
# Consumer credit default prediction using machine learning:
## an application of deep learning

Alan Wrigglesworth
(14029512)

Department of Mathematics and Applied Mathematics
University of Pretoria
2021

# Consumer credit default prediction using machine learning:
## an application of deep learning

Alan Wrigglesworth
(14029512)

Submitted in partial fulfilment of the requirements for the degree

**Magister Scientiae**
**Financial Engineering**
to the Department of Mathematics and Applied Mathematics
in the Faculty of Natural and Agricultural Sciences

**Supervisor:** Prof Eben Maré

I declare that this work has been composed solely by myself and that it has not been submitted, in whole or in part, in any previous application for a degree. Except where states otherwise by reference or acknowledgement, the work presented is entirely my own.

Images and figures used were either produced by myself or adapted from online sources and properly acknowledged.

Alan Wrigglesworth
July 1, 2021

**Abstract**

Over the last couple of years, we have seen much advancement in mathematical analysis and computational capabilities. This advancement, coupled with the increased availability of big data, has made it possible to commoditise machines and enable them to act as risk managers and financial analysts. In this dissertation, we will briefly review machine learning and consumer credit risk/scoring. We look at different methods and models proposed in the literature and thoroughly explore the mathematical theory behind deep learning. We then apply this knowledge and other recent advancements in the field to build a fully connected feed-forward deep neural network using open source credit card default data from a large Taiwanese retail bank. Our deep neural network aims to improve upon other models proposed in the literature regarding accuracy and other metrics such as ROC-AUC, Cohen's Kappa, precision, recall and F1-score. We then conclude that deep neural networks are competitive in terms of performance compared to other machine learning models and outperform traditional models. We highlight the potential that deep learning has yet to achieve in finance and pay close attention to the hurdles faced in complexity, development costs and regulatory roadblocks.

***Keywords:*** machine learning, neural networks, deep learning, consumer credit risk, credit scoring

"It ain't what you don't know that gets you into trouble. It's what you know for sure that just ain't so." - *Mark Twain (1835–1910)*

# Acknowledgements

I want to thank my supervisor, Prof. Eben Maré. I will always be grateful for the opportunity, his help, guidance and support throughout this research, but most importantly for encouraging me throughout and being patient with me.

To my fiancée, thank you for your unwavering support throughout this whole journey. I want to thank my parents for their love and support and for making all of this possible. Thank you for always supporting my dreams and goals. I also would like to thank Riskworx for their support, both financial and research related.

Last but not least, I would like to thank God for the talents and opportunities with which He has blessed me.

**Alan Wrigglesworth**
**2021**

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# List of Acronyms

**GFC -** Global Financial Crisis

**CCS -** Consumer Credit Scoring

**AI -** Artificial Intelligence

**ML -** Machine Learning

**NN -** Neural Network

**DL -** Deep Learning

**SME -** Small and Medium Enterprises

**ANN -** Artificial Neural Network

**CCR -** Consumer Credit Risk

**SVM -** Support Vector Machine

**MDA -** Multiple Discriminant Analysis

**CART -** Classification and Regression Trees

**MARS -** Multivariate Adaptive Regression Splines

**DNN -** Deep Neural Network

**PD -** Probability of Default

**LGD -** Loss Given Default

**RR -** Recovery Rate

**EAD -** Exposure at Default

**EL -** Expected Loss

**UL -** Unexpected Loss

**EC -** Economic Capital

**SME -** Small Medium Enterprise

**LTV -** Loan to Value

**RBP -** Risk-Based Pricing

**NN -** Neural Networks

**FP -** False Positive

**TP -** True Positive

**TN -** True Negative

**FN -** False Negative

**ROC -** Receiver Operating Characteristic

**ReLU -** Rectified Linear Unit

**GLM -** Generalised Linear Model

**BP -** Backpropogation

**SGD -** Stochastic Gradient Descent

**MBGD -** Mini-Batch Gradient Descent

**RMSE -** Root Mean Square Error

# Glossary of Frequently Used Terms

**Artificial Intelligence** is a branch of computer science dealing with the development of machines or computers. These machines can perform tasks that humans normally perform because they require human intelligence and discernment, such as visual perception, speech recognition, decision-making, and translation between languages.

**Machine Learning** is an application of artificial intelligence that provides systems with the ability to learn and improve from experience without being explicitly programmed. Put differently, machine learning is a type of artificial intelligence that enables self-learning from data and then applies that learning without human intervention.

**Data Science** is the field of study that combines domain expertise, programming skills, and knowledge of mathematics and statistics to extract meaningful insights from large amounts of complex data or big data.

**Big Data** is a term used to refer to massive data sets that can only be analysed computationally to reveal patterns, trends, and associations, especially relating to human behaviour and interactions.

**Neural Network** is a subset of machine learning, and it refers to systems of neurons, artificial. This system of neurons is a series of algorithms that aim to recognise underlying patterns and relationships in a set of data through a process that mimics how the human brain operates, i.e. learning by example (or trial and failure).

**Deep Learning** is a subset of machine learning and neural networks where the networks contain many layers of neurons (making the network deep) so that it is capable of learning from data that is unstructured or unlabeled. Deep learning is also known as deep neural learning or deep neural networks.

# List of Notation

$y$ - dependent / output variable

$x$ - independent / input variable

$\beta$ - regression coefficient

$\epsilon$ - regression error

$\hat{y}$ - predicted output

$w_i$ - weight / connections strength of input $x_i$

$\mathbf{X}$ - vector of input variables / features

$Y$ - binary dependent outcome variable

$p$ - probability of default or $P(Y = 0)$

$q_{i,j}$ - transition probability of going from state $i$ to state $j$.

$P_e$ - rate of incorrectly classified instances

$P_a$ - rate of correctly classified instances

$\kappa$ - Cohen's Kappa

$k$ - number of folds in a cross-validation procedure

$n$ - number of times a cross-validation procedure is repeated

$\mathcal{F}_1$ - F1 score

$AUC$ - ROC Area Under Curve

$x_i$ - input variables

$y/z$ - output variable

$\boldsymbol{\theta}$ - parameter vector

$\omega_{ij}$ - weight if input $i$ for the $j$th GLM

$\beta_i^{(l)}$ - $i$th offset / bias term of layer $l$

$\Phi()$ - activation function

$\mathbf{x}$ - vector of inputs

$n$ - number of input variables / features

$h_m^{(l)}$ - $m$th hidden unit of layer $l$

$M_l$ - the number of hidden units in layer $l$

$\mathbf{b}^{(l)}$ - bias vector of layer $l$

$\mathbf{W}^{(l)}$ - weight matrix of layer $l$

$L$ - number of layers in our model

$K$ - number of nodes in the output layer

$\mathcal{T}$ - training data set

$\mathcal{D}$ - data set

$J()$ - cost function

$L()$ - loss function

$\hat{\boldsymbol{\theta}}$ - approximate parameter matrix

$p$ - number of training instances

$q$ - number of instances in our data set

$\boldsymbol{\theta}_t$ - parameter vector for epoch $t$

$\lambda$ - step size of the gradient descent method

$\hat{\boldsymbol{g}}_t$ - approximation of the true gradient at step $t$

$E$ - number of epochs

$\nabla_{\boldsymbol{\theta}} L(\mathbf{x}_i, \mathbf{y}_i, \boldsymbol{\theta})$ - the gradient of the loss function w.r.t. $\boldsymbol{\theta}$

$\hat{\boldsymbol{g}}_t$ - the approximation of the gradient of the loss function w.r.t. to $\boldsymbol{\theta}$

$p_{mb}$ - sample size of a mini-batch

$R()$ - regularisation term

$\alpha$ - regularisation coefficient

$J^R()$ - regularised cost function

$\rho$ - dropout probability

$\lambda_t$ - the learning rate during epoch $t$

$\mathbf{v}_t$ - the update vector

$\gamma$ - momentum parameter

$\lambda_{t,i}$ - step size for parameter $\theta_i$ at time $t$

$\boldsymbol{\theta_t}$ - vector of parameters $\theta_i$ at time $t$

$g_{t,i}$ - partial derivative of the cost function w.r.t $\theta_i$ at time $t$

$\mathbf{G}_t$ - diagonal matrix with elements $g_{t,i}$

$\mathbf{m}_t$ - an exponentially decaying average of past gradients $vecg_t$

# Chapter 1

# Introduction

> "Essentially risk management involves getting data, analysing it, and making decisions based on the results. The first two parts of that process usually require someone to write some code. But risk managers rarely have the luxury of large teams of dedicated technologists which a front office managing director can call upon. They have to be highly resourceful, and do more with less. Often much of the grunt work has to be done in house. To do this well requires knowledge of an esoteric set of coding languages and tools." - Robert Carver.

## 1.1 Background

In the wake of the 2008 Global Financial Crisis (GFC), monitoring consumer behaviour and credit risk has moved into centre focus for all banks and creditors. Because of this, any tools that provide prospective insights regarding consumer credit from historical data have become invaluable.

Luckily, recent advancements in software, computational hardware, mathematical analysis, combined with the increased availability of big data, have made it possible to commodify machines that can learn to operate in a similar way to credit analysts and risk managers. Much research has been done over the past two decades into how ML can be used to measure and manage credit risk (this is detailed in Section 1.3).

The focus of this study is two-fold: Consumer credit scoring (CCS) and artificial intelligence (AI)/machine learning (ML) modelling techniques, specifically neural networks (NN)/deep learning (DL).

## 1.2 Motivation

There are many decisions involved and factors to consider when considering whether or not to extend a line of credit in the consumer lending business. These necessitate reliance on algorithms and models over the use of human discretion, as models and algorithms base decisions on rules and hard information, for example, the characteristics contained in a consumer's credit file that credit agencies have collected.

This is where credit scoring models play their role. *Credit scoring* is a discipline that was developed in the 1960s and has since been widely adopted and dubbed "the grease" that supports decision-making in countless businesses and banks around the world (Anderson 2007). However, until 2008 the volume of literature available in this vital field was scarce.

A credit scoring model is used to generate numerical scores to summarise the creditworthiness of a credit client/applicant. The issue is that, although these models are generally able to produce reasonably accurate ordinal measures, they fail to adjust quickly over time and can be quite insensitive to changing market conditions (Khandani et al. 2010). This failure is a significant drawback considering how quickly consumer credit can deteriorate. It has become apparent that there is a clear need from banks and regulators for more timely cardinal credit risk measures (Khandani et al. 2010). That is why we consider the use of ML in this dissertation.

ML is simply a collection of algorithms designed to address computationally demanding pattern recognition problems with enormous data sets. Consumer credit risk (CCR) analytics often deal with large sets of highly complex data. The possible relationships among transactional data and consumer characteristics are so convoluted that humans cannot analyse them. Some of the methods used in ML literature are ideally suited to this exact problem.

Another major challenge in consumer credit, especially in emerging economies, is that the cost of credit is often remarkably high. This high cost sometimes results in the financial exclusion of small borrowers such as households and SMEs. These small borrowers play a significant role in contributing to the prosperity and growth of a country (Sahay 2015).

Recent advances in computing technology, the boom in the availability of big data, and the FinTech industry's growth have made lending to these borrowers more accessible by reducing the cost of credit and thereby increasing financial inclusion. The AI/ML methods that lie at the heart of these FinTech lending solutions are not without their faults as they have mainly been considered a black box for regulators and a non-technical audience.

While the idea of machines with human levels of intelligence is not a new concept, the success in creating AI machines has been scarce over the decades since its proposal. AI mostly seemed to be a failed concept (Culkin & Das 2017) due to the lack of computing power required to realise its true potential. That is, until the aforementioned technological revolution. The field of AI/ML has gained new momentum, and in recent years they have come to dominate some industries and now, after many years of resistance from regulators, AI/ML is now moving from the research desk to the application stack. This new momentum has created a renaissance in computational modelling, of which CCR modelling is just one of many recent examples. Other examples include: credit card fraud detection (Maes et al. 2002, Awoyemi et al. 2017, Yee et al. 2018), derivative pricing (Culkin & Das 2017, De Spiegeleer et al. 2018, Antonov et al. 2020), algorithmic trading (Andersen & Mikelsen 2012, Colianni et al. 2015) and portfolio management (Agarwal et al. 2006, Yun et al. 2020) to name a few.

In this dissertation, we consider the use of NNs, which are a sub-field of ML (see Figure 1.1 for details of AI and all its general sub-fields fit together). In 1950s the first perceptron model was created based of work done by Rosenblatt (1958) combined with Hebb (1949)'s model of brain cell interaction and with Samuel (1959)'s ML efforts.

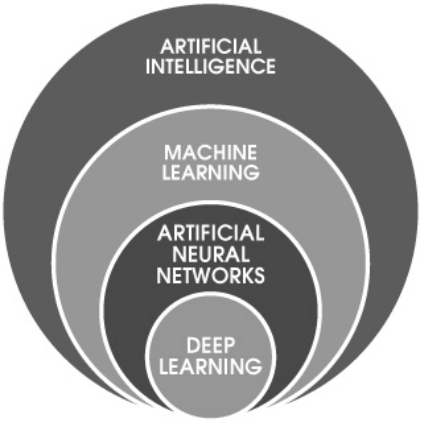



Figure 1.1: An illustration of how artificial intelligence, machine learning and neural networks fit together.

(Source: (Hauptfleisch 2016))

The perceptron (see Figure 1.2) was the first neural network and consisted of a set of input nodes (organised in a layer) which were all connected to an output node. Through a series of simple linear regression models and a specific training algorithm, the simple model can learn many types of relationships between input and output variables. Since then, the field has grown significantly. As a result, we have gained many different, consequential types of neural network architectures (each far more advanced than the perceptron), including DL architectures that include multi-layer perceptrons, convolutional neural networks, and recurrent neural networks. Each serves a specific role in solving different problems, from spam filters to time-series forecasting to image recognition and much more.

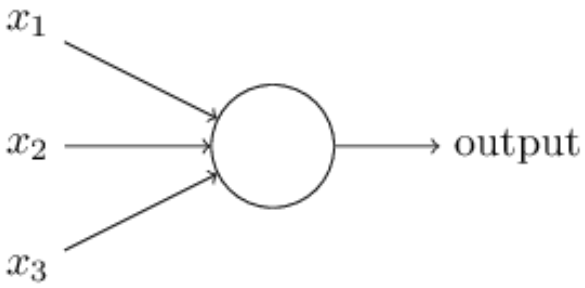



Figure 1.2: A simple perceptron model. Here the $x_i$s are the input to the model.

The adjective "deep" in deep learning refers to the use of multiple layers in the network as seen in Figure 1.3.

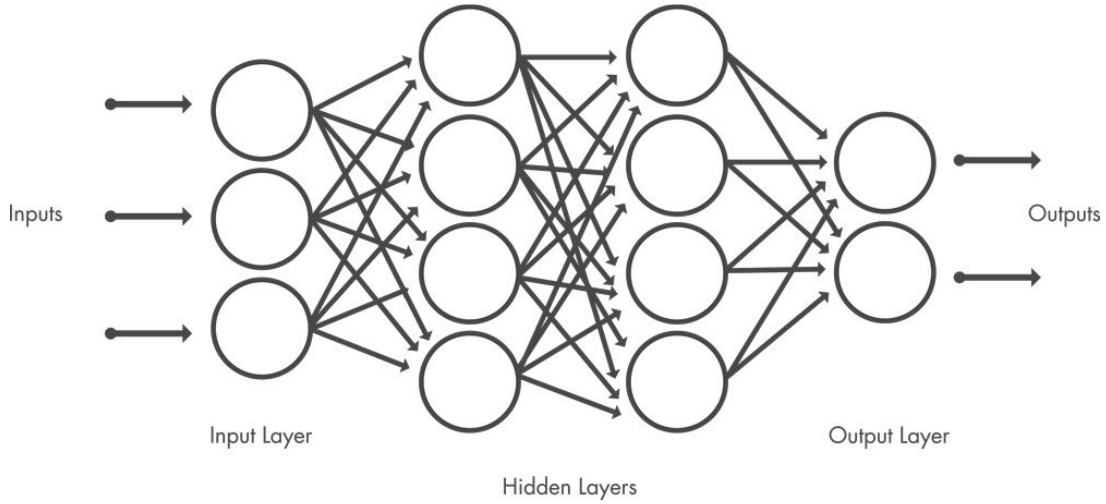



Figure 1.3: An illustration of a deep neural network.

(Source: (MathWorks n.d.))

Deep learning networks or deep neural networks (DNNs) are known to theoretically be powerful enough to approximate any function (Hornik et al. 1989) (i.e. learn/find an underlying relationship in a data set, should one exist). The greater availability of data has made it possible to showcase the advantages that NNs have over traditional ML and the advantage that DL has above both. The real power of DL can only be utilised once we have large amounts of data on which to train our model. Figure 1.4 gives an illustration comparing the performance of DNNs versus other methods as the amount of data available.adjective "deep" in deep learning refers to the use of multiple layers in the network as seen in Figure 1.3.

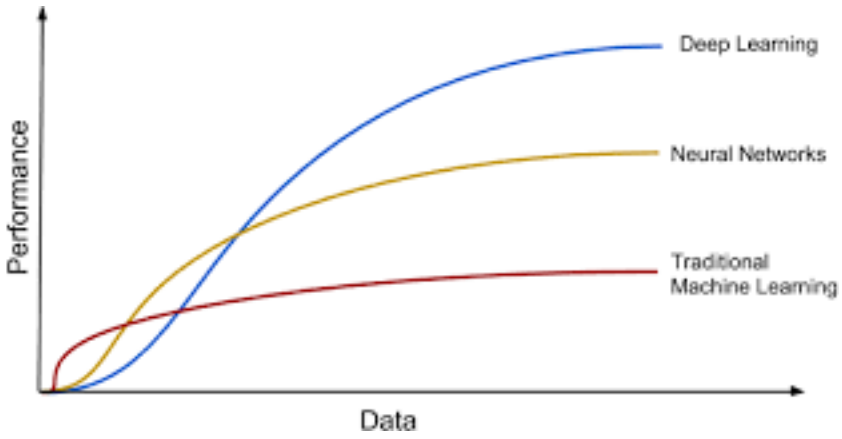



Figure 1.4: A comparison of the performance of DNNs, ANNs and Traditional ML as the amount of data available for training increases.

(Source: (Wasicek 2018))

Because of this potential power that DNNs have, we have selected them as our credit default prediction tool of choice. In this dissertation, we will put them to the test against other ML algorithms used in practice and proposed in the literature.

## 1.3 Literature Review

While we acknowledge that using ML in credit default prediction is no new research topic, we found that the literature provides mixed results. We wanted to put the power of DL to the test for ourselves by also utilising modern data science and ML techniques to obtain the best possible results.

There are a few different case studies that fall under the blanket term "credit default". These include consumer credit defaults (credit card delinquencies -our focus- and loan defaults), peer-to-peer lending loan defaults, commercial and corporate defaults (loan defaults and business failures). Discriminant and logit analyses have traditionally been the most popular approaches to predicting each of these situations (Gepp et al. 2010). However, there is also a range of promising non-parametric techniques in ML that can alternatively be applied. We briefly review some of the ML models used in the literature to provide some context for the rest of the dissertation. We will only provide a high-level overview of the models themselves, as they will be described in greater detail in Section 4.3. We will consider decision tree classifiers, support vector machines and artificial neural networks.

### 1.3.1 Decision Tree Classifiers

Decision trees, also referred to as classification and regression trees (CART), are simply sets of cascading questions that perform either classification or regression, depending on whether the response variable is categorical or numeric. With classification trees, we start with a data point (or, more specifically, a set of features and their values) and then use the value of the given feature to answer a question. By answering each question, the decision rule will then determine the next question. At the end of this sequence of questions, we will have a probability of the data point belonging to a specific class. Figure 1.5 is a simple but clear illustration of this process. This method will be described more completely in Section 4.3.1.

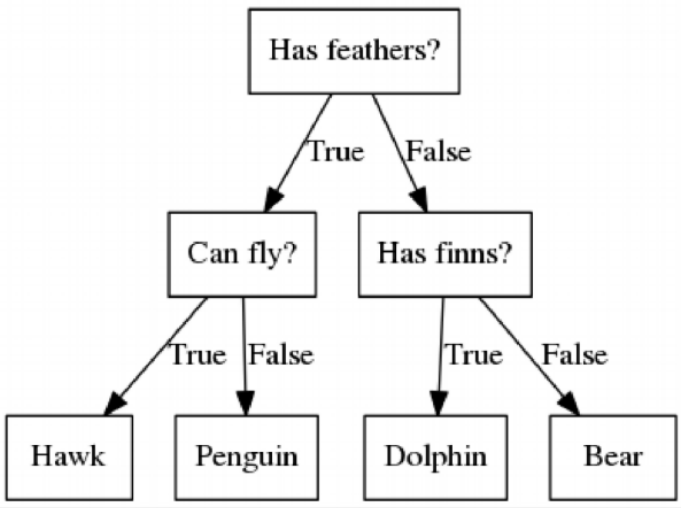



Figure 1.5: An example of a simple decision tree classifier.

(Source: (Chowdary 2020))

One of the earliest papers mentioning "decision trees" was by Belson (1959). Since then, decision trees have evolved significantly with the advent of big data and the ad-

vancement of computational power. Modern variations or adaptions of decision trees include random forests (Breiman 2001) (which is an application of bootstrap aggregation (Efron 1992) or bagging) and gradient boosted trees (GBTs) (Friedman 2001) (including algorithms such as XGBoost (Chen et al. 2015) and LightGBM (Ke et al. 2017)). These are examples of ensemble-based learning techniques where a set of weaker ML algorithms are grouped to achieve better results (ensemble methods are described in Section 3.4.2). These are beyond the scope of this research but are well worth looking into in the future.

In a similar way to Figure 1.5, decision trees can and have been used to determine the probability of a customer/business/corporate defaulting on their credit obligations based on a set of inputs.

These models have proven very powerful in this aspect. They have been widely studied in literature from as early as 2000 by Galindo & Tamayo (2000) to as recently as 2021 by Madaan et al. (2021).

To start with, we look at Galindo & Tamayo (2000) who found that CART models provided the best estimation for default with an average 8.31% error rate for a training sample of 2,000 records. The second was NNs with an average error of 11.00%, and the third was the K-Nearest Neighbour algorithm with an average error rate of 14.95%. The models above outperformed the standard probit algorithm, which attained an average error rate of 15.13%. Later, Lee et al. (2006) used CART models and multivariate adaptive regression splines (MARS) to demonstrate the effectiveness of credit scoring on banking credit card data using these techniques. Their results showed that CART and MARS outperform other methods such as logistic regression, NNs, discriminant analysis and support vector machines in terms of accuracy.

Looking more into modern variations of decision trees Bastos (2007) proposed a credit scoring model based on boosted decision trees. When benchmarked against the multi-layer perceptron and support vector machines, they showed that a boosted decision tree is a competitive technique for implementing credit scoring models when evaluated using credit card application data sets. Later work in consumer credit card default prediction was done by Khandani et al. (2010) who applied the CART models on a combination of customer transaction data and credit bureau data. They then constructed out-of-sample forecasts that significantly improve the classification rates of credit card holder delinquencies and defaults. In the same year, Gepp et al. (2010) suggested, based on their results, that decision trees could be superior predictors of business failure as compared to discriminant analysis.

More recently, Chang et al. (2016) integrated bootstrap aggregating (Bagging) with a synthetic minority over-sampling technique (SMOTE) into a credit risk model to improve the decision tree stability and its performance on unbalanced data. The results of their experiment on a real-world case of SME loan data from Taiwan showed that their proposed model's classifying recall rate and the precision rate was superior to the logistic regression and Cox proportional hazards models. Later, Sayjadah et al. (2018) provided a performance evaluation of credit card default prediction. Comparing logistic regression, decision trees, and random forests, they found that random forest proved to have the higher accuracy and area under the curve. are used to test the variable in predicting credit default, and random forest proved to have the higher accuracy (82%) and area under the curve (77%).

Chang et al. (2018) used eXtreme gradient boosted trees (XGBoost) to build a credit risk assessment model for financial institutions. They used cluster-based under-sampling to process imbalanced data and the area under the receiver operative curve, and the accuracy of classifications to assess their model's performance. When compared to logistic regression and support vector machines, they found that the XGBoost classifier achieved

better results than the other models and can serve as a superior tool for developing credit risk models for financial institutions. Similarly, Zhou et al. (2019) applied gradient boosting decision trees (GBDT), XGBoost and light gradient boosting machine (LightGBM) to create an ensemble learning-based default prediction model to predict defaults in peer-to-peer lending. In comparison with benchmark models, they found that their prediction model achieved desirable results and could effectively solve the challenge of predictions based on high-dimensional and imbalanced data, as is familiar with peer-to-peer lending data.

Furthermore, Golbayani et al. (2020) compared bagged decision trees, multi-layer perceptrons, support vector machines and random forests when trying to predict corporate defaults. Their results show superior performance for decision tree-based models.

### 1.3.2 Support Vector Machines

A support vector machine (SVM) is a robust ML algorithm that attempts to classify points in a feature space by finding a hyperplane that separates these points based on the value of the response variable. Figure 1.6 gives a simple illustration of this, but this method will be described more thoroughly in Section 4.3.2.

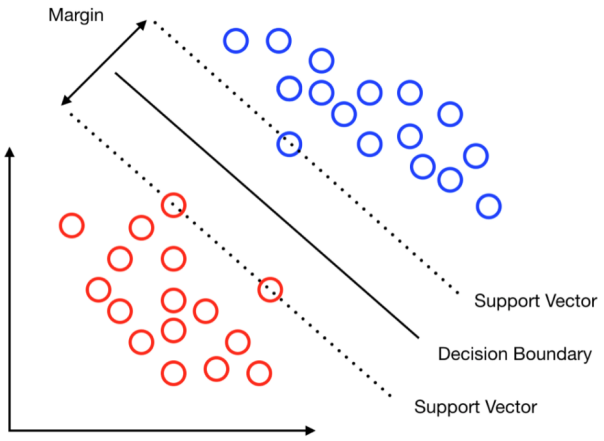



Figure 1.6: A simple illustration of a support vector. In this example our feature space is 2D and the axes represent the values of our two features.

(Source: (iUnera n.d.))

The use of this algorithm has been prevalent in literature. It has delivered impressive results against other popular algorithms. The groundwork for the algorithm was proposed by Vapnik & Chervonenkis (1964). SVMs close to their current form was first introduced by Boser et al. (1992) and the current standard version we use today was introduced by Cortes & Vapnik (1995).

As far back as 2003, Van Gestel et al. (2003) experimented with the use of least squares support vector machines, which at that stage was a newly modified version of SVMs. They reported significantly better results when contrasted with the classical techniques. Baesens et al. (2003) who studied the performance of various modern classification algorithms (including logistic regression, discriminant analysis, NNs, SVMs and decision trees) reported the same results in favour of SVMs.

Soon after Shin et al. (2005) and Min & Lee (2005) investigated SVMs when applied to bankruptcy prediction problems. They showed that their approach outperformed NNs

in the problem of corporate bankruptcy prediction. They also showed that their models' accuracy and generalisation performance were better than that of NNs as the training set size decreases.

Later Li et al. (2006) developed a consumer loan application evaluation model using SVMs. In addition, they also developed a visual decision-support tool to be used in loan application evaluation. Their tests using real-world data concluded that SVMs surpass traditional neural network models in generalisation performance. The visualisation via the tool mentioned above helps decision-makers determine appropriate loan evaluation strategies. Adding to the expansion of interpretability of SVMs, Martens et al. (2007) use rule extraction techniques on SVMs to increase human interpretability. They test these techniques on several classification problems, including credit scoring on publicly available data sets. They conclude that rule extraction techniques sacrifice only a small percentage in performance and therefore rank at the top of comprehensible classification techniques.

Using the data sets from UCI Machine Learning Database, Huang & Scott (2007) compared the use of NNs, genetic programming, and decision tree classifiers with an SVM classifier. They showed that the SVM achieved an identical classification accuracy to the other models with relatively few input features. Additionally, they combined genetic algorithms with an SVM classifier in the hybrid GA-SVM strategy. Experimental results showed that SVMs are a promising addition to existing data mining methods.

Bellotti & Crook (2009) tested SVMs against methods such as discriminant analysis and logistic regression using an extensive credit card database. Their findings showed that SVMs are competitive and can be used to identify the most statistically significant features in determining default risk. Shortly after, Kim & Sohn (2010) provided an SVM model for predicting the default of funded SMEs in Korea by considering various input variables such as financial ratios, economic indicators, and technology evaluation factors. Their results showed that their SVM model outperformed back-propagation NNs and logistic regression when measuring the accuracy of predictions.

Moreover, recently Khemakhem & Boujelbene (2017) performed a comparative study of logistic regression, artificial NNs and SVMs when used to determine the probability of default as a tool to measure credit risk in a Tunisian commercial bank. Their results show that SVMs using the radial basis function outperformed the other models.

### 1.3.3 Artificial Neural Networks

NNs or artificial neural networks (ANNs) are powerful ML algorithms that can perform regression or classification tasks. Their use in credit default prediction has been widely studied. With non-payment predictions, Charitou et al. (2004) suggested that the logit models are superior when compared to other methods. However, it was also suggested that traditional ANNs and Logit models are equally superior in overall predictive ability. Koh et al. (2006) noted that logistic regression is the most stable model despite ANNs having the best overall accuracy rate. Vojtek et al. (2006) suggested that even though the AI-based methods such as ANNs can perform with missing values and multi-collinearity issues, the processes are mathematically demanding, and some techniques are complicated to elaborate. Finally, even though ANNs can create fantastic results, their failure to give details of the outcome is a real drawback (Eddy & Abu Bakar 2017). As a summary, we will look at some of the other literature that put NNs to the test against some non-traditional models.

As early as 1992, Jensen (1992) studied the use of back-propagation NNs to predict loan defaults based on credit applicant characteristics, achieving an accuracy of up to

80%. Some time later West (2000) found that multi-layer perceptrons outperformed methods such as linear discriminant analysis, logistic regression, k nearest neighbour, kernel density estimation, and decision trees in credit scoring applications. Soon after, Atiya (2001) developed a neural network model for bankruptcy prediction and proposed novel indicators for the model. Their results show that these indicators and traditional financial ratios provide significant improvement in the out-of-sample prediction accuracy.

Further down the line, Angelini et al. (2008) developed both a standard feed-forward network and a network with special-purpose architecture. They tested their models on real-world data (Italian small business loans). They showed that NNs, with careful data analysis and data pre-processing, successfully learn and estimate the default tendency of a borrower. Khashman (2010) and Eletter et al. (2010) both proposed NNs to decide whether to approve or reject a credit application. Their results suggest which neural network model with which learning scheme delivers the most optimal performance.

Exploring DL, Sun & Vasarhelyi (2018) developed a credit default prediction system using a deep neural network. They used credit card data from customers of a large bank in Brazil. Similarly, Wang et al. (2018) also used DL to predict loan defaults in the peer-to-peer lending sector. Motivated by the research in natural language processing, the authors used online operation behaviour data of borrowers. They proposed a consumer credit scoring method based on an attention mechanism called a long-short term memory (LSTM) network (a type of recurrent neural network), a novel application of DL. Their results showed that the proposed solution could effectively increase the predictive accuracy compared with the traditional models.

More recent work by Hsu et al. (2019) developed a DL model, in the form of a recurrent neural network (RNN), to predict credit card client defaults. Numerical experiments confirmed that the RNN model provides the best performance compared to the other benchmark models. Finally, Giudici et al. (2019) proposed to enhance credit risk accuracy of peer-to-peer platforms by leveraging topological information embedded into similarity networks derived from borrowers' financial information. Topological coefficients describing borrowers' importance and community structures are employed as additional explanatory variables, leading to the improved predictive performance of credit scoring models.

## 1.4 Goals of the Dissertation

It is necessary to give the reader a clear indication of the goals of this dissertation. We outline the three primary goals as follows:

The **first goal** of this dissertation is to explore the topics of credit risk/credit scoring as well as introduce the unfamiliar reader to the concept of ML. We will tie these two topics together and review credit scoring methods and models used in practice today and proposed in the literature. The main focus of this dissertation is NNs and DL, so once the basics of ML are covered, we want to explore NNs and the mathematics that underpin them.

The **second goal** of this dissertation is to develop an ANN model for consumer credit default prediction. We will use actual consumer credit card data to train the model, and we will then test this model against other credit scoring models. We will also build a DNN while also using recent advancements to improve accuracy and reduce overfitting. Compared to other ML models such as logistic regression, naive Bayes, decision trees and traditional ANNs, DNNs have been shown to have a better overall predictive performance. They have produced the highest F1 scores and area under the receiver operating characteristic curve (Sun & Vasarhelyi 2018).

The **third and final goal** is to tune our chosen ANN and DNN and see if we can

outperform the other models currently proposed in the literature by running rigorous tests.

## 1.5 Structure of the Dissertation

Here we provide an outline of this dissertation with a short description of the contents of each chapter. This description is supplemented by Figure 1.7, which gives the reader a clear indication of how each chapter fits together.

### Part I - Theory

#### Chapter 2: Consumer Credit Risk and Scoring

This chapter will look at credit scoring by introducing the concept of credit risk and, more specifically, consumer credit risk. We will go over some of the basic concepts related to credit risk, such as measurement and management, and other related concepts, such as risk-based pricing. We will then introduce scorecards and the rationale behind them. This chapter will be crucial for the reader unfamiliar with finance and credit risk.

#### Chapter 3: A Brief Overview of Machine Learning

This chapter will be a gentle non-technical introduction to ML and its various sub-fields. This chapter will provide some background on the field and give a slightly more comprehensive view than is necessary for the dissertation. However, ultimately it will open up the discussion to the power of NNs and DL.

#### Chapter 4: Credit Scoring Methods, Models and Evaluation

We will review the various methods and models that have been developed in the field. We will also look at various metrics used to evaluate our models and review some techniques from Data Science to increase the predictive ability of our models. This chapter illustrates the cross-field nature of this dissertation and brings together concepts from Chapters 2 and 3.

#### Chapter 5: Mathematics of Neural Networks and Deep Learning

Here we start to build an understanding, in a mathematical sense, of the tools used in this dissertation. We start with some background to the field and construct a neural network model by starting with a single perceptron model and eventually arriving at a deep neural network. It is crucial to note that this chapter will focus on the mathematics of the model as it is required, along with a technical understanding of the model, to prove in theory that it is possible to build an accurate credit scoring model. More importantly, it will also provide a better understanding of how NNs are so superior at function approximation.

#### Chapter 6: Challenges and Advancements in the field of Neural Networks and Deep Learning

This final chapter of the theoretical background takes a look at recent advances in the field of DL. We also see how they address some of the issues with training deep networks. We explore various methods designed to increase the accuracy, execution speed and reliability of NNs.

## Part II - Application

### Chapter 7: A Neural Network Model for Credit Default Prediction

This chapter starts by looking at our data set and exploring what features are available to us. We then proceed to "fix" the class imbalance issue in our data set and then prepare our data set for training. We will then describe the experiment design and examine the five models we will be training and comparing. The ultimate goal is to create a highly accurate Deep Neural Network. We then review and compare the results from our experiments and analyse each model's performance in different aspects and rank them accordingly.

### Chapter 8: Conclusion and Suggestions for Future Study

This final chapter concludes whether DL is an excellent alternative to conventional and other credit scoring models. We also highlight some of the drawbacks and challenges that DL approaches face in the financial sector (especially in banking, where regulatory hurdles are plenty) and consider some resolutions. We also provide an overview of the potential future research options related to this work.

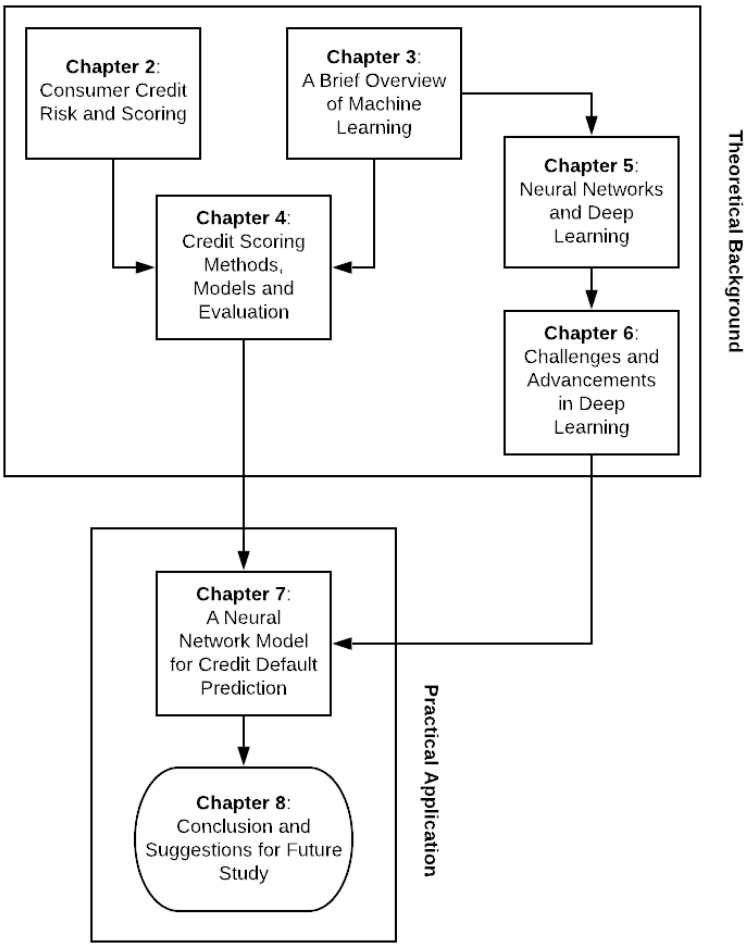



Figure 1.7: How each chapter ties together.

# Part I

# Theoretical Context

# Chapter 2

# Consumer Credit Risk and Scoring

> "Pay to all what is owed to them: taxes to whom taxes are owed, revenue to whom revenue is owed, respect to whom respect is owed, honour to whom honour is owed." - *Romans 13:7*

## 2.1 Introduction

In this chapter, we will look at credit scoring by introducing the concept of credit risk and, more specifically, consumer credit risk. We will consider some basic concepts related to credit risk, such as measurement and management, and other related concepts, such as risk-based pricing. We will then introduce scorecards and the rationale behind them. This chapter will be crucial for the reader unfamiliar with finance and credit risk.

## 2.2 Background

Since the GFC, banks have increased effort to understand their total credit risk profile better. This increased effort was followed by much attention being placed on counterparty credit risk (which is defined as the likelihood or probability that one of those involved in a transaction might default on its obligation), as is highlighted in Figure 2.1*.

*This data was queried and extracted from https://app.dimensions.ai/analytics/publication/overview/timeline. The results were obtained by running a search across all major journal publications, from 2005 to 2020, for papers where the title or abstract mentioned "counterparty credit risk" and "consumer credit risk" or "retail credit risk".

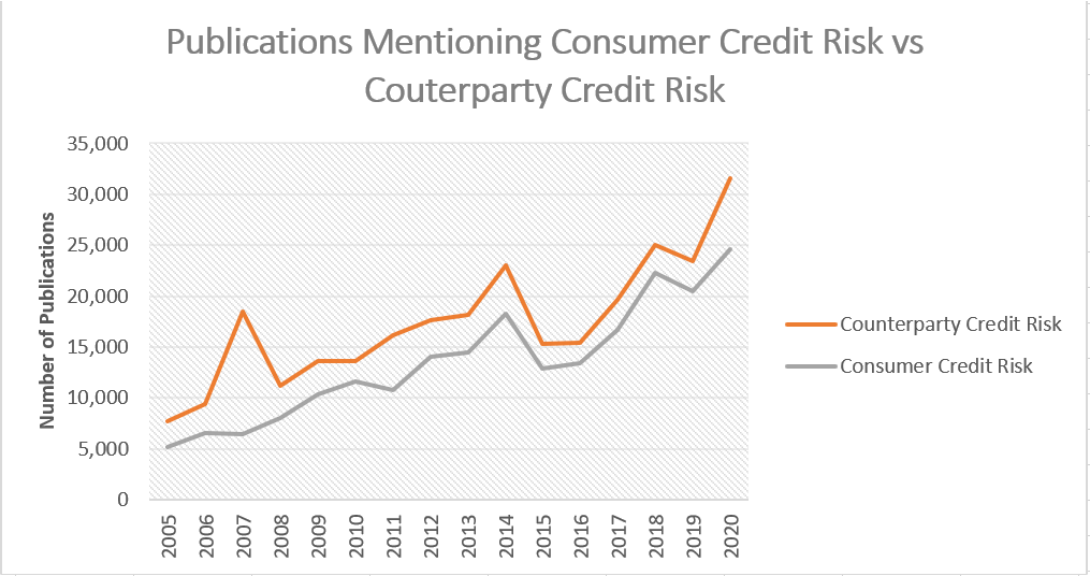



Figure 2.1: On average, over the last 15 years, counterparty credit risk has been studied in 40% more papers than consumer credit risk.

However, it is apparent that **consumer credit risk** also deserves attention and stands to benefit from recent technological innovation. In this dissertation, we will be exclusively focusing on credit scoring in the context of consumer credit risk and not in counterparty credit risk.

## 2.3 What is Credit Risk?

The Bank for International Settlements (1999) defines credit risk as:

> **Definition 1** *Credit risk is the potential that a borrower or counterparty will fail to meet its obligations per agreed terms.*

UniCredit (2012) distinguish between three types of credit risk:

- **Credit default risk**, which is the risk of a borrower is unable to repay their debt obligations in full or is past due more than 90 days on their credit obligation leading to financial loss for the lender.
- **Concentration risk**, which is the risk that arises in credit portfolios when borrowers all face common risk factors (such as exchange rates and interest rates). This could be an issue for the lender since they affect both the borrower's ability and willingness to meet their debt obligations which could amount to large simultaneous losses.
- **Country risk**, which is the risk of loss arising due to sovereigns failing to meet foreign debt obligations (be it intentional or not); this type of risk is associated with a country's macroeconomic conditions as well as political stability.

We will be concerned with credit default risk. According to De Laurentis et al. (2010), default risk can be broken down further into three sub-categories:

- **Default Risk**, which relates to a borrower's inability to make promised payments and is measured by **probability of default** (PD), which can be determined using various methods.

– **Recovery Risk**, which is the risk that the recovered amount, in the event of default, is less than the full nominal amount that is due. It is measured using **recovery rate** (RR), which is the extent to which principal and accrued interest on defaulted debt can be recovered, expressed as a percentage of face value. It is the inverse of **loss given default** (LGD), which is the share of principal and accrued interest that is lost in the event of a default. LGD is simply 1 – RR.

– **Exposure Risk**, which is the risk that a credit exposure at the time of default increases relative to its current exposure. It is measured by **exposure at default** (EAD).

The key parameters mentioned above - that is PD, LGD and EAD - are defined by The Bank for International Settlements (2003) as follows:

**Definition 2** *Probability of default measures the likelihood that the borrower will default over a given time horizon.*

**Definition 3** *Loss given default measures the proportion of the exposure lost if a default occurs.*

**Definition 4** *Exposure at default, which for loan commitments measures the amount of the facility that is likely to be drawn if a default occurs and is determined as follows:*

$$EAD = drawn\ amount + (limit - drawn\ amount) \times LEQ,$$

*where:*
*drawn amount is the amount of the credit facility currently used,*
*limit is the maximum amount granted by a bank to the borrower and*
*LEQ is the loan equivalency factor (rate of usage of available limit beyond ordinary use).*

## 2.4 Credit Risk Measurement and Management

The most important concepts in understanding the management and measurement of credit risk are expected loss, unexpected loss and economic capital.

### 2.4.1 Expected and Unexpected Losses

First we consider the definition of expected loss as presented in Schroeck (2002):

**Definition 5** *Expected loss (EL) is the mean loss in the long run generated from credit facilities. The EL is calculated as*

$$EL = PD \times LGD \times EAD. \tag{2.1}$$

EL is determined based on expectations and is a cost that is incorporated into business and credit decisions (Schroeck 2002). Credit losses are not constant over time. The business normally provisions for EL and attempt to bear it as part of normal operating cash flows. However, EL might differ from actual losses. Finally, the EL of a portfolio $P$

is equal to the sum the individual $EL_i$s,

$$EL_P = \sum_i PD_i \times LGD_i \times EAD_i, \tag{2.2}$$

where $i$ is the number of obligors.

Next, we consider the definition of unexpected loss as presented by Ebrary.net (nd.):

**Definition 6** *Unexpected losses (ULs) are loss percentiles ($L^\alpha$) in excess of the EL, i.e. the additional loss beyond the EL and up to the loss percentile used for defining the credit value at risk ($CVaR^\alpha$).*

$$CVaR^\alpha - L^\alpha - EL. \tag{2.3}$$

*Unexpected losses do not include exceptional losses beyond the loss percentile defined by a prescribed confidence level $\alpha$ (established by an institution's risk tolerance or - in compliance terms - by regulatory authorities, such as the Basel Committee on Banking Supervision) (See Figure 2.2).*

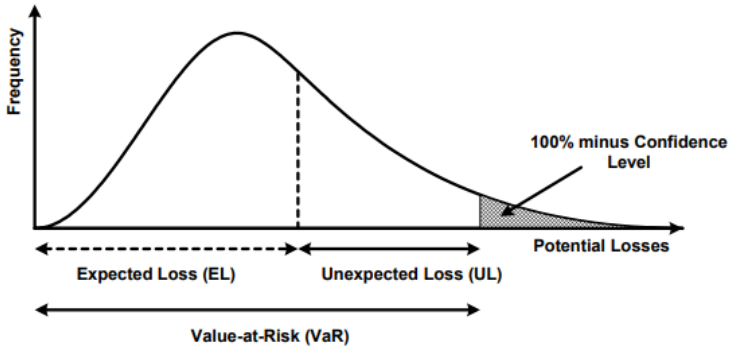



Figure 2.2: An illustration of expected loss and unexpected loss using a credit loss distribution curve.

(Source: (Genest & Brie 2013))

For a single exposure UL is calculated as follows (Schroeck 2002, Farid 2014):

$$UL = EAD \times \sqrt{PD \times \sigma^2_{LGD} + LGD^2 \times \sigma^2_{PD}}, \tag{2.4}$$

where

$$\sigma^2_{PD} = PD \times (1 - PD). \tag{2.5}$$

It is essential to point out some assumptions made here. In Eq. (2.4) we assume that PD and LGD are uncorrelated. Also, in Eq. (2.5) we assume that a default event is a random variable with a Bernoulli distribution with parameter PD, hence $\sigma^2_{PD}$ is simply the variance of a Bernoulli random variable.

Finally, if we had to calculate the UL for a portfolio $P$ we would use

$$UL_P = \sqrt{\sum_i^n \sum_j^n \rho_{ij} UL_i UL_j}, \tag{2.6}$$

where $\rho_{ij}$ is the correlation between the PD of asset $i$ and asset $j$.

Credit losses, both expected and unexpected, are very important to measure, estimate and monitor. ULs present a problem because they can jeopardise the viability of a bank as a going concern and, if not managed correctly, can grow. To cover itself against these ULs, a bank must hold sufficient capital. Capital is built up in good times from profits and is designed to absorb ULs in stressed times. Banks can derive PDs from clients' credit scores/ratings and use internal or regulatory models to estimate clients' LGD and EAD. These are important in determining the overall capital contributions needed by banks.

### 2.4.2 Economic Capital

Credit risk events can cause catastrophic economic losses for the bank. As soon as the bank has calculated its EL on its loan portfolio ($EL_P$), it must set aside reserves in anticipation to absorb these losses. On the other hand, to guard against the ULs it is estimating at a predetermined confidence level, the bank needs to hold additional capital over and above normal reserves, known as economic capital.

**Definition 7** *Jones & Mingo (1998) defines economic capital (EC) as the level of capital a financial institution needs to hold in order to cover its losses at a certain probability or confidence level, which is related to the desired rating. It is calculated as the difference between expected loss and unexpected loss at a specific confidence level ($\alpha$):*

$$EC = UL_P^{\alpha} - EL_P. \tag{2.7}$$

Figure 2.3 gives a good illustration of the EL, UL and EC by describing it on a credit loss distribution curve. For a more mathematical explanation of how EC is calculated, consider the following derivation by Schroeck (2002):

Let $X_T$ be a random variable representing the loss and let $z$ be the confidence level. Also, let $v$ be the minimum EC required to keep the bank solvent over horizon $t$, then:

$$P(X_T \leq v) = z. \tag{2.8}$$

Now, given the desired confidence level of $z$ we can determine the amount of EC to hold as

$$P(X_t - EL_P \leq EC) = z, \tag{2.9}$$

we define the capital multiplier $C$ as

$$C = \frac{EC}{UL_P}, \tag{2.10}$$

and then

$$P\left(\frac{X_T - EL_P}{UL_P} \leq C\right) = z. \tag{2.11}$$

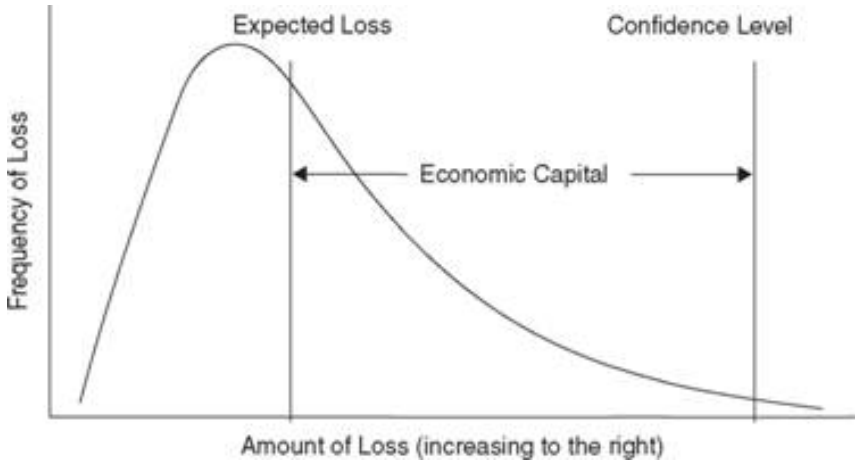



Figure 2.3: An illustration of expected loss and unexpected loss and how economic capital is determined.

(Source: (Ebrary.net nd.))

### 2.4.3 Credit Loss Distributions

When we want to model credit risk, we are primarily concerned with the distribution's right tail. It is also important to note that the normal distribution is not appropriate here as credit losses are highly skewed. Also, credit losses are limited below at zero, and although extreme losses can happen, they are quite rare. It is because of this that the beta distribution used to be the favoured distribution (Schroeck 2002).

Today, the Vasicek distribution, pioneered by Vasicek (2002), is the preferred distribution to model portfolio credit losses. This distribution was adopted by Basel (Chatterjee 2015), and provides more accurate results compared to the Beta distribution.

## 2.5 Consumer Credit Risk

**Definition 8** *Consumer credit risk (CCR) or retail credit risk, is the risk of default by a consumer on a consumer credit product.*

CCR is one of the most significant risks faced by retail banks (Crouhy et al. 2006, Ghosh 2012). The retail banking industry primarily focuses on taking deposits from clients and then lending those deposits to consumers and SMEs. When we use the term loan, we are referring to lines of credit which include home mortgages, personal loans, vehicle finance, small business loans and revolving credit facilities like overdrafts and credit cards, to name a few. From the bank's perspective, these individual loans or lines of credit together form an extensive portfolio designed to reduce the incremental risk to anyone exposure.

In the years leading up to the GFC, banks offered customers products they could not afford with risks that were more than what customers could bear (Ellis 2008). Loan-to-value (LTV) ratios (where $LTV = \frac{\text{Mortgage Amount}}{\text{Property Value}}$ (Hayes 2020)) on mortgaged properties were very high and borrowers with weaker credit were given mortgages (Goodman & Zhu 2018). These strategies backfired when housing prices collapsed, which resulted in mortgages often exceeding the value of the properties themselves (Ellis 2008).

As mentioned before, when measuring CCR, we require estimates of PD, LGD, and

EAD. When each line of credit is issued, the bank will have to determine PD for each customer, not only at the outset but also throughout the life of the credit product, to understand and effectively manage their risk. PD is difficult to estimate, but De Laurentis et al. (2010) provides some methods that are used to determine a customer's PD:

- Analysing historical default frequencies of a borrower;
- Using mathematical and statistical tools to measure risk on an *ex ante* basis (i.e., before an event);
- Using a hybrid approach that combines mathematical and judgmental analyses. The mathematical results are generated automatically and are corrected using qualitative analysis.

Default risk is typically measured over one year. However, measuring cumulative probabilities of default beyond one year is also essential (De Laurentis et al. 2010).

### 2.5.1 Corporate Credit vs Consumer Credit

There are several differences between consumer and corporate credit risk. The first noticeable distinction is the size of the exposures. First, let us define corporate credit risk.

**Definition 9** *Corporate credit risk, is the risk of default by a corporate entity on corporate credit products, such as loans, bonds, revolving credit facilities, etc.*

Retail portfolios are mainly comprised of many small exposures, while corporate portfolios are made up of a few significant exposures. A single default in the retail portfolio has little impact on the lender's portfolio, whilst a single default in the corporate portfolio can have far-reaching economic consequences.

Because of the inherent diversification of a retail credit portfolio and its behaviour in regular markets, estimating its default percentage allows a bank to effectively treat this loss as a cost of doing business and factor it into the prices that it charges its customers (Crouhy et al. 2006).

Banks will have the opportunity to take pre-emptive action to reduce retail credit risk when it appears that customer behaviour is changing and there is a potential rise in defaults. This is because retail portfolio ELs are calculated based on PDs that the bank calculates internally based on its models so that ratings can be adjusted quickly (The Bank for International Settlements 2001). Pre-emptive actions could entail marketing to lower risk customers and raising interest rates for riskier customers (risk-based pricing). This signalling usually does not occur with corporate portfolios, as most banks rely on external ratings (Santos 2009) produced by rating agencies (which may take time to adjust their ratings). The bank might only detect problems when it is too late (Crouhy et al. 2006).

## 2.6 Credit Pricing, Monitoring and Other Considerations

### 2.6.1 Risk Based Pricing

More lenders have moved to risk-based pricing (RBP) and away from charging a single price for a product to all customers as they have recognised that this may lead to adverse

selection (Wolff 2020). RBP involves lenders charging different customers different prices based on their risk profiles. RBP is often used in the retail financial sector for credit cards, home mortgages, and auto loans (Luthi 2018).

Along with credit scoring, lenders use product and customer profit scoring measures to evaluate the potential profitability of a specific product and the potential profitability of a specific customer. According to Crouhy et al. (2006), in determining the price to charge a borrower, the following key factors are considered:

- the probability that the customer will accept the product,
- the client's credit risk profile (i.e. PD, LGD and EAD),
- the cost of capital (interest rate) and whether the lender can bear it,
- cost of capital that needs to be allocated to the transaction and
- operating expenses.

Prices are split into tiers, and allocation is based on credit score bands. To provide an example of credit score bands, we will look at FICO scores (a credit scoring system used mainly in the US) which are split into the following bands (Akin 2020):

- scores between 800 and 850 are exceptional,
- scores between 740 and 799 are very good,
- scores between 670 and 739 are good,
- scores between 580 and 669 are fair and
- scores between 300 and 579 are fair,

The lender can then map pricing strategies to metrics such as profit/loss, revenue, market share, and risk-adjusted return at the various score bands. Effectively utilising RBP allows senior management to evaluate the inevitable trade-offs among profitability, market share, and risk with the short and long-term goal of increasing shareholder value (Crouhy et al. 2006).

### 2.6.2 Importance of Credit-Line Risk Management

When a bank has extended a line of credit to a customer, they must then actively monitor the customer's creditworthiness. Whether the customer's creditworthiness improves or deteriorates, their line of credit may be increased or decreased, and their interest rate may be lowered or raised. Therefore, CCS and ongoing monitoring should be the primary concern of any retail credit business. Decisions on whether a credit line is to be extended or not should therefore be based on a model. This will ensure that changes in risk profiles are captured accurately and promptly and enable the bank to take corrective action if needed, e.g. for high-risk customers taking action by reducing credit limits, raising interest rates, or even both.

### 2.6.3 Trade-off between Profitability and Creditworthiness

A bank's primary concern is not just risk and creditworthiness; they also base their decision making on profitability. For example, suppose a credit card is issued to a customer with a very high credit score (i.e. low risk of default). They settle their debt in full each month or utilise a minimal amount of their credit line. In that case, the bank is going to

earn little to no interest from that customer. On the other hand, if the bank issues that same credit card to a customer with a low credit score (i.e. greater risk of default), they may earn more interest and face the risk that the customer will default. In this default event, they might not be able to recover the capital amount in full.

While it is challenging for banks to manage this trade-off efficiently, it is even more challenging to correctly identify customers' risk profiles, i.e. determining which customers are genuinely high-risk and are in danger of falling behind on repayments. If done incorrectly, many low-risk customers could be incorrectly classified as high-risk (opportunity cost) or many high-risk customers incorrectly classified as low-risk (greater subsequent delinquencies and losses).

This trade-off is common and is present in all classification problems. It is the statistical trade-off between type I and type II errors. In the context of CCR management, this trade-off represents a cost/benefit analysis of false positives (FP) or type I errors vs false negatives (FN) or type II errors. By adjusting the threshold, we can attempt to optimise some criterion function in which such costs and benefits are inputs (Khandani et al. 2010).

## 2.7 Credit Scoring

It is crucial to determine a customer's PD. It enables the lender to price the product offered correctly and enables them to hold the correct amount of capital to compensate for the risk they are taking on. A credit scoring model takes information about an applicant, typically from a credit scorecard, and outputs a number to be used for assessing credit risk. It is simple: a high number means a low probability of default by the borrower. Credit scoring models process large amounts of customer information and produce a single risk measure, all in a single automated process.

Due to recent advancements in computing technologies and the availability of affordable computing power, credit scoring has become the cornerstone in modern credit risk management (Mashanovich 2017). Now, banks and lenders, especially micro-lenders and SME financiers, can not only apply credit scoring in assessing lending decisions, but they can also now use it as a tool to perform ongoing credit risk management and collection strategies (Khandani et al. 2010) (credit-line risk management).

Credit scorecards are used to gather information from credit bureau reports and customer applications. Weights are then assigned to each element of the scorecard depending on their importance. Some of the questions/entries can include any of the following: Size and source of income, education, employment record and status, age, residential address, monthly expenses, marital status and number of dependents (Mashanovich 2017).

The attributes are the responses to these questions. The scoring models will then determine if these attributes are positive or negative and assign a weight to each attribute based on historical data points and the associated probability of default. In addition to these questions, each individual's credit file contains the following information (Equifax nd.):

- Personal information (which is not factored into the scoring model);
- Records of credit inquiries when a file is accessed;
- Data on collections, reported by entities that provide credit or agencies that collect outstanding debts;
- Legal (public) records on bankruptcies, tax liens, and judgments;

– Account and trade line information gathered from receivables information sent to credit bureaus by grantors.

## 2.8 Conclusion

The idea of consumer credit scoring is simple: the lender wants to know whom they are lending money to know their risk. Beyond that, a borrower's circumstances change, so it is clear that ongoing monitoring of risk is necessary. Finally, now that it has been established that credit scoring is so valuable, we can see it is clear that powerful and highly accurate tools are required to help measure and monitor this risk.

We have formed a clear understanding of the concepts of CCR. We now look at the tools used in practice and others that have been proposed in the literature. To achieve this, it is necessary first to get acquainted with the concept of ML. The next chapter provides a gentle introduction to the concept of ML for the unfamiliar reader.

# Chapter 3

# A Brief Overview of Machine Learning

> "Machine learning uses patterns in data to label things. Sounds magical? The core concepts are actually embarrassingly simple. I say "embarrassingly" because if someone made you think it's mystical, they should be embarrassed."
> - *Cassie Kozyrkov, Chief Decision Scientist at Google.*

## 3.1 Introduction

This chapter provides a gentle non-technical introduction to machine learning and its various sub-fields. This chapter will provide some background on the field and give a slightly more comprehensive view than is necessary for the dissertation. However, ultimately it will prompt a discussion of the power of neural networks and deep learning.

## 3.2 What is Machine Learning

Machine learning, which is a sub-field of AI, enables machines (computers) to learn by themselves. By using different algorithms, the machine can sometimes identify complex patterns in observed data sets and then build models that attempt to explain the natural world and make predictions without having explicit pre-programmed rules and models (Maini & Sabri 2017).

Using a combination of mathematical and statistical models and sets of data, we attempt to fit a suitable model to a set of data. We attempt to build models, using observed data, that are ultimately able to make predictions using new data, i.e. that generalise well. An elementary example is linear regression, where we attempt to fit a straight line to data. However, ML problems frequently deal with more complex models.

The notion of ML is not new. In the 1950s, Samuel (1959) created a computer program that could play checkers and also designed mechanisms that allowed his program to improve (or learn). Arthur Samuel is credited with coining the phrase "machine learning" in 1952 (Foote 2019).

ML, and AI in general, had initially been seen as a failure (due to the lack of necessary computational power), and it is only in the last few decades that we have seen a resurgence in the field (Culkin & Das 2017). This resurgence was primarily due to advances made in computer processing power and the availability of Big Data. ML can be divided into classical ML, reinforcement learning, ensemble methods and neural networks/deep learning.

## 3.3 Classical Machine Learning

Classical ML can be split into two fields: Supervised and unsupervised learning.

### 3.3.1 Supervised Learning

In this case, the machine is given labelled data, i.e. the input set with its respective output. The machine then uses these training examples (instances) to learn and identify relationships. The two main problems of supervised learning are:

- **Classification:** When the output variable is a category. Using models like Logistic Regression, Decision Trees, K-nearest Neighbours, SVMs.
- **Regression Problems:** When the output variable is a real value. Using models like Linear Regression, Polynomial Regression, Ridge/LASSO regression.

To illustrate how supervised learning works, let us consider, for example, the simple linear regression model

$$y = \beta x + \epsilon. \tag{3.1}$$

When training the machine (also referred to as learning the model), we will provide it with an input $x$ and an output $y$. Given enough labelled data, the supervised learning algorithm can approximate the relationship (should one exist) between the inputs and outputs, i.e. approximate $\beta$ and $\epsilon$ (this can is done using various optimisation methods). After a model has been trained, it can make predictions for $y$ based on unseen $x$ values. If the model's predictions are reasonably accurate (close to the actual $y$), we have a model which generalises well.

### 3.3.2 Unsupervised Learning

In this case, the machine is presented with unlabelled data and tasked with sorting them and finding underlying patterns. The two main problems of unsupervised learning are:

- **Clustering Problems:** Where the objective is to discover inherent groupings in the data.
- **Association Problems:** Where the objective is to discover rules that describe large segments of the data.

In Table 3.1 we summarise the different classical machine learning problems with some examples of the models used and the application thereof.

| Problem Type | Description | Models/Models | Application |
|---|---|---|---|
| **Regression** | Fitting a line that best explains the data points. | Linear regression, Ridge/LASSO regression, polynomial regression. | The two primary uses for regression in business are forecasting and optimisation, e.g. helping managers predict such things as future demand for their products by looking at past trends. |
| **Classification** | Categorising a set of data points based on their features. | Logistic regression, support vector machine, K-Nearest Neighbours, decision trees. | Popular use cases include tumor classification, flower species identification and consumer and corporate credit classification. |
| **Clustering** | Discovering inherent groupings in a set of unlabelled data. | Partitioning methods, hierarchical clustering, model-based clustering, density-based clustering. | Spam filtering, identifying fake news, marketing and sales targeting. |
| **Association** | Discovering a set of rules that describe a large portion of the data set. | Apriori algorithms, eclat algorithm and FP-growth algorithm. | Web usage mining, intrusion detection, continuous production, recommender systems and bioinformatics. |

Table 3.1: A summary of the descriptions and applications of the main classical machine learning problems.

## 3.4 Beyond Classical Machine Learning

As mentioned, the other areas of ML include ensemble methods, reinforcement learning and NN/DL.

### 3.4.1 Reinforcement Learning

Reinforcement learning (Sutton 1992) (also known as semi-supervised learning) is a very popular and essential midpoint between unsupervised and supervised learning. Here we make use of both labelled and unlabelled data. Unlike supervised learning, where the model is given labelled data, reinforcement learning models learn through trial and error while performing a task and maximising long-term reward. Through this type of learning, we can develop a system that can learn how to function in real-world scenarios (Maini & Sabri 2017).

We provide an example of how reinforcement learning works at a high level. There is always an agent (the model) trying to accomplish a task or action in an environment in

reinforcement learning. Take, for example, a machine trying to walk across a terrain. The agent is the machine, the task is walking, and the environment is the terrain it is trying to cross. For the machine to learn to navigate the terrain (the goal), it must be provided with feedback. For example, the machine could fall or stumble over uneven terrain or bump into an object. This is negative feedback, and the machine learns from it. Next iteration of walking it would know to avoid the activity that caused it to fail so that it manages to walk further without making a mistake. This is the reward. The process is summarised in Figure 3.1.

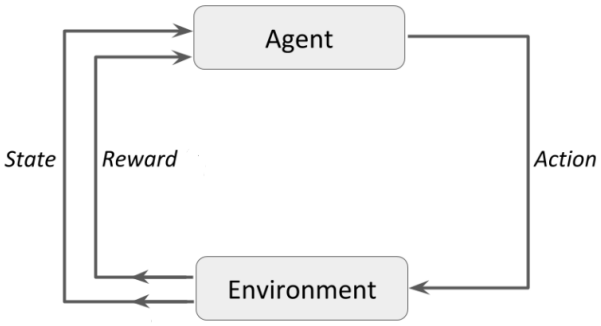



Figure 3.1: Illustration of the process of reinforcement learning.

(Source: (Bhatt 2018))

Some real-world examples of reinforcement learning include:

- **Google's DeepMind:** A computer program that was able to beat the top-rated GO player and then later the top-rated chess program.
- **Robotics:** The field of robotics very often relies upon reinforcement learning to help robots to learn to perform better in the environment they are in, e.g. self-driving cars.

### 3.4.2 Ensemble Methods

The core idea behind ensemble methods (Dasarathy & Sheela 1979, Hansen & Salamon 1990, Schapire 1990) is to combine different ML algorithms in an attempt to construct more accurate and efficient predictive models. These various models are combined strategically to solve specific problems. The premise is simple: by combining predictions from different models, we will average out idiosyncratic errors and produce more accurate predictions overall. Figure 3.2 presents a basic ensemble procedure:

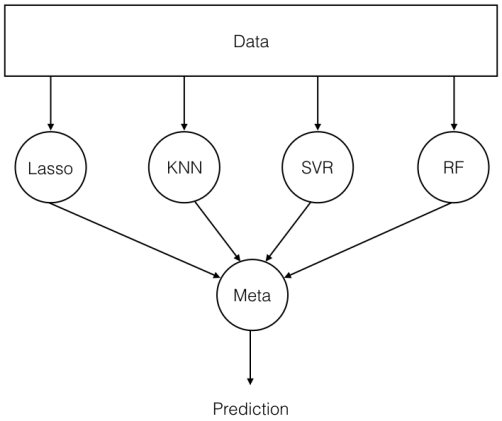



Figure 3.2: Illustration of an ensemble method.

(Source: (Keim 2019))

Some popular ensemble methods include:

- **Bagging (also known as Bootstrap Aggregation):** Bootstrap (Efron 1992) is a sampling technique, whereby we randomly select several observations from a data set. After creating our bootstrapped samples, we train separate models (or the same model) on these samples. The individual outputs are averaged to produce our final prediction.
- **Boosting:** Boosting (Schapire 1990) is a sequential learning technique. We train our model on the entire training set, then build subsequent models. Each subsequent model's prediction is weighted by accuracy, and more attention is given to those data points that were mispredicted by the previous one. Each of the predictions is then given weighted accuracy scores, and the results are aggregated to arrive at a final prediction.
- **Voting-based ensemble learning:** Voting ensemble learning (Polikar 2009, Brown 2010) is when we create multiple separate models using a common data set. Then the voting-based model aggregates all predictions. We can then use this construction to make predictions on unseen data. Each sub-model's (called ensemble members) predictions are assigned weights using stacked aggregation, which is used to learn how to weigh these predictions in the best possible way.

### 3.4.3 Neural Networks and Deep Learning

Neural networks (Rosenblatt 1958) are a specific set of algorithms created to mimic the way the human brain works, i.e. by receiving a range of stimuli (input) and then parsing it through layers of neurons that learn to associate input with output. They are so flexible and efficient that they can be applied to almost any ML problem with a complex mapping from the input to the output variables. NNs can solve classification (e.g. logistic regression) and regression problems (e.g. linear regression) and can be applied to supervised and unsupervised learning problems. Some applications include:

- function approximation,

– object identification on photos and videos,

– speech recognition and synthesis,

– image processing, style transfer and

– machine translation.

The basic layout of a NN is a collection (layer) of neurons and their respective connections. A neuron is represented by a function (called activation functions) provided with a set of aggregated inputs and then mapped to one output. Figure 3.3 represents a simple perceptron model. In ML, the perceptron is an algorithm for supervised learning of binary classifiers and is the simplest form of a neural network.

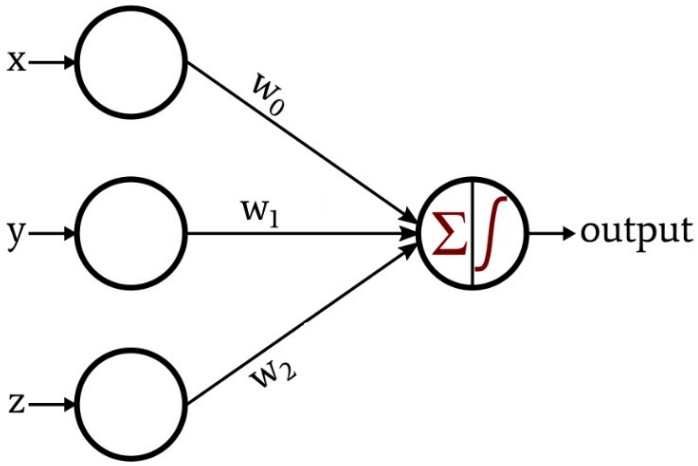



Figure 3.3: Illustration of a simple perceptron. In this illustration, $x, y, z$ represents the inputs to the model, the $w_i$s the weights associated with each input, $\Sigma$ represents the aggregation and the Sigmoidal symbol represent the application of the activation function.

(Source: (Paul 2018))

The connections between neurons feed outputs from one neuron to inputs for neurons in the next layer. Each connection has only one parameter – weight ($w_i$). This weight can be interpreted as the strength of the signal. All of the neurons are not randomly linked but by layers. Neurons in the same layer are not connected, but each layer is connected to the previous and subsequent layers. In feed-forward NNs, data flow strictly from the input layer to the output layer. Layers between the input and output layers are called hidden layers (all of this is illustrated in Figure 3.4). When a network has two or more hidden layers, we refer to it as a deep neural network. In practice, we do not physically construct neurons and connections. Instead, we represent the network as a set of matrices.

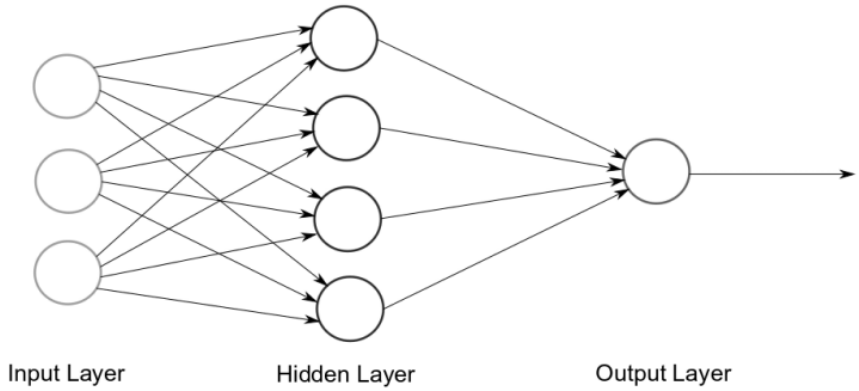



Figure 3.4: Illustration of a multi-layered perceptron with one hidden layer.

After a network is constructed, the task assigns weights to each input so that neurons will react correctly to incoming signals. As with supervised learning, during the training of the network, we have labelled data. We start by randomly assigning each input variable $x_i$ with a weight $w_i$. Data then flow through the network and arrive at an output $\hat{y}$. We then check the prediction against the actual $y$ and adjust each weight if the prediction is off. After many such cycles of "infer-check-punish", the idea is that the weights are corrected and act as intended. This process is called back-propagation.

DL excels at approximating unknown functions, particularly in situations where the data are complex. ANNs are known as universal function approximators because they can learn any function, proved by the Universal Approximation Theorem (Cybenko 1989). An informal version of the theorem follows:

**Theorem 3.4.1** ***The Universal Approximation Theorem*** *states that feed-forward NNs constructed of artificial neurons can approximate real-valued continuous functions on compact subsets of* $\mathbb{R}^n$*, to arbitrary accuracy.*

The theorem thus implies that a simple NN can, in principle, be applied to nearly any problem (given enough features and training instances), as they can approximate essentially any function of interest. Early versions of the theorem by Cybenko (1989) and Hornik (1991) considered networks of arbitrary width (number of nodes per layer). A simple general formulation was given by Pinkus (1999). Later versions by Lu et al. (2017) and Hanin & Sellke (2017) considered the "dual" problem for networks of arbitrary depth (number of layers). The point, however, is clear; DNNs are proving to be extremely effective in practice. Fan et al. (2019) provides a more in-depth discussion of this theorem as well as providing proof.

## 3.5 Conclusion

The Universal approximation theorem clearly illustrates the power NNs have over traditional ML, and the advantage DL has over both.

There also exist many different NN architectures for different problems. Chapter 5 will consider a mathematical investigation of feed-forward NNs and delve deeper into the theory behind it. Other architectures include convolutional NNs, recurrent NNs, generalised adversarial networks and auto-encoders.

We can now start examining the different models (statistical-based and machine learning-

based) used in practice and literature and consider some methods used to measure their performance and increase it as well. In the next chapter, we will do just this.

# Chapter 4

# Credit Scoring Methods, Models and Evaluation

"Essentially, all models are wrong, but some are useful." - *George E.P. Box*

## 4.1 Introduction

We will review the various methods and models that have been developed in the field. We will also look at various metrics used to evaluate our models and review some techniques from data science to increase the predictive ability of our models.

Credit scoring is used to determine who receives credit and how much is given as well as helping with ongoing monitoring of borrowers. Because of this, it is imperative that the models used are highly accurate or at least give the lender an excellent idea of the credit risk they are facing. We start by looking at the two types of scoring models: statistical-based and ML-based models.

## 4.2 Statistical-Based Models

### 4.2.1 Probit and Logit Models

Probit models (Abdou & Pointon 2011) and logit models (Bolton et al. 2010) are techniques for estimating the probability of an event occurring (a borrower defaulting) by predicting a binary dependent outcome $Y$ based on a set of independent inputs $\mathbf{X}$ (attributes).

Consider working with default data where the outcome of the dependent variable $Y$ is either $Y = 1$ for non-default or $Y = 0$ for default. Rather than modelling $Y$, probit and logit models model the probability that $Y$ is equal to 1 (non-default) or 0 (default), i.e. $p = P(Y = 0)$ is then the probability of default.

$P(Y = 0|\text{balance})$, the probability the a customer will default, given their balance owing, or $P(\text{balance})$ for short, will have a value between 0 and 1. We can predict the probability of default given a value for balance. For example, one might predict $Y = 0$ for any individual for whom $P(\text{balance}) > 0.5$. However, if the bank prefers a conservative approach in predicting which borrowers might default, then they can choose a lower threshold, for example, $P(\text{balance}) > 0.45$. The goal is to model the probability of default $p$ by specifying the model,

$$p = f(\alpha + \beta X_i), \tag{4.1}$$

where

$\alpha$ is the intercept and
$\beta_i$ is the coefficients for respective $X_i$.

For probit models we use the cumulative normal distribution function. Eq. (4.1) becomes

$$p = \int_{-\infty}^{\alpha+\beta} \frac{1}{\sqrt{2\pi}} exp\left(-\frac{1}{2}t^2\right) dt. \tag{4.2}$$

For logit models the formula is

$$logit(p) = log\left(\frac{p}{1-p}\right) = \alpha + \beta_1 X_1 + ... + \beta_n X_n. \tag{4.3}$$

To estimate the parameters, we use maximum likelihood estimation, which is a method of estimating the parameters of a probability distribution by maximising a likelihood function so that under the assumed statistical model, the observed data are most probable.

Bolton et al. (2010) suggested that when it comes to credit scoring, logistic regression is easy to elaborate and execute. **It is for this reason that the technique has been the chosen method in the banking industry**.

### 4.2.2 Linear Discriminant Analysis

Linear discriminant analysis (LDA) (Bansal et al. 2008) is a technique that assumes there are two populations, the defaulters and the non-defaulters. The population is then categorised into these two classes based on a set of attributes $X$ using

$$Z = \alpha + \sum_{i=1}^{n} \beta_i X_i, \tag{4.4}$$

where $X_i$s are the explanatory variables or attributes and $\alpha$ and $\beta$ are the intercept and coefficients, respectively, with $Z$ being the discriminant z-score. $Z$ is used to classify an observation (borrower) into a particular class based on a certain pre-defined threshold. This method is one of the early statistical techniques that have been used to build credit scoring models. The drawback is that it is quite restrictive in its assumptions and is quite sensitive to outliers, and the size of the smallest group must be larger than the number of predictors variables.

Using LDA we assume (Büyüköztürk & Çokluk-Bökeoğlu 2008):

- Independent variables are normal for each level of the grouping variable;
- Variances among group variables are the same across levels of predictors (homoscedasticity);
- Predictive power can decrease with an increased correlation between predictor variables (multicollinearity);
- The score for one variable is assumed to be independent of scores for other variables.

These statistical assumptions are seldom satisfied in real life (Eddy & Abu Bakar 2017).

### 4.2.3 Markov Chain Analysis

A Markov chain or transition matrix (Smith & Lawrence 1995, Greenidge & Grosvenor 2010) is another tool that is used to define the probability that a borrower will move from one state (good) to another (bad) or vice versa. The matrix (see Figure 4.1) is developed based on available data with each entry defining a transition probability $P_{i,j}$ of going from state $i$ to state $j$. The time to transit from one state to another ranges from one hour to one year or even longer. The matrix is then converted to a steady-state long-run probability of the borrower being good or bad.

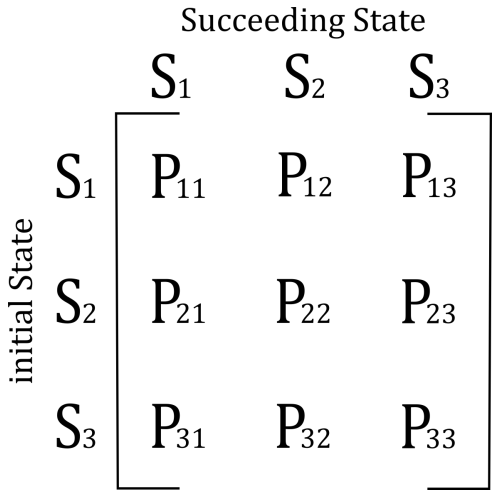



Figure 4.1: An illustration of a transition matrix.

## 4.3 AI/ML-based Models

The core focus of this dissertation is to explore the use of NNs/DL in credit default prediction. These models are based on the structure of the human brain and "learn" through experience. An ANN processes specific characteristics (independent variables of features) and produces output or responses similar to the human brain. Their success in credit default prediction has been demonstrated (Angelini et al. 2008, Khashman 2010, Eletter et al. 2010, Alabi et al. 2013).

We will aim to replicate these results for ourselves and attempt to understand their potential while also applying modern techniques to boost their performance. However, in order for us to demonstrate their power, we need other ML models used in literature to compare our results with. We have decided to use two of the most popular models: CART models and SVMs. Hence, we will proceed to describe how these models work in order to provide a better understanding of what our neural networks will be measured against. We will dedicate Chapter 5 to describing how neural networks work; hence we will not go into further detail here.

### 4.3.1 Classification and Regression Trees (CART)

Decision trees also referred to as classification and regression trees (CART) (Breiman et al. 1984), are simply sets of cascading questions that perform either classification or regression, depending on whether the response variable is categorical or numeric. With classification trees, we start with a set of features and then use the value of a given feature to answer a question. By answering each question, the decision rule will then determine

the next question. At the end of this sequence of questions, we will have a probability of the data point belonging to a specific class.

A decision tree is an upside-down structure that branches out from the initial node (the root node). From each node, there are two branches are leading to new nodes. Each new node represents a partition of the feature space that satisfies the condition of the question at the previous node. A node on the left branch includes the data points in the feature space for which the condition of the previous node was true, with the remaining data points being included in the node on the right branch. The terminal nodes at the end of the tree are used to perform classification or regression and are referred to as the tree leaves. The nodes in between the terminal and initial nodes are called intermediate nodes, and they are used to arrive at the final prediction. Figure 4.2 provides an example of a decision tree being applied in the approval of loans.

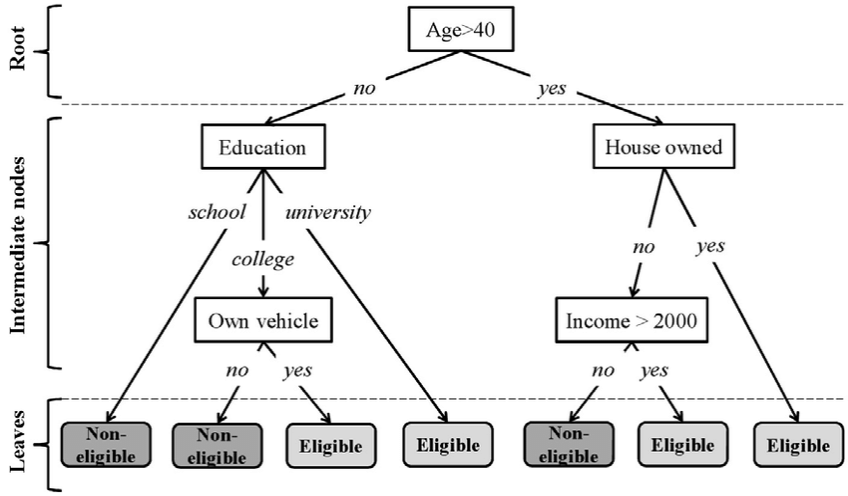



Figure 4.2: An example of a decision tree classifier for approving loans. In this case, the features are age, education, own vehicle, house owned and income. Moreover, as we can see, each branch represents and answers the question about the respective feature.

(Source: (Aleksandrova 2017))

The challenging part of decision trees is to understand how they are built. The question is, how do we set up this cascading sequence of questions and determine the optimal split created by each decision rule?

We start with our entire feature space as one region. Then, through a numerical procedure called recursive binary splitting, the algorithm iterates through all the dimensions of our feature space and determines what split would lead to the most significant reduction in error (as this is a supervised learning algorithm, and we measure the predicted value against the actual value during the training/creation of our tree model). Once a split has been performed, and a decision rule determined, the algorithm repeats the process on the new sub-regions created due to the split. This process is repeated until a specific stopping criterion is met (e.g. we stop if the current sub-region, represented by a leaf node on the tree, contains a certain amount of the data set, referred to as the leaf).

This process of continual splitting is also called a greedy approach since the algorithm determines the split based on the most significant reduction in error without considering the sequence of partitions that would follow from that split onward. So, in reality, the first split might lead to the most significant reduction, but it might not be the most optimal split that could be made.

We consider a simple example. Let us start with a two-dimensional feature space of

$\{x_1, x_2\}$ as in Figure 4.3.

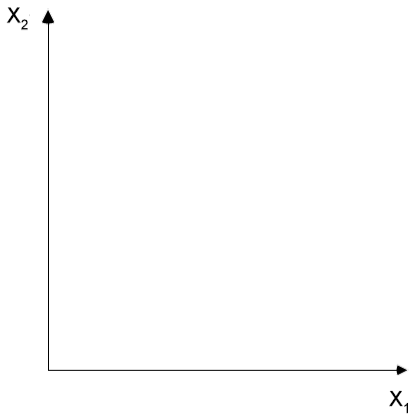



Figure 4.3: A 2 dimensional feature space with two input variables $\{x_1, x_2\}$

In the example we are using, the tree performs classification trying to determine if an outcome is either good or bad. The algorithm first determines that the optimal split is determined by the decision rule $x_1 < L_1$, thereby splitting the feature space into two regions (in the Cartesian plane in Figure 4.4, the partitioning is performed by the vertical line at $L_1$). The tree model creates its first split creating two branches and two intermediate nodes. So, our first question is, is $x_1 < L_1$? The data points for which this statement is true are split to the node on the left and the other to the node on the right (the standard layout of a decision tree is that the node on the left corresponds to the "true" or yes answer to question whilst the other corresponds to the "false" or no answer).

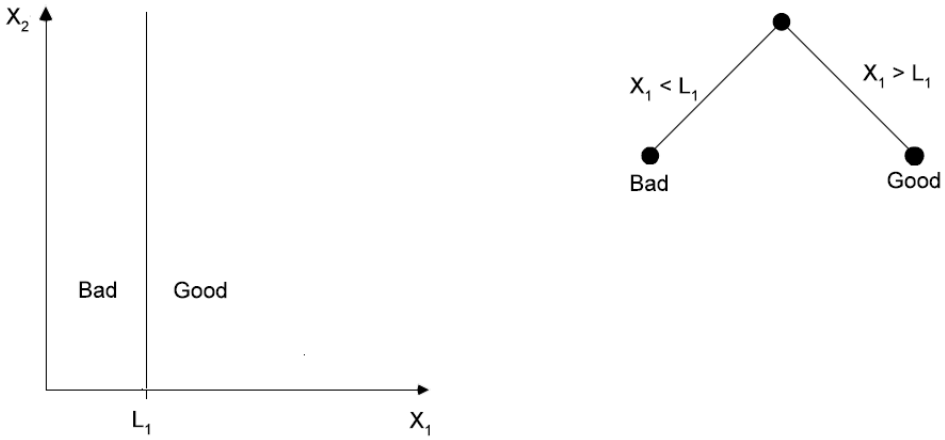



Figure 4.4: On the left, our feature space after being partitioned and on the right, a view of our decision tree.

Next, the algorithm will determine the most optimal split in each of the two new sub-regions. First, on the region or left of $L_1$, the algorithm determines that the most optimal split is determined by the rule $x_2 < L_2$, while in the other region, it is $x_1 < L_3$. These rules lead to two new partitions of each sub-region or four total sub-regions (as illustrated in Figure 4.5).

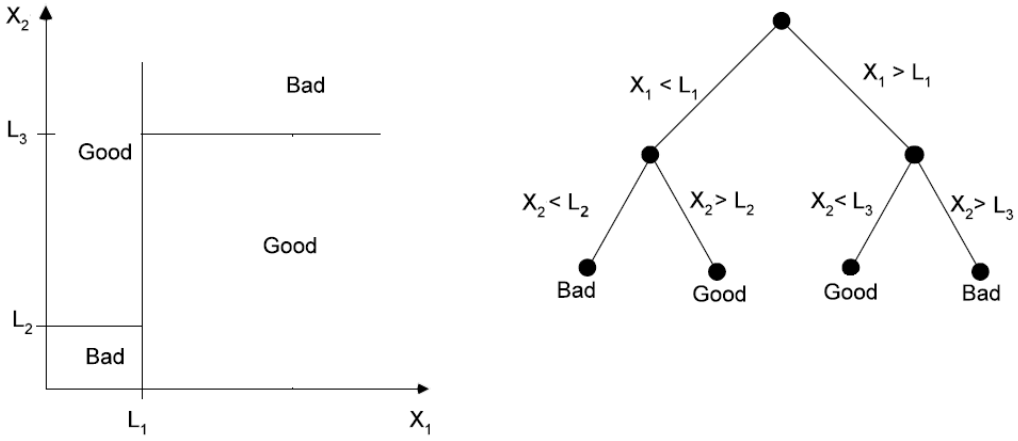



Figure 4.5: On the left, our feature space after two additional partitions and on the right, our decision tree with four additional branches.

The process continues until the stopping criterion is reached. The final model (Figure 4.6) implies that all realisations of $\{x_1, x_2\}$ in which $x_1 < L_1$ and $x_2 < L_2$ are categorised as a bad outcomes, and all realisations $\{x_1, x_2\}$ in which $x_1 < L_1$ and $x_2 \geq L_2$ are categorised as a good outcomes, etc. In general, the $\{L_i\}$s are chosen to minimise the error in the fitted values for the dependent variable using the appropriate metric (usually mean-squared error).

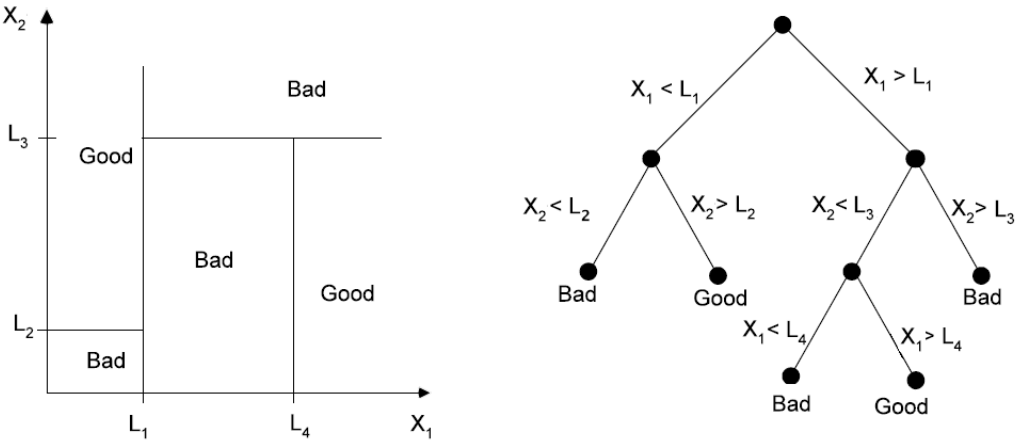



Figure 4.6: A CART model with two input variables $\{x_1, x_2\}$ and a binary dependent variable with two possible values (bad or good).

Once the model has been trained, optimised and finalised, forecasts can easily be made for new realisations $\{\tilde{x_1}, \tilde{x_2}\}$ by starting at the top of the decision tree and moving along the tree from its top to bottom.

The reason why CART models have gained much popularity is that it overcomes the drawbacks of, for example, probit and logit models where dependent variables are forced to fit a single linear model throughout the entire input space (Khandani et al. 2010). CART models can detect non-linear relationships between input and output variables and produce interpretable results with clearly laid out logic. This aspect is vital in banking sector applications, where "black-box" models are viewed with scepticism (Khandani et al. 2010).

Over the past two decades, tree-based algorithms have proven to be very powerful but are likely to overfit the data, leading to a poor test set performance (James et al. 2013). It is essential to keep this in mind, as it might be necessary to apply techniques such as pruning to improve the generalisation of our model. This technique is described and demonstrated in Section 7.4.3.

Modern variations or adaptions of decision trees include random forests (Breiman 2001) (which is an application of bootstrap aggregation (Efron 1992) or bagging) and gradient boosted trees (GBTs) Friedman (2001) (including algorithms such as XGBoost (Chen et al. 2015) and LightGBM (Ke et al. 2017)).

### 4.3.2 Support Vector Machines (SVM)

A support vector machine (SVM) (Boser et al. 1992, Guyon et al. 1993, Drucker et al. 1997) is a robust ML algorithm that attempts to classify points in a feature space by finding a hyperplane/decision boundary that separates these points in the feature space based on the value of the response variable. Once the model is trained, new instances can then be mapped into that same space and classified into a category based on the side of the hyperplane or gap on which they fall. More formally, an SVM constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks like outlier detection. SVMs are more commonly used in classification problems, and as such, we will focus on this. SVM classifiers are non-probabilistic binary linear classifiers.

As before, we will describe the model using a two-dimensional feature space (see Figure 4.7). We start by defining the hyperplane. In this case, the hyperplane is a line. When we train the model, the goal is to find a line that best separates the points in the feature space based on their characteristics. The points in each of the two subsets (assuming binary classification) represent a particular class (measured by the response variable).

Next, we need to define the support vectors. These are the data points nearest to the hyperplane. If these points were to be removed, it would alter the position of the dividing hyperplane. It is for this reason that these points are considered the critical elements of a data set. Lastly, we need to define the margin of the model. Margin is simply the distance between the support vectors. Intuitively, a good separation in the feature space is achieved by the hyperplane with the largest distance to the nearest training-data point of any class. In general, the larger the margin, the lower the generalisation error of any class of the classifier. So the margin is an important measure to compare different hyperplanes.

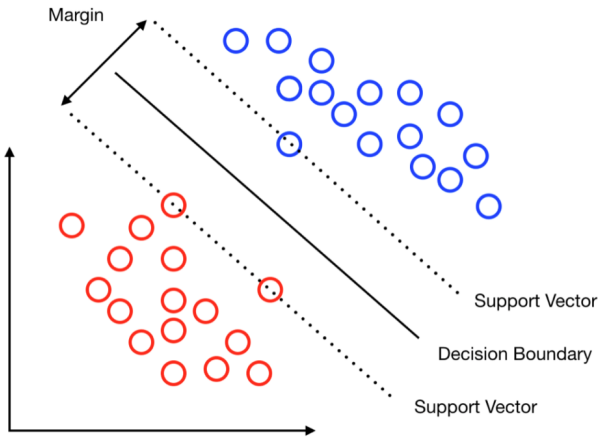



Figure 4.7: A simple illustration of a support vector. In this example, our feature space is two-dimensional, and the axes represent the values of our two features.

(Source: (iUnera n.d.))

The final question that remains is what happens if there is no clear hyperplane? In practice, data are rarely as clean as in our simple example. Data points from separate classes will often be mixed, making a clear linear separation impossible in the current dimension. To overcome this, we need to move away from a two-dimensional view of the data to a three-dimensional representation, i.e. we need to map our data into a higher dimension. This process is known as kernelling and is illustrated in Figure 4.8. The name kernelling derives from the use of kernel functions. Popular kernel functions in the construction of SVMs include linear kernels, polynomial kernels and the radial basis functions (Pedregosa et al. 2011).

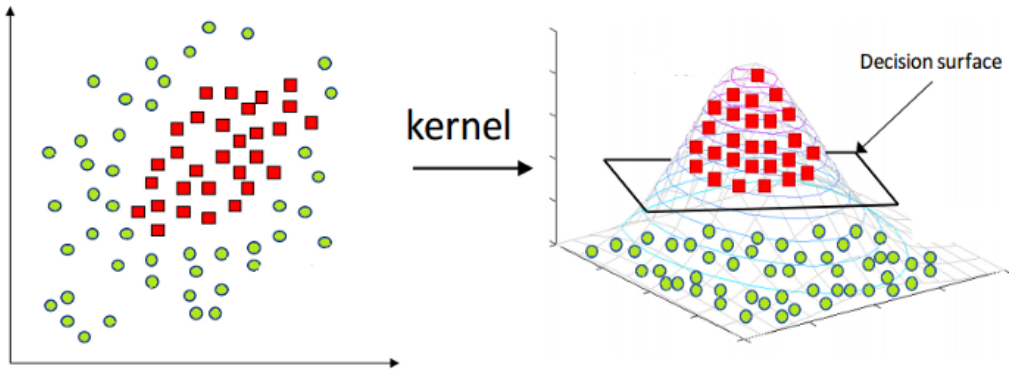



Figure 4.8: An example illustrating how kernelling transforms our feature space into a higher dimensional space to attempt to find a clearer separation.

(Source: (Sharma 2019))

Now, because our data is represented in three dimensions, our hyperplane can no longer be a line and must be a plane (or decision surface), as shown in the example above. This process aims to continue to map our feature space into higher and higher dimensions until a hyperplane can be formed to segregate it.

SVMs have been shown to perform well in many applications and are considered by some as one of the best "out of the box" classifiers (James et al. 2013). Some of the

benefits of SVMs include their high level of accuracy. They work well with smaller sets of data and they can be very efficient because they use a subset of training points instead of the entire feature space. On the other hand, the drawbacks are that they are not well suited to large data sets and are less effective if our data set is noisy. These models have also been applied with much success in credit default prediction (Min & Lee 2005, Shin et al. 2005, Kim & Sohn 2010, Khemakhem & Boujelbene 2017).

## 4.4 Evaluation of Statistical and AI/ML-Based Credit Classification Methods

We will look at an often used performance measurement tool in ML called the **confusion matrix** which is a $2 \times 2$ contingency table as shown in Figure 4.9.

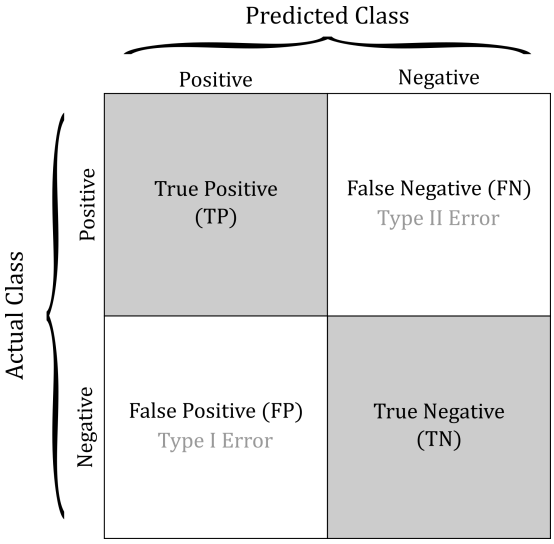



Figure 4.9: A confusion matrix.

The columns of the matrix represent the classes predicted by the model or the *ex-ante* classifications. In the case of credit classification, the classes would be "defaulted" and "non-defaulted". The rows, on the other hand, represent the actual classes or *ex-post* realisations of delinquency.

As seen in Figure 4.9 the lower-right entry is called the true negative (TN) or a type-I error. The upper-left entry is called true positive (TP) or a type-II error. Our goal is to decrease the type-II error subject to a set type-I error. Put differently, we want to maximise the power of our model subject to a fixed size.

Based on the entries of the matrix we can now calculate the following six performance metrics:

1. **Precision**, which measures the model's accuracy when it classified a customer as bad and is calculated as $TN/(TN + FN)$.

2. **Recall**, which measures the number of bad customers correctly classed by the model over the actual number of bad customers and is given by $TN/(TN + FP)$.

3. **TP Rate**, which is given by $TP/(TP + FN)$.

4. **FP Rate**, which given by $FP/(FP + TN)$.

5. The rate of incorrectly classified instances given by $P_e = (FP + FN)/N$.

6. The rate of correctly classified instances given by $P_a = (TP + TN)/N$.

An ideal situation is one where we have a model with **high precision and recall** or one that has **low FP rate and a high TP rate**.

Next, we will look at the percentage of incorrectly and correctly classified instances. This may not necessarily be a sufficient measure of a model's predictive power when the distribution of the predicted variable is highly skewed (i.e. many customers never default). Because of this, we also need to consider a few more sophisticated measures.

Another tool we can use to evaluate our model is the **receiver operating characteristic (ROC)** curve. By applying our model and using different classification thresholds, we can visualise the trade-off between TP and FP as seen in Figure 4.10.

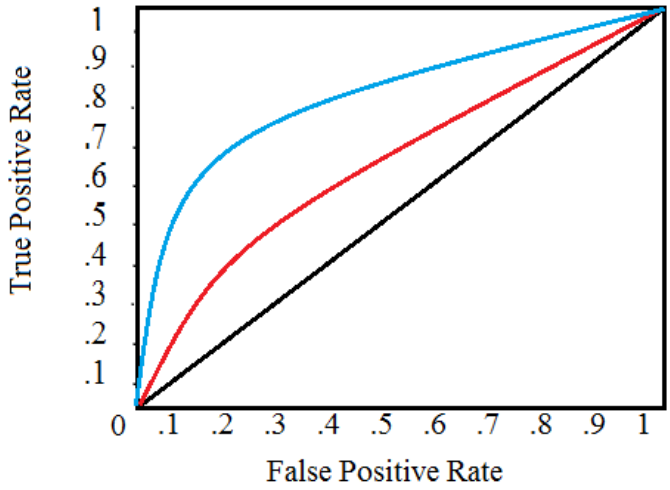



Figure 4.10: An example of a ROC curve.

(Source: (Sukhadeve 2017))

The blue and red lines are the ROC curve and are essentially pairwise plots of the TP and FP rates for different thresholds (represented by the black line). The blue line would represent a good model, while the red line would represent a bad one. The trade-off is non-linear.

When analysing the ROC curve we can also calculate the **area under the curve (or ROC-AUC score)** (Bradley 1997) to compare models. ROC-AUC can be interpreted as the probability of the classifier assigning a higher score, i.e., a higher probability of an instance from class 1 being labelled as being from class 1 than from a class 0. Some important characteristics of the ROC-AUC Score are:

- ROC-AUC $\in [0, 1]$:
  - If our model's predictions are 100% wrong then ROC-AUC = 0.0
  - If our model's predictions are 100% correct then ROC-AUC = 1.0.
  - If our model is just a random classifier we will have a ROC-AUC of 0.5 (for balanced data).
- ROC-AUC score is independent of the threshold set for classification because it only considers the rank of each prediction and not its absolute value.

It is calculated as:

$$AUC = \frac{1 + TP_{rate} - FP_{rate}}{2}. \tag{4.5}$$

The **kappa statistic**, or Cohen's kappa (Cohen 1960), is a score that measures the prediction successes in a classification problem, where success is measured relative to purely random classifications. It is calculated as follows:

$$\kappa = \frac{P_a - P_e}{1 - P_e}. \tag{4.6}$$

A $\kappa$ between 0.6 and 0.8 indicates substantial agreement, and a $\kappa$ greater than 0.8 implies almost perfect agreement (Landis & Koch 1977).

Finally, we will consider the **F1 Score**, which can also indicate significant predictive power and is the harmonic mean* of the model's precision and recall and is given by:

$$F1 = \frac{2 \times \text{Recall} \times \text{Precision}}{\text{Recall} + \text{Precision}} \quad \text{and} \quad F1 \in [0, 1]. \tag{4.7}$$

When our positive class is small, then the F1 score is preferred to the ROC-AUC score. Unbalanced classes are prevalent in credit classification problems as people are far more likely not to default than they are default.

## 4.5 Cross-validation

Cross-validation is a method used to measure the skill of an ML model. It is commonly used in applied ML to evaluate and compare models. It is easy to interpret, simple to implement, and the metrics resulting from the process tend to have a lower bias than other methods. **Cross-validation is the gold standard to evaluate these models** but can be very time consuming to execute (Ghosh et al. 2020).

k-fold cross-validation is the most popular form of cross-validation. When we have a situation where we have a small data set, and we want to preserve as much data as possible for the model's training, we will use k-fold cross-validation. The procedure is quite simple but powerful. We start by splitting up the data set into $k$ subsets, called folds, where each fold is used as the test set, and the rest of the data are used as the training set, and this is repeated $n$ number of times (see Figure 4.11). After all the folds have been fitted, the results (performance metrics) are averaged to give the final result.

*The harmonic mean can be expressed as the reciprocal of the arithmetic mean of the reciprocals of the given set of observations.

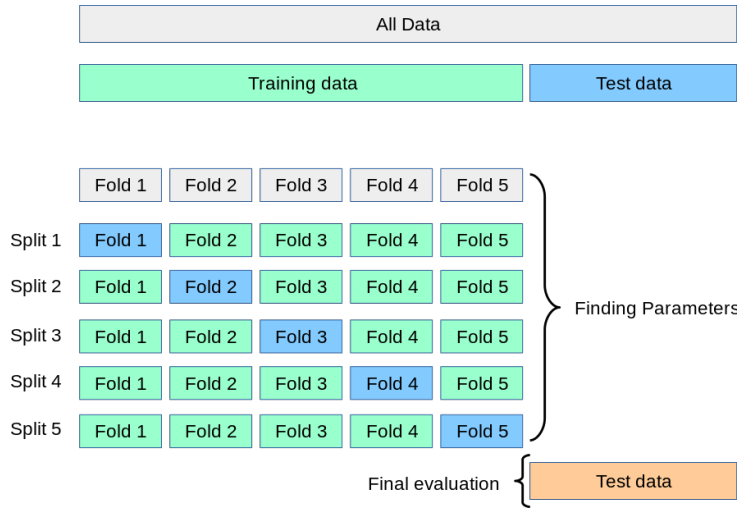



Figure 4.11: An illustration of the k-fold cross-validation procedure.

(Source: (Scikit-Learn 2020))

**Variations on Cross-Validation**
There are different versions of the cross-validation procedure. These include:

- **Train/Test Split:** $k$ can be set to 2 such that a single training and testing data split is created.
- **LOOCV:** $k$ can also be set to the total number of observations in the data set. This allows each test sample to be used in the hold-out data set. This process is called leave-one-out cross-validation (LOOCV).
- **Stratified:** Here, each fold is split in such a way that the categorical split of the data represents the split in the data set.
- **Repeated:** Here the k-fold cross-validation procedure is repeated $n$ times. The data is also shuffled before each cycle which results in a different split of the sample.
- **Nested:** This is essentially splitting each fold into another set of folds. This process is often used to perform hyper-parameter tuning during model evaluation.

Before performing cross-validation, we need to determine how many folds we need to fit (i.e. choose $k$). We can choose $k$ such that each of the samples are large enough to be statistically representative of the broader data set. The preferred method in practice seems to be $k = 10$ as it has shown to produce model skill estimates with low bias and modest variance (Brownlee 2020*a*).

## 4.6 Class Imbalance Problem

One of the greatest issues facing classification problems, particularly credit classification problem, is the **class-imbalance problem** (Brown & Mues 2012, Bischl et al. 2016). Class imbalance is when the number of instances in one class is far less than the other instances. It is a pervasive problem in practice, especially in credit default data sets, since a default event is far less likely to occur than a non-default.

Most ML algorithms learn more effectively when the data set is roughly balanced. When the split is heavily skewed to the one side, problems arise, such as the algorithm not having enough examples of the minority class to effectively learn the decision boundary. When it comes to solving this problem, there are a few methods to consider.

### 4.6.1 Cost function based approaches

Here we will, for example, weigh a false negative more heavily if we believe that one FN is worse than one FP, and then the ML algorithm will try to make fewer FNs compared to FPs as it is less costly (lower error).

### 4.6.2 Sampling-based approaches

Some sampling-based techniques include:

- **Undersampling** involves removing some majority class instances. The drawback here is that we are decreasing our sample size.
- **Oversampling** involves duplicating some minority class instances. The drawback here is that we are synthetically increasing our sample size, which might affect our model's ability to generalise.
- **Hybrid approach** involves combining oversampling and undersampling approaches. This approach gives us the advantages and drawbacks of both approaches, i.e. we will be increasing our sample size synthetically by less than before since our majority class has already been downsampled (by less than was required before due to oversampling of the minority class).

### 4.6.3 Synthetic Minority Over-Sampling Technique (SMOTE)

SMOTE (Chawla et al. 2002) involves selecting instances that are close in the feature space and sampling new points along the line between the two instances in the feature space.

We start by randomly choosing an instance of the minority class, and then a number of the nearest points (observations) are found (typically five). One of the points is randomly chosen, and a synthetic example is then created by randomly selecting a point between the two points in the feature space.

This newly created instance can create as many synthetic instances as needed for the minority class. **It is suggested to use SMOTE for oversampling after random undersampling of the majority class has been applied** to balance the class distribution (Chawla et al. 2002). The downside of this approach is that these new instances are created without considering the proximity of instances from the majority class. This can lead to ambiguous instances being created when there is a substantial overlap of the classes.

### 4.6.4 Other Approaches

Other approaches have been proposed in the literature, such as Underbagging (Barandela et al. 2003), RUSBoost (Seiffert et al. 2008) and SMOTEBagging (Wang & Yao 2009) and are all regarded as improvements on SMOTE. **SMOTE, however, remains popular due to its simplicity**.

## 4.7 Classification Thresholds

As mentioned earlier, in the context of CCR management, the trade-off between profitability and creditworthiness represents a cost/benefit analysis of FPs or Type-I errors vs FNs

or Type-II errors. Therefore, we can adjust the classification threshold and attempt to optimise some criterion function in which such costs and benefits are inputs.

First, we need to define a classification threshold. ML algorithms perform classification by looking at the probability of an instance being in one class. This is done using a threshold (default = 0.5) where all values greater or equal than that threshold are mapped to one class, and all other values are mapped to another class.

In a practical context, this serves a strategic purpose: If the bank aims to identify high-risk customers by using a lower threshold aggressively, this will result in more incorrect classifications of less risky customers as high risk. This will lead to the opportunity cost of lost interest income that could have been earned from those customers who will then have their credit lines cut or not increased (Crouhy et al. 2006).

When our data set has a severe class imbalance, the default threshold could result in poor performance. We can improve our model by tuning the threshold that is used to map probabilities to class labels.

We can use the ROC Curve to calculate the optimal threshold directly, or we can also use a GridSearch algorithm (Pedregosa et al. 2011) (a hyper-parameter tuning algorithm which is applied in Chapter 7) to tune the threshold and find the optimal point.

## 4.8 Conclusion

It is clear that when it comes to model choice and performance metrics, there are several options. It just depends on what the problem and goal at hand are. When it comes to performance-boosting, a great understanding of the data and the problem is required and the model being used (this is where the field of data science comes in).

One thing is clear; we now have the understanding and tools necessary to start to build an effective credit classification tool. We first need to build a deeper understanding of our chosen tool; the neural network. In the next chapter, we will consider a more mathematical description of neural networks and examine some of the intricacies of model hyper-parameters.

# Chapter 5

# Neural Networks and Deep Learning

> "Thou shalt not make a machine to counterfeit a human mind." — *Frank Herbert*

## 5.1 Introduction

Here we start to build an understanding, in a mathematical sense, of the tools used in this dissertation. We will begin by giving some background to the field and constructing a neural network model by first considering a single perceptron model and eventually arriving at a deep neural network. It is crucial to note that this section will focus on the mathematics of the model as it is required, along with a technical understanding of the model, to prove in theory that it is possible to build an accurate credit scoring model. More importantly, it will also provide a better understanding of how neural networks are so superior at function approximation.

## 5.2 Background

ANNs are popular machine learning techniques that simulate the mechanism of learning in biological organisms (Aggarwal 2018). Rosenblatt (1958) combined Hebb (1949)'s model of brain cell interaction with Samuel (1959)'s ML efforts and created the perceptron (see Figure 5.1) . In ML, the perceptron is a binary classifier that can decide whether or not an input, represented by a vector of numbers, belongs to some specific class.

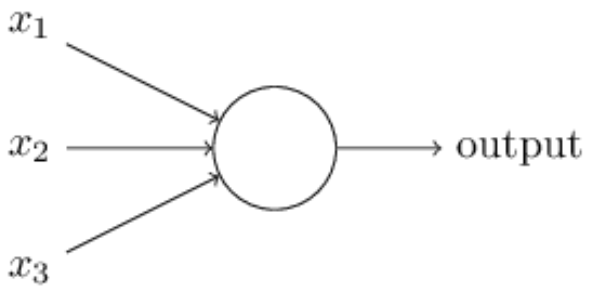



Figure 5.1: A simple perceptron. The $x_i$s represent the input into the model.

Rosenblatt (1958) 's perceptron software was intended for image recognition. Although the perceptron did look promising, it could not recognise many visual patterns (such as faces). This setback caused much frustration, and NN research was stalled (Foote 2019). Progress in the field was slow for several years (known as the AI winter) until its resurgence

in the 1990s. Since then, significant advancements have been made in the computing power of machines and the availability of data, considerably boosting the limits of what NN algorithms could achieve (Foote 2019).

## 5.3 A Neural Network Model

This section will set up a mathematical representation of a simple neural network (a perceptron model) and systematically build up a robust neural network with multiple layers (multi-layered perceptron).

### 5.3.1 Model Set-Up and Notation

Most of this chapter follows a derivation adapted from Lindholm et al. (2019) which has also been supplemented by work from Fan et al. (2019). Consider a function describing the output variable $y$ as a non-linear function of $n$ inputs $x_i$

$$y = f(x_1 + x_2 + ... + x_n; \boldsymbol{\theta}) + \epsilon, \tag{5.1}$$

where $\epsilon$ is the residual term $\boldsymbol{\theta}$ represents the parameters of function $f$. For convenience, we will define the output as $z$ without the residual term $\epsilon$

$$z = f(x_1 + x_2 + ... + x_n; \boldsymbol{\theta}). \tag{5.2}$$

Suppose $f$ is the multivariate linear regression model

$$z = \beta_1 + \omega_1 x_1 + ... + \omega_n x_n. \tag{5.3}$$

In Eq. (5.3), $z$ is simply the sum of all terms $\omega_i x_i$ and offset term $\beta$*. Next, we introduce and apply a non-linear scalar function, termed the **activation function**, $\Phi : \mathbb{R} \to \mathbb{R}$, to describe the relationship between $\mathbf{x} = [1\ x_1\ x_2\ ...\ x_n]^T$ and $z$. Equation (5.3) is now called a **generalised** linear regression model and is given by

$$z = \Phi(\beta_1 + \omega_1 x_1 + ... + \omega_n x_n). \tag{5.4}$$

Figure 5.2 illustrates the distinction between a linear regression model and a generalised linear regression model.

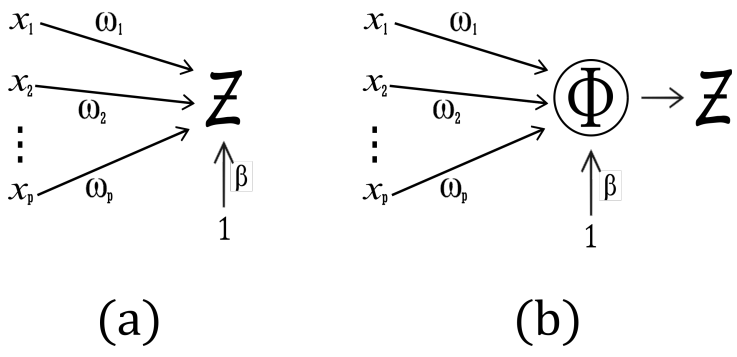



Figure 5.2: (a) Linear regression model. (b) Generalised linear regression model.

*$\beta$ is also often referred to as the bias term

When it comes to the choice of activation function $\Phi$, there are a few options. For regression problems the most widely used include:

- **Sigmoid function**: $\Phi(x) = \frac{1}{1+e^{-x}}$.
- **Rectified Linear Unit (ReLU):** $\Phi(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$.
- **Hyperbolic Tangent:** $\Phi(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ .

Figure 5.3 gives a graphical illustration of these functions.

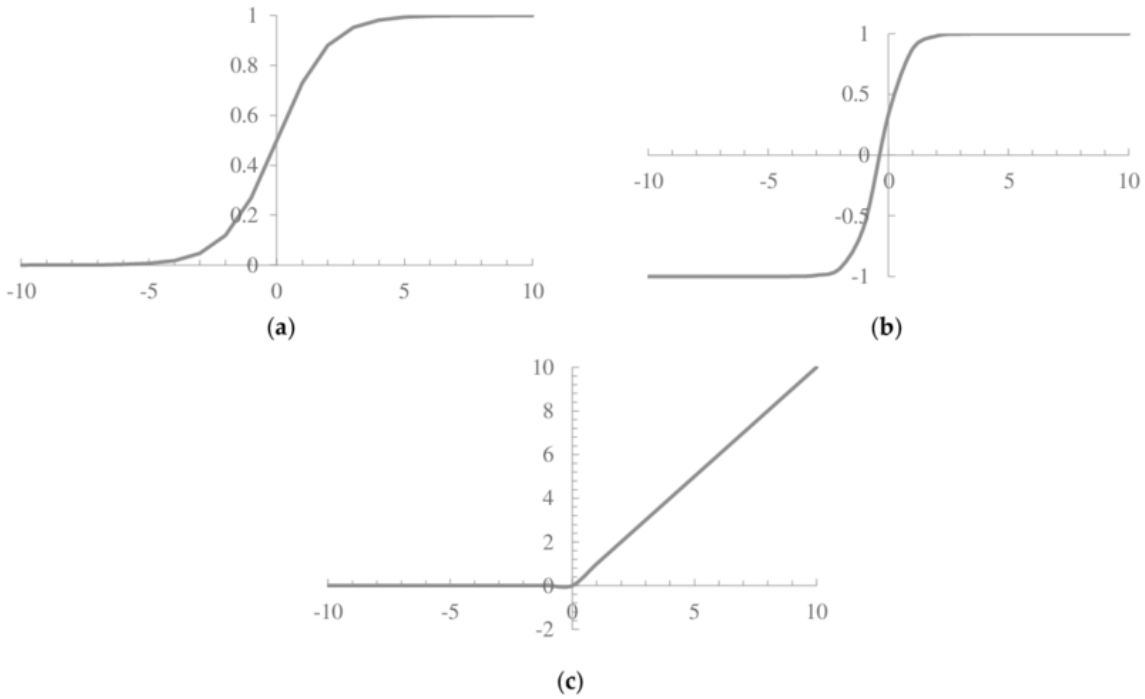



Figure 5.3: Examples of the most widely used activation functions: (a) Sigmoid (b) hyperbolic tan (c) ReLU

In the past, the sigmoidal function has been the function of choice for practitioners. However, recently the ReLU activation function has gained much popularity in modern NNs due to the ease in training multi-layered NNs (Aggarwal 2018). At this point, it might be clear that Eq. (5.3) is not yet capable of modelling/explaining very complex relationships between $z$ and $\mathbf{x}$. To achieve this, we will now stack several of these models to construct a **layer**. Each layer consists of a set of nodes that act by aggregating all weighted inputs received, applying the activation function and producing a single output (as shown in Figure 5.4). Hence each node is a generalised linear model.

We will then stack these individual layers **sequentially** to form a **multi-layered feed-forward** neural network. Sequential stacking means that the nodes from each layer are connected to the nodes in the next layer and not randomly. By "feed-forward", we mean that each neuron's output from one layer will become an input to neurons in the next layer.

We will now start constructing our multi-layered NN of arbitrary width. For this, we will have an input layer containing all the inputs $\{x_i\}_{i=1}^n$, the output layer and one intermediate layer called a hidden layer. Each of the $M$ nodes in the hidden layer will receive input variables from the previous layer (the input layer), weight and aggregate

them using the generalised linear model (GLM) (using the set of weight and bias parameters specific to that node). The node will then apply the activation function, producing a single output. That output, the node's output, is then passed as input to each node in the next layer, which has a new set of parameters. Figure 5.4 illustrates the process.

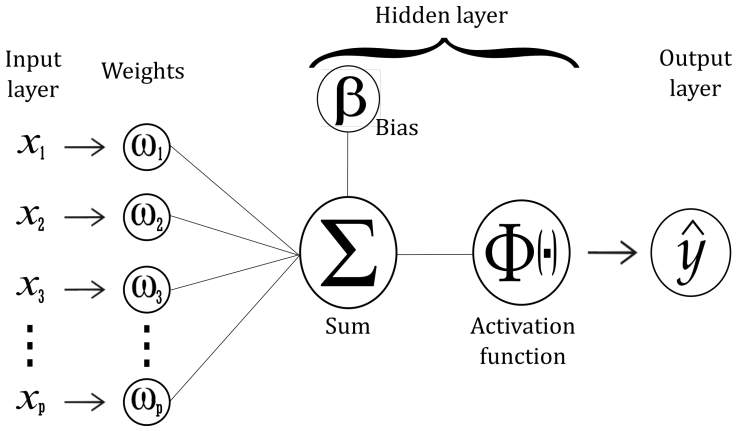



Figure 5.4: Illustration of how inputs flow through a neuron.

The parameters of the $i$-th GLM (or node) of layer one are denoted as $\beta_i,\ \omega_{1i},\ \ldots\ \omega_{ni}$ and output is denoted by $h_i$, so that we have

$$h_i = \Phi(\beta_i + \omega_{1i}x_1 + ... + \omega_{ni}x_n) \quad \text{for } i = 1, ..., M. \tag{5.5}$$

These intermediate outputs $h_i$ are called **hidden units**, and each of the $M$ units $\{h_i\}_{i=1}^{M}$ will feed forward as input to the next layer

$$z = \beta + \omega_1 h_1 + ... + \omega_M h_M. \tag{5.6}$$

In order to separate layers, we add superscripts (1) and (2) to distinguish between layer one and layer two. Thus

$$\begin{aligned} h_1 &= \Phi(\beta_1^{(1)} + \omega_{11}^{(1)}x_1 + ... + \omega_{n1}^{(1)}x_n), \\ h_2 &= \Phi(\beta_2^{(1)} + \omega_{12}^{(1)}x_1 + ... + \omega_{n2}^{(1)}x_n), \\ &\vdots \\ h_M &= \Phi(\beta_M^{(1)} + \omega_{1M}^{(1)}x_1 + ... + \omega_{nM}^{(1)}x_n), \\ z &= \beta^{(2)} + \omega_1^{(2)}h_1 + ... + \omega_M^{(2)}h_M. \end{aligned} \tag{5.7}$$

Let us recap what our model looks like so far. We have an input layer consisting of $n$ inputs $\{x_i\}_{i=1}^{n}$. Following this layer, we have our single hidden layer (denoted with superscript (1)), which has $M$ nodes, or hidden units, which are essentially individual GLM models, each with their own set of weight and bias parameters (the collection of which is denoted by $\theta_i^{(1)}$). The hidden layer is parametrised by $\boldsymbol{\theta^{(1)}}$, which is defined as $\boldsymbol{\theta^{(1)}} = \left[\theta_1^{(1)}, ..., \theta_M^{(1)}\right]$ and the activation function $\Phi$ for each node is defined per layer. Also, each node in the input layer is connected to each hidden unit in the hidden layer. Finally, we have our output layer, which, in our case, consists of a single node, which is also a GLM with its own activation function and set of parameters $\boldsymbol{\theta^{(2)}}$. Each hidden unit

in the hidden layer is again connected to the output node, which weights and aggregates all inputs $\{h_i\}_{i=1}^{M}$ received from the hidden units and applies the activation function to produce the final output of the network (denoted by $z$). Figure 5.5 gives us an illustration of the model we have constructed up to now.

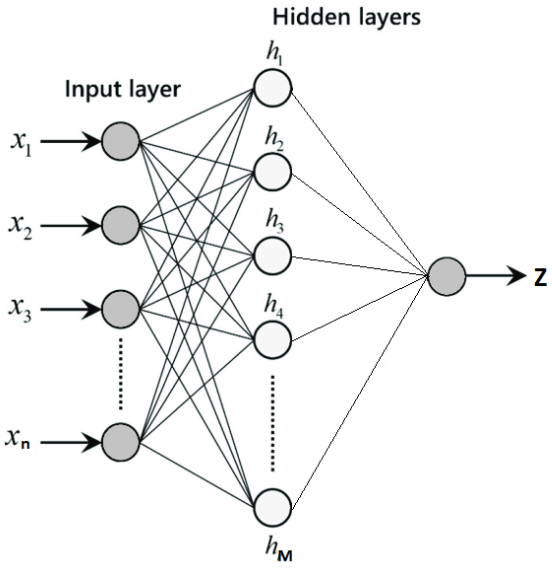



Figure 5.5: A multi-layered neural network one output node and one hidden layer.

#### 5.3.1.1 Matrix Notation

For convenience and compactness Lindholm et al. (2019) rewrite the system of equations in Eq. (5.7) in matrix form as

$$\mathbf{b}^{(1)} = \begin{bmatrix} \beta_1^{(1)} & \dots & \beta_M^{(1)} \end{bmatrix}, \quad \mathbf{W}^{(1)} = \begin{bmatrix} \omega_{11}^{(1)} & \dots & \omega_{1M}^{(1)} \\ \vdots & \dots & \vdots \\ \omega_{n1}^{(1)} & \dots & \omega_{1M}^{(1)} \end{bmatrix}, \quad \mathbf{b}^{(2)} = \begin{bmatrix} \beta^{(2)} \end{bmatrix}, \quad \mathbf{W}^{(2)} = \begin{bmatrix} \omega_1^{(2)} \\ \vdots \\ \omega_M^{(2)} \end{bmatrix},$$

where $\mathbf{W}$ is the **weight matrix** and $\mathbf{b}$ is the **offset/bias vector**. Now our model can be rewritten as

$$\mathbf{h} = \Phi(\mathbf{W}^{(1)T}\mathbf{x} + \mathbf{b}^{(1)T}), \tag{5.8}$$

$$z = \mathbf{W}^{(2)T}\mathbf{h} + \mathbf{b}^{(2)T}, \tag{5.9}$$

where subscript $T$ indicates the transpose of the vector/matrix. Note that our activation function $\Phi$ acts element-wise. We also denote the parameters of our model as

$$\boldsymbol{\theta} = \begin{bmatrix} vec\left(\mathbf{W}^{(1)}\right)^T & \mathbf{b}^{(1)} & vec\left(\mathbf{W}^{(2)}\right)^T & \mathbf{b}^{(2)} \end{bmatrix}^T, \tag{5.10}$$

where $vec(\mathbf{W}^{(l)})$ represents the vectorisation of $\mathbf{W}^{(l)}$[†].

[†]In mathematics the vectorisation of a matrix is a linear transformation which converts the matrix into a column vector.

### 5.3.2 Loss Function

The most important part of a neural network is its loss function, as this is what enables the model to train and improve. Neural networks are very versatile and can be used to solve either regression problems or classification problems. The distinction lies in the loss function used.

#### 5.3.2.1 Loss Function for Regression Networks

The loss function is a crucial choice in defining the outputs in a way that is sensitive to the problem at hand (Aggarwal 2018). Let's consider the training data set $\mathcal{T} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{p}$, consisting of $p$ samples of input $\mathbf{x}$ and true output $\mathbf{y}$[‡]. This set is a subset of our full data set $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{q}$, so $\mathcal{T} \subset \mathcal{D}$ and $p < q$. In order to measure the accuracy of our prediction $z$, and therefore train the model, we use the **square error** loss function

$$\boldsymbol{\theta} = \min_{\theta} J(\boldsymbol{\theta}), \tag{5.11}$$

where

$$J(\boldsymbol{\theta}) = \frac{1}{p}\sum_{i=1}^{p} L(\mathbf{x}_i, y_i, \boldsymbol{\theta}) \quad \text{and} \quad L(\mathbf{x}_i, y_i, \boldsymbol{\theta}) = |y_i - f(\mathbf{x}_i, \boldsymbol{\theta})|^2 = |y_i - z_i|^2,$$

where $J$ and $L$ are called the **cost function** and **loss function** respectively and $\boldsymbol{\theta}$ is a matrix representing all the parameters of the entire network. For most multi-layered networks finding a global minimum is computationally infeasible because the number of parameters of the model increases significantly as the network grows. Hence instead of trying to compute exact $\boldsymbol{\theta}$, we will simplify our analysis by making a quadratic approximation of our cost function $J$ in the neighbourhood of the value of the weights that obtains minimal unregularised training cost

$$\hat{\boldsymbol{\theta}} = arg\ \min_{\theta} J(\boldsymbol{\theta}). \tag{5.12}$$

In Eq 5.12, arg min (or arguments of the minima) are the points of the domain of the function at which the cost function values are minimised. To solve this problem, we will need to employ numerical optimisation methods like gradient descent.

#### 5.3.2.2 Loss Function for Classification Networks

Before defining the loss function for a classification network, we need to make a slight modification to our output layer. For us to be able to use our network for binary classification problems, where outputs are either $z \in [0, 1]$, we can change the activation function (see Eq. (5.13)) of our single output node in the output layer to the sigmoid function. For multi-class classification, we can extend our prior network by adding the **softmax** activation function (see Eq. (5.14)) to each of the now multiple nodes (one for each class) in the output layer.

$$\Phi(z) = \frac{1}{1 + e^{-\mathbf{z}}}. \tag{5.13}$$

[‡]note here that we are using vector notation to specify a vector of outputs and vector of inputs

$$\Phi(\mathbf{z}) = \frac{1}{\sum_{j=1}^{K} e^{z_j}} \left[e^{z_1} \dots e^{z_K}\right]^T. \tag{5.14}$$

Adding these functions to the nodes in the final layer(s) of the network, will mean that our final output is determined as follows:

$$\begin{gathered} p(1|\mathbf{x}_i) = \Phi(z) \quad \text{for the sigmoid activation} \\ \text{OR} \\ [p(1|\mathbf{x}_i) \dots p(K|\mathbf{x}_i)] = \Phi(\mathbf{z}) \quad \text{for the softmax activation} \end{gathered} \tag{5.15}$$

The softmax function essentially maps the final layer's output $\mathbf{z}$, which is the weighted sum of outputs from the final layer, to the modelled class probabilities $p(1|\mathbf{x}_i), \dots, p(K|\mathbf{x}_i)$, i.e. $\Phi(\mathbf{z}) : \mathbb{R}^K \to [0,1]^K$. In this case, elements of $\mathbf{z}$ are referred to as **logits**. Also, because of the way the softmax function is constructed, all class outputs from this function will sum to 1. The sigmoid function works similarly, but in that it maps the final layers output $z$ to a single probability, $\Phi(z) : \mathbb{R} \to [0,1]$. These probabilities are then used, in conjunction with classification thresholds, to determine whether a data point belongs to a specific class. The mapping for binary classification is done in the following way:

$$\hat{y} = \begin{cases} 0 & \text{if } z < t_c \\ 1 & \text{if } z \geq t_c, \end{cases} \tag{5.16}$$

where $\hat{y}$ is the predicted class, and $t$ is the classification threshold, which is usually 0.5.

Now we turn our attention to the loss function. As before the training data set $\mathcal{T} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{p}$, consisting of $p$ samples of input $\mathbf{x}$ and true output $\mathbf{y}$. For multi-class classification we will use what is called **one-hot encoding** for output $\mathbf{y}_i$. This works as follows: for a problem with $K$ different classes (i.e. $\mathbf{y}_i$ consists of $K$ elements $\mathbf{y}_i = [y_{i1} \dots y_{iK}]^T$), if point $i$ belongs to class $k$ then $y_{ik} = 1$ and $y_{ij}$ for $j \neq k$. For our multi-class classification network we will use the **cross-entropy** loss function

$$L(\mathbf{x}_i, \mathbf{y}_i, \boldsymbol{\theta}) = -\sum_{k=1}^{K} y_{ik} \log p(k|\mathbf{x}_i, \boldsymbol{\theta}). \tag{5.17}$$

This function approaches its minimum as $p(k|\mathbf{x}_i, \boldsymbol{\theta})$ approaches 1 for $k$ where $y_{ik} = 1$. For our binary classification network we will use the **binary cross-entropy** loss function

$$L(\mathbf{x}_i, \mathbf{y}_i, \boldsymbol{\theta}) = -\left(y \log p(1|\mathbf{x}_i, \boldsymbol{\theta}) + (1-y) \log(1 - p(1|\mathbf{x}_i, \boldsymbol{\theta}))\right). \tag{5.18}$$

## 5.4 A Deep Neural Network Model

Now that we have the essential parts of our model architecture set up and defined, we can now increase the depth (number of layers) of our model. To realise an ANN's real descriptive power and to model complicated relationships, we need to stack intermediate or hidden layers. We construct a deep neural network, i.e. any ANN with two or more hidden layers called a deep neural network.

We begin by enumerating the layers of the model with index $l$. Again, each layer has its own set of parameters $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$. There are also multiple layers of hidden units and each of those layers have $M_l$ hidden units $\mathbf{h}^{(l)} = \left[h_1^{(l)}, ..., h_{M_l}^{(l)}\right]$. Mapping from layer $l-1$ to layer $l$ is denoted as

$$\mathbf{h}^{(l)} = \Phi(\mathbf{W}^{(l)T}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)T}). \tag{5.19}$$

A DNN with $L$ layers can be described mathematically as

$$\begin{aligned}
\mathbf{h}^{(1)} &= \Phi(\mathbf{W}^{(1)T}\mathbf{x} + \mathbf{b}^{(1)T}),\\
\mathbf{h}^{(2)} &= \Phi(\mathbf{W}^{(2)T}\mathbf{h}^{(1)} + \mathbf{b}^{(2)T}),\\
&\vdots\\
\mathbf{h}^{(L-1)} &= \Phi(\mathbf{W}^{(L-1)T}\mathbf{h}^{(L-2)} + \mathbf{b}^{(L-1)T}),\\
z &= \mathbf{W}^{(L)T}\mathbf{h}^{(L-1)} + \mathbf{b}^{(L)T},
\end{aligned} \tag{5.20}$$

with parameters

$$\boldsymbol{\theta} = \left[vec\left(\mathbf{W}^{(1)}\right)^T \quad \dots \quad vec\left(\mathbf{W}^{(L)}\right)^T \quad \mathbf{b}^{(1)} \quad \dots \quad \mathbf{b}^{(L)}\right]^T. \tag{5.21}$$

Finally, it is important to take note of the different dimensions of each layer's parameters. Note that, in DL it is not uncommon to have multiple outputs, so $z$ could be a vector $\mathbf{z} = [z_1, ..., z_K]$. Table 5.1 provides a summary of these dimensions.

| **Layer** | **Weight Matrix** | **Offset Vector** |
|---|---|---|
| $l = 1$ | $\mathbf{W}^{(1)} : n \times M_1$ | $\mathbf{b}^{(1)} : 1 \times M_1$ |
| $l = 2, ..., L-1$ | $\mathbf{W}^{(l)} : M_{l-1} \times M_l$ | $\mathbf{b}^{(l)} : 1 \times M_l$ |
| $l = L$ | $\mathbf{W}^{(L)} : M_{L-1} \times K$ | $\mathbf{b}^{(L)} : 1 \times K$ |

Table 5.1: Parameter dimensions per layer.

## 5.5 Training a Network

In order for us to construct a model that can make good predictions, we need to find an estimate for $\boldsymbol{\theta}$, the actual set of network parameters. To do this, we need to solve

$$\hat{\boldsymbol{\theta}} = arg\ \min_{\theta}\ J(\boldsymbol{\theta}) \quad \text{where} \quad J(\boldsymbol{\theta}) = \frac{1}{p}\sum_{i=1}^{p} L(\mathbf{x}_i, \mathbf{y}_i, \boldsymbol{\theta}). \tag{5.22}$$

The form of function $J$ depends on the problem at hand. The optimisation problem in Eq. (5.22) cannot be solved in closed-form and requires the use of numerical optimisation. For DNNs, it is common to use gradient descent methods.

We start off by initialising $\boldsymbol{\theta}_0$, we then update the parameters as $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \lambda\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t)$ for $t = 1, 2, ...$ and then stop once some criteria is met and set $\boldsymbol{\theta}$ equal to the last $\boldsymbol{\theta}_t$. $\lambda$ is the step size and is often referred to as the **learning rate**.

Picking $\boldsymbol{\theta}_0$ could be challenging since cost functions in the training of neural network are not necessarily convex. It means that training is very sensitive to $\boldsymbol{\theta}_0$ (convexity being ideal since it guarantees convergence regardless of $\boldsymbol{\theta}_0$). Typically initialisation is random, and each parameter is set to some random number such that we ensure that different hidden units encode different aspects of the data. If the ReLU function is used, the elements of $\omega_0$ are initialised to small, non-negative numbers to operate in the positive range of the ReLU function.

### 5.5.1 Back-Propagation and Stochastic Gradient Descent

Back-propagation (BP) is widely used in training feed-forward NNs. While we are training the NN, we will compute the gradient of $L$ w.r.t. to the weights/parameters of the network. The BP algorithm can efficiently calculate the gradient, which works very well with optimisation methods such as gradient descent or stochastic gradient descent. Together these methods update the weights to minimise loss. BP works by computing the gradient of the $L$ w.r.t. each weight by using the chain rule, computing the gradient layer by layer and then iterating backwards from the last layer to avoid redundant calculations (Goodfellow et al. 2016).

When working with multi-layer networks, the loss function becomes a complex composition of all weights in every layer. The gradient of this loss function is computed using the BP algorithm. As mentioned before, this algorithm uses the chain rule of differential calculus, which allows us to represent the error gradients as a sum of local gradient products over the various paths from a node to the output. As the network becomes wider and deeper and the number of components (paths) increases, the size of the error gradient increases exponentially. However, using BP, which is a direct application of dynamic programming, we can compute the gradient efficiently. BP consists of two main phases, the forward phase and the backward phase.

1. **Forward Phase:** Here a single $\mathbf{x}$ with the current $\boldsymbol{\theta}_i$ is fed forward through the network resulting in calculations being done layer by layer from input layer to output layer resulting in an output $\mathbf{z}$. This output is then compared to the true output $\mathbf{y}$ and we calculate $\lambda \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ which is the derivative of the cost function w.r.t. $\boldsymbol{\theta}_i$ in all layers in the backwards phase.

2. **Backward Phase:** Next, the algorithm will attempt to approximate the actual gradient of the loss function w.r.t. different $\boldsymbol{\theta}_i$ by applying the chain rule. This approximation, along with the step size, is used to update $\boldsymbol{\theta}_i$. From here, the forward phase starts again until a stopping criterion is reached.

### 5.5.2 Gradient Descent Optimisation

The next consideration is the method of optimisation to use. As eluded to earlier, we will now be looking at **stochastic gradient descent** (SGD) which was first presented by Robbins & Monro (1951). In stochastic (or "on-line") gradient descent, the true gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ for example, is approximated by $\hat{\boldsymbol{g}}_t$, the gradient at a single instance

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \lambda \hat{\boldsymbol{g}}_t. \tag{5.23}$$

As the algorithm runs through the training set, it performs the above update after every training pass. The number of passes made over the training set is called **epochs**

(denoted by $E$). After each epoch, the data is shuffled to prevent cycles. In pseudocode, SGD can be described as follows (Lindholm et al. 2019):

**Algorithm 1:** Stochastic gradient descent algorithm

Initialise all the parameters $\boldsymbol{\theta}$ in the network and set $t \leftarrow 1$.;
**for** $k = 1$ *to* $E$ **do**
  a.) Randomly shuffle the training data $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{p}$;
  b.) Approximate the gradient of the loss function using $\hat{\boldsymbol{g}}_t = \frac{1}{p}\sum_{i=1}^{p} \nabla_{\boldsymbol{\theta}} L(\mathbf{x}_i, \mathbf{y}_i, \boldsymbol{\theta})$
  c.) Do a gradient step $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \lambda \hat{\boldsymbol{g}}_t$
  d.) Update the iteration index $t \leftarrow t + 1$
**end**

The question now arises, how well does SGD perform theoretically in terms of minimising $J(\boldsymbol{\theta})$? When it comes to the convex case, it is well understood in the literature that, with proper choices of the step sizes $\lambda$, SGD is guaranteed to achieve both consistency and asymptotic normality (Robbins & Monro 1951, Kiefer et al. 1952, Polyak & Juditsky 1992, Bottou 1998, Kushner & Yin 2003).

However, this all changes when we consider non-convex loss functions in DL. In this case, finding a global minimum is computationally infeasible at worst. Fortunately, recent work by Du et al. (2019) and Allen-Zhu et al. (2019) finds a way around this problem. They show that, as long as the NN is sufficiently over-parametrised, SGD converges linearly towards a global minimum. If this condition is met and there is also random initialisation of parameters, faster convergence of the SGD algorithm is ensured.

As Fan et al. (2019) notes, basic SGD does come with a set of challenges when training DNNs. Firstly, slow convergence even though there are theoretical guarantees for well-behaved problems. Second, the computational cost of calculating the gradient for the entire training set for each epoch. Lastly, the learning rates ($\lambda$) can be challenging to tune in practice. In the following section, we will consider a few variants of SGD, namely mini-batch SGD, momentum-based SGD, and SGD with adaptive learning rates.

### 5.5.3 Learning Rate

The learning rate $\lambda$, or the step size of the gradient descent algorithm, is a crucial hyper-parameter in SGD. A hyper-parameter is one whose value is used to adjust the learning procedure. Another hyper-parameter, for example, is the number of epochs. There are many more hyper-parameters to tune, which will be introduced in the next chapter.

When picking $\lambda$, one needs to consider the trade-off between the rate of convergence and the likelihood of overshooting the global minima. While the direction of descent determined by $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$, the $\lambda$ determines the step size in that direction (Nesterov 2013). When working with SGD, one strategy is to pick a constant $\lambda$. However, there are risks to this strategy, as too big a step size may cause the algorithm to over-shoot the minima, and too small a step could cause our algorithm to converge very slowly (as illustrated in Figure 5.6).

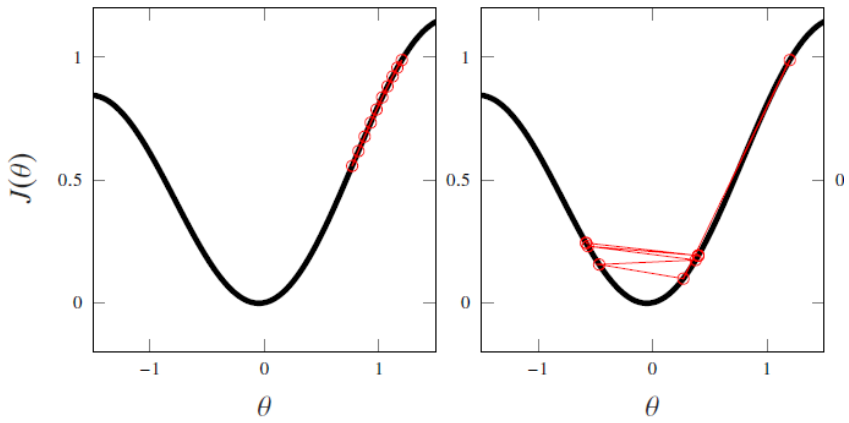



Figure 5.6: Illustration of the consequences of incorrect learning rate in finding the minima of the objective function. In these graphs, we plot the value of the objective function against the network's parameters. On each plot, we see the red points on the curve, which represent the incremental updates of the estimate for $\boldsymbol{\theta}$, which shows the step sizes. On the left, we see that step sizes that are too small may cause the algorithm to converge very slowly. On the right, we see that step sizes that are too big may cause the algorithm to over-shoot and oscillate over the minima (also causing slow convergence, if it even converges at all).

(Source: (Lindholm et al. 2019))

To ensure faster convergence and prevent oscillations, the learning rate is updated as training progresses (as in Figure 5.7). This updating is done either according to a learning rate schedule, using learning rate decay or using what is referred to as adaptive learning techniques such as momentum-based learning techniques or parameter-specific learning rates (Lau 2017). These will be discussed in more detail in the next chapter.

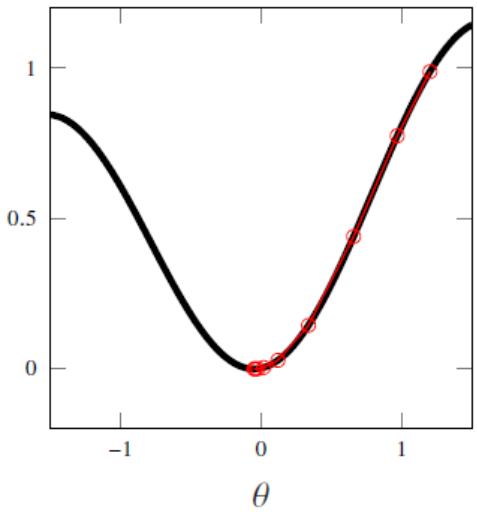



Figure 5.7: An illustration of the benefits of an adaptive learning rate in finding the minima of the objective function. Here, as we approach the minima, the descent algorithm adapts the step sizes by making the steps smaller and therefore slow down to not over-shoot.

(Source: (Lindholm et al. 2019))

## 5.6 Conclusion

A neural network is a powerful tool. The issue that is made clear in this chapter is that a few parameters require some expertise to "tune" correctly to realise the full potential of a neural network. The reality is that the more complex the function we are trying to approximate, the more complex our neural network and the more numerous the parameters are to tune (which becomes harder to tune correctly).

In the next chapter, we will consider some challenges faced in deep learning and proposed techniques to approach them. We will also consider various proposed techniques to fine-tune model performance and efficiency.

# Chapter 6

# Challenges and Advancements in Deep Learning

> "But spectacular advances in information technology suggest we are approaching a historical discontinuity in humanity's relationship with machines" - *Phillip E. Tetlock, Super Forecasting*

## 6.1 Introduction

In this chapter we take a look at recent advances in the field of deep learning and how they address some of the issues experienced with training deep networks. We explore various methods that were designed to increase the accuracy, execution speed and reliability of neural networks.

Despite NN's reputation as universal function approximators, there still exists some considerable challenges w.r.t. actually training them to provide this level of performance. The most important of these challenges is most certainly overfitting (Aggarwal 2018). In addition, computational efficiency and convergence is also of high importance hence we will look at methods that help us to overcome these obstacles.

## 6.2 Mini-Batch Gradient Descent

Mini-batch gradient descent (MBGD) is a variation of the gradient descent algorithm that splits the training data into smaller batches. These smaller batches are used to calculate the prediction error of the model and update model coefficients accordingly.

Suppose that when training the network, $p$ (the size of our data set) is very large (in the order of hundreds of thousands of data points). The BP algorithm's execution might be very costly. We, therefore, assume that many of the data points are similar. This assumption means that we can also assume the gradient for, e.g. the first half of the data set is the same as for the other half, i.e.

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \approx \sum_{i=1}^{\frac{p}{2}} \nabla_{\boldsymbol{\theta}} L(\mathbf{x}_i, \mathbf{y}_i, \boldsymbol{\theta}) \approx \sum_{i=\frac{p}{2}+1}^{p} \nabla_{\boldsymbol{\theta}} L(\mathbf{x}_i, \mathbf{y}_i, \boldsymbol{\theta}), \tag{6.1}$$

where all variables are defined as in Chapter 5.

It is, therefore, unnecessary to compute the gradient for the entire training set. So we proceed to calculate the gradient based on the first half of the data set, then update the

parameters and calculate the gradient again based on the last half of the data. Hence, the computation time is roughly reduced by half.

An extreme variation of this approach could be to compute the gradient after each step. However, this would be very costly computationally. MBGD is a good compromise. Before using the algorithm, we need to define $p_{mb}$, which is the size of each mini-batch. Hence we perform an update of the gradient after each mini-batch of $p_{mb}$ training examples using

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \lambda \sum_{i=1}^{p_{mb}} \nabla_{\boldsymbol{\theta}} L(\mathbf{x}_i, \mathbf{y}_i, \boldsymbol{\theta}_t). \tag{6.2}$$

So, in simple terms, we are updating the gradient more frequently for smaller subsets of the data to improve computational efficiency. Algorithm 1 can therefore be extended by using smaller sets for training. The size of a mini-batch can range from $p_{mb} = 10$ to $p_{mb} = 100$ to $p_{mb} = 1000$, depending on the size of the overall data set. It is important is to ensure these mini-batches represent the data set by drawing the points at random. This random drawing is achieved by shuffling the data before dividing the data set. This modified procedure executes as follows:

**Algorithm 2:** Mini-batch gradient descent algorithm. Source: (Lindholm et al. 2019)

Initialise all the parameters $\boldsymbol{\theta}$ in the network and set $t \leftarrow 1$.;
**for** $k = 1$ *to* $E$ **do**
  a.) Randomly shuffle the training data $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{p}$;
  b.)**for** $j = 1$ *to* $\frac{p}{p_{mb}}$ **do**
    I.) Approximate the gradient of the loss function using the mini-batch $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=(j-1)p_{mb}+1}^{jp_{mb}}$
      $\hat{\boldsymbol{g}}_t = \frac{1}{p_{mb}} \sum_{i=(j-1)p_{mb}+1}^{jp_{mb}} \nabla_{\boldsymbol{\theta}} L(\mathbf{x}_i, \mathbf{y}_i, \boldsymbol{\theta})|_{\theta=\theta_t}$
    II.) Do a gradient step $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \lambda \hat{\boldsymbol{g}}_t$
    III.) Update the iteration index $t \leftarrow t + 1$
  **end**
**end**

## 6.3 Regularisation Techniques

Building a DNN that generalises well is not a simple task due to the issue of overfitting. Overfitting occurs when our model performs well on the training set but predicts poorly on unseen data, i.e. it does not generalise well. Overfitting is the biggest issue in applied ML, and it happens when a model learns too well. It starts to explain the noise in the training data, i.e. the random fluctuations in the training data are learned as concepts by the model. The problem is that this noise does not apply to unseen data. We want to limit this as much as possible as it negatively impacts the model's performance on new data. The more complex the model, the more likely overfitting becomes.

It is common practice when evaluating ML algorithms to hold back a validation data set to monitor and limit overfitting. This subset of the training data is held separate from our training data. After the initial setup of our model, it will learn on the training set while the validation set is used throughout training to evaluate the learned model. This ongoing evaluation provides us with an idea of how the model might perform out of

sample, i.e. on unseen data and helps us monitor overfitting. If we have a large set of data, using a validation data set is highly recommended.

The most important thing to keep in mind during this section is that a neural network learns a set of weights that best map inputs to outputs. The longer we train the network, the more specialised the weights will become to the training data, overfitting the training data. The weights will grow in size to handle the specifics of the examples seen in the training data. Large weights make the network unstable. Although the weight will be specialised to the training data set, minor variation or statistical noise on the expected inputs will result in significant differences in the output.

One approach to improve generalisation is to use a high capacity model and apply regularisation during training. In ML, regularisation is the process of adding information in order to solve an ill-posed problem or to prevent overfitting (Buühlmann & Geer 2012). Regularisation can be seen as a way to control the learning process to avoid overfitting. Regularisation has proven to reduce overfitting, lead to faster optimisation of the model and greater general performance. We will look at four of the most common and effective techniques.

### 6.3.1 Parameter Norm Penalty

This form of regularisation is one of the simplest and perhaps most common regularisation methods. It entails simply adding a penalty to $L$, the loss function in proportion to the size of the weights of our model.

We penalise the loss function based on complexity. Of all functions $f$, $f = 0$ is surely the most simple function, hence measuring a functions complexity would entail simply measuring its distance from zero. One way we can do this for a function $f(\mathbf{x}) = \boldsymbol{\omega}^T\mathbf{x}$ is by calculating $||\boldsymbol{\omega}||^2$. We then add the norm as a penalty term to the minimisation problem by replacing the objective function $J(\boldsymbol{\theta})$ with $J^R(\boldsymbol{\theta})$ as follows

$$J^R(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \alpha R(\boldsymbol{\theta}) \quad \text{where } \alpha \in \mathbb{R}^+, \tag{6.3}$$

where $R(\theta)$ is called the regularisation term. Adding this penalty term to the cost function and penalising larger weights more heavily (larger penalty), we force the optimisation algorithm to push the model to have smaller weights. In other words, we want to keep the weights no larger than needed to perform well on the training data set.

The two most popular penalty methods are $L_1$ and $L_2$ regularisation*. In these methods, the norm used in the regularisation term is the 1- and 2-norm, respectively.

#### 6.3.1.1 $L_2$ parameter regularisation

$L_2$ or Tikhonov regularisation is one of the most common forms of regularisation and has been very effective over the past few decades for simple linear models and NNs. It is also known as ridge regression or weight decay.

When learning a linear function $f(\mathbf{x}) = \boldsymbol{\omega}^T\mathbf{x}$, one can add the $L_2$-norm of the vector $\boldsymbol{\omega}$ to the objective function† in order to prefer solutions with smaller norms. Here $R(\boldsymbol{\theta})$ from before is $\frac{1}{2}||\boldsymbol{\omega}||_2^2 = \frac{1}{2}\sum_i \omega_i^2 = \frac{1}{2}\boldsymbol{\omega}^T\boldsymbol{\omega}$. The objective function is expressed as:

*__Elastic Net__ is a regularised regression method that linearly combines the $L_1$ and $L_2$ penalties of the LASSO and ridge methods.

†note that only the weight vector is penalised and not the bias vector. The biases typically require less data to fit accurately than the weights.

$$J^R(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \alpha||\mathbf{w}||_2^2 \quad \text{where } \alpha \in [0, \infty) \tag{6.4}$$
$$= J(\alpha) + \frac{\alpha}{2}\boldsymbol{\omega}^T\boldsymbol{\omega}. \tag{6.5}$$

and our objective function in Eq. (5.12) becomes

$$\nabla_{\boldsymbol{\theta}} J^R(\boldsymbol{\theta}) = \alpha\boldsymbol{\omega} + \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}). \tag{6.6}$$

When taking a single gradient step we make the update

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \nabla_{\boldsymbol{\theta}} J^R(\boldsymbol{\theta}_t). \tag{6.7}$$

So we have multiplicatively shrunk $\boldsymbol{\omega}$ by a constant factor on each step prior to performing a gradient update.

#### 6.3.1.2 $L_1$ parameter regularisation

$L_1$ regularisation is another common form of regularisation and is also known as least absolute shrinkage and selection operator; or LASSO. It is used most commonly in feature selection. It simplifies the ML problem by picking the best subset of features. Here $R(\boldsymbol{\theta})$ from before is $\frac{1}{2}||\boldsymbol{\omega}||_1 = \frac{1}{2}\sum_i |\omega_i|$. The objective function is expressed as:

$$J^R(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \alpha||\mathbf{w}||_1 \quad \text{where } \alpha \in [0, \infty) \tag{6.8}$$
$$= J(\alpha) + \frac{\alpha}{2}\sum_i |\omega_i|, \tag{6.9}$$

and our objective function in Eq. (5.12) becomes

$$\nabla_{\boldsymbol{\theta}} J^R(\boldsymbol{\theta}) = \frac{\alpha}{2}\sum_i \frac{\omega_i}{|\omega_i|} + \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}). \tag{6.10}$$

### 6.3.2 Weight Constraint

Unlike weight regularisation, a weight constraint is a trigger that forces weights above a pre-defined magnitude threshold to be below that threshold, which forces weights to be small. It is very similar to parameter norm penalties in that both force the weights to be small. The critical difference is that weight constraint directly keeps the weights small while the penalty method achieves it by penalising the cost function.

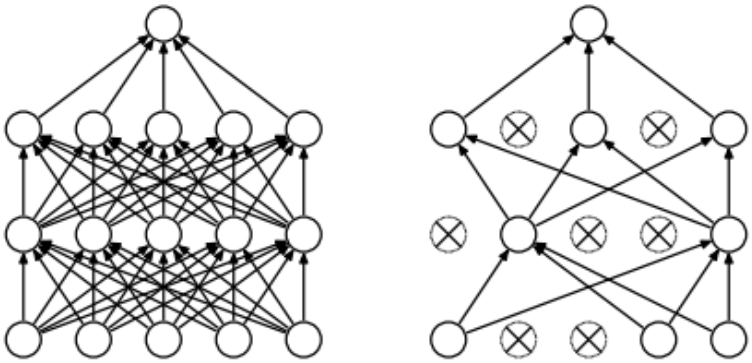
Some examples of constraints that could be used include limiting $||\boldsymbol{\omega}||_1 < 1.0$ or limiting $||\boldsymbol{\omega}||_\infty$. The $L_\infty$-norm, also called max-norm, is a popular constraint because it is less aggressive than other norms such as the unit norm, simply setting an upper bound.

### 6.3.3 Dropout

Dropout (Hinton, Srivastava, Krizhevsky, Sutskever & Salakhutdinov 2012, Srivastava et al. 2014) is the practice of randomly dropping out subsets of features during training. It was a significant discovery in deep learning that solved the issue of overfitting and

expanded the possibilities of NNs. The main problem it was able to solve was co-adaption, where, if all weights are trained together, some connections have more predictive power than others. The strength of these connections is being overstated because they are over-trained. Some connections may have shown promising results in a few training epochs. Thus the algorithm keeps promoting it even though it might not necessarily be correct, in turn leading to weaker connections (which might be of importance) being under-trained to the point that they stop participating. Weight decay might seem like a good solution; however, this technique regularises based on predictive abilities to exacerbate the issue. Some features become close to deterministic in choosing and rejecting weights (Srivastava et al. 2014). Dropout has the effect of making the training process noisy. It forces nodes within a layer to probabilistically take on more or less responsibility for the inputs (Brownlee 2018).

With dropout, we drop nodes from the network randomly for each epoch. This process creates a sub-network of the original network, as illustrated in Figure 6.1. Sampling is done randomly with pre-defined probability $\rho$, of which units to drop. Each collection of dropped units is independent of the collection of dropped units in another sub-network. Units that are dropped also have their connection (incoming and outgoing) dropped. Hidden units, as well as input nodes, can be dropped. The benefit is that since each network is a sub-network of the same original network, they all share some parameters. This sharing of parameters means that we can train the ensemble of sub-networks in an efficient manner.



(a) Standard Neural Net (b) After applying dropout.

Figure 6.1: Dropout Neural Network Model. Left: A standard neural net with 2 hidden layers. Right: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

(Source: (Srivastava et al. 2014))

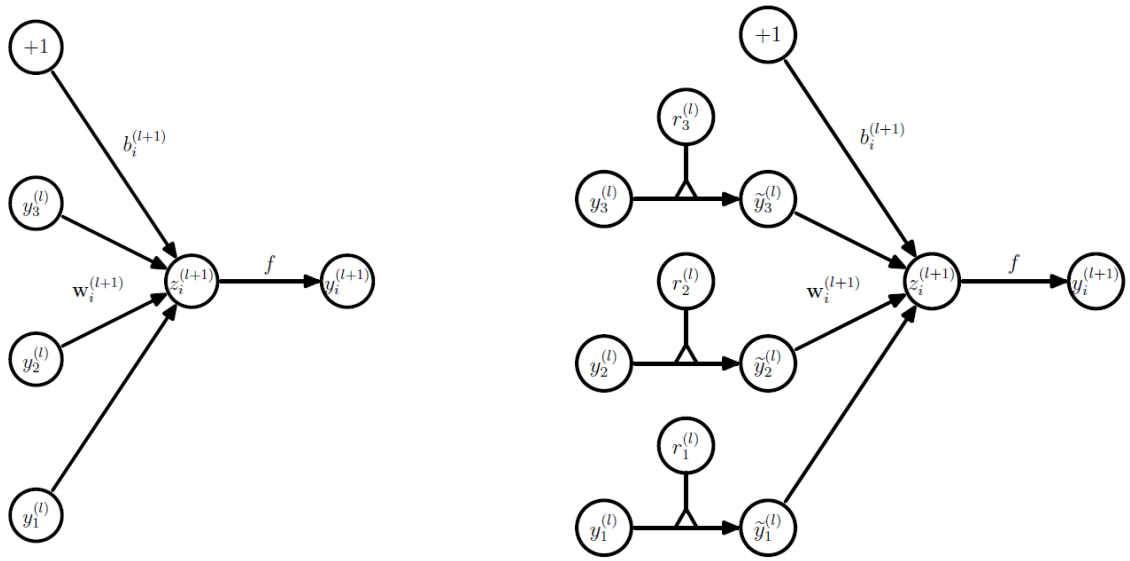
When training with dropout, we use MBGD. Each mini-batch at each gradient step of the algorithm approximates the gradient and is computed as before. The difference is that instead of computing the gradient for the entire network, we only have to compute it for a random sub-network. The gradient is computed as if the dropped units were not present, and a gradient step is then performed. Updates are only made to parameters present in the sub-network. This process is repeated until some stopping condition is met.

For the $l$-th layer of our NN, instead of propagating all the features in $\mathbf{h}^{(l)}$ for later computations, the dropout procedure randomly omits some of its entries by

$$\mathbf{h}^{(l)}_{drop} = \mathbf{h}^{(l)} \odot \mathbf{mask}^{l}, \tag{6.11}$$

where $\odot$ represents element-wise multiplication and $\mathbf{mask}^l$ is a vector of i.i.d Bernoulli random variables[‡] with $P(\delta_i = 1) = \rho$ (see Figure 6.2).

$$\mathbf{mask}^l = \begin{bmatrix} \delta_1 \\ \vdots \\ \delta_M \end{bmatrix} \quad \text{where } \delta_i \sim Bernoulli(\rho). \tag{6.12}$$




(a) Standard network

(b) Dropout network

Figure 6.2: Comparison of the basic operations of a standard and dropout network. The mask is applied to each incoming connection of the nodes in the layer. The mask acts as a type of "information gate". If $\delta_i = 1$ the information is permitted to flow through whereas $\delta_i = 0$ means that the connection is blocked, essentially dropping that connection.

(Source: (Srivastava et al. 2014))

The outputs of each layer, $\mathbf{h}^{(l)}$, are multiplied (element-wise) with the mask vector, which is re-sampled each time. This creates $\mathbf{h}^{(l)}_{drop}$. These thinned outputs are then used as input in the next layer. This process is repeated at each layer. For learning, the derivatives of the $L$ are back-propagated through the sub-network. When we test our network on the test data set, we scale the weights using $\omega^{(l)}_{train} = \rho\omega^{(l)}$. So, the NN used for testing is used without dropout. The final node output for a classification network is modified as

$$p(1|\mathbf{x}_i) = \sigma\left[\mathbf{W}^{(L)T}\left(\mathbf{h}^{(L-1)} \odot \mathbf{mask}^{(L-1)}\right) + \mathbf{b}^{(L)T}\right]. \tag{6.13}$$

### 6.3.4 Early Stopping

Early stopping is a regularisation technique that prevents overfitting when training a model using an iterative method, like SGD or MBGD. These methods update the model parameters to improve the fit on the training data with each iteration. This process improves the model's out-of-sample performance up to a point after which further improving

[‡]There is also a variation of dropout called Gaussian Dropout where $\delta_i \sim \mathcal{N}(1, \sigma^2)$

the fit to the training data leads to a rise in the out-of-sample error or overfitting starts to occur. Early stopping involves stopping training when this point is reached or when the model's performance no longer improves. The rule for early stopping provides guidance as to how many iterations can be run before the model starts to overfit. These rules have been used in various ML methods, with varying amounts of theoretical foundation.

As mentioned before, we will have a validation set apart from our training set during training to help monitor the generalisation error. If this error rises while the error of the training set continues to decrease (as in Figure 6.3), we know that the model is overfitting. At this point, we wish to stop the training as training longer will only make the model worse. However, it is essential not to stop after a slight rise in the out-of-sample error; in fact, it is advisable to continue training just a few more epochs to be sure. This is sometimes referred to as the patience parameter (Brownlee 2020*b*).

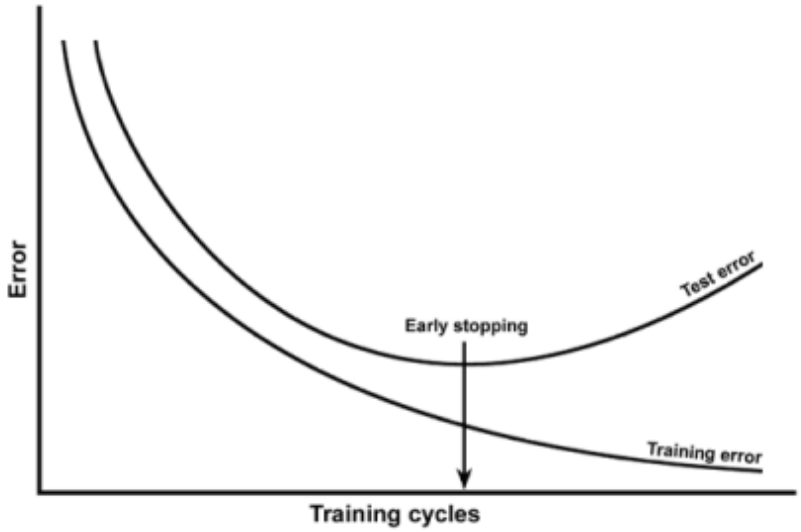



Figure 6.3: Illustration of when to terminate training early to prevent overfitting. As the error in the validation test set begins to rise while the error in the training set continues to decrease, the algorithm will terminate training as overfitting is occurring.

(Source: (Flitton 2018))

One way to think of early stopping is as a very efficient hyper-parameter selection algorithm. In this view, the number of training steps is just another hyper-parameter (Goodfellow et al. 2016). The stopping point is determined in hindsight. Each time the out-of-sample error decreases, we save a copy of the parameters. As soon as the algorithm terminates, the best set of parameters is recalled and chosen as our final set of parameters. The algorithm terminates when no parameters have improved over the best-recorded validation error ($\epsilon_{validation}$) for some pre-specified number of iterations (called patience). This procedure is specified more formally in Algorithm 3. The advantages of this regularisation technique are clear: it is easy to implement, can be used in conjunction with other techniques, and is effective. The drawback is that we require additional data for validation. Consider the following pseudo-code:

**Algorithm 3:** The early stopping meta-algorithm for determining the best amount of time to train. Source: (Lindholm et al. 2019)

Let $n$ be the number of steps between evaluations.
Let $p$ be the "patience", the number of times to observe worsening validation set error ($\epsilon_{validation}$) before stopping.
Let $\boldsymbol{\theta}_0$ be the initial parameters.
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_0$
$i \leftarrow 0$
$j \leftarrow 0$
$v \leftarrow \infty$
$\boldsymbol{\theta}^* \leftarrow \theta$
$i^* \leftarrow i$
**while** $j < p$ **do**
  Update $\boldsymbol{\theta}$ by running the training algorithm for $n$ steps.
  $i \leftarrow i + n$
  $\acute{v} \leftarrow \epsilon_{validation}(\boldsymbol{\theta})$
  **if** $\acute{v} < v$ **then**
    $j \leftarrow 0$
    $\boldsymbol{\theta}^* \leftarrow \theta$
    $i^* \leftarrow i$
    $v \leftarrow \acute{v}$
  **end**
  **else**
    $j \leftarrow j + 1$
  **end**
**end**
**Result:** Best parameters are $\boldsymbol{\theta}^*$ , best number of training steps is $i^*$

### 6.3.5 Regularisation Recommendations

If our training set $\mathcal{T}$ does not consist of millions of points, it is recommended that you include some forms of regularisation from the start (Goodfellow et al. 2016). The general rule of thumb is to design an under-constrained NN that uses regularisation to prevent overfitting. Early stopping is almost universally recommended in addition to methods such as those mentioned in this section Goodfellow et al. (2016) suggests some combinations:

- **Classical** - using early stopping with weight decay.
- **Alternate** - using early stopping with weight constraint.
- **Modern** - using early stopping and dropout with weight decay.

## 6.4 Batch Normalisation

Training DNNs with many layers can be challenging due to their sensitivity to initial random weights and learning algorithm configuration. One possible reason for this difficulty is that the distribution of the inputs to layers deep in the network may change after each mini-batch when the weights are updated. This will slow down or even prevent the convergence of the learning algorithm. This change in distribution is called **internal covariate shift**.

Batch normalisation is a technique for training DNNs and works by standardising inputs to a layer after each mini-batch. This technique stabilises the learning process

and significantly reduces the training time required. The algorithm does this by scaling the output of the layer per mini-batch. Standardisation refers to rescaling the data to have a mean of zero and a standard deviation of one, e.g. a standard Gaussian. Batch normalisation can also reduce the generalisation error and, when used, allows dropout to be omitted (due to the noise in the estimate of the statistics used to normalise each variable) (Goodfellow et al. 2016).

## 6.5 Learning Rates

While normal MBGD may provide an improvement over SGD, there are still issues as good convergence is not guaranteed. The previous chapter also clarifies that a constant learning rate is not desirable because it poses a dilemma to the analyst. The challenge is the choice of learning rate. If $\lambda$ is too small, it can lead to slow convergence, while large $\lambda$ can lead the algorithm to quickly approach the minima but then oscillate around the point for a long time, or diverge in an unstable way, if the high rate of learning is maintained. The basic approach here should be straightforward: update the learning rate as training progresses.

One approach is to use learning rate schedules (Robbins & Monro 1951) to try to adjust the learning rate during training by reducing $\lambda$ according to a pre-defined schedule or when the change in objective between epochs falls below a threshold. The issue, however, is that these schedules and thresholds must be defined in advance and cannot adapt to our data set's characteristics (Darken et al. 1992). Additionally, if our data set is sparse, and the features differ in frequencies, updating them to the same extent might not be desirable. We might want to perform more significant updates for rarely occurring features.

Another challenge we might face when training NNs is minimising highly non-convex error functions and avoiding getting stuck at saddle points that are usually located at or near plateaus of values with similar errors. This makes it notoriously hard for algorithms like MBGD to escape, as the gradient is close to zero in all dimensions. We will discuss several strategies to avoid these issues.

### 6.5.1 Learning Rate Decay

Learning rate decay involves setting the learning rate to a decreasing function of the epoch. So as training progresses, the step sizes get smaller. Two of the most common decay functions are inverse and exponential decay. Before we continue, we have to define $\lambda_t$ as the learning rate during epoch $t$. For each model respectively, $\lambda_t$ is defined as

$$\lambda_t = \frac{\lambda_0}{1 + kt}, \quad \text{or} \tag{6.14}$$

$$\lambda_t = \lambda_0 e^{-kt}. \tag{6.15}$$

Here $\lambda_0$ represents the initial step size, and $k$ is the parameter determining the rate at which step sizes decay. Another simple approach is to reduce the step size by a particular factor after every few epochs, e.g. the learning rate might be multiplied by some factor every few epochs. The approach in practice is to track the loss of the validation set and then reduce the learning rate as soon as the validation error stops decreasing. This approach is very commonly used along with different gradient descent algorithms. However, it does not address many of the other issues discussed.

Another approach involves starting with a relatively high learning rate $\lambda_{max}$ and then having it decay to a certain level $\lambda_{min}$ using the rule

$$\lambda_t = \lambda_{min} + (\lambda_{max} - \lambda_{min})e^{-\frac{t}{\tau}}. \tag{6.16}$$

So as $t \to \infty$, $\lambda_{max} \to \lambda_{min}$. However, picking $\lambda_{min}$ and $\lambda_{max}$ is more an art than an exact science. Under certain conditions and if the Robbins-Monro condition (Robbins & Monro 1951) holds: $\sum_{t=1}^{\infty} \lambda_t = \infty$ and $\sum_{t=1}^{\infty} \lambda_t^2 < \infty$, then the SGD algorithm almost surely converges to a local minima. In order to satisfy the Robbins-Monro condition, we need $\lambda_t \to 0$ as $t \to \infty$. However, this is not typically the case in practice, so we instead choose $\lambda_{min} > 0$ and use a scheme like Eq. (6.16). This has proven to work better in most cases while we knowingly sacrifice theoretical convergence of the algorithm (Lindholm et al. 2019).

### 6.5.2 Momentum-Based Learning Techniques

SGD has trouble navigating ravines (areas where the surface curves much more steeply in one dimension than in another). In this case, the SGD algorithm will oscillate across the ravine slopes while making plodding progress towards the local minima. Consider Figure 6.4. The figure represents an objective function surface, with the inner-most part representing the local minimum. The lines illustrate how the standard SGD algorithm would oscillate down this ravine-like slope whilst approaching the minimum.

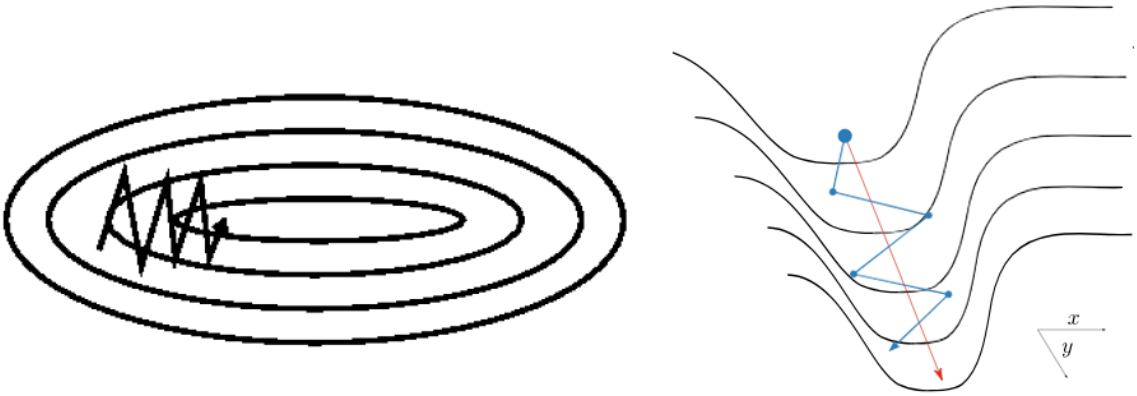



Figure 6.4: An illustration of SGD without momentum.

(Source: (Orr 1999))

These oscillations result from highly contradictory steps that cancel out one another and slow the progress in the correct direction. Momentum (Qian 1999) is a method that accelerates the SGD algorithm in the correct direction and dampens the effect of oscillations. This is achieved by adding a fraction $\gamma$ of the update vector $\mathbf{v}_t$ to the current update vector $\mathbf{v}_{t+1}$. Essentially in momentum-based descent, $\mathbf{v_t}$ is modified with exponential smoothing, where $\gamma \in (0, 1)$ is a smoothing parameter

$$\mathbf{v}_{t+1} = \gamma \mathbf{v}_t + \lambda \hat{\boldsymbol{g}}_t, \tag{6.17}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \mathbf{v}_t, \tag{6.18}$$

with $\gamma$ usually initialised to 0.9. Large $\gamma$ values assist in picking up a consistent velocity in the correct direction. When $\gamma = 0$, Eq. (6.17) is just plain SGD or MBGD. We can therefore think of $\gamma$ as the **momentum parameter** or the **friction parameter**[§]. In essence, our momentum term will increase at points where gradients point in the same directions and decrease where gradients change directions. As a result, we gain faster convergence (momentum) in the correct direction, and the effect of oscillations will be reduced as shown in Figure 6.5.

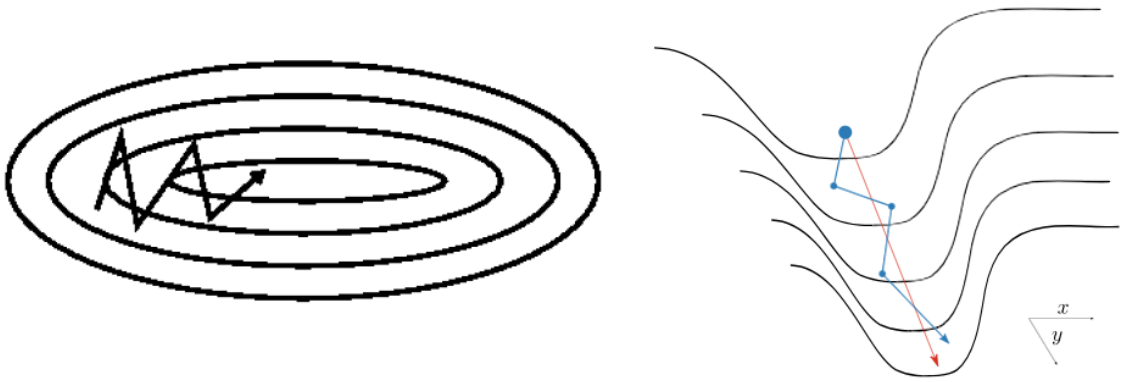



Figure 6.5: The effect of momentum on convergence.

(Source: (Orr 1999))

### 6.5.3 Parameter-Specific Learning Rates

When using momentum methods, we are leveraging consistency in the gradient direction of specific parameters. This consistency speeds up the updates. The same result can be achieved by having different learning rates for each parameter. We will consider four examples discussed by Ruder (2016): AdaGrad, AdaDelta, RMSProp and Adam.

#### 6.5.3.1 AdaGrad

Adagrad (Duchi et al. 2011) is an algorithm that assigns different learning rates to each parameter, e.g. performing smaller updates (smaller $\lambda$) for frequently occurring features and more significant updates (larger $\lambda$) for the parameters of infrequently occurring features. This methodology makes Adagrad perfect for sparse data sets. So, essentially, this means that we control the size of our gradient descent step in each dimension.

UUp to now every parameter $\theta_i$ has used the same $\lambda$ so we have performed an update for all parameters $\boldsymbol{\theta}$ at the same time. The AdaGrad algorithm uses a different $\lambda$ for every $\theta_i$ at every time step $t$, which is denoted as $\lambda_{t,i}$ and is vectorised as $\boldsymbol{\lambda}_t$. We will denote the partial derivative of the cost function w.r.t. $\theta_i$ at step $t$ by

$$g_{t,i} = \nabla_{\boldsymbol{\theta}_t} J(\theta_{t,i}). \tag{6.19}$$

The update rule for $\theta_i$ at step $t$ is given by

$$\theta_{i,t+1} = \theta_{i,t} - \lambda_{t,i} \cdot g_{t,i}. \tag{6.20}$$

[§]The word "friction" is derived from the fact that small values of $\gamma$ act as "brakes", much like friction

In this rule, the general learning rate $\lambda$ is modified at each step $t$ for each $\theta_i$ based on past gradients computed for $\theta_i$

$$\lambda_{t,i} = \frac{\boldsymbol{\lambda}_t}{\sqrt{\mathbf{G}_{t,ii} + \epsilon}}. \tag{6.21}$$

Here $\mathbf{G}_t \in \mathbb{R}^{d \times d}$ is a diagonal matrix where each diagonal element is the squared sum of gradients w.r.t. $\theta_i$ up to step $t$, while $\epsilon$ is a smoothing term that avoids division by zero. So now Eq. (6.20) is written in matrix form as an element-wise multiplication of $\mathbf{G}_t$ and $\mathbf{g}_t$

$$\boldsymbol{\theta_{t+1}} = \boldsymbol{\theta_t} - \frac{\boldsymbol{\lambda}_t}{\sqrt{\mathbf{G}_{t,ii} + \epsilon}} \odot \mathbf{g}_t, \tag{6.22}$$

where $\odot$ is defined as in Eq. (6.11).

The main benefit of the AdaGrad algorithm is that the need to tune the learning rate manually is eliminated. It also has a significant drawback: the cumulative sum of the squared gradients in the denominator will keep growing during training. This is because each term added is positive. This growth in the denominator results in the learning rate becoming infinitesimally small. At this point, the algorithm is no longer able to acquire additional knowledge.

#### 6.5.3.2 AdaDelta

Adadelta (Zeiler 2012) aims to solve AdaGrad's fatal flaw by seeking to reduce its aggressive, monotonically decreasing learning rate. Instead of accumulating all past squared gradients, AdaDelta restricts the window of accumulated past gradients to some fixed size $u$. We calculate the running average of the $u$ previous squared gradients $E\left[\mathbf{g}^2\right]_t$ at time step $t$ as a fraction $\gamma$ of the running average at the previous step and the current gradient[¶]

$$E\left[\mathbf{g}^2\right]_t = \gamma E\left[\mathbf{g}^2\right]_{t-1} + (1-\gamma)\mathbf{g}_t^2. \tag{6.23}$$

We can now rewrite the plain SGD update as:

$$\boldsymbol{\Delta\theta_t} = -\boldsymbol{\lambda_t} \cdot g_{t,i} \tag{6.24}$$

$$\boldsymbol{\theta_{t+1}} = \boldsymbol{\theta_t} + \boldsymbol{\Delta\theta_t}. \tag{6.25}$$

The parameter update vector of AdaGrad that we derived previously thus takes the form

$$\boldsymbol{\Delta\theta_t} = -\frac{\boldsymbol{\lambda}_t}{\sqrt{\mathbf{G}_t + \epsilon}} \odot \mathbf{g}_t. \tag{6.26}$$

Now we replace $\mathbf{G}_t$ with $E\left[\mathbf{g}^2\right]_t$

$$\boldsymbol{\Delta\theta_t} = -\frac{\boldsymbol{\lambda}_t}{\sqrt{E\left[\mathbf{g}^2\right]_t + \epsilon}} \mathbf{g}_t, \tag{6.27}$$

[¶]We set $\gamma$ to a similar value as the momentum term, around 0.9.

and see that the denominator is simply the root mean squared error (RMSE) criterion of the gradient $RMS\left[\mathbf{g}\right]_t$, so:

$$\boldsymbol{\Delta\theta_t} = -\frac{\boldsymbol{\lambda}_t}{\sqrt{RMS\left[\mathbf{g}\right]_t + \epsilon}}\mathbf{g}_t. \tag{6.28}$$

#### 6.5.3.3 RMSprop

Root Mean Square Propagation (RMSprop) has an interesting history. It was devised by Hinton, Srivastava & Swersky (2012) while suggesting a random idea during an online lecture.

RMSprop and AdaDelta were developed independently around the same time. Their development stemmed from the need to resolve the one fatal drawback of AdaGrad. These two algorithms have identical update vectors. Hinton, Srivastava & Swersky (2012) suggests $\gamma = 0.9$ and $\lambda = 0.001$.

#### 6.5.3.4 Adam

Adaptive Moment Estimation (Adam) (Kingma & Ba 2014) is a method of computing adaptive learning rates for each parameter while storing an exponentially decaying average of past squared gradients $\mathbf{v}_t$ (as with AdaDelta and RMSprop). Similar to momentum-based methods, it also keeps an exponentially decaying average of past gradients $\mathbf{m}_t$

$$\mathbf{m}_t = \gamma_1\mathbf{m}_{t-1} + (1-\gamma_1)\mathbf{g}_t, \tag{6.29}$$
$$\mathbf{v}_t = \gamma_2\mathbf{v}_{t-1} + (1-\gamma_2)\mathbf{g}_t^2. \tag{6.30}$$

$\mathbf{m}_t$ and $\mathbf{v}_t$ are estimates of the mean and uncentered variance (first and second moments) of the gradients respectively. $\mathbf{m}_t$ and $\mathbf{v}_t$ are initialized as zero vectors, thus the authors observe that the estimates are biased towards zero at $t = 0$, and more so when $\gamma_1$ and $\gamma_2$ are close to 1. To counteract these biases they compute the bias-corrected estimates:

$$\hat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1-\gamma_1^t}, \tag{6.31}$$
$$\hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1-\gamma_2^t}. \tag{6.32}$$

The update rule can then be written as

$$\boldsymbol{\theta_{t+1}} = \boldsymbol{\theta_t} - \frac{\boldsymbol{\lambda}_t}{\sqrt{\hat{\mathbf{v}}_t} + \epsilon}\hat{\mathbf{m}}_t. \tag{6.33}$$

Kingma & Ba (2014) propose default values of $\gamma_1 = 0.9$ and $\gamma_2 = 0.999$ and $\epsilon = 10^{-8}$. They go on to show empirically that Adam works well in practice and compares favourably to other adaptive learning-method algorithms.

#### 6.5.3.5 Which optimiser is best?

There are many other methods in literature such as Nesterov Accelerated Gradient (NAG) (Nesterov 1983) (a way to give our momentum term in momentum-based methods a rough

idea of where our parameters are going to be, i.e. to know to slow down before the next uphill), AdaMax (Kingma & Ba 2014) (a generalisation of the Adam update rule to the $l_p$ norm) and Nadam (Dozat 2016) (which incorporates NAG into Adam) to name a few. For this dissertation, we will consider only the principal methods described previously.

Ruder (2016) provides the following comparison: RMSprop can be viewed as an extension of Adagrad, and aside from the fact that Adadelta uses the RMS of parameter updates in the numerator update rule (see Eq. (6.22) vs Eq. (6.28)), Adadelta and AdaGrad are also identical. In addition, the Adam algorithm also adds bias correction and momentum to RMSprop. Given all the above, the algorithms we have considered are very similar in most cases. However, Kingma & Ba (2014) shows that bias-correction gives Adam a slight advantage over RMSprop towards the end of optimisation as gradients become sparser. Hence, Adam is considered to be the best overall choice.

## 6.6 Conclusion

Overfitting is by far the biggest hurdle in deep learning. Fortunately, there are a few techniques that we can employ to overcome this problem (and the simpler ones turn out to be the most powerful). This chapter has also provided many different techniques created to improve the performance of our model and fine-tune our most important hyperparameters. This tuning will allow us to create a powerful and efficient classifier.

In the next part of this dissertation, we will build our neural network and deep neural network, and various other models to be used as benchmarks. We will be applying a lot of the concepts and techniques we have discussed and considered in this part, and we will attempt to demonstrate the power of deep learning.

# Part II

# Application

# Chapter 7

# A Neural Network Model for Credit Default Prediction

> "Model building is the art of selecting those aspects of a process that are relevant to the question being asked. As with any art, this selection is guided by taste, elegance, and metaphor; it is a matter of induction, rather than deduction. High science depends on this art." - *John Henry Holland*

## 7.1 Introduction

This chapter will start by looking at our data set and exploring what features are available to us. We then proceed to "fix" the class imbalance issue in our data set and prepare our data set for training. We will also describe the experiment design and look at the five models we will be training and comparing. The ultimate goal is to create a highly accurate deep neural network and then review and compare the results from our experiments. We will then analyse each model's performance in different aspects and rank them accordingly. All code for the models can be found on Github. All models were built-in Python 3.8 using both the Sci-kit learn and TensorFlow (with Keras back-end) libraries.

## 7.2 Experiment Design

We will start by analysing our data set and then prepare it for training. Next, we will build and describe four initial models: a logistic regression model, a support vector machine, a CART model and an artificial neural network.

We will then perform hyper-parameter tuning on each of them using the Grid Search technique to find the best parameters for the most accurate results from each model. Grid search (Pedregosa et al. 2011) is a tuning technique that attempts to compute the optimum values of hyper-parameters. It is an exhaustive search performed on the specific parameter values of a model and is available via the Sci-kit learn library in python.

We will then train our model on a training set and test it on a hold outset to perform some initial evaluations such as looking at the confusion matrix, ROC curve and Kappa score. Then, each model will undergo repeated stratified 10-fold cross-validation to fully evaluate them using metrics such as accuracy, precision, recall, F1 score and ROC-AUC. Each of the model-building processes will follow a similar flow (see Figure 7.1).

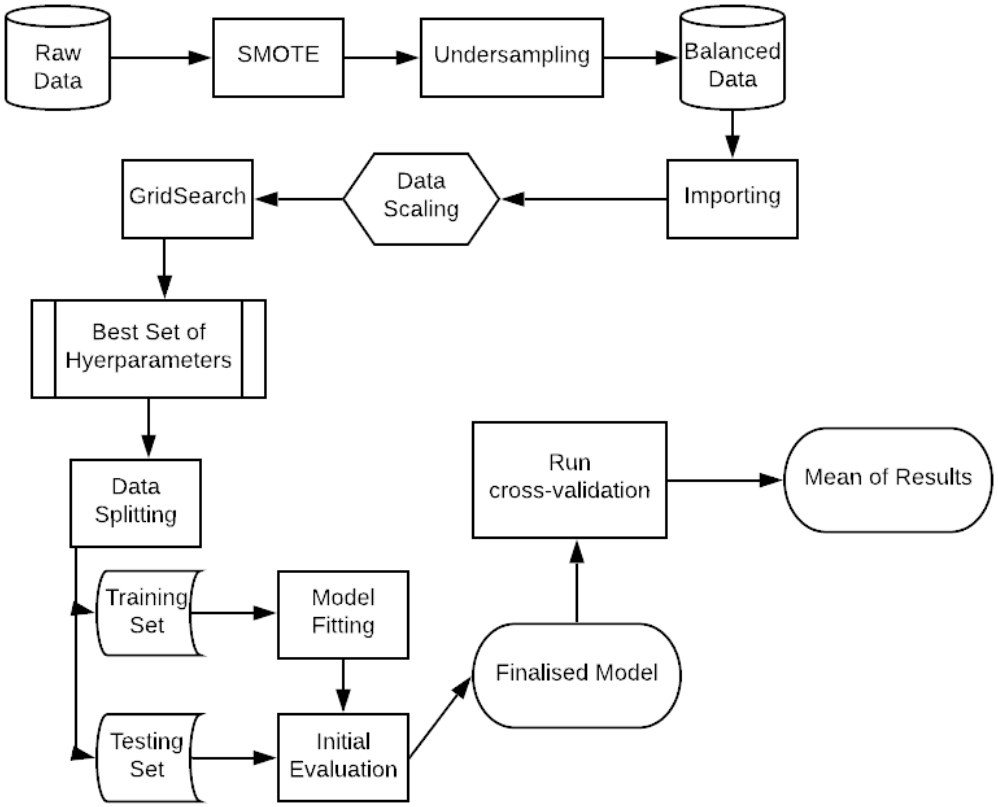



Figure 7.1: Model Building Process Flow

## 7.3 The Data Set

The data set we will use for our analyses was extracted from UCI Machine Learning repository* (Yeh & Lien 2009). It represents Taiwanese credit card defaults in the year 2005 along with three months payment history up to the month of default. We will attempt to determine if these features can allow us to predict the three-month default probability using our various models and compare their performance, and then attempt to show the advantage deep learning theoretically has over the other models.

### 7.3.1 Data Description

The data set comprises credit card customer data from a large retail bank in Taiwan for April to September 2005. It contains one binary dependent variable ($Y$) and 23 independent variables ($X_1$-$X_{23}$). There are 30,000 individual records in the data.

The response variable is binary. It labels whether the customer has defaulted on a payment (defaulted = 1, non-defaulted = 0). The independent or explanatory variables are as follows:

$\mathbf{X_1}$: Credit amount given (NT dollar). This includes individual consumer credit and family (supplementary) credit.

$\mathbf{X_2}$: Gender (1 = male; 2 = female).

$\mathbf{X_3}$: Education (1 = graduate school; 2 = university; 3 = high school; 4 = others).

$\mathbf{X_4}$: Marital status (1 = married; 2 = single; 3 = others).

*This data set is licensed under a CC0 1.0 License

$\mathbf{X_5}$: Age (year).
$\mathbf{X_6}$-$\mathbf{X_{11}}$: Past payment history (0 = pay on time; 1 = payment delayed for one month, . . . , 9 = payment delayed for nine months or more). The records are for April to September and are described using the following variables:

$X_6$ = the repayment status in September, 2005;

$X_7$ = the repayment status in August, 2005;

. . .

$X_{11}$ = the repayment status in April, 2005.

$\mathbf{X_{12}}$-$\mathbf{X_{17}}$: Amount of bill statement (NT dollar).

$X_{12}$ = bill for September, 2005;

$X_{13}$ = bill for August, 2005;

. . .

$X_{17}$ = bill for April, 2005.

$\mathbf{X_{18}}$-$\mathbf{X_{23}}$: Previous payment amount (NT dollar).

$X_{18}$ = amount paid in September, 2005;

$X_{19}$ = amount paid in August, 2005;

. . .

$X_{23}$ = amount paid in April, 2005.

### 7.3.2 Data Exploration

The data set contains 30,000 unique rows of data. The distribution of the response variables can be seen in Figure 7.2 :

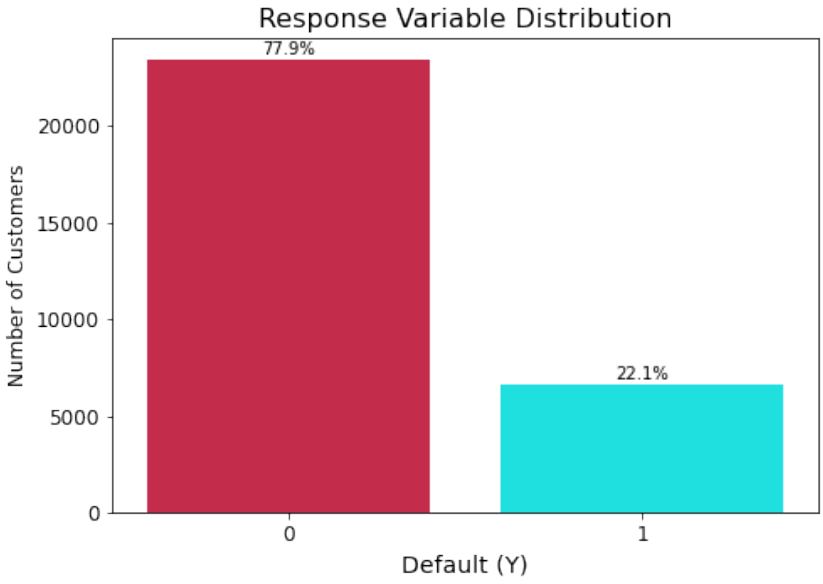



Figure 7.2: Histogram of response variable distribution before resampling.

As we can see, our data set is highly skewed. This skew is called a class imbalance problem. The class imbalance will negatively affect our model's ability to learn the decision boundary effectively and accurately predict both classes as the training algorithms are biased towards $Y = 0$.

### 7.3.3 Data Preparation

Since this a well-studied problem and the data set is very commonly used, the data itself is very clean. It contains no missing values, and all variables are numeric. There are no duplicates and no redundant entries. This leaves only two tasks to perform before we can start our analyses: resampling and scaling.

#### 7.3.3.1 Resampling

The first step in our preparation of the data is to correct the skew of the class variable $Y$. For this we will use a combination of undersampling of the majority class and SMOTE to up-sample the minority class. After applying these techniques, our response variable distribution is shown by Figure 7.3 :

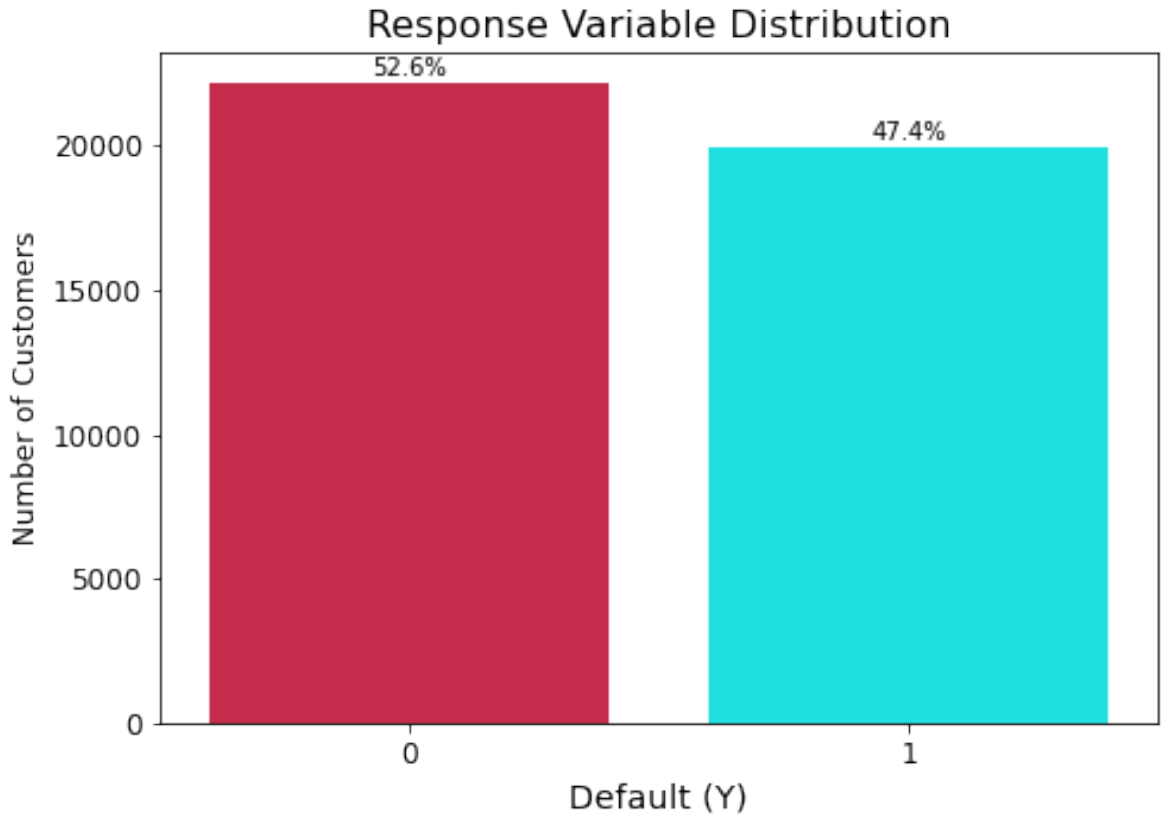



Figure 7.3: Histogram of response variable distribution after resampling.

Our minority class is no longer under-represented, and class imbalance will not affect our models' performance.

#### 7.3.3.2 Scaling

The scaling of a data set is standard in many ML applications. Our models' performance could be unstable if the individual features are not approximately standard normally distributed data, as the objective function of our learning algorithms assumes this. If a feature has a standard deviation much larger than that of the others, it might dominate the objective function. This domination would make the estimator unable to learn from other features correctly. Observing Figure 7.4, we can see what our data looks like unscaled.

| X15 | X16 | X17 | X18 | X19 | X20 | X21 | X22 | X23 |
|---|---|---|---|---|---|---|---|---|
| -1528.000000 | -1528.000000 | -1504.000000 | 8800.000000 | 10183.000000 | 1504.000000 | 0.000000 | 24.000000 | 204395.000000 |
| 3615.000000 | 4402.000000 | 5173.000000 | 2000.000000 | 1500.000000 | 400.000000 | 1000.000000 | 1000.000000 | 500.000000 |
| 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 75299.000000 | 76933.000000 | 78953.000000 | 3000.000000 | 3200.000000 | 3000.000000 | 3400.000000 | 3300.000000 | 3500.000000 |
| -43.000000 | -43.000000 | -43.000000 | 1000.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 24547.362646 | 21848.314290 | 20394.507169 | 60447.550698 | 2837.252597 | 3549.228961 | 20973.596985 | 1599.967692 | 1600.129232 |
| 19153.889448 | 20231.038547 | 20896.175000 | 1434.853057 | 1454.503507 | 1463.935723 | 1402.210506 | 994.682976 | 1228.130323 |
| 63880.676452 | 65218.704393 | 66541.644988 | 2864.931442 | 2336.305001 | 2290.332942 | 2367.250294 | 2407.357360 | 2427.282931 |
| 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 28065.337508 | 28060.215621 | 29197.741998 | 2168.595612 | 2680.331662 | 800.845881 | 2223.472101 | 2394.050785 | 2352.893418 |

Figure 7.4: Sample of raw data unscaled.

We need to scale our data. There are two methods for this: min-max scaling and standard scaling. Through experimentation, we have found that standard scaling works best in our case. So, simply put, we will standardise features by removing the mean and scaling to unit variance. This is done as follows:

$$X_i^{scaled} = \frac{(X_i - \mu_i)}{\sigma_i},$$

where $X_i$ is the feature, $\mu_i$ its mean and $\sigma_i$ its standard deviation. After scaling, our data can be shown as in Figure 7.5 :

| X15 | X16 | X17 | X18 | X19 | X20 | X21 | X22 | X23 |
|---|---|---|---|---|---|---|---|---|
| -0.688032 | -0.680810 | -0.672681 | 0.271712 | 0.261426 | -0.193010 | -0.303486 | -0.305695 | 12.572087 |
| -0.608688 | -0.584092 | -0.561483 | -0.195954 | -0.176437 | -0.263658 | -0.230859 | -0.233150 | -0.252553 |
| -0.664459 | -0.655888 | -0.647634 | -0.333503 | -0.252078 | -0.289255 | -0.303486 | -0.307479 | -0.284002 |
| 0.497224 | 0.598883 | 0.667249 | -0.127180 | -0.090710 | -0.097278 | -0.056553 | -0.062194 | -0.063858 |
| -0.665122 | -0.656589 | -0.648350 | -0.264729 | -0.252078 | -0.289255 | -0.303486 | -0.307479 | -0.284002 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| -0.285752 | -0.299544 | -0.307984 | 3.823748 | -0.109003 | -0.062131 | 1.219772 | -0.188555 | -0.183357 |
| -0.368960 | -0.325921 | -0.299629 | -0.234822 | -0.178731 | -0.195574 | -0.201647 | -0.233545 | -0.206755 |
| 0.321066 | 0.407824 | 0.460551 | -0.136469 | -0.134264 | -0.142691 | -0.131559 | -0.128543 | -0.131330 |
| -0.664459 | -0.655888 | -0.647634 | -0.333503 | -0.252078 | -0.289255 | -0.303486 | -0.307479 | -0.284002 |
| -0.231478 | -0.198228 | -0.161375 | -0.184359 | -0.116916 | -0.238007 | -0.142001 | -0.129532 | -0.136009 |

Figure 7.5: Sample of scaled data.

#### 7.3.3.3 Splitting

When performing our initial evaluation, we will need our data to be split into a training set, a testing set, and for our neural networks, a validation set. First is the train-test split, where we randomly sample 20% of our data into a test set and the other 80% to be

used for training. We will use the testing set as a validation set for our neural network during training to monitor convergence and overfitting.

#### 7.3.3.4 Feature Selection

Usually, in practice, as part of the initial data exploration and model building, we will perform feature selection which is the process of selecting those inputs that have the more explanatory power when it comes to predicting our output. In our case, we look at the correlation plot (see Figure 7.6) and the SHAP values (Roth 1988) produced by our initial logistic regression model to find which features have the most explanatory power. In doing this, we created a different subset of our data set. However, at best, we could only match our accuracy on all models with one of the subsets while the other subsets performed much worse. It is for this reason that we will stick to the 23 features from the original data set.

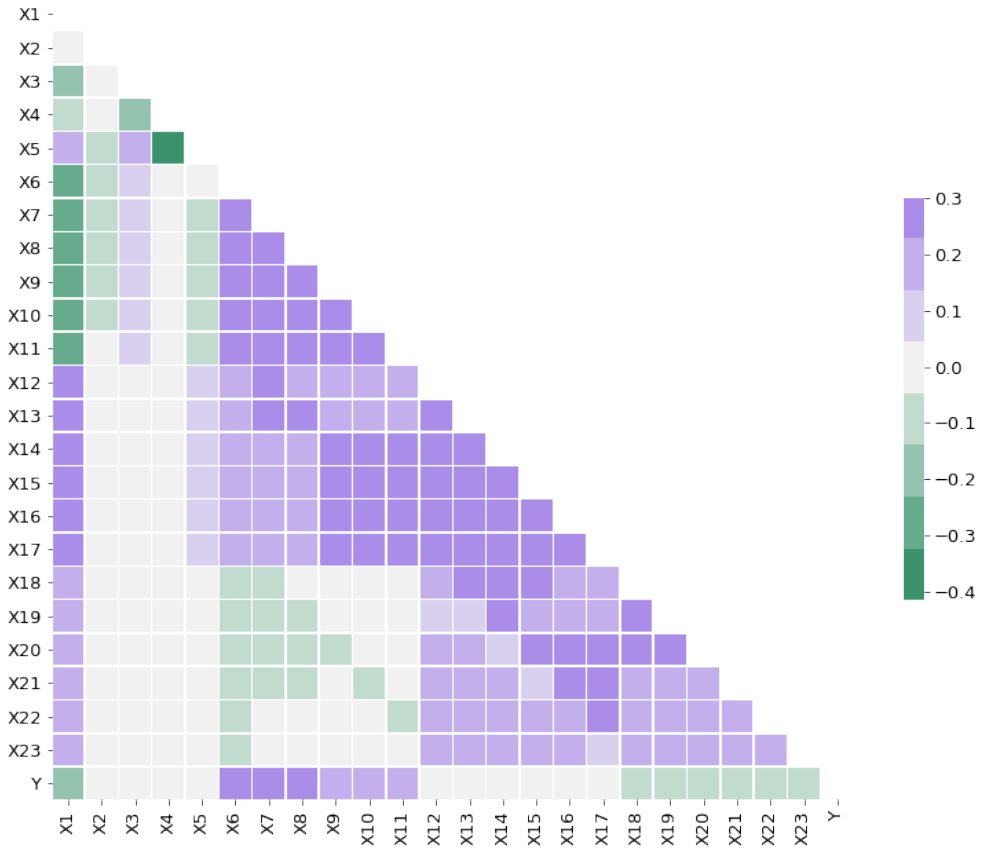



Figure 7.6: Correlation matrix of our features and response variables.

## 7.4 Building All Models

Now we will build, tune, evaluate and compare each of our five models. We will first describe four models: logistic regression, support vector machine, CART classifier and an artificial neural network with one hidden layer. For each model, we perform Grid Search to find the most optimal set of hyper-parameters. Then each model will be fit to a training set, and we will perform some initial evaluation. Next, we run each of our models through stratified 10-fold cross-validation repeated three times. We evaluate the following metrics: accuracy, ROC-AUC, precision, recall and F1 score. We will calculate

the mean value of the scores across all folds and the standard deviation. The final results will be summarised at the end of this chapter in Table 7.5.

### 7.4.1 Logistic Regression

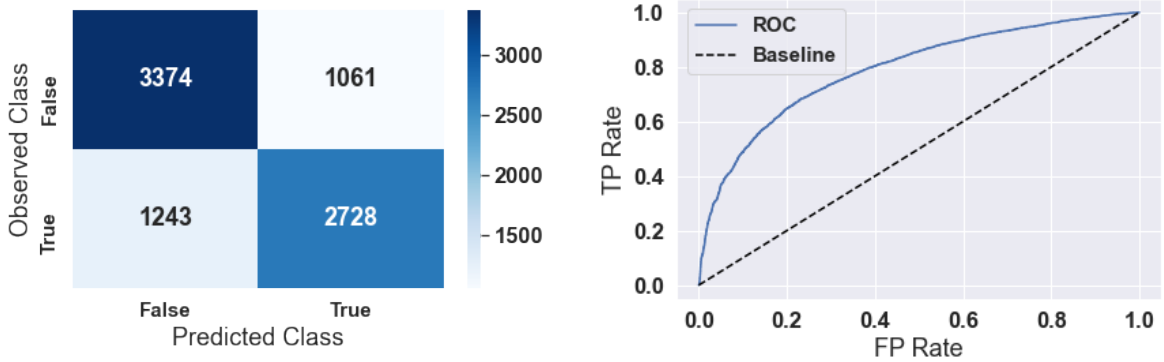
The first of our models is a simple but powerful and widely used logistic regression Model. We build this model using the Scikit-learn ML library. The first step is performing Grid Search. The hyper-parameters are given in Table 7.1 :

| **Hyper-parameter** | **Description** | **Grid Values** |
|---|---|---|
| Penalty | Used to specify the norm used in the penalisation. | $L_1$, $L_2$, Elastic Net |
| C | The inverse of regularisation strength. It must be a positive number. | 0.1, 1, 10, 100 |

Table 7.1: Grid Search parameters for logistic regression.

We will run the Grid Search algorithm using 3-fold cross-validation while scoring the model on accuracy. The algorithm will essentially fit eight ($4 \times 2$) different models fitting three folds for each. This procedure tells us that the best hyper-parameter combination is C = 1 using $L_2$ norm penalisation.

The second step is to fit our model, using the parameters above, to our training set (training time: 0:00:00.22) and then evaluate it using our testing set. We can now look at the confusion matrix (Figure 7.7a), ROC curve (Figure 7.7b) and Kappa score.




(a) Logistic regression model confusion matrix. (b) Logistic regression model ROC curve.

Figure 7.7: Preliminary results for our Logistic Regression model

**Cohen's Kappa: 0.45**

### 7.4.2 Support Vector Machine

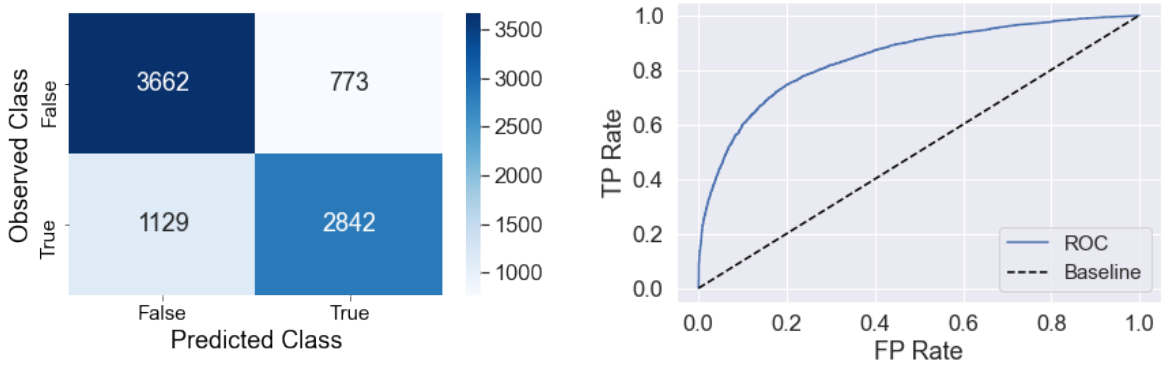
Next, we fit a support vector machine to our data set. This model is a popular alternative to the logit model and has been proposed many times in literature. The hyper-parameters are given in Table 7.2 :

| Hyper-parameter | Description | Grid Values |
|---|---|---|
| Kernel | Specifies the kernel type to be used in the algorithm. | radial basis function, polynomial, sigmoid, linear. |
| C | Regularisation parameter. | 0.1, 1, 10, 100 |
| Gamma | Kernel coefficient. | 1, 0.1, 0.01, 0.001 |

Table 7.2: Grid Search parameters for our support vector machine.

Next, we run the Grid Search algorithm using three-fold cross-validation while scoring the model on accuracy. This time the algorithm will fit 64 ($4^3$) different models fitting three folds for each. This procedure tells us that the best hyper-parameter combination is C = 1 and gamma = 0.1 using the Radial Basis Function as the kernel function.

The second step is to fit our model, using the parameters above, to our training set (training time: 0:00:40) and then evaluate it using our testing set. We can now look at the confusion matrix (Figure 7.8a), ROC curve (Figure 7.8b) and Kappa score.




(a) SVM model confusion matrix.

(b) SVM model ROC curve.

Figure 7.8: Preliminary results for our Support Vector Machine model

**Cohen's Kappa: 0.54**

### 7.4.3 Decision Tree

Here we will construct a CART model. Since CART models are very likely to overfit, we require some extra care in the construction of this model. We will begin by performing grid search on two hyper-parameters which are given in Table 7.3 :

| Hyper-parameter | Description | Grid Values |
|---|---|---|
| Splitting Criterion | The function to measure the quality of a split. | Gini impurity, Entropy (information gain) |
| Min Samples per Leaf (min_leaves) | The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min_leaves training samples in each of the left and right branches. | 1,...,50 |

Table 7.3: Hyper-parameters for our Decision tree.

We will run the Grid Search algorithm using three-fold cross-validation while scoring the model on accuracy. The algorithm will essentially fit 100 ($50 \times 2$) different models fitting three folds for each. The results of this procedure are:

**Criterion**: Gini

**Min samples per leaf**: 49

Next, we fit the model to check its accuracy. The learning algorithm constructs a tree with a depth of 34 and 4918 leave nodes. We find an accuracy of 75.52%. Next, we perform Cost Complexity Pruning, which involves pruning back our full tree to obtain a sub-tree such that we get the lowest test error rate or highest accuracy, i.e. we are trying to reduce the effects of overfitting. To achieve this, we need to find a sub-tree with the lowest $\alpha$ associated with it. Cost complexity pruning is a form of regularisation similar to parameter norm penalties, where $\alpha$ is used to define the cost-complexity measure that is similar to Eq. (7.1):

$$R^{\alpha}(T) = R(T) + \alpha|T|, \tag{7.1}$$

where $|T|$ is the number of terminal nodes in $T$ and $R(T)$ is traditionally defined as the total miss-classification rate of the terminal nodes.

Next, we plot out our cost-complexity pruning path in Figure 7.9, which illustrates the effective alphas of each of the sub-trees during pruning and their corresponding impurities.

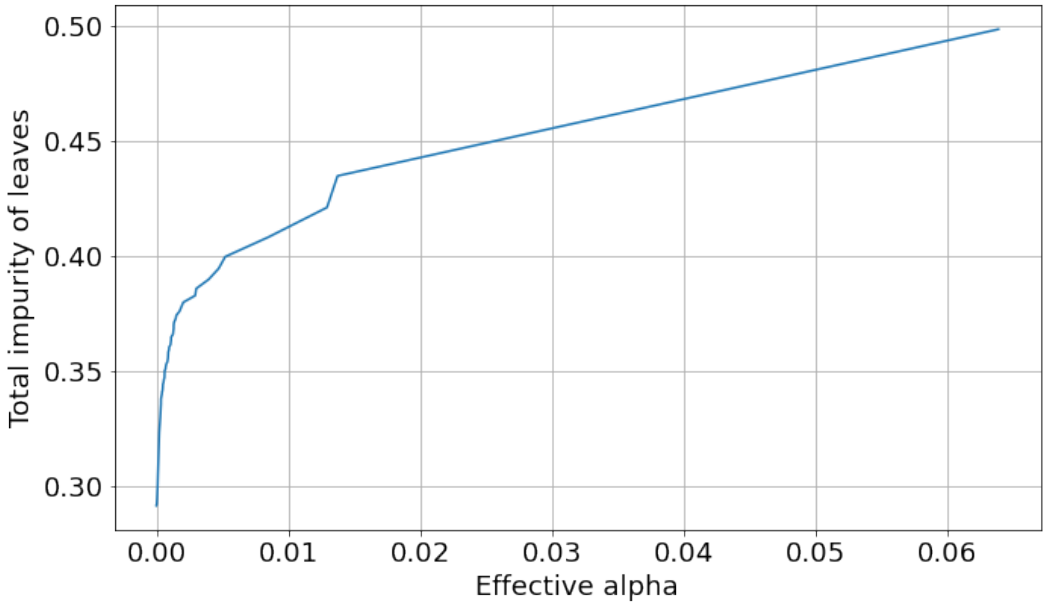



Figure 7.9: Resulting alphas of our cost-complexity pruning path.

Next, for each of these alphas we fit a separate tree to see what the resulting depth is as seen in Figure 7.10:

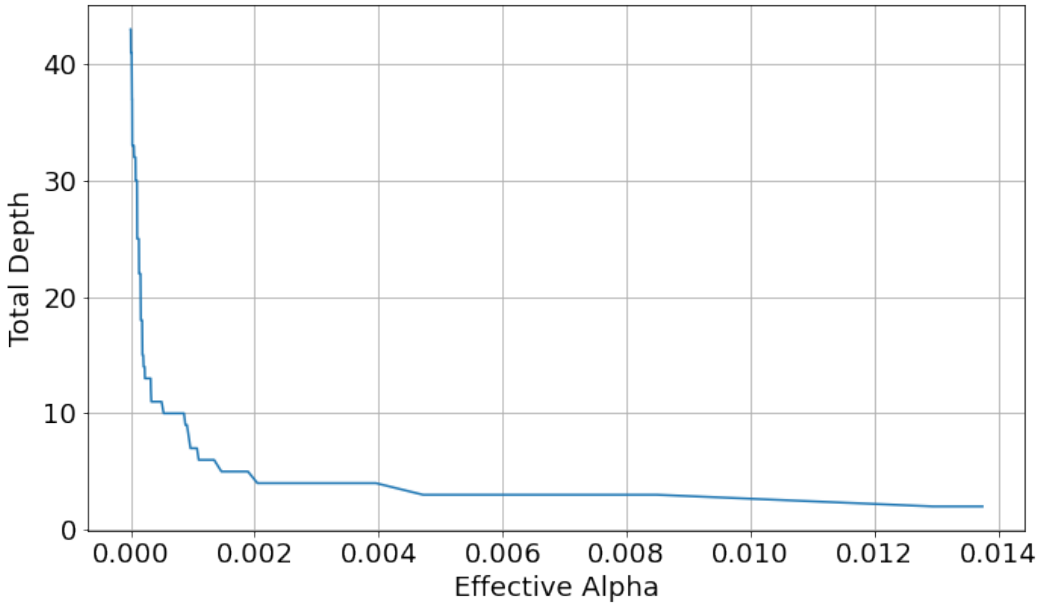



Figure 7.10: Effective alpha vs resulting tree depth.

Finally we measure the accuracy of each sub tree and plot out the results as seen in Figure 7.11:

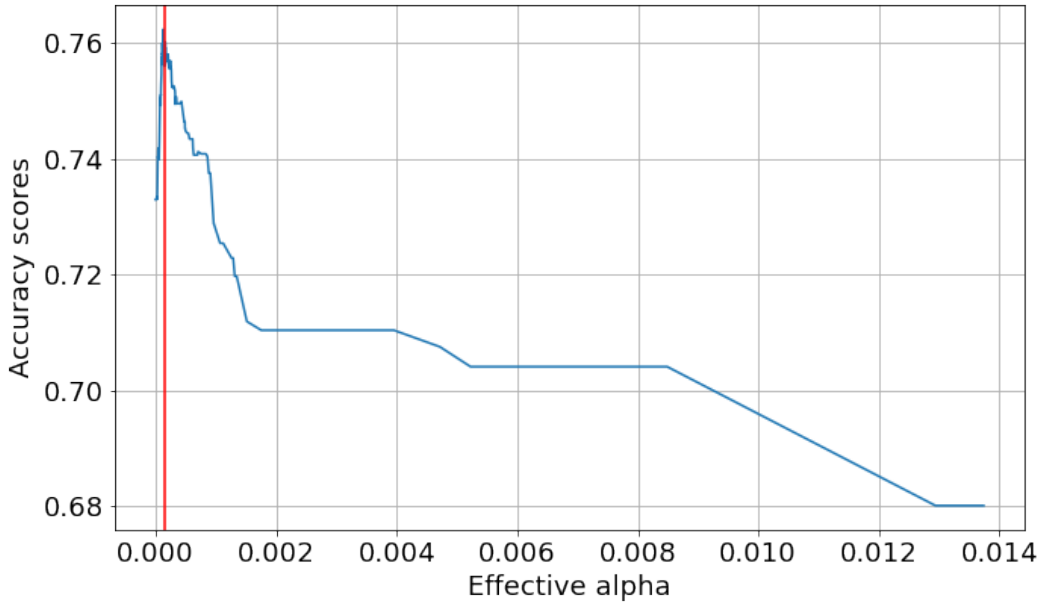



Figure 7.11: Effective alpha vs resulting tree accuracy.

By inspection, we can see on Figure 7.11, that the peak accuracy is at $\alpha \approx 0.00025$. We can also see that some pruning is better than no pruning. So we proceed to fit our final tree using the following parameters:

**Criterion**: Gini

**Alpha**: 0.00025

Using the parameters above, fitting our model to our training set (training time: 0:00:01.11) yields a tree with a depth of 13 and 120 leaf nodes. Figure 7.12 gives us a high level view of our decision tree. This diagram is too large to display here[†], but we aim to illustrate to the reader that the benefit of CART classifiers is that we can visually interpret how our model makes decisions and arrives at a certain prediction.

[†]the software used allows us to zoom into this tree structure to observe the decision rules at each node more clearly

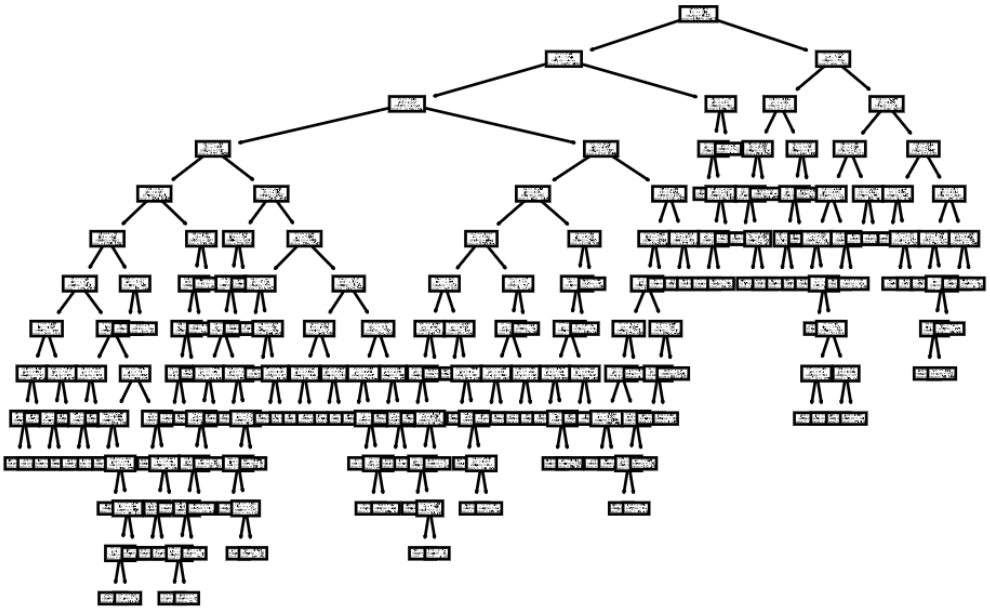
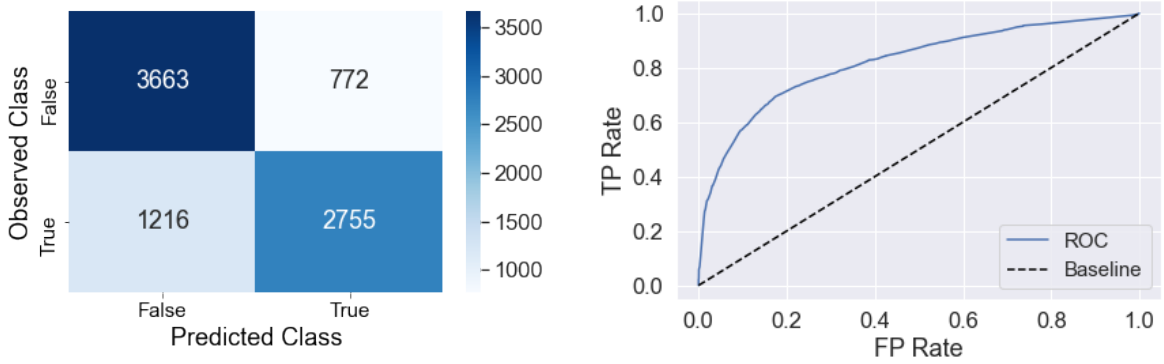


Figure 7.12: Illustration of our pruned CART model. Each box represents a node on the tree related to a specific decision rule. The nodes at the end of each branch are the leaves of the tree, and this is where the model produces its output, i.e. a probability of a data point belonging to a specific class.

We can now look at the confusion matrix (Figure 7.13a), ROC curve (Figure 7.13b) and Kappa score:




(a) CART model confusion matrix.

(b) CART model ROC curve.

Figure 7.13: Preliminary results for our CART model

**Cohen's Kappa: 0.52**

### 7.4.4 Artificial Neural Network

Finally, we construct an artificial neural network. Here the idea is to create as basic of a network as possible and then see if we can improve on it using techniques and methods from Chapter 6, i.e. construct two models.

For the first model, we will aim to keep many of these parameters as basic as possible while tuning only the necessary. For instance, we will use SGD as our optimisation

algorithm, no mini-bathes, no adaptive learning rates, no batch normalisation, no early stopping, no regularisation and no dropout. We will also use Relu and sigmoid as the activation functions on our hidden and output layers respectively. However, we will tune the following parameters given in Table. 7.4 :

| **Hyper-parameter** | **Description** | **Grid Values** |
|---|---|---|
| Epochs | Number of training epochs. | 50, 100, 200 |
| Nodes | Number of nodes on the hidden layer. | 5, 12, 24, 40 |

Table 7.4: Grid Search parameters for our base neural network.

This time the Grid Search algorithm will fit 12 ($4 \times 3$) different models fitting three folds for each. This procedure tells us that the best hyper-parameter combination is Epochs = 200 and Nodes = 12. Our model architecture is shown in Figure 7.14.

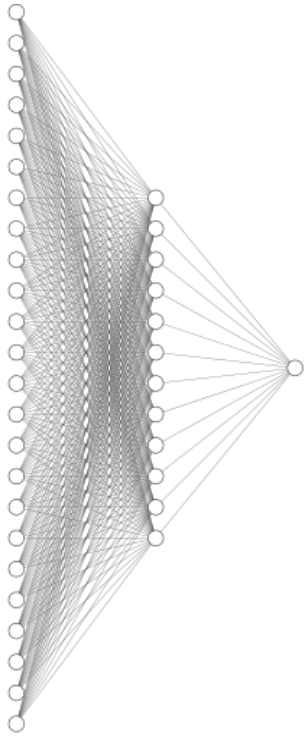


Figure 7.14: Visualisation of our neural network architecture.

For our second model we will use the same architecture but introduce techniques to improve performance. Using Grid Search we find that the following hyper-parameters give optimal results:

Epochs: 200

Mini-batch Size: 500

Kernel Initialisers: Normal

Optimiser: RMSProp

Batch Normalisation: Yes

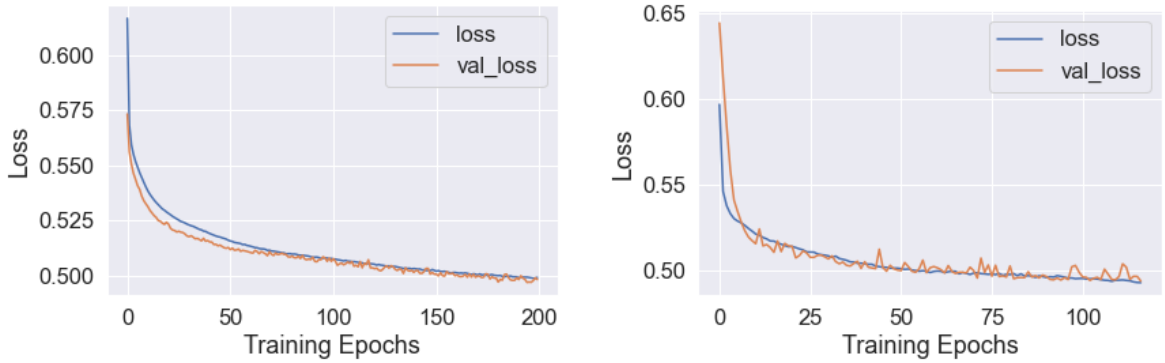
Early Stopping: Yes

While training our models, we will evaluate the error (Figure 7.15) and accuracy (Figure 7.16) of our model on a validation set to monitor overfitting.


(a) Base neural network

(b) Boosted neural network

Figure 7.15: Training accuracy vs validation accuracy of our neural networks during training.

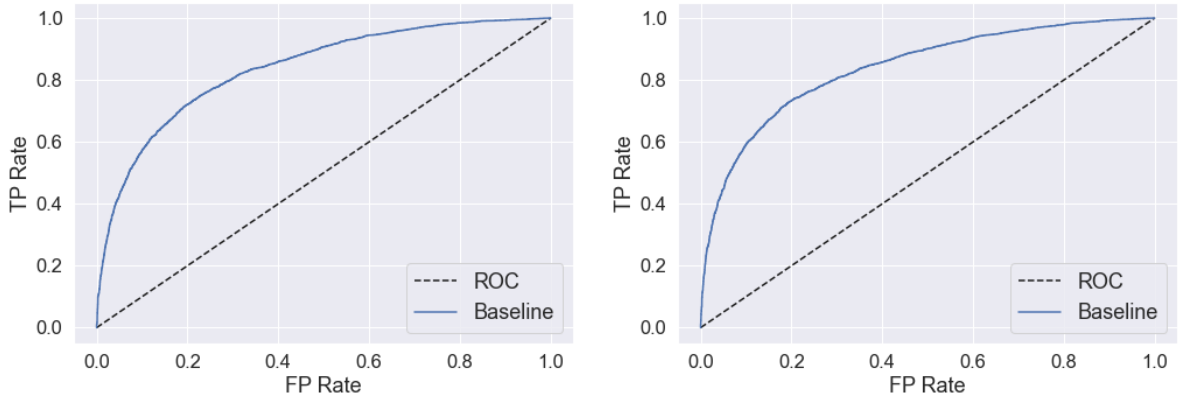



(a) Base neural network

(b) Boosted neural network

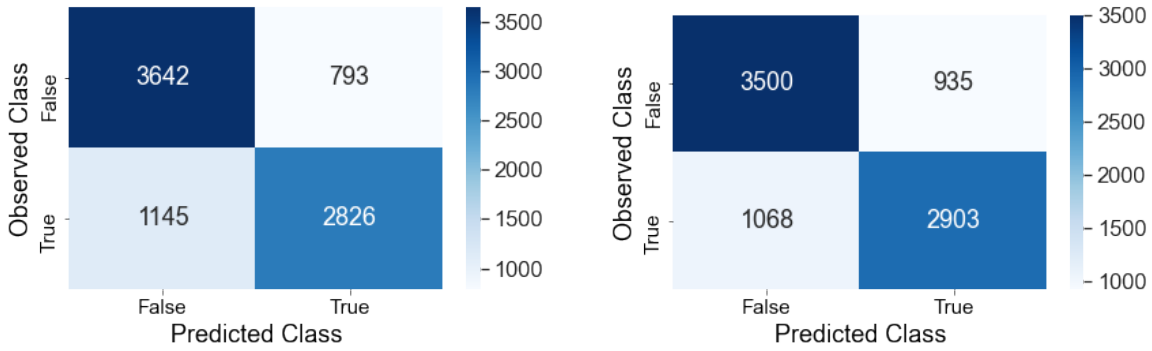
Figure 7.16: Training loss vs validation loss of our neural networks during training.

As we can see, training is relatively stable, there is fast convergence and there appears to be no overfitting. We can also see that our boosted model converges much faster. After fitting both models (training time: 0:02:16 for the base model and 0:00:11 for the boosted model), we do our initial evaluation, by looking at the ROC curve for the base model (Figure 7.17a) and the boosted model (Figure 7.17b), as well as the confusion matrix of the base model (Figure 7.18a) and the boosted model (Figure 7.18), and the results are compared below:




(a) Base neural network (b) Boosted neural network

Figure 7.17: ROC curves of our neural networks.




(a) Base neural network (b) Boosted neural network

Figure 7.18: Confusion matrices of our neural networks.

**Base Neural Network Cohen's Kappa: 0.52**
**Boosted Neural Network Cohen's Kappa: 0.52**

### 7.4.5 Summary of Initial Model Results

We will now summarise the results of our cross-validation process for each of our models in Table. 7.5, to get an idea of what our benchmark is:

| | Accuracy | Precision | Recall | F1-Score | AUC | Kappa |
|---|---|---|---|---|---|---|
| **Logistic Regression** | 72.01% | 71.46% | 68.13% | 69.75% | 78.51% | 0.45 |
| **SVM** | **77.88%** | **79.22%** | **72.25%** | **75.57%** | **84.76%** | **0.54** |
| **CART** | 75.53% | 77.24% | 68.55% | 72.62% | 82.35% | 0.52 |
| **ANN (Base)** | 76.15% | 76.84% | 71.37% | 73.91% | 83.52% | 0.52 |
| **ANN (Modern)** | 76.58% | 77.55% | 71.21% | 74.23% | 83.76% | 0.52 |

Table 7.5: Summary of cross-validation results for initial models

The models perform relatively well so far, with our machine learning models outperforming the traditional logistic regression model by at least 3% in terms of accuracy.

However, there is much room for improvement on each of these models. So far, our results highlight that the SVM model is the superior classifier, but not by a large margin.

There does not appear to be any significant difference between our two networks in terms of performance. However, the clear advantage that our boosted model has over the base model is efficiency. Our boosted model trains 12 times faster than our base network, and from Figure 7.16 we can see it converges much faster. This, combined with the fact that their performance is almost similar, shows that the techniques in Chapter 5 have significantly benefited the field of neural networks.

Both these models also outperformed the CART in all metrics except Kappa. The one clear benefit that the CART model has is that it trains much faster despite being slightly less accurate. The algorithm can easily visualise the tree, giving the user the benefit of explaining the tree's methodology to an extent.

Now we arrive at the central question of this dissertation: can a deep neural network outperform all these models?

### 7.4.6 Deep Neural Network

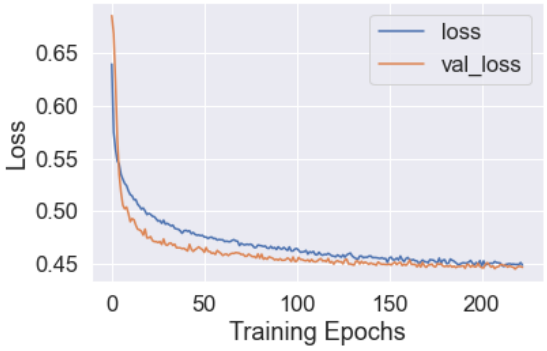
Finally we will build our deep neural network. Our approach follows the recommendation of Goodfellow et al. (2016) which is to over-parameterise our model (make it very wide and deep) and then apply regularisation to limit overfitting. Through trial and error we have used the following NN architecture: five hidden layers with 50, 100, 50, 100 and 50 nodes on each layer respectively. Using multiple runs of the Grid Search algorithm, we find that the following hyper-parameters, which are given in Table 7.6, give optimal results:

| **Hyper-parameter** | **Choice** |
|---|---|
| Epochs | 150 |
| Mini-batch Size | 500 |
| Weight initialisation | Normal |
| Optimiser | Adam |
| Batch Normalisation | Yes |
| Early Stopping | Yes |
| Dropout | Yes with probability of 0.2 on each layer |

Table 7.6: Grid Search results for our deep neural network.

We then proceed to fit our model to the training data while monitoring the error of our model on a validation set to monitor overfitting. Initially, we trained the model without regularisation, and we can see overfitting occurring (see Figure 7.19a). Hence we applied regularisation (in the form of dropout, batch normalisation and early stopping), and we can see that it did remedy the issue (see Figure 7.19b).

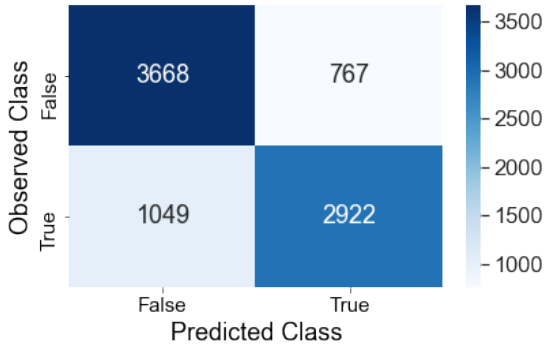

(a) Validation error without regularisation.



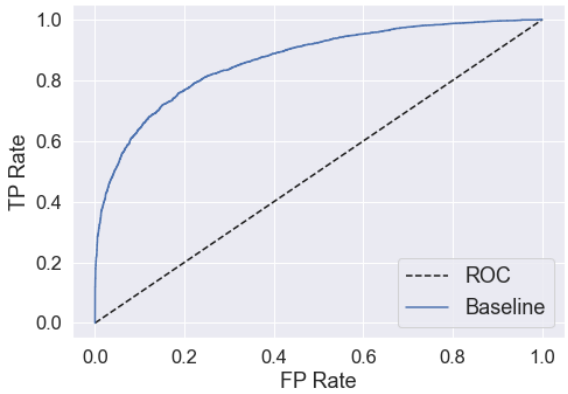

(b) Validation error with regularisation.

Figure 7.19: Result of applying regularisation to prevent overfitting on our Deep Network

After fitting our deep network model (Training time: 0:01:14), we can now look at the confusion matrix (Figure 7.20a), ROC curve (Figure 7.20b) and Kappa score:




(a) Deep neural network model's confusion matrix.




(b) Deep neural network model's ROC curve.

Figure 7.20: Result of applying regularisation to prevent overfitting on our Deep Network

**Cohen's Kappa: 0.56**

## 7.5 Conclusion

We can see that the DNN model performs very well and is a potentially powerful tool. We will examine the cross-validation results of each model in the next chapter. We will critically evaluate the drawbacks of each model and highlight a potential improvement to be made to the experiments. We will then conclude on whether deep learning can rival traditional and non-traditional models and discuss whether it has a place in finance by considering multiple arguments.

# Chapter 8

# Conclusion and Suggestions for Future Study

> "The point is now indisputable: when you have a well-validated statistical algorithm, use it" - *Phillip E. Tetlock, Super Forecasting*

## 8.1 Introduction

In this final chapter, we conclude whether Deep Learning is an excellent alternative to conventional and other credit scoring models. We also highlight some of the drawbacks and challenges that Deep Learning approaches face in the financial sector (especially in banking), where regulatory hurdles are plenty and consider some resolutions proposed.

## 8.2 Results and Evaluation

After performing stratified 10-Fold cross-validation repeated three times on each of our models and averaging the results, we arrive at the results in Table 8.1.

| | **Accuracy** | **Precision** | **Recall** | **F1-Score** | **AUC** | **Kappa** |
|---|---|---|---|---|---|---|
| **Logistic Regression** | 72.01% | 71.46% | 68.13% | 69.75% | 78.51% | 0.45 |
| **SVM** | 77.88% | 79.22% | **72.25%** | 75.57% | 84.76% | 0.54 |
| **CART** | 75.53% | 77.24% | 68.55% | 72.62% | 82.35% | 0.52 |
| **ANN (Base)** | 76.15% | 76.84% | 71.37% | 73.91% | 83.52% | 0.52 |
| **ANN (Modern)** | 76.58% | 77.55% | 71.21% | 74.23% | 83.76% | 0.52 |
| **DNN** | **78.06%** | **79.9%** | 71.8% | **75.6%** | **86.07%** | **0.56** |

Table 8.1: Final results of cross-validation for all models.

To reiterate, all models perform relatively well, with our machine learning models outperforming the traditional logistic regression model. In our results, it is necessary to highlight that the SVM model was superior to the other traditional machine learning classifiers. The decision tree performed almost as well and had the added benefit of speed and interpretability.

Turning back to our initial neural networks, it is worth noting that even though they performed almost the same, the boosted model trained much faster, which is a clear

indication that the methods we considered in Chapter 6 are adequate. Not only that, but our neural networks performed very well in all aspects.

However, the main result to highlight is the performance of the deep neural network. First to note is that it performed better than our standard neural networks, which is what we expected. Second, our model outperformed all the other classifiers, except the SVM in terms of recall.

## 8.3 Conclusion

It is important to note that our deep neural network outperformed the SVM by a very slim margin; one could say it is statistically insignificant. However, this is still a robust result, considering that deep neural networks only show their true power the more the data set increases (Brownlee 2019).

Retail banks would have massive sets and more features. Firstly, we know that deep neural networks performance increases the more data is used, while traditional models have a limit, as seen in Figure 8.1. There may be instances where more training examples will not improve performance, e.g. high bias models will do not benefit from more data. However, they do benefit from more features (Amatriain 2015). Second, more features would make it harder for the traditional model to learn the relationship between the features and response variable correctly. Theoretically, deep neural networks would perform well in these situations.

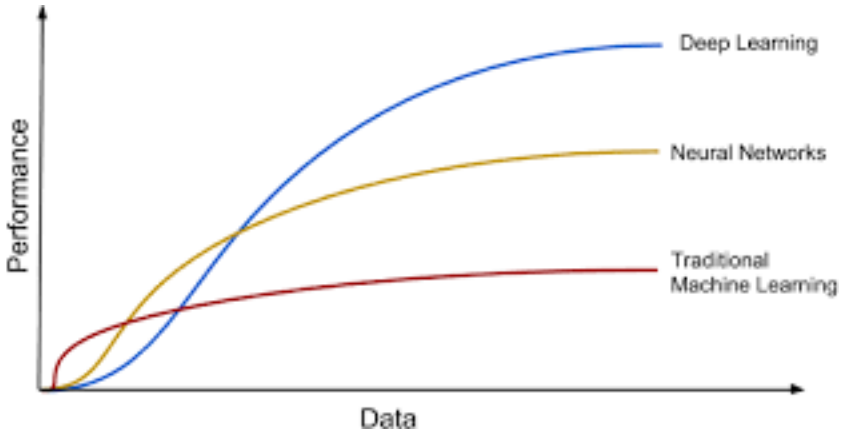



Figure 8.1: Comparison of the performance of DNNs, ANNs and Traditional ML over the amount of data used.

(Source: (Wasicek 2018))

In an interview with Wired Magazine (Garling 2018), Dr Andrew Ng, chief scientist at Baidu and one of the leaders of the Google Brain Project, shared the following analogy for deep learning:

> "I think AI is akin to building a rocket ship. You need a huge engine and a lot of fuel. If you have a large engine and a tiny amount of fuel, you won't make it to orbit. If you have a tiny engine and a ton of fuel, you can't even lift off. To build a rocket, you need a huge engine and a lot of fuel."

In summary, all of the above shows that there is potential for deep learning and machine learning, in general, to replace traditional models in the future. However, as we will note, later on, there are still some hurdles for machine learning in finance that needs to be addressed.

## 8.4 Final Thoughts: Deep Learning in Finance

The field of AI/ML has gained new momentum, and while the idea of machines with human levels of intelligence is not a new concept, the success in creating AI machines has been scarce over the decades since its proposal and seemed, for the most part, to be a failed concept.

Today, it is undeniable that DL is powerful, and our work barely scratches the surface of its capabilities. DL has come to dominate some industries in recent years, and now, after many years of resistance from regulators, AI/ML is moving from the research desk to the application stack. The finance industry is one of the most influential industries impacted by new findings in AI. However, ML models are often unfit to be successfully incorporated into the ongoing risk monitoring of financial institutions. ML models can be overly complex and sensitive to overfitting data. Their level of complexity makes it challenging to apply jurisdictionally consistent definitions of data, and the models are too complex for regulatory purposes as it difficult for auditors to interpret them. They are primarily considered a "black box" for regulators and a non-technical audience (Bazarbash 2019).

We will briefly consider some of the arguments for and against DL in finance. We will also consider some hurdles that may need to be navigated for DL to be fully embraced by financial analysts and regulators alike.

### 8.4.1 Arguments For Deep Learning in Finance

As technology evolves, more open-sourced algorithms become available, and the volume of literature in the field increases, it becomes difficult to imagine a future in financial services without ML and specifically DL. Some of the current DL applications in finance already include algorithmic trading, fraud detection, portfolio management ("Robo-advisor"), loan/insurance underwriting and many more.

As we have seen up to now, there is certainly the potential for DL to help improve credit decisions and decrease credit risk given the right amount of expertise and enough data. Beyond this, there are many other potential benefits. Despite the challenges, many companies are already taking advantage of this technology. De Jager (2018) proposes other benefits of using DL such as:

- yielding insights humans do not see in large sets of data;
- higher accuracy of predictions over and above our current models;
- removing human biases from decisions;
- increasing productivity and less time spent on laborious tasks.

Also, thanks to the quantitative nature of the financial domain and massive stores data, DL, as we now know, has the potential to enhance many aspects of the financial ecosystem.

Another argument for DL and AI is that it allows us to rethink and reconstruct the traditional credit allocation process to be more inclusive and fair. AI can avoid traditional credit scoring mechanisms that perpetuate existing bias, making it a unique opportunity to change the status quo (De Jager 2018). This opportunity to enhance financial inclusion is created by leveraging non-traditional data sources that may create a clearer picture of a borrowers potential risk and perhaps enhance the assessment of their track record. There are, of course, situations such as profound structural changes occurring, borrowers counterfeiting specific indicators, and agency problems arising from

information asymmetry, highlighting the importance of data relevance. Because data plays an essential role in ML-based models, it should always be ensured that data remains valid and relevant. To ensure digital financial inclusion and avoid redlining, we should also avoid the use of variables that trigger discrimination when assessing credit ratings (Bazarbash 2019).

### 8.4.2 Arguments Against Deep Learning in Finance

AI can have equally unfavourable effects, such as exacerbating existing bias, creating cycles that reinforce biased credit allocation while making discrimination in lending even harder to find (Klein 2020). This seems to be the main argument against ML/DL in finance and credit risk management. This is echoed in the book by O'Neil (2017), titled "Weapons of Math Destruction". O'Neil (2017) 's main argument is that algorithms (such as the ones that underpin DL) can and do perpetuate inequality. While the O'Neil (2017) does note the numerous benefits AI can have, they also claim that the algorithms have "harmful outcomes and often reinforce inequality, keeping the poor poor and the rich rich". The author goes on to highlight that:

> "[machine learning algorithms] are often proprietary or otherwise shielded from prying eyes, so they have the effect of being a black box. They affect large numbers of people, increasing the chances that they get it wrong for some of them. And they have a negative effect on people, perhaps by encoding racism or other biases into an algorithm or enabling predatory companies to advertise selectively to vulnerable people, or even by causing a global financial crisis".

The black box argument is a sentiment shared by regulators. If a DL algorithm arrives at a specific forecast/score/classification, it would be impossible for the ML/Data Science engineer to explain how it got to that answer. This is more prevalent in DL than ML in general, given the highly complex nature of DL architectures. While much progress has been made to increase the interpretability of ML models (such as SVMs and decision trees), the progress in DL seems to be lacking. This is possibly due to the complexity involved. However, at the rate that AI technology and computing power are evolving and with the amount of attention and funding going towards research, we may yet see an era where DL becomes standard in many risk management frameworks.

Another critical argument one can present against deep learning in finance is that, because many financial disasters have a low probability of occurring and thus might not be included in the historical data (or might be included highly infrequently), deep learning models might not be able to correctly learn how to predict these tail events given certain factors. This is a problem because it means that while deep learning models might be great at reproducing the belly of a loss distribution, they will almost certainly be unable to predict tail events. That is a significant shortcoming that needs to be considered when using these models in risk management. Tail losses, while infrequent, may be devastating to institutions and would mean that we are understating our losses. This, in turn, means we are not holding the correct amount of capital to absorb such losses. One proposed strategy to combat this is to combine deep learning models (or machine learning in general) with tail-risk protection strategies. Spilak & Härdle (2020) follows this hybrid approach and produces an ensemble classifier that produces a meta tail risk protection strategy improving both generalisation and trading performance.

### 8.4.3 Challenges for Deep Learning in Finance

Aside from the regulatory challenges discussed above, another straightforward challenge stands in how DL is fully adopted in finance: DL is complicated. This complexity makes the adoption of DL technologies difficult for two reasons:

1. it is not easy to convince team members and especially upper management to consider an AI/DL solutions;

2. there is a general lack of education/information on the matter informal education.

To effectively adopt AI/DL in an organisation, there must exist a function that can educate (Faggella 2021). This may seem like a quick fix, but in reality, teaching most people about the complexities of a deep learning architecture proves to be difficult as it requires backgrounds in mathematics, statistics and data science.

Finally, aside from the many benefits that DL holds, most financial services companies are still not ready to extract the real value from this technology for the following reasons:

- businesses often have completely unrealistic expectations towards machine learning and its value for their organisations.;
- research and development in ML is costly;
- the shortage of Data Science/ML engineers;
- financial incumbents are not agile enough when it comes to updating data infrastructure.

These challenges may plague the financial industry for the decade to come. However, the point is clear: early and effective adoption of these technologies will not only be an effective tool to help manage risk effectively, but it will also find or create value where we would have never thought to look.

## 8.5 Future Research

Some exciting research avenues are well worth exploring. First, we believe, given some more recent data sets (which we would ideally source from a large South African bank), we could increase the accuracy of our deep learning model. Second, deep learning extends beyond multi-layered perceptrons. We want to explore the use of recurrent neural networks and convolutional neural networks to attempt to perform credit default prediction.

Next, there also appears to be some exciting and highly relevant research being done using machine learning to reduce the costs of credit risk management in the peer-to-peer lending space. Since this industry is still relatively young, it is mostly unregulated, meaning it is a great space to deploy these ML models potentially. Another benefit is that the companies in this industry are still reasonably young (primarily startups). They have no legacy system issues meaning there is an appetite for innovation. Now, before this can be realised, more research is needed to find which models present the optimal balance of accuracy, efficiency, scalability and interpretability. This could be a potential avenue to explore in the future, and seeing as the data might be different from that used by banks, the same models might not perform equally as well as in this dissertation. Potential models for this scenario could include a family of robust ensemble-based CART default classifiers called gradient boosted trees algorithms. These include popular algorithms such as XGBoost and LightGBM. These models have proven to be excellent classifiers and tend to cope well with large sets of data.

Two other areas of interest in the space of default classification are corporate default prediction (using publicly available data) and small-business failure prediction. Now both these differ pretty significantly from the areas mentioned thus far. It would be worth looking into current research to see what has been done and see where we could improve. This improvement could be achieved by either building a new model or taking a model that has already proven effective and potentially expanding it further by utilising newer techniques in machine learning that are constantly being developed.

Finally, it would be fascinating to see if we could take any of our successful models from theory to practice by building a proof-of-concept application that could perform default prediction at loan origination and throughout the life of the loan.

# Bibliography


Abdou, H. A. & Pointon, J. (2011), 'Credit scoring, statistical techniques and evaluation criteria: a review of the literature', *Intelligent systems in accounting, finance and management* **18**(2-3), 59–88.

Agarwal, A., Hazan, E., Kale, S. & Schapire, R. E. (2006), Algorithms for portfolio management based on the newton method, *in* 'Proceedings of the 23rd international conference on Machine learning', pp. 9–16.

Aggarwal, C. C. (2018), *Neural networks and deep learning*, Springer.

Akin, J. (2020), 'What are the different credit score ranges?', [online] Experian.com <https://www.experian.com/blogs/ask-experian/infographic-what-are-the-different-scoring-ranges/>. Accessed: 2021-05-23.

Alabi, M., Issa, S. & Afolayan, R. (2013), 'An application of artificial intelligent neural network and discriminant analyses on credit scoring', *Math Theory Model* **3**(11), 20–28.

Aleksandrova, M. (2017), Matrix factorization and contrast analysis techniques for recommendation, PhD thesis, Université de Lorraine.

Allen-Zhu, Z., Li, Y. & Song, Z. (2019), A convergence theory for deep learning via over-parameterization, *in* 'International Conference on Machine Learning', pp. 242–252.

Amatriain, X. (2015), 'In machine learning, what is better: More data or better algorithms', [online] KDnuggets <https://www.kdnuggets.com/2015/06/machine-learning-more-data-better-algorithms>. Accessed: 2021-03-30.

Andersen, A. C. & Mikelsen, S. (2012), 'A novel algorithmic trading framework applying evolution and machine learning for portfolio optimization', *Diss. Master's Thesis, Faculty of Social Science and Technology Management, Department of Industrial Economics and Technology Management* .

Anderson, R. (2007), *The credit scoring toolkit. Theory and Practice for Retail Credit Risk Management and Decision Automation*, Oxford University Press.

Angelini, E., di Tollo, G. & Roli, A. (2008), 'A neural network approach for credit risk evaluation', *The quarterly review of economics and finance* **48**(4), 733–755.

Antonov, A., Konikov, M. & Piterbarg, V. (2020), 'Neural networks with asymptotics control', *Available at SSRN 3544698* .

Atiya, A. F. (2001), 'Bankruptcy prediction for credit risk using neural networks: A survey and new results', *IEEE Transactions on neural networks* **12**(4), 929–935.

Awoyemi, J. O., Adetunmbi, A. O. & Oluwadare, S. A. (2017), Credit card fraud detection using machine learning techniques: A comparative analysis, *in* '2017 International Conference on Computing Networking and Informatics (ICCNI)', IEEE, pp. 1–9.

Baesens, B., Van Gestel, T., Viaene, S., Stepanova, M., Suykens, J. & Vanthienen, J. (2003), 'Benchmarking state-of-the-art classification algorithms for credit scoring', *Journal of the operational research society* **54**(6), 627–635.

Bansal, G., Sinha, A. P. & Zhao, H. (2008), 'Tuning data mining methods for cost-sensitive regression: a study in loan charge-off forecasting', *Journal of Management Information Systems* **25**(3), 315–336.

Barandela, R., Valdovinos, R. M. & Sánchez, J. S. (2003), 'New applications of ensembles of classifiers', *Pattern Analysis & Applications* **6**(3), 245–256.

Bastos, J. (2007), Credit scoring with boosted decision trees, *in* 'MRPA Papers'.

Bazarbash, M. (2019), *Fintech in financial inclusion: machine learning applications in assessing credit risk*, IMF Working Paper.

Bellotti, T. & Crook, J. (2009), 'Support vector machines for credit scoring and discovery of significant features', *Expert systems with applications* **36**(2), 3302–3308.

Belson, W. A. (1959), 'Matching and prediction on the principle of biological classification', *Journal of the Royal Statistical Society: Series C (Applied Statistics)* **8**(2), 65–75.

Bhatt, S. (2018), '5 things you need to know about reinforcement learning', [online] KDnuggets <https://www.kdnuggets.com/2018/03/5-things-reinforcement-learning.html>. Accessed: 2021-05-30.

Bischl, B., Kühn, T. & Szepannek, G. (2016), On class imbalance correction for classification algorithms in credit scoring, *in* 'Operations Research Proceedings 2014', Springer, pp. 37–43.

Bolton, C. et al. (2010), Logistic regression and its application in credit scoring, PhD thesis, University of Pretoria.

Boser, B. E., Guyon, I. M. & Vapnik, V. N. (1992), A training algorithm for optimal margin classifiers, *in* 'Proceedings of the fifth annual workshop on Computational learning theory', Vol. 9, pp. 144–152.

Bottou, L. (1998), 'Online learning and stochastic approximations', *On-line learning in neural networks* **17**(9), 9–42.

Bradley, A. P. (1997), 'The use of the area under the roc curve in the evaluation of machine learning algorithms', *Pattern recognition* **30**(7), 1145–1159.

Breiman, L. (2001), 'Random forests', *Machine learning* **45**(1), 5–32.

Breiman, L., Friedman, J., Stone, C. J. & Olshen, R. A. (1984), *Classification and regression trees*, CRC press.

Brown, G. (2010), *Ensemble Learning*, Springer US, Boston, MA.

Brown, I. & Mues, C. (2012), 'An experimental comparison of classification algorithms for imbalanced credit scoring data sets', *Expert Systems with Applications* **39**(3), 3446–3453.

Brownlee, J. (2018), 'A gentle introduction to dropout for regularizing deep neural networks', [Online] Machinelearningmastery.com <https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/>. Accessed: 2021-05-30.

Brownlee, J. (2019), 'How to improve deep learning performance', [online] Machine Learning Mastery <https://machinelearningmastery.com/improve-deep-learning-performance/>. Accessed: 2021-03-30.

Brownlee, J. (2020*a*), 'A gentle introduction to k-fold cross-validation', [online] Machine Learning Mastery <https://machinelearningmastery.com/k-fold-cross-validation/#:~:text=Cross%2Dvalidation%20is%20a%20resampling,k%2Dfold%20cross%2Dvalidation.>. Accessed: 2021-03-30.

Brownlee, J. (2020*b*), 'Use early stopping to halt the training of neural networks at the right time', [online] Machine Learning Mastery <https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time-using-early-stopping/>. Accessed: 2021-05-30.

Buühlmann, P. & Geer, S. v. d. (2012), *Statistics for high-dimensional data*, 1 edn, Springer.

Büyüköztürk, Ş. & Çokluk-Bökeoğlu, Ö. (2008), 'Discriminant function analysis: Concept and application.', *Eurasian Journal of Educational Research (EJER)* **8**(33), 73–92.

Chang, Y.-C., Chang, K.-H., Chu, H.-H. & Tong, L.-I. (2016), 'Establishing decision tree-based short-term default credit risk assessment models', *Communications in Statistics-Theory and Methods* **45**(23), 6803–6815.

Chang, Y.-C., Chang, K.-H. & Wu, G.-J. (2018), 'Application of extreme gradient boosting trees in the construction of credit risk assessment models for financial institutions', *Applied Soft Computing* **73**, 914–920.

Charitou, A., Neophytou, E. & Charalambous, C. (2004), 'Predicting corporate failure: empirical evidence for the uk', *European accounting review* **13**(3), 465–497.

Chatterjee, S. (2015), *Modelling credit risk*, number 34 *in* 'Handbooks', Centre for Central Banking Studies, Bank of England.

Chawla, N. V., Bowyer, K. W., Hall, L. O. & Kegelmeyer, W. P. (2002), 'Smote: synthetic minority over-sampling technique', *Journal of artificial intelligence research* **16**, 321–357.

Chen, T., He, T., Benesty, M., Khotilovich, V., Tang, Y., Cho, H. et al. (2015), 'Xgboost: extreme gradient boosting', *R package version 0.4-2* **1**(4).

Chowdary, D. H. (2020), 'Decision trees explained with a practical example', [online] Medium <https://pub.towardsai.net/decision-trees-explained-with-a-practical-example-fe47872d3b53>. Accessed: 2021-05-30.

Cohen, J. (1960), 'A coefficient of agreement for nominal scales', *Educational and psychological measurement* **20**(1), 37–46.

Colianni, S., Rosales, S. & Signorotti, M. (2015), 'Algorithmic trading of cryptocurrency based on twitter sentiment analysis', [online] Standford University <http://cs229.stanford.edu/proj2015/029_report.pdf>. Accessed: 2021-06-06.

Cortes, C. & Vapnik, V. (1995), 'Support-vector networks', *Machine learning* **20**(3), 273–297.

Crouhy, M., Galai, D. & Mark, R. (2006), *The Essentials of Risk Management*, McGraw-Hill.

Culkin, R. & Das, S. R. (2017), 'Machine learning in finance: the case of deep learning for option pricing', *Journal of Investment Management* **15**(4), 92–100.

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

Darken, C., Chang, J., Moody, J. et al. (1992), Learning rate schedules for faster stochastic gradient search, *in* 'Neural networks for signal processing', Vol. 2, IEEE, pp. 3–12.

Dasarathy, B. V. & Sheela, B. V. (1979), 'A composite classifier system design: Concepts and methodology', *Proceedings of the IEEE* **67**(5), 708–713.

De Jager, P. (2018), 'How machine learning can improve your debt collection process', [online] Insights.principa.co.za <https://insights.principa.co.za/machine-learning-improve-debt-collection-process#:~:text=Credit%20application%20scoring,accuracy%20of%20the%20predictions%20further.>. Accessed: 2021-03-30.

De Laurentis, G., Maino, R. & Molteni, L. (2010), *Developing, Validating and using internal ratings*, Wiley Online Library.

De Spiegeleer, J., Madan, D. B., Reyners, S. & Schoutens, W. (2018), 'Machine learning for quantitative finance: fast derivative pricing, hedging and fitting', *Quantitative Finance* **18**(10), 1635–1643.

Dozat, T. (2016), 'Incorporating nesterov momentum into adam', [online] <http://cs229.stanford.edu/proj2015/054_report.pdf>.

Drucker, H., Burges, C. J., Kaufman, L., Smola, A. J. & Vapnik, V. (1997), Support vector regression machines, *in* 'Advances in neural information processing systems', Vol. 9, pp. 155–161.

Du, S., Lee, J., Li, H., Wang, L. & Zhai, X. (2019), Gradient descent finds global minima of deep neural networks, *in* 'International Conference on Machine Learning', pp. 1675–1685.

Duchi, J., Hazan, E. & Singer, Y. (2011), 'Adaptive subgradient methods for online learning and stochastic optimization.', *Journal of machine learning research* **12**(7), 121–159.

Ebrary.net (nd.), 'Unexpected loss and var', [online] Ebrary.net <https://ebrary.net/12971/management/unexpected_loss#:~:text=Unexpected20losses20are20loss20percentiles,to20market20and20credit20risk>. Accessed: 2021-05-23.

Eddy, Y. L. & Abu Bakar, E. M. N. E. (2017), 'Credit scoring models: Techniques and issues', *Journal of Advanced Research in Business and Management Studies* **7**(2), 29–41.

Efron, B. (1992), Bootstrap methods: another look at the jackknife, *in* 'Breakthroughs in statistics', Springer, pp. 569–593.

Eletter, S. F., Yaseen, S. G. & Elrefae, G. A. (2010), 'Neuro-based artificial intelligence model for loan decisions', *American Journal of Economics and Business Administration* **2**(1), 27–34.

Ellis, L. (2008), 'The housing meltdown: Why did it happen in the united states?'.

Equifax (nd.), 'What is a credit report and what does it include? — equifax', [online] Equifax.com <https://www.equifax.com/personal/education/credit/report/what-is-a-credit-report-and-what-does-it-include/>. Accessed: 2021-05-23.

Faggella, D. (2021), 'Machine learning in finance - present and future applications', [online] Emerj <https://emerj.com/ai-sector-overviews/machine-learning-in-finance/>. Accessed: 2021-03-30.

Fan, J., Ma, C. & Zhong, Y. (2019), 'A selective overview of deep learning', *arXiv preprint arXiv:1904.05526* .

Farid, J. (2014), 'Expected loss unexpected loss, economic capital case study', [online] FinanceTrainingCourse.com <https://financetrainingcourse.com/education/2014/12/unexpected-loss-ul-expected-loss-economic-capital-case-study>. Accessed: 2021-05-23.

Flitton, M. (2018), 'Early stopping with sk-learn', [online] Medium, <https://medium.com/@maxwellflitton/early-stopping-with-sk-learn-5e949cc9f015>. Accessed: 2021-05-30.

Foote, K. D. (2019), 'A brief history of machine learning', [online] Dataversity <https://www.dataversity.net/a-brief-history-of-machine-learning/>. Accessed: 2021-03-30.

Friedman, J. H. (2001), 'Greedy function approximation: a gradient boosting machine', *Annals of statistics* **9**(5), 1189–1232.

Galindo, J. & Tamayo, P. (2000), 'Credit risk assessment using statistical and machine learning: basic methodology and risk modeling applications', *Computational Economics* **15**(1-2), 107–143.

Garling, C. (2018), 'Andrew ng: Why deep learning is a mandate for humans, not just machines', [online] Wired Magazine <https://www.wired.com/brandlab/2015/05/andrew-ng-deep-learning-mandate-humans-not-just-machines/>. Accessed: 2021-05-23.

Genest, B. & Brie, L. (2013), 'Basel ii irb risk weight functions', *Demonstration and analysis, Global Research and Analytics* p. 5.

Gepp, A., Kumar, K. & Bhattacharya, S. (2010), 'Business failure prediction using decision trees', *Journal of forecasting* **29**(6), 536–555.

Ghosh, A. (2012), *Managing risks in commercial and retail banking*, John Wiley & Sons.

Ghosh, S., Stephenson, W. T., Nguyen, T. D., Deshpande, S. K. & Broderick, T. (2020), 'Approximate cross-validation for structured models', *arXiv preprint arXiv:2006.12669* .

Giudici, P., Hadji-Misheva, B. & Spelta, A. (2019), 'Network based scoring models to improve credit risk management in peer to peer lending platforms', *Frontiers in Artificial Intelligence* **2**, 3.

Golbayani, P., Florescu, I. & Chatterjee, R. (2020), 'A comparative study of forecasting corporate credit ratings using neural networks, support vector machines, and decision trees', *The North American Journal of Economics and Finance* **54**, article no: 101251.

Goodfellow, I., Bengio, Y. & Courville, A. (2016), *Deep learning*, MIT press.

Goodman, L. S. & Zhu, J. (2018), 'What fueled the financial crisis? an analysis of the performance of purchase and refinance loans', *Journal of Fixed Income* **28**(1), 27–37.

Greenidge, K. & Grosvenor, T. (2010), 'Forecasting non-performing loans in barbados.', *Journal of Business, Finance & Economics in Emerging Economies* **5**(1), 80–108.

Guyon, I., Boser, B. & Vapnik, V. (1993), Automatic capacity tuning of very large vc-dimension classifiers, *in* 'Advances in neural information processing systems', pp. 147–155.

Hanin, B. & Sellke, M. (2017), 'Approximating continuous functions by relu nets of minimal width', *arXiv preprint arXiv:1710.11278* .

Hansen, L. K. & Salamon, P. (1990), 'Neural network ensembles', *IEEE transactions on pattern analysis and machine intelligence* **12**(10), 993–1001.

Hauptfleisch, K. (2016), 'This is how artificial intelligence will shape elearning for good', [online] eLearning Industry <https://elearningindustry.com/artificial-intelligence-will-shape-elearning>. Accessed: 2021-05-30.

Hayes, A. (2020), 'How the loan-to-value (ltv) ratio works', [online] Investopedia <https://www.investopedia.com/terms/l/loantovalue.asp>. Accessed: 2021-03-30.

Hebb, D. O. (1949), 'The first stage of perception: growth of the assembly', *The Organization of Behavior* **4**, 60–78.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. R. (2012), 'Improving neural networks by preventing co-adaptation of feature detectors', *arXiv preprint arXiv:1207.0580* .

Hinton, G., Srivastava, N. & Swersky, K. (2012), 'Neural networks for machine learning lecture 6a overview of mini-batch gradient descent', [online] <http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf>. Accessed: 2021-05-30.

Hornik, K. (1991), 'Approximation capabilities of multilayer feedforward networks', *Neural networks* **4**(2), 251–257.

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

Hsu, T.-C., Liou, S.-T., Wang, Y.-P., Huang, Y.-S. et al. (2019), Enhanced recurrent neural network for combining static and dynamic features for credit card default prediction, *in* 'ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)', IEEE, pp. 1572–1576.

Huang, E. & Scott, C. (2007), Credit risk scorecard design, validation and user acceptance, *in* 'Credit Scoring and Credit Control X Conference'.

iUnera (n.d.), 'A brief overview of support vector machines', [online] iunera <https://www.iunera.com/kraken/fabric/support-vector-machines-svm/>. Accessed: 2021-05-30.

James, G., Witten, D., Hastie, T. & Tibshirani, R. (2013), *An introduction to statistical learning*, Vol. 112, Springer.

Jensen, H. L. (1992), 'Using neural networks for credit scoring', *Managerial finance* **18**(6), 15–26.

Jones, D. S. & Mingo, J. (1998), 'Industry practices in credit risk modeling and internal capital allocations: Implications for a models-based regulatory capital standard: Summary of presentation', *Economic Policy Review* **4**(3).

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q. & Liu, T.-Y. (2017), 'Lightgbm: A highly efficient gradient boosting decision tree', *Advances in neural information processing systems* **30**, 3146–3154.

Keim, R. (2019), 'How to train a basic perceptron neural network', [online] All About Circuits <https://www.allaboutcircuits.com/technical-articles/how-to-train-a-basic-perceptron-neural-network/>. Accessed: 2021-05-30.

Khandani, A. E., Kim, A. J. & Lo, A. W. (2010), 'Consumer credit-risk models via machine-learning algorithms', *Journal of Banking & Finance* **34**(11), 2767–2787.

Khashman, A. (2010), 'Neural networks for credit risk evaluation: Investigation of different neural models and learning schemes', *Expert Systems with Applications* **37**(9), 6233–6239.

Khemakhem, S. & Boujelbene, Y. (2017), 'Artificial intelligence for credit risk assessment: Artificial neural network and support vector machines', *ACRN Oxford Journal of Finance and Risk Perspectives* **6**(2), 1–17.

Kiefer, J., Wolfowitz, J. et al. (1952), 'Stochastic estimation of the maximum of a regression function', *The Annals of Mathematical Statistics* **23**(3), 462–466.

Kim, H. S. & Sohn, S. Y. (2010), 'Support vector machines for default prediction of smes based on technology credit', *European Journal of Operational Research* **201**(3), 838–846.

Kingma, D. P. & Ba, J. (2014), 'Adam: A method for stochastic optimization', *arXiv preprint arXiv:1412.6980* .

Klein, A. (2020), 'Reducing bias in ai-based financial services', [online] Brookings <https://www.brookings.edu/research/reducing-bias-in-ai-based-financial-services/>. Accessed: 2021-03-30.

Koh, H. C., Tan, W. C. & Goh, C. P. (2006), 'A two-step method to construct credit scoring models with data mining techniques', *International Journal of Business and Information* **1**(1), 96–118.

Kushner, H. & Yin, G. G. (2003), *Stochastic approximation and recursive algorithms and applications*, Vol. 35, Springer Science & Business Media.

Landis, J. R. & Koch, G. G. (1977), 'The measurement of observer agreement for categorical data', *Biometrics* **33**(1), 159–174.

Lau, S. (2017), 'Learning rate schedules and adaptive learning rate methods for deep learning', [Online] TowardDataScience https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning. Accessed: 2021-03-30.

Lee, T.-S., Chiu, C.-C., Chou, Y.-C. & Lu, C.-J. (2006), 'Mining the customer credit using classification and regression tree and multivariate adaptive regression splines', *Computational Statistics & Data Analysis* **50**(4), 1113–1130.

Li, S.-T., Shiue, W. & Huang, M.-H. (2006), 'The evaluation of consumer loans using support vector machines', *Expert Systems with Applications* **30**(4), 772–782.

Lindholm, A., Wahlström, N., Lindsten, F. & Schön, T. B. (2019), *Supervised Machine Learning: Lecture notes for the Statistical Machine Learning course*, Department of Information Technology, Uppsala University.

Lu, Z., Pu, H., Wang, F., Hu, Z. & Wang, L. (2017), The expressive power of neural networks: A view from the width, *in* 'Advances in neural information processing systems', Vol. 30, pp. 231–239.

Luthi, B. (2018), 'What is risk-based pricing?', [online] Experian.com <https://www.experian.com/blogs/ask-experian/what-is-risk-based-pricing/>. Accessed: 2021-05-23.

Madaan, M., Kumar, A., Keshri, C., Jain, R. & Nagrath, P. (2021), Loan default prediction using decision trees and random forest: A comparative study, *in* 'IOP Conference Series: Materials Science and Engineering', Vol. 1022, IOP Publishing, p. 012042.

Maes, S., Tuyls, K., Vanschoenwinkel, B. & Manderick, B. (2002), Credit card fraud detection using bayesian and neural networks, *in* 'Proceedings of the 1st international naiso congress on neuro fuzzy technologies', pp. 261–270.

Maini, V. & Sabri, S. (2017), 'Machine learning for humans', [online] Everythingcomputerscience.com <https://everythingcomputerscience.com/books/MachineLearningforHumans.pdf>. Accessed 2021-04-29.

Martens, D., Baesens, B., Van Gestel, T. & Vanthienen, J. (2007), 'Comprehensible credit scoring models using rule extraction from support vector machines', *European journal of operational research* **183**(3), 1466–1476.

Mashanovich, N. (2017), 'Credit scoring: Part 1 - the development process from end to end', [online] Worldprogramming.com <https://www.worldprogramming.com/blog/datascience/credit_scoring_development_pt1/>. Accessed: 2021-05-23.

MathWorks (n.d.), 'Convolutional neural network', [online] Mathworks <https://www.mathworks.com/discovery/convolutional-neural-network-matlab.html>. Accessed: 2021-05-30.

Min, J. H. & Lee, Y.-C. (2005), 'Bankruptcy prediction using support vector machine with optimal choice of kernel function parameters', *Expert systems with applications* **28**(4), 603–614.

Nesterov, Y. (1983), A method for unconstrained convex minimization problem with the rate of convergence o (1/kˆ 2), *in* 'Doklady an ussr', Vol. 269, pp. 543–547.

Nesterov, Y. (2013), *Introductory lectures on convex optimization: A basic course*, Vol. 87, Springer Science & Business Media.

O'Neil, C. (2017), *Weapons of math destruction*, Penguin Books.

Orr, G. B. (1999), 'Momentum and learning rate adaptation', [online] Willamette University <https://www.willamette.edu/~gorr/classes/cs449/momrate.html>. Accessed: 2021-05-30.

Paul, S. (2018), 'Ensemble learning in python', [online] Datacamp <https://www.datacamp.com/community/tutorials/ensemble-learning-python>. Accessed: 2021-05-30.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. & Duchesnay, E. (2011), 'Scikit-learn: Machine learning in Python', *Journal of Machine Learning Research* **12**(2011), 2825–2830.

Pinkus, A. (1999), 'Approximation theory of the mlp model in neural networks', *Acta numerica* **8**(1), 143–195.

Polikar, R. (2009), 'Ensemble learning', *Scholarpedia* **4**(1), article no: 2776. revision #186077.

Polyak, B. T. & Juditsky, A. B. (1992), 'Acceleration of stochastic approximation by averaging', *SIAM journal on control and optimization* **30**(4), 838–855.

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

Robbins, H. & Monro, S. (1951), 'A stochastic approximation method', *The annals of mathematical statistics* **22**(3), 400–407.

Rosenblatt, F. (1958), 'The perceptron: a probabilistic model for information storage and organization in the brain.', *Psychological review* **65**(6), 386–408.

Roth, A. E. (1988), *The Shapley value: essays in honor of Lloyd S. Shapley*, Cambridge University Press.

Ruder, S. (2016), 'An overview of gradient descent optimization algorithms', *arXiv preprint arXiv:1609.04747* .

Sahay, M. R. (2015), *Financial inclusion*, International Monetary Fund.

Samuel, A. L. (1959), 'Some studies in machine learning using the game of checkers', *IBM Journal of research and development* **3**(3), 210–229.

Santos, K. (2009), 'Corporate credit ratings: a quick guide', [online] Treasurer's Companion <https://www.treasurers.org/ACTmedia/ITCCMFcorpcreditguide.pdf>. Accessed: 2021-03-30.

Sayjadah, Y., Hashem, I. A. T., Alotaibi, F. & Kasmiran, K. A. (2018), Credit card default prediction using machine learning techniques, *in* '2018 Fourth International Conference on Advances in Computing, Communication & Automation (ICACCA)', IEEE, pp. 1–4.

Schapire, R. E. (1990), 'The strength of weak learnability', *Machine learning* **5**(2), 197–227.

Schroeck, G. (2002), *Risk management and value creation in financial institutions*, John Wiley & Sons.

Scikit-Learn (2020), 'Cross-validation: evaluating estimator performance', [online] Scikit-learn <https://scikit-learn.org/stable/modules/cross_validation.html>. Accessed: 2021-05-30.

Seiffert, C., Khoshgoftaar, T. M., Van Hulse, J. & Napolitano, A. (2008), Rusboost: Improving classification performance when training data is skewed, *in* '2008 19th International Conference on Pattern Recognition', IEEE, pp. 1–4.

Sharma, S. (2019), 'How to classify non-linear data to linear data?', [online] Medium, <https://medium.com/analytics-vidhya/how-to-classify-non-linear-data-to-linear-data-bb2df1a6b781>. Accessed: 2021-05-30.

Shin, K.-S., Lee, T. S. & Kim, H.-j. (2005), 'An application of support vector machines in bankruptcy prediction model', *Expert systems with applications* **28**(1), 127–135.

Smith, L. D. & Lawrence, E. C. (1995), 'Forecasting losses on a liquidating long-term loan portfolio', *Journal of Banking & Finance* **19**(6), 959–985.

Spilak, B. & Härdle, W. K. (2020), 'Tail-risk protection: Machine learning meets modern econometrics', *Available at SSRN 3714632* .

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. (2014), 'Dropout: a simple way to prevent neural networks from overfitting', *The journal of machine learning research* **15**(1), 1929–1958.

Sukhadeve, A. (2017), 'Introduction to logistic regression', [online] Analyticsinsight.net <http://www.analyticsinsight.net/introduction-to-logistic-regression/>. Accessed: 2021-05-30.

Sun, T. & Vasarhelyi, M. A. (2018), 'Predicting credit card delinquencies: An application of deep neural networks', *Intelligent Systems in Accounting, Finance and Management* **25**(4), 174–189.

Sutton, R. S. (1992), Introduction: The challenge of reinforcement learning, *in* 'Reinforcement Learning', Springer, pp. 1–3.

The Bank for International Settlements (1999), *Principles for the Management of Credit Risk*, Basel Committee on Banking Supervision.

The Bank for International Settlements (2001), *The Internal Ratings-Based Approach*, Basel Committee on Banking Supervision.

The Bank for International Settlements (2003), *Overview of The New Basel Capital Accord*, Basel Committee on Banking Supervision.

UniCredit (2012), 'Credit risk', [online] UniCreditGroup.eu <https://web.archive.org/web/20130927181311/https://www.unicreditgroup.eu/en/investors/risk-management/credit.html>. Accessed: 2021-03-30.

Van Gestel, I. T., Baesens, B., Garcia, I. J. & Van Dijcke, P. (2003), A support vector machine approach to credit scoring, *in* 'Forum Financier-revue Bancaire et Financiare Bank en Financiewezen', Citeseer, pp. 73–82.

Vapnik, V. N. & Chervonenkis, A. (1964), 'On a perceptron class', *Automation and Remote Control* **25**(1), 112–120.

Vasicek, O. (2002), 'The distribution of loan portfolio value', *Risk* **15**(12), 160–162.

Vojtek, M., Koèenda, E. et al. (2006), 'Credit-scoring methods', *Czech Journal of Economics and Finance (Finance a uver)* **56**(3-4), 152–167.

Wang, C., Han, D., Liu, Q. & Luo, S. (2018), 'A deep learning approach for credit scoring of peer-to-peer lending using attention mechanism lstm', *IEEE Access* **7**, 2161–2168.

Wang, S. & Yao, X. (2009), Diversity analysis on imbalanced data sets by using ensemble models, *in* '2009 IEEE Symposium on Computational Intelligence and Data Mining', IEEE, pp. 324–331.

Wasicek, A. (2018), 'Artificial intelligence vs. machine learning vs. deep learning: What's the difference?', [online] Sumo Logic <https://www.sumologic.com/blog/machine-learning-deep-learning/>. Accessed: 2021-05-30.

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

Wolff, V. (2020), 'Risk-based pricing: Strategies for a new reality in banking', [online] ThoughtWorks <https://www.thoughtworks.com/insights/blog/risk-based-pricing-strategies-new-reality-1>. Accessed: 2021-05-23.

Yee, O. S., Sagadevan, S. & Malim, N. H. A. H. (2018), 'Credit card fraud detection using machine learning as data mining technique', *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)* **10**(1-4), 23–27.

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

Yun, H., Lee, M., Kang, Y. S. & Seok, J. (2020), 'Portfolio management via two-stage deep learning with a joint cost', *Expert Systems with Applications* **143**, article no: 113041.

Zeiler, M. D. (2012), 'Adadelta: an adaptive learning rate method', *arXiv preprint arXiv:1212.5701* .

Zhou, J., Li, W., Wang, J., Ding, S. & Xia, C. (2019), 'Default prediction in p2p lending from high-dimensional data based on machine learning', *Physica A: Statistical Mechanics and its Applications* **534**, article no: 122370.