to obtain the equal class ratio (50:50). We note that it is uncommon to re-sample both the test and training sets. For example, in the literature that introduces SMOTE by Bowyer et al. [2011], the up-sampling technique was only performed on the test set. By applying the method to both datasets, the practical application of the model will be diminished, as real-world datasets are unlikely to have a 50:50 class ratio.

Deng [2016] compares Logistic Regression, K-Nearest Neighbors and Random Forest. Similar to A.Bagherpour, Deng shows that the logistic regression model performs best on the 'Paying' versus 'Default' classification task achieving an AUC score of 0.968. Unfortunately, Deng's work does not include specific implementation details, and as such, it is difficult to thoroughly analyse the results.

Conclusion

In this chapter, we outlined the motivation for the research and provided a brief introduction to previously published work in the field of mortgage default. This research showed that metrics such as loan age, FICO, Loan-to-Value ratio, Debt-to-Income ratio and local economic factors appear to correlate with mortgage foreclosure rates and prepayment likelihood. Finally, we introduce two pieces of literature,

[7]Area Under the Curve (AUC) is a classification performance metric that considers the proportion of correctly predicted instances. The performance metrics contained within the thesis are detailed in section 6.

Bagherpour (2017) and Deng [2016], that implement models similar to that which are proposed in this work, where they achieved an optimal AUC score of 0.860 and 0.968 respectively.

Chapter 2

Background

2.1 Credit Risk

This section introduces the concept of different credit scores and outlines how these are used in this paper.

2.1.1 Credit Scoring

A credit score is a numerical value used to represent an individuals credit worthiness. Crouhy [2011] states that a credit score is primarily determined from a credit report issued by credit bureaux, which typically include payment history, debt, the length of time the individual has been on file, account diversity and the number of new credit inquiries.

A financial institution often uses a credit score to help decide whether an individual should be accepted for a loan or not. Commonly, providers use a threshold system, whereby customers who have a score above the threshold are less risky, and more likely to meet the financial repayment obligations; see Parker et al. [2011].

There are two types of credit scoring: application scoring and behavioural scoring. Application scoring is determined at the start of the loan, whereas behavioural scoring is used as an ongoing metric over some period of time. Only the application score, which we refer to as the 'FICO' score, is provided within the dataset. We do however create additional features which attempt to capture the trends of ongoing loans, and we use these as behavioural scoring metrics; discussed in more detail in section 4.4.

2.2 Resampling and Normalisation

This section provides an overview of the different normalisation and resampling techniques that we later compare in the results section (6).

2.2.1 Resampling Methods

Resampling methods allow for repeated sampling of the original data to build a new sample distribution. There are two main types of resampling, under-sampling and over-sampling. Resampling methods have been shown to improve classification performance when handling imbalanced data; see Buda et al. [2017]. They are typically used to obtain more equal class ratios, where either the minority class is over-sampled, or the majority class is under-sampled.

SMOTE

The Synthetic Minority Over-Sampling Technique (SMOTE) was introduced by Bowyer et al. [2011]; since its introduction it has become a widely used and researched oversampling technique. As explained by Hoens and Chawla [2013], the algorithm creates synthetic instances that belong to the minority class, and, in general, have been extrapolated from existing minority class instances. A new synthetic instance is created by randomly selecting a minority class instance, it then uses the nearest neighbour search algorithm to find the k-nearest minority class samples. The new instance is created by selecting a neighbour at random, taking the vector between the neighbour and the original instance, and multiplying by x, where x is a random value between 0 and 1. This vector is then added to the original instance to create a new synthetic instance.

There are a few disadvantages to the SMOTE algorithm. It is computationally expensive to perform SMOTE on a large dataset because, as stated by Xie [2016], the time complexity of the k-nearest neighbour method is linear to the size of the training dataset. Another problem is that SMOTE can lead to the newly synthesised instances which are very similar to existing samples which increases the probability of overfitting[1] the data, as explained by Hoens and Chawla [2013].

Random Under-sampling

Random under-sampling is where instances from the majority class are removed at random until some desired number of samples is reached, while at the same time preserving the minority class. Typically, under-sampling is suitable for large datasets where the potential loss of information from removing random samples is less. However, it is not uncommon for under-sampling to result in worse classification performance due to this loss in information. While more complex undersampling methods exist, e.g. sampling specific samples that lie further from the decision boundary, research by Japkowicz [2000] showed that these methods resulted in no statistically significant improvement when compared to random under-sampling.

2.2.2 Normalisation

Data normalisation is an important data pre-processing step which enforces the integrity of the data by ensuring consistency across all the values. This consistency is especially important when training deep neural networks, as significantly different values across features can cause the weights in the network to favour features with more extreme ranges. The normalisation techniques we have chosen are Min-Max, Z-Score and Decimal Point normalisation, as compared by Mustaffa and Tusof [2011]. The formula for **Min-Max normalisation** is shown in equation 2.2.1:

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}} \quad , \tag{2.2.1}$$

where,

X': the new value;

X: the current value;

[1]Overfitting is where the model fits the training data too closely which adversely affects its ability to generalise.

X_{min}: the minimum value in dataset;

X_{max}: the maximum value in dataset.

The formula for **Z-Score normalisation** is shown in equation 2.2.2:

$$X' = \frac{X - \bar{X}}{\sigma} \quad , \tag{2.2.2}$$

where,

X': the new value;

X: the current value;

$\bar{X}$: the mean of feature column associated with X;

σ: the standard deviation of feature column associated with X.

The formula for **Decimal Point normalisation** is shown in equation 2.2.3:

$$X' = \frac{X}{10^j} \quad , \tag{2.2.3}$$

where,

X': the new value;

X: the current value;

j: the smallest integer such than $\mathbf{Max}(|X'|) < 1$.

Conclusion

In this chapter we have outlined what types of credit scored exists within the dataset, and provided a technical introduction to different normalisation and resampling techniques.

Chapter 3

Artificial Neural Networks

This chapter provides the reader with background information on neural networks, focusing on the architectures and optimisation techniques used in this research.

The concept behind a neural network is formed from how neurons function within the human brain. It is built from the idea of an artificial neuron called a perceptron, which was developed in the 1950s-60s, by Rosenblatt. His work was inspired by a neurophysiologist named Warren McCulloch, and Walter Pitts, a mathematician, who published a paper which is now considered to be one of the foundational components that shaped artificial intelligence and cognitive science. They introduced the concepts of logical (threshold) neurons and neural networks which can be represented in a feedforward structure; see McCulloch and Pitts [1943].

3.1 Feedforward neural networks

Feedforward networks are primarily used for supervised[1] learning tasks, where the data is neither sequential or time-series dependent. These networks contain a large number of computational units called neurons. These neurons are arranged into a layered structure, where the first layer is called the input layer, the intermediates are known as the hidden layers, and the final is called the output layer. A network can have an arbitrary number of hidden layers; networks with many hidden layers

[1]For a given input object, X, the desired output y is known.

are called deep neural networks. In a given layer, each neuron is connected to every neuron in the next layer. These connections enable information to flow through the network in a feedforward nature.

3.1.1 Perceptrons

A neural network, in its most basic form, consists of a single neuron with no hidden layers, only an input layer, and an output layer. This is a type of supervised learning model, outlined by Rosenblatt [1958], and is referred to as a Perceptron. This type of network is used for the classification of linearly separable data, i.e. patterns that can be separated with a single hyperplane[2]. A perceptron takes as input several variables, $x_1, x_2, ..., x_m$ and produces a single output. Each input has an associated synaptic weight variable, $w_1, w_2, ..., w_m$, and bias, b, both of which are adjusted and learned during training.

Consider the formula for v:

$$v = \sum_{i=1}^{m} w_i x_i + b \quad , \tag{3.1.1}$$

the value of v is not bounded, it can be any value ranging from $-\infty$ to ∞. However, the goal of a perceptron is to correctly classify the inputs into one of two classes, l_0 or l_1. We therefore apply a decision rule, which acts as the activation function (discussed further in 3.2), in order to define the output y. An example would be:

$$y = \begin{cases} 0 & \text{if} \quad \sum_{i=1}^{m} w_i x_i + b \quad \leq \quad 0 \quad , \\ 1 & \text{if} \quad \sum_{i=1}^{m} w_i x_i + b \quad > \quad 0 \quad . \end{cases} \tag{3.1.2}$$

3.1.2 Multi-layer perceptron

A multi-layer perceptron (MLP) is a type of feedforward neural network, which contains more than one perceptron. MLPs contain an arbitrary number of hidden

[2]We refer the reader to Cevikalp [2017] for a detailed explanation of hyperplanes.

layers and are well suited to non-linear classification tasks. Funahashi [1989] showed that multi-layer perceptron networks are capable of approximating any continuous function to any accuracy, provided there be enough hidden units/layers. This type of network is trained with Stochastic Gradient Descent (SGD) using a process called back-propagation to update the gradients and biases, discussed later in section 3.4.1 & 3.4.5. We will be concerned with only this type of neural network for the rest of the thesis unless explicitly stated otherwise.

3.2 Activation functions

Activation functions are vital in neural networks; they decide if the neuron should be activated or not. The equation in 3.1.2 shows that these functions can act as decision rules where the inputs are mapped to outputs which are commonly bounded between the range -1 and 1. More formally, there exists a non-linear mapping between the inputs to an activation function and the bounded output. Without this non-linearity, multiple layers would be equivalent to a single layer within the network. A critical feature of an activation function is that it must be fully differentiable, meaning we can find the slope of the curve at any point. This is important because the value of the derivative is used as a multiplier to update the weights of the network during backpropagation while training.

There are three main problems to consider when selecting an activation function, these are:

1. The vanishing gradient problem[3];
2. Zero-centring;
3. Computational requirement.

We will now introduce three different types of activation functions.

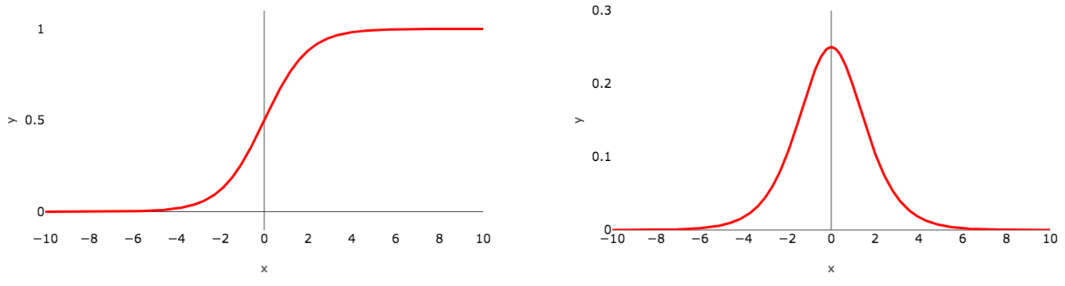
[3]Over time the neurons in the early layers have smaller and smaller gradients, meaning that they are the slowest to train.

3.2.1 Sigmoid

The sigmoid activation function is bounded between (0,1), and is defined as:

$$\sigma(x) = \frac{1}{1+e^{-x}} \quad . \tag{3.2.3}$$

The function and its derivative are shown in figure 3.1.



(a) *sigmoid* (b) *sigmoid derivative*

Figure 3.1: Sigmoid Activation Function

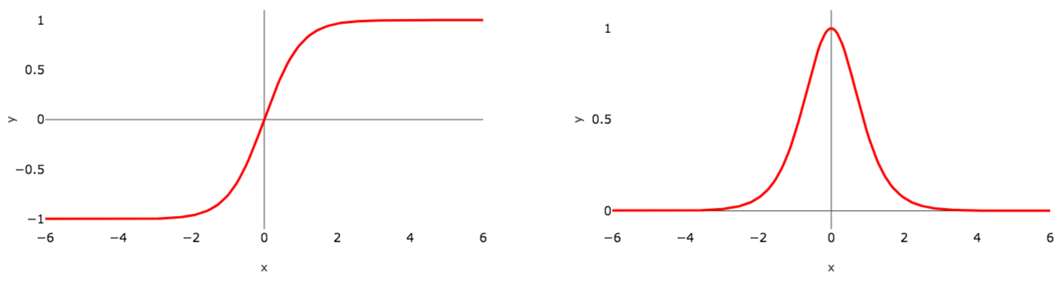
The sigmoid function suffers from the vanishing gradient problem, as explained by Zhang and Woodland [2015]. The sigmoid graph (3.1a) shows that the gradients near the asymptotes tend towards zero. This tendency is confirmed by looking at the derivative (3.1b), which approaches 0 for large negative or large positive values. The problem with this is that, as mentioned previously, the value of the derivative is used to update the network during backpropagation. We can see that the derivative of the sigmoid function is bounded between (0, 0.25]. As a consequence, the update parameters of the network will continue to get smaller and smaller until learning stops altogether. Note that neurons with tiny weights, whereby no more learning can occur, are referred to as 'saturated' or 'dead' neurons.

The sigmoid activation function is not zero-centered. This characteristic is undesirable since it is possible for neurons in later layers to receive input that is either all positive or all negative. This fluctuation can introduce 'zig-zag' dynamics, where a path through the network switches from positive to negative values between each layer during optimisation, as explained by Zhang and Woodland [2015].

The sigmoid function uses the exponential function, e^{-x}, which is computationally an expensive function in comparison to other non-linear activations e.g. ReLU.

3.2.2 Tanh

The tanh activation function is bounded between (-1,1). The function and it's derivative are shown in figure 3.2.



(a) $tanh(x)$ (b) $tanh(x)$ *derivative*

Figure 3.2: Hyperbolic Tan Activation Function

Similarly, the tanh function also suffers from the vanishing gradient problem. From the graph (3.2), it can be seen that the gradients near the asymptotes tend towards zero. This characteristic is confirmed by looking at the derivative graph (3.2b), which tends to 0 for large negative or large positive values. However, in contrast to the sigmoid function, we see that the derivative of tanh is bounded between (0, 1]. When this is combined with the fact that tanh is zero-centered, it results in the function providing stronger gradients during training. The zero-centre means that negative/positive inputs will be mapped strongly in the negative/positive direction; see Godina et al. [2017]. However, for large positive or large negative values, the output still tends towards 0, and as such is still susceptible to the vanishing gradient problem.

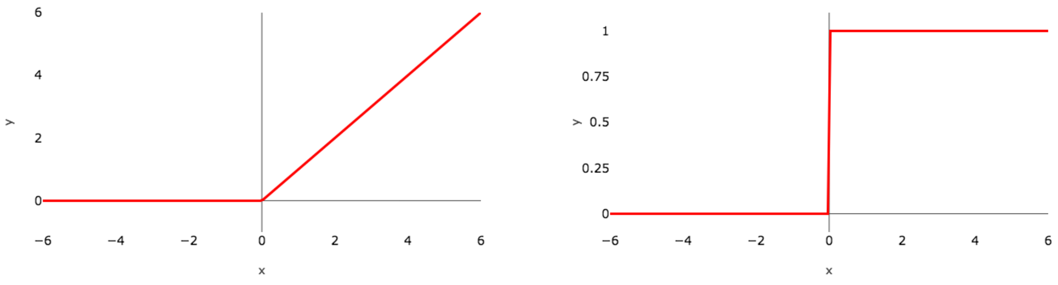
The tanh function also uses the exponential function, e^{-x}, which, as mentioned in section 3.2.1, is computationally an expensive function in comparison to other non-linear activation's e.g. ReLU.

3.2.3 ReLU

The ReLU activation function is bounded between $[0,\infty)$, and is defined as:

$$f(x) = max(0, x) \quad . \tag{3.2.4}$$

The function and its derivative are shown in figure 3.3.



(a) *relu* (b) *relu derivative*

Figure 3.3: ReLU Activation Function

ReLU is the most commonly used activation function and is the one we select to use in our network, as it does not suffer directly from the vanishing gradient problem. As can be seen in the graph (3.3a), for any values where $x < 0$ the output is 0, and for any values where $x > 0$ the output is x. This simple threshold rule makes ReLU very computationally efficient, in comparison to the tanh and sigmoid function. The derivative of ReLU (3.3b) allows the network to learn without neuron saturation, where the output is 1 for large values of x and 0 for negative values; see Zhang and Woodland [2015]. Although ReLU is not zero-centred, as seen in figure 3.3a, this problem can be overcome by regularisation techniques which are discussed in more detail in section 3.6.

3.3 Loss function

A loss or error, function $J(p, \hat{p})$ is calculated by computing the difference between the predicted output $\hat{p}$ and actual output p. The output of a loss function is used to update the weights and biases in a neural network using a process called Back-Propagation as discussed later in section 3.4.5. There exist many different loss functions which produce different results given the same values for $\hat{p}$ and p. Selecting a loss function is highly dependent on the type of machine learning problem, be it regression or classification.

3.3.1 Cross-Entropy Loss

Cross-Entropy, or log, loss is a commonly used loss function for classification problems. It is calculated using the formula:

$$CrossEntropy(p, \hat{p}) = -\sum_i p_i \log \hat{p}_i \quad , \tag{3.3.5}$$

where $\hat{p}$ is the predicted output, and p is the actual output.

The loss function imposes an exponentially increasing error as the predicted output and the actual output diverge. Similarly, as the difference between the predicted output and actual output is reduced, the output of the loss function becomes exponentially smaller.

Softmax Cross-Entropy Loss

The Softmax Cross-Entropy loss is similar to Cross-Entropy loss with the addition of a Softmax activation layer, where the output of the Softmax layer is used as input to the Cross-Entropy loss function.

$$Softmax(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \quad , \tag{3.3.6}$$

where x_i is the i^{th} output of final layer.

The Softmax layer, as shown in formula 3.3.6, is advantageous as it introduces non-linearity by producing a probability distribution whose sum is equal to 1, i.e. it outputs the probability of each class. Therefore this type of loss function is typically suited to multi-class classification problems, where each input belongs to exactly 1 class. Our classification task, as defined fully in section 5.1, fits this criteria. We therefore choose the Softmax Cross-Entropy loss function in our network.

3.4 Training neural networks

This section is a technical overview that outlines the training steps involved when training a feed forward multi-layered neural network. It also introduces a number of specific optimisation techniques that we implement in the final model. The mathematical formulae in this section are based on the explanations from the book Neural Networks and Deep Learning, Nielsen [2018].

3.4.1 Gradient Descent

Gradient descent, or steepest descent, is a type of algorithm used to find the minimum of a function. In the case of a neural network, gradient descent is used to minimise the loss function, the difference between the predicted output $\hat{p}$ and actual output p. This objective is achieved by iteratively updating the weights and biases by minimising the gradient of the loss function with respect to both the weights and biases. We can define the general update rule for a given weight as:

$$w_{kj} = w_{kj} - \eta \Delta w_{kj} \quad , \tag{3.4.7}$$

where w_{kj} is the weight associated with the j^{th} neuron in layer l and k^{th} neuron in layer $l+1$, and where η is the learning rate. Unfortunately computing Δw_{kj} is not trivial and is discussed in more depth in Section 3.4.5.

3.4.2 Mini-Batch Gradient Descent

In order to find the exact gradient of the loss function with respect to the weights and biases, it is necessary to calculate the sum of gradient over all data points in the training set. Operating large datasets, as we are in this thesis (see section 4.1), means that this calculation is computationally expensive and would take an unfeasible amount of time to train. We adopt Mini-Batch Gradient Descent, which calculates an approximation of the gradient based on a subset (mini-batch) of the dataset. The equation is defined as follows:

$$\Delta w_{kj} = \frac{1}{b} \sum_{i=0}^{b} \Delta w_{kj} \quad , \tag{3.4.8}$$

where b is the size of the mini-batch, and Δw_{kj} is the average gradient over all b samples. During training, the algorithm will iterate over the dataset a specified number of times, where a single cycle of the entire dataset is known as 1 epoch. Note that it is important to randomly shuffle the dataset after each epoch to prevent repeating update cycles.

3.4.3 Learning rate

The learning rate η is used to control how much the weights should be adjusted with respect to the gradient of the loss function. Small learning rates will take longer to minimise the loss function and have a higher probability of converging in a local minimum. Conversely, high learning rates have a higher probability of not converging or even diverging.

Adaptive learning rates, where the rate reduces over time, are an approach to mitigate these problems. We define an adaptive learning rate using the following formula:

$$\eta = \frac{\eta}{1 + kt} \quad , \tag{3.4.9}$$

where t is the number of epochs, and k is an arbitrary constant.

3.4.4 Momentum

Momentum is a technique we adopt in this work. It is used to help prevent the network getting stuck in a local minimum. With momentum m, the weight update rule is defined as:

$$\Delta w_{kj}^{t} = \eta \Delta w_{kj}^{t} + m \Delta w_{kj}^{t-1} \quad , \tag{3.4.10}$$

where m is a parameter between 0 and 1 (determined by trial and error), and t is the step number for gradient descent method. Momentum uses a fraction of the previous weight gradient, Δw_{kj}^{t-1}, to determine the current gradient value. This calculation means that when the gradient is pointing in the same direction, the update factor will be larger, gaining momentum. It is therefore necessary to reduce the learning rate η significantly when using a momentum value close to 1.

3.4.5 Backpropagation

A feedforward neural network learns in two stages, the forward pass and backwards pass. The forward pass refers to the calculation process through all neurons from the first to the last layer, producing the predicted output $\hat{p}$. The backward pass refers to a calculation process through all neurons from the last to the first layer, where the weights and biases are updated based on how different the predicted output $\hat{p}$ is from the actual output p.

The backpropagation algorithm is a special case of automatic differentiation and is the most common method for updating neural networks in the backward pass. The objective of the algorithm is to update the weights and biases in each layer by computing the partial derivative of the cost function J with respect to each weight w and bias b in a neural network. In a multi-layer perceptron, backpropagation allows the error to propagate back through the network layer by layer, updating the weights and biases accordingly.

This section will evaluate how the weights are updated using the Mean Squared Error (MSE) loss function, J.

First consider a multi-layer perceptron network with l layers, where $\boldsymbol{a^l}$ is the output vector of the activation functions at layer l, $\boldsymbol{b^l}$ is the biases for the neurons in layer l, and $\boldsymbol{W^l}$ is the weight vector associated with layers l and $l+1$.

For the forward pass of the network we have:

$$\boldsymbol{a^l} = \sigma^l(\boldsymbol{W^l} \cdot \boldsymbol{a^{l-1}} + \boldsymbol{b^l}) \tag{3.4.11}$$

where σ^l represents an activation function at layer l.

We denote z_k^l to represent the input sum to the activation function of the k^{th}

neuron in layer l:

$$z_k^l = \sum_j w_{kj}^l a_j^{l-1} + b_k^l \tag{3.4.12}$$

where w_{jk}^l is the weight associated with the j^{th} neuron in layer l and k^{th} neuron in layer $l+1$, and b_k^l is the bias associated with the k^{th} neuron in layer l.

We rewrite the forward pass as:

$$a_k^l = \sigma^l(z_k^l) \tag{3.4.13}$$

where σ^l represents an activation function at layer l.

Consider a loss, or error function J. The error function in classic backpropagation is Mean Squared Error (MSE), which we denote as:

$$J = \sum_{j=0}^{n-1} (a_j^l - y_j)^2 \quad . \tag{3.4.14}$$

We denote the derivative of the loss function J with respect to the weights as:

$$\frac{\partial J}{\partial w_{kj}^l} = \frac{\partial J}{\partial z_k^l}\frac{\partial z_k^l}{\partial w_{kj}^l} = \frac{\partial J}{\partial a_k^l}\frac{\partial a_k^l}{\partial z_k^l}\frac{\partial z_k^l}{\partial w_{kj}^l} = \Delta w_{kj} \quad . \tag{3.4.15}$$

We consider the first term $\frac{\partial J}{\partial a_k^l}$ from 3.4.15, using the equation for J defined in 3.4.14:

$$\begin{aligned}
\frac{\partial J}{\partial a_j^l} &= \frac{\partial}{\partial a_j^l}\left(\sum_{j=0}^{n-1}(a_j^l - y_j)^2\right) \\
&= \frac{\partial}{\partial a_j^l}\left((a_0^l - y_0)^2 \ + \ \ldots \ + \ (a_j^l - y_j)^2 \ + \ \ldots\right) \\
&= \frac{\partial}{\partial a_j^l}\left((a_0^l - y_0)^2\right) \ + \ \ldots \ + \ \frac{\partial}{\partial a_j^l}\left((a_j^l - y_j)^2\right) \ + \ \ldots \\
&= 2(a_j^l - y_j) \quad .
\end{aligned} \tag{3.4.16}$$

We consider the second term $\frac{\partial a_k^l}{\partial z_k^l}$ from 3.4.15, using the equation for a_k^l defined in 3.4.13:

$$\begin{aligned}\frac{\partial a_j^l}{\partial z_j^l} &= \frac{\partial}{\partial z_j^l}\left(\sigma^l(z_j^l)\right)\\ &= \sigma^{l\,\prime}\left(z_j^l\right) \quad .\end{aligned} \tag{3.4.17}$$

We consider the third term $\frac{\partial z_k^l}{\partial w_{kj}^l}$ from 3.4.15, using the equation for z_k^l defined in 3.4.12:

$$\begin{aligned}\frac{\partial z_k^l}{\partial w_{kj}^l} &= \frac{\partial}{\partial w_{kj}^l}\left(\sum_i w_{kj}^l a_j^{l-1} + b_k^l\right)\\ &= \frac{\partial}{\partial w_{kj}^l}\left((w_{k0}^l a_0^{l-1} + b_k^l) \ + \ \dots \ + \ (w_{kj}^l a_j^{l-1} + b_k^l) \ + \ \dots\right)\\ &= \frac{\partial}{\partial w_{kj}^l}(w_{k0}^l a_0^{l-1} + b_k^l) \ + \ \dots \ + \ \frac{\partial}{\partial w_{kj}^l}(w_{kj}^l a_j^{l-1} + b_k^l) \ + \ \dots\\ &= a_j^{l-1} \quad .\end{aligned} \tag{3.4.18}$$

We have now evaluated all three terms from 3.4.15, we denote the gradient with respect to the weights for the MSE loss function to be:

$$\begin{aligned}\Delta w_{kj} = \frac{\partial J}{\partial w_{kj}^l} &= \frac{\partial J}{\partial a_k^l}\frac{\partial a_k^l}{\partial z_k^l}\frac{\partial z_k^l}{\partial w_{kj}^l}\\ &= \left(2(a_j^l - y_j)\right) \ \left(\sigma'^{\,l}\,(z_j^l)\right) \ \left(a_j^{l-1}\right) \quad .\end{aligned} \tag{3.4.19}$$

We can use the formula for Δw_{kj} in equation 3.4.19 to update the weights as described in Section 3.4.1. The update rule under the MSE loss function would be:

$$\begin{aligned}w_{kj} &= w_{kj} - \eta\Delta w_{kj}\\ &= w_{kj} - \eta\left(2(a_j^l - y_j)\right) \ \left(\sigma'^{\,l}\,(z_j^l)\right) \ \left(a_j^{l-1}\right) \quad .\end{aligned} \tag{3.4.20}$$

3.4.6 Weight Initialisation

Weight initialisation is an important consideration when training a deep multi-layered neural network. The initial weight values can significantly affect both the speed and optimality of convergence during stochastic gradient descent. That is, if the weight values are small, the output of the loss function will also be small, resulting in small updates to the network. In turn, reducing the speed of convergence and increases the probability of converging in a sub-optimal minimum.

The network in this paper uses the Xavier initialisation algorithm, which is a popular weight initialisation technique. The objective of the algorithms is to produce weight values that follow a distribution (typically Gaussian or uniform), with zero mean and a specified variance, across all the neurons in the layer. We refer the reader to the original paper by Xavier Glorot, 2010 for more implementation details Glorot and Bengio [2010].

3.5 Layers

3.5.1 Batch Normalisation

Batch Normalisation is a common technique for improving stability and the performance of a neural network. Since it's introduction by Ioffe and Szegedy [2015], it has quickly become a standard optimisation technique when training deep neural network. It works by standardising the input layer, x, by subtracting the mean and dividing by the standard deviation. The standardised layer, z, is multiplied by an arbitrary parameter g, and an arbitrary parameter b is added to the resulting product.

The batch normalisation process means that the weights within the network do not become imbalanced with extremely high or low values since the standardisation is included in the SGD process. The learned parameters g, b mean that the *Covariate Shift*[4] between the layers is reduced. The backpropagation algorithm then allows the network to learn by how much each layers' activation should be normalised.

The addition of batch normalisation can significantly increase the speed at which training occurs, and reduce the probability that a particular batch cycle will over influence the training process. It allows for the use of higher learning rates as the

[4]The change in the distribution of network activation's due to a change in network parameters

output values do not deviate significantly from the mean, reducing the likelihood of an exploding or vanishing gradient problem as well as getting stuck in a sub-optimal minimum.

3.6 Regularisation

Regularisation is a very important technique in deep learning to prevent overfitting[5]. In general, regularisation reduces high valued weights in the network, reducing the flexibility of the model and making it less likely to fit noise within the training data. There exists a variety of regularisation strategies, each with their advantages and drawbacks; this section will give an overview of some of these techniques. We inform the reader that these techniques are analysed in further detail and compared with batch normalisation in the results section (6.5).

3.6.1 Dropout

Dropout is a type of regularisation technique that was first introduced by Hinton et al. [2012]. The literature explains that dropout is where "each hidden unit is randomly omitted from the network with a probability." It is later stated that dropout can be used to reduce overfitting and can be viewed as a very efficient way of performing model averaging within the network.

3.6.2 L^2 Regularisation

L^2 Regularisation, or weight decay, is a regularisation strategy that moves the weights closer to the origin[6] by adding a regularisation term. Slatton [2014] showed that large weights are more often found in networks where overfitting occurs, L^2 regularisation attempts to mitigate this by keeping the weights small.

[5]A problem where the classifier learns features that are specific to the training data and therefore performs badly on unseen validation data.

[6]The weights could be regularised towards any point in space and still achieve a regularising effect.

L^2 regularisation is achieved by modifying the loss function and adding a weight penalty, which causes larger weights within the network to have a higher loss. The equation for L^2 regularisation is the following:

$$J = J_0 + \frac{\lambda}{2n} \sum_l \boldsymbol{W}^{[l]^2} \quad , \tag{3.6.21}$$

where J_0 is the unregularised cost function, n is the size of the training set and $\boldsymbol{W}^{[l]}$ is the weight vector between layers l and $l + 1$. Note that a typical value for regularisation term, λ, falls within the range 0.01 to 0.05; as stated by Slatton [2014].

Conclusion

In this chapter we provided a detailed introduction to multi-layered feedforward neural networks. We introduced different types of activation functions, explaining both the advantages and drawbacks. We provided a technical explanation of back-propagation, the algorithm by which neural networks update their weights and biases relative to the chosen loss function. We discussed the importance of weight initialisation, its effect on the speed and optimality of convergence during stochastic gradient descent (SGD), and introduced the Xavier initialisation approach. We then explained how Batch Normalisation can increase the speed of convergence and reduce the *Covariate Shift* during SGD. Finally, we introduced two Regularisation techniques, Dropout and L^2 Regularisation.

Chapter 4

Data

In this chapter, we introduce the data used in this work and discuss methods by which we processed and engineered new features.

We used the Freddie Mac Single-Family dataset which contains approximately 95 million mortgages, with over 2 billion monthly observations covering approximately 60% of all mortgages originated in the U.S. from January 1999 to January 2017. The loans are fully amortising 15, 20, and 30-year fixed-rate mortgages. The data itself is divided into origination records and performance records. We will refer to these separately, as 'Origination' features (described in Appendix A.1), and 'Performance' features (described in Appendix A.2). A summary of key origination features and loan length can be seen in table 4.1.

Loan Feature	**Mean**	**Median**	**Min**	**Max**
Original Unpaid Balance ($)	191,114	148,388	9	5,400,000
Interest Rate	5.6	5.7	0	19.4
Loan-To-Value Ratio	75	78	0	203
FICO Credit Score	727	743.5	300	850
Loan Length	21.1	20	0	140

Table 4.1: Summary statistics for loan length and a subset of Origination features

4.1 Data Format

The dataset is split into two sections, 'Origination' features and 'Performance' features. Individually, these records are separated into files and folders on a per quarterly basis. Each quarter contains the origination data for all mortgages that began during that period, as well as the corresponding performance updates over the lifetime of the mortgages. Each file is in CSV format, and range in size from 100MBs to 1.6GBs.

4.2 Data Labelling

A loan can be in status: Current, 30, 60, 90+ days delinquent, Foreclosed[1], REO (Real-Estate Owned) or Fully Paid, which are defined in more detail in table 4.2. To determine a loan's status, we use the data columns; 'Delinquency Status', 'Zero Balance Code' and 'Repurchase Flag'. The exact rules we used can be seen in Appendix A.4, along with the Zero Balance Code definitions in Appendix A.3.

[1]It's is not uncommon for a vendor to move a mortgage into a default/foreclosed status to provoke repayment. After repayment, the loan transitions from default back to current.

Loan Status	Description
Current (0)	Loan is up to date on all outstanding payments, or within grace period.
30 days delinquent (1)	Loan has not been current for 30 to 60 days.
60 days delinquent (2)	Loan has not been current for 60 to 90 days.
90+ days delinquent (3)	Loan has not been current for 90+ days.
Foreclosed (4)	Loan has been foreclosed because payments have stopped. Property is being taken back by mortgage lender.
REO (Real-Estate Owned) (5)	Loan has been foreclosed because payments have stopped. Property has been taken back by mortgage lender.
Fully Paid (6)	Loan has been fully repaid, either at the expiration, or as a result of a prepayment.

Table 4.2: Loan Status Descriptions

4.2.1 Label: Default or Fully Paid

To determine the binary ’Default’ or ’Fully Paid’ label, we identified the final status of each loan from the last performance update. There exist three possible final statuses, Foreclosed, REO (Real-Estate Owned) or Fully Paid. Once we have identified the final status of the loan, we reduce the status to either Fully Paid or Default. We classify loans in the Fully Paid status as Fully Paid, and loans in Foreclosed or REO status as Default. Finally, we reject all loans without a final terminating status. As many of the loans are still active, and therefore do not have a terminating status, this reduces the dataset to approximately a sixth of its original size. The class distribution between Default and Fully Paid loans can be seen in table 4.3.

Metric	Default	Fully Paid
Total Number	84,745	15,214,584
Ratio	0.006	0.994

Table 4.3: Class Distribution between Default and Fully Paid

4.3 Data Cleaning

There exists a small percentage of loans in the dataset with missing data values. Two possible reasons are either a reporting error by the original mortgage vendor or incomplete information provided by the borrower during the mortgage application. We, therefore, removed any entries in the origination data where the features FICO score, original balance, or original interest rate were missing. These features are required by a vendor when taking out the loan, and therefore we deduce that the absence of this data must be due to a reporting error. From the performance updates, we remove all loans which are missing the features Current Unpaid Balance, Delinquency Status, or Current Interest rate.

There exist additional features in the dataset that would not be required by all vendors at origination, for example, debt-to-income ratio. In these instances, we add a feature which indicates whether the data was missing or not (1 missing, 0 not missing). We then impute the missing value using the median. For features with categorical variables, we forgo the additional feature indicator and simply replace the value with a 'U' for Unknown.

After analysing the performance updates, we observed that many of the defined features are only relevant at the end of the mortgage, e.g. net sale proceeds, expenses, zero balance flag. We therefore removed all features from the performance updates that do not provide additional information during the active lifetime of the loan. We observed that a number of loans do not have a final state as a consequence of a reporting error. In these cases, they are removed. We also removed all loans that were ongoing, i.e. they still had an Unpaid balance in Q1 2017, and were not Foreclosed or REO. After this filtering process, we were left with approximately 15.3

million unique loans from which we had 326 million performance updates.

For each categorical feature in the dataset, we defined a set of allowed values. For instances that contained values not defined in the set, we assigned a 'U' for Unknown. Similarly, for each numerical feature in the dataset, we defined an allowed value range. For instances that fell outside of the defined range, we added a feature indicating whether the data was missing or not (1 missing, 0 not missing), and impute the missing value using the median of the feature.

4.4 Feature Engineering

4.4.1 Origination Features

We observed that the origination feature set supplied by Freddy Mac (see appendix A.1) contained a number of well researched and common mortgage indicators. For example, Danis and Pennington-Cross [2005] found that higher FICO scores resulted in higher prepayment rates. We found that lower FICO scores resulted in an increased likelihood of Default, with the opposite being true for higher scores. This is shown in density plot in figure 4.1.

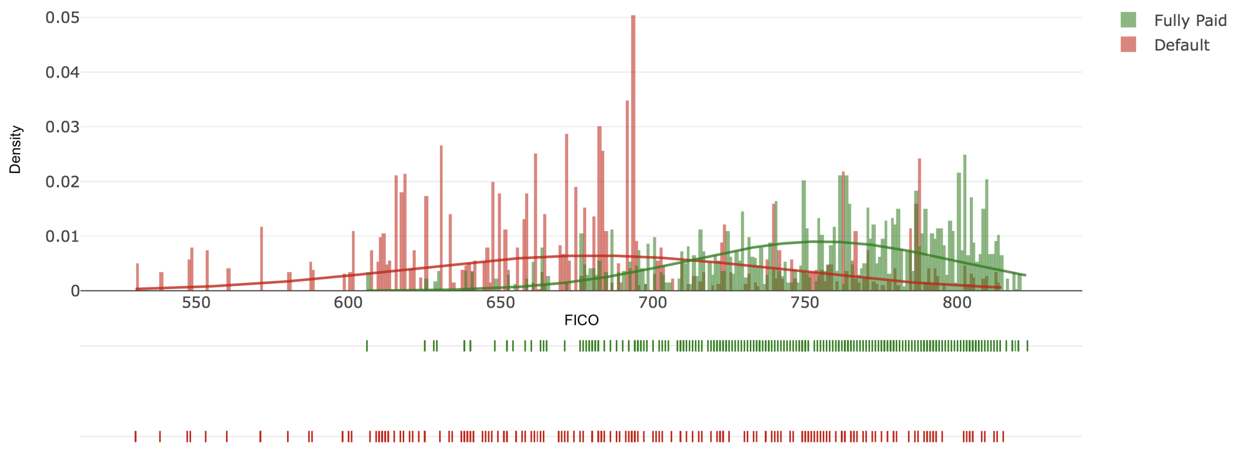


Figure 4.1: Density Plot with Normal Distribution and Rug plot - A comparison between Fully Paid and Default loans using the feature FICO Score

Another example is the feature Loan-to-Value (LTV) ratio. Ambrose and Capone [1998] found that higher Loan-to-Value ratios had a higher foreclosure probability.

Our findings seem to support this observation. The density plot in figure 4.2 shows a small correlation between higher LTV ratios and the probability of default.

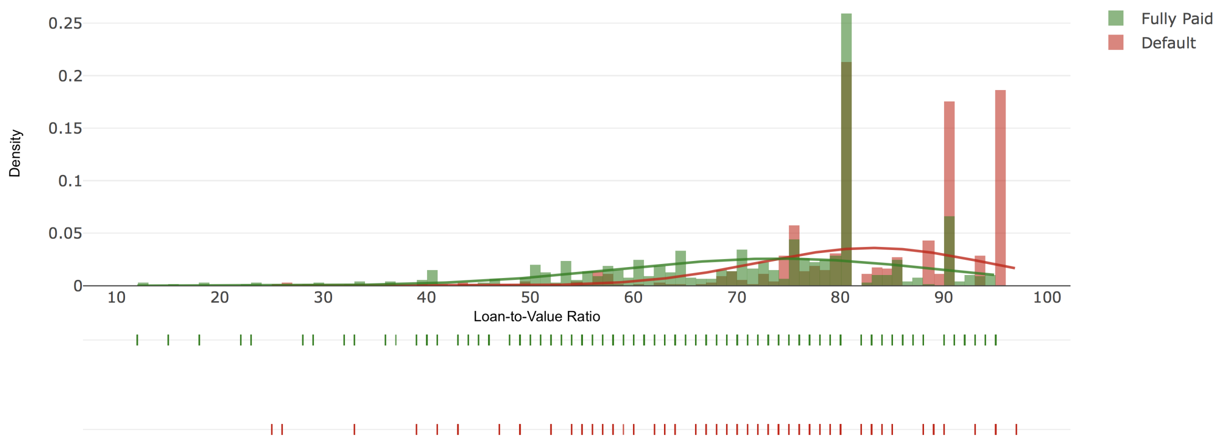


Figure 4.2: Density Plot with Normal Distribution and Rug plot - A comparison between Fully Paid and Default loans using the feature Loan-to-Value Ratio

We observed that in both figure 4.2 and figure 4.1 there was a moderate level of separability. The normal distribution curves for 'Fully Paid' and Default' overlap significantly on both graphs; this shows that the data is not linearly separable using only these features. This is confirmed by the distribution of the rug plots which show that the spread of data is large across the two categories ('Fully Paid' and 'Default') on both graphs. This observation suggests that the machine learning methods (such as neural networks) should out-perform linear regression models, as they are able to find non-linear relationships in the origination features to perform the classification task.

4.4.2 Performance Features

As mentioned in Section 4.3, many of the features defined in the performance updates are only relevant at the end of the mortgage. We, therefore, removed these features from the performance vector. However, the frequency and completeness of the performance updates that are relevant allowed us to create new features. Specifically, we captured historical trends for each loan, obtained from performance updates that preceded the current month. For example, we calculated the number of times a particular loan had been 30 days delinquent.

Previous work has validated the predictive performance of these indicators; Danis and Pennington-Cross showed that loans with more frequent delinquency periods, when combined with higher FICO (Credit) scores, were less likely to foreclosure. The method we used to track delinquency periods is as follows. We calculated the occurrence of each **Status**[2] for a particular loan, by iterating over all previous performance updates for that loan. This produces multiple features per loan that act as status counters, i.e. every time a particular loan enters a state, the counter will be incremented.

Additionally, the work by Chana et al. [2014] and Sirignano et al. [2016] found that using indicators that capture short-term trends are beneficial when predicting foreclosure rate. We therefore created another set of features which only considered the occurrence of each **Status**[2] over the last 12-months.

A full list of the performance update features can be seen in table 4.4.

[2]The status of a loan is defined in this work as either current, 30, 60, 90+ days delinquent, Foreclosed, REO (Real-Estate Owned) or Fully Paid.

Loan Feature	**Values**
Current Status	**Status**
Current Unpaid Principal Balance	Continuous
Current Interest Rate	Continuous
Loan Age	Continuous
Months Remaining	Continuous
Percentage change between Last Balance & Current Balance	Continuous
Occurrences of **Status** [3]	Continuous
Occurrences of **Status** in the last 12 months [3]	0-12
Interest Rate - National Interest Mortgage Rate [3]	Continuous
Number of Months that Mortgage Interest Rate < National Interest Rate [3]	Continuous

Status = {Current, 30, 60, 90+ Days Delinquent, Foreclosed[0], REO or Fully Paid,}

Table 4.4: Performance Features

For clarity, we explain a subset of features from table 4.4 using a language description:

- **Occurrences of 30-dd** represents the number of times the current loan has entered the 30 days delinquent status up to the current time.
- **Occurrences of 30-dd in the last 12 months** represents the number of times the current loan has entered the 30 days delinquent status in the last 12 months.

[3]Feature concept taken from Sirignano et al. [2016]

Evaluation of 12-month performance indicator

Figure 4.3 and 4.4 show the comparison between Fully Paid and Default loans using the feature 'Occurrences of 30-dd' and 'Occurrences of 30-dd in the last 12 months' respectively. The chart in 4.3 shows that there are more Fully Paid than Default loans that have fewer occurrences of 30-day delinquency when considering the full lifetime of a loan.

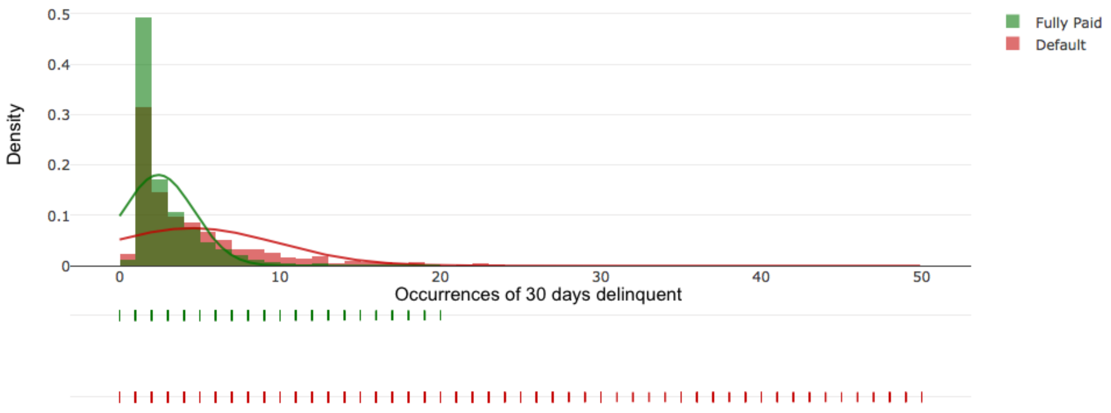


Figure 4.3: A comparison between Fully Paid and Default loans using the feature 'Occurrences of 30-dd'.

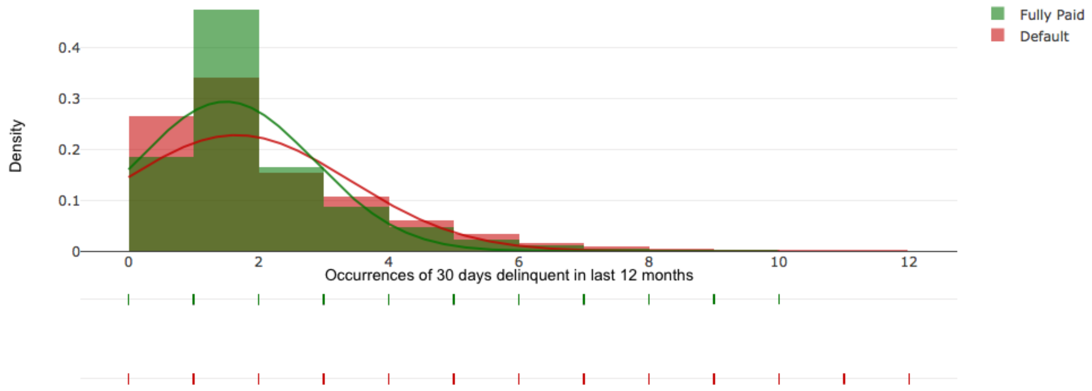


Figure 4.4: A comparison between Fully Paid and Default loans using the feature 'Occurrences of 30-dd in the last 12 months'.

If we compare figure 4.3 and 4.4, we find that the relationship is less apparent in 4.4. This suggests that loan performance indicators which are based on only recent loan history, i.e. the last 12 months, are less powerful and relevant to the prediction.

Nevertheless, we use both types of indicators as features in our data as there may exist non-linear relationships that are more difficult to identify.

4.4.3 Economic Features

The magnitude of the dataset allows us to obtain novel features based on location (per State and per Zipcode). A full list of the Economic features can be seen in table 4.5.

Table 4.5: Economic Performance Features

Loan Feature	**Values**
Monthly National Mortgage Interest Rate [4]	Continuous
Monthly Housing Price Index per State	Continuous
Monthly Unemployment Rate per State [4]	Continuous
Number of Loans (active) per State	Continuous
Number of Loans (active) per Zipcode	Continuous
Number of Loans (taken out) per State	Continuous
Number of Loans (taken out) per Zipcode [4]	Continuous
Number of Loans (taken out) per State in the last 12 months	1-12
Number of Loans (taken out) per Zipcode in the last 12 months	1-12
Default Rate per State	Continuous
Default Rate per Zipcode	Continuous
Default Rate per State in the last 12 months	Continuous
Default Rate per Zipcode in the last 12 months [4]	Continuous
Occurrences of Paid Off & Default per State	Continuous
Occurrences of Paid Off & Default per Zipcode	Continuous
Occurrences of Paid Off & Default per State in the last 12 months	1-12
Occurrences of Paid Off & Default per Zipcode in the last 12 months	1-12

[4]Feature concept taken from Sirignano et al. [2016]

As discussed in section 1.2, the work by Chana et al. [2014] and Sirignano et al. [2016] found that using indicators that capture short-term economic trends are beneficial when predicting foreclosure rate. They also found an association with neighbouring characteristics and higher default rates. Building on these findings, we produced several location-based features that capture both short and long-term trends. We were able to calculate the total number of loans, total number of Paid Off loans, and total number of Default loans, on a per state and zip code basis across each month (the complexity of this is discussed in Section 4.5.3). Using this data, we calculated the Default rate on a per state and zip code basis across each month. We also created an additional feature set which only considered the last 12 months of history, similar to section 4.4.2.

For clarity, we explain a subset of features from table 4.5 using a language description:

- **Occurrences of Default per State** represents the number of loans that have Defaulted in the same state as the current loan up to the current time.
- **Occurrences of Paid Off per Zipcode in the last 12 months** represents the number of loans that have Fully Paid in the same zipcode as the current loan over the last 12 months.

From figure 4.6 and 4.5, we observe higher levels of separability between 'Default' and 'Fully Paid' when comparing the distribution on feature 'Occurrences of Default per State in the last 12 months' (figure 4.5) to 'Occurrences of Default per State' (figure 4.6). The graph that considers only the last 12 months indicates that the majority of Fully Paid loans occur in states where the Occurrence of Default is lower. We concluded that geographically based features which consider only recent history, i.e. the last 12 months, can be more powerful and relevant to the prediction than features that capture the entire historical period (from 1999 up to the current time of the loan). This difers from our findings in section 4.4.2 where we state that loan performance indicators which consider only recent history, i.e. the last 12 months, seems to be less powerful when compared to looking at the entire duration of the loan.

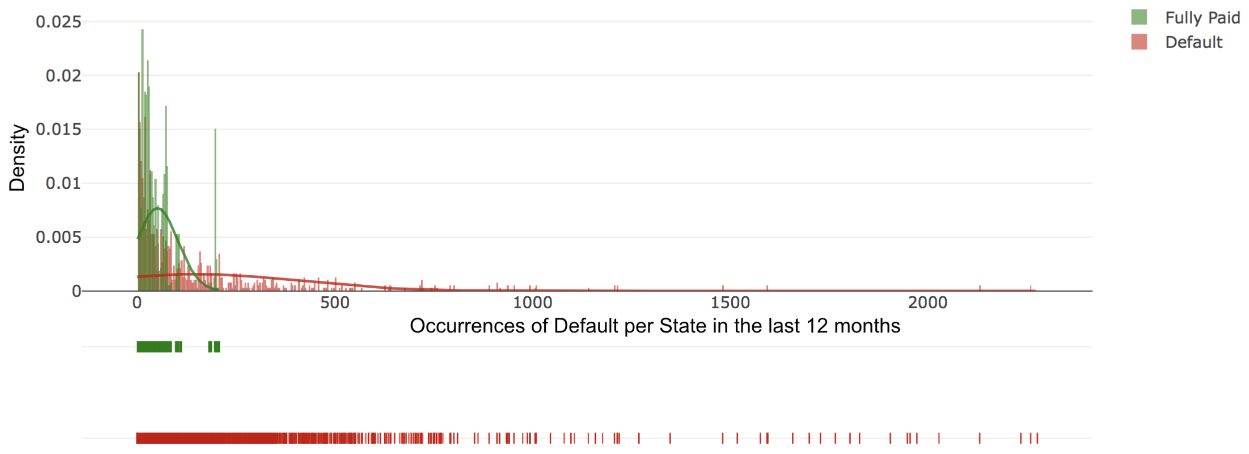


Figure 4.5: Density Plot with Normal Distribution and Rug plot - A comparison between Fully Paid and Default loans using the feature 'Occurrences of Default per State in the last 12 months'

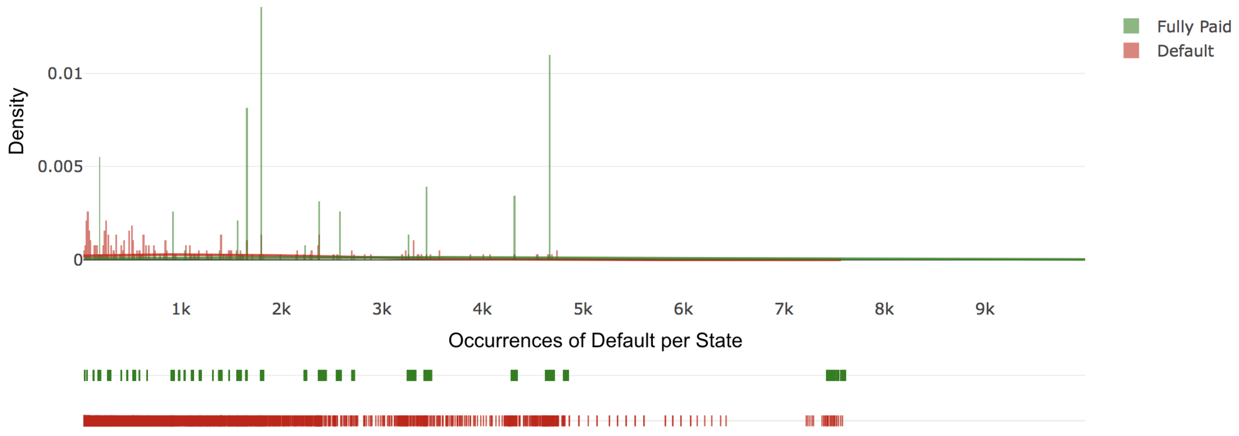


Figure 4.6: Density Plot with Normal Distribution and Rug plot - A comparison between Fully Paid and Default loans using the feature 'Occurrences of Default per State'

Additionally, the work by Chana et al. [2014] and Sirignano et al. [2016] showed a correlation between economic information such as housing market conditions and the rate of foreclosure. We, therefore, introduced three new datasets; Monthly Housing Price Index per State, Monthly Unemployment Rate per State, and National Monthly Interests rates, all acquired from the US Bureau of Labor Statistics [50]. A comparison of the feature Housing Price Index can be seen in figure 4.7, where we observe significant levels of separability between 'Default' and 'Fully Paid'. We, therefore, concluded that we find a correlation between Housing Price Index and the occurrence of a default.

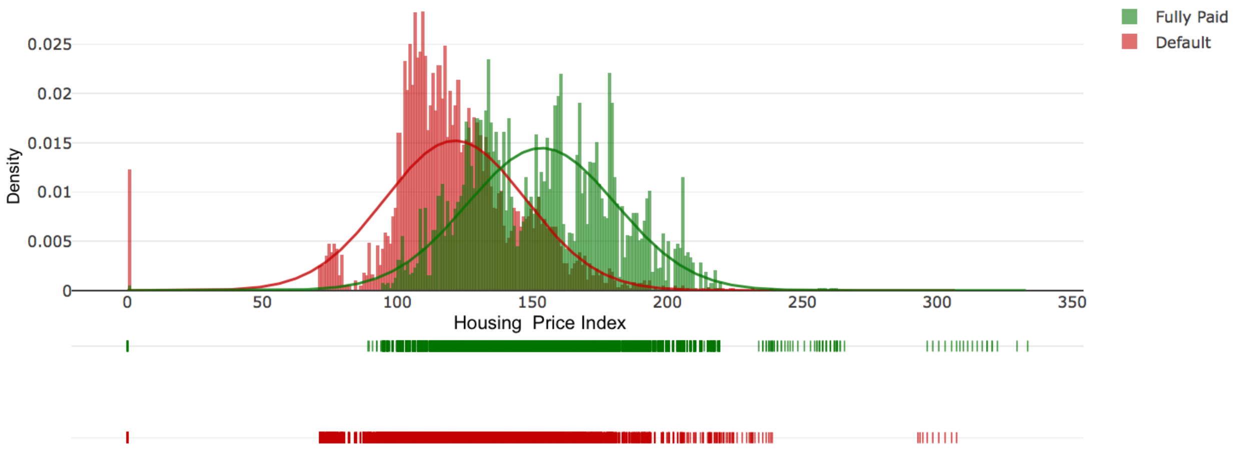


Figure 4.7: Density Plot with Normal Distribution and Rug plot - A comparison between Fully Paid and Default loans using the feature Housing Price Index

4.5 Data Processing

The dataset in its raw form was over 150GBs. As a consequence, we required significant computation processing capacity in order to manipulate the data and train the model. We used the University of Bristol's Super-Computer, Blue Crystal 4, utilising the Computational Processing Units (CPUs) for data manipulation as discussed in Section 4.5.3, and the Graphical Processing Units (GPUs) for training the model as discussed in Section 4.5.2. To access Blue Crystal 4, we were required to log-in remotely to a Linux Server using Secure Shell (SSH[5]). This provided access to a 'login' node from which we could submit code to be run on the CPUs and GPUs, using an open-source cluster management and job scheduling system called Slurm.

4.5.1 Implementation

We chose to use the programming language Python for the implementation of this work. Python has numerous well documented libraries for large-scale data manipulation, namely Numpy [39] and Pandas [40], as well as a comprehensive library for building and training neural networks, namely Tensorflow [49]. The use of these libraries meant that it was not necessary to implement the low-level code needed to train a neural network, saving time and eliminating unnecessary complexity. The information in table 4.6 gives an overview of the main Python libraries that were used in this work.

[5]The SSH protocol is a method for secure remote login from one computer to another

Table 4.6: Overview of Python Libraries and Use-Cases

Python Library	**Version**	**Use-Case**
Tensorflow [49]	1.2	Training neural network.
Tensorboard [48]	1.6.0	Visualisation of training & results of neural network
CUDA [11]	8.0	Interface with Nvidia GPU
Pandas [40]	0.22.0	Data manipulation
Numpy [39]	1.14.1	Data manipulation
Multitasking [36]	0.0.7	Asynchronous loading

4.5.2 Loading

The dataset was split over multiple CSV (Comma Separated Values) files. For each year, the data was split into four quarters, each containing two files. To manipulate the data we used Pandas Data Frames in Python. In order to speed up the file I/O (Input/Output), we used the HDF5 [6] file format with the main benefit being the application of data chunking. This meant that the data could be stored in a non-continuous block of memory, which allowed us to read/write Pandas Data Frames directly to secondary memory much more efficiently. In the worst case, this reduced the load time for one file from 35s to 11s, and in the best case from 169s to 39s. When applied to all files, this provided significant speed benefits. One drawback to using the HDF5 file format is that the files sizes increase significantly and required, on average, five times more space to store the same data.

Nevertheless, the speed improvement was especially useful when training the neural network as it contributed to the seamless transition between batches across multiple files. To achieve this transition, we created an asynchronous loading pattern. The pattern works as follows: We load and process the current file, the network begins training on the new data, at which point we asynchronously requested the

[6]a file format designed to store and organise large datasets in hierarchical structures

next file and processed the data ready for the network. This processing technique allowed us to perform manipulation such as data resampling before the neural network has finished iterating over the previous data. Although it would have been possible to resample the entire dataset before training, by writing the resampled data to file, the asynchronous approach allowed for a much faster and friction-less workflow. Note that, for training and testing the models, we used a Nvidia Pascal P100 GPU.

4.5.3 Manipulation

Performing operations on data of this scale turned out to be very challenging. The main issue was that to perform operations on the data in any reasonable amount of time; it was necessary to use group-wise operations using functions from Python Pandas Library. This restriction meant that the task of calculating the cumulative totals, i.e. the number of defaulted loans per state, as mentioned in Section 4.4.3, across the entire dataset was especially challenging.

To process the data, we initially used a 14 core 2.4 GHz Intel E5-2680 v4 (Broadwell) CPU with 128 GiB of RAM. However, we encountered some memory issues when manipulating the dataset. A python 'MemoryError' was the most common, indicating that the virtual address space limit was reached. This issue subsided after switching the same CPU chip with 512 GiB of RAM. To perform the operations mentioned in Section 4.4 took 5 days. Further, we used only single precision floating point values, and although this does not affect the output of the model, it approximately halves the memory requirement which resulted in a computational speed up.

4.5.4 Model Preparation

We used a one hot encoding format to represent all categorical data. This format allowed categorical variables to be represented using binary vectors. For example, the feature 'st' is the state category where each state is represented using a two-letter string ('NY', 'WT'... etc.). In this case, we created a new feature for each state ('st_NY', 'st_WT'... etc.), where either a 1 or 0 is used to represent the presence of that state.

It is important to randomly shuffle the data before passing the information to the model. We used a native Python random shuffle function, using a date-stamp (HH + MM + SS), as the random seed. Although not cryptography secure, this source of randomness is simple, quick and frequently changes, which is more than suitable for the intended purpose.

4.6 Formal Description

We use three vectors to construct the dataset for the final model, the origination vector (appendix A.1), performance vector (table 4.4), and economic performance vector (table 4.5). The following description is used to formalise the structure of the data.

We adopt a discrete time interval for the periods $t = 0, 1, 2, ..., months$. We define:

- **O** to be the Origination vector of length p;
- $\mathbf{M}(t)$ to be the Performance vector, at time t, of length q;
- $\mathbf{E}(t)$ to be the Economic performance vector, at time t, of length r;

i.e.

$$\mathbf{O} = \begin{bmatrix} O_0 \\ O_1 \\ \vdots \\ O_p \end{bmatrix} \quad , \quad \mathbf{M}(t) = \begin{bmatrix} O_0(t) \\ O_1(t) \\ \vdots \\ O_q(t) \end{bmatrix} \quad \text{and,} \quad \mathbf{E}(t) = \begin{bmatrix} E_0(t) \\ E_1(t) \\ \vdots \\ E_r(t) \end{bmatrix} \quad . \tag{4.6.1}$$

Let $\mathbf{X}(t)$ be a vector of length $p + q + r + 3$ consisting of elements $X_i(t)$ for $i \,\epsilon\, \{0, 1, ..., p + q + r + 2\}$, and

where:

$$X_i(t) = \begin{cases} O_i & \forall \quad i \,\epsilon\, [0, p], \\ M_{i-(p+1)}\,(t) & \forall \quad i \,\epsilon\, [p + 1, p + q + 1], \\ E_{i-(p+q+2)}\,(t) & \forall \quad i \,\epsilon\, [p + q + 2, p + q + r + 2]. \end{cases} \tag{4.6.2}$$

Define X be the matrix formed by the columns ($\mathbf{X}(0)$, $\mathbf{X}(1)$, ...).

Conclusion

In this chapter, we introduced the Freddie Mac Single-Family dataset and the methods by which we used to process, load and prepare the data for training the model. We discussed our approach when creating new features based on geography and loan performance information from within the dataset. As well as introducing additional economic data, which included Housing Price Index, Unemployment rates, and National Interests rates. We then introduced the same set of features but which only consider the recent past, i.e. the last 12 months. We found that these indicators are more effective when applied to economic and geographical based features and less so when applied to individual loan performance features.

Chapter 5

Models

In this chapter, we introduce the prediction model definition and provide an overview of the network's hyper-parameters. We discuss implementation challenges related to the imbalanced datasets which we will refer back to in our results section.

5.1 Default Prediction

The Default Prediction model will take as input, n mortgage features at time t, and output a prediction as to whether the mortgage is more likely to be Fully Paid (negative) or Default (positive), i.e. the borrower pays back in full, or defaults on the loan. The model outputs a value between 0 and 1; therefore we use a threshold of 0.5. Values above this point are classified as Default and below are Fully Paid. The model will consider the issue of class importance whereby a false positive prediction (a Default loan that is classified incorrectly) will be penalised at the same relative rate as a false negative prediction (a Fully Paid loan that is classified incorrectly). This will be achieved using class weights, discussed in Section 5.1.1. We will refer to this objective as the *Default, Prediction, Objective.*

Parameter	**Description**
Input Features	133
Number of classes	2
Number of layers	4
Nodes in layers	133 : 100 : 100 : 2
Layer types	FC : FC_{BN} : FC_{BN} : FC
Loss Function	Softmax Cross Entropy
Activation Functions	ReLU
Weight Initialisation	Xavier Initialiser
Weight Regularisation	L2 Regulariser, $\lambda = 0.1$
Learning Rate	0.0005
Learning Rate Type	Adaptive Learning rate, $k = 500$ (see 3.4.3)
Momentum	$m = 0.9$ (see 3.4.4)
Gradient Descent Optimiser	Momentum Optimiser (Tensorflow)
Training Epochs	750
Training Size	10,700,000
Batch Size	5,000
Sampling Method	Under-Sampling / 15:85

FC = Fully Connected, FC_{BN} = Fully Connected with Batch Normalisation

5.1.1 Class Imbalance

In real word problems, it is often the case that binary classification needs to be performed on an imbalanced [1] dataset. It is also common that the unlikely events that occur in the minority class are difficult to separate from the event that occurs most often. For example, the majority of mortgage loans do not default during their active duration, but we would like to determine the probability of this event occurring.

In the default classification problem, there is a significant class imbalance, as

[1]Where more samples belong to one class than the other.

shown in table 4.3 in section 4.2. This imbalance is problematic as traditional algorithms are often biased towards the majority class, whereby they prioritise overall accuracy of the classifier above the accuracy of true positive and true negative predictions. This bias occurs because the loss functions optimise for metrics such as error and accuracy scores, and do not take into account the distribution of the data. The effect of imbalanced datasets has been well researched in academia; see Ho [1998] and Estabrooks [2000]. They can adversely affect a range of classifiers, impacting overall accuracy and negatively affecting the identification of rare features.

To help overcome this problem, we weight the output of the loss function proportional to the inverse class ratio for each mini-batch during training. More precisely, we create an inverse class ratio vector, $class\_weight(p)$, using the ground truth labels, p:

$$\begin{aligned} inv\_class\_ratio(p) = [\ & 1 - class\_ratio_0(p), \\ & 1 - class\_ratio_1(p)\ ] \end{aligned} \tag{5.1.1}$$

where $class\_ratios_0(p)$ is the class ratio for the Default (minority) class and $class\_ratios_1(p)$ is the class ratio for the Fully Paid (majority) class. The resulting effect is down-weighting the loss for the majority class and up-weighting the loss for the minority class, and this allows the loss function to treat both classes equally.

5.1.2 Cost-Sensitive Learning

We extend the idea further by implementing a cost-sensitive learning technique, where we add a Class Weight Scale-Factor, γ, which provides the ability to favour one class over the other. The Default classification problem has not only the issue of class imbalance but also the issue of class importance; where incorrectly predicting a Default loan instance will likely result in a higher loss (in a real-world application) than incorrectly predicting a Fully Paid instance. The formula for our cost-sensitive learning technique can be seen in equation 5.1.2.

$$class\_weights(p) = [\ (1 - class\_ratio_0(p)\,) * \gamma, \\ (1 - class\_ratio_1(p)\,) * (2 - \gamma)\ ] \quad , \qquad (5.1.2)$$

$where, \ \ 0 < \gamma < 2.$

We then adapt the cost function using equation 5.1.2 such that:

$$J(p, \hat{p}) = J(p, \hat{p}) * class\_weights(p) \quad , \qquad (5.1.3)$$

where $\hat{p}$ is the predicted output labels, and p is the real output labels. It should be noted that the Class Weight Scale-Factor, γ, is a parameter which is manually chosen before training to produce the desired Recall to Specificity ratio.

5.1.3 Model Architecture

The neural network architecture plays an important role in its ability to learn highly non-linear relationships. Determining the optimum number of hidden layers and nodes is typically very challenging without using trial and error approach on the specific dataset; as explained by Geman et al. [1992]. If a network contains too few nodes, it can lead to high error rates as the features might be too complex for the model to learn. Similarly, too many nodes will result in the model overfitting the training data, causing the out of sample performance to decrease.

Selecting the correct number of nodes for a given layer is not trivial. The procedure is one that is very problem specific, and the publications that address this selection process often recommended ranges that are broad. Given that, from Geman et al. [1992], L.Blum and Rivest [1992] and K [1996], we can conclude that the recommended number of nodes falls between half the number of inputs and twice the number of inputs.

Determining the number of hidden layers is also not a simple procedure. For linear problems, it is possible to forgo the hidden layers altogether. Moreover, even if the problem is slightly non-linear, this may still be the best approach as it provides good generalisation; as explained by K [1996]. In Siegelmann [1991] they state that in Multi-Layer Perceptron networks with threshold activation functions, only two hidden layers are required for full generality. Additionally, we observe that Sirignano et al. [2016], a paper that addresses a similar deep learning problem on an imbalanced dataset of similar scale, found network depths of 3 and 5 produced the lowest loss and highest AUC.

Given the variety in the above observations, we compare a number of different architecture in section 6.2,.

Chapter 6

Results

This chapter compares different techniques used to optimise our deep learning model, as well as discussing reasons for these results. For the default classification task, where a loan is labelled either Fully Paid or Default, we refer to the Full Paid class as the negative class and Default as the positive class. As we are using highly imbalanced data, it would be simple to obtain high accuracy by simply predicting every data point as the majority class. Therefore, our results section will focus on three performance metrics, as defined by Powers [2007]:

- **Recall** (True positive rate): The performance of the minority (Default) class
  - The proportion of Default instances that are correctly predicted as Default.
- **Specificity** (True negative rate): The performance of the majority (Fully-Paid) class
  - The proportion of Fully Paid instances that are correctly predicted as Fully Paid.
- **AUC** (Area Under the Curve): The correctness of the classifier
  - The proportion of correctly predicted instances.
  - AUC is $\frac{Recall+Specificity}{2}$

The individual techniques discussed in this section were evaluated under the optimum model parameters, as listed in Section 5, except for the parameter under evaluation. The majority of the techniques and parameters chosen were based on previous academic work, and are referenced where applicable. In some instances, we used the academic publications to gauge appropriate value ranges and then performed a parameter search within space. All techniques were evaluated using 5-fold cross-validation. The magnitude of the dataset allowed us to split the data into large enough subsets such that the learning ability of our model would not be affected.

Table 6.1 shows the performance results under the optimum model parameters.

Table 6.1: Default Prediction Model Results

Model	**Recall**	**Specificity**	**AUC**
Default Classification	0.985	0.982	0.984

6.1 Cost-Sensitive Learning (γ)

As discussed in Section 5.1.1, we propose a weighted loss function with a Class Weight Scale-Factor (γ). A comparison of different γ values can be seen in Table 6.2.

Table 6.2: Class Weight Scale-Factor (γ) Comparison Results

γ	**Recall**	**Specificity**	**AUC**
0.700	0.000	1.000	0.500
0.800	0.300	1.000	0.650
0.900	0.760	0.997	0.879
1.000	**0.985**	**0.982**	**0.984**
1.100	0.993	0.800	0.897
1.200	1.000	0.380	0.690
1.300	1.000	0.000	0.500

The results in table 6.2 and graph in figure 6.1 show there exists an apparent trade-off between Recall and Specificity.

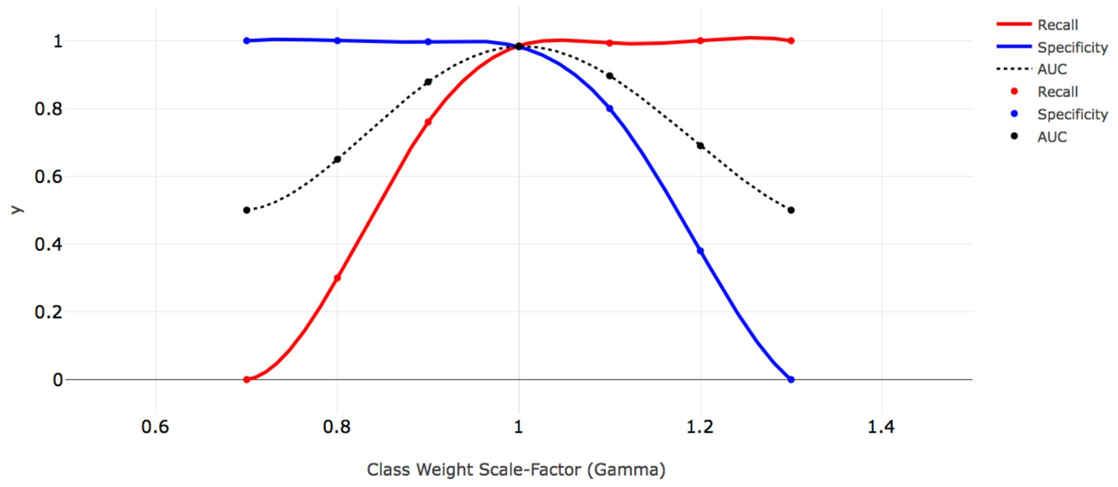


Figure 6.1: A comparison between Recall, Specificity and AUC at different Class Weight Scale-Factor (γ) values.

The optimum trade-off point between Recall, Specificity and AUC is where the lines in figure 6.1 meet, at $\gamma = 1.0$. A γ value of 1 means that the loss function favours Recall and Specificity equally, irrespective of class ratios which have already been accounted for (as explained in 5.1.1). Additionally, as discussed in section 3.3, the neural network updates its weights and biases by minimising the output of loss function during training. This process means that we would expect to find that the optimal AUC value at $\gamma = 1.0$, which is confirmed by our findings in table 6.2.

We believed it was important to explore the results of cost-sensitive learning. If we consider the real world implications of the model, we hypothesise that incorrect classification of a Default loan would lead to greater financial consequences than miss-classifying a Fully Paid loan. In this scenario, the appropriate γ value would be selected in order to best optimise the objective. An example of an objective would be the valuing of a loan portfolio, where it may be beneficial increase Recall performance. However, as this research has no outside objective, we will simply focus on producing a model that outputs the optimal AUC value, which is at $\gamma = 1.0$.

6.2 Model Architecture

The neural network architecture plays an important role in its ability to learn highly non-linear relationships. As discussed in 5.1.3, selecting the optimal model architecture is not obvious without some trial and error.

We use as a baseline a single-layered network which acts as a linear regression model. We compare the architecture from Sirignano et al. [2016], with a number of selected alternatives based on the research discussed in Section 5.1.3. The results can be seen in table 6.3.

Table 6.3: Model Architecture Comparison Results

Architecture	**Recall**	**Specificity**	**AUC**
133 : 2 (Linear)	0.605	0.993	0.799
133 : 100 : 2	0.961	0.957	0.959
133 : 100:100 : 2	**0.985**	**0.982**	**0.984**
133 : 100:100:100:100:100 : 2	0.953	0.912	0.933
133 : 200 : 2 [1]	0.954	0.851	0.953
133 : 200:140 : 2 [1]	0.961	0.971	0.966
133 : 200:140:140:140:140 : 2 [1]	0.951	0.899	0.920

Where, Architecture = **Input Layer : (Hidden Layer:)^i : Output layer**

We found that the optimum architecture, given the $Default\,Prediction\,Objective$, consists of 2 hidden layers each with 100 nodes. Across all architectures, we found that two hidden layers perform better than both one and five hidden layers. The performance of the linear model is dominated by every other architecture with at least one hidden layer. The linear model has the lowest AUC and Recall values which indicate there exist complex non-linear relationships in the data that cannot be separated using linear regression. These results also suggest that adding multiple hidden layers allows the model to learn more complex non-linear relationships,

[1]From Sirignano et al. [2016]

directly resulting in higher predictive power and performance. We hypothesis that in order to obtain a better result, using a higher number of hidden layers, we should decrease the learning rate and increase the number of epochs during training, as recommended by Delalleau and Bengio [2011]. However, due to time restriction, based on the fact that the model takes over eight hours to train, we did not test this hypothesis.

6.3 Resampling

Resampling methods allow for repeated sampling of the original data to build new sample distribution. There are two main types of resampling; under-sampling and over-sampling. Resampling methods can be beneficial when handling imbalanced data, the work by Japkowicz [2000] concludes that "both over-sampling the minority class and down-sizing the majority class are very effective methods of dealing with the problem."

Random Under-Sampling vs SMOTE (1% subset)

We have chosen to compare random under-sampling (Section 2.2.1(and SMOTE oversampling (Section 2.2.1). Due to the computational complexity of the SMOTE algorithm, we first performed a comparison on a 1% subset of the data to obtain its potential effectiveness. Note that, in order to prevent overfitting, cross-validation was performed before any re-sampling methods were applied. Additionally, the resampling methods were only applied to the training set and not the validation set.

Table 6.4: Resampling Techniques Comparison Results (1% subset)

Technique / Class Ratio	**Dataset %**	**Recall**	**Specificity**	**AUC**
None / 0.01:99.9	100%	0.359	0.955	0.657
SMOTE / 15:85	130%	0.627	0.811	0.719
SMOTE / 33:77	166%	0.659	0.730	0.693
SMOTE / 50:50	200%	0.697	0.659	0.678
Under-Sampling / 15:85	0.06%	0.797	0.740	0.769
Under-Sampling / 33:77	0.03%	0.707	0.800	0.754
Under-Sampling / 50:50	0.02%	0.567	0.845	0.706

The results in Table 6.4 show that SMOTE does not outperform random under-sampling. Batuwita and Palade [2010] found that the SMOTE method increased the minority class recognition rate and sacrificed less overall clarification performance when compared to under-sampling methods, concluding that the main benefit of random oversampling is less information loss. However, we did not find this to be true on the 1% subset. One explanation is that, after randomly under-sampling, the size of the dataset is still sufficiently large to be representative of the entire subset, and therefore information loss is much lower. Moreover, it is stated in the original literature that "the highest tested imbalanced training set was only about 10%". Therefore, as the imbalance in our dataset is much more significant than 10%, any comparison drawn between these two sets of results should be considerate of that fact.

Random Under-Sampling

The results in section 6.3 show that the SMOTE sampling method does not outperform under-sampling at any class ratio on a 1% subset of the data. Given this, and that we estimate it would take significant computation time for the entire training dataset to be up-sampled using SMOTE, we decided to not compare this method further. We also noted that under-sampling is generally outperformed by SMOTE on smaller datasets. The fact that this did not occur in our comparison on a 1%

subset of the data means that we have more reason not to expect it to increase performance when applied to the entire dataset.

Nevertheless, we compared the under-sampling technique at different class ratios to see if our findings in Section 6.3 hold true when applied to the entire dataset.

Table 6.5: Resampling Techniques Comparison Results

Technique / Class Ratio	**Dataset %**	**Recall**	**Specificity**	**AUC**
None / 0.1:99.9	100%	0.413	0.991	0.702
Under-Sampling / 15:85	0.06%	**0.985**	**0.982**	**0.984**
Under-Sampling / 33:77	0.03%	0.977	0.965	0.971
Under-Sampling / 50:50	0.02%	0.971	0.966	0.685

From the result in table 6.5 and 6.4, we observed that both the SMOTE algorithm and under-sampling technique outperform our base case, where no re-sampling method is applied. We observe that when no resampling method is applied, irrespective of the weighted class ratios, the model performs poorly. We hypothesise that because the minority class instances are processed so infrequently, it makes the decision regions very small relative to the overall vector space and therefore it is difficult for the network to converge optimally. The low minority-to-majority class ratio means that the minority class is significantly underrepresented in the data. These findings agree with that by Hensman and Masko [2015], which also found that the classifier performance is reduced when no re-sampling method is applied.

We also observe that classification performance, given the *Default Prediction Objective*, decreases as the re-sample ratios become closer to 50:50. An explanation for this is that, if the training dataset is re-sampled at the 50:50 ratio, there now exists a large difference in the class ratios between the training data and the validation data. When this difference is large, the representation that the model is learning becomes less representative of the validation data, and therefore classification performance decreases. Additionally, in the case of under-sampling, as the class ratios become closer to 50:50, the total % of the dataset being processed by the model decreases. This information loss could be another explanation for the decrease in classification performance.

Another consideration when analysing these results is that we are using a weighted loss function, discussed in section 5.1.2, and data re-sampling. We can observe that the majority of published work has either focused on re-sampling techniques or weighted loss functions. However, recent publications in the medical industry have successfully combined these two techniques, see Hsu et al. [2015]. We believe further research is required to support the effectiveness of combining both fully.

6.4 Normalisation

Data Normalisation is an important data pre-processing step which enforces the integrity of the data by ensuring consistency across all the values. The normalisation techniques have chosen to compare are Min-Max, Z-Score and Decimal Point normalisation.

We compare the different normalisation techniques in Table 6.6.

Table 6.6: Normalisation Techniques Comparison Results

Optimisation	**Recall**	**Specificity**	**AUC**
Baseline	0.949	0.941	0.945
Min-Max Normalisation [1]	0.970	0.959	0.965
Z-Score Normalisation [1]	0.961	0.972	0.967
Decimal Point Normalisation [1]	**0.985**	**0.982**	**0.984**

From results in table 6.6, we observe that applying a normalisation technique improves classification performance across every metric, in comparison to the baseline (no normalisation). We observe only a small difference between the three techniques compared in table 6.6, with Decimal Point normalisation producing the highest classification performance. These results agree with the findings by Mustaffa and Tusof

[2]From Mustaffa and Tusof [2011]

[2011], where they compared the same three techniques, finding that Decimal Point outperforms both Min-Max and Z-Score Normalisation techniques with lower loss value and higher accuracy.

Although the baseline model does not perform significantly worse than the other models with normalisation applied, if we remove the batch normalisation layer, performance decreases dramatically. This result highlights the importance of normalisation both, at the input level, and between layers, when training neural networks.

6.5 Batch Normalisation, Dropout & L^2 Regularisation

Batch Normalisation (BN) is an optimisation technique for improving stability and the performance of a neural network. In our implementation, we normalised the input data, as explained in section 6.4. We apply batch normalisation after the activation functions between each layer; as explained in section 3.5.1. Note that we do not need to use batch normalisation before the first layer as we provide the model with normalised data.

Dropout is a regularisation technique used to reduce overfitting. In our model, we applied dropout after each fully connected hidden layer and set the dropout probability to 0.5, as suggested by Hinton et al. [2012].

L^2 Regularisation is another regularisation strategy, which achieves its objective by using a regularisation term, as discussed in Section 3.6.2. In our model, we apply L^2 Regularisation before each hidden layer using a regularisation term, λ, of 0.1; as suggested by Demir-Kavuk et al. [2011].

Table 6.7: Batch Normalisation, Dropout & L^2 Regularisation Comparison Results

Technique	**Recall**	**Specificity**	**AUC**
Baseline	0.911	0.798	0.855
L^2 Reg	0.891	0.928	0.909
Dropout	0.851	0.918	0.885
Dropout & L^2 Reg	0.850	0.908	0.879
Batch Norm	0.967	0.919	0.943
Batch Norm & L^2 Reg	**0.985**	**0.982**	**0.984**
Batch Norm & Dropout	0.0.947	0.929	0.938
Batch Norm & Dropout & L^2 Reg	0.945	0.926	0.935

The results in table 6.7 show that Batch Normalisation is an effective technique to improve classification performance. We observed that it outperformed dropout and that it was improved with the addition of L^2 Regularisation. We found that the need for dropout was strongly reduced when combined with batch normalisation. Additionally, we observed that all techniques showed improved performance over the baseline model.

6.6 Feature Selection

Feature selection is the process of selecting a subset of relevant features that provide the necessary information to train a model for a particular task. As discussed in Section 4.4, we have engineered a variety of novel features and removed redundant information, both of which should help to model in the classification task.

6.6.1 With Monthly Performance Features

In this section, we will evaluate how the model performs under different subsets of features, using both the origination and monthly performance data to determine default prediction. We define the following subsets, using notation similar to that in Section 4.6:

- **Selection A**: Origination vector, $\mathbf{O}$
- **Selection B**: Selection A and Monthly updates, $\mathbf{M(t)}$
- **Selection C**: Selection B and Economic performance update vector, $\mathbf{E_A(t)}$
  - Where $\mathbf{E_A(t)}$ are features from Sirignano et al. [2016] as marked in Table 4.5.
- **Selection D**: Selection C and Economic performance update vector, $\mathbf{E_B(t)}$
  - Where $\mathbf{E_B(t)}$ are novel features as shown in Table 4.5

Table 6.8: Feature Selection Comparison Results (With Performance Updates)

Feature Selection Type	**Recall**	**Specificity**	**AUC**
Feature Selection A	0.690	0.724	0.707
Feature Selection B	0.862	0.925	0.893
Feature Selection C	0.973	0.952	0.963
Feature Selection D	**0.985**	**0.982**	**0.984**

From the results in table 6.8, we observe an improvement in classification performance as we increase the number of features through Selection [A-D].

Selection A

Selection A only considers a single data vector per loan, each containing information from the origination point. This constraint means that the sample selection size is much smaller than the others proposed, [B-D]. The classifier effectively acts as an indicator which could be used by the lenders at the time the loan is originally being considered to help evaluate a prospective mortgage buyer. The selection performs better than random[3], which suggests it is possible to predict the outcome of a loan given only the information directly provided by the mortgage vendor at origination.

[3]A random prediction model would, on average, result in Recall, Specificity and AUC score of 0.5.

Selection B

Selection B resulted in the most significant increase in classification performance. After introducing the monthly performance vector, $\mathbf{M(t)}$, each loan has, on average, 21 input vectors (i.e. the average duration of a loan). We hypothesise that it becomes easier for the model to determine the outcome of a loan as the age of the loans increase, resulting in a higher classification performance using Selection B over Selection A. We can validate this claim by examining the model's output on a sample of 50 default loans. We observe that on average, as the age of the loan increases, the output of the model moves closer to 1. This indicates that the model has a higher level of certainty, on average, as the loan age increases.

Selection C

Selection C introduces the economic vector, $\mathbf{E_A(t)}$, as suggested by Sirignano et al. [2016]. This vector contains features in addition to what has been provided by the mortgage vendor in both the origination and monthly update vectors; discussed in detail in Section 4.4.3. We observe a significant increase in classification performance across all three metrics when comparing Selection B and C. We deduce that the addition of geographically specific and time-based features does, in fact, improve overall performance.

Selection D

Selection D introduces the economic vector, $\mathbf{E_B(t)}$. This vector contains features over and above what has been proposed by Sirignano et al. [2016]. We chose to extend the geographic-based features to both State and Zip-code, across multiple different metrics, as well as introduce additional economic data; discussed in detail in Section 4.4.3. We again observed an increase in classification performance across all three metrics when comparing Selection C and D. We deduced that the addition of more geographically specific and time-based features did, in fact, further improve overall performance. We can see from figures 4.5 and 4.7 in Section 4.4.3, that these additional features can be used to help distinguish between 'Fully Paid' and 'Default' loans. We note that this observation does not guarantee that the performance of the model will improve; it is possible that the neural network using existing information already captures the observed separation between 'Fully Paid' and 'Default' on these features. However, this is likely not the case, as the results show an improvement in classification performance.

6.6.2 Without Monthly Performance Features

In this section, we will evaluate how the model performs under different subsets of features, using only origination data to determine Default prediction. i.e. For each Selection, we only have a single input vector per loan that was derived from data obtained at the origination date of the loan. We defined the following subsets, using notation similar to that in Section 4.6:

- **Selection A**: Origination vector, $\mathbf{O}$
- **Selection A1**: Selection A and Economic performance update vector, $\mathbf{E_A(t)}$
  - Where $\mathbf{E_A(t)}$ are features from Sirignano et al. [2016] as marked in Table 4.5.
- **Selection A2**: Selection A1 and Economic performance update vector, $\mathbf{E_B(t)}$
  - Where $\mathbf{E_B(t)}$ are novel features as shown in Table 4.5

Table 6.9: Feature Selection Comparison Results (Origination Only)

Feature Selection Type	**Recall**	**Specificity**	**AUC**
Feature Selection A	0.690	0.724	0.707
Feature Selection A1	0.866	0.813	0.840
Feature Selection A2	0.910	0.851	0.881

From the results in table 6.8, we observed an improvement in classification performance as we increased the number of features through Selection [A-A2].

Selection A

Same as discussed in Section 6.6.1.

Selection A1

Selection A1 introduces the economic vector, $\mathbf{E_A(t)}$, as suggested by Sirignano et al. [2016]. We observed an improvement in performance classification across all metrics when comparing Selection A1 with A. We concluded that the reason for this was the same as discussed for Selection C in Section 6.6.1.

Selection A2

Selection A2 introduces the economic vector, $\mathbf{E_B(t)}$. We observed an improvement in performance classification across all metrics when comparing Selection A2 with A1. We concluded that the reason for this is the same as discussed for Selection D in Section 6.6.1.

It is interesting to note that the addition of $\mathbf{E_A(t)}$ and $\mathbf{E_B(t)}$ significantly improves classification performance at origination. This classification model could, in theory, be used by a mortgage vendor at the time the loan is originally being considered in order to help evaluate a prospective mortgage buyer

Conclusion

This section compared the results from a variety of methods; these include resampling, normalisation, regularisation and cost-sensitive learning. We also evaluated performance under different feature selections and model architectures, where all comparisons are evaluated using 5-fold cross-validation. The model achieved an AUC score of 0.984 under the optimum choice of parameters and techniques.

Chapter 7

Conclusion

In this thesis, we introduce the concept of using deep neural networks to classifying loans into two classes, Default or Fully Paid. We began by outlining the motivation for the research and providing an overview of the previous related literature. This literature showed that metrics such as loan age, FICO, Loan-to-Value ratio, Debt-to-Income ratio and local economic factors correlate with mortgage foreclosure rates and prepayment likelihood. We then produced a technical background of the methods that were introduced later in the paper.

Subsequently, we detailed the techniques used to load and prepare the data for training and discussed our approach to analysing existing features where we showed a correlation between features such as FICO score and increased occurrence of default loans. We then outlined how we created new features based on geography and loan performance information from within the dataset, as well as introduce additional economic data which included Housing Price Index, Unemployment rates, and National Interests rates. Subsequently, we created the same set of features but which only consider the recent past, i.e. the last 12 months. We found that these indicators are more effective when applied to economic and geographical based features and less so when applied to individual loan performance features.

In our results section, we compared a variety of methods where we evaluated the model performance under each using 5-fold cross validation. We confirmed that a deep neural network (0.984 AUC) significantly outperformed a basic linear regression model (0.799 AUC), therefore validating the performance benefits of neural network models in this domain.

If we compare our work to similar research as published by Bagherpour [2017] and Deng [2016], both papers implement models that attempt to classify loans that are 'Paying' versus 'Default' (our work looks at 'Default' versus 'Fully Paid' and does not consider active loans) using a logistic regression model. They achieved AUC scores of 0.860 and 0.968 respectively. Our results compare favourably to these. However, we did note that there is a difference in the exact classification objective between their research and our paper.

We showed that the addition of novel features improved the classification performance increasing the AUC score from 0.893 to 0.984. As far as we are aware, many of the features introduced in the economic vector, $\mathbf{E_B(t)}$, have not been previously researched in academic literature. We believe that this is due to limitations in data source up to now, and also the computational requirement to process data on this scale. We, therefore, believe this provides a new and useful contribution to the field.

7.1 Applications

The model has many potential applications. It could be used by risk departments in financial institutions to more accurate value loan portfolios, helping to determine their exposure on current assets holdings. It could also be used by asset managers directly to determine the strength and valuation of different Mortgage-Backed-Securities.

We see viable commercial applications where the model could be used in conjunction with other indicators in an ensemble process. As we discussed in chapter 2, banks are expected to have adequate early warning systems in place to identify and monitor high-risk loans ahead of time. The type of model we are proposing in this thesis could, as part of a larger decision process, contribute to such an early warning system.

If we consider how this type of model would be used in practice, we believe that the Cost-Sensitive Learning approach would be beneficial. Although it does not directly impact the results of this paper, the ability to control the trade-off point between Recall, Specificity and AUC could be advantageous given different outside objectives. For example, in the medical industry it is used to reduce the chance of a false positive; see Hsu et al. [2015].

7.2 Limitations

The ability of our findings to generalise across multiple countries may be limited as our dataset is restricted to only the United States. Although we have found new features which help the classifier to separate between Fully Paid and Default loans, these same features may not result in increased performance if considering data from another economy.

A limitation of our approach in practice is that we disregard a large number of loans because they are still active. One approach could be to train two models, one that looks at the outcome of a loan at termination and another which attempts to predict its active state at some future point in time.

Another limitation is that we only compare our neural network model to a linear regression model. We believe that in order to fully evaluate the effectiveness of deep learning for the classification of mortgages, the results of other machine learning methods such as Support Vector Machines and Random Forest algorithms should be analysed and compared.

7.3 Future work

Mortgage Prediction

This thesis has evaluated the predictive performance of deep neural networks in classifying loans into two classes, Default or Fully Paid. The classification task has been performed on a broad spectrum of loans spread across the United States.

Future work could include an investigation into specific sub-classes of loans. For example, it would be interesting to know how well a neural network would perform using only Non-Performing Loans (NPLs)[1]. Similarly, it would be interesting to build separate models for different clusters within the data, for example, a predictive model for each state within the USA.

Deep Learning

We believe that further research is required to understand the optimum re-sampling ratio better when using a weighted loss function. In overcoming the issue of an imbalanced dataset, we used a weighted loss function and a random under-sampling method with a class ratio of 15:85 (Default : Fully Paid). We observe an interesting relationship between the re-sampled class ratios and the performance of the model; as the re-sample ratios move from 15:85 to 50:50, the classification performance decreases. However, if we compare ratio 15:85 to 0.1:99.9 (where no re-sampling method has been applied), we also observe a decrease in performance. We cannot definitively explain why this is the case, however, our hypothesis can be seen in section 6.3.

A method we overlooked due to time limitations was that of an ensemble approach. This approach combines multiple models where the output is the average taken across all the classifiers. This type of approach is known to limit overfitting and reduce the variance of the prediction. It can also be advantageous because typically the network converges at different local minimums due to the variation in weight initialisation (discussed in Section 3.4.6) and randomised ordering of the mini-batch samples (discussed in Section 3.4.2. It would be interesting to evaluate whether an ensemble method with multiple neural networks, or even different machine learning methods would improve overall classification performance.

[1]Loans are Non-Performing if they are 90+ days delinquent

Bibliography

1. Brent Ambrose and Charles A. Capone. Modeling the conditional probability of foreclosure in the context of single-family mortgage default resolutions. *Real Estate Economics*, 26:391–429, 02 1998.

2. Ali Bagherpour. Predicting mortgage loan default with machine learning methods. 2017.

3. Rukshan Batuwita and Vasile Palade. Efficient resampling methods for training support vector machines with imbalanced datasets. *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2010.

4. Kevin W. Bowyer, Nitesh V. Chawla, Lawrence O. Hall, and W. Philip Kegelmeyer. SMOTE: synthetic minority over-sampling technique. *CoRR*, abs/1106.1813:16, 2011.

5. Mateusz Buda, Atsuto Maki, and Maciej A. Mazurowski. A systematic study of the class imbalance problem in convolutional neural networks. *CoRR*, abs/1710.05381, 2017.

6. Tim S Campbell and J Kimball Dietrich. The determinants of default on insured conventional residential mortgage loans. *Journal of Finance*, 38(5):1569–81, 1983.

7. Dennis Capozza and Thomas Thomson. Subprime transitions: Lingering or malingering in default? *The Journal of Real Estate Finance and Economics*, 33 (3):241–258, Nov 2006.

8. Hakan Cevikalp. Best fitting hyperplanes for classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39:1076–1088, 2017.

9. Sewin Chana, Claudia Sharyginb, Vicki Beenc, and Andrew Haughwoutd. Pathways after default: What happens to distressed mortgage borrowers and their homes? *The Journal of Real Estate Finance and Economics*, 48(2):342–379, Feb 2014.

10. Michel Crouhy. Risk management failures during the financial crisis. 11 2011.

11. CUDA. Documentation. `https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html`, Accessed: 2018-01-29.

12. Michelle A. Danis and Anthony Pennington-Cross. The delinquency of subprime mortgages. 2005.

13. Michelle A. Danis and Anthony Pennington-Cross. The delinquency of subprime mortgages. *Journal of Economics and Business*, 60(1):67 – 90, 2008.

14. Olivier Delalleau and Yoshua Bengio. Shallow vs. deep sum-product networks. *Advances in Neural Information Processing Systems 24*, pages 666–674, 2011.

15. Ozgur Demir-Kavuk, Mayumi Kamada, Tatsuya Akutsu, and Ernst-Walter Knapp. Prediction using step-wise l1, l2 regularization and feature selection for small data sets with large number of features. *BMC Bioinformatics*, 2011.

16. Grace Deng. Analyzing the risk of mortgage default. 2016.

17. A. Estabrooks. A combination scheme for inductive learning from imbalanced datasets (m.sc. thesis). 2000.

18. Mark Flood, H. V. Jagadish, Albert Kyle, Frank Olken, and Louiqa Raschid. Using data for systemic financial risk management. *CIDR*, 2011.

19. Kenichi Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2:183–192, 1989.

20. Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4:1–58, 1992.

21. Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. *International Conference on Artificial Intelligence and Statistics*, pages 249–256, 01 2010.

22. Frederic Godina, Jonas Degravea, Joni Dambrea, and Wesley De Neve. Dual rectified linear units (drelus): A replacement for tanh activation functions in quasi-recurrent neural networks. *CoRR*, abs/1707.08214, 2017.

23. Paulina Hensman and David Masko. The impact of imbalanced training data for convolutional neural networks. 2015.

24. Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

25. Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20:832–844, 1998.

26. T.Ryan Hoens and Nitesh V. Chawla. Imbalanced datasets: From sampling to classifiers. 2013.

27. Jia-Lien Hsu, Ping-Cheng Hung, Hung-Yen Lin, and Chung-Ho Hsieh. Applying under-sampling techniques and cost-sensitive learning methods on risk assessment of breast cancer. *Journal of Medical Systems*, 39:1–13, 2015.

28. Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ICML*, 2015.

29. Occas Ional, Geoff Kenny, and J. Brad Morgan. Some lessons from the financial crisis for the economic analysis. 2011.

30. Nathalie Japkowicz. The class imbalance problem: Significance and strategies. *Proceedings of the 2000 International Conference on Artificial Intelligence ICAI*, 06 2000.

31. Philippe Jorion. Risk management lessons from the credit crisis. *European Financial Management*, 15(5):923–933.

32. Citation Swingler K. Applying neural networks: A practical guide. 1996.

33. Avrim L.Blum and Ronald L. Rivest. Training a 3-node neural network is np-complete. *Neural Networks*, 5(1):117 – 127, 1992.

34. Freddie Mac. Single family loan-level dataset general user guide. `http://www.freddiemac.com/research/pdf/user_guide.pdf`, Accessed: 2018-01-29.

35. Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, pages 115–133, 1943.

36. Multitasking. Documentation. `https://docs.python.org/2/library/threading.html`, Accessed: 2018-02-13.

37. Zuriani Mustaffa and Yuhanis Tusof. A comparison of normalication techniques in predicting dengue outbreak. *International Journal of Computer Theory and Engineering*, 3(4), 2011.

38. Michael A. Nielsen. Neural networks and deep learning. 2018.

39. Numpy. Documentation v1.4. `https://docs.scipy.org/doc/numpy-1.4.x/reference/`, Accessed: 2018-01-29.

40. Pandas. Documentation v0.22.0. `https://pandas.pydata.org/pandas-docs/stable/whatsnew.html#v0-22-0-december-29-2017`, Accessed: 2018-01-29.

41. Jonathan A. Parker, Nicholas S. Souleles, David S. Johnson, and Robert McClelland. Consumer Spending and the Economic Stimulus Payments of 2008. January 2011.

42. David Powers. Evaluation: From precision, recall and f-factor to roc, informedness, markedness correlation. 2007.

43. Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

44. J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

45. Hava T. Siegelmann. Turing computability with neural nets. 1991.

46. Justin Sirignano, Apaar Sadhwani, and Kay Giesecke. Deep learning for mortgage risk. 2016.

47. Thomas Grant Slatton. A comparison of dropout and weight decay for regularizing deep neural networks. 2014.

48. Tensorboard. Documentation. `https://www.tensorflow.org/programmers_guide/summaries_and_tensorboard`, Accessed: 2018-01-29.

49. Tensorflow. Documentation v1.2. `https://www.tensorflow.org/versions/r1.2/api_docs/python/`, Accessed: 2018-01-29.

50. US Bureau of Labor Statistics. Housing price index, unemployment rate per state, and national interests rates. `https://www.bls.gov`, Accessed: 2018-03-17.

51. Kerry D. Vandell. Default risk under alternative mortgage instruments. *Journal of Finance*, 33(5):1279–96, 1978.

52. Ioan Voicu, Marilyn Jacob, Kristopher Rengert, and Irene Fang. Subprime loan default resolutions: Do they vary across mortgage products and borrower demographic groups? *The Journal of Real Estate Finance and Economics*, 45 (4):939–964, Nov 2012.

53. George von Furstenberg. Default risk on fha-insured home mortgages as a function of the terms of financing: A quantitative analysis. *Journal of Finance*, 24 (3):459–77, 1969.

54. Bruce G. Webb. Borrower risk under alternative mortgage instruments. *Journal of Finance*, 37(1):169–83, 1982.

55. Ying Xie. Knn++: An enhanced k-nearest neighbor approach for classifying data with heterogeneous views. pages 13–23, 11 2016.

56. Ernst Young. Remaking financial services: risk management five years after the crisis. 2013.

57. C. Zhang and P. C. Woodland. Parameterised sigmoid and relu hidden activation functions for dnn acoustic modelling. 2015.

Appendices

Table A.1: Origination Feature Descriptions

Loan Features	**Description**
fico	**FICO Credit Score** is a number, prepared by third parties, summarising the borrowers creditworthiness.
dt_first_pi	**First Payment Date** is the date of the first scheduled mortgage payment.
flag_fthb	**First Time Buyer Flag** indicates whether the Borrower had no ownership interest (sole or joint) in a residential property during the three-year period preceding.
dt_matr	**Maturity Date** is the month of the final scheduled payment.
cd_msa	**Metropolitan Statical Area** is an indicator, based on a geographical region with a relatively high population density
mi_pct	**Mortgage Insurance Percentage** is the percentage of loss coverage from mortgage insurer.
cnt_units	**Number of Units** denotes whether the mortgage is a 1, 2, 3 or 4 unit property
occpy_sts	**Occupancy Status** denotes whether the mortgage type is owner occupied, second home, or investment property.
cltv	**Combined Loan-To-Value** is the ratio obtained by dividing all of the borrowers outstanding mortgage loan amounts by the appraised value of all mortgaged propertys.
dti	**Debt-To-Income** is the debt to income ratio of the borrower.
orig_upb	**Original Unpaid Principal Balance** is the portion of the loan at origination that has not yet been remitted to the lender.

Continued on next page

Continued from previous page

Loan Features	**Description**
ltv	**Loan-To-Value** is the ratio obtained by dividing the original mortgage loan amount by the mortgaged propertys appraised value.
int_rt	**Interest Rate** is the original rate as indicated on the mortgage note.
channel	**Channel** is the third part that disclosed the mortgage (Retail, Broker, Correspondent)
ppmt_pnlty	**Prepayment Penalty Mortgage Flag** Denotes whether the mortgage is a PPM. Meaning a borrower obligated to pay a penalty in the event of certain repayments of principal.
prod_type	**Product Type** Denotes that the product is a fixed-rate mortgage.
st	**State** is a two-letter abbreviation indicating a US state.
prop_type	**Property Type** Denotes whether the property type secured by the mortgage is a condominium, leasehold, planned unit development (PUD), cooperative share, manufactured home, or Single Family home.
zipcode	**Post/Zip Code** is the postal code for the location of the mortgaged property
id_loan	**Loan Sequence Number** is the unique identifier assigned to each loan.
loan_purpose	**Loan Purpose** Indicates whether the mortgage loan is a Cash- out Refinance mortgage, No Cash-out Refinance mortgage, or a Purchase mortgage.

Continued on next page

Continued from previous page

Loan Features	**Description**
orig_loan_term	**Original Loan Term** is a calculation of the number of scheduled monthly payments based on the First Payment Date and Maturity Date.
cnt_borr	**Borrower Count** is the number of Borrower(s) who are obligated to repay the mortgage note.
seller_name	**Seller Name** is the entity acting in its capacity as a seller of mortgages.
servicer_name	**Servicer Name** is the entity acting in its capacity as the servicer of mortgages.
flag_sc	**Super Conforming Flag** indicates mortgages that exceed conforming loan limits with origination dates on or after 10/1/2008 and settlements on or after 1/1/2009.

[34]

Table A.2: Original Monthly Performance Feature Descriptions

Loan Features	**Description**
id_loan	**Loan Sequence Number** is the unique identifier assigned to each loan.
svcg_cycle	**Monthly Reporting Period** is the as-of month for loan information contained in the loan record.
current_upb	**Current Unpaid Principal Balance** is the portion of the loan at current time that has not yet been remitted to the lender.
delq_sts	**Delinquency Status** is a value corresponding to the number of days the borrower is delinquent (overdue).
loan_age	**Loan Age** is the number of months since the note origination month of the mortgage.
mths_remng	**Months Remaining** is the remaining number of months to the mortgage maturity date.
repch_flag	**Repurchase Flag** indicates loans that have been repurchased.
flag_mod	**Modification Flag** indicates mortgages with loan modifications.
cd_zero_bal	**Zero Balance Code** is a code indicating the reason the loan's balance was reduced to zero.
dt_zero_bal	**Zero Balance Date** is the date on which the event triggering the Zero Balance Code took place.
current_int_rt	**Current Interest Rate** is the current rate as indicated on the mortgage note.
non_int_brng_upb	**Current Deferred Unpaid Principal Balance** is the current non-interest bearing UPB of the modified mortgage.

Continued on next page

Continued from previous page

Loan Features	**Description**
dt_lst_pi	**Date of Last Paid Instalment** is the due date that the loans scheduled principal and interest is paid through, regardless of when the instalment payment was actually made.
mi_recoveries	**Mortgage Insurance Recoveries** are the proceeds received by the vendor in the event of credit losses.
net_sale_proceeds	**Net Sales Proceeds** are the amount remitted to Freddie Mac resulting from a property disposition once allowable selling expenses have been deducted from the gross sales proceeds of the property.
non_mi_recoveries	**Non Mortgage Insurance Recoveries** are proceeds received by vendor on non-sale income such as refunds (tax or insurance), hazard insurance proceeds, rental receipts, positive escrow and/or other miscellaneous credits.
expenses	**Expenses** include allowable expenses that vendor bears in the process of acquiring, maintaining and/ or disposing a property.
legal_costs	**Legal Costs** are the the amount of legal costs associated with the sale of the property.
maint_pres_costs	**Maintenance and Preservation Costs** are the amount of maintenance, preservation, and repair costs on the property.
taxes_ins_costs	**Taxes and Insurance** are the amount of taxes and insurance owed that are associated with the sale of a property.
misc_costs	**Miscellaneous Costs** are the miscellaneous expenses associated with the sale of the property.
actual_loss	**Actual Loss** is calculated using, 'Actual Loss' = ('Default UPB' 'Net Sale Proceeds') + 'Delinquent Accrued Interest' - 'Expenses' 'MI Recoveries' 'Non MI Recoveries'.
modcost	**Modification Cost** is the cumulative modification cost amount calculated when vendor determines such mortgage loan has experienced a rate modification event.

Continued on next page

[34]

[34]

Table A.3: Zero Balance Code Descriptions

Zero Balance Code	**Description**
01	Prepaid or Matured (Voluntary Payoff)
03	Foreclosure Alternative Group (Short Sale, Third Party Sale, Charge Off or Note Sale)
06	Repurchase prior to Property Disposition
09	REO Disposition

Table A.4: Loan Status Rules

- Current:
  1. Delinquency Status equals 0
- 30 Days Delinquent:
  1. Delinquency Status equals 1
- 60 Days Delinquent:
  1. Delinquency Status equals 2
- 90+ Days Delinquent:
  1. Delinquency Status greater than or equal to 3
- Foreclosed:
  1. Zero Balance Code equals 3
  2. Zero Balance Code equals 6
- REO (Real-Estate Owned):
  1. Zero Balance Code equals 9
  2. Delinquency Status equals ’R’
- Fully Paid:
  1. Zero Balance Code equals 1
  2. Repurchase Flag equals ’N’