MACHINE AND DEEP LEARNING BASED CREDIT DEFAULT PREDICTION

A RISK-BASED PRICING MODEL APPLIED IN PEER-TO-PEER LENDING*

Mads Jensen†

3. juni 2019

MAY BE PUBLISHED 3TH JUNE 2019

AARHUS UNIVERSITY, DEPARTMENT OF ECONOMICS AND BUSINESS
SCHOOL OF BUSINESS AND SOCIAL SCIENCES
MASTER OF SCIENCE IN ECONOMICS

---

*Superviser: Morten Berg Jensen, Departement of Economics, Aarhus University
†StudyID: 201408587
E-mail: madsjensen231@gmail.com
Key strokes: 187650 ~78 normal pages
Danish Title: Forudsigelse af konkursbegæring baseret på maskinlæringsmetoder

Abstract

Credit to Small and Medium sized Enterprises (SMEs) is still well below the amount needed to close the European credit supply gap. Two technological developments are currently helping to close the gap. Machine- and deep learning based credit risk models and peer-to-peer market designs. The thesis looks at several machine and deep learning based credit risk models, to illustrate how and why the technologies improve the supply of SME credit. The credit risk models are trained on a broad collection of European SMEs. Using risk-based pricing, the performance of the models are evaluated. First, the models are evaluated on their ability to form risk specific portfolios of SME credit, with the lowest amount of bad investments in them. The benchmark logistic regression model produces the lowest rate of defaults across risk specific portfolios, and the gradient boosting the most. Using this result, the best models are compared on their ability to produce profits across different risk categories, in a simplified peer-to-peer lending application. Here, the gradient boosting algorithm produces by far the highest profit. The fact that the worst model, in terms of error rate, is the most profitable, forms the discussion for the clear moral hazard problem that is inherent to the peer-to-peer industry. I conclude that regulatory oversight, transparency concerning credit risk, and informed private investors will be critical parameters for the the peer-to-peer sector's continued ability to decrease the SME credit supply gap.

Keywords: machine learning, credit risk, non-symmetrical loss, peer-to-peer lending

Contents

List of Figures

List of Tables

1 Introduction

Small- and medium-sized enterprises (SMEs) are daily confronted with challenges such as high operational risks, capital constraints, and high financing costs. The current monetary climate, with rock bottom interest rates, has not materialized in lower debt financing costs for SMEs, on the same scale it has for larger corporations. Significant discrepancies between debt service costs for SMEs and large corporations still persist. The SME credit market is characterized by low competition, fragmentation, difficulty in risk assessment and consequently, significant premiums for the traditional banks that dominate the SME credit market. As a result, solutions to improve the SMEs financing conditions focus on alternative market designs and optimization of credit risk analysis tools. Recently, the peer-to-peer lending market design has moved into the market for SME credit, with Funding Circle being the most prominent. Funding Circle has recently expanded from its original UK market to all major European markets and the US. The new peer-to-peer market design's low cost base and advanced credit risk models have brought disruption to the once cozy SME credit market. Advances in machine and deep learning based credit risk analysis, along with the peer-to-peer market design have helped to decrease the SME credit supply gap[1].

At first glance, machine learning might seem similar to econometrics. Both fields are based on finding empirical relations in data using statistical models. There is however, a big difference in the way this relationship is modeled. Whereas econometrics developed as a tool for economists to explain the world, machine learning is only concerned with prediction. Economists prefer parametric models as they relate explaining variables to an output variable. We (as in Economists) can then model a real-world phenomenon, such as default risk and understand how each component relates to each other and what effect each of the explaining variables has on the output variable. In machine learning, we apply non-parametric models with no assumptions on the underlying structural form of the problem. With forecasting, the goal is not necessarily an understanding of the underlying functional form of the problem, here the explaining variables effect on the probability of default, but more so with providing true predictions about future events. Hence, in machine learning, it is not possible to make causal inference. The non-parametric forecasting tools such as machine learning can be a black-box when an explanation on why the specific prediction has been made. The question of whether machine learning is the right tool for credit default prediction among SME's boils down to the following: is predictive ability more important than causal inference? In the particular case of providing SME finance at a competitive price, the answer is yes.

[1]SME finance Monitor Q2 2018, by BVA BDRC. The report writes that the credit supply gap is falling. They accredit some of the decrease to new alternative finance solutions such as peer-to-peer lending.

The performance of the chosen machine and deep learning models will be compared to the benchmark logistic regression model. The best performing model is the gradient boosting and random forest model from the tree family, which both achieved an Area Under the Curve Score (AUC) of 0.82. However, the logistical regression model is not far behind with an AUC score of 0.76. Section 6 gives a full presentation of all the model results.

In this thesis, I set out to build the best possible prediction model for a 24−month SME default probability, using the historical financial performance of the three previous years leading up to the forecast. This situation mirrors the real problem of an SME credit investor. Should the SME receive a loan running for 24 months? If yes, then what should the price of risk be? Here, the existing literature stops, and as will be seen, several important discoveries are related to this question. The profitability of this investment decision ultimately boils down to the models' ability to determine and pricing risk correctly. To answer this question, I develop a firm-specific credit rating tool and use the credit ratings to investigate the profitability of each of the models. Later, pricing implications of a peer-to-peer market design will be compared to the traditional banking business model.
Economists face new problems with the use of machine learning over econometrics, and this is also true for the increased use of peer-to-peer market designs, compared to traditional banking. The issues relating to both technological developments will be discussed as well.

A central part of the superior prediction power of machine and deep learning comes from the ability to handle large amounts of explanatory variables. To make the most of it, all relevant information available was used for this task. The Orbis database stores a fairly large amount of data on private SMEs, considering that these companies are private. The data scrape includes 20 variables from the financial statements of each SME, sampled at three different time-periods. With 42,306 observations, the data scrape consists of over 2.5 million data points, covering financial information of private European SMEs. The raw data is then transformed using feature engineering consistent with the literature, to expand the feature space to $29 \cdot 3 = 87$ explanatory features. I explain how, in prediction, the transformation of raw data to features is an area that has not moved considerably since default probability was first quantified in literature. Advances in machine and deep learning have given the models internal capabilities of feature engineering, and the manual data preprocessing part has become increasingly irrelevant. Future research in predictive power using raw data as inputs, along with behavioral data, is urgently needed. Economists could play an important role in this development. A complete description of the extensive data collecting will

be unfolded in section 5.

An area where the predictive power of the models is still dependent on their creator is when the data contains significant imbalances. Although the data contains 42,306 observations, only 3% of the observations are classified as default. The imbalanced data severely limits the models' ability learn what a default looks like, and different variations in characteristics of default. Consequently, conventional and advanced synthetic measures are taken to ensure that the models will learn from balanced training data. The computational expenses related to these synthetic methods puts a hardware constraint on the dimensions of the training data. Subject to RAM constraints, I end up with balanced training data containing 9090 active classifications and 9315 default observations. The theoretical framework and practical considerations related to the imbalanced data will be fully explained in section 5.

Although the historical financial information related to each SME can be used to determine the default probability with good results, the optimal prediction model will use other types of data as well. Behavioral data, qualitative data relating to the industry, business, and management could significantly increase the predictive power. Also, data explaining the terms and conditions for the SMEs current credit contracts for long and short term debt could improve the predictive power of the models, and thus maximize the profit-objective function. There are endless avenues to explore, and from which data can be incorporated into the prediction models. There are however constraints on the amount of data available. Since the SMEs are inherently private, the publicly available data is very limited. There are two possible solutions to expanding the explanatory data. The hands-off approach is to proxy the qualitative data through industry codes and other available data. The hands-on approach requires the ability to monitor the SME. Once a loan is approved by Funding Circle, they get access to the accounts of the SMEs and can hence effectively increase the data complexity and the data sampling frequency[2]. The increase in data complexity makes it easier to distinguish different companies, and the information asymmetry between the principal and the agent is decreased considerably. Ultimately, as the data complexity increases, so will the ability to provide loans at better prices for the SMEs. Specific suggestions for data and model improvement will be discussed in section 7 and 8.

There are however plenty of pitfalls with respect to the predictions made by the models. First, since the models are built on correlations and non-structural, they emulate the patterns of yester-

[2]This is not explicitly written, but it can be deducted from their data gathering process, explained various places, for example https://www.indexventures.com/index-insight/behind-the-scenes-with-funding-circle.

day. The decision of whether or not to provide a loan at a given price depends on the previous decisions made since a rejection of a loan application might default the company. Consequently, a fully implemented and automated SME credit investment decision model will change the data generating process on which the prediction model was based. Second, the behavior of economic agents might change and lead to adverse behavior to increase credit conditions Edelberg 2006. In such a situation, economists are better suited than any to solve these pitfalls. Even though the problem of default prediction has increasingly moved towards the expertise of computer scientists, the fundamental behavior and issues related to this market are inherently related to classical economics.

A note on the academic contributions of the thesis. First, while other authors have trained machine and deep learning models to describe their potential, they end their research with a comparison of AUC scores. While this requires a considerable amount of work, it fails to answer the question of performance in a real world application. In this thesis, I develop an economic framework for mirroring the real world performance of the individual models. It is to the best of my knowledge, the first time professional credit ratings and peer-to-peer pricing structures have been incorporated to test model performance. I believe this thesis stands out by using an economic framework, with real-world specifications, to compare model performance. I have not elsewhere seen a formal quantification of model performance in terms of credit ratings, interest rates, returns, and market design.
Second, the literature does not cover as many different machine and deep learning models as this thesis. The relatively large amount of models is used to close the research gap in the literature. Specifically, the performance of the two leading models has not been tested against each other.
Third, with inspiration from the literature covering default prediction of consumer credit, I implement, for the first time, a Long Short-Term Memory model to predict SME default, using firm-specific financial panel-data.

1.1 Thesis structure

The thesis is structured as follows. Section 2 covers the literature of default prediction and the newest literature using machine learning to predict default among SMEs. A SME credit market description follows the literature review, in which the market failures are used to motivate this thesis. Next, a real-world solution to the market failures is exemplified through a description of Funding Circle, following a presentation of industry standard SME credit rating methodology. Section 3 formally introduces the central concepts in machine learning. In section 4, the eight

different algorithms used to predict default are formally introduced. Section 5 covers the data and the extensive preprocessing methods used. Section 6 presents the results across models and an analysis and discussion of the best models. Section 7 applies the results through an economic framework to evaluate the model in a financial technology application. Section 8 concludes.

2 Machine learning applications in SME credit risk

2.1 Literature review

The study of enterprise default prediction is an enormously studied area within econometrics. Beaver 1966 originally developed an uni-variate toolbox, based on financial ratios from a firm's income statement and balance sheet to predict the risk of default. Altman 1968, Altman et al. 1977 and later Pompe and Bilderbeek 2005 used Multiple Discriminant Analysis (MDA), to develop default prediction models based on financial ratios. Pompe and Bilderbeek 2005 builds on Altman's classical MDA study and compares the forecasting ability of MDA with Neural Network models. They find that Neural Network based models show overall better forecasting ability compared to the MDA models. The clear disadvantage of MDA is that it assumes an identical variance-covariance matrix for both the non-default and the default population. Consequently, observations that do not fit within this assumption will decrease the forecasting precision of MDA. Earlier research Ohlson 1980, Aziz et al. 1988, Aziz and Lawson 1989 compares the default prediction performance of MDA against the logistic regression model. They find that the logistic regression model outperforms Multivariate Discriminate Analysis. The lesser degree of statistical assumptions in the logistic regression, namely that no underlying assumption is made on the classification's variance-covariance matrix, gives the logistical model an advantage compared to MDA. Besides being better at predicting default, the logistical model also offers superior inference capabilities, known from classical economics. Based on these previous findings, this thesis will not use Multiple Discriminate Analysis to perform default prediction. Instead, this thesis will focus on non-parametric models from machine learning and evaluate their respective forecasting performance against the benchmark logistic regression model. However, the strict assumptions from classical parametric models, where the response variable is directly related to the predictor variable in an econometric framework limit the real world application. If the goal is to provide superior predicting performance, the non-parametric frameworks from machine learning need to be included in the pool of possible models.

Narrowing down to the lesser researched area of default prediction among private SMEs, which this thesis will focus on. Recent similar papers have shown superior prediction performance of different machine learning models, compared to the logistic regression model. Addo et al. 2018, Zhu et al. 2017 find that the gradient boosting algorithm, along with random-forest out-performs both the regularized logistic regression (elastic-net model) and simple neural networks with different amount of layers. They do find, however that a three hidden layer neural network is the best among the

neural networks, and not far behind the two superior models. Min and Lee 2005 investigate the same problem, but with a large emphasis on support vector machines instead. They find that a SVM with radial kernel and a $3 - degree$ polynomial kernel are the best models. Similarly, their three hidden-layer neural network is not far behind in prediction performance. Their logistic regression model was better than MDA, but had a significantly lower accuracy compared SVM and neural network. Hence, existing literature suggests that gradient boosting, random forest or a SVM with a radial or 3−degree polynomial kernel should be the best models at predicting SME default risk. The three-layered neural network setup is not far behind the respective models, in both papers. This paper will use these previous findings as a starting point for model development. Also, a comparative study between the superior support vector machine from Min and Lee 2005 and the two tree-based methods from Addo et al. 2018 and Zhu et al. 2017 is not covered in the literature. A comparative study in model performance will be conducted in this thesis.

Looking at the literature on the more technology developed private banking market, where the target is personal credit default risk. Here, peer-to-peer lending platforms have revolutionized the industry, especially in Asia. The underlying credit risk assessment technology for peer-to-peer private lending is based on machine and deep learning models[3]. The new technology-based entrants do not have the same legal requirements with respect to transparency in their credit risk assessment, as traditional banks. Thus, for these peer-to-peer lenders, the black-box problem of machine learning is not a huge legal issue. Among the newest research is an application of the LSTM model in peer-to-peer consumer lending in China Wang et. al 2019 and Zhang et. al 2017, incorporating large amount of behavioral and qualitative data, that traditionally would have been thought irrelevant. The LSTM model has not yet been applied using SMEs financial information to predict SME specific default risk in the literature. Hence, this thesis will implement the LSTM model on this problem for the very first time. An artificial neural network with three hidden layers, will be implemented as well. Besides bringing methods from the more advanced private banking peer-to-peer market into the context of SME credit risk, this paper will test the best models achieved from Addo et. al. 2018, Zhu et al. 2017 (Gradient boosting, random forest) against the best models of Lee and Min 2005 (Support Vector Machines). At last the machine and deep learning methods will be compared to the benchmark logistic regression model.

[3]https://www.rstudio.com/wp-content/uploads/2015/10/Funding-Circle-Story1.pdf

2.2 The SME credit market

The Euro-zone SME credit demand could reach a high of almost EUR 1.4 tn. or 12% of Euro-zone GDP[4] in 2019. This compares with an estimated SME credit supply of EUR 1022 b., or 9% of Eurozone GDP. Since European SMEs depend on banks for 70% of their external financing (against 40% in the US), any gap between loan demand and supply could lead to lower investment growth, if the European SMEs don't have access to alternative financing. This is a serious constraint for Euro-zone economic growth. SMEs represent more than 99% of all European non-financial corporations and employ over 90 million people, accounting for almost 70% of total employment in the EU-28-non-fiancial sector. They also generate close to 60% of total gross value added.

Loans issued to SME's carry a considerable positive interest spread compared to large loans. On average, SMEs with loans on less than EUR 1m pay 1.5% percentage points more on debt service for loans running 1-3 years, compared to larger firms with larger loans. This is what is known as the size-spread in SME credit. Across EU countries, the size spread is between 0.5 to 3 percentage points. This is somewhat surprising since traditional finance theory suggests that ceteris paribus, the risk of default increases with loan size, Stiglitz 1972. Several factors can be used to explain the size-spread phenomenon. First, in the presence of fixed screening costs, small loans carry a higher interest rate in order to cover these initial costs. Second, smaller lenders are more heterogeneous by nature and may possess different characteristics Moore and Craigwell 2003**,** or use the borrowed funds for a variety of financing purposes, e.g. working capital, instead of long term investments in plant, property, and equipment. The fact that the size spread is especially large for short term loans less than three-months provides support for both the hidden characteristics and fixed cost argument. Third, banks controls a higher degree of power over SMEs than larger corporations. Larger corporations can effectively increase the supply side by auctioning off their debt through corporate bonds, whereas SMEs practically only have the offer that their current bank gives them. The relatively large switching costs that SMEs face by taking a competing offer from another bank, severely limits competition in the SME credit market Lam et al. 2009, Panther and Farquhar 2004. Thus, new methods are needed to solve the risk pricing problem and the market problem as well. This paper investigates the solutions through machine learning based credit risk and the new peer-to-peer market design.

[4]Euler Hermes " European SMEs: filling the bank financing gap"

2.2.1 Funding Circle: A Machine learning application on the SME credit market

Advances in technology and lucrative premiums have enabled new types of entrants to increase the competition in the SME credit market. Funding Circle is the most prominent Financial Technology based entrant into the SME credit market. Funding Circle's market design is based on a peer-to-peer lending platform. They provide SMEs with loans at better prices than the incumbent banks, by connecting private and public investors to SMEs. Funding Circle does the credit risk analysis and give each loan a credit rating, which is then offered to investors. Their credit risk models build on at least two years of firm-specific financial data and other quantitative and qualitative data[5]related to each SME. Based on the SMEs financial data, loan size, payback period, financing purpose and qualitative data, a credit rating is given to the SMEs. Based on the credit rating, Funding Circle then matches SME loans with different risk profiles and interests to investors. Without going any further into the business model, what effectively happens is that the supply side of SME credit market increases, by making SME credit available to investors who would not be able to invest in SME credit otherwise. Although Funding Circle takes the largest share of the added value created, it does result in lower prices, at better terms for the individual SME. Funding Circle's popularity among SMEs and investors was reflected in the £1.5 billion market capitalization that was achieved after their IPO in September 2018. Their profitability ultimately boils down to determining and pricing risk correctly. A machine learning approach to credit risk analysis together with classical credit risk analysis is at the core of their credit risk assessment and pricing[6]. The business model of Funding Circle will be the departure for the real world application, upon which the models will be compared i section 7. Later follows an in-depth discussion on the vast possibilities, but also new risks that this market design has.

2.2.2 TRIBrating: A SME credit rating agency

Euler Hermes ratings and Moody's launched a joint SME credit rating agency called TRIBrating in 2017. They have trained an undisclosed statistical model for each major European country, using country-specific firms and variables. Their undisclosed statistical credit risk model is based on the two financial statements leading up a forecast, and a forecast period of two years as well. They use 42.000 distinct French SMEs to train their French model[7]. Their final credit rating is a weighting, where the probability of default, calculated through their credit risk model, counts

[5]https://support.fundingcircle.com/hc/en-us/articles/115004728803-How-are-the-businesses-I-am-lending-to-assessed-

[6]https://support.fundingcircle.com/hc/en-us/articles/115004728803-How-are-the-businesses-I-am-lending-to-assessed-

[7]Euler Hermes Rating GmbH "Rating Validation Study, 27 September 2018"

Table 1: Default rate to credit rating conversion TRIBrating

Rating	Predicted probability of default
AA or higher	0%
A	0.28%
BBB	0.78%
BB	1.87%
B	4.91%
CCC or lower	10.61%

for 70%, the business profile is 17.5%, and the sector profile is 12.5%. Hence, the majority of a credit rating given by a professional credit rating agency comes from the implied probability of default created by the prediction model. The TRIBrating's implied default rate to credit rating conversion for the French market is presented in table 1.

What is not shown here, but can can be seen from their country-specific credit risk models; country level heterogeneity persist across Europe. Both in terms of general economic environment, the willingness of banks to extend credit and what is considered poor financial performance. This thesis uses pan-European data in order to increase the sample size. Initially, the goal was to develop a Denmark specific SME credit risk model however, due to the sparsity in data, a European model is built instead. The predicted default rate to credit rating conversion made by TRIBrating on the French market is approximately a representation of a European average. Hence, I will use this methodology build a pan-European SME credit rating model.

2.3 A financial technology application

I will start out by building default prediction models, using the best machine and deep learning methods from literature and compare them to the benchmark logistic regression model. Next, I will use a risk-based pricing system Oliver and Wells 2001, Keeney and Oliver 2005, to develop a SME credit rating and pricing model. The risk and pricing structures will be applied in a Funding Circle like implementation, to asses the profitability of the different models in a peer-to-peer application. Thus, this thesis contains elements from both computer science and economics. As also noted by L.C. Thomas 2009, this is an area where multi-disciplinary requirements are needed.

3 Central concepts in machine learning

This section will go through the central concepts of machine learning. This section starts with a brief comparison between classical econometrics and machine learning, using the difference between the two as a starting point for the central concepts in machine learning. In econometrics, the starting point for any empirical analysis is derived from economic theory. Under the assumption that the theory is a true representation of the underlying data generating process, valid inference on relations in data can be made. In machine learning, the main focus is narrower in the sense that the only interest is in building a model that provides the best out-of-sample predictions. Machine learning is also wider than econometrics in the sense that since no underlying data generating process is assumed, the models need to be flexible enough to handle highly complex patterns in data. This is often impossible in econometrics. Machine learning models however, needs to be constrained in the searchable space that is available during the learning process. Specifically, it is necessary to control for over-fitting to random temporary variations in the training data. Over-fitting, which is the main risk of training unconstrained machine learning models will result in poor out-of-sample predictions that, most often than not, will make the classical models from econometrics better at predicting because they capture average ceteris paribus effects. In this section, practical solutions to solve the over-fitting problem, as well as the theory behind it, will be presented.

First, follows a presentation of the different classifiers which as a group are known as learnable functional classes. This is done in section 3.1. Each learnable functional class is trained to learn from the training data based on a loss function. This is an identical concept to econometrics. Different types of loss functions, both algorithm specific and universally applied to all classifiers will be presented in section 3.2. The bias-variance trade-off will be presented and analyzed in section 3.3. The bias-variance trade-off captures the relationship between having an unconstrained model, over-fitted to capture all patterns in the training data, and the effect it has on the out-of-sample error loss. Section 3.4 contains a theoretical extension and in-sights to the bias-variance trade-off. Section 3.5 and 3.6 contains practical solutions to the over-fitting problem. Regularization is the subject of Section 3.5. Regularization is a systematic solution that controls for the complexity during the model training. By adding a regularization term to the loss function, any further complexity in the classifier must explain more than the regularization term adds to the total loss. The art of finding the right level of punishment for complexity is the subject of tuning in section 3.6. By analyzing the effects that different values of complexity punishment have on proxy out-of-sample prediction loss, it is possible to find optimal values for the complexity. This is known as cross-

validation. Both automatic multi-value search algorithms and manual trial-and-error approaches can be used to find optimal complexity during the model tuning. The proxy out-of-sample error loss is calculated for different values of the tuning parameters through cross-validation on partitioned parts of the training data. The subject of cross-validation is likewise explained in section 3.6. Section 3.7 introduces the metrics that will be used to compare the performance of the different models. Most of section 3 builds heavily on Vapnik 1992.

3.1 The fundamental prediction problem

Let A denote any algorithm used to produce a prediction model, $\hat{f}$. On a side note - The prediction model $\hat{f}$ will be mentioned interchangeably as the classification model, since the prediction problem of this thesis is a matter of predicting a classification. First, the functional class over which the algorithm, A, searches to find a classifier $\hat{f}_A$ needs to be specified. After the functional class of the classifier $\hat{f}_A$ is specified, the loss function of the classifier needs to be specified in order to measure prediction performance. This can be summed up in the following

$$A : (\lambda, \alpha; S \mid \mathcal{F}_A, L, R) \mapsto \hat{f}_S \in \mathcal{F}$$
$$\hat{f}_S = \arg\min_{f \in \mathcal{F}} \mathbb{E}\left[L\left(f\left(\boldsymbol{X}, S\right), Y\right) \mid S\right] + \lambda \cdot R\left(f\right).$$

The algorithm A is a mapping that produces a prediction model $\hat{f}_S : \mathcal{X} \mapsto \mathcal{Y}$, where $(\boldsymbol{X} \in \mathcal{X})$, based on the sample training data, $S = \{(x_i, y_i) : i = 1, ..., N\}$. Each algorithm A is based on different conceptual frameworks and is therefore restricted in the learnable functional class $\mathcal{F}$ before the algorithm is trained on the sample data. The learnable function class $\mathcal{F}$ captures how the algorithm transforms explanatory data x into a prediction $\hat{y}$. This transformation is governed by a set of parameters, known collectively as α.

The loss function L measures how well the prediction model $\hat{f}_S$, created with a learnable functional class $\mathcal{F}$, given the training data S, performs at predicting the right class. The loss function is both used to tune the model parameters on the cross-validation set, in order to optimize the proxy out-of-sample performance, as well as an out-of-sample metric to compare different algorithms. Since different algorithms use different learnable functional classes $\mathcal{F}$, it is also interesting to analyze why different mappings of $\hat{f}_S : \mathcal{X} \mapsto \mathcal{Y}$, given the structure of the training data performs better than others. To avoid over-fitting to the training data, the complexity of the classifier is constrained by the choice of α and by regularization through $R : [0, 1] \mapsto [0, \infty)$. The tuning parameter λ controls the cost of adding complexity to the classifier. The optimal value of λ makes the classifier $\hat{f}_S$

ignore the temporary random patterns in the training data, which leads to an increased training error, but minimum proxy out-of-sample prediction error loss. The optimal values of α and λ are thus directly determined by the training data in the tuning process.

3.2 Learnable functional class, $\mathcal{F}$

The functional class, $\mathcal{F}$, describes the types of classifiers that a given algorithm can produce. Using ordinary OLS as an example, the functional class is all affine transformations of y, $\mathcal{F}_{ols} = \{P_y : P \in M_{1\times n}(\boldsymbol{X})\}$ where $M_{1\times n}(\boldsymbol{X})$ denotes all $(1 \times n)$ matrices that can be created by transformations of $\boldsymbol{X}$. The functional class $\mathcal{F}$, will often carry a lot more information in machine learning compared to econometric estimators, as measured by the Vapnik-Chervonenkis dimension, Drucker et al. 1994

Conducting inference in econometrics requires an in- depth analysis of the variance-covariance structure in the feature space. And to obtain an empirical solution for the estimators, this, in turn, requires a much simpler functional class to comprehend ceteris paribus effects. Since machine learning's goal is not to identify ceteris paribus effects, machine learning algorithms allows for a much more flexible feature space, searching for optimal solutions in highly non-linear spaces. Machine learning's superior empirical performance is driven by this fundamental difference. Together with advances in computing power and code efficiency, the available Vapnik-Chervonenkis feature space available for an average laptop is huge.

3.3 Loss Functions, $\mathcal{L}$

Loss functions can be thought of as the optimization objective of the machine learning algorithms and thus is similar to the synonymous econometric concept. In econometrics, OLS is based on the loss of squared residuals. Hence, loss functions are the quantification of the functional classes predictive power. It is what allows the optimization to be conducted.

Let $L : \mathcal{Y} \times \{0; 1\} \rightarrow [0, \infty)$ be a function that maps $(f(x_i), y_i)$ to the non-negative real numbers. It follows that $L(f(\boldsymbol{x_i}), y_i)$ will capture the loss associated with classifying observation i as $f(\boldsymbol{x}_i) = \hat{y}_i$, when the correct classification is y_i. The loss associated with an untrue classification can be computed based on different measures. It depends on how the severity of a wrong classification should affect the real world problem. Should the loss be symmetrical when the loss of a false negative and false positive is applied to the real world problem, that the predictior is trying to

solve? In this case, should the loss be similar for a situation where a loan is approved to a company that defaults within two years and a situation where a loan is not approved however, the company does not default, and the investment opportunity is foregone? One could argue for both a symmetrical loss and a non-symmetrical loss in this real-world application. The thesis will use symmetrical loss functions L on the classification problem for model training. Based on the intended use of the algorithms, non-symmetrical loss would be preferred. The use of non-symmetrical loss functions in the form of objective functions directly implemented in loss function, will be discussed in section 7. Common choices for loss functions include:

$0-1$ loss: $L\left(f\left(x_i\right), y_i\right) = I\left\{f\left(x_i\right) \neq y_i\right\}$, where I denotes the indicator function, which equals 1 if the condition in $I\left\{f\left(x_i\right) \neq y_i\right\}$is true and zero otherwise. Hence, $0-1$ loss will simply count the number of wrong classifications made during the prediction.

Squared error loss: $L\left(f\left(x_i\right), y_i\right) = \left(f\left(x_i\right) - y_i\right)^2$. Squared error loss is used during OLS and is hence well-known in econometrics.

Hinge loss: $L\left(f\left(x_i\right), y_i\right) = \max\left(0, 1 - t \cdot y\right)$, where y is the raw output from the classifiers decision function, not the predicted class. For a linear Support vector classifier, $y = w \cdot x + b$, where (w, b) are the parameters of the hyper-plane and x is the point to classify. The classification is based on t, which takes the values $t = \pm 1$. When t and y have the same sign, meaning that y predicts the right class and $|y| \geq 1$ (point out-side margin), the hinge loss is $L\left(f\left(x_i\right), y_i\right) = 0$. When t and y have the opposite sign, meaning the point is on the wrong side of the hyper-plane, the loss increases linearly with y (one-sided error). When t and y have the same sign but, $|y| < 1$ the prediction is correct but within the margin.

Binary Cross Entropy: $L\left(f\left(x_i\right), y_i\right) = -y_i \log\left(f\left(x_i\right)\right) - \left(1 - y_i\right) \log\left(1 - f\left(x_i\right)\right)$. Cross-Entropy loss is widely used within neural networks. It is used in neural networks because it is differentiable, which is a necessary condition in order to perform back-propagation.

$1-$AUC: Where AUC is defined in section 3.8

Note that different machine- and deep learning algorithms can use different loss functions. The different conceptual frameworks hold different attributes that may make them predisposed for a particular loss function structure. It is, however, possible to compare the performance of the different models based on several metrics. The models are, however ultimately tuned to optimize $1-$AUC, which is the ultimate loss metric that the models will be compared on.

3.4 The bias-variance trade off

In the following, large case letter will indicate stochastic variables, and lower case letters, realizations of that variable. Bold is used for matrices. Let $\boldsymbol{X} = (X_1, X_2, ..., X_n) \in \mathcal{X}$ be a stochastic variable which is used to predict $Y \in \{0, 1\}$. Let $P_{X,Y}$ be the unknown joint probability distribution.

The purpose of this thesis is to find a function f that provides the best out-of-sample default prediction. Here the function f will be dependent on the the sample, S. In order to find the optimal function, f will need to be evaluated out-of-sample. To do this, a loss function $L = L\left(f\left(x\right), y\right)$ is used to quantify the error associated with predicting according to $\hat{y} = f(x)$, given $\hat{y} \neq y$.
Given a loss function, the expected loss associated with a classifier f can be expressed as $\mathcal{L}(f) = \mathbb{E}\left[L(f)\right] = \mathbb{E}\left[L\left(f\left(\boldsymbol{X}\right), Y\right)\right]$, where the expectation parameter is taken over the joint distribution of $(\boldsymbol{X}, Y)$, $P_{\boldsymbol{X},Y}$. Consequently, L is a measure of the predictive quality of a specific classifier, f, where $\mathcal{L}$ measures the performance of the algorithm that produces the classifier. Producing good quality predictions with any machine or deep learning model will therefore be a matter of minimizing the loss associated with the algorithm$\mathcal{L}\left(f\right)$. In classical economics, a specific structure could be imposed on the joint distribution of $(\boldsymbol{X}, Y)$, e.g. a linear dependency that could be simulated by OLS. However, as described before, the advantages that a joint distribution structure provides in regards to modeling and inference is not a requirement in machine and deep learning. Instead, imposing a specific structure on the joint distribution will limit the scope of the possible functional classes $\mathcal{F}$, that could prove to be the most accurate depiction of the data. Hence, the search for the optimal classifier $f^{\star}$, that minimizes the expected loss of the classifier $\mathcal{L}$ can be described as

$$f^{\star} = \arg\min_{f \in \mathcal{F}} \mathcal{L}\left(f\right).$$

In order to determine $f^{\star}$, the joint distribution of $(\boldsymbol{X}, Y)$,$P_{\boldsymbol{X},Y}$ is still required to be known. However, since the true joint distribution is unknown, is not feasible to assume that an optimal classifier, $f^{\star}$ can be found. What is feasible instead, is to use the sample approximation of the expected loss of the classifier $\mathcal{L}$, defined as

$$\hat{\mathcal{L}}\left(f\right) = \frac{1}{N}\sum_{i=1}^{N} L\left(f\left(x_i\right), y_i\right),$$

which in turn can be used to find an estimator for the classifier

$$\hat{f} = \arg\min_{f \in \mathcal{F}} \hat{\mathcal{L}}(f).$$

Using the sample approximation carries its drawbacks however. Specifically, the sample approximation leads to the bias-variance trade-off. In order to understand the bias-variance trade-off, a decomposition of the out-of-sample error is needed. The out-of-sample error using the sample approximation can be decomposed as follows by comparing it to the optimal loss $\mathcal{L}^\star$

$$\underbrace{\mathcal{L}\left(\hat{f}\right)}_{\text{Loss of classifier}} = \underbrace{\mathcal{L}^\star}_{\text{Irreducible error}} + \underbrace{\left[\mathcal{L}(f^\star) - \mathcal{L}^\star\right]}_{\text{Approximation error}} + \underbrace{\mathcal{L}\left(\hat{f}\right) - \mathcal{L}(f^\star)}_{\text{Estimation error}}.$$

Irreducible error: The irreducible error is the error that even the optimal classifier $f^\star$ carries with it. The loss associated with the optimal classifier $L(f^\star)$ can be thought of as random variations in data that can not be captured by any model.

Approximation error: The approximation error captures the error created by restricting the classifier f to $\mathcal{F}$. Hence, in order to minimize the approximation error, the functional class $\mathcal{F}$ must be large enough.

Estimation error : The estimation error captures the direct error that is created by the sample estimated classifier not being the optimal classifier. Hence, the greater the difference between the two, the greater the estimation error. Section 3.6 will investigate the estimation error in finer detail. For now, the estimation error captures the in-sample error that is created by using a sup-optimal functional-class and the unseen over-fit to the sample that will not be visible before applied to the test data. Hence, the estimation error captures three error types, where two of the errors are opposite in effect. The first of the two opposing errors are visible during the in-sample fit, and arise as a consequence of the functional class $\hat{f}$ not being fitted "tight" enough to the sample data. On the other hand, the estimation error that relates to the unseen over-fit is a consequence of the classifier $\hat{f}$ being to tightly fitted during the in-sample estimation, and hence will only show out-of-sample.

The bias variance trade off is the result of trying to minimizing the estimation error $\mathcal{L}\left(\hat{f}\right) - \mathcal{L}(f^\star)$. It follows that when one tries to minimize one of the two opposing errors types, the other will increase. In order to see this directly, I will implement the square-loss function known from

econometrics. The squared loss function can be defined as

$$\begin{aligned}
\mathcal{L}\left(\hat{f}\right) &= \mathbb{E}\left[\left(Y - \hat{f}\left(\boldsymbol{X}\right)\right)\right]^2 \\
&= \mathbb{E}\left[\left(Y - f^\star\left(\boldsymbol{X}\right)\right) - \left(\hat{f}\left(\boldsymbol{X}\right) - f^\star\left(\boldsymbol{X}\right)\right)^2\right] \\
&= \underbrace{\mathbb{E}\left[\left(Y - f^\star\left(\boldsymbol{X}\right)\right)^2\right]}_{\text{Irreducible error (noise) + approximation error}} \\
&+ \underbrace{\mathbb{E}\left[\left(\hat{f}\left(\boldsymbol{X}\right) - f^\star\left(\boldsymbol{X}\right)\right)^2\right]}_{\text{Mean squared error}} - \underbrace{2\mathbb{E}\left(Y - f^\star\left(\boldsymbol{X}\right)\right)\left(\hat{f}\left(\boldsymbol{X}\right) - f^\star\left(\boldsymbol{X}\right)\right)}_{\text{covariance of noise and bias}}.
\end{aligned}$$

The mean squared error term from the prediction $\hat{f}$ can be further decomposed as follows

$$\mathbb{E}\left[\left(\hat{f}\left(\boldsymbol{X}\right) - f^\star\left(\boldsymbol{X}\right)\right)^2\right] = \underbrace{\mathbb{E}\left[\left(\hat{f}\left(\boldsymbol{X}\right) - \mathbb{E}\left[\hat{f}\left(\boldsymbol{X}\right)\right]\right)^2\right]}_{\text{Variance of prediction}} + \underbrace{\left(f^\star\left(x\right) - \mathbb{E}\left[\hat{f}\left(\boldsymbol{X}\right)\right]\right)^2}_{\text{Squared bias}}.$$

It follows from above that, that the prediction loss is increasing in both the variance of the prediction, and the squared bias. Hence, the optimal values for both will be small. The usual unbiased estimator assumption from econometrics[8], implies that unbiased estimation comes from minimizing the in-sample variance. This also implies that the sample covariates are fitted to match the sample predictor in order to achieve an unbiased predictor. Now, since some of the variation in y may be noise, there is a chance that the covariates are fitted to the noise contained in y instead. Thus, when the predictor is applied out-of-sample, the variance will increase due to the over-fit to the in-sample noise. Consequently, since random noise is present in most data, there will exist a trade-off between having bias and variance. This also means that the requirements for unbiased predictors, will rarely be optimal for prediction, since the risk of over-fitting will increase dramatically. Thus, the objective for forecasting in general, is to allow just enough bias in the system so that the learner does not incorporate the signals received from noise into the predictor.

Section 3.6 and 3.7 will discuss theoretical and practical methods to ensure that just the right amount of bias is present. Section 3.8 describes ways to evaluate if too much or too little bias is present in the predictor.

[8] E.g. from minimizing the sum of squared residuals in OLS

3.5 Theoretical bounds of error

This section provides a formal explanation of the sample approximation that was made in the previous section. This section ultimately arrives at a formal explanation of why a large and rich data set is the key to reducing error. However, due to stochasticity and sub-optimal data, a bias-variance trade-off will always be present during prediction modeling. As with the previous section, this formulation builds heavily on Vapnik 1992.

Again, the expected performance of an algorithm can be defined as $\mathcal{L}(f) = \mathbb{E}[L(f)] = \mathbb{E}[L(f(\boldsymbol{X}), Y)]$, where the expectation is applied to the uncertainty of a new observation $(\boldsymbol{X}, Y)$. Similarly, the irreducible error can be defined as $\mathcal{L}^\star = \min_f \mathcal{L}(f)$, with the optimal prediction model for an algorithm as $f^\star = \arg\min_{f \in \mathcal{F}} \mathcal{L}(f)$. In order to better understand the trade-off in selecting the right classifier for the problem, a slightly more theoretical and in depth explanation is needed, compared to the previous section.
Assuming a sample $S = [\{(x_i, y_i) : i = 1, ..., N\}]$ is observed, where the sequence $\{(x_i, y_i)\}_{i=1}^N$ is a series of *iid* realizations of $(\boldsymbol{X}, Y)$, drawn from the joint distribution $P_{\boldsymbol{X},Y}$. Let $f_S() = f(, S)$ denote a classifier based on S. Prior to observing the sample and building the classifier, it is noted that there is two types of stochasticity present. The first relates to the sample S, and the second relates to the new data which will be predicted want, $(\boldsymbol{X}, Y)$. The following contains an explanation of this.

By conditioning on the sample, the expected loss of a prediction becomes

$$\mathcal{L}_S = (f_S) = \mathbb{E}[L(f(\boldsymbol{X}, Y), Y) \mid S].$$

By the law of iterated expectations, $\mathcal{L}_S$ can be related to $\mathcal{L}$ by $\mathbb{E}[\mathcal{L}_S(f_S)] = \mathbb{E}[\mathbb{E}[L(f(\boldsymbol{X}_i, S), Y) \mid S]] = \mathbb{E}[L(f(\boldsymbol{X}, S), Y)] = \mathcal{L}(f_S)$. The expectation is taken both over the distribution of $(\boldsymbol{X}, Y)$ and S, since $\mathcal{L}(f_S)$ does not condition on S. Following the approach in Vapnik 1992, the sample loss on a new observation, $\mathcal{L}_S(f_S)$, given a model trained on a sample is

$$\hat{\mathcal{L}}(f) = \frac{1}{N} \sum_{i=1}^{N} L(f(x_i), y_i),$$

with the classifier model as

$$\hat{f}_S = \arg\min_{f \in \mathcal{F}} \hat{\mathcal{L}}(f).$$

Since an underlying structure of the underlying joint distribution of $(\boldsymbol{X}, Y)$, $P_{\boldsymbol{X},Y}$ has not been imposed, only $\hat{\mathcal{L}}\left(\hat{f}_S\right)$ is observable, since it only relies on the sample data. Based on the approximation of $\hat{\mathcal{L}}(f) \approx \mathcal{L}_S(f_S)$ and $\mathcal{L}_S(f_S) \approx \mathcal{L}(f_S)$, it follows that $\hat{f}_S \to f^\star$ for a large number of sample data N, given the law of large numbers. Similarly, if the functional class $\mathcal{F}$ is large and rich enough, it follows that $\mathcal{L}(f^\star) \to \mathcal{L}^\star$ as $\mathcal{F}$ increases in complexity. Since $\mathcal{L}^\star$ is defined as the irreducible error that is present due to stochasticity, this is obviously non-observable.
Hence in any forecasting situation, the larger the sample size N, and the richer the learnable functional class $\mathcal{F}$, the closer the practical and applied sample approximated classifier $\hat{f}$ will come to the optimal classifier $f^\star$. Many considerations, both in the data collection, cleaning and feature engineering process can help to get closer to the optimal classifier. Also, since different learnable functional classes $\mathcal{F}$ posses different charectiristics, they will differ in prediction performance, when trained on the same data.

3.6 Regularization

This section explores the practical solutions that can be applied to regulate the bias-variance trade-off. Specifically, regularization is performed to reduce the unseen over-fit to the sample that produces out-of-sample error. This section will hence build on the error decomposition made in section 3.3 on the bias-variance trade-off. It was concluded that the unseen over-fit to the sample i related to the Estimation error in the decomposition. The error decomposition made in section 3.3 on the bias-variance trade-off, arrived at the following term.

$$\underbrace{\mathcal{L}\left(\hat{f}_S\right)}_{\text{Loss of classifier}} = \underbrace{\mathcal{L}^\star}_{\text{Irreducible error}} + \underbrace{\left[\mathcal{L}(f^\star) - \mathcal{L}^\star\right]}_{\text{Approximation error}} + \underbrace{\mathcal{L}\left(\hat{f}_S\right) - \mathcal{L}(f^\star)}_{\text{Estimation error}}.$$

Here the sample parameter has been added to the notation to clarify that the classifier has been trained on the sample training data. Based on the discovery made in section 3.3, the estimation error is made up off three parts. Consequently, the estimation error term above can be expanded to showcase these effects. Again, by using the same method in section 3.3, the constituent parts of the estimation error can illustrated by comparing the sample trained classifier $\hat{f}_S$ with the optimal classifier $f^\star$,

$$\underbrace{\mathcal{L}\left(\hat{f}_S\right) - \mathcal{L}(f^\star)}_{\text{Estimation error}} = \underbrace{\hat{\mathcal{L}}\left(\hat{f}_S\right) - \hat{\mathcal{L}}(f^\star)}_{\text{In-sample training error}} + \underbrace{\mathcal{L}\left(\hat{f}_S\right) - \hat{\mathcal{L}}\left(\hat{f}_S\right)}_{\text{Unseen overfit}} + \underbrace{\hat{\mathcal{L}}(f^\star) - \mathcal{L}(f^\star)}_{\text{Random variation}}.$$

In-sample training error : The in sample-training error captures the direct error that is produced when the sample based classifier is evaluated on the training data. If the sample based classifier produces no in-sample training error, any error produced out-of-sample must be due to the test and training data not being *iid*.

Unseen overfit : The second term is a direct product of the realization that the training and test data are almost never *iid*. Hence, any variation in the training-sample that might be important for explaining data, might be irrelevant in predicting the test data. The unseen over-fit is a direct product of minimizing the in-sample training error too much. However, the loss $\mathcal{L}\left(\hat{f}_S\right)$ is not discovered until the classifier $\hat{f}_S$ is applied to the test data, hence it remains unseen until applied to the test data.

Random variation : The random variation term captures the true stochastic behavior that is present. The term builds on the optimal classifier, $f^\star$ and has therefore nothing to do with the training or test error. Hence, the term captures the truly random variation in data.

As stated previously in this section, the goal is to minimize the loss of the classification error $\hat{f}_S = \arg\min_{f\in\mathcal{F}} \hat{\mathcal{L}}(f)$. However, if the loss of the classifier is minimized with respect to the in-sample training error $\hat{\mathcal{L}}\left(\hat{f}_S\right) - \hat{\mathcal{L}}(f^\star)$, it comes with the expense of an increased unseen over-fit in $\mathcal{L}\left(\hat{f}_S\right) - \hat{\mathcal{L}}\left(\hat{f}_S\right)$. The estimated loss based on the sample trained classifier, $\hat{\mathcal{L}}\left(\hat{f}_S\right)$ figures with opposite sign in the in-sample training error term and the unseen over-fit term. Hence, the goal is to constrain the sample based classifier $\hat{f}_S$, so that it does not over-fit to the training data. Creating a non-perfect in-sample classifier by allowing the right kind of in-sample training error, the total estimation error, $\mathcal{L}\left(\hat{f}_S\right) - \mathcal{L}(f^\star)$ will decrease based on a lower unseen over-fit. The process of analyzing which variation in the training data can be classified as temporary movements and which can be classified as inherent markers for this data type, is in many ways a trial-and-error approach. The unseen over-fit is not visible until after the classifier $\hat{f}_S$ has been applied to the test data. Hence, the trial-and-error approach works by changing the classifier to neglect certain patterns of the data and see how it performs on a validation set. Formerly, this can be described as controlling the complexity in the explanatory variables α through $\mathcal{F}$ by varying one parameter in $\mathcal{F}$ and control for any changes in the unseen over-fit. There is however also a more structured manner that this knowledge discovery process on the bias-variance trade-off can be controlled for.

The method is to use a regularization term on the object function for the sample based classifier

$$\hat{f}_S = \arg\min_{f \in \mathcal{F}} \hat{\mathcal{L}}(f) + \underbrace{\lambda \cdot R(f)}_{\text{Regularization term}} .$$

The regularization term directly punishes increased complexity in the classifier by increasing the loss, since the term figures as a positive. The tuning parameter λ controls by how much the regularization term will increase the loss, by adding complexity in $\mathcal{F}$. Intuitively, and with many related concepts from economics and econometrics such as Bayes information criterion, the added complexity in $\mathcal{F}$ must decrease the loss function with more than the regularization term add to the overall loss, before the increased complexity is justified. In this way, the included complexity in the classifier only contains complexity that represents clear and definitive (depending on λ) patterns in data. Consequently, by reducing complexity in the classifier, bias will arise. However, the lower complexity will decrease the risk of unseen over-fit, i.e., out-of-sample variance.

3.7 Tuning

Most of the machine and deep learning models applied to this forecasting problem are prone to over-fitting on the training sample. Since the algorithms learn from the training data, they become masters at predicting based on the training data. However, since there are often considerable differences between the training data and the test data, the algorithms produce significant losses out-of-sample if the model is fitted too hard to the training data. Besides controlling the complexity in α, through $\mathcal{F}$ in the classifier $\hat{f}_S$, by the regularization term, there are several parameters in the algorithms that control the bias-variance trade-off. In the regularization term, λ is a tuning parameter that controls the bias-variance trade-off. The optimal value of λ is the value that produces the lowest error loss on a validation set. In order to mimic the real world forecasting problem, where the test data is still unknown, the tuning process of finding the optimal value of λ must be conducted through $k - fold$ cross-validation. The idea is to partition the training data into several parts and cross-validate the error from training and testing differently on each fold. Specifically, $k - fold$ Cross-validation works by the following.

1. Split the training data into k equally sized folds. This means that $K(i) \in [1, ..., k]$ is the fold where the $i'th$ observation belongs.

2. Leave out one fold, j, and fit the model to the remaining $k-1$ folds, for a specific combination of parameters (α, λ). Let $-K(i) = \{1, ..., k\} / k(i)$ denote all the folds that observation i do

not belong to, and $\hat{f}_S^{-k(i)}$ as the classification for an observation $(y_i, \boldsymbol{x}_i)$, which is based on the $k-1$ folds in which that specific observation does not occur.

3. Calculate the average prediction loss for all the $k-folds$

$$CV_j(\alpha, \lambda) = \frac{K}{n} \sum_{\{i:K(i)=j\}} L\left(\hat{f}_S^{k(i)}\right)$$

4. Then, by calculating the average cross-validation error across all folds $\{1, ..., k\}$, it is possible to arrive at a metric that can be used to compare performance of different values of the tuning parameters

$$CV(\alpha, \lambda) = \frac{1}{k} \sum_{j=1}^{k} \left[\frac{k}{N} \sum_{\{i:K(i)=j\}} L\left(\hat{f}_S^{k(i)}\right) \right] = \frac{1}{N} \sum_{i=1}^{N} L\left(\hat{f}_S^{-k(i)}\right)$$

5. Repeat step (1-4) by a trial and error approach manually with different values of (α, λ) or use commands that execute grid search over the tuning parameters automatically.

6. Choose the optimal parameters $(\alpha^\star, \lambda^\star)$ that gives the lowest cross validation error $(\alpha^\star, \lambda^\star) = \arg\min CV(\alpha, \lambda)$.

Now, since the CV error is an approximation of the out-of-sample loss $\mathcal{L}\left(\hat{f}_S(\cdot \mid \alpha, \lambda)\right)$, the $k-fold$ cross-validation method is not immune to differences in training and test data. However, it is the best proxy metric of the out-of-sample error loss. Hence, minimizing the cross-validation loss should approximately minimize the out-of-sample loss.
An alternative to cross-validation is based on bootstrapping. This is specifically done during the bootstrapped machine learning methods such as random forest and bagging. Based on the bootstrapped sample, a third of the observations are left out for each separate model fitting. Hence, with a similar approach, bootstrapped samples can be cross-validated using the same method as above, but using the out-of-bag samples that were not used in the model fitting. Both methods are identical in that a data partition is used on the training data, to compare different values of (α, λ) on the left out training data.

3.7.1 Choosing the number of folds, k

Cross-validation aims to proxy the out-of-sample error loss with different values of the tuning parameters (α, λ). The difference in using a low (two) and large (ten) number folds lies in the

Table 2: Confusion matrix and origin of loss metrics

	True outcome		
Predicted Outcome	$y_i = 1$	$y_i = 0$	
$y_i = 1$	True Positive (TP)	False Positive (FP)	$precision : \frac{TP}{TP+FP}$
$y_i = 0$	False Negative (FN)	True Negative (TN)	$accuracy : \frac{TP+TN}{TP+FP+TN+FP}$
	$Sensitivity(Recall) = \frac{TP}{TP+FN}$	$Specificity : \frac{FP}{FP+TN}$	

fact that classifiers have access to $\frac{k-1}{k}$ percent of the total training data available. Consequently, increasing the number of folds increases the total share of the training data available for each of the k classifiers. The larger the number of folds the more of the total training data is available for model training, and by the law of large numbers, the closer the sample based classifier $\hat{f}_S$ will get to the optimal classifier $f^\star$. Note also, that the model training will have to run k times as well. Increasing the number of folds k is thereby also associated with longer computational times as well. A common range for k in machine learning literature is usually between $k \in \{5, 10\}$. The number of folds chosen for cross-validation has been set to $k = 5$, as the training set is fairly large and computational time a necessary priority due to the scope of this thesis.

3.8 Measuring predicting performance

Assume the prediction model $\hat{f}_A$ produces some measure of confidence, $\hat{f}_A(x_i)$ that the default prediction made is true $y_i = 1$. For this application, the measure reflect the $default\ risk$. How sure is the model that the default prediction is true? For many models, $\hat{f}_S(x_i)$ will denote the estimated probability $p(y_i = 1 \mid x_i)$. However, these probabilities can be monotonically transformed into classifications by some cut-off point, Murphy 2012. The decision rule for predicted classifications can be represented by a cut-off point τ,

$$\hat{y}_i = 1 \text{ if } \hat{f}_S(x_i) > \tau$$
$$\hat{y}_i = 0 \text{ if } \hat{f}_S(x_i) \leq \tau.$$

Based on the cut-off point τ, observations with a measure of confidence $\hat{f}_A(x_i)$ above the threshold value can be classified as a default, $\hat{y}_i = 1$. Consequently, this will split the predictions into four categories. This is represented in table 2.

Table 2 also contain the three metrics that summarize the specific error types. Sensitivity measures the share of true positives, meaning that the classification has been made towards default, $\hat{f}_S(x_i) = 1$ and the SME actually defaults $y_i = 1$. The true positive rate is the sample based

version of $p\left(\hat{f}_S\left(x_i\right)=1 \mid y=1\right)$, denoting the share of defaults that is made by the prediction model. Likewise, specificity is the share of active companies that actually remains active, which is the sample based version of $p\left(\hat{f}_S\left(x_i\right)=0 \mid y=0\right)$. Precision is the share of the predicted defaulted SMEs that actually goes default, which is the sample based version of $p\left(y=1 \mid \hat{f}_S\left(x_i\right)=1\right)$.

All three metrics (Sensitivity, Specificity, Precision) gives valuable information about the trade-off between the four error types. AUC stands for Area under the Receiver Operating Curve, and it provides a measure that captures sensitivity and specificity across all possible choices of cut-off points,τ. By varying over τ for the spectrum $\tau \in (0,1)$, a series of sensitivities and specificities can be computed and subsequently plotted. The curve created by the mapping of the sensitivity $(y-axis)$ and 1-specificity $(x-axis)$ series, creates the Receiver Operating Characteristics curve (ROC curve). The ideal prediction model would reach the upper left corner with sensitivity$=1$ and $1-$specificity$=0$. Hence, the AUC can be used to compare the performance of different prediction models. However, if the ROC curves intersect, the conclusion that the respective model with the highest AUC value delivers the best result, cannot be made Hand 2009. In such a situation, one will have to do a task-specific analysis, based on the cost of error. Is the loss from not providing a loan to an actual active company the same as the loss taken from giving a loan to a company that defaults? In such a case, the ROC structure will have to be evaluated based on the answer to this question. Specifically, what cut-off point is interesting for this application, and what is the real world objective function that this model is trying to solve? The ROC curves, AUC scores, and task-specific objective functions will be investigated further in section 6 and 7.

4 Selected Machine and Deep learning algorithms

4.1 The benchmark logistic regression model

The logistic regression is the classic econometric approach to a classification problem. It serves as a benchmark with which the machine and deep learning models will be compared to. The logistic regression models is extensively used in the banking industry, since it provides ceteris paribus effects. The coefficients can conveniently be interpreted as log-odds. It is easy to use, and fairly simple to comprehend and communicate. The logistic regression model builds on a linear decision boundary. It does so by modeling the conditional probability (posterior probability in Bayesian

jargon) $P(Y \mid \boldsymbol{X})$.

$$P(Y = 1 \mid \boldsymbol{X} = x) = \frac{\exp\left(w^T x + w_0\right)}{1 + \exp\left(w^T x + w_0\right)}$$
$$P(Y = 0 \mid \boldsymbol{X} = x) = \frac{1}{1 + \exp\left(w^T x + w_0\right)}.$$

By monotone transformation of $p \mapsto \log\left(\frac{p}{1-p}\right)$ on $P_{Y|\boldsymbol{X}}$, the log-odds ratio given x can be formulated as follows

$$\ln\left(\underbrace{\frac{P(Y = 1 \mid \boldsymbol{X} = x)}{P(Y = 0 \mid \boldsymbol{X} = x)}}_{\text{Odds of } Y=1}\right) = w^T x + w_0.$$

Hence, the linear decision boundary in the logistic regression model can be formulated as

$$\left\{x' \mid f(x') = \ln\left(\frac{P(Y = 1 \mid \boldsymbol{X} = x)}{P(Y = 0 \mid \boldsymbol{X} = x')}\right) = 0\right\}.$$

Since $\frac{\partial f(x)}{\partial x} = w^T$, the coefficient w^T can be interpreted as changes in log-odds.

A maximum likelihood approach can be used estimate the coefficients. The maximum likelihood approach is based on the binomial distribution, where the goal is to maximize the log-likelihood. In machine learning terminology, the goal is to maximize the negative loss function.

$$L(\theta) = \prod_{i=1}^{N} P(Y = y_i \mid \boldsymbol{X} = x)^{y_i}$$
$$l(\theta) = \log\left[L(\theta)\right] = \sum_{i=1}^{N} y_i \ln\left[P(Y = y_i \mid \boldsymbol{X} = x)\right] = -\sum_{\mathcal{I}_1} ln\left[1 + exp\left(w^T x + w_0\right)\right].$$

Here, $\mathcal{I}_1$ denotes the first information set, that is the training set. Since the logistic model is trained upon the entire training set, with no regularization, it may over-fit. The logistic model can however be fitted a L_1-norm or an L_2-norm penalty in the loss function, so that regularization is performed and the logistic model becomes the Lasso, Ridge, or elastic-net model. I use the logistic regression model with no regularization. I do not use regularization on the logistic regression model since I want to compare the machine learning models against what is considered classical econometrics. The Lasso model was first theorized by Tibshirani 1996, and does not yet have status as a classical model. Later Zou and Hastie 2005 combined the ridge and lasso regression

to the elastic-net model. The non-regularized model should not be underestimated though. There exist several older credit risk studies, using early machine learning, where the logistic regression model did out-perform all other models, such as Arminger et al. 1997.

4.2 Tree based machine learning models

4.2.1 A single classification tree

Classification trees are at the core of many machine learning algorithms. It is simple to use and very intuitive, and in many cases, provide good predictions. The simplicity and fairly good performance have helped spread the classification tree to all research areas. It is already applied frequently in economics, such as Giannoccaro et al. 2012, which uses the classification tree to map behavioral decisions, based on incentives. The single classification tree is used in this thesis to provide an overview of its forecasting performance, as well as getting early insights on what determines whether a SME will default within two years or not. The single classification tree is also the foundation for the weak learners as well as bagging and random forest. These models use different methods to generate many individual classification trees and gain knowledge based on a "wisdom of the crowd" approach. The single classification tree is very prone to over-fitting to the training data, and this issue, is what the other more advanced methods seek to avoid. It is possible however, to use a regularization term on the single classification tree, to prevent it from over-fitting. This is known as *pruning*, and will be investigated later in this section.
The classification tree works by recursively partitioning the feature space into regions, based on the split that minimizes the loss metric the most. On one side the feature level split, meaning a lower or higher value than the split, the algorithm will classify that side as a default and the other as active. The algorithm starts with the split that reduces the loss function the most. The first split is the root of the tree, and hence the most important variable. The tree is then grown into branches every timer there is a feature level split. The algorithm continues to split the data intro branches until an minimum number of observations is left in each final node. The following formal explanation of the algorithm builds on Friedman et al. 1999.

Let $\mathcal{R} = \{R_j : j = 1, ..., m\}$ denote the m regions that is created by each classification split. R_j contains the explanatory variables x_i that has be partitioned to create region j. The formal definition of the learnable functional class is then

$$\mathcal{F} = \left\{ \sum_{j=1}^{m} p_j \cdot I\left\{x_i \in R_j\right\} \mid \forall_j \in \{1, ..., m\} : p_j \in [0, 1] \right\}.$$

A classification tree is fitted using a greedy manner. The greedy function looks at each split independently of past and future splits. If the algorithm searches for full trees, one at a time, it would result in an almost impossible computational task. Hence, the algorithm works iteratively at each node, independent of the previous and future nodes. The algorithm works as follows, where the subscript t illustrates the sequential nature of the algorithm.

1. Map the full training data
2. Determine the optimal splitting point $s_t \in \mathcal{T}_t$ for each feature x_t. $\mathcal{T}_t$ denotes the sorted feature space from smallest to largest value.
3. Split the node in s_t based on x_t that creates largest decrease in the loss function
4. Repeat (2) and (3) for every node until the stopping criteria is met.

The optimal splitting point $s_t \in \mathcal{T}_t$ partitions the training data, given x_t in two parts $R_1(x_t, s_t) = \{x_i \mid x_{i,t} \leq s_t\}$ and $R_2(x_t, s_t) = \{x_i \mid x_{i,t} > s_t\}$. Finding the optimal splitting points first requires solving for the loss created in the two regions $\hat{p}_1$ and $\hat{p}_2$, created by each feature split. The lowest combined loss for the two groups gives the optimal splitting point for that feature $s_t^\star$, that gives a series of optimal splitting points for each feature $\{x_t, s_t^\star\}_{t=1}^K$,

$$\min_{s_t \in \mathcal{T}_t} \left[\sum_{\mathcal{I}^{R_1(x_t.s_t^\star)}} L(\hat{p}_1, y_i) + \sum_{\mathcal{I}^{R_2(x_t,s_t^\star)}} L(\hat{p}_2, y_i) \right].$$

After all the optimal feature splits and their associated loss have been calculated, the optimal splitting feature that gives the highest reduction in error loss and its associated $s_t^\star$is chosen trough

$$\min_{t \in \{1,\ldots,K\}} \left[\sum_{\mathcal{I}^{R_1(x_t.s_t^\star)}} L(\hat{p}_1, y_i) + \sum_{\mathcal{I}^{R_2(x_t,s_t^\star)}} L(\hat{p}_2, y_i) \right],$$

which decides what feature and its associated split will create the root of the tree. This structure is then repeated until a region is exhausted through a minimal number of observations in each terminal node, or by another stopping criteria.
It is now possible to derive a prediction on the classification based on the region in which the feature level is. In order to evaluate the predictions made in a specific region, a squared error loss

metric can be applied through the following

$$\hat{p}_j = \arg\min_p L_{(f(x)-y)^2}\left(f\left(x_i\right), y_i\right) = \arg\min_p \sum_{\mathcal{I}^{R_j}} \left(y_i - p_j\right)^2$$

$$\iff 0 =$$

$$\iff \hat{p}_j = \frac{1}{N^{R_j}} \sum_i y_i = \frac{N_1^{R_j}}{N^{R_j}},$$

which is the proportion of defaults in that region. An area with at large proportion of defaults can be thought of as a high risk area of default. Observing feature values in this region for any prediction indicates that the SME will default within two years. .

Regularization through tree pruning: The single classification tree is prone to over-fitting. Due to the nature of the algorithm, the data partitioning will continue until all patterns in the data are represented, resolving in a very deep classification tree. It is clear that a fully specified model will have strong in-sample predicting capabilities, however this will result in a large out-of-sample error loss. To combat this issue, a regularization term can be added to the tree growing process described above, and several values of the tuning parameter λ evaluated through cross-validation. Each extra terminal node m_T added carries λ extra loss to the cross-validation error. The tuning parameter will effectively choose the tree size as it determines the number of feature splits. The cross validation error loss based on the regularization term that controls the tree-size, can be described as

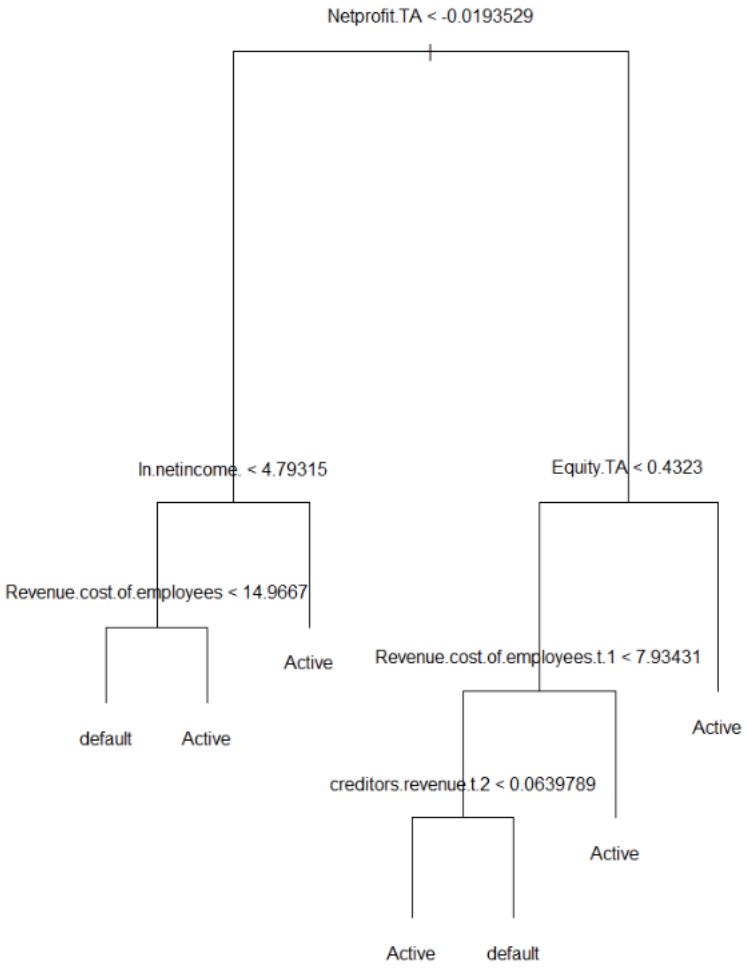
$$CV_{tree} = \sum_{i=1}^{N} L\left(\hat{p}_{jk}, y_i\right) + \lambda \cdot m_T.$$

Given λ is it possible to find the optimal sub-tree $T_\lambda \in \mathcal{T}$, that minimizes CV_λ. This can be described as $T_\lambda = \arg\min_{T\in\mathcal{T}} CV_{tree}$. Practically, the number of terminal nodes m_T is chosen by cross-validation after which the tuning parameter λ is chosen. The cross-validated optimal number of terminal nodes that minimizes $T_\lambda = \arg\min_{T\in\mathcal{T}} CV_{tree}$, is equal to $m_T = 7$. The single classification tree that produces the lowest proxy out-of-sample error loss is illustrated in figure 1.

4.3 Ensemble methods

Ensemble methods build on the single classification tree but seek to reduce bias and variance further by creating an ensemble of classification trees. The ensemble methods reduce the in-sample error and the unseen out-of-sample error by a wisdom of the crowds approach, since it produces

Figure 1: A single trained & pruned classification tree



predictions based on more than one single classification tree. Two methods - bagging and boosting, with different variations in the two will be trained and considered. In the following, bagging and its random forest variation will be formerly explained in section 4.3.1 following gradient boosted trees in section 4.3.2.

4.3.1 Bagging & Random Forest

Bagging is a contraction of bootstrap aggregation. It was first proposed by Breiman 1996. Bootstrapping is a sample increasing technique, that uses a random extraction algorithm to draw observations with replacement from the original sample to create new sub-samples. Aggregation comes from the aggregation of several classifiers $\hat{f}_b$ estimated on each of the sub-samples $b \in [1, ..., B]$ created by bootstrapping. Each of the classifiers $\hat{f}_b$ is trained on the sub-sample, creating a classification tree based on the specific sub-sample. Averaging over all B classifiers produces the final bagging classifier

$$\hat{f}_{BAG}(\boldsymbol{X}) = \frac{1}{B}\sum_{b=1}^{B}\left[\hat{f}_b(\boldsymbol{X})\right].$$

Since the distribution of (x_i, y_i) remains the same across all sub-samples, since it is drawn from the same underlying distribution, the sub-samples are identically distributed. Consequently, the bias of the mean is identical across all sub-samples, and thus the goal is to reduce the variance of the classifier, without changing the bias.

To motivate the random forest model, it is necessary to investigate the variance structure of the bagging classifier. Even though the sub-samples created from bootstrapping are identically distributed, they are not independent. Given a random variable z, the following moments can be described as $\mathbb{E}[z] = \mu$,$\mathbb{V}[z] = \sigma^2$and $\mathbb{E}[(z_i - \mu)(z_j - \mu)] = \rho\sigma^2$ and the variance of the mean over N realizations can be described as $\mathbb{V}(\bar{z}) = \frac{\sigma^2}{N} + \frac{N-1}{N}\rho\sigma^2$. Using L'Hôpital's rule $\lim_{\to\infty}\mathbb{V}(\bar{z}) = \rho\sigma^2$. Hence, by increasing the number of sub-samples the variance of $\bar{z}$, $\left(\frac{\sigma^2}{N}\right)$ decreases, but the correlation term creates a fixed asymptotic variance, independent of the number of sub-samples. Since some correlation will be present among the explanatory variables, the individual classifiers $\hat{f}_b$ will be correlated in their predictions with a lower reduction in variance as a result.
In order to avoid correlated individual classifiers $\hat{f}_b$, the random forest model works by only allowing a randomly chosen subset of the explanatory variables m_{try} to considered at each split in each classification tree. For classification, the recommended number of random variables is $m_{try} = \sqrt{K} = \sqrt{87} \approx 10$, where K is the number of explanatory variables. By not considering the full feature space at each split, the random forest algorithm arrives at a lower correlated group

of classifiers $\hat{f}_b$, as they have different variables in each split across trees. Thus, increasing the number of sub-samples B, the random forest reduces the variance taken as an average across all classification trees, without changing the bias. Now, it is clear that since only a random subset of the explanatory variables are considered at each split, the most important variables could be left out. It is therefore not always true that the random forest model gives better predictions than bagging. The question of whether bagging or random forest is better in a given situation thus depends on the severity in correlation among explanatory variables. Superior prediction performance by the random forest classifier will be expected if there are clear signs that correlation could be an issue. Given that Addo et al. 2018 finds that random forest are among the top-performing models in a similar setting, could indicate, that features derived from financial statement line-items carry correlation. From an intuitive point of view, this makes sense. This issue will be further discussed and analyzed in section 6.

Regularization All the B trees generated from bootstrapping are independently fitted to their respective sub-samples. Thus, as the number of trees increases, it does not lead to over-fitting in the final classifier $\hat{f}_{BAG}$. What does happen though is that after some number of B, additional classifiers $\hat{f}_b$ does not carry any extra information and will thus not change the overall classifier $\hat{f}_{BAG}$ Murphy 2012. Hence, increasing B does not lead to over-fitting, but a too small number of B leads to under-fitting. Using a recommended size $B = 500$ will ensure that more than enough bootstrapped sub-samples exists for model fitting. Besides the number of bootstrapped sub-samples, the most important regularization parameter is the number of random explanatory variables considered at each split, m_{try} in the random forest model. The recommended $m_{try} = \sqrt{K} = \sqrt{87} \approx 10$ is chosen.

4.3.2 Weak learners: Gradient boosted trees

The first weak learner implementation was done by Robert Schapire and Yoav Freund in 1990 through their development of the adaboost algorithm, Schapire, Freund 1990**,** Freund and Schapire 1995. The weak learner applied here is based on a gradient descent optimization algorithm. The algorithm searches for the optimal model parameters to reduce the loss function. It does so sequentially, by updating model parameters slowly. The slow adaptive nature of the algorithm is the reason why it is called a weak learner. The boosting algorithm can be applied to many learnable functional classes $\mathcal{F}$, however, in this case, it builds on the single classification tree. In the following, an intuitive introduction to the model is made, after which the model will be formally introduced.

Boosting works by adding a new learner f_1 to the initial prediction model F_0, $F_1 = f_1 + F_0$, which improves the error of the first model f_1. Likewise, F_1 can be improved by adding another classifier to it. By sequentially increasing the model sequence, the final prediction model looks as follows after m iterations

$$F_m = F_{m-1} + \eta \cdot f_m = F_0 + \eta \sum_{q=1}^{m} f_q. \tag{1}$$

$\eta \in (0, 1)$ is learning rate, which indicate how much the new model should learn from the newly estimated classification tree f_m. It works as a regularization term, to prevent the combined model F_m to learn too much from each classifier f_m - that is to include a model that is trained to tightly to the training data with the risk of unseen over-fit. As a result, the full model F_m will require more iterations, m to reach optimum. However, the use of a learning rate η does usually increase performance out-of-sample, as it would be expected based on the principles of the bias-variance trade-off. Just as, the learning rate control for the bias-variance trade-off, so does the number of iterations m. As m increases, combined classification model F_m learns the patterns of data. If these patterns are repeated too many times, the signal will get amplified by the classification model F_m and hence over-fit to data. Practically, a learning rate η is set ex-ante, and the optimal number of iterations $m^\star$ is the one that minimizes the proxy out-of-sample error through cross-validation, given the learning rate η.

The gradient descent optimization In this section, the gradient descent based boosting algorithm will be formally introduced. The approach follows Friedman et al. 2001. Recall the definition of the optimal prediction algorithm within a learnable functional class

$$f^\star = \min_{f \in \mathcal{F}} \mathcal{L}(f) = \min_{f \in \mathcal{F}} \mathbb{E}\left[L\left(f\left(\boldsymbol{X}\right), Y\right)\right] = \min_{f \in \mathcal{F}} \mathbb{E}_{\boldsymbol{X}}\left[\mathbb{E}_Y\left[L\left(f\left(\boldsymbol{X}\right), Y\right) \mid \boldsymbol{X}\right] \boldsymbol{X}\right]. \tag{2}$$

Boosting works by treating $f(\boldsymbol{X})$ as parameter for each value of $\boldsymbol{X}$, and then minimizing equation 2 over the parameter set $\{f(\boldsymbol{X}) \mid \boldsymbol{X} \in \mathcal{X}\}$, by numerical optimization. This implies estimating f as a sum of parameters $F_m = F_0 + \sum_{q=1}^{m} f_q$. Gradient descent is the numerical optimization algorithm that chooses the next element f_q, in the combined model F_m. The gradient descent problem for addition of a new classifier f_q, can be described as

$$f_q(\boldsymbol{X}) = -d_q(\boldsymbol{X}) \cdot s_q, \tag{3}$$

where $-d_q(\boldsymbol{X})$ is the negative gradient of the expected loss and s_q is the step size to take in the specific direction. Assuming that there exists sufficient regularity to move between differentiation

and integration by Leibniz's rule, the gradient can be formally defined as

$$d_q(\boldsymbol{X}) = \frac{\partial \mathbb{E}_Y\left[L\left(F_{q-1}(\boldsymbol{X}), Y\right) \mid \boldsymbol{X}\right]}{\partial F_{q-1}(\boldsymbol{X})} = \mathbb{E}_Y\left[\frac{\partial L\left(F_{q-1}(\boldsymbol{X}), Y\right)}{\partial F_{q-1}(\boldsymbol{X})} \mid \boldsymbol{X}\right],$$

where $F_{q-1}(\boldsymbol{X})$ is the classification model after $q-1$ iterations. The gradient $-d_q(\boldsymbol{X})$ points in the direction of the maximal decrease in the expected loss function. The step factor s_q that is multiplied by the direction factor $-d_q(\boldsymbol{X})$, tells the optimizer the optimal distance to move in the specific direction in order to move closer to a minimum of the expected loss function. Given the direction $d_q(\boldsymbol{X})$, the optimal classifier f_q is reached by choosing the distance s_q that minimizes the expected loss of the combined classifier,

$$s_q = \arg\min_s \mathcal{L}\left(F_{q-1}(\boldsymbol{X}) + f_q(\boldsymbol{X}), Y\right) = \arg\min_s \mathcal{L}\left(F_{q-1}(\boldsymbol{X}) - d_q(\boldsymbol{X}) \cdot s, Y\right). \tag{4}$$

As can be concluded from the section above: The optimal classifier f_q to add in each step of the sequential model building is the one that closes the residual between the true outcome Y and the prediction of the previous model $F_{q-1}(\boldsymbol{X})$. This can be further explained using sample estimates for the gradient based on training data $\{f(x_i) \mid x_i \in S\}$, which can be described as follows

$$\hat{d}_q(x_i) = \frac{\partial L\left(F_{q-1}(x_i), y_i\right)}{\partial F_{q-1}(x_i)}. \tag{5}$$

where the combined classification function $F_{q-1}(\boldsymbol{X})$ is replaced by the sample estimate $F_{q-1}(x_i)$. Intuitively, $\hat{d}_q(x_i)$is the gradient evaluated on the training data loss function, e.g. the prediction error for SME i in model F_{q-1}. The goal is then to generalize the gradient estimates $\left\{\hat{d}_q(x_i)\right\}_{i=1}^N$ to a gradient direction that will work with any unseen data point, $\left\{\hat{d}_q(\boldsymbol{X}) \mid \boldsymbol{X} \in \mathcal{X}\right\}$. This is turn is another optimization problem. Yes, an optimization problem within an optimization problem. This optimization problem is solved by fitting a prediction model $g_q(\boldsymbol{X})$ to the realized gradients in the training set $\left\{\hat{d}_{q,i}\right\}_{i=1}^N$, using the corresponding covariates for each observation $\{x_i\}_{i=1}^N$, in order to predict the gradient at step q, d_q. Thus, at each sequential model fitting, the newly added models predict the errors of the previous models instead of the outcome directly.
Similarly as equation 4, now given prediction model for the realized gradients, g_q instead of the

gradient, the optimal step size s_q can be defined as

$$s_q = \arg\min_s \hat{\mathcal{L}} \left(F_{q-1}(x_i) + s \cdot g_q(x_i), y_i \right) = \arg\min_s \sum_{i=1}^{N} L \left(F_{q-1}(x_i) + s \cdot g_q(x_i), y_i \right). \tag{6}$$

Combining the prediction model for the realized gradients, the final combined prediction model (without the learning rate η), after m iterations can be defined as

$$F_m(\boldsymbol{X}) = F_0 + \sum_{q=1}^{m} s_q + g_q(\boldsymbol{X}).$$

Specifying the learnable function class to be a classification tree.
Until now the learnable function class has not been specified. Since the gradient boosting algorithm works on top of the single classification tree as learnable function class $\mathcal{F}$, this will now be specified. In this context, the gradient prediction model $g_q(\boldsymbol{X})$ is now the function that decides where the optimal data partitioning will be. Recall that this means the gradient prediction model g_q partitions the covariates $\boldsymbol{X}$ into J regions, see section 4.2.1 for a recap. The full model can be written as

$$\begin{aligned} F_q(\boldsymbol{X}) &= F_{q-1}(\boldsymbol{X}) + s_q \cdot \sum_{j=1}^{J} p_{j,q} \cdot I\{\boldsymbol{X} \in R_{jq}\} \\ &= F_{q-1}(\boldsymbol{X}) + \sum_{j=1}^{J} \gamma_{jq} \cdot I\{\boldsymbol{X} \in R_{jq}\} \\ \gamma_{jq} &= s_g \cdot p_{jq}. \end{aligned}$$

The variable p_{jq} is the predicted gradient in region region j for the q'th iteration.This means that the algorithm is working on a region by region basis, hence a classifier is fitted for each of the J regions. Solving for the the point and iteration specific $\gamma_{jq} = s_g \cdot p_{jq}$, and then aggregating it to all regions of the classification tree γ, gives

$$\begin{aligned} \{\gamma_{jq}\}_{j=1}^{J} &= \arg\min_{\{\gamma_j\}_{j=1}^{J}} \sum_{i=1}^{N} L \left(F_{q-1}(\boldsymbol{X}) + \sum_{j=1}^{J} \gamma_j \cdot I\{\boldsymbol{X} \in R_{jq}\}, y_i \right) \\ \forall j \in \{1, ..., J\} : \gamma_{jq} &= \arg\min_{\gamma} \sum_{\mathcal{I}^{R_{jq}}} L \left(F_{q-1}(\boldsymbol{X}) + \gamma, y_i \right). \end{aligned}$$

Applying a squared loss error metric as in section 4.2.1, the optimal γ_{jq} is the average residual within the partition.

$$\begin{aligned}\gamma_{jq} &= \arg\min_{\gamma} \sum_{\mathcal{I}^{R_{jq}}} L\left(F_{q-1}\left(\boldsymbol{X}\right) + \gamma, y_i\right) \\ &= \arg\min_{\gamma} \sum_{\mathcal{I}^{R_{jq}}} \left[y_i - \left(F_{q-1}\left(\boldsymbol{X}\right) + \gamma\right)\right]^2 \\ \gamma_{jq} &= \frac{1}{N^{R_{jq}}} \sum_{\mathcal{I}^{R_{jq}}} \left[y_i - F_{q-1}\left(\boldsymbol{X}\right)\right].\end{aligned}$$

Conclusively, it can be sen that in every iteration, the gradient boosted classification tree algorithm determines the best partitioning, based on the residuals from the previous model F_{q-1}. During model training, the algorithm adds a constant γ_{jq} for every partitioning, which corresponds to the average residual in that partition.

Regularization in gradient boosted trees The procedure for regularization of the gradient boosted tree algorithm is to find an optimal value of iterations m, given an ex-ante specified learning rate, η. As explained in the beginning of this section, the learning rate regularizes the model directly, by controlling the numerical optimization rate of the sequential boosting process, such that

$$F_q\left(\boldsymbol{X}\right) = F_{q-1}\left(\boldsymbol{X}\right) + \eta \cdot s_q \cdot f\left(\boldsymbol{X}\right).$$

Further, a max interaction depth regularization term has been added to the algorithm. The term constrains the optimizer, such that it does not grow a tree beyond the max interaction depth size. Like the learning rate, the max interaction depth is a direct control for the unseen over-fit. Given the max interaction depth and the learning rate, the *gbm* package runs cross-validation to find the optimal number of iterations m. The optimal number of iterations $m^\star$ is the one that minimizes the cross-validation error. A visual representation of the cross-validation process is provided in Appendix C.

4.4 Support Vector Machines

First, the idea of a maximal margin classifier is introduced. Based on the non-practical implementation of the maximal margin classifier, the soft margin is motivated. This section moves on to explain the linear support vector classifier formally, after which non-linearity in the hyper-plane is introduced through kernels, to create the support vector machine. This section builds on Friedman

et al. 2001 and Boser et al. 1992.

The motivation behind the maximal classifier is to create a decision boundary with a hyper-plane, dividing the p-dimensional input space into two halves. On one side of the hyper-plane, the classification is made towards "active" and on the other side a classification is made towards "default". If the hyper-plane perfectly divides the input space into the correct classifications, the hyper-planer that creates the largest margin between the two classes is known as the maximal margin classifier. In most data sets, however, perfect separation of the classifications in the p-dimensional input space is unachievable. This begs the question of how to create a decision boundary if there is no clear hyper-plane that separates the classes from each other? The idea is to introduce slack, meaning a parameter that controls how many observations that can be wrongly classified. The slack parameter creates a soft-margin on either side of the hyper-plane, to form a band on slack around the hyper-plane. Since many classification tasks, including this one, requires non-linearity in the p-dimensional input space to create an optimal decision boundary, a kernel is introduced in the support vector classifier to introduce non-linearity in the hyper-plane. Specifically, the kernel allows the feature space to interact non-linearly. The non-linear hyper-plane with a soft margin is known as the support vector machine.

The support vector classifier A hyper plane $\mathcal{H}$, can be defined as

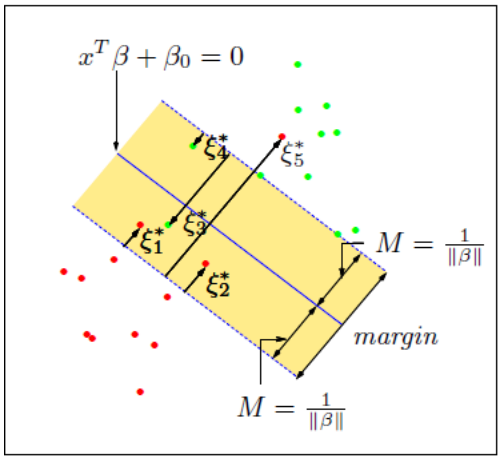
$$\mathcal{H} = \left\{ x \in \mathbb{R}^p \mid g\left(x\right) = x^T \boldsymbol{\beta} + \beta_0 = 0 \right\}, \ ||\beta|| = 1,$$

where β is a unit vector. The classifier $f\left(\boldsymbol{X}\right)$ can be formulated as a product of the hyper-plane, $g\left(\boldsymbol{X}\right)$:

$$f\left(\boldsymbol{X}\right) = \text{sign}\left[\boldsymbol{X}^T \boldsymbol{\beta} + \beta_0\right] = \text{sign}\left[g\left(\boldsymbol{X}\right)\right].$$

The sign parameter specifies that on one side of the hyper-plane the classification will be made towards "active", and the other side of the hyper-plane towards "default". Understanding the practical implementation of this formulation requires a quick insight into the geometry behind the hyper-plane. Specifying the geometry in the p-dimensional space, the hyper-plane can be represented by a normal vector, $v_{\mathcal{H}}$. The projection of x onto the direction of $v_{\mathcal{H}}$ has length $\langle x, v_{\mathcal{H}} \rangle$, with length in units of coordinates as $\frac{\langle x, v_{\mathcal{H}} \rangle}{||v_{\mathcal{H}}||}$. Hence, the distance from any point, x, in the p-dimensional space, to the hyper-plane $\mathcal{H}$, is given by the distance $d\left(x, \mathcal{H}\right) = \frac{\langle x, v_{\mathcal{H}} \rangle}{||v_{\mathcal{H}}||}$. This result serves as the connection between geometry and algebra, since it implies that $\frac{g(x)}{||g'(x)||} = \frac{x^T \boldsymbol{\beta} + \beta_0}{||\boldsymbol{\beta}||}$ measures the distance of the point, to the hyper-plane.

Figure 2: Illustration of the support vector classifier, Friedman et al. 2001



With the basic understanding of how the hyper-plane is created, the optimal placement of the hyper-plane $\mathcal{H}$, and the margin $\mathcal{M}$ can now be analyzed. The problem is a classical problem for economists. Maximize the margin $\mathcal{M}$ i.e. get the clearest decision boundary subject to not exceeding the maximum allowed slack. The slack parameter $\xi = \{\xi_1, ..., \xi_n\}$ allows the individual observation n, to be on the wrong side of the the margin, or the hyper-plane. Specifically, ξ_n measures the distance between the classification specific margin, to the hyper plane. By definition a wrong classification is a prediction on the wrong side of the hyper-plane. The optimization problem becomes

$$\begin{aligned}
&\max_{\boldsymbol{\beta},\beta_0,||\boldsymbol{\beta}||=1} \mathcal{M} \text{ s.t.} \\
&\forall i \in \{1, ..., N\} : \; y_i \left(x_i^T \boldsymbol{\beta} + \beta_0\right) \geq \mathcal{M}\left(1 - \xi_i\right) \\
&\forall i \in \{1..., N\} : \; \xi_i \geq 0 \\
&\text{For } C \geq 0 : \; \sum_{i=1}^{N} \xi_i \leq C.
\end{aligned}$$

By letting $C = 0$ there will be no slack allowed in the optimization $\xi_i = 0 \forall i \in \{1, ..., N\}$, and hence the problem becomes a maximal margin classifier problem. Intuitively, C controls how much total slack $\sum_{i=1}^{N} \xi_i$ that is allowed in the system. Since ξ_i is a distance measure from the margin to point i, $\xi_i = 0$ implies that the i'th observation lies on the margin. When $\xi_i \in (0, 1]$, the observation is between the margin and the hyper plan. For $\xi_i \geq 1$, the observation is wrongly classified, and hence on the wrong side of the hyper-plane. The concepts of the support vector classifier is explained in Figure 2.

The solution to the optimization problem can be solved by convex optimization via Lagrange optimization. The constraint $||\boldsymbol{\beta}|| = 1$ can be replaced by the constraint $y_i \left[x_i^T \boldsymbol{\beta} + \beta_0\right] \geq ||\boldsymbol{\beta}|| \mathcal{M} (1 - \xi_i) \, \forall i \in \{1, ..., N\}$. By using this inequality, the margin can be set to $\mathcal{M} = \frac{1}{||\boldsymbol{\beta}||}$ and the optimization problem can be rewritten to enable convex optimization:

$$\min_{\boldsymbol{\beta}, \beta_0, \xi_i} \left\{ \frac{1}{2} ||\boldsymbol{\beta}||^2 + C \sum_{i=1}^{N} \xi_i \right\} \text{ s.t.}$$
$$\forall i \in \{1, ..., N\} : \; y_i \left(x_i^T \boldsymbol{\beta} + \beta_0\right) \geq 1 - \xi_i$$
$$\forall i \in \{1, ..., N\} : \; \xi_i \geq 0, \tag{7}$$

for some $C > 0$. Since C introduces slack into the system, it can be interpreted as how tightly the support vector classifier is allowed to fit to the training data. As the total slack allowed in the system C, decreases, so does the margin $\mathcal{M}$. The smaller the margin, the higher the risk of unseen over-fit to the training data. Returning to the optimization, the Lagrange multipliers can be expressed as $\lambda_{1,i}$ and $\lambda_{2,i}$ $\forall i \in \{1, ..., N\}$ with the Lagrange primal function being

$$\mathcal{L}_{primal} = \frac{1}{2} ||\boldsymbol{\beta}||^2 + C \sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \left\{ \lambda_{1,I} \left[y_i \left(x_i^T \boldsymbol{\beta} + \beta_0\right) \geq 1 - \xi_i \right] \right\} - \sum_{i=1}^{N} \lambda_{2,i} \xi_i. \tag{8}$$

The first order condition is derived by minimizing equation 8 with respect to the parameters:

$$\frac{\partial \mathcal{L}_{primal}}{\partial \boldsymbol{\beta}} = 0 \Leftrightarrow \boldsymbol{\beta} = \sum_{i=1}^{N} \lambda_{1,i} y_i x_i \tag{9}$$

$$\frac{\partial \mathcal{L}_{primal}}{\partial \beta_0} = 0 \Leftrightarrow \sum_{i=1}^{N} \lambda_{1,i} y_i = 0 \tag{10}$$

$$\frac{\partial \mathcal{L}_{primal}}{\partial \xi_i} = 0 \Leftrightarrow \lambda_{1,i} = C - \lambda_{2,i} \forall i \in \{1, ..., N\} \tag{11}$$

Substituting equation (9-11) into equation 8 and rearranging the terms, gives the Lagrangian dual objective function

$$
\begin{aligned}
\mathcal{L}_{dual} &= \frac{1}{2}\left\|\sum_{i=1}^{N}\lambda_{1,i}y_i x_i\right\|^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\left[\lambda_{1,i}y_i\left(x_i^T\left(\sum_{i'=1}^{N}\left(C-\lambda_{2,i'}\right)y_{i'}x_{i'}\right)+\beta_0\right)\right] \\
&\quad + \sum_{i=1}^{N}\left[\left(C-\lambda_{2,i}\right)\left(1-\xi_i\right)\right] - \sum_{i=1}^{N}\lambda_{2,i}\xi_i \\
&= \frac{1}{2}\left\|\sum_{i=1}^{N}\lambda_{1,i}y_i x_i\right\|^2 - \sum_{i=1}^{N}\left[\lambda_{1,i}y_i\left(x_i^T\left(\sum_{i'=1}^{N}\lambda_{1,i'}y_{i'}x_{i'}\right)+\beta_0\right)\right] + \sum_{i=1}^{N}\lambda_{1,i} \\
&= -\frac{1}{2}\sum_{i=1}^{N}\sum_{i'=1}^{N}\left[\lambda_{1,i}\lambda_{1,i'}y_i y_{i'}x_i^T x_i'\right] - \beta_0\underbrace{\sum_{i=1}^{N}\lambda_{1,i}y_i}_{=0} + \sum_{i=1}^{N}\lambda_{1,i}
\end{aligned}
$$

$$
\mathcal{L}_{dual} = \sum_{i=1}^{N}\lambda_{1,i} - \frac{1}{2}\sum_{i=1}^{N}\sum_{i'=1}^{N}\left[\lambda_{1,i}\lambda_{1,i'}y_i y_{i'}x_i^T x_i'\right]. \tag{12}
$$

Equation 12 forms the lower bound for the objective function in Equation 7, $\min_{\boldsymbol{\beta},\beta_0,\xi_i}\left\{\frac{1}{2}||\boldsymbol{\beta}||^2 + C\sum_{i=1}^{N}\xi_i\right\}$. In addition, the Kuhn-Tucker conditions to the Lagrange problem are as follows

$$
\lambda_{1,i}\left[y_i\left(x^T\boldsymbol{\beta}+\beta_0\right)-\left(1-\xi_i\right)\right] = 0 \tag{13}
$$

$$
\left[y_i\left(x_i^T\boldsymbol{\beta}+\beta_0\right)-\left(1-\xi_i\right)\right] \geq 0 \tag{14}
$$

$$
\lambda_{2,i}\xi_i = 0. \tag{15}
$$

The complete solution to the optimization problem for the Lagrangian primal and dual is then achieved by combining the Lagrangian dual objective function with the first order conditions, along with the Kuhn-Tucker conditions. In the following, the solution to the primal problem is explained. Note that $\boldsymbol{\beta}$ is estimated through $\hat{\boldsymbol{\beta}} = \sum_{i=1}^{N}\lambda_{1,i}y_i x_i$. It follows from equation (13) that $\hat{\lambda}_{1,i}$ is non-zero when the constraint in equation (14) holds. The observations that are characterized by the inequality in equation (14) are known as support vectors, because only $\hat{\boldsymbol{\beta}}$ estimated through these observations. These support vectors have $\xi_i = 0$, meaning they lie right on the edge of the margin. It follows from equation 11 and 15, that $0 \leq \hat{\lambda}_{1,i} < C\ \forall i \in \{1,...,N\}$. Similarly, points with slack $\hat{\xi}_i > 0$ must abide by $\hat{\lambda}_{1,i} = C$ from equation 15. Hence, the $\mathcal{L}_{dual}$ is maximized subject to $0 \leq \lambda_{1,i} \leq C$.

The solution to the dual Lagrangian problem is then achieved by solving for $\hat{\boldsymbol{\beta}}$ and β_0. Without

further ado, the solution gives the support vector classifier's classification rule for a new point

$$\hat{f}(x_0) = \text{sign}\left[\mathrm{x}_0^{\mathrm{T}}\hat{\boldsymbol{\beta}} + \hat{\beta}_0\right].$$

The Support Vector Machine - Adding kernels to the classifier The support vector classifier can only consider linear hyper-planes, which is often sub-optimal in more complex data structures that is characterized by highly non-linear relations. Hence, in order to introduce non-linearity in the hyper-plane $\mathcal{H}$ and margin $\mathcal{M}$, a feature transformation function $\psi(\boldsymbol{X})$ is introduced in Lagrangian dual in order to change the feature space to accommodate non-linearity. Including the feature transformation function in the classifier, it is now formulated as follows $\hat{f}(x_0) = \text{sign}\left[\psi(x_0)^T\hat{\boldsymbol{\beta}} + \beta_0\right]$, where $\left\{\hat{\boldsymbol{\beta}}, \hat{\beta}_0\right\}$is the solution to the new Lagrangian dual

$$\mathcal{L}_{dual} = \sum_{i=1}^{N}\lambda_{1,i} - \frac{1}{2}\sum_{i=1}^{N}\sum_{i'=1}^{N}\left[\lambda_{1,i}\lambda_{1,i'}y_i y_{i'}\left\langle\psi(x_i), \psi(x_{i'})\right\rangle\right]. \tag{16}$$

Instead of specifying the the transformation, the kernel trick is applied to the solution by computing the inner product in equation 16. The kernel can then be expressed as

$$\mathcal{K}(X, X') = \left\langle\psi(X), \psi(X')\right\rangle,$$

where the following polynomial and radial kernel have been applied to the prediction problem:

$$\begin{aligned}\text{Degree } d \text{ polynomial kernel}: \;& \mathcal{K}(X, X') = \left(\alpha + \gamma\left\langle X, X'\right\rangle\right)^d,\ \alpha \geq 0,\ \gamma > 0\\ \text{Radial Kernel}: \;& \mathcal{K}(X, X') = \exp\left\{-\gamma\left\|X - X'\right\|^2\right\},\ \gamma > 0.\end{aligned}$$

The kernel trick works by replacing the inner products of the transformed feature space with the kernel values. This gives the new Lagrangian dual with the kernel trick implemented,

$$\mathcal{L}_{dual} = \sum_{i=1}^{N}\lambda_{1,i} - \frac{1}{2}\sum_{i=1}^{N}\sum_{i'=1}^{N}\left[\lambda_{1,i}\lambda_{1,i'}y_i y_{i'}\mathcal{K}(x_i, x_{i'})\right]. \tag{17}$$

Finally, the classifier for the support vector machine looks as follows

$$\hat{f}(x_0) = \text{sign}\left[\sum_{i=1}^{N}\hat{\lambda}_{1,i}, y_i\mathcal{K}(x_0, x_i) + \hat{\beta}_0\right]$$

Regularization of the Support Vector Machine As with the support vector classifier, the cost parameter C governs the total amount of slack allowed in the system. As the total slack allowed in the system C, decreases, so does the margin $\mathcal{M}$. The smaller the margin, the higher the risk of unseen over-fit to the training data. In bias-variance terms, a small margin might decrease in-sample bias, but risks increasing the out-of-sample variance. The regularization term $\gamma \in \{0, 1\}$, controls how non-linear the decision boundary will be. The more wiggly, meaning high γ, the more tightly the classifier is fitted to the training data. This decreases the in-sample bias, but risks increasing the unseen variance. The optimal regularization parameters $C^\star, \lambda^\star$ is found through a grid search, where an optimization algorithm searches for the lowest cross-validation error based on different values of the tuning parameters. The output from the grid search can be seen in appendix C.

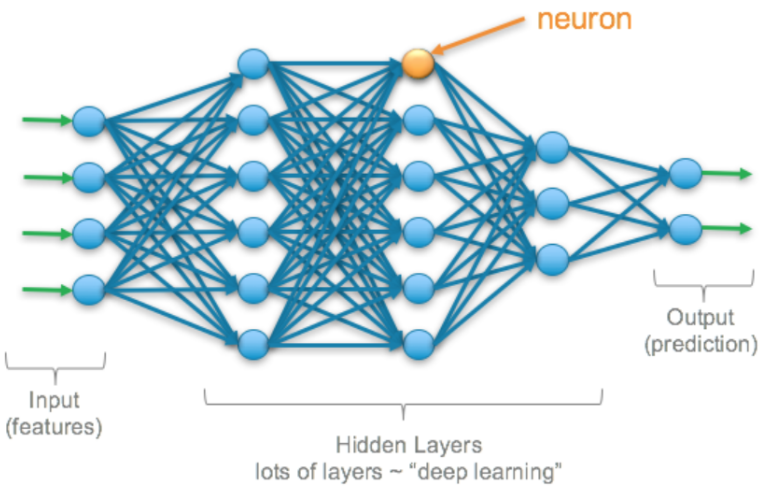
4.5 Deep Artificial Neural Network: The Multi-layer Perceptron

4.5.1 Architecture

Artificial Neural Networks (ANNs) were originally developed as mathematical models of information processing capabilities of biological brains. Although they are not exactly functioning like biological brains, they enjoy increasing popularity as pattern recognition classifiers. The basic structure of an ANN consists of a network of small processing units, joined together by weighted connections. The weighted connections represented the strength of the synapses between the neurons. The network is activated by providing an input to some or all of the input nodes, and this activation of the units spreads through the system through the weighted connections. Figure 3 shows a graphical representation of this process.[9] The units in a multi-layer perceptron (MLP) are arranged in layers, with connections feeding forward from one layer to the next. The input features are fed through the input layer, propagated through the hidden layers in a series of combination and activation functions in each neuron, known collectively as transformation functions. At last, the signal that was processed by the hidden layers reaches the out-put layer, which consists of two units - one for each of the classifications. The strongest signal given to the two output neurons will provide the classification for a given observation. This process is known as the forward pass. Hence, a MLP with a particular set of weight values transforms the input vector of features to an output vector. It has been proven by Hornik et al. 1989 that a MLP with a single hidden

[9]The picture is taken from https://mc.ai/deep-learning-common-architectures/. Although it is not a full representation of the MLP implemented in this thesis, it does have the same three hidden layer structure, as well as two out-put neurons. The input- and hidden layers implemented in this thesis contain a lot more neurons, but with the goal of intuition, this picture serves its purpose.

Figure 3: Representation of Multi-layered Perceptron



layer, containing a sufficient number of units can approximate any continuous function on a compact input domain to arbitrary precision. For this reason, MLPs are said to be universal function approximators. The following section builds on Graves 2012.

4.5.2 Forward pass

Consider a MLP with I input units, activated by an input vector x. Each unit in the first hidden layer calculates a weighted sum of the input units. For a hidden unit, h, in a hidden layer, the network input to that unit h is denoted by α_h, which is known as the combination function. The activation function θ_h is then applied to the network input that the hidden unit h receives through the combination function α_h, yielding the final activation b_h of the unit. Denoting the weight from unit i to unit j as w_{ij}, this information can be presented as

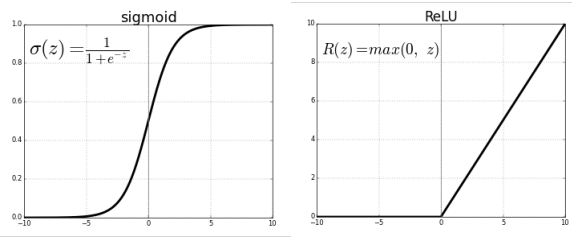
$$\alpha_h = \sum_{i=1}^{I} w_{ih} x_i \tag{18}$$

$$b_h = \theta\left(\alpha_h\right). \tag{19}$$

Several activation functions can be used to process the signal from the combination function. Figure 4 is a graphical representation of the activation functions used in the MLP prediction model for this thesis[10]. Note the non-linearity in both the sigmoid and the ReLU activation function.

[10]The picture is taken from https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

Figure 4: Rectified Linear Unit (ReLU) and Sigmoid Activation function



The non-linearity in the activation function is what allows the MLP to operate in non-linear spaces. Should it be decided only to use linear activation functions throughout the hidden layers, then an argument stating that the MLP would be an unnecessary complication of the logistical regression function, could be made. Hence, the ReLU activation function $R(x) = \max(0, x)$ is used in the hidden layers. The motivation for using a ReLU activation function can be seen in Figure 4. If the signal received in the hidden neuron h, is not strong enough, the neuron will stay inactive. Consequently, it will not send an irrelevant signal on to the next layers. This build-in regularization structure of the activation function, allows the MLP to focus on the strong signals and thus produce predictions of higher quality. The sigmoid activation function $\sigma(x) = \frac{1}{1+e^{-x}}$ is used in the out-put layer exclusively.

Another key property of the two activation functions is that they are differentiable, which allows the network to be trained through gradient descent optimization. Gradient descent optimization will be used to tune the model through back-propagation. This is investigated further in section 4.5.3.. The first order derivatives of the activation functions are

$$\frac{\partial ReLU(x)}{\partial(x)} = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \end{cases} \tag{20}$$

$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x)). \tag{21}$$

The first order derivative is not defined in $x = 0$ for the ReLU activation function. What is practically done in this situation, is that, the first order derivative will be set to zero, $\frac{\partial ReLU(x)}{\partial(x)} = 0$, for $x = 0$. This does not undermine the underlying structure of the ReLU activation function, and hence its qualities as a gatekeeper in the system.

Having calculated the activation function of the first hidden layers, the processes of addition and summation in the combination function, following activation by the activation function, continues for the rest of the hidden layers. The network input received by unit h in the l'th hidden layer H_l

can be represented as

$$\alpha_h = \sum_{h' \in H_{l-1}} w_{h'h} b_{h'}, \tag{22}$$

with that unit's activation denoted as

$$b_h = \theta_h (\alpha_h). \tag{23}$$

Out-put Layer The out-put vector y of a MLP is given by the activation of the units in the out-put layer. The network input α_k to each out-put unit k, is calculated by summing over the units connected to it, as with the hidden unit. The combination function of an out-put unit:

$$\alpha_k = \sum_{h \in H_L} w_{hk} b_h, \tag{24}$$

The activation function used in the out-put layer is the logistical sigmoid activation. Note however, that the logistical sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$ has the same out-put interpretation as the logistical regression function. The logistical sigmoid activation function is used since it allows the out-put to be interpreted as probabilities. Thus, when the feed-forward pass is complete, the MLP assigns probabilities to each of the classifications, represented by the two output neurons. With $K = 2$ classes and out-put neurons, the output neuron that receives the highest probability, will get activated and the classification will be made towards the classification represented by that out-put neuron. Consequently, this MLP is configured to run as a multi-class classification network, for the special case with only two classes. This activation function is also known as a softmax activation function Briddle 1990**,** where the logistic sigmoid with two out-put neurons is a $K = 2$ case of the general softmax activation function**.** The out-put activation using the softmax activation/logistic sigmoid with two out-put neurons, gives the following class probabilities at each out-put neuron

$$p(C_k \mid x) = y_k = \frac{e^{\alpha_k}}{\sum_{k'=1}^{K} e^{\alpha_{k'}}}, \tag{25}$$

which is also known as a multi-nominal logistic model in econometric jargon. C_k represent the correct classification. Note, that for any classification, the sum of the out-put probabilities in the two out-put neurons, equals 1. In order to accommodate the two output neurons, the out-put vector containing the classifications has been transformed using one-hot-encoding, into an out-put matrix of size $N \times K$. One-hot-encoding creates a column vector for each classification, and sets the value in it equals 1 if it belongs to that class, and zero otherwise. The result is an out-put string of $default = \{0, 1\}$ and $active = \{1, 0\}$ attached to each observation, corresponding to the

values created by the two out-put neurons. Using this scheme, the following convenient form of the target probabilities can be formulated as

$$p\left(z \mid x\right) = \prod_{k=1}^{K} y_k^{z_k}, \tag{26}$$

where $z = 0$ if a wrong classification has been made and $z = 1$ if it is a correct classification. Given the above definitions, the use of MLPs for pattern classification is straight forward. Simply feed an input vector to the input layer, activate the network, and choose the class label corresponding to the most active out-put neuron.

4.5.3 Back-propagation

Since the MLP uses differentiable activation and combination functions, they can be trained to minimize prediction error through gradient descent. The gradient descent is the numerical optimization algorithm used for gradient boosted trees as well. In the boosted trees, the gradient of the loss function was evaluated based on different partitions of $\boldsymbol{X}$ to create decision trees. Here, in Neural Networks, the gradient descent optimization algorithm works by finding the gradient of the loss function with respect to different weight values. The weights are then adjusted in the direction of the steepest negative slope. The back-propagation algorithm is used to efficiently calculate the gradients and update the weights, Williams and Zipser 1995. Back-propagation is simply a repeated application of the chain rule for partial derivatives. The first step is to calculate the derivatives of the loss function with respect to the output units. For a multi-class network, which this in effect is, a general loss function $\mathcal{L}$ based on the probabilities created in the two output neurons, can be described as

$$\mathcal{L}\left(x, z\right) = -\sum_{k=1}^{K} z_k \ln y_k. \tag{27}$$

This result is achieved by substituting equation 26 into equation 66 from appendix A. Differentiating equation 27 yields

$$\frac{\partial \mathcal{L}\left(x, z\right)}{\partial y_k} = -\frac{z_k}{y_k}. \tag{28}$$

Remembering that the activation of a unit in the output layer depends on all previous inputs from the network, the chain is applied, yielding

$$\frac{\partial \mathcal{L}\left(x, z\right)}{\partial \alpha_k} = \sum_{k'=1}^{K} \frac{\partial \mathcal{L}\left(x, z\right)}{\partial y_{k'}} \frac{\partial y_{k'}}{\partial \alpha_k}. \tag{29}$$

Remember that the combination function in the output layer is $\alpha_k = \sum_{h \in H_L} w_{hk} b_h$. Hence,$\alpha_k$ denotes the signal that is received by the out-put neuron corresponding to class k, with its respective weightsw_{hk}. Differentiating equation 25 gives

$$\frac{\partial y_{k'}}{\partial \alpha_k} = y_k \delta_{kk'} - y_k y_{k'}. \tag{30}$$

Then, by substituting equation 30 and 28 into equation 29, the partial change in the loss function by changing the signal in the out-put layer through its weights, can be defined as

$$\frac{\partial \mathcal{L}(x, z)}{\partial \alpha_k} = y_k - z_k, \tag{31}$$

where the fact that the total probabilities assigned to the classes must sum to one ,$\sum_{k=1}^{K} z_k = 1$ is used. The chain rule can now be applied continuously**,** working backwards through the hidden layers. At this point it is helpful to introduce the following notation

$$\delta_j \stackrel{\text{def}}{=} \frac{\partial \mathcal{L}(x, z)}{\partial \alpha_j}, \tag{32}$$

where j is any unit in the network. For the units in the last hidden layer, it is true that

$$\delta_h = \frac{\partial \mathcal{L}(x, z)}{\partial b_h} \frac{\partial b_h}{\partial \alpha_h} = \frac{\partial b_h}{\partial \alpha_h} \sum_{k=1}^{K} \frac{\partial \mathcal{L}(x, z)}{\partial \alpha_k} \frac{\partial \alpha_k}{\partial b_h}, \tag{33}$$

where $b_h = \theta_h(\alpha_h)$ is the activation value that the last hidden layer feeds forward to the out-put layer. Differentiating equation 24, equation 19, and then substituting into equation 33 gives

$$\delta_h = \theta'(\alpha_j) \sum_{k=1}^{K} \delta_k w_{hk}. \tag{34}$$

The δ terms for each hidden layer H_l, before the last hidden layer, can then be calculated recursively:

$$\delta_h = \theta'(\alpha_h) \sum_{h \in H_{l+1}} \partial_{h'} w_{hh'}. \tag{35}$$

Once the δ terms for all the hidden units is calculated, equation 18, $\alpha_h = \sum_{i=1}^{I} w_{ih} x_i$ can be used to calculate all the derivatives with respect to each of the network weights:

$$\frac{\partial \mathcal{L}(x,z)}{\partial w_{ij}} = \frac{\partial \mathcal{L}(x,z)}{\partial \alpha_j} \frac{\partial \alpha_j}{\partial w_{ij}} = \delta_j b_i. \tag{36}$$

The process of back-propagation continues until the slope of the gradient stabilizes, meaning no further reduction in loss can be achieved by changing the weight values.

4.5.4 Regularization

I use two types of regularization on the three hidden-layer neural network. The first is related to the back-propagation procedure. I use the ADAM or Adaptive Moments optimizer, developed by Kingma et al. 2014. The ADAM optimizer controls the learning rate. The learning rate has the same interpretation as in boosting but applied here to the weights instead. The learning rate controls how much the weights are updated at each back-propagation iteration. The learning rate is set to 0.001 to prevent over-fitting. The low learning rate also means that the computational time increases. As computations are made to update the weights, the learning rate shrinks through exponential decay, meaning the model will become very stable beyond some point. The exponential decay further limits the risk of over-fitting to training data.

The second method to regularize the MLP is by using drop-out between the layers. For each iteration, a ratio corresponding to the drop-out ratio$\in (0,1)$, will disregard the ratio of neurons in the specific layer. Using drop-out instills randomness into the model, which help to prevent over-fitting further. I use two drop-out layers, one between the input-layer and the first hidden layer, the second between the first hidden layer, and the second hidden layer.

The neural network models are notoriously difficult to tune since they can be customized on a lot of parameters. The number of hidden layers was set to 3, based on the results from literature, Addo et al. 2018 and Zhu 2017. Besides the regularization parameters above, the amount of neurons at each hidden layer is a tuning parameter, as well as the amount of layers, batch size, and epochs. Batch size is the number of observations going through the forward pass at a time. Epochs are the amount of time the entire training data is pushed through the system. Just as with boosting, too many epochs/iterations and the model will over-fit to the training sample.

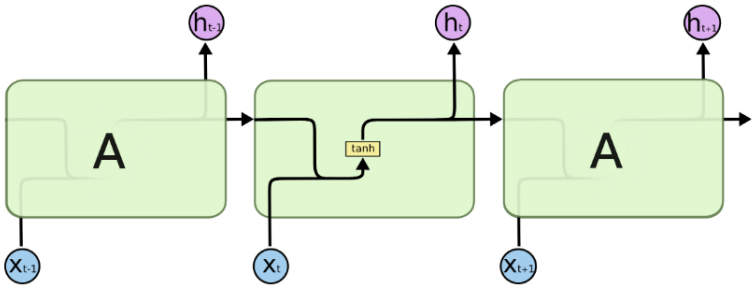
4.6 Deep Recurrent Neural Networks: The Stacked Long Short-Term Memory

The motivation for using a Recurrent Neural Network (RNN) comes from the fact that the data structure is a panel of data, spanning several firms, measuring a lot of variables at different time intervals. Specifically, the three financial statements leading up to the prediction at time t, is available for each company. The previous structures, including the MLP does not understand time. Thus, the data structure that is used for prediction is converted to cross-sectional data without the time-series dimension. However, the single classification tree in figure 1, shows that two lagged variables with time-stamp $t-1$ and $t-2$ are among the top 6 explanatory variables. This, among other intuitive reasons, shows that the past values of explanatory variables, along with the evolution in the level of some explanatory variables compared to others, will help increase the prediction power. Consequently, when working with panel data instead, the feature space is moving from a 2-dimensional input-space to a 3-dimensional input-space. Section 5.4 will go into details of how this is modeled and the coding framework required. The 3-dimensional data structure is modeled through a sequential process that enables the Neural Network to understand time. This structure is known as a Recurrent Neural Network. The RNN structure will be unfolded briefly, as a motivation for the Long Short-Term Memory model.

Long Short Term Memory networks (LSTM) are a special kind of RNNs, capable of learning long-term dependencies. They were introduced by Hochreiter and Schmidhuber 1997, and has since been deployed in a lot of different tasks. Most notably as a Natural-Language-Processor (Speech recognition), used in Amazon's Alexa, Apple's Siri and Google translate. This task is based on classifying a specific response to an input string of words used in conjunction with each other, over a varying time interval between words in the sentences. The LSTM is used for this task, since it can store the short term memory of the first words over a long period of time. Specifically, the LSTM solves the vanishing gradient problem that is inherent to standard Recurrent Neural Networks, Hochreiter and Schmidhuber 1997.

The LSTM works tremendously well on a large variety of problems that have nothing to with language processing but has the same sequential data structure. It is widely used in consumer credit risk and can incorporate many types of data, for example, behavioral data, Zhang et. al. 2017 and Zhu et. al. 2018. The LSTM model shows general superiority when long time horizons are modeled. The prediction of the S&P 500 index in Fischer and Krauss 2018, is a good example. For SME default prediction using LSTM, the literature and applications using panel data are very

Figure 5: Representation of an RNN layer



sparse, but one of them is Yang 2018. As far as I know, the LSTM has never been applied in a SME default prediction task using a panel of SMEs financial statements. Hence, this might be a first for the LSTM model.

4.6.1 Recurrent Neural Network Architecture

All recurrent neural networks have a chain of repeating modules. In standard RNNs, this repeating module will have a very simple structure, such as a single hyperbolic tangent activation layer $\tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$. A note on the hyperbolic tangent activation, or just tanh. The tanh activation is a sigmoidial function, but unlike the sigmoid activation function, the tanh function spans $\tanh(x) \in (-1, 1)$. These properties allow the recurrent network to add or delete memory from the previous unit. These properties are fundamental for the LSTM network, as will become obvious in section 4.6.2. Figure 5[11] represents the feed-forward structure of a simple RNN. The simple RNN feeds a vector of observations x_{t-2}**,** made at time $t-2$, into a unit that processes that signal through a tanh activation function. The signal is then both stored in a memory cell h_{t-2} and used as input at time $t-1$. The unit a time $t-1$ uses both the previous signal, and the observations at time x_{t-1} in the combination function, before the the signal is processed through the tanh activation function. This recurrent structure is what allows the network to understand the time-series dimension of the data. The RNN does however suffer from the vanishing/exploding gradient problem**.** The mathematical proof is not within the scope of this thesis, and I refer to Hochreiter et al. 2001, for an in-depth explanation. Intuitively, the vanishing/exploding gradients problem arises when the influence of a given input to the hidden layer either blows up exponentially or decays completely. The repeated weight multiplications in the recurrent steps between time-points means, that if a recurrent weight has a value lower/larger than one, then the large

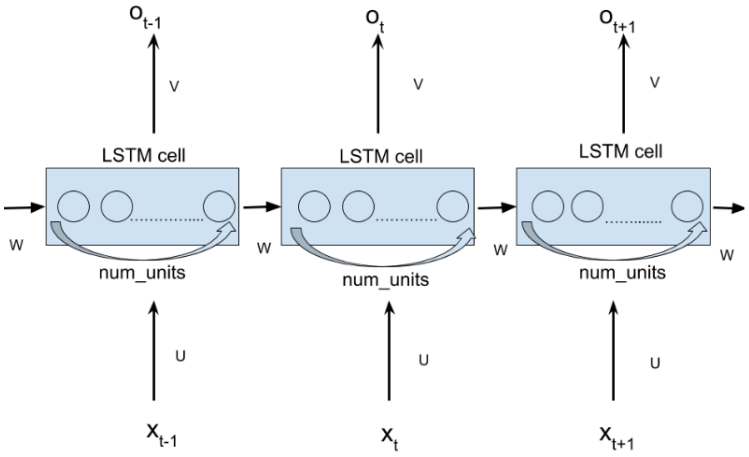
[11] Taken from colah's github page. This article was developed as part of in-house teaching material for Google employees. https://colah.github.io/posts/2015-08-Understanding-LSTMs/#fnref1. I use the 3 time-steps denoted t, $t-1$ and $t-2$. This article also uses three-time steps, but they are denoted differently - hence the mismatch in time notation.

amount of repeated multiplications, decreases/explodes the weight towards zero/∞ over time. As will be unfolded in the following, the LSTM architecture avoids this issue by a diversion of recurrent information through a series of *input*, *output* and $forget$ gates, that have their own separate weights. The following builds on Graves 2012.

4.6.2 LSTM architecture

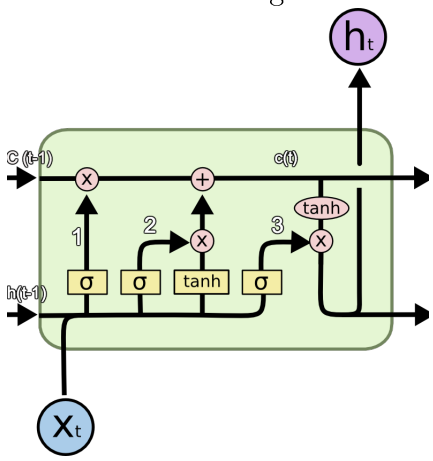
Figure 6 shows an illustration of what a LSTM layer looks like. The layer contains three LSTM cells, one for each time period. Within each cell, a number of units/neurons combines the information through a combination function, process the signal through a tanh activation function and sends the information through the system. This results in two out-puts from each unit/neuron: a memory cell value and unit out-put. The memory cell value controls the long term memory through each LSTM cell, where the unit/neuron out-put controls the short-term memory.

Figure 6: LSTM architecture for one layer



The memory cell has a linear activation function, and is controlled by the forget and input gate. The memory cell controls the long-term memory of the system, spanning the entire LSTM cell. The weights and activation function related to the out-put gate, and the neuron out-put activation controls jointly the short-term memory of the system. The full representation of a LSTM unit/neuron can be seen in figure 7. Each unit corresponds to one of the circles in figure 6.

Figure 7: Representation of a single unit in the LSTM cell



The σ's represent the sigmoid activation function $\sigma(x) = \frac{1}{1+e^{-x}}$, that is attached to both the out-put gate (3), the forget gate (2) and the input gate (1). Since all the gates use a sigmoid activation function, they will give an out-put in the interval $\sigma \in (0, 1)$, corresponding to either a closed gated $\sigma = 0$, or an open gate $\sigma = 1$. The tanh activation $\tanh(x) = \frac{\left(e^{x}-e^{-x}\right)}{\left(e^{x}+e^{-x}\right)}$ in yellow, shows the activation function for the memory cell state $c(t)$ and the tanh activation function shown in red, is the final out-put from the unit/neuron, which is also stored in $h(t)$. Both the cell memory, and the unit out-put is passed on to the next unit, which also receives a new input x.
Based on the activation of the gates, the following in table 3 can be summarized about the long-term memory of the cell state:

Table 3: Cell memory behavior based on gate values

Input gate	forget gate	behavior
0	1	remember the previous value
1	1	add to the previous value
0	0	erase the value
1	0	overwrite the value

If the forget is on and the input gate is off, the memory cell simply computes the identity function, which is a useful default behavior. But the ability to read and write from the previous value is what enables the ability of more complex computations, and hence updates to the long-term memory of the cell.

When implementing an LSTM network, there is obviously a lot of things going on, both in the feature space, but also the time space. The system of equations can be represented in the following,

where $i^{(t)}$ denotes the input gate, $f^{(t)}$ denotes the forget gate, $o^{(t)}$ denotes the out-put gate, $g^{(t)}$ denotes the yellow cell memory activation function. The cell memory is denoted as $c^{(t)}$ and the neuron out-put as $h^{(t)}$.

$$\begin{pmatrix} i^{(t)} \\ f^{(t)} \\ o^{(t)} \\ g^{(t)} \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ tanh \end{pmatrix} W \begin{pmatrix} x^{(t)} \\ h^{(t-1)} \end{pmatrix} \tag{37}$$

$$c^{(t)} = f^{(t)} \circ c^{(t-1)} + i^{(t)} \circ g^{(t)} \tag{38}$$

$$h^{(t)} = o^{(t)} \circ \tanh\left(c^{(t)}\right), \tag{39}$$

where $\circ$ denotes element-wise multiplication (Hadamard product). The multiplication is exemplified in figure 7 by the $\times$ notation. Equation 38 gives the update rule for the memory cell, and equation 39 defines the out-put of the unit/neuron.

4.6.3 Forward pass

As with a standard RNN, the forward pass is calculated for a length T input sequence x, by starting at $t = -2$, recursively applying the update equations while incrementing t. The order at which the forward pass is calculated is important, as can be seen from the unit structure in figure 7. In the following, the standard notation for the combination function α and the activation function b is used. The subscripts $\mathcal{I}$, f, o is used for the input, forget and out-put gate respectively. The values I, K, H and C denotes the number of input variables, output variables, hidden neurons and cell memories respectively. In the following, the ordering of the system equations is important. Thus, the ordering represents how information flows through the units/neurons.
The first step of the forward pass is through the input gate

$$\alpha_{\mathcal{I}}^t = \sum_{i=1}^{I} w_{i\mathcal{I}} x_i^t + \sum_{h=1}^{H} w_{h\mathcal{I}} b_h^{t-1} + \sum_{c=1}^{C} w_{c\mathcal{I}} b_c^{t-1} \tag{40}$$

$$\alpha_{\mathcal{I}}^t = \sigma\left(\alpha_{\mathcal{I}}^t\right). \tag{41}$$

Next comes the forget gate

$$\alpha_f^t = \sum_{i=1}^{I} w_{if} x_i^t + \sum_{h=1}^{H} w_{hf} b_h^{t-1} + \sum_{c=1}^{C} w_{cf} b_c^{t-1} \tag{42}$$

$$b_f^t = \sigma\left(\alpha_f^t\right), \tag{43}$$

and then the memory cells, which depends on the internal values of the input and forget gates

$$\alpha_c^t = \sum_{i=1}^{I} w_{ic} x_i^t + \sum_{h=1}^{H} w_{hc} b_h^{t-1} \tag{44}$$

$$b_c^t = b_f^t b_c^{t-1} + b_{\mathcal{I}}^t \tanh\left(a_c^t\right). \tag{45}$$

The information is then processed through the out-put gate

$$\alpha_o^t = \sum_{i=1}^{I} w_{io} x_i^t \sum_{h=1}^{H} w_{ho} b_h^{t-1} + \sum_{c=1}^{C} w_{co} b_c^{t-1} \tag{46}$$

$$b_o^t = \sigma\left(\alpha_o^t\right), \tag{47}$$

Until the unit/neuron output is calculated using the memory cell and the output gate

$$b_h^t = b_0^t \tanh\left(b_c^t\right). \tag{48}$$

This process continues through all the neurons, LSTM cells and layers.

4.6.4 Back-propagation through time

Back-propagation through time is just as the name indicates, a prorogation through all the layers, where time the time dimension is included. Just as the order of layers is important in back-propagation, so is the order of time in back-propagation through time. The BPTT backward pass is calculated starting at $t = T$ and then recursively calculating the unit derivatives while decrementing t to one Williams and Zipser 1995, Werbos 1990. Like standard back-propagation, BPTT consists of a repeated application of the chain rule. For RNNs, the loss function depends on the activation of the hidden layer, not only through its influence on the output layer, but also

through its influence on the hidden layer, at the next time step. Hence,

$$\delta_h^t = \theta'\left(\alpha_h\right)\left(\sum_{k=1}^{K}\delta_k^t w_{hk} + \sum_{h'=1}^{H}\delta_{h'}^{t+1} w_{hh'}\right), \tag{49}$$

where

$$\delta_j^t \stackrel{\text{def}}{=} \frac{\partial \mathcal{L}}{\partial \alpha_j^t}. \tag{50}$$

The complete sequence of δ terms can be calculated by starting at $t = T$ and recursively applying equation 49**,** decrementing t at each step. Note that $\delta_j^{T+1} = 0 \forall j$, since no error can occur outside the specified time dimension. Bearing in mind that the weights are time invariant, the summation is done over the whole sequence to get the derivatives with respect to the network weights:

$$\frac{\partial \mathcal{L}}{\partial w_{ij}} = \sum_{t=1}^{T}\frac{\partial \mathcal{L}}{\partial \alpha_j^t}\frac{\partial \alpha_j^t}{\partial w_{ij}} = \sum_{t=1}^{T}\delta_j^t b_i^t. \tag{51}$$

The same principle is used in the LSTM model, however each of the three gates carry their own separates weights, as does the memory cell. Since the ordered structure of the forward pass is important, so is the BPTT backward pass. The order follows the inverse order as the forward pass.

$$\epsilon_h \stackrel{\text{def}}{=} \underbrace{\frac{\partial \mathcal{L}}{\partial b_h^t}}_{\text{Loss of neuron out-put}} \qquad \epsilon_c \stackrel{\text{def}}{=} \underbrace{\frac{\partial \mathcal{L}}{\partial b_c^t}}_{\text{Loss of memory cell}} .$$

Neuron output, where subscript g indicates the output activation function

$$\epsilon_h = \sum_{k=1}^{K} w_{ck}\delta_k^t + \sum_{g=1}^{G} w_{cg}\delta^{t+1}. \tag{52}$$

Then follows the output gates

$$\delta_o^t = \sigma'\left(\alpha_o^t\right)\sum_{c=1}^{C} h\left(b_c^t\right)\epsilon_h^t, \tag{53}$$

and then the memory cell states

$$\epsilon_c^t = b_o^t \tanh'\left(b_c^t\right)\epsilon_h + b_f^{t+1}\epsilon_c^{t+1} + w_{c\mathcal{I}}\delta_{\mathcal{I}}^{t+1} + w_{cf}\delta_f^{t+1} + w_{co}\delta_o^t, \tag{54}$$

following the loss from the memory cell

$$\delta_c^t = b_{\mathcal{I}}^t \tanh'\left(\alpha_c^t\right) \epsilon_c^t. \tag{55}$$

The forget gates

$$\delta_f^t = \sigma'\left(\alpha_f^t\right) \sum_{c=1}^{C} b_c^{t-1} \epsilon_c, \tag{56}$$

and at last the input gates

$$\delta_{\mathcal{I}}^t = \sigma'\left(\alpha_{\mathcal{I}}^t\right) \sum_{c=1}^{C} \tanh\left(\alpha_c^t\right) \epsilon_c^t. \tag{57}$$

For multiple-layered deep networks, the process is repeated through all layers.

Regularization Regularization was conducted using the same methods as the MLP model. See section 4.5.4 for a recap.

5 Data

5.1 Data collecting process

The data is extracted from the Orbis database. The Orbis database collects data from more than 160 national and regional databases. Orbis stores financial data from companies for up to five years. Only yearly financial data is available for private companies. All 5 years are used during the data collection process. The oldest three yearly financial statements are used for explanatory data, and the two newest financial statements are used to ensure that the companies remain active for the forecasting horizon. Consequently, the data set is structured as panel data, with specific financial statement line-items, taken from a large number of individual private companies, measured over three time periods. The data extraction procedure of raw financials was made as follows.

1. Include only non-financial companies
2. Include companies that are active and companies that fall into the category of default.
3. Include only companies that have an operation revenue between DKK 10 M and DKK 500 M.
4. Include only companies that have a minimum of 10 and a maximum of 350 employees.

5. Include only companies in Europe

6. Ensure that a minimum of 5 yearly financial statements is available for active companies and at least 3 for companies that default, with an explicit default date available within the two year forecasting period.

7. Include only companies that have values for the other 18 financial statement line-items.

Since the Orbis database only keeps financial statements for a limited amount of time, and a typical search yielded 20,000 active companies and 200 default companies, older records were used as well to increase the sample size. The imbalanced structure of the two classifications creates another problem in itself, which will be addressed later in this section. The University Library store older data extractions on hard-disks, these were used as well - to increase the sample size for the default minority class. Financial statements as old as 2009 was used to increase the sample size for default companies. The total time-period included spans from 2009 to 2018. The active companies related to the forecasting periods 2017-2018 and 2015-2016 were included in the sample with a total of 41,062 companies classified as active. To clarify what is meant with a forecasting period e.g. 2015-2016, it means that the three previous financial statements 2012, 2013, 2014 is used to forecast a default or non-default for the next two financial periods, spanning the fiscal year 2015 and 2016. The sample includes the forecasting periods 2013-2014, 2014-2015, 2015-2016, 2016-2017, and 2017-2018 for the defaulted companies. In total this yielded 1,244 companies that are classified as a default. The raw sample size created from the data extraction procedure amounts to 42,306 observations. With a total of 20 line-items from the financial statement, measured over three time-periods, the total explanatory data amounts to $42306 \cdot 20 \cdot 3 = 2,538,360$ observations. Due to the private nature of the SMEs selected, they do not publish very detailed accounts of their financial statements. Hence, the 20 variables represent the full amount of information in a given fiscal year, that could be relevant for this thesis.

After the raw data is collected, feature engineering was applied to create 29 explanatory variables for each fiscal year. Including the three time periods for which there is data, the resulting number of explanatory variables related to each classification is 87. Table 4 shows the 29 explanatory features. The feature engineering was done according to literature such as Addo et. al. 2018 and Zhu 2017. The explanatory data set is based on a mixture of ratios and raw financial inputs that relates both to operational metrics and financial health metrics.

A default is defined as a company defaulting on payments, entering insolvency proceedings, entering liquidation, declared bankrupt by a court or completely dissolved due to liquidation. These are standard definitions of what constitutes a default, based on legal definitions.

Table 4: Feature Engineering

Feature	Variable Name
EBITDA/Total Assets	EBITDA.TA
EBITDA/Equity	EBITDA.Equity
Equity/Total Assets	Equity.TA
Equity/Fixed Assets	Equity.FA
Equity/Liabilities	Equity.Liabilities
Long Term Liabilities/Total Assets	LTL.TA
Total Liabilities/Total Assets	Liabilities.TA
Long Term Liabilities/Fixed Assets	LTL.FA
Cashflow/Equity	Cashflow.Equity
Cashflow/Total Assets	Cashflow.TA
Cashflow/Revenue	Cashflow.Revenue
Grossprofit/Revenue	Grossprofit.Revenue
Netprofit/Revenue	Netprofit.Revenue
Netprofit/Total Assets	Netprofit.TA
Netprofit/Equity	Netprofit.Equity
Netprofit/Number of Employees	Netprofit.Employee
Cashflow/Short Term Liabilities	Cashflow.STL
Cashflow/Long Term Liabilities	Cashflow.LTL
Current Assets/Short Term Liabilities	Current.ratio
EBITDA/Financial Expenses	EBITDA.Financial.expenses
Revenue/Cost of Employees	Revenue.Cost.of.employee
Revenue/Cost of Goods Sold	Revenue.Cost.of.goods.sold
Debtors/Revenue	Debtors.Revenue
Creditors/Revenue	Creditors.Revenue
ln($Revenue$)	ln.Revenue
ln($Total\ Assets$)	ln.TA
ln($Long\ Term\ Liabilities$)	ln.LTL
ln($Short\ Term\ Liabilities$)	ln.STL
ln($Netprofit$)	ln.Netincome

5.1.1 Forecast horizon

A forecasting horizon of two years, based on three years of financial data leading up to the forecasting period, is identical to the real world problem of deciding whether or not to provide a loan running for 24 months to a SME, based on the previous three years of financial history. Pompe and Bilderbeek 2005 show that the financial year leading up to the default should not be used as a stand-alone measure for default prediction. Hence, the financial data for the three years leading up to a default is used together. Funding Circle requires at least two years worth of financial

history[12], and the credit risk models created by TRIBrating uses two years of financial information as well[13]. I use three years since this is the maximum amount of information available.

5.2 Data preprocessing

The real-world data is generally noisy, incomplete and inconsistent. Many factors affect the success of using machine learning methods on a given problem, but the quality of the data may be the most important one, Kotsiantis et al 2006. For example, if the data is redundant, noisy, irrelevant or unreliable, a machine learning model may not find patterns during the training process. Besides a theoretical understanding of the fundamental drivers behind the classification, a good quality data preprocessing scheme will help to minimize these problems. Data preprocessing includes data cleaning, transformation, normalization, and feature selection among the most frequently used. The following subsection will discuss choices made in the data preprocessing.

5.2.1 Preliminary descriptive analysis

The preliminary descriptive analysis showed two groups of firms that were clearly outliers. The first was quite interesting from a personal standpoint, and that is mid-level English football clubs, such as Huddersfield Town football club. Based on the financial statements from this group, they should have been declared bankrupt a long time ago. Football clubs, along with rental firms in the real estate business were excluded from the sample. Rental firms with real estate assets were excluded for two reasons. A business closure due to a full liquidation of assets results in a default classification. A liquidation event can occur either voluntarily or by legal measures brought by creditors. The preliminary analysis showed that the liquidation of real estate in the rental business was based on voluntary liquidation, since the underlying financial health, of what in effect are investment firms, were solid. Due to this discovery, all rental businesses were excluded to avoid any voluntary liquidation to be classified as default. Besides, the target group is non-financial SMEs and football clubs and real estate rental businesses do not represent the target group for this thesis.

5.2.2 Missing Values

Missing values of the raw data extraction were avoided through only including firms who have data available for all financial line-items, for all relevant years. The issue of missing values did

[12]Funding Circle homepage
[13]Euler Hermes Rating GmbH "Rating Validation Study, 27 September 2018"

arise during the feature engineering process. The two features with the only noticeable amount of missing values is $EBITDA/FinancialExpenses$ and $\ln(Netprofit)$. Missing values, which in these cases are created by non-defined ranges, occurs due to a company not having any financial expenses (divide by zero) or a company had a negative profit for that year (log negative). Negative net-profit has a frequency of around 20% and zero financial expenses $< 1\%$. The missing value scheme implemented is a zero-fill scheme, which replaces missing values with zero. One has to be careful when implementing a scheme for missing values since it will introduce bias to the data Garcia et al. 2015. A argument for a zero-fill scheme being the optimal solution for this prediction problem, is that it introduces a low amount of bias into the system while preserving observations.

5.2.3 Feature selection and correlation heat-map

Irrelevant features create unnecessary noise to the system. By producing a correlation heat-map, highly correlated features can be found. During the initial trial-and-error feature engineering process, the correlation heat-map was used to discard features that were highly correlated with other variables.

5.2.4 Normalization

Within a single feature, there is often large differences between the minimum and maximum value. Normalization is a feature transformation method, that scales down the values within a feature to a narrower range of values, Kotsiantis et al. 2006. This is either done through a min-max normalization or a standardization procedure towards a normal distribution. This thesis uses the min-max normalization, which is described as

$$x_{norm} = \frac{x_i - x_{i,min}}{x_{i,max} - x_{i,min}}.$$

The min-max feature is used when the learner works through distance measures, such as $k-nearest$ neighbor and support vector machine. These methods calculate the Euclidean distance to the nearest neighbor and to the margin respectively. Normalized features give a cleaner and faster computation of distance measures. Besides this argument, a normalization of the feature space makes sense when the features represent different scales, such as log and fractions. All models except tree-based learners and the logistic regression model uses scaled features.

5.3 Solutions to an imbalanced data-set

The data contains $\frac{1242}{40923} \approx 3\%$ default classifications. This significant imbalance creates problems during model training if the model is trained on the imbalanced data. Since there are so few cases of defaults in the system, the algorithms will neglect the patterns that these observation show during model training. Thus, when the trained model is presented with unseen test data, it will classify all observations as non-default, and achieve a 3% error rate. This is not a desirable out-come since the minority class, default, is the most interesting and important classification to learn and correctly classify. In order to allow the learning process to be applied to the patterns that represent a default, the training data needs to be balanced. There are several ways in which this can be done. The applied methods will be discussed in the following.

5.3.1 Under-sampling

Under-sampling works by truncating the majority class, such that the number of majority classes equals the number of minority classes. However, since there are only a total of 1242 default samples, and under a 50/50 training-test data split, it results in 621 cases of each classification to learn from. Although this is not a catastrophically low training sample, the big data dimension will get lost during such a scheme, and inadvertently loss of general application to another test data. Section 5.1 mentions how the data collecting period for the active companies only happens over two forecasting periods, compared to five for the default observations. Hence, under-sampling was already performed during the data-collection phase, as an imbalanced data set was known to be a problem.

5.3.2 Over-sampling

Over-sampling works by a continuous replication of the minority classifications. This can be done as many times as desired. It does however lead to over-fitting, since the same patterns are repeated during training. Thus, a test default observation that does not show the characteristics of the over-sampled training data, will not be correctly classified. From Section 5.1, it can likewise be seen that over-sampling is done through two ways in the data collection period. First, there exist default classified companies that figure both in the forecasting period 2014-2015 and 2015-2016. If a company defaulted in 2015, it will figure in both sub-samples, meaning they will have two overlapping time-points in feature space (FY2012, FY2013). However, since they both have different feature values at the most important time period - the financial statement leading up to the forecast period, the over-sampling does not introduce a large amount of bias, although some bias is.

Second, the data collecting period for the active companies happens over two forecasting periods, compared to five for the default observations, meaning this can also be seen as an over-sampling in the time dimension. These measures were taken during the data collecting phase to ensure a large number of defaults and the largest possible variation between them available for model training. Although this is not ideal, it was considered a necessary measure.

Most studies using imbalanced data applies a combination of the random under-sampling and random over-sampling methods. As mentioned, under-sampling means that active classifications will have to be discarded. Over-sampling implies that the models will loose its generalization to the test set, exemplified through a high variance. The synthetic minority over-sampling methods described in the following seeks to circumvent these two problems.

5.3.3 SMOTE

The Synthetic Minority Over-Sampling Technique (SMOTE) Chawla et al. 2002 tackles the imbalanced data set problem, by synthetically creating new features using a $k - nearest$ neighbor method. The SMOTE algorithm works sequentially by looking at a random real minority classification. The feature space is then copied, however, each feature is recalculated based an average from the $k - nearest$ neighbors of the real feature value. The process is repeated over the entire feature space, to create a synthetic minority observation, based on the $k - nearest$ neighbors. Before the creation of a new synthetic feature based on the average of the $k - nearest$ neighbors, a random neighbor value is selected and scaled randomly within a range to induce randomness and variation in the synthetic samples. Thus, no SMOTE generated feature, will be identical.
The $k - nearest$ neighbor algorithm works by calculating the distance, in this case, the Euclidean distance $\|x_j - x_i\|$. As with any application of the Euclidean distance, it is important to normalize the data before applying the $k - nearest$ neighbor algorithm. Similarly, any application of a $k - nearest$ neighbors algorithm uses k as a tuning parameter. The lower the amount of neighbors, k, the higher the risk of over-fitting to the training sample, and vice versa. In this case, a high number of neighbors decreases the scope of variation in the minority class feature space during training. Thus, k, controls the scope and density of variation in the feature space, upon which the pattern learning takes place. As with any tuning parameter, the optimal value of k is chosen through cross-validation. Cross-validation gave $k = 3$ a favorable out-come over the standard $k = 5$ application of SMOTE. Consequently, $k = 3$ is applied in the SMOTE algorithm. The relatively low value of k indicates that the feature space for the default classification contains a lot of variation, and hence must be represented clearly through a low k.

One downside with using the $k - nearest$ neighbor algorithm is the curse of dimensionality. It is computationally heavy to compute $k - nearest$ neighbor averages, since all the k neighbors will get stored over the entire calculation of the balanced data set. Since Rstudio stores the data necessary for these computations using RAM, there exist a dimensional boundary due to hardware limitations. The SMOTE algorithm is used to calculate the largest feasible balanced data set, subject to RAM limitations. The result is a balanced training data set, containing 9563 active classifications and 9315 default classifications. The total balanced training set contains $(9563 + 9315) \cdot 87 = 1,642,386$ feature observations. Hence, the balanced SMOTE training set is created from both a synthetic over-sampling of the minority class, a and random under-sampling of the majority class.

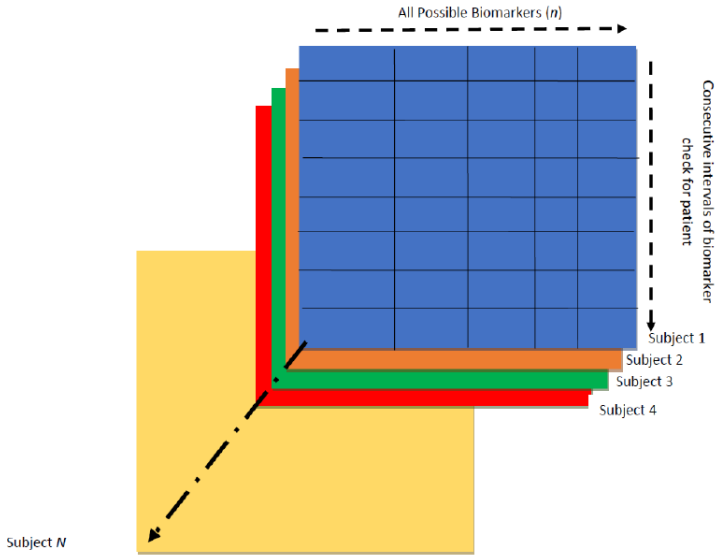
5.3.4 Tomek-Link

A Tomek-link is a pair of nearest neighbors which belongs to different classifications while being each other's nearest neighbors, Kubat et. al 1997. Having many Tomek-links in a training-sample creates a more unclear decision boundary. This problem is combated in the SVM through the introduction of slack. However, more can be done in the data preprocessing phase to create a smoother decision boundary, without losing important information. The TOMEK algorithm works by measuring the Euclidean distances in the feature space and searches for the existence of Tomek-links. The Tomek-links are then broken by removing the observation that belongs to the majority classification. Both observations can be removed, however, this application of the TOMEK algorithm only removes the majority class. This is done in order to preserve both the number of default observations and the density of default observations that is situated close to the decision boundary.

5.3.5 SMOTE + TOMEK

The imbalanced data set problem is combated by using a SMOTE+TOMEK approach. First, the data is balanced through synthetic minority over-sampling and regular under-sampling of the majority class. Again, random under-sampling was done due to RAM limitations. After SMOTE was performed to balance the training data, the Tomek-Links are broken using the TOMEK algorithm, which creates a smoother decision boundary by removing the majority classification of a Tomek-link. More 2016 shows that SMOTE + TOMEK is one of the best, if not the best method to combat imbalanced training data. The final training data - after the SMOTE + TOMEK approach is applied, results in 9090 observations which are classified as active, and 9315 observations which are classified as default. The total amount of feature observations in the training sample are

Figure 8: Representation of Tensor data for the LSTM using panel data



$(9090 + 9315) \cdot 87 = 1,601,235.$

5.4 Software and data preparation for the LSTM

The machine learning models are build and applied using R with the RStudio interface. The main packages used are show in table 5. The code for the deep neural networks is written in Python using the Keras code-base**,** which I run on top of the Tensorflow API, developed by Google. I access the TensorFlow back-end from Anaconda. I use the Tensorflow API since it allows for the modulation of tensors, meaning multidimensional data spaces. The ability to work with multidimensional data spaces is fundamental for the application of the LSTM model using panel data. In order for the LSTM model to understand time, the model is fed a $3-dimensional$ tensor with dimensions $N \times T \times \left(\frac{K}{T}\right)$. Each observation is presented by a $2-dimensional$ tensor with dimensions $T \times \left(\frac{K}{T}\right) = 3 \times 29$. The pattern recognition problem now becomes an image recognition task, where each company is represented by a 87 pixel frame. Each line in the frame represents the features measured at a specific point in time. The Tensor data can be represented as shown in figure 8, taken from Rekik et. al. 2018. Here, the bio-markers are instead features, which are engineered from line-items in a SMEs financial statement. Similarly, instead of sampling when the patients visit the doctor, the data is sampled for each financial year.

Table 5: Overview of R packages used

Algorithm	Package name
Subset split	caTools
SMOTE	DMwR
TOMEK	UBL
Logistic regression	caret
ROC	ROCR
AUC	pROC
Classification tree	tree
Bagging/Random Forest	randomForest
Boosting	gbm
SVM	e1071
Plots	ggplot2

6 Results

6.1 Model Performance

In the following, the model performance of the individual models will be evaluated. Table 6 shows the main results of the thesis.

There is no single over-all best model, instead the gradient boosted tree along with random forest are jointly the best overall predictors. They out-perform all other algorithms with a superior and identical AUC score of 0.82, and an identical confidence interval of $0.81 - 0.83$. The related bagging model does, however, come in close with an AUC score of 0.805. The superiority of the gradient boosting and random forest algorithm is in line with similar studies, Addo et. al. 2018, Zhu et al. 2017. An AUC score of 0.82 is a very good result since the forecasting model does not contain any other explanatory features than what can be deducted from their financial statements. However, the result is far from perfect, and it does reflect the difficulty in predicting two time periods ahead. The current financial situation may be good, but the next year's financial result may be catastrophic, which could result in a default 19 months in. Likewise, the current financial situation for an SME may be bad, but orders may be in the pipeline, and the bank willing to extend credit until the situation returns to normal. Hence, the forecasting period of two years brings considerable uncertainty to the forecasting problem. In Section 6.2, the results of the gradient boosting and the random forest model are described in detail. The random forest model performs marginally better than the bagging model, suggesting that some of the features might contain irrelevance or correlation cf. section 4.3.1.. The irrelevance of features can be exemplified through the $t-test$ of

Table 6: Model Performance measured in Area Under ROC-curve

Model	AUC	Confidence interval
Logistic	0.76	$0.74 - 0.78$
Classification Tree	0.72	$0.70 - 0.74$
Bagging	0.805	$0.79 - 0.82$
Random Forest	0.82	$0.80 - 0.83$
Gradient Boosting	0.82	$0.80 - 0.83$
Support Vector Machines	0.62	$0.60 - 0.64$
Multi-layer Perceptron	0.76	$0.74 - 0.78$
Long Short-Term Memory	0.78	$0.76 - 0.80$

the coefficients from the logistic model, which will be addressed later in this section. See Appendix B for a full overview of the significant variables. The correlation among features will be addressed later in this section as well.

The Long Short-Term Memory Model performs decently with an AUC score of 0.78. The MLP is not far behind witn an AUC score of 0.76. Addo et. al. 2018 and Zhu et al. 2017, find a three hidden layer ANN model to be the best configuration using this type of data. That result motivated the use of a three hidden layered MLP and a three-layered LSTM framework, used for this prediction task. I do find that any other configuration than this performs worse. The results show that the deep neural networks should not be discarded when forecasting credit default, however they need to be sufficiently complex to be a serious contender to the gradient boosting algorithm and random forest

Third, logistic regression performs very well with an AUC score of 0.76. The strong score of the logistic regression model is not far behind the best models. What is also noticeable is the relatively low difference in performance among the models, despite the models being vastly different. It is noteworthy, that logistic regression is not far behind the most complex model - the LSTM. The reason behind the strong logistic score might lie in the extensive feature engineering conducted in the preprocessing phase.

Fourth, The Support Vector Machine performs very poorly with an AUC score of 0.62. Both the polynomial and radial kernel was considered as suggested by Min and Lee 2005. Even though, the SVMs were tuned on a very large grid of parameter values[14], similar to Min and Lee 2005, they did not improve more than the 3rd order polynomial included here. It might be, that the removal

[14]The full grid-search can be seen in appendix

of Tomek-links, removes the data points that would otherwise have been used for support vectors. This is empirically backed by Sain and Purnami 2015 and Elhassan and Aljurf 2017. Both find that when the SMOTE+TOMEK method is used, and the minority class has less than 10% of total observations, the SMOTE+TOMEK method will perform worse than only using SMOTE. I did tune the model on a SMOTE only training data, and the SMOTE Support Vector Machine did perform better. I kept the model trained on SMOTE+TOMEK because it is an interesting result, and I believe that future researchers within this area need to be aware of this problem.

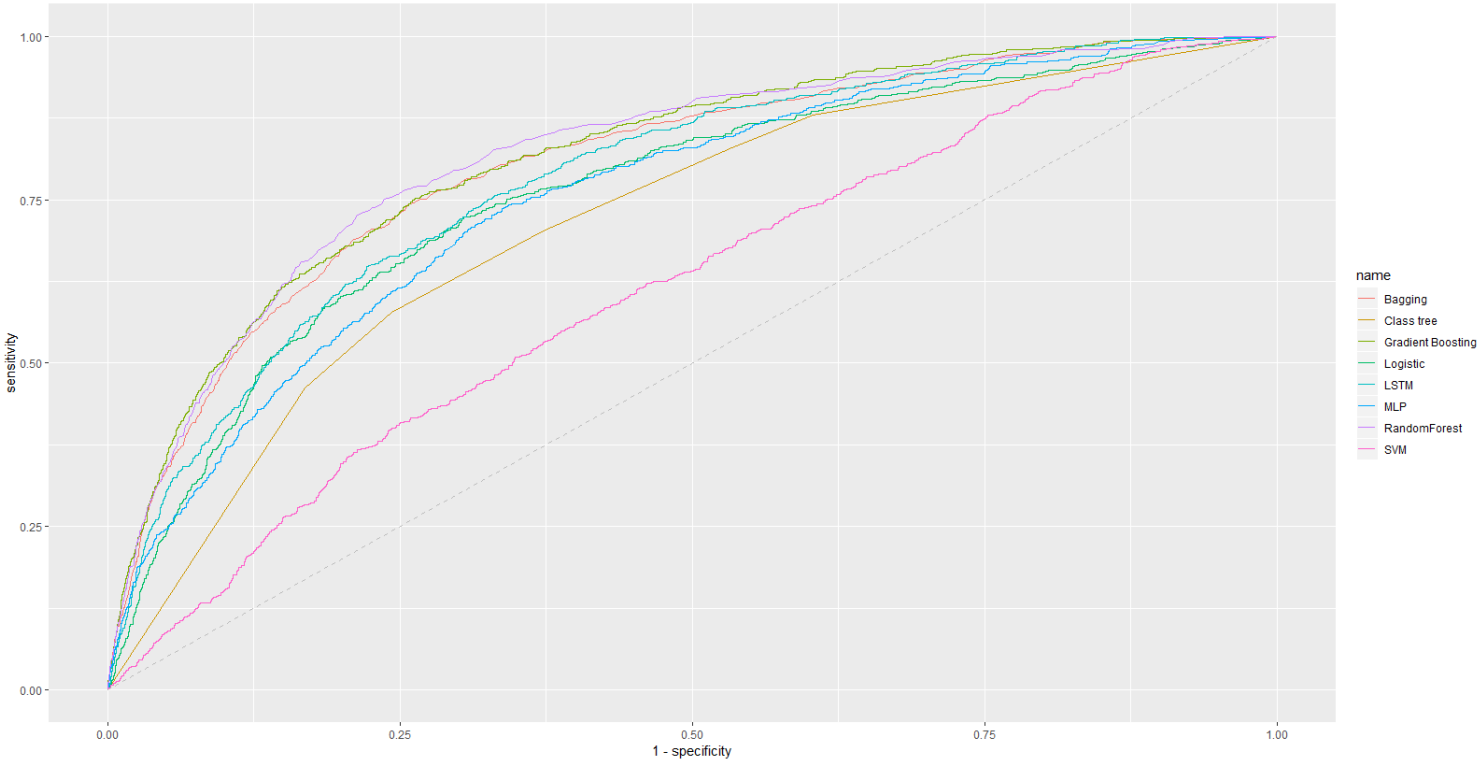
Although all the tree-based methods improved the AUC score compared to the single classification tree, the single classification tree performed fairly well, not far behind its more advanced extensions. What is particularly interesting is the simplicity and intuition that it brings to the forecasting problem. If the AUC score should be compared to how easy it is to explain to non-machine learning professionals, the single classification tree would be a top score.

Table 6 shows the AUC scores achieved for each model. The confidence interval is calculated using 1000 bootstrapped samples. Based on the confidence intervals, I conclude that gradient boosting, bagging and random forest are the three best models.

6.1.1 Performance illustrated in ROC

The ROC-curve plots the different combinations of sensitivity(recall)= $\frac{TP}{TP+FN}$, meaning the share of correct default predictions, relative to all the default predictions. Sensitivity is also called the false negative rate and is shown on the $y-axis$. The $x-axis$ shows $1-specificity$ also known as the false positive rate $\frac{FP}{TP+FP}$. The false positive rate denotes the share of active SMEs that has been wrongly predicted to be a default, as a share of all default predictions. If a ROC curve passes through the point (0.5, 0.80), it means that the model allows the trade-off of having correctly identified 4 in 5 active companies, at the cost of incorrectly classifying 1 in 2 active SMEs as defaults. The ROC curves show the trade-off in trying to raise the 4 in 5 correct default predictions to a higher rate. If the desired rate is 9 in 10, then the error rate for classifying active SMEs wrongly as defaults, increases with it. This means that in order to secure that only 1/10 approved loans default, the investor will have to accept that a lower fraction of good investments are actually realized. This is the trade-off that the ROC curves represent.

Figure 9: ROC-curves



Hence, what the investor would be interested in, is a prediction model that has a sensitivity close to 1, with the lowest amount of 1-specificity as possible. Meaning, the investor will maximize the amount of good investments made, given that the amount of loans that default should be ~zero. Based on the ROC curves, it looks like this happens at the point $(0.8, \sim 1)$. Further, the boosting, LSTM and random forest models looks to achieve a sensitivity close to one "first", although there is a very small difference. Based on this, and the great difference in conceptual frameworks between the LSTM and boosting algorithm, I will compare the LSTM, boosting and logistic regression model in further analysis. This is motivated below.

Remember, section 3.8 points out that using AUC values to evaluate the best mode, only applies when the ROC curves do not intersect. If a ROC curve does intersect with another, then a given trade-off between the false positive rate and the false negative rate will be better for another model, over a series of cut-off points, τ. Thus, before a true winner is declared, there must first be an analysis of the optimal cut-off point. To do this, the models will be applied to a real-world problem through a peer-to-peer investment platform, with a task-specific loss function. Based on the loss function, a true winner can be determined. This will be analyzed in section 7. For now,

the best model will be the one with the highest AUC score. Consequently, computing the measures for F1-score, false positive rate, false negative rate, accuracy and similar performance measures for a given cut-off point will be irrelevant - until the optimal cut-off point and loss function is specified further.

6.2 The superior models (Random Forest & Boosting)

6.2.1 Variable importance

Although machine learning does not provide the same ceteris paribus effects as logistic regression, it is possible to retrieve a variable importance measure for both gradient boosting and random forest. The variable importance measure is less informative than the measures obtained from the logistic regression model since they only show the increase in predictive power by adding a variable. The variable importance measure does not show which classification each feature helps predict, or by how much - it just shows the increase in predictive power of the model as a whole. Machine learning methods are hence often described as black-box when it comes to deriving ceteris paribus effects. The single classification tree does, however, increase the ability to make some kind of inference.

Formally, the variable importance measure is defined as follows from Friedman et al. 2001. The importance of a variable X_j in a specific tree T_t is denoted $VI_j(T_t)$. The variable importance measure for a single tree can calculated as follows

$$VI\left(T_t\right) = \sum_{k=1}^{J} \hat{i}_k \cdot \underbrace{I\left(s\left(k\right) = X_j\right)}_{\text{Only splits where variable j is used}}$$

, where $\hat{i}_k$ measures the prediction error reduction in node k. J is the number of terminal nodes and $s\left(k\right)$ is the split used at node k. Hence, the total variable importance measure is just the sum, taken over $n.trees$ and B for the gradient boosting and the random forest respectively, denoted by m. The total variable importance measure for variable j is defined as $VI_j = \sum_{t=1}^{m} VI_j\left(T_t\right)$, meaning the sum in the reduction of prediction loss, for every variable, across all nodes and trees. The variable importance measure for the gradient boosted model is shown in table 7. A few interesting deductions can be made from the variable importance measure. This is presented in the following.

Table 7: Top 10 variable importance scores for gradient boosting

Variable	Importance
log(Net-profit)	6.11
Liabilities/Total Assets	6.04
Revenue/Cost of Employees t-2	4.1
Revenue/Cost of Employees	4.0
Cash-flow/Short Term Liabilities	3.8
Net-profit/Number of Employees	3.1
Creditors/Revenue	3.1
Equity/Total Assets	2.9
EBITDA/Financial Expenses	2.6
Creditors/Revenue t-2	2.2

The two most important variables are $\ln(Netprofit)$ and Liabilities/Total Assets. Liabilities/Total Assets measures the capital structure of the firms. Since assets are either financed through debt or equity, a small value of Liabilities/Total Assets indicates that there is room for extra debt financing. Hence, should the firm move into liquidity problems, the SME would have room to increase debt. The fact that$\ln(Netprofit)$ is the most important variable should not surprise any. Looking isolated at profit can however be deceiving, since a small Liabilities/Total Assets ratio could indicate that a hypothetical non-profitability could be financed by debt until the SME returns to profitability. Remember from the data preprocessing section that a zero-fill scheme is used for missing/undefined features. This means that SMEs with negative profit will have a $\ln(Netprofit) = 0$. The strong importance shows that the variable still remains important even though the feature has been augmented. Thus, I assume with respect to importance, that the zero-fill scheme did not alter the underlying structure of $\ln(Netprofit)$ as a predictor for default probability.

Second, the lagged value of Revenue/Cost of Employees$(t-2)$, has the third largest variable importance measure. This is quite surprising, but it shows that the past profitability and margins delivered has some kind of effect on the system as a whole. It could be that the past performance has a direct effect on structural variables such as Liabilities/Total Assets, several periods into the future. The highest correlated variable with Revenue/Cost of Employees$(t-2)$, is $\ln(Long-Term\ Liabilities)$ with $corr = -0.13$. The higher the revenue to cost of employees ratio, the lower long term debt, and vice versa. This supports the correlation hypothesis.
The value of Revenue/Cost of Employees could also be a proxy for the sector(12.5%) and business profile(17.5%) weights used by TRIBrating, i.e. how is the margin in the given sector, or how good

is the firm at delivering margins, given the sector. Naturally, low margin sectors and low margin business will be more predisposed to default risk.

The third important factor is based on the SME's ability to meet its short term debt obligations, both measured as a total and as its obligations to suppliers. The Creditors/Revenue ratio measures how important trade finance is for the financing of day-to-day operations. Although the creditors line-item only provides a snapshot of the account, at the date of the financial report, it can be used as a proxy for a yearly average. If a SME has low liquidity, it might be forced to finance its operations through trade credit, provided by suppliers at a premium and top-up with short term debt financing. If the Cashflow/Short Term Liabilities ratio gets too small, the SME will not be able to keep its operations running, should the bank close its line of credit. The fact that the lagged variable of Creditors/Revenue($t-2$), is the 10th most important feature is interesting as well. It might have the same effect on the system as the lagged Revenue/Cost of employees. In order to escape a low liquidity situation, the SME might be forced to increase both short and long-term debt, which will affect the Liabilities/Total Assets ratio several periods into the future, and ultimately affect the probability of default.
It is also noticeable that no $t-1$ feature is present among the top ten most important variables. The reason for this might be that the features at $t=0$ directly explains the performance at $t-1$, and the derived effects from past features take time to show up in the system. Also, the latest features at $t=0$ are the most important.

The EBITDA/Financial expenses has a variable importance score of 2.6. The interpretation is straight forward. If the ratio is too small, the SME uses a large share of EBITDA on debt service. The higher the required amount of EBITDA necessary to service debt, the higher the risk of default should the EBITDA for a given year, not be enough to meet the debt service payments. If the measure is low, it indicates that a debt increase is not an option, and that the SME has to divert considerable resources to debt service instead of day-to-day operations. Thus, the feature both reflect the liquidity situation, as well as a proxy for deciding whether their bank will be willing to increase debt.

One should notice how the variables are selected at each split in the random forest model, and the boosting model, given that they perform the best. Only a random sub-sample corresponding to $m.try$ =10 variables are considered at each split for the random forest model. This means that if a variable such as Creditors/Revenue($t-2$) is correlated with a very important variable such as Liabilities/Total Assets, the variable importance score of Creditors/Revenue($t-2$) will get affected.

If Liabilities/Total Assets is not selected and a correlated variable like Creditors/Revenue$(t-2)$ is, then the hidden information that Creditors/Revenue$(t-2)$ carries on Liabilities/Total Assets will increase the importance of Creditors/Revenue$(t-2)$.
The gradient boosting algorithm works through the greedy selection of the best variable at each split. If Creditors/Revenue$(t-2)$ is selected at a split, the algorithm works by selecting the next variable that explains the most, given the all-ready selected variables. If Creditors/Revenue$(t-2)$ and Liabilities/Total Assets is correlated, then Liabilities/Total Assets will not get selected in the next split, since some of its variation is all ready explained by Creditors/Revenue$(t-2)$.
Further, the two models only consider a sub-sample of the training data at each tree fitting. This is known as out-of-bag in random forest and $bag.fraction$ in gradient boosting. The random hold-out of data at each sequential tree fitting introduces randomness to the model training. This further prevents over-fitting to noise created by correlation among the explanatory variables.
The fact that the two best algorithms are the best at handling correlation indicates that there still is some correlation present among the explanatory variables. In general, when correlation is present there is an increased risk for the unrestricted model to over-fit during the training process. Hence, finding the right tuning parameters is especially important in such a situation.

6.2.2 Tuning process

The tuning process for the gradient boosted tree was conducted using the *gbm* package in *R*. The tuning parameters were found using a plug-and-play solution that keeps the $bag.fraction = 0.5$ and the learning rate, $shrinkage = 0.1$, along with the minimum number of observations in terminal nodes, $n.minobsinnode = 10$ at their default values, and then running cross-validation on the optimal number of iterations *n.tree*. The max tree-size that the algorithm was allowed to grow was set to $interaction.depth = 6$. This choice was made based on the single pruned classification tree which grew an optimal size tree to 6. A grid search strategy can be applied manually, however the preliminary trial-and-error tuning showed no improvement when the model took on non-default values. Hence, the tuning process was conducted based on holding the above parameters fixed and running $5-fold$ cross-validation to find the optimal value for the amount of iterations made, *n.tree*. The graphical illustration of the cross-validation procedure can be seen in Appendix C. The illustration shows that the cross-validation error is a convex function of iterations, *n.tree*. By finding the minimum of the cross-validation function, the optimal number of iterations $m^{\star}$ is found, given a learning rate $\eta = 0.1$.

6.3 Simple classification models

For some applications, an intuitive and explainable model may be preferred to a complex model, even if it has lower performance. The logistic model, along with the single classification tree is the most intuitive and simple applications. Logistic regression is used extensively in the industry, and all credit analyst will be familiar with it. The classification tree is lesser known, but it is very intuitive and can be applied through a check-list approach, even if the professional applying it may not know how the model is generated.

Figure 1 shows the best pruned single classification tree. The tree scores an AUC of 0.72 while only addressing 6 out of 87 features. The 6 variables used for the split are Netprofit/Total Assets, Equity/ Total Assets, Revenue/ Cost of Employees $t-1$, Creditors/Revenue $t-2$, $\ln(Netprofit)$ and Revenue/Cost of Employees. Despite only calculating the 6 features and comparing them to a value, the single classification tree still produces a decent partition of the active and default companies.
In short, the simple yes/no model answers four basic questions. a) does the company make a profit? b) is the revenue to salary larger than some threshold now and last year? c) Is the majority part of the capital structure debt? and d) Was day-to-day operations financed through short-term debt larger than some threshold two years ago? By answering these questions related to the six partitions, it is possible to get a classification for any SME, made by any person. This ability does have a value that is not reflected in the AUC score.

Logistic regression achieved better than the single pruned classification tree, with an AUC score of 0.76. The obvious benefit of using logistic regression is that inference can be conducted under further assumptions. Around half of the logistic regression model's coefficients are statistically significant. When the coefficients are ranked based on their $t-statistics$, a "proxy" variable importance measure for the logistic regression model can be derived as well. The logistic regression model ranks approximately the same variables, as the single classification tree, as the most important. It is also evident, that the same group of variables, as used in the single classification tree, are statistically significant for almost all time periods. An overview of the significant variables can be found in Appendix B.

This could indicate that a relatively low amount of variables, in a simple and intuitive prediction model, could be used, to asses credit default risk with a fairly good result. This is an important result for the general application of this thesis, by professionals, who are not trained in machine

learning.

6.4 Potential ways to increase model performance

Although the chosen models show good results, there are a few ways to improve the models further. First, the balanced training sample could be increased. As the number of observations increase, the model should be able to learn more patterns in data, including the ones that are currently underrepresented. Due to the curse of dimensionality that is connected to the $k-nearest$ neighbor method in SMOTE+TOMEK, the balanced data set in constrained on RAM limitations.

Although the Orbis database contains a vast amount of information, the pool of information relating to private SMEs was completely emptied during the data collection process. Access to behavioral, and qualitative data in general could greatly increase the prediction model. Also, access to data, sampled at a higher frequency could be especially valuable in a LSTM model.

Raw data could be used instead of extensive feature engineering. Modern and complex neural networks such as LSTM and convolutional networks, develop their own internal feature dependencies structure. Hence, they are less in need of extensive feature engineering in the data preprocessing part, compared to other models. A study on performance for raw vs. engineered training data could be interesting.

A note on the LSTM model The fact that the LSTM model did not show superior performance even though it is by far the most complex model, can be explained in several ways. First, as explained, the LSTM model is not as dependent on extensive data preprocessing, as other models. Since extensive feature engineering was conducted, the advantage from complexity is eliminated. Second, the ability to handle panel data might not be as important when only three time periods are considered. A similar prediction problem with the use of monthly data could give the LSTM model an advantage over the other models. A particularly interesting application could be for short-term debt contracts in the area of supply chain finance, e.g. a loan running for 3 months.

7 Market implementation

Since the ROC curves intersect, it is not possible to definitively conclude which model is the best, for the task at hand. Before a winner can be determined, the models need to be related to a real-world problem. This involves specifying a new custom loss function describing the goal of

the application. The application could be a peer-to-peer lending platform like Funding Circle. In order to build an investment decision and pricing model, the point of departure is the methodology used by TRIBrating, a professional SME ratings agency. With the use of their default probability to credit rating conversion (Table 1), the individual SMEs can be assigned a credit score. This methodology is known as risk based pricing, formalized by Oliver 2001 and Keeney and Oliver 2003. Based on the credit score, an optimal price of risk can be assigned to each risk profile. Using the pricing structures of Funding Circle, the profitability of the different models will be examined. In the following, I will conceptualize an economic framework to evaluate the models using an objective function based on structures from industry.

7.1 Choosing the optimal cut-off point

The default probability to credit score conversion is based on TRIBrating's own model, trained on French SMEs. The data used in this thesis builds on a broad European collection of SMEs, and large country heterogeneity exists on the SME credit market. Thus, the credit conversion made in Table 8 is an approximation to the French SME credit market. I have created the ranges by extending the average values given in table 1 equally to either side. It can be deducted from table 8 that a SME with a predicted default probability higher 14%, will be deemed too risky for credit investment, with rare exceptions. Hence, what will determine which model is the best application in a Funding Circle like setting, will be the one that performs best with a cut-off point of $\tau = 0.14$. In general, when the cut-off point for the classification rule decreases, the amount of false negatives decreases, but the amount of false positive increases. Here, the false negatives are SMEs that are predicted as active but turns out to default. The false negatives are thus an investment that turns out to default. False positives are SMEs that are classified as default, but remain active during the forecast period. The false positives are synonymous with missed investment opportunities. Given this trade-off under the cut-off point $\tau = 0.14$, the best model will be the one that produces the highest amount of good investments. Hence, the best model will be the one with the highest amount of true negatives, which is synonymous with the lowest amount of false positives. However, the objective function also includes a cost parameter based on the false negatives, meaning investments that turn out to default. This custom loss function thus directly incorporates the profit function and is based on the non-symmetrical loss between a false positive and a false negative. Table 8 plots the false negative prediction rate for the logistic, LSTM and gradient boosting model for the different credit scores.

Table 8: Predicted probability of default to credit rating conversion

Credit Rating	Predicted Probability of default	Logistic: FN/(FN+TN)	LSTM: FN/(FN+TN)	Boosting: (FN/FN+TN)	Best Model
AA or higher	$0 - 0.01\%$	$4/609 = 0.6\%$	$14/4031 = 0.35\%$	$23/5914 = 0.39\%$	LSTM
A	$0.01 - 0.5\%$	$3/588 = 0.51\%$	$53/5672 = 0.93\%$	$92/7422 = 1.24\%$	Logistic
BBB	$0.5 - 1\%$	$2/245 = 0.82\%$	$18/819 = 2.2\%$	$29/1186 = 2.45\%$	Logistic
BB	$1 - 3\%$	$5/620 = 0.81\%$	$20/1327 = 1.51\%$	$51/1711 = 2.99\%$	Logistic
B	$3 - 7\%$	$7/910 = 0.77\%$	$21/889 = 2.36\%$	$32/1094 = 2.93\%$	Logistic
CCC or lower	$7 - 14\%$	$18/1617 = 1.11\%$	$18/700 = 2.57\%$	$43/813 = 5.29\%$	Logistic

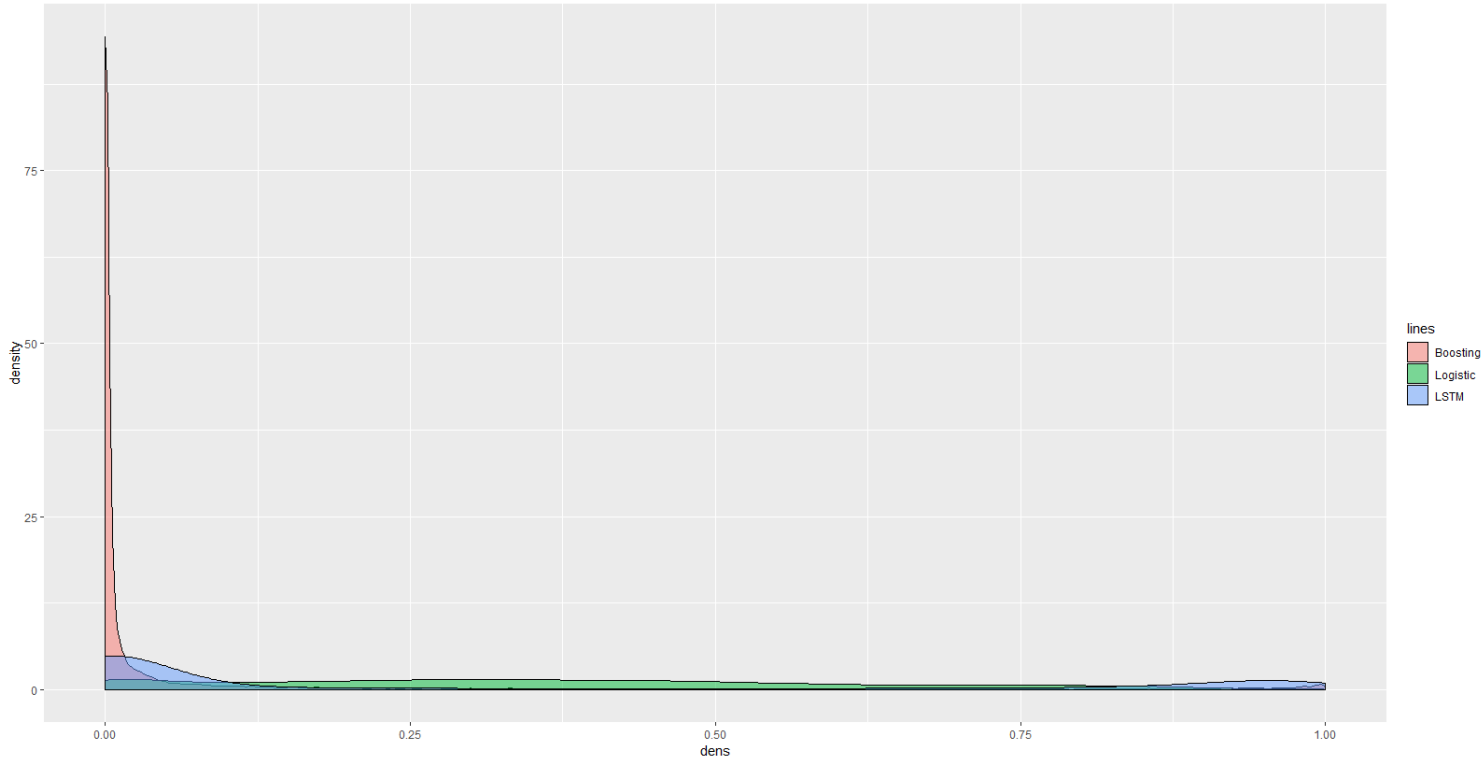
The results of table 8 can be summed up in the following three results:

1. The best model is the logistic regression model when measured as the model with the lowest share of issued loans that will default. This is measured through the false negative prediction rate $\frac{FN}{(FN+TN)}$.

2. The ability to correctly predict default is correlated with the credit rating. In general, as the credit rating lowers, the error rate increases. The lower predictive power and associated higher risk must be compensated through a higher risk premium. Hence, the cut-off point $\tau = 0.14$ is chosen since the ability to price risk correctly is deemed too low beyond this point.

3. The lower tails of the probability density distributions are much wider for LSTM and boosting, compared to logistic regression. This results in a much larger number of approved loans using LSTM and boosting compared to logistic regression with a cut-off of $\tau = 0.14$. The density plots can be seen in figure 10.

Figure 10: Probability Density Plot



The differences in the probability densities can be found in the respective loss functions used to train the models. The LSTM is coded as a multi-classification problem, with the special case $k = 2$. The binary cross-entropy loss function associated with this assigns a loss value to the probability created in each neuron, where the total probability sums to one. Consequently, the algorithm is forced to set a high value for one of the cases, in order to minimize the loss function. This results in a u-shaped probability density function, which is completely opposite to the probability density of the logistic regression. The gradient boosting algorithm uses a Bernoulli log-likelihood loss function, which is an exponential loss function. This again forces the boosting algorithm to make predictions close to 1 and 0, since a large loss will be associated with a prediction in the middle. This underlying difference in loss functions forces the LSTM and boosting algorithm to make a lot of predictions below the cut-off point $\tau = 0.14$, where risk can be priced correctly, given the predicted probabilities reflects the true probability of default. Thus, based on the structure of the probability densities, the Bernoulli log-likelihood loss associated with the boosting algorithm seems most appropriate for maximizing the objective function.

The implications of the different loss functions can be quantified through their profitability in

a Funding Circle like implementation. A simplified risk-based pricing approach, show the return and loss achieved by the three selected models. The calculation is based on a portfolio of the AA+ rated SMEs approved for a loan by each model. Cross-referencing with Funding Circle's price guide shows that a loan running for 2-3 years (forecasting period is two years), carries an annual interest rate of $3-6\%$. Since the SMEs in this group have an AA+ credit rating, the formula uses the lowest 3%p.a. interest charge. I assume for simplicity that the loan is DKK 1 million, unsecured and paid-in-full at repayment date. Also, interest charges are paid end of year and all defaults occur within the first year. The simplified total profit from a portfolio containing AA+ rated credit running for two years can be calculated as

$$\begin{aligned}
\text{Profit from Logistic: } & 2 \cdot (TN \cdot L \cdot i_{SME}) - FN \cdot L = 2 \cdot (605 \cdot 1 \cdot 0.03) - 4 \cdot 1 = 32.5m \\
\text{Profit from LSTM: } & 2 \cdot (TN \cdot L \cdot i_{SME}) - FN \cdot L = 2 \cdot (4017 \cdot 1 \cdot 0.03) - 14 \cdot 1 = 227m \\
\text{Profit from Boosting: } & 2 \cdot (TN \cdot L \cdot i_{SME}) - FN \cdot L = 2 \cdot (5891 \cdot 1 \cdot 0.03) - 23 \cdot 1 = 330m.
\end{aligned}$$

Here, TN shows the true negatives, i.e. the companies that remain active. The loan size is denoted L, and the false negatives FN are the number of companies that default during the loan period.

When both the false negative prediction rate$\frac{FN}{TN+FN}$ and the probability density distribution is taken into account, the best model is the boosting algorithm. Even though the boosting model has the worst false negative prediction rate $\frac{FN}{TN+FN}$, it compensates more than enough through a higher total amount of active classifications for that risk category. Thus, the nominal amount of true negatives is more important than the relative amount of false negative, given the probability density. Hence, when the model performance is translated into a business model with direct implications and non-symmetrical loss, it is clear that machine and deep learning methods are far superior to logistic regression. However, a final conclusion on the performance of the models cannot be made yet. The performance of each model needs to be evaluated at each credit rating. Instead of evaluating the models in the objective function above, I create a new objective function that comes as close as possible to the objective function of Funding Circle, based on the information that is publicly available. This will be defined in the following.

7.1.1 The best model in a peer-to-peer lending application

The three models will be compared based on a simplified objective function, mirroring a Funding Circle application. Given this structure, the ability to produce the highest profit at each credit

rating will determine the overall best model. In this way, the thesis brings an updated review of Oliver and Hand 2005, with respect to machine learning applications in risk-based pricing.

Funding Circle generates revenue by a sign-up fee for all loans $(TN + FN)$ and a servicing fee is charged on a yearly basis as well for surviving SMEs[15], TN. The servicing fee corresponds to the 3% p.a. related to the AA+ credit rating. Sign-up fees vary as well, but the lowest value 3% is chosen to reflect the AA+ credit rating. Marketing material from Funding Circle "promises" an estimated 4.2% annual return for investors in the conservative risk category. Since the SME loan portfolio is AA+ rated, it can be characterized as a conservative risk group. The same assumptions and simplifications used above are applied here as well. The annual percentage rate charged to SMEs in this risk category, must therefore deliver returns to cover the platform's fees, and an annual net-of-loss peer-to-peer return of 4.2%, denoted i_{peer}. This means that the credit risk is directly related to the peers, and not the platform in this market design. The example uses the results for the LSTM model in the AA+ credit rating. Solving for the annual percentage rate needed by the SMEs, i_{SME} to provide the peers with an annualized 4.2% return, given the pricing structure:

$$-\left[\underbrace{(FN)\cdot L}_{\text{Peer Loss from defaults}} + \left(\underbrace{(FN+TN)\cdot L\cdot 0.03}_{\text{Sign-up fees to FC)}}\right) + \left(\underbrace{((TN)\cdot L\cdot 0.03)\cdot 2}_{\text{Interest paid by SMEs to FC}}\right)\right] = \underbrace{(TN)\cdot L\cdot i_{Peer}\cdot 2}_{\text{Net peer interest}}$$

$$\underbrace{(TN)\cdot L\cdot i_{SME}\cdot 2}_{\text{Interest paid by SMEs}}$$

$$i_{peer} = i_{SME} - 0.047$$
$$i_{peer} = 4.2\% p.a.$$
$$\iff i_{SME} = 8.9\%\ A.P.R.$$

The risk-based pricing model puts the annual percentage rate charged to SMEs with an AA+ credit rating at 8.9%, using the LSTM model. The boosting algorithm arrives at a similar result. The logistic model arrives at an annual percentage rate of 9.05%. Funding Circle prices a two-year

[15]The full pricing structure depends on the specific loan type, duration and financial information. However, general pricing structures have been reviewed by third-parties, such as https://www.simplybusiness.co.uk/knowledge/articles/2019/04/compare-the-best-small-business-loans/ and Funding Circle's own official price-overview https://support.fundingcircle.com/hc/en-us/articles/214639426-What-fees-do-Funding-Circle-charge-.

loan to SMEs with a $7.60\% - 35.5\%$ annual percentage rate[16]. Although this a simplification, the performance of the models clearly mirrors the results achieved in industry, when the industry's credit rating and pricing structure is used.

The annual percentage rates are directly related to the negative predicted value $\frac{FN}{TN+FN}$, and does not reflect the differences in probability densities. Hence, in order to determine the profit created for each credit rating, the probability densities are taken into account. Table 9 shows the most profitable model at each credit rating.

Table 9: Most profitable models at each credit rating

Credit Rating	Predicted Probability of Default Range	Most profitable model
AA or higher	$0 - 0.01\%$	Boosting
A	$0.01 - 0.5\%$	Boosting
BBB	$0.5 - 1\%$	Boosting
BB	$1 - 3\%$	Boosting
B	$3 - 7\%$	Boosting
CCC or lower	$7 - 14\%$	Logistic

Based on table 9, I conclude that the overall best model of the three models considered is the boosting algorithm. This is in line with the AUC score achieved. Surprisingly, the logistic regression model gives the highest profit in the lowest credit rating. This is an interesting result, and it is directly related to its loss function. The other interesting result is that in 4/5 cases, the boosting model delivers superior profits, while at the same time having the highest default rate among its investments. This raises the question of moral hazard problems in the peer-to-peer industry. These two results will be discussed in the following.

The fact that the logistic model produces the largest amount of true negatives in the highest risk category, while at the same time having the lowest false negative predicted rate $\frac{FN}{TN+FN}$, indicates that it is the best model at approximating the true data generating process, within that risk group. On the other hand, this result is not transferred into superior profits, since the lower tail of the logistic distribution is not wide enough. Thus, if the tails are removed, the logistic regression model probably emulates the true data generating process the best, of all the models. This result exemplifies the differences between machine learning and econometrics. While econometrics have

[16]A loan price is offered when Funding Circle have access to company data. Third-party researches have gathered data based on these calculations and made them publicly available. One of them is https://www.nerdwallet.com/blog/small-business/funding-circle/, where this number is taken from.

sought to describe average effects by generalizations, computer scientists have focused on optimizing algorithms to solve a specific task.

Elements of both fields are needed to create the optimal prediction model for maximizing profits. Both fields are needed when the objective function is incorporated directly into the loss function, as was done here. This method can be further optimized through reinforced learning. Reinforced learning incorporates the objective function directly into the loss function of the prediction algorithm. This enables the model to learn that different profits arise from assigning different predicted probabilities of default to a given SME, since the cost of credit is based on the credit rating. The optimization of a reinforced learner in this setting requires millions of observations to perfect, but the method seems particularly well suited for this problem. The synthetic data generating methods used in this thesis could be used for this task.

7.2 Discussion on market design and moral hazard

The departure for this discussion is based on the two different regulatory environments for the peer-to-peer SME lending market. The discussion will build on the qualitative analysis made in Nemoto et al 2019. The traditional regulatory framework gives peer-to-peer platforms legal status as a *lender*, which effectively means that all credit risk must directly be related to the peer-to-peer lending platform. The progressive framework consists of a true peer-to-peer SME lending market design. The peer-to-peer lender takes legal status as a $platform$ that connects lenders with borrowers. Consequently, the lending platform does not have to take on any credit risks. UK-based Funding Circle is regulated through UK standards, which has legal requirements in the middle of the two extremes, but falls within the category of $platform$ legal status[17]. The UK legal system is praised by Nemoto et al. 2019 as the right balance between the two market designs. This result will be analyzed in the following.

With the traditional lending model, capital is raised based on expected returns, and any loss from defaults will be placed on the lender. Consequently, the lender will have higher incentives to keep losses at a minimum, and in order to keep the false negatives at a minimum, they will inadvertently have to decrease the cut-off point (cf. table 8). Increasing the cut-off points leads to an increase in false positives, meaning a lower amount of investment opportunities are realized. Conversely, since the peer-to-peer legal framework puts the loss of default directly on to the indi-

[17]https://support.fundingcircle.com/hc/en-us/articles/214636466-What-happens-if-a-borrower-misses-a-payment-

vidual investors, a peer-to-peer business model will have an incentive to decrease the false positive prediction rate $\frac{FP}{TP+FP}$, with a higher amount of false negatives accompanying it. Thus, the difference is directly related to where the cost of this trade-off between false positive and false negative, is placed in the objective function. Consequently, the peer-to-peer business model has another optimal cut-off point $\tau^\star$ and credit rating ranges, than the classical *lender* business model. This result is formally tested by Emekter et. al. 2015[18], which conclude that the lower the credit rating given by the platform, the larger the difference between the real price of risk and the interest rates offered. In other words, the interest rates charged from the highest risk customers is not enough to compensate for the increased default risk. Thus, the lending platform's optimal cut-off point $\tau^\star_{peer}$ is higher than that of the traditional *lending* business model $\tau^\star_{platform} > \tau^\star_{lender}$. This is however completely rational behavior cf. the objective function of the peer-to-peer platform. Hence, the peer-to-peer lending structure creates a classic moral hazard problem.

It is clear that on the one hand, peer-to-peer lending could be the technological revolution needed to close the supply gap in SME finance. On the other hand, it does so by having an incentive structure that increases over-all credit risk, and subsequently putting the risk onto private investors. Reading from the investors page from Funding Circle's home page, targeting private individual investors, it says that "Lend to at least 200 businesses & Lend no more than 0.5% of your total to each one[19]".
Peer-to-peer SME finance is growing across Europe. It will definitely help to close the SME finance gap, but regulatory oversight, transparency with respect to credit risks, and informed private investors is key to making sure, that it will continue to be a success.

8 Conclusion

This thesis took its departure in the SME credit supply gap that has remained persistently high, even under the current monetary expansionist programs. The thesis has looked towards industry and literature to highlight the central role that machine learning has, and will increasingly have, in the area of credit default prediction. The thesis have used the best models from previous literature, along with a new one, and applied them as they would have been in the peer-to-peer SME credit industry. I believe that a market application approach, deserves more attention than it has been given previously in literature. The multi-disciplinary requirements for this implementation makes

[18] Although the test was conducted on data from Lending Club - a peer-to-peer consumer credit platform, the same result can be applied to Funding Circle.

[19] Funding Circle homepage

economists especially well-suited for future research within this area.

Specifically, I have used machine and deep learning models to analyze and price SME credit risk. This was done based on ex-ante predictions of the SME specific default probability. First, industry-standard credit-rating methodology was used to develop a portfolio of investment opportunities in SME credit, for different risk categories. Next, risk-based pricing was used to developed a simplified non-symmetrical loss function, mirroring the objective function of a peer-to-peer lender. The developed machine and deep learning models were then evaluated based on their ability to generate profit. In the last section, the moral hazard problem that is inherent to the peer-to-peer credit industry was analyzed.

This thesis is in no way exhaustive of the issues related to machine and deep learning based SME credit default prediction. However, three results stand out.

First, machine learning in SME credit risk is fairly new, but already adopted by industry. Machine learning focuses on predicting an outcome instead of estimating feature specific effects on the default probability, as previously done through econometric methods. From a SME credit investor's point of view, this makes sense. The core competence of an SME credit investor is to determine and price the risk of default correctly. The ability to do it consistently, and with the lowest margin of error as possible, determines if the investor will make money or not. Embedded in the goal of minimizing error and machine learning's ability to handle an extremely large amount of features, the industry is incorporating new avenues of explanatory data. In a world where the amount of data is constantly growing, traditional statistical methods may not be the best tool to predict default risk. More interestingly, as the machine learning models increase in complexity, the need for large scale data preprocessing and conversion of raw data to features will become less important. Combine this with reinforced learning adaptation, and the need for traditional econometric methods looks increasingly irrelevant. There is however still a need for classical economics in this evolution. The ability to explain real world phenomena from a wide range of quantitative and qualitative data, will be a key competence in knowing which data avenue to incorporate. As John Naisbitt said ahead of his time in 1982 *"We are drowning in information, but starved for knowledge".*

Second, while machine learning models derive a large part of their superior predictive power from their flexibility and few theoretical restrictions, they will ultimately depend on the data which they are trained upon. This also implies that the models will be highly sensitive to changes in the underlying data generating process. Since no assumptions are made to understand the underlying

data generating process, the ability to draw inference is severely limited, compared to econometrics. While the variable importance measures are used as a substitution, they do not hold the same unbiased and ceteris paribus properties that are known from econometrics. Hence, no causal interpretation can be done on the machine learning parameters, *even* if the identifying assumptions from econometrics are used. While these are all problems that we(as in economists) are used to have solutions to, the computer scientists do not. I hope that future studies by economists will incorporate the limitations that machine learning methods have in respect to making inference - with the goal of pushing research towards new solutions.

The third, and final main point will be practical. Since all prediction involves estimation, there will be times when the prediction models are terribly wrong. Consequently, machine learning based prediction models should never be a stand alone decision tool. Arguments from traditional economic and financial analysis should be used to verify the predictions made.

The main objective of this thesis has not been to prove if the models are good enough to work in practice; they are already used in the peer-to-peer SME credit industry. Instead, the unit of analysis moves further than similar studies and looks at how and why they are used in industry, by emulating a peer-to-peer application. Without access to the industry's intellectual property, a full comparative study between the models developed here, and the one used in industry is impossible. I do note however, that the best model's implied interest rate, for a given credit rating, using the pricing structure from industry, comes remarkably close to the ones offered in the peer-to-peer market.

References

[1] Peter Addo, Dominique Guegan, and Bertrand Hassani. Credit risk analysis using machine and deep learning models. *Risks*, 6(2):38, 2018.

[2] Edward I Altman. Financial ratios, discriminant analysis and the prediction of corporate bankruptcy. *The journal of finance*, 23(4):589–609, 1968.

[3] Edward I Altman, Robert G Haldeman, and Paul Narayanan. Zetatm analysis a new model to identify bankruptcy risk of corporations. *Journal of banking & finance*, 1(1):29–54, 1977.

[4] Gerhard Arminger, Daniel Enache, and Thorsten Bonne. Analyzing credit risk data: A comparison of logistic discrimination, classification tree analysis, and feedforward networks. *Computational Statistics*, 12(2), 1997.

[5] Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and Robert E Schapire. Gambling in a rigged casino: The adversarial multi-armed bandit problem. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 322–331. IEEE, 1995.

[6] Abdul Aziz, David C Emanuel, and Gerald H Lawson. Bankruptcy prediction-an investigation of cash flow based models [1]. *Journal of Management Studies*, 25(5):419–437, 1988.

[7] Abdul Aziz and Gerald H Lawson. Cash flow reporting and financial distress models: Testing of hypotheses. *Financial Management*, pages 55–63, 1989.

[8] William H Beaver. Financial ratios as predictors of failure. *Journal of accounting research*, pages 71–111, 1966.

[9] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992.

[10] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[11] John S Bridle. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing*, pages 227–236. Springer, 1990.

[12] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[13] Harris Drucker, Corinna Cortes, Lawrence D Jackel, Yann LeCun, and Vladimir Vapnik. Boosting and other ensemble methods. *Neural Computation*, 6(6):1289–1301, 1994.

[14] Wendy Edelberg. Risk-based pricing of interest rates for consumer loans. *Journal of monetary Economics*, 53(8):2283–2298, 2006.

[15] T Elhassan and M Aljurf. Classification of imbalance data using tomek link (t-link) combined with random under-sampling (rus) as a data reduction method.". 2016.

[16] Riza Emekter, Yanbin Tu, Benjamas Jirasakuldech, and Min Lu. Evaluating credit risk and loan performance in online peer-to-peer (p2p) lending. *Applied Economics*, 47(1):54–70, 2015.

[17] Thomas Fischer and Christopher Krauss. Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, 270(2):654–669, 2018.

[18] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

[19] Salvador García, Julián Luengo, and Francisco Herrera. *Data preprocessing in data mining*. Springer, 2015.

[20] Giacomo Giannoccaro and Julio Berbel. The determinants of farmerâs intended behaviour towards the adoption of energy crops in southern spain: An application of the classification tree-method. *Bio-based and Applied Economics*, 1(2):199–212, 2012.

[21] Alex Graves. Supervised sequence labelling with recurrent neural networks. 2012. *URL http://books. google. com/books*.

[22] David J Hand. Measuring classifier performance: a coherent alternative to the area under the roc curve. *Machine learning*, 77(1):103–123, 2009.

[23] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[24] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[25] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[26] Ralph L Keeney and RM Oliver. Designing win-win financial loan products for consumers and businesses. *Journal of the Operational Research Society*, 56(9):1030–1040, 2005.

[27] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[28] Sotiris Kotsiantis and Dimitris Kanellopoulos. Association rules mining: A recent overview. *GESTS International Transactions on Computer Science and Engineering*, 32(1):71–82, 2006.

[29] Miroslav Kubat, Stan Matwin, et al. Addressing the curse of imbalanced training sets: one-sided selection. In *Icml*, volume 97, pages 179–186. Nashville, USA, 1997.

[30] Regan Lam, Suzan Burton, and Hing-Po Lo. Customer tradeoffs between key determinants of sme banking loyalty. *International journal of bank marketing*, 27(6):428–445, 2009.

[31] Jae H Min and Young-Chan Lee. Bankruptcy prediction using support vector machine with optimal choice of kernel function parameters. *Expert systems with applications*, 28(4):603–614, 2005.

[32] Winston Moore and Roland Craigwell. The relationship between commercial banks' interest rates and loan sizes: evidence from a small open economy. *Applied Financial Economics*, 13(4):257–266, 2003.

[33] Ajinkya More. Survey of resampling techniques for improving classification performance in unbalanced datasets. *arXiv preprint arXiv:1608.06048*, 2016.

[34] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[35] Naoko Nemoto, Bihong Huang, and David J Storey. Optimal regulation of p2p lending for small and medium-sized enterprises. 2019.

[36] James A Ohlson. Financial ratios and the probabilistic prediction of bankruptcy. *Journal of accounting research*, pages 109–131, 1980.

[37] Robert M Oliver and Eric Wells. Efficient frontier cutoff policies in credit portfolios. *Journal of the Operational Research Society*, 52(9):1025–1033, 2001.

[38] Tracy Panther and Jillian Dawes Farquhar. Consumer responses to dissatisfaction with financial service providers: an exploration of why some stay while others switch. *Journal of Financial Services Marketing*, 8(4):343–353, 2004.

[39] Paul PM Pompe and Jan Bilderbeek. The prediction of bankruptcy of small-and medium-sized industrial firms. *Journal of Business venturing*, 20(6):847–868, 2005.

[40] Islem Rekik, Gozde Unal, Ehsan Adeli, and Sang Hyun Park. *PRedictive Intelligence in MEdicine: First International Workshop, PRIME 2018, Held in Conjunction with MICCAI 2018, Granada, Spain, September 16, 2018, Proceedings*, volume 11121. Springer, 2018.

[41] Hartayuni Sain and Santi Wulan Purnami. Combine sampling support vector machine for imbalanced data classification. *Procedia Computer Science*, 72:59–66, 2015.

[42] Robert E Schapire. The strength of weak learnability. *Machine learning*, 5(2):197–227, 1990.

[43] Joseph E Stiglitz. Some aspects of the pure theory of corporate finance: bankruptcies and take-overs. *The Bell Journal of economics and management Science*, pages 458–482, 1972.

[44] LC Thomas, RW Oliver, and DJ Hand. A survey of the issues in consumer credit modelling research. *Journal of the Operational Research Society*, 56(9):1006–1015, 2005.

[45] Lyn C Thomas. *Consumer credit models: pricing, profit and portfolios: pricing, profit and portfolios*. OUP Oxford, 2009.

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

[47] Vladimir Vapnik. Principles of risk minimization for learning theory. In *Advances in neural information processing systems*, pages 831–838, 1992.

[48] Chongren Wang, Dongmei Han, Qigang Liu, and Suyuan Luo. A deep learning approach for credit scoring of peer-to-peer lending using attention mechanism lstm. *IEEE Access*, 7:2161–2168, 2019.

[49] Paul J Werbos et al. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[50] Ronald J Williams and David Zipser. Gradient-based learning algorithms for recurrent. *Backpropagation: Theory, architectures, and applications*, 433, 1995.

[51] Guanting Chen Shiqi Yang. Application of deep learning to credit risk modeling. Master's thesis.

[52] You Zhu, Chi Xie, Gang-Jin Wang, and Xin-Guo Yan. Comparison of individual, ensemble and integrated ensemble machine learning methods to predict chinaâs sme credit risk in supply chain finance. *Neural Computing and Applications*, 28(1):41–50, 2017.

[53] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the royal statistical society: series B (statistical methodology)*, 67(2):301–320, 2005.

9 Appendix A

Probabilistic classification

This section contains an intuitive explanation of why tuning the neural network weights will lead to a lower loss. Also, it bridges the theoretical gap between probability based classifiers and regular classification. The description follows Graves 2005.

Classifiers that directly output class labels, of which support vector machine are a well known example, are sometimes referred to as discriminant functions. An alternative approach is probabilistic classification, where the conditional probabilities $p(C_k \mid x)$ of the K classes given the input pattern x are first determined, and the most probable is then chosen as the classifier output $h(x)$:

$$h(x) = \arg\max_k p(C_k \mid x). \tag{58}$$

One advantage of the probabilistic approach is that the relative magnitude of the probabilities can be used to determine the degree of confidence the classifier has in its outputs. Another is that it allows the classifier to be combined with other probabilistic algorithms in a consistent way.

Training probabilistic classifiers

In a probabilistic classifier h_w yields a conditional distribution $p(C_k \mid x, w)$ over the class labels C_k given input x and parameters w, the product can be taken over the independent and identically distributed *i.i.d.* input-target pairs in the training set S, to get

$$p(S \mid w) = \prod_{(x,z)\in S} p(z \mid x, w), \tag{59}$$

which can be inverted with Bayes' rule to obtain

$$p(w \mid S) = \frac{p(S \mid w)\, p(w)}{p(S)}. \tag{60}$$

In theory, the posterior distribution over the classes for some new input x can then be found by integrating over all possible values of w:

$$p(C_k \mid x, S) = \int_w p(C_k \mid x, w)\, p(w \mid S)\, dw. \tag{61}$$

In practice w is usually very high dimensional and the above integral, which is known as the classifier's predictive distribution, is intractable. A common approximation, known as the maximum a priori (MAP) approximation, is to find the single parameter vector w_{MAP}, that maximizes $p(w \mid S)$ and use this to make predictions:

$$p(C_k \mid x, S) \approx p(C_k \mid x, w_{MAP}). \tag{62}$$

Since $p(S)$ is independent of w, equation 60 implies that

$$w_{MAP} = \arg\max_w p(S \mid w)\, p(w). \tag{63}$$

The parameter prior $p(w)$ is usually referred to as the regularization term. Its effect is to weight the classier towards those parameter values which are deemed a priori more probable. In accordance with Occam's razor, usually assume that more complex parameters (where "complex" is typically interpreted as "requiring more information to accurately describe") are inherently less probable. For this reason $p(w)$ is sometimes referred to as an Occam factor or complexity penalty. In the particular case of a Gaussian parameter prior, where $p(w) \propto |w|^2$, the $p(w)$ term is referred to as weight decay. If, on the other hand, a uniform prior is assumed over the parameters, the $p(w)$ term can removed the from equation (63), to obtain the maximum likelihood parameter w_{ML}

$$w_{ML} = \arg\max_w p(S \mid w) = \arg\max_w \prod_{(x,z)\in S} p(z \mid x, w). \tag{64}$$

The explicit assumption that that the classifier depends on the weights, w, is dropped and the probability that x is correctly classified by h_w is denoted $p(z \mid x)$ throughout section 4.5 and 4.6.

Maximum-likelihood loss functions

The standard procedure for finding $w_{ML}$is to maximize a maximum-likelihood loss function $\mathcal{L}(S)$ defined as the negative logarithm of the probability assigned to S by the classifier

$$\mathcal{L}(S) = -\ln \prod_{(x,z)\in S} p(z \mid x) = -\sum_{(x,z)\in S} \ln p(z \mid x). \tag{65}$$

Note that since logarithm is monotonically increasing, minimizing $-\ln p(S)$ is equivalent to maximizing $p(S)$. Observing that each training sample $(x, z) \in S$ contributes to a single term in the

above sum, the example loss $\mathcal{L}(x, z)$ can be defined as

$$\mathcal{L}(x, z) = -\ln p(x \mid z), \tag{66}$$

with

$$\mathcal{L}(S) = \sum_{(x,z)\in S} \mathcal{L}(x, z) \tag{67}$$

$$\frac{\partial \mathcal{L}(S)}{\partial w} = \sum_{(x,z)\in S} \frac{\partial \mathcal{L}(x, z)}{\partial w}. \tag{68}$$

Equation 66-68 shows that it is suffices to derive $\mathcal{L}(x, z)$ and $\frac{\partial \mathcal{L}(x,z)}{\partial w}$ to completely define a maximum-likelihood loss function and its derivatives with respect to the network weights, w.

10 Appendix B

This section shows output of interest from the different models.

Figure 11: Significant variables from the logistic regression

Variable Name	Significance level $pr(>\|z\|)$
EBITDA/TA	0
EBITDA/EQUITY	0
Cashflow/Equity	0
Cashflow/Revenue	0
Netprofit/Revenue	0
Netprofit/TA	0
Netprofit/Equity	0
Netprofit/Employees	0
Revenue/Cost of Employees	0
Revenue/Cost of goods	0.05
Creditors/Revenue	0.05
ln(Revenue)	0
ln(LTL)	0.1
ln(Netprofit)	0
EBITDA/Equity t-1	0
Equity/Liabilities t-1	0.1
LTL/TA t-1	0
Cashflow/Equity t-1	0.01
Cashflow/Revenue t-1	0
Netprofit/Revenue t-1	0.1
Netprofit/Equity t-1	0.01
Cashflow/STL t-1	0
Current Ratio t-1	0
Debtors/Revenue t-1	0.01
ln(Revenue) t-1	0
ln(TA) t-1	0
ln(LTL) t-1	0.05
ln(STL) t-1	0.05
ln(Netprofit)	0
Equity/Liabilities t-2	0.01
LTL/FA t-2	0.05
Cashflow/Revenue t-2	0
Grossprofit/Revenue t-2	0
Cashflow/STL t-2	0.01
Current ratio t-2	0
Revenue/Cost of Employees t-2	0
Revenue/Cost of goods t-2	0.01
Debtors/Revenue t-2	0.05
Creditors/Revenue t-2	0
ln(Revenue) t-2	0.05
ln(TA) t-2	0
ln(LTL) t-2	0
ln(Netprofit) t-2	0

11 Appendix C

Figure 12: Variable importance plot, Bagging

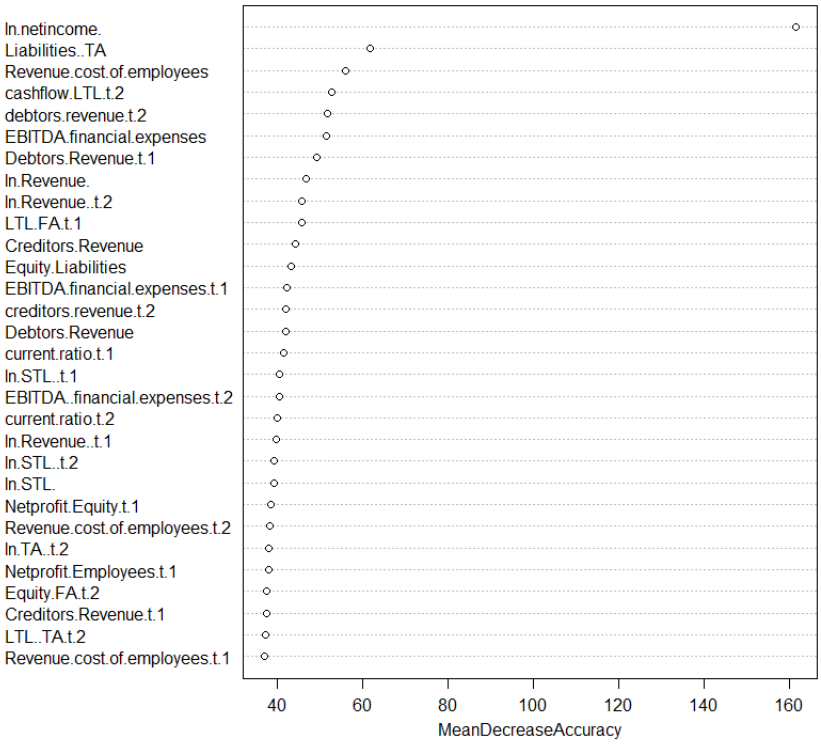


Figure 13: Variable Importance plot, Random Forest

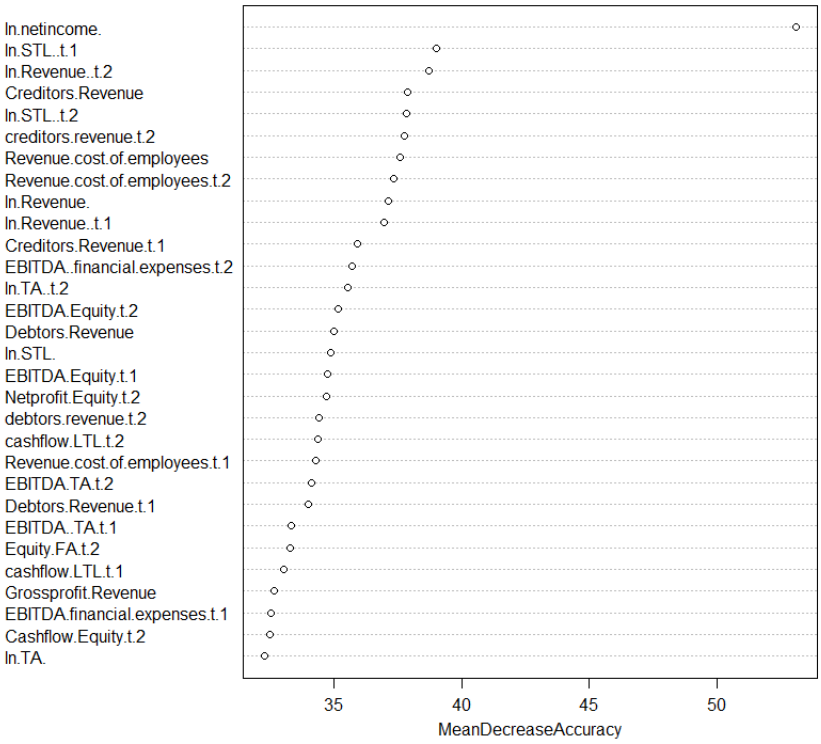
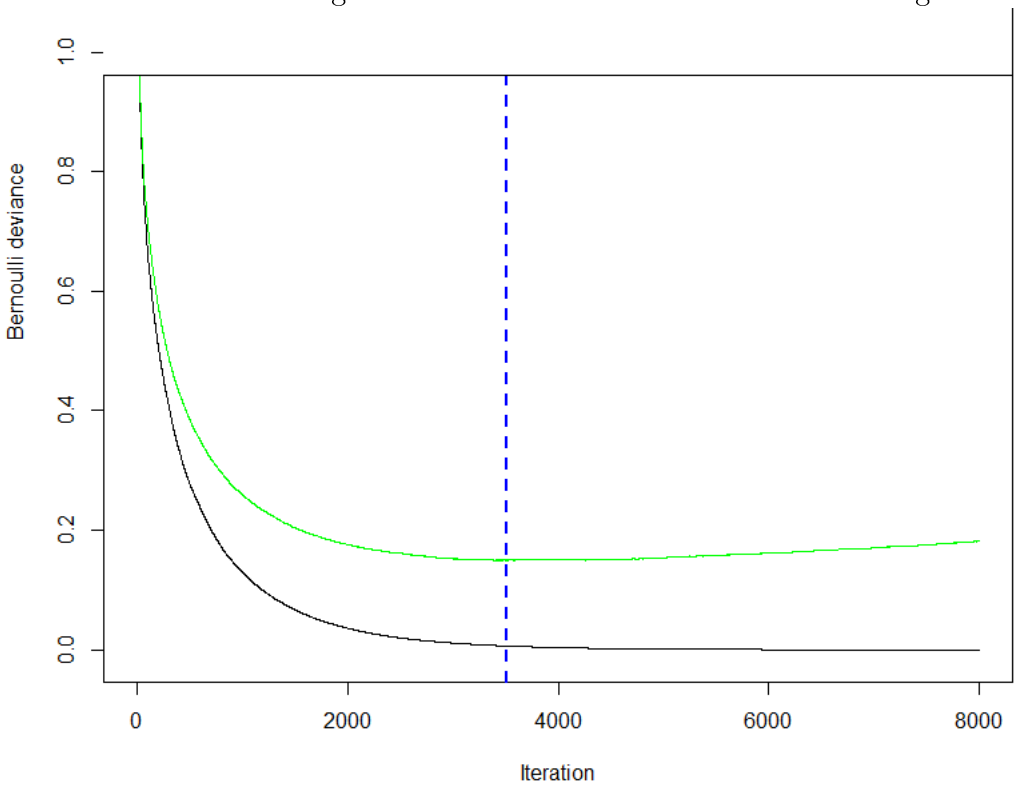


Figure 14: Cross Validation Error in Boosting

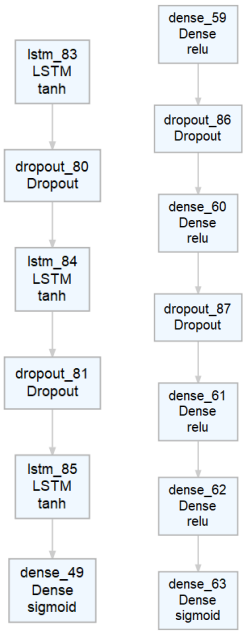


- The green curve show the cross-validation error as function of iterations
- The black curve shows the in-sample error.
- The dashed blue line show the minimum of the cross-validation curve, and hence the optimal amount of iterations. The optimal amount of iterations is then used to train the model.

Table 10: Grid Search: Cross-validation error from SVM with 3-degree polynomial kernel

C	γ			
	2^{-1}	2^{-3}	2^{-5}	2^{-7}
2^{1}	0.022	0.37	0.104	0.018
2^{3}	0.022	0.260	0.068	0.147
2^{5}	0.022	0.023	0.050	0.110
2^{7}	0.020	0.023	0.038	0.083
2^{9}	0.022	0.023	0.033	0.064
2^{11}	0.023	0.024	0.033	0.05
2^{12}	0.022	0.24	0.033	0.044

Figure 15: Graphical representation of the LSTM and MLP Architecture implemented



12 Appendix D

Table 11: Overview of tuning parameters

Algorithm	R package	Parameter	Optimal Value
Logistic regression	caret	–	–
Classification tree	tree	*tree.size*	6
Random Forest	randomForest	*mtry*	10
		B	500
Gradient Boosted Trees	gbm	*n.trees*	3510
		η	0.1
		interaction.depth	6
		$bag.fraction$	0.5
		n.minobsinnode	10
Support Vector Machine	e1071	*degree*	3
		C	16
		λ	0.25
MLP	–	*hidden layers*	3
		hidden units	22, 8, 4
		dropout	0.4, 03
LSTM	–	*hidden layers*	2
		hidden units	29, 29
		dropout	0.4, 0.4
		recurrent dropout	0, 3 0, 3