Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

Petter Aarseth Moen

Bankruptcy prediction for Norwegian enterprises using interpretable machine learning models with a novel timeseries problem formulation

Master's thesis in Applied Physics and Mathematics
Supervisor: Jo Eidsvik
July 2020

NTNU
Norwegian University of Science and Technology

Petter Aarseth Moen

Bankruptcy prediction for Norwegian enterprises using interpretable machine learning models with a novel timeseries problem formulation

Master's thesis in Applied Physics and Mathematics
Supervisor: Jo Eidsvik
July 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

NTNU
Norwegian University of
Science and Technology

Abstract

Prediction of corporate bankruptcy is a topic of great relevance to both investors, creditors, banks, and regulators, offering significant potential for economic gains. Using a dataset of financial statements from more than 175000 Norwegian small and medium-sized enterprises spanning 8 years, we implement and train three static statistical models (logistic regression, neural networks, and CatBoost) and test their performance on a representative set of performance metrics. New for the field is the gradient boosting framework CatBoost, which produces an AUC score of 0.8735 on a balanced test dataset, compared to 0.8437 of the traditionally used logistic regression model.

This performance increase is partly facilitated by the introduction of a novel categorical feature, containing information about the industry of a company. We present and compare several ways of integrating such categorical features into the model frameworks, and find that a target encoding generally performs the best. We also find that more compact feature subsets of 30 financial ratio features (as opposed to the full 156 feature set) achieve comparable performance in all cases.

We then formulate the bankruptcy prediction problem as a timeseries prediction problem, using subsequent years of financial ratios to construct timeseries containing 1-4 years of such data. We implement two neural network based timeseries models, namely recurrent neural networks and long short term memory networks, which are found to produce balanced test set AUC scores of 0.8651 and 0.8698, respectively. While worse than the CatBoost model (and similar to the standard neural network) for the timeseries with only 1 year of data, we find that the timeseries models produce significantly better results for timeseries with 3 and 4 years of data, with AUC scores of 0.8827 and 0.8891 for the LSTM model, respectively.

Finally, we outline a theoretically sound model interpretation framework, named SHAP, providing values for individual feature contributions to any model prediction. We then demonstrate how this framework can be applied to our considered bankruptcy prediction models, both for feature selection and analysis of learned model behaviour. The former is found to perform comparatively to a more exhaustive feature selection search method.

Sammendrag

Prediksjon av konkurs hos selskaper er et emne som er relevant både hos investorer, kreditorer, banker og regulatorer. I denne oppgaven bruker vi et datasett bestående av årsrapporter fra mer enn 175000 norske små- og mellomstore bedrifter over 8 år til å trene tre statiske statistiske modeller (logistisk regresjon, nevrale nettverk og CatBoost), og tester ytelsen på et representativt sett ytelsesmetrikker. Nytt for området er prediksjonsrammeverket CatBoost, som gir en AUC-score på 0.8735 på et balansert testdatasett, sammenlignet med 0.8437 hos den mer tradisjonelle logistisk regresjonsmodellen.

Ytelsesøkningen kommer delvis av introduksjonen av en ny kategorisk variabel som inneholder informasjon om industrområdet til selskapet. Vi presenterer og tester også forskjellige måter å integrere kategoriske variabler i modellen, og finner at target encoding gir generelt best resultater. Vi finner også at et mer kompakt variabelsett med 30 nøkkeltallvariabler (i motsetning til 156 i det fulle variabelsettet) gir sammenlignbar ytelse i alle tilfeller.

Videre formulerer vi konkursproduksjonsproblemet som et tidsrekkeprediksjonsproblem, og bruker følgende år med nøkkeltall til å konstruere tidsrekker med 1-4 år av denne dataen. Vi implementerer to tidsrekkemodeller basert på nevrale nettverk, RNN og LSTM, som produserer testsett AUC-scorer på henholdsvis 0.8651 og 0.8698. Tidsrekkemodellene yter verre enn CatBoost-modellen (og sammenlignbart med det vanlige nevrale nettverket) på tidsrekker med 1 år tilgjengelig data, men produserer signifikant bedre resultater på tidsrekker med 3 og 4 år med data, med AUC-scorer på henholdsvis 0.8827 og 0.8891 for LSTM-modellen.

Til slutt presenterer vi et teoretisk solid rammeverk for modellinterpretasjon, kalt SHAP, som gir verdier for individuelle variabelbidrag til enhver modellprediksjon. Vi demonstrerer så hvordan dette rammeverket kan brukes på våre konkursprediksjonsmodeller, både til variabelseleksjon og analyse av modellens lærte oppførsel. Vi finner at førstnevnte produserer sammenlignbare resultater som med mer komplekse variabelseleksjonsmetoder.

Preface

This thesis is written as the final part of my studies in *Industrial mathematics*, concluding my five years of studies at the Norwegian University of Technology (NTNU). Studying at NTNU has been an exceptionally rewarding experience throughout, offering invaluable academical, social, and personal experiences that I will cherish for as long as I live.

First and foremost, I want to thank my supervisor, Professor Jo Eidsvik, for valuable discussions, insights, and reflections, for which without this thesis would not be possible. My greatest gratitude also goes to PhD candidate Ranik Raaen Wahlstrøm, who has been an invaluable resource throughout the process, consistently providing relevant insights and pointers that would otherwise be lost on me.

Finally I want to thank my family, for their unconditional support and affliction, and my fellow students and friends, for the wonderful memories that I will always remember.

Contents

Chapter 1

Introduction

In this introductory chapter, we will provide the relevant background for the study of bankruptcy prediction, motivating and then highlighting the specific areas of focus chosen in this thesis. In the following sections, we further give an overview of the overall thesis structure, and detail our concrete contributions to the field.

1.1 Background and motivation for bankruptcy prediction

Corporate bankruptcy is an ever prevalent issue in the global financial world, incurring large costs to both the company and its stakeholders, related financial institutions and even the economical ecosystem as a whole (Alaka et al., 2017). Therefore, predicting bankruptcies ahead of time has been of great interest to both academics and practitioners, including investors, financial institutions and regulators. The topic has gotten increased attention after the 2007-08 global financial crisis, which exposed major shortcomings in financial risk management systems, resulting in reorganization and bankruptcies in many companies, including many of the world's largest financial institutions.

Bankruptcy prediction sits at the intersection between the fields of finance and statistics. The latter field has seen significant developments over the past decade, specifically in the fields of machine learning (or statistical learning), where increased computing power and data availability combined with new algorithmic models (or improvements to existing ones) are leveraged to give significant improvements in predictive performance.

Despite this, most financial institutions still typically deploy more traditional statistical methods to model prediction of bankruptcies, partly due to their practical interpretability, compared to more the complex modern models (Zhang and Thomas, 2015). In an increasingly competitive financial industry with new alternatives to traditional bank loans (such as crowdfunding and peer-to-peer lending) entering the markets, more accurate analysis of companies are more important than ever, motivating the importance of better-performing bankruptcy prediction models.

It is important to still consider the importance of practical interpretability of such models, as most business practitioners prefer more interpretable models over complex ones, even if the latter is outperforming the former (Jones et al., 2017). Considering the fact that even small performance increases in bankruptcy prediction models can lead to significant economical gains (Stein, 2005), this proves the concrete value model interpretability holds amongst practitioners.

While small and medium-sized enterprises (SMEs) comprise the majority of global companies (indeed, 99% of all EU enterprises in 2015 were considered SMEs (Papadopoulos et al., 2015)), the literature of bankruptcy prediction is focused mostly on large or listed companies (Wahlstrøm et al., 2020). This is mainly due to easier available financial data and the possibility of using marked-based information. Despite this, the economical magnitude of SMEs make them obvious candidates for study, as they are crucial to both national, regional

and global economies (Gupta et al., 2018), particularly in job creation (Neumark et al., 2008). Furthermore, SMEs often have trouble acquiring financing due to high capital requirements and greater economical uncertainties (especially after the EU's new Basel III regulatory requirements (Bank of International Settlements, 2017)), motivating studies of improved risk modelling and prediction capabilities for this segment.

Another common problem for bankruptcy prediction studies is a lack of extensive data; many recent studies use samples from only 400 or less companies (Kumar and Ravi 2007; Veganzones and Séverin 2020). The data is also often susceptible of significant sample bias, as it is typically sourced from e.g. a single set of customers from a specific bank (Veganzones and Séverin, 2020). This does not only weaken the statistical strength and significance of the analysis, but also limits the potential of many modern machine learning models, which typically require both considerable and representative data in order to produce unbiased performance gains (Pasini, 2015).

It is reasonable to believe that multiple previous years of financial data can be relevant to predict bankruptcy. By modelling bankruptcy prediction as a timeseries problem, one may capture information about how a company's financial situation has developed over time, which intuitively could be very relevant to predicting bankruptcy. Despite this, there is little in the current literature that explore this. One reason for this may be in the in the corresponding increase in required data, making a significant number of timeseries samples even harder to construct.

1.2 Focus of thesis & thesis structure

The main focus of this thesis will be concerned with the application of modern machine learning methods to the problem of bankruptcy prediction in the segment of Norwegian SMEs. We will leverage a large and consistent dataset of financial statements for such Norwegian SMEs, spanning from the years $2006 - 2014$, while also considering the practical requirements involved with potential applications of such models. Motivated by the importance of increased model performance, there will be a general focus towards achieving good model performance for a set of representative metrics, while also introducing and demonstrating the use of a new model interpretability framework to help bridge the previous gap in interpretability between the traditional and more complex models.

There are still some relevant issues that are left outside the scope of this thesis. The most apparent are those concerned with the inherent imbalance between the number of bankrupt and non-bankrupt companies (the *unbalanced data problem*, discussed in Section 2.1.3), as well as some practical problems throughout, that would typically require more qualitative economical analysis. We will generally not perform any such analysis, focusing instead on quantitative and reproducible results.

In Chapter 2, we discuss all practical considerations and relevant background for bankruptcy prediction. We will also highlight relevant previous work, and detail the Norwegian SME financial statement dataset that will be used throughout.

In Chapter 3, we provide the theoretical foundations for all statistical methods and models that will be considered for performing bankruptcy prediction, introducing both static (i.e. non-time dependent) and timeseries models. We will also highlight and discuss the relevant strengths and weaknesses associated with each of the methods, as well as address some practical considerations to help solve the latter.

In Chapter 4, we describe the general frameworks under which we will compare and evaluate our bankruptcy prediction models. Specifically, a set of four metrics will be introduced and their strengths and weaknesses highlighted. We will then describe both the theoretical and practical aspects of the SHAP model interpretability framework, which we will later use to analyze and interpret the learned behaviours of our models.

In Chapter 5, we detail the experimental setup deployed when training and analyzing the models. This includes practical considerations such as data preprocessing, train and test dataset splitting, sampling methodologies and model implementations. While the methodology was designed to adhere to the highest standards of machine learning best

practice, any potential weaknesses or shortcomings of any of these processes will also be highlighted.

In Chapters 6 and 7, we present the results for the conducted bankruptcy prediction experiments, with a focus on comparative model performance on each of our selected metrics. We will leverage our temporal validation split in order to analyze the temporal stability of our models, as well a holdout test set to analyze expected out of sample performance. By reproducing previously popular models in the field (logistic regression and neural networks), we perform benchmark analysis on our dataset in order to demonstrate the relative gains of the more complex gradient boosting and timeseries models. Finally, we will demonstrate how to use SHAP values to perform model interpretation analysis for each of the considered models. While the analysis for the static and timeseries methods are somewhat similar, we contain them to separate chapters for clarity.

Finally, we summarize and conclude our findings in Chapter 8, and point towards some relevant avenues for further work in the field.

1.3 Contributions

The main contributions of this work will be towards increased predictive performance for bankruptcy prediction models, specifically in the context of SMEs, which has historically received less attention in the literature. We will leverage both modern models and a large dataset of more than 175000 Norwegian SMEs in order to achieve this goal, following machine learning best practices throughout.

One major contribution towards this goal is the structuring of the bankruptcy prediction problem as a timeseries problem, and the application of recurrent neural networks to perform bankruptcy prediction. This is partly facilitated by our vast dataset spanning multiple years, as limited datasets have made such analysis hard to perform in the past.

Another contribution is the introduction of the company industry as a categorical feature, by leveraging the standardized NACE system. We also demonstrate different ways of incorporating such categorical features into both previous and newer models, and analyze the relative performances of each of these methods.

Finally, we will demonstrate how to use the SHAP model interpretability framework to interpret the learned behaviour of the bankruptcy prediction models in a theoretically consistent way. This offers previously unseen insights about how the more complex models learn and perform predictions, offering comparative levels of interpretability as the simpler models. This may be highly relevant to practitioners, and help facilitate adoption of better performing (although more complex) models in practical contexts, which in turn may produce significant economical gains.

Chapter 2

Background

2.1 Bankruptcy prediction

The problem of predicting the bankruptcy of a company is both an important and difficult problem which has been at the intersection between the fields of finance and statistics for multiple decades. It is of great relevance to both investors and creditors wanting to perform credit risk management (Härdle et al., 2009), as well as by banks for estimating interest rates, and overall adherence to regulatory frameworks such as Basel III (Bank of International Settlements, 2017). Bankruptcy prediction models are also used by regulators to assess the state of the financial markets, such as the SEBRA model of used by Norges Bank (Bernhardsen and Larsen, 2007).

Stein (2005) found that even small performance increases in such models can lead to significant economic benefits to a bank, motivating increased attention towards leveraging both the availability of data and continuous developments in prediction models, in order to achieve the best possible performing bankruptcy prediction models.

In this section, we will provide the relevant background and context related to bankruptcy prediction, briefly describing the definition and data context for the problem.

2.1.1 Definition of bankruptcy

While the exact legal definition of bankruptcy is often complex and may vary across jurisdictions and factors related to the company, the general concept is simple; a company is bankrupt if it is illiquid (meaning that its short-term obligations exceed their liquid assets), and that it is insufficient (essentially implying that the total liabilities exceed the total assets). While there are many legal intricacies associated with bankruptcies, this general definition will be sufficient for our considerations (Konkursrådet, 2020).

In this thesis and the works that are considered, bankruptcy will be considered as a binary event, based on the date that the company filed for bankruptcy. The exact definition of the bankruptcy variable that will serve as the prediction target often varies in the literature. Our definition is detailed in Section 2.3.3.

2.1.2 Accounting-based input variables

A company is a complicated entity whose financial situation may be influenced by a plethora of factors, such as macroeconomic trends, political events, popular opinion, or even pandemic events. While all of these factors may be relevant to the probability of a company's bankruptcy, capturing such information in terms of data is difficult, or even impossible.

In order to apply a statistical bankruptcy prediction model to the space of all companies, it would need to be based on a shared data source, for which all companies have both available and consistent data. One very reliable source for such information is the publicly available

annual financial statements of the companies. While some technical and qualitative nuances may be left out of the yearly accounting reports, this data source provides excellent and consistent data thaxt offers reliable insights into many of the aspects of a company which one would find relevant to the possibility of a bankruptcy.

As is standard practice in financial analysis, one often prefers to evaluate different ratios of accounting variables, rather than the raw account entries themselves. These transformations help normalize the data, making companies easier to compare across size, time, and industry. This principle is typically carried over into bankruptcy prediction models, where one generally uses financial ratios over the raw entries (Wahlstrøm et al., 2020).

Three additional strengths of financial ratios are highlighted by Agarwal and Taffler (2008): A single year's accounting numbers capture an accumulation of company performance over time, the double-entry system of accounting helps assure consistency, and finally, loan covenants are often based on accounting numbers.

Much research has been done to determine which sets of financial ratios and accounting data provide the most explanatory power for predicting bankruptcies. Examples of such works include Bellovary et al. (2006), Kumar and Ravi (2007), and the seminal work of Beaver (1966). These works are aggregated by Wahlstrøm et al. (2020) to make up a set of 155 different financial ratios and variables, which will make up the dataset which this thesis is based on, described in Section 2.3. The complete list of features is given in Appendix A.

2.1.3 The unbalanced data problem

One inherent property of bankruptcies, is that they are relatively rare events. As we will see in Section 2.3, only 1.514% of our considered financial statements were followed by a bankruptcy (according to our binary event definition specified in Section 2.3.3). For learning-based classification and prediction models, this often imposes difficulties, as the loss functions on which the learning process is based (described in Section 3.1) often tends to become biased. This is known as the *unbalanced data problem*, and is the subject of continuous research (Somasundaram and Reddy 2016; Krawczyk 2016; Kotsiantis et al. 2005).

While this is a very relevant problem to both our application and many others, we will generally consider detailed handling of this problem beyond the scope of this thesis. As will be seen throughout Chapters 6 and 7, we will build on the work of Wahlstrøm et al. (2020), focusing on improving performance on an idealized dataset, where the unbalanced data problem is circumvented by balancing the dataset through sampling. Such sampling obviously imposes a bias of itself, which we will study by performing tests on the unbalanced dataset. The sampling technique deployed is detailed in Section 5.2.

2.1.4 Model interpretability

In the aforementioned contexts where bankruptcy prediction models are applied, the statistical performance of the models are not necessarily the only important aspect. One would often also want to learn why and how the model made specific predictions, known as *model interpretability*, which may help guide decision making processes or reveal errors. Indeed, practitioners often prefer simpler, more interpretable models over more complex ones, even if they perform better (Jones et al., 2017). In the case of for instance interest rate decisions, providing reasonable decision explanations often become a necessary requirement. Stricter legal requirements, such as Article 13-15 of EU's GDPR (Council of European Union, 2014), also become a strong motivator for good model interpretability.

The importance of model interpretability has proven to be a major motivator behind particular model choices in both finance in general and bankruptcy prediction applications specifically, gearing preference towards simpler, although more interpretable solutions, such as linear models (Jones et al., 2017). The ability to perform accurate model interpretability is then important to consider before introducing more complex models, such as tree-based models or neural networks, into this context.

2.2 Previous work

As mentioned in Section 2.1, company bankruptcy prediction has been a historically popular topic of analysis, dating back to FitzPatrick (1932). Over the years, the topic has been continuously revisited in the literature along with corresponding innovations in statistical methodologies and increased data availability. Some recent works are comprehensively reviewed in Bellovary et al. (2006) and Kumar and Ravi (2007).

The earlier models used for bankruptcy prediction typically used discriminant analysis, such as in Altman (1968), applied on U.S. listed companies. However, logistic regression methods, first introduced by Martin (1977) and Ohlson (1980) (also developed on the listed U.S. companies) quickly became the model of choice. They are often preferred due to both less restrictive assumptions, ease of interpretation and strong predictive performances (Wahlstrøm et al., 2020).

However, logistic regression sees a few drawbacks, namely that it is very sensitive to outliers, missing values and multicollinearity (Balcaen, 2006). In the case of bankruptcy prediction, the financial ratios used as features typically share numerators or denominators, making the case of multicollinearity especially problematic. This has motivated extensive research into feature selection methods for optimal sets of financial ratios to use as features, which is the main focus of Wahlstrøm et al. (2020).

For later years, machine learning models such as neural networks and random forests have become more popular (Alaka et al. 2017; Kumar and Ravi 2007). These models can capture more complicated, non-linear relationships without making error distribution assumptions, and was found by Alaka et al. (2017) to significantly outperform logistic regression. Even more recently, Zięba et al. (2016) deployed the gradient boosting framework XGBoost (Chen and Guestrin, 2016). These methods also typically leverage more input variables (often 10 or more, as opposed to less than 5 typically used by the earlier models).

For timeseries structuring of the bankruptcy prediction problem, the literature is more sparse. Some investigations into such a formulation of the problem include Kahya and Theodossiou (1999), who use cumulative sums (CUMSUM) model on a set of 150 U.S. listed firms, and Arora and Saini (2013), who use an adaptive neuro-fuzzy inference system (ANFIS). The lack of research in this area may be due to restrictions related to data availability, as longer timeseries naturally require more data to reliably establish.

This thesis will largely be an extension of the works of Wahlstrøm et al. (2020), which use the same extensive Norwegian SME dataset to do bankruptcy prediction, although with a focus on feature selection methods, examining the effects of different feature subsets amongst the 155 input features. They implement logistic regression and neural network models, which are popular throughout the literature. An interesting finding was that the model performance generally increases in terms of number of features included, although the gain from feature additions started decreasing after 15 features, and plateaued even more after around $25 - 30$ features.

An important note when comparing bankruptcy prediction works, is that the resulting performance metrics are heavily dependent on the underlying dataset, both in terms of historical period (some periods are more economically irregular, such as the global financial crisis of 2007-08), sampling considerations (such as which country or economy the data is sourced from), and other factors (such as whether the dataset was balanced or not) (Filipe et al., 2016). This makes both previous and future works harder to compare, and each method generally has to be implemented and tested for each new dataset, in order to facilitate accurate comparisons of models. Motivated by this, we will focus our comparison and reimplementations to the results of Wahlstrøm et al. (2020), as they use the same dataset.

2.3 Dataset

In this section, we will describe in detail the dataset used throughout the thesis. We will define both the source and span of the data, our chosen definition of the bankruptcy target

variable, a discussion of the financial ratio features used, as well as a description of the NACE-system and how it was used to construct a novel categorical feature, which will be used to provide additional information about the industry of a company.

2.3.1 Norwegian accounting data

The dataset used throughout this thesis is an extensive dataset comprised of 987065 annual financial statements from 178812 Norwegian limited liability (small and medium sized) companies (after data exclusion, see Section 2.3.2). The data considered spans a period of 11 years, from 2006 to 2017. As our bankruptcy target variable definition involves a 3 year window (see Section 2.3.3), this limits the financial statement data to the period 2006-2014. The bankruptcy information is sourced from bankruptcy filing data, and combined with the financial statements to produce the target bankruptcy variable. All of this data is publicly available, and gathered from the Norwegian government agency Brønnøysund Register Centre.

2.3.2 Excluded data

As was done in Wahlstrøm et al. (2020), our dataset is focused on small and medium-sized enterprises (SMEs). They are defined as companies with turnover less than USD 50 million, which follows the definition of Bank of International Settlements (2017). In order to exclude inactive companies, we also require companies to have total assets of at least 500000 NOK, as done by Wahlstrøm et al. (2020) and Bernhardsen and Larsen (2007). The dataset also excludes all publicly traded companies and all consolidated financial statements, the latter as we focus our study on company level.

As is also the convention in previous literature, companies in the industries of *'Financial and insurance activities'*, *'Real estate activities'*, *'Electricity and gas supply'*, and *'Water supply, sewerage, waste'* (based on their NACE-code, see Section 2.3.5) are also excluded (Mansi et al., 2010). Companies without any reported NACE-code are also excluded.

2.3.3 Bankruptcy target variable definition

As the bankruptcy prediction models would typically require a binary target variable, one has to associate each of the financial statements with an indicator variable of whether or not the company went bankrupt. This is not necessarily trivial, as there is often a time gap between the economic default of a company, and the official registration of a filed bankruptcy - up to 3 years, according to the UK study of Hernandez Tinoco and Wilson (2013). Indeed, in our dataset, 99.2%, 85.9%, and 22.7% of bankrupt companies filed for bankruptcy within one, two, and three years from their final financial statement, respectively.

Based on this, we associate a financial statement with a bankruptcy target variable of 1 if it is the last financial statement of the respective company, and the company (or its creditors, through a court), has filed it for bankruptcy within three years from the date of this last financial statement. All other financial statements are categorized as non-bankrupt, i.e. 0 in the target variable. This is the same definition used by Wahlstrøm et al. (2020), and is consistent with e.g. the SEBRA model in Bernhardsen and Larsen (2007). The resulting number of non-bankruptcies and bankruptcies per year following this definition can be seen Table 2.1.

While well defined in the above sense, we note that a binary bankruptcy target variable is an inherently noisy one. For instance, some companies may be able to continue to deliver financial statements while they are in deep financial distress (where other companies would file for bankruptcy), due to external factors. In our binary target variable definition, financial statements of such essentially bankrupt companies would be considered the same as for perfectly healthy companies, until their very last financial statement. Another source of noise would be strategic bankruptcy, where some otherwise healthy companies may file for bankruptcy in order to for instance break undesired contracts (Mansi et al., 2010).

With these considerations, perfect prediction of the binary bankruptcy target variable is evidently infeasible. Our chosen target variable definition is, however, both consistent and well-defined.

Year	Non-bankrupt	Bankrupt
2006	93510	1334
2007	100744	2190
2008	103935	1918
2009	104614	1764
2010	106326	1541
2011	109006	1605
2012	113403	1395
2013	118068	1629
2014	122519	1564

Table 2.1: Breakdown of number of non-bankrupt and bankrupt financial statements per accounting year, following the definition specified in Section 2.3.3. Note the high level of class imbalance.

2.3.4 Financial ratios as features

As described in Section 2.1.2, financial statement ratios serve as consistent and reliable features for bankruptcy prediction models. Wahlstrøm et al. (2020) has assembled a set of 155 different input variables used throughout the literature, which will serve as the foundation for our bankruptcy prediction dataset. These are listed in Appendix A.

Only four of the input variables are non-ratios. These consist of two binary variables, and $age\ in\ years$ and $total\ assets$. As a final note, for ratio features where the denominator is zero, the feature is also set to zero, following Wahlstrøm et al. (2020). While potentially somewhat problematic, we will be following this notion for consistency.

2.3.5 NACE system

The Statistical Classification of Economic Activities in the European Community, commonly referred to as *NACE* (the abbreviation stemming from the French form), is the standard industry classification system used in the EU, which has been adopted and used in Norway (Council of European Union, 2006). The system is built on a hierarchical code of one letter and up to up to five digits, where each additional digit refers to an additional level of specification (i.e. the letter code is the most general one, see Table 2.2 for an example).

NACE code	Name of level
F	Construction
41	Construction of buildings
41.1	Development of building projects
41.10	Development of building projects
41.101	House building cooperative

Table 2.2: Example of the NACE code system for a property development company.

A breakdown of the number of bankruptcies and non-bankruptcies in our dataset for each of the highest level NACE-codes is shown in Table 2.3. For our purposes, the NACE system will be used to categorize companies into their respective industries, both in order to ensure more comparative sampling, and as a feature during training (see Chapters 5 and 6).

NACE code	Non-bankrupt	Bankrupt	Total
A	21800	254	22054
B	7302	46	7348
C	80585	1295	81880
F	188481	3360	191841
G	254787	5414	260201
H	56759	749	57508
I	40088	1197	41285
J	51254	472	51726
M	141073	888	141961
N	50638	745	51383
O	338	0	0
P	10205	88.0	10293
Q	38360	109.0	38469
R	15865	180.0	16045
S	14590	143.0	14733

Table 2.3: Number of bankruptcies and non-bankruptcies for each of the top level NACE codes. Note that these are numbers of individual financial statements in the dataset, rather than unique companies.

Engineered NACE code feature

One of the contributions of this thesis is the use of the NACE code as a categorical feature, in order to provide the model with potentially relevant information about the company's industry. However, the lowest level of the NACE code system is often very specific, including only a few companies.

As we will see in Chapter 3, different ways of handling categorical features will often require multiple samples within each category in order to work appropriately. Motivated by this, we leverage the hierarchical nature of the NACE system in order to produce a feature $nace\_code$ that uses the NACE level such that a sufficient amount of companies are included in each NACE category. This ensures that maximum informativeness is kept, while still including sufficient samples to be used.

In specific, we require at least 500 samples (a sufficient number, given heuristic experiments) in each level for the level to be included, sequentially traversing upwards in the hierarchy until the requirement is met. Note that this requirement is for number of samples (i.e. yearly financial statements), rather than number of unique companies. For those companies that does not satisfy the requirement even for the highest level, a category $other$ is assigned instead of the NACE code.

This procedure reduces the number of unique NACE codes from 728 to 306. Of the 178812 total companies, 2955 falls into the artificial $other$ category.

2.3.6 Key assumptions

As mentioned in Section 2.1.2, there are multiple factors economical factors that may effect the financial future and indeed solvency of a company. When using our temporally distributed financial statements, we are making several assumptions regarding the data. First and foremost, we are assuming some degree of homogeneity in time, ignoring macro economical and other external factors. Secondly, we assume that the reported financial data is indeed correct, which may not always be the case (in the case of for instance data entry errors or fraud). Other assumptions regarding i.e. the bankruptcy target variable was described in Section 2.3.3.

While the homogeneity assumption may seem unrealistic, we will take sufficient care in order to minimize its potentially adversarial effects. This is especially important in the experimental

designs, and our specific choices and considerations in order to mitigate these effects will be highlighted in Chapter 5.

Chapter 3

Methods for bankruptcy prediction

In this chapter, we will describe the theory and methodology behind the methods we will consider for performing bankruptcy prediction. In Section 3.1, we will first formulate the general predictive learning problem as a function estimation problem, mainly following the formulation of Hastie et al. (2001), and describe some general numerical optimization techniques for estimating such functions.

In Sections 3.2-3.4, we will then specify some static (i.e. non-time dependent) model frameworks that solve this problem. We first describe the theoretical foundation behind each method, and then briefly discuss them in the context of bankruptcy prediction. Finally, in section 3.5, we will propose two neural network frameworks, namely the RNN and LSTM extension, for solving a timeseries formulation of the prediction problem.

3.1 Function estimation

In the predictive learning problem, the general goal is to estimate some mapping between a set of random "explanatory" variables $\mathbf{x} \in \mathbb{R}^p$ (often called *features*) and a corresponding random *target* variable y, which we assume to be independent and identically distributed according to some unknown distribution. Typically, we have a sample of n such pairs of "training" data points sampled from this distribution, which comprises a *dataset* $\mathcal{D} = \{(\mathbf{x}_i, y_i), i = 1, \ldots, n\}$, $\mathcal{D} \subset \mathcal{S}$, with $\mathcal{S}$ being the joint distribution of all possible pairs $\{\mathbf{x}, y\}$. The desired mapping can be formulated as a function $F^*(\mathbf{x}) = y$, for which we want to find an approximation $\hat{F}(\mathbf{x})$.

We then specify some loss function $L(y, F^*(\mathbf{x}))$, with the property that its expectation is minimized by the function $F^*(\mathbf{x})$ over all $(\mathbf{x}, y) \in \mathcal{S}$:

$$F^* = \arg\min_F E_{(\mathbf{x},y)\in\mathcal{S}} L(y, F(\mathbf{x})). \tag{3.1}$$

The particular loss function $L(y, F)$ may be chosen according to the specifics of the problem (e.g. the characteristics of y), although a desirable property is convexity and differentiability (as we will see in Section 3.1.1). Examples of loss functions are the squared-error $(y-F)^2$ and absolute error $|y - F|$ when $y \in \mathbb{R}$, and the negative binomial log-likelihood $yF - \log(1 + e^F)$ and the binary cross entropy $y \log(F) + (1 - y) \log(1 - F)$ when $y \in \{0, 1\}$ (i.e. binary classification/prediction) (Hastie et al., 2001).

To allow for a meaningful approximation of $F(\mathbf{x})$, a popular procedure is to restrict it to a class of parameterized functions $F(\mathbf{x}|\mathbf{P})$, often called a *model*, where $\mathbf{P} = P_1, P_2, \ldots$ is a finite set of parameters. The specific choice of model is of great importance, and

typically encapsulates some assumptions one may have about the unknown data generating distribution.

3.1.1 Numerical optimization

After choosing a parameterized model $F(\mathbf{x}|\mathbf{P})$, we can formulate the function optimization (3.1) as an optimization in parameter space, seeking the optimal set of parameters $\mathbf{P}^*$:

$$\mathbf{P}^* = \arg\min_{\mathbf{P}} E_{(\mathbf{x},y)\in\mathcal{S}} L(y, F(\mathbf{x}|\mathbf{P})). \tag{3.2}$$

In practice, we do not have access to the full joint distribution $\mathcal{S}$, and therefore have to find some approximation using our dataset sample $\mathcal{D}$.

Depending on the choice of $F(\mathbf{x}|\mathbf{P})$, numerical optimization is often required to solve (3.2). To allow for this, we can express the solution $\mathbf{P}^*$ in an incremental form

$$\mathbf{P}^* = \sum_{i=1}^{I} \mathbf{p}_i, \tag{3.3}$$

starting from an initial guess $\mathbf{p_0}$ with I successively estimated "steps" $\mathbf{p_i}, i = 1, \ldots, I$. In this setting, an additional increment $\mathbf{p}_j$ added to the parameter set is typically an incremental "step" towards a lower expectation of the loss function L, given the current parameter set $\sum_{i=1}^{j-1} \mathbf{p}_i$.

Gradient descent

One simple, yet often deployed technique for computing the steps $\mathbf{p}_i$ is *gradient descent*, which makes use of the gradient of the expected loss $\mathbf{g}_i$, given the previous $i-1$ steps:

$$\mathbf{g}_i = \{g_{ji}\} = \left\{ \left[\frac{\delta \mathbf{\Theta}(\mathbf{P})}{\delta P_j} \right]_{\mathbf{P}=\mathbf{P}_{i-1}} \right\}$$

where $\mathbf{g} = [g_1, g_2, \ldots, g_m]$ (with corresponding parameter vector $\mathbf{P}$), j refers to the jth component of $\mathbf{g}$, and

$$\mathbf{\Theta}(\mathbf{P}) = E_{(\mathbf{x},y)\in\mathcal{D}} L(y, F(\mathbf{x}|\mathbf{P})).$$

The next step $\mathbf{p}_i$ is then set as

$$\mathbf{p}_i = -\alpha_i \mathbf{g}_i$$

for some step length $\alpha_i \in \mathbb{R}$ (often called *learning rate*).

Intuitively, gradient descent takes a "step" in the direction of the steepest descent of the expected loss in parameter space, where the length of the step α_i can either be some constant for all i, or adaptively estimated at each step, via e.g. a line search for the optimal value.

For convex functions, gradient descent can be shown to converge to a global minimum. While also applicable to non-convex functions, they are not guaranteed to converge to the global minimum and may get "stuck" in stationary points or local minima (Hastie et al., 2001).

While adding more advanced features to the optimization process (such as second derivatives or momentum) can improve both estimation and efficiency, the general idea of step-wise iteration according to some gradient is often central.

Stochastic gradient descent

The gradient update in (3.25) is computed over the whole dataset $\mathcal{D}$. If the dataset becomes large, the required computation time and memory for each iteration may become high, potentially causing practical issues. Indeed, for very large datasets, the computations may not even fit entirely in the memory of the computer. Another issue is that using the full dataset often shows worse generalization on unseen data, although the underlying reasons for this remains largely unknown (Hoffer et al., 2017).

A solution to this problem is *stochastic gradient descent*, where each training iteration is instead performed on a random subset of samples, $\mathcal{D}_k \in \mathcal{D}$, called a *minibatch*. The size (i.e number of samples) of each minibatch is a tuning parameter. This allows the dataset to be processed in more manageable batches, and allows the process to be parallelized, significantly speeding up computing (Sra et al., 2012).

One way to view stochastic gradient descent is as a way to increase the number of iterations performed, at the cost of the number of samples used in each update. As the full dataset $\mathcal{D}$ is just a sample from $\mathcal{S}$, training using smaller, more frequent samples does not necessarily weaken performance.

3.1.2 Function regularization

A recurring theme in predictive learning function estimation is the trade-off between optimally fitting the function F to the data sample $\mathcal{D}$ (i.e. finding the optimal solution for (3.1)), while still maintaining good predictive performance on the full data distribution $\mathcal{S}$. The ability to do the latter is often referred to as *generalization* (i.e. the model's ability to *generalize* the properties in the data), while the failure to do so (while having a good fit on the data in $\mathcal{D}$) is called *overfitting* the data in $\mathcal{D}$. This often occurs when the chosen function class has too high of a flexibility, which in the case of a parameterized function F leads to *overparameterization*. To counteract this, one typically deploys some restrictions to the model function, which is called *model regularization*.

The form of regularization is highly dependent on the particular model, but generally expresses some prior belief of a certain type of smooth behaviour in the data. Regularization is often deployed in the form of a *penalty function*, penalizing certain behaviours during the numerical optimization, or by using a separate *validation dataset* to continuously evaluate overfitting during training (see Section 5.3). We will detail the relevant model specific regularization techniques in their respective model sections later in this chapter.

3.2 Logistic regression

A logistic model is a model that uses the *logit*, or the *log-odds*, to model the probability of a binary target variable $y \in \{0, 1\}$. The logit is defined as the logarithm of the odds:

$$\text{logit}(y) = \log\left(\frac{s}{1-s}\right),$$

where $s \in (0, 1)$ is the probability of $y = 1$, and the logarithm is the natural logarithm (although other bases for the logarithm can be used). This can then be used to perform binary prediction according to some prediction rule, typically classifying the target as 1 if $s > \tau$ for some threshold τ (often set to 0.5), and classifying as 0 otherwise.

A popular logistic model is logistic regression, where we assume a linear relationship between $\text{logit}(y)$ and an observation vector $\mathbf{x} \in \mathbb{R}^p$:

$$\text{logit}(y) = \alpha + \beta\mathbf{x}, \tag{3.4}$$

where $\beta^\intercal \in \mathbb{R}^p$ are the regression coefficients, and $\alpha \in \mathbb{R}$ the bias term. To obtain the conditional probability estimates of y, one can invert (3.4) to obtain

$$P(y=1|\mathbf{x}) = \frac{e^{\alpha+\beta\mathbf{x}}}{1+e^{\alpha+\beta\mathbf{x}}}$$

which corresponds to the modelling function $F(\mathbf{x}|\mathbf{P})$ with regression parameters $\mathbf{P} = (\alpha, \beta)$.

Following the process outlined in Section 3.1, we now seek a convex and differentiable loss function L which tends to zero when $F \to 1$ and $y = 1$, and when $F \to 0$ and $y = 0$. The two following functions satisfy these criteria, respectively:

$$\begin{aligned} -\log(F(\mathbf{x}|\alpha,\beta)), &\quad y=1 \\ -\log(1-F(\mathbf{x}|\alpha,\beta)), &\quad y=0. \end{aligned}$$

These can be combined to the *log loss function*

$$L(y, y') = -y\log(y') - (1-y)\log(1-y').$$

where $y' = F(\mathbf{x}|\alpha, \beta)$, which is convex and differentiable. The regression parameters (α, β) can then be estimated by (3.2) (approximated over the dataset D):

$$\alpha, \beta = \arg\min_{\alpha,\beta} \left\{ \sum_{y,\mathbf{x}\in\mathcal{D}} y\log(F(\mathbf{x}|\alpha,\beta)) - (1-y)\log(1-F(\mathbf{x}|\alpha,\beta)) \right\}. \tag{3.5}$$

Specifically, maximum likelihood estimation is often deployed, using the conditional likelihood $P(y|\mathbf{x})$ (Hastie et al., 2001).

An important note is that logistic regression assumes little to no multicollinearity (i.e. correlation between features). In the presence of multicollinearity, the attributed parameter gains may be distributed arbitrarily amongst the correlated features (Hastie et al., 2001).

Regularization for logistic regression

In the context of logistic regression, the linear model hypothesis (3.4) already imposes some regularization to the model's flexibility. Nonetheless, overfitting may still occur. A commonly deployed technique is then to impose some penalty on large values of parameters β, often by introducing a penalty function $\phi(\lambda)$.

In practice, this modifies the optimization process (3.5) with the additional penalty function:

$$\alpha, \beta = \arg\min_{\alpha,\beta} \left\{ \sum_{y,\mathbf{x}\in\mathcal{D}} y\log(F(\mathbf{x}|\alpha,\beta)) - (1-y)\log(1-F(\mathbf{x}|\alpha,\beta)) - \phi(\lambda|\alpha,\beta) \right\}. \tag{3.6}$$

There are typically three types of penalty functions $\phi(\lambda)$ applied to logistic regression, namely the L_1 penalty $\phi(\lambda|\alpha,\beta) = \lambda\sum_{j=1}^{p}|\beta_j|$, the L_2 penalty $\phi(\lambda|\alpha,\beta) = \lambda\sum_{j=1}^{p}\beta_j^2$, and the Elastic Net $\phi(\lambda|\alpha,\beta) = \sum_{j=1}^{p}(\lambda_1|\beta_j| + \lambda_2\beta_j^2)$. Note that we typically do not penalize the intercept term.

In all cases, the hyperparameter λ acts as a controllable regularization strength, where higher values will lead to stronger regularization (and in the case of the elastic net, controls the trade-off between the L_1 and L_2 components). The specifics of each of the methods varies across applications, however in general L_1 penalty (and consequently elastic net, depending on the values of λ_1 and λ_2) tends to result in some or more $\beta_j = 0$, thus performing an implicit feature selection procedure, which is often desirable (such as in Wahlstrøm et al. (2020)).

3.3 Gradient Tree Boosting

Another popular family of function classes are the gradient boosting machines, first described in Friedman (2000). This technique produces an ensemble of base functions ("weak learners"), iteratively built in an stage-wise manner. The base functions most often deployed (indeed in all the gradient boosting methods we will consider) are decision trees, and we will start this section with a description of this prediction function.

3.3.1 Decision trees

While there are multiple variations of the decision tree model framework, we will focus on the procedure described in Hastie et al. (2001), which builds on the methodology introduced by Breiman et al. (1984), often referred to as Classifcation And Regression Trees (*CARTs*).

A decision tree is a model built by partitioning the feature space $\mathbb{R}^m$ into J disjoint regions $R_1, R_2, \ldots, R_J$, represented by tree nodes. The regions are obtained by sequentially considering binary splits at a single feature x_k for some splitting value s, until a terminal node is reached, for which a value is assigned to the prediction (see Figure 3.1). This results in a tree structure, as seen in Figure 3.1. We can formulate the decision tree prediction function as

$$h(\mathbf{x}|\{c_j, R_j\}_{j=1}^{J}) = \sum_{j=1}^{J} c_j \mathbb{1}_{\{\mathbf{x}\in R_j\}}, \tag{3.7}$$

where J is the number of terminal regions, and c_j determines the prediction value assigned to the region R_j. The parameters c_j are set to minimize the loss within each terminal node:

$$c_j = \arg\min_{c_j} L(y, c_j)|\mathbf{x} \in R_j. \tag{3.8}$$

In a regression context, with L a the mean-squared error, this minimization results in c_j being set as the mean of the target values y within each terminal node. Similarly, for binary predictions with L being the log-loss, c_j becomes the proportion of positive target values y within R_j.

We then want to determine the terminal regions R_j such that the specified loss function L is minimized for all $(y_i, \mathbf{x}_i) \in \mathcal{D}$. However, considering all possible feature space partitions becomes infeasible (Hastie et al., 2001). Therefore, we adopt a greedy algorithm that recursively builds the tree by considering the current best splitting feature x_k and corresponding best splitting value s, that produces the greatest reduction in L:

$$k, s = \arg\min_{k,s} \left[\min_{c_1} \sum_{x_i \in \hat{R}_1(k,s)} L(y_i, c_1) + \min_{c_2} \sum_{x_i \in \hat{R}_2(k,s)} L(y_i, c_2) \right] \tag{3.9}$$

where $\hat{R}_1(k, s)$ and $\hat{R}_2(k, s)$ are the half-planes such that

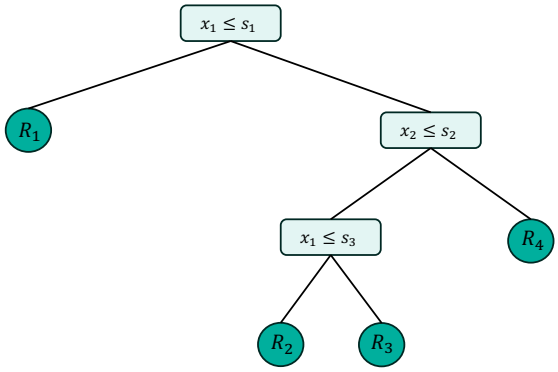
$$\hat{R}_1(k, s) = \{\mathbf{x} : x_k \leq s\},$$
$$\hat{R}_2(k, s) = \{\mathbf{x} : x_k > s\}.$$

The hat indicates that the regions $\hat{R}$ are not necessarily terminal regions. When the best greedy split is found, the process is repeated on each of the regions $\hat{R}_1, \hat{R}_2$ in a recursive manner, until a specified tree size is reached. We have then obtained our terminal region set $\{R_j\}_{j=1}^{J}$, and the corresponding values c_j can be determined according to (3.8), completing the parameter set $\{c_j, R_j\}$ in (3.7).

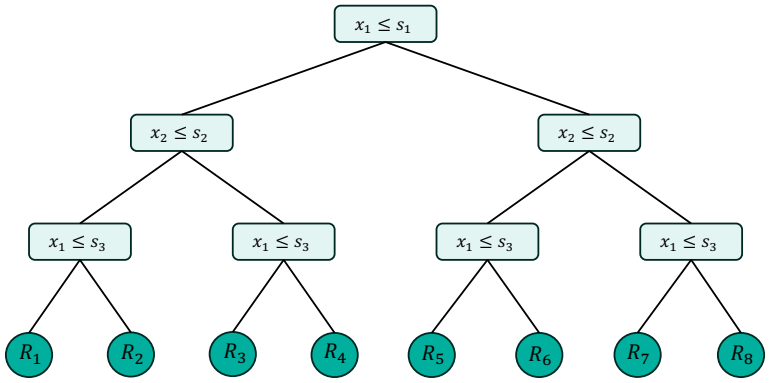
The question is then how many nodes the tree should have. By letting the tree grow too large, the regions R_j may partition the feature space too finely, which results in overfitting.

Too see this, let the number of regions J equal the number of training samples n. We may then perfectly fit the training data, as each training sample will have its own terminal node. Clearly, this will not generalize well on unseen data, and we thus need to restrict the size of each tree.

On the other hand, too shallow trees may restrict the model's flexibility too much, leaving it unable to learn more complicated relations in the data. Following this argument, we can control the model's flexibility with the tree size, which is a tuning parameter.



(a) An example decision tree built using CART.



(b) An example oblivious decision tree; note that the same splitting criterion is applied throughout each level.

Figure 3.1: Decision trees assigns values to a feature vector (x_1, x_2) by considering a series of binary operations at each node. Here, the process goes down the left edge if the inequality in the node holds true, and down the right if not, until a terminal node region R_j is reached.

Cost complexity pruning

Hastie et al. (2001) suggests adaptively choosing the tree size based on the data. In specific, using a tree pruning strategy called *cost complexity pruning*.

First, we grow a tree T_0 until some (relatively large) minimum node size is reached, likely to overfit the data. The idea is then to sequentially prune T_0 (i.e. collapsing internal nodes) until some criterion is met. We define a subtree $T \subset T_0$ to be any tree obtainable by pruning

T_0 (i.e. collapsing any number of its internal nodes). The cost-complexity criterion, which we want to minimize, is then defined as

$$C_\lambda = \sum_{l=1}^{|T|} L_l(T) + \lambda|T|, \tag{3.10}$$

where $L_l(T)$ denotes the loss for sub-tree T in node l, $|T|$ is the number of terminal nodes in T and $\lambda \geq 0$ is a tuning parameter. Conceptually, λ acts as a regularization parameter punishing large trees (i.e. many terminal nodes): Large values of of λ results in smaller trees T_λ, and vice versa.

Each optimal subtree T_λ is found by *weakest link pruning*: We successively look for which internal node that collapsing produces the smallest per-node increase in $\sum_l L_l$, continuing until the single node root tree is reached. This produces a finite sequence of subtrees, which can be shown to always contain T_λ (Hastie et al., 2001).

Oblivious trees

As the tree building procedure described above always considers the best greedy split (3.9), it may grow a tree of any shape, often resulting in asymmetrical trees (see Figure 3.1a). However, imposing some constraints on the tree building process may result in some desirable properties.

Oblivious trees are trees where we require every split that is on the same level in the tree (i.e. equal distance to the root node in terms of number of edges) to use the same split criterion, on the same feature. Thus, the optimization (3.9) is performed across the whole level at once, rather than being done individually at each node. An example of such a tree is shown in Figure 3.1b.

This results in symmetrical and balanced trees, which in turn speeds up execution at testing time (Kohavi and Li, 1995). It has also been shown that oblivious trees may help prevent overfitting, especially when they are used in model ensembles (Prokhorenkova et al., 2018).

Limitations of decision trees

The tree representation of a decision tree allows for an intuitive interpretation of the learned model, as one can follow the tree's "decision process" for each individual feature and splitting value, as is examplified in Figure 3.1. However, a model based on a single decision tree has several limitations. A problem stated in Hastie et al. (2001) is their high variance, caused by the hierarchical nature of the process; an error in the top split of the tree will be propagated downwards to all splits below it. Another issue is the lack of smoothness in the prediction surface, caused by the disjoint regions. This problem is especially apparent in a regression setting, where one would typically prefer the prediction surface to have local smoothness.

3.3.2 Gradient boosting machines

Gradient boosting machines are a model class built as an ensemble of multiple base predictors. As opposed to many other parameterized models (such as logistic regression), the gradient boosting machine ensemble is obtained by optimizing in function space, rather than parameter space. We first formulate $F^*(\mathbf{x})$ in (3.1) as

$$F^*(\mathbf{x}) = \sum_{i=0}^{I} f_i(\mathbf{x}),$$

where each $f_i(\mathbf{x})$ are incremental weak prediction functions ("boosts"), starting from an initial guess $f_0(\mathbf{x})$. Analogously to Section 3.1.1, we can estimate the consecutive functions using the negative gradient

$$f_i(\mathbf{x}) = -\rho_i g_i(\mathbf{x}) \tag{3.11}$$

for some step length ρ_i. Now, the gradient $\mathbf{g}_i(\mathbf{x})$ is defined as

$$\mathbf{g}_i(\mathbf{x}) = \left[\frac{\delta \mathbf{\Theta}(F(\mathbf{x}), y)}{\delta F(\mathbf{x})}\right]_{F(\mathbf{x})=F_{i-1}(\mathbf{x})}$$

with

$$\mathbf{\Theta}(F(\mathbf{x}), y) = E_{y \in \mathcal{S}} L(y, F(\mathbf{x})|\mathbf{x}).$$

and

$$F_{i-1}(\mathbf{x}) = \sum_{j=0}^{i-1} f_j(\mathbf{x}) \tag{3.12}$$

(note the change in the expectation compared to (3.2)).

Assuming sufficient regularity for allowing interchanging differentation and integration, one arrives at the expression

$$g_i(\mathbf{x}, y) = E_y \left[\frac{\delta L(y, F(\mathbf{x}))}{\delta F(\mathbf{x})}|\mathbf{x}\right]_{F(\mathbf{x})=F_{i-1}(\mathbf{x})}. \tag{3.13}$$

Friedman (2000) then states that for a finite data sample $\mathcal{D} \subset \mathcal{S}$, this method breaks down, as one is unable to estimate $E_y[\cdot|\mathbf{x}]$ (especially so for unseen data, i.e. values of $\mathbf{x}$ not in $\mathcal{D}$). This can be solved by imposing smoothness on the solution ("borrowing strength from nearby data points"), which again can be done by parameterizing the functions and doing parameter optimization. Specifically, Friedman (2000) proposes a "greedy-stagewise" approach, iteratively updating the parameter set with

$$(\beta_m, \mathbf{a}_m) = \arg\min_{\beta, \mathbf{a}} \sum_{i=1}^{n} L(y_i, F_{m-1}(\mathbf{x}_i) + \beta h(\mathbf{x}_i|\mathbf{a})), \quad m = 1, 2, \ldots, M, \tag{3.14}$$

where

$$F(\mathbf{x}|\{\beta_m, \mathbf{a}_m\}_{m=1}^M) = \sum_{m=1}^{M} \beta_m h(\mathbf{x}|\mathbf{a}_m), F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \beta_m h(\mathbf{x}|\mathbf{a}_m). \tag{3.15}$$

Here, the function $h(\mathbf{x}|\mathbf{a})$ is the generic base ("weak") prediction function, parameterized by the parameters $\mathbf{a} = \{a_1, a_2, \ldots\}$ (note that these individual parameters varies for different $\mathbf{a}_m$), and β can be thought of as a step size (somewhat analogous to the learning rate discussed in Section 3.1). Note the difference between this *stagewise* strategy and the stepwise strategy proposed in Section 3.1.1; while the stepwise strategy additively adds parameter terms, this strategy readjusts previously entered terms when new ones are added.

For practical choices of the loss $L(y, F)$ and base function $h(\mathbf{x}|\mathbf{a})$, the solution to (3.14) is often difficult to obtain. However, we can view the function $\beta_m h(\mathbf{x}|\mathbf{a}_m)$ as the best greedy step towards $F^*(\mathbf{x})$ given a current approximator $F_{m-1}(\mathbf{x})$, similar to gradient descent in

Section 3.1.1. In this context, we constrain our step "direction" $h(\mathbf{x}|\mathbf{a_m})$ to be a member of the class of base learners $h(\mathbf{x}|\mathbf{a})$. This leads to the unconstrained negative gradient

$$-g_m(\mathbf{x_i}) = -\left[\frac{\delta L(y_i, F(\mathbf{x_i}))}{\delta F(\mathbf{x_i})}|\mathbf{x}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})} \tag{3.16}$$

for $(x_i, y_i) \in \mathcal{D}$, analogous to (3.13), but now in our data sample space $\mathcal{D}$. While this gradient gives the best gradient descent direction $-\mathbf{g}_m = \{-g_m(\mathbf{x}_i)\}_i^N$, it cannot be generalized to other values for $\mathbf{x}$ as it is only defined for data in $\mathcal{D}$. We can again generalize by looking to the parameter space of the base model $h(\mathbf{x}|\mathbf{a}_m)$, and choosing the parameters $\mathbf{a}_m$ that produces the $\mathbf{h}_m = \{h(\mathbf{x}_i|\mathbf{a}_m)\}_{i=1}^n$ most parallel to $-\mathbf{g}_m$. This corresponds to the $h(\mathbf{x}|\mathbf{a})$ most highly correlated with $-g_m(\mathbf{x})$ over $\mathcal{D}$, and can be obtained from the solution of

$$\mathbf{a}_m = \arg\min_{\mathbf{a},\beta} \sum_{i=1}^{N} (-g_m(\mathbf{x}_i) - \beta h(\mathbf{x}_i|\mathbf{a}))^2. \tag{3.17}$$

In other words, we circumvent the difficult optimization (3.14) by finding the parameters $\mathbf{a}_m$ that results in the least squares approximation $h(\mathbf{x}|\mathbf{a})$ of the gradient $-g_m$. We then use this constrained negative gradient $-h(\mathbf{x}|\mathbf{a}_m)$ in place of $-g_m(\mathbf{x})$ in the gradient descent strategy, giving the approximation update steps

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \rho_m h(\mathbf{x}|\mathbf{a}_m),$$

where ρ_m is the step size, as before.

Conceptually, gradient boosting adds consecutive weak learners to the ensemble by adding the weak learner h_m that best approximates the steepest descent gradient of the loss, given the current ensemble F_{m-1}. In practice, this can be seen as fitting the weak learners to the residuals of the current model. (Friedman, 2000) summarizes this in the following generic gradient boosting algorithm:

Data: Number of trees M, $(\mathbf{x_i}, y_i)_{i=1}^n$
$F_0(\mathbf{x}) = \arg\min_\rho \sum_{i=1}^n L(y_i, \rho)$;
for $m = 1$ *to* M **do**
  $\tilde{y}_i = -\left[\frac{\delta L(y_i, F(\mathbf{x_i}))}{\delta F(\mathbf{x_i})}|\mathbf{x}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}$, for $1, \ldots, n$;
  $\mathbf{a}_m = \arg\min_{\mathbf{a},\beta} \sum_{i=1}^N [\tilde{y}_i - \beta h(\mathbf{x}_i|\mathbf{a})]^2$;
  Set ρ_m;
  $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \rho_m h(\mathbf{x}|\mathbf{a}_m)$;
end

Algorithm 1: Generic gradient boosting (Friedman, 2000)

Gradient boosting with decision trees as base predictors

When using decision trees as base learners in gradient boosting, some considerations can be made to slightly modify Algorithm 1. Recall from Section 3.3.1 that a decision tree $h_m(\mathbf{x})$ can be written as

$$h_m(\mathbf{x}|\{c_{jm}, R_{jm}\}_{j=1}^J) = \sum_{j=1}^{J} c_{jm} \mathbb{1}_{\{\mathbf{x}\in R_{jm}\}},$$

which inserted into the update step (3.15) may be expressed as

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \sum_{j=1}^{J} \phi_{jm} \mathbb{1}_{\{\mathbf{x}\in R_j\}}$$

with $\phi_{jm} = \beta_m c_{jm}$. One can view this update step as adding J different basis functions to F_{m-1}, which one can directly embed into the decision tree coefficient optimization (3.8) as

$$\{\phi_{jm}\}_{j=1}^{J} = \arg\min_{\{\phi_j\}_{j=1}^{J}} \sum_{i=1}^{N} L\left(y_i, F_{m-1}(\mathbf{x}_i) + \sum_{j=1}^{J} \phi_j \mathbb{1}_{\{\mathbf{x}\in R_{jm}\}}\right).$$

As the regions R_{jm} are disjoint, each $\mathbf{x}_i$ will only have one non-zero term in the second sum and one can simplify this to

$$\phi_{jm} = \arg\min_{\phi} \sum_{\mathbf{x_i}\in R_{jm}} L\left(y_i, F_{m-1}(\mathbf{x}_i) + \phi\right).$$

This is the optimal update in each terminal rode region, given the current $F_{m-1}(\mathbf{x})$ (Friedman, 2000).

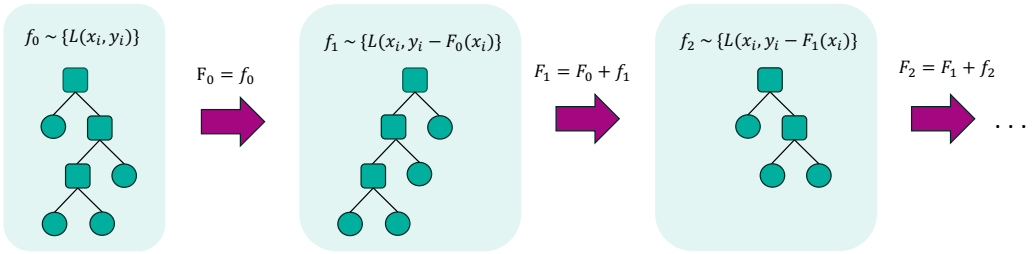


Figure 3.2: A simplified representation of the gradient boosting algorithm (with decision trees as weak learners). In a simplified sense, the ensemble F_m is built by sequentially adding weak learners f_m, which can be thought of to fit the residuals of the previous ensemble, F_{m-1}. The tilde indicates what data the function f_i is based on.

Regularization for gradient boosting trees

Due to the large number potential of parameters and weak learners in the ensemble, gradient boosting trees has a large amount of inherent model flexibility, and thus, potential to overfit. A natural regularization parameter then becomes the maximum number of allowed learners in the ensemble, where a lower number of trees results in reduced model flexibility and thus lowers the risk of overfitting.

Another regularization method involves controlling the contribution of each tree, by scaling the β parameter in (3.15) by an additional factor $0 < \nu < 1$ - often referred to as *shrinkage* (Hastie et al., 2001). This reduces the variation in the individual learners as they are added, and consequently the risk of overfitting (Chen and Guestrin, 2016).

(Hastie et al., 2001) also suggest a regularization method called *subsampling*. Similarly to the Stochastic Gradient Descent discussed in Section 3.1.1, each consecutive weak learner is fit on a subset of the data, according to some sample fraction. While obtaining the same computational advantages as Stochastic Gradient Descent, subsampling often results in more accurate models (Hastie et al., 2001).

Additionally, one can also impose the L_1, L_2 or Elastic Net penalty functions on the parameters, as discussed for logistic regression in Section 3.2.

3.3.3 CatBoost

While the gradient boosting tree methodology described above holds solid theoretical foundation and flexibility, there are numerous extentions of the framework that offer different

variations and optimizations in their implementation, such as XGBoost (Chen and Guestrin, 2016), LightGBM (Ke et al., 2017), and CatBoost (Prokhorenkova et al., 2018). In this section we will cover the latter, which has shown to outperform the others in terms of predictive performance on many benchmark datasets (Prokhorenkova et al., 2018).

The CatBoost algorithm describes and addresses one main statistical issue with the general gradient boosting framework (and indeed in other gradient boosting implementations as well). Considering the optimization (3.17), Prokhorenkova et al. (2018) addresses the following chain of shifts, dubbed a *prediction shift*:

1. The conditional distribution of the gradient $g_m(\mathbf{x}_i)|\mathbf{x}_i, \mathbf{x} \in \mathcal{D}$ (as defined in (3.16)) is shifted from the distribution on a test sample $g_m(\mathbf{x})|\mathbf{x}, \mathbf{x} \in \mathcal{S} \setminus \mathcal{D}$.
2. This results in the base predictor $h(\mathbf{x}|\mathbf{a}_m)$ from (3.17) to be biased from the real solution.
3. The result is a shift in the generalization of the final ensemble F.

Conceptually, the shift occurs because each base predictor h_m added to the ensemble is based on the gradient of the same full dataset $\mathcal{D}$, resulting in target leakage (i.e. target values in the dataset being improperly available during training). With access to unlimited training data, this problem is easy to solve: Sample independent training sets $\mathcal{D}_m \subset \mathcal{S}$ at each iteration m and use this sample to train the base predictor h_m, resulting in it being unbiased.

In most practical applications, however, only finite training data is available. In this case, to ensure that the model gradient $g_m(\mathbf{x}_k)$ is unshifted, the model F_m that it is based on can not be trained on the sample $\mathbf{x}_k$. While this may seem intractable, Prokhorenkova et al. (2018) propose a solution to this: By using a *supporting model* M_k that is never updated using the gradient for $\mathbf{x}_k$, we can safely estimate the gradient on $\mathbf{x}_k$ using M_k and use this gradient to perform the update.

This is done by only using a data partition $\mathcal{D}_k = \{\mathbf{x}_j : \sigma(j) < \sigma(k)\}$, where the data is ordered according to some random data permutation σ, to train the supporting model M_k. Thus, the supporting model M_k has only been trained on the ordered data up until k, giving the name *ordered boosting*. In practice, the robustness of this algorithm is then increased by keeping several random partitions $\{\sigma_r\}_{r=1}^s$, from which a partition is sampled at each iteration, producing the set of supporting models $M_{r,j}$. The ordered boosting algorithm for one such iteration is shown in Algorithm 2.

Data: Number of trees I, σ_r, $(\mathbf{x_i}, y_i)_{i=1}^N$ ordered according to σ_r
$M_i = 0$ for $i = 1, \ldots, n$;
for $t = 1$ *to* I **do**
  for $i = 1$ *to* n **do**
    $\tilde{y}_j = -\left[\frac{\delta L(y_j, a)}{\delta a}|\mathbf{x}_j\right]_{a=M_i(\mathbf{x}_j)}$, for $j = 1, \ldots, i-1$;
    $M = FitTree((\mathbf{x}_j, \tilde{y}_j) \text{ for } j = 1, \ldots, i-1)$;
    $M_i = M_i + M$;
  end
end

Algorithm 2: Ordered boosting (Prokhorenkova et al., 2018)

The practical implementation of CatBoost (available in the `CatBoost` package) also offers a number of computational features and improvements, often using algorithmic optimization or approximations, which we will omit in this description.

3.3.4 Categorical features

In the framework above we have assumed all features $x_k \in \mathbf{x}$ to take only numerical values. In many practical applications, however, we often encounter some *categorical features* $x_l \in A$, i.e. x_l may take any nominal value in some discrete set A, where the elements may have no inherent numerical relation to each other. To solve this, one typically apply some sort of transformation $g : A \to \mathbb{R}^p$, $p > 0$. Note that these methods are also applicable to the other methods mentioned in this chapter.

One-hot encoding

A popular and straight forward transformation is *one-hot encoding*, where one maps a categorical feature $x \in A$ to a binary vector $\hat{\mathbf{x}}$ of size l, where l is the cardinality of A. Each component in the vector $\hat{\mathbf{x}}$ represents one element of the set A, and a value 1 in the corresponding component represents the presence of this element. All other components are 0, giving the name "one-hot" encoding. Formally:

$$\hat{\mathbf{x}} = [\mathbb{1}_{\{x=a_1\}}, \mathbb{1}_{\{x=a_2\}}, \ldots, \mathbb{1}_{\{x=a_l\}}],$$

where $A = \{a_1, a_2, \ldots, a_l\}$.

One-hot encoding offers a simple transformation to allow categorical variables to be used in the function approximation frameworks. However, the approach suffers from some setbacks, namely that for categorical sets A with large cardinalities, the encoded vectors become sparse, making the dimensions of the input vectors grow potentially large.

Another issue with one-hot encoding is that the model will treat each category as completely independent variables, essentially removing information about their relation to eachother (which, in turn, has to be re-learned by the model).

Also, if one seek to perform some sort of model interpretation process (as we will describe in Section 4.2), the model's learned importance of the specific categorical feature may become distributed across each component of the encoded vector, which makes interpretation more difficult.

Target statistics

Target statistics is a method that encodes a categorical feature $x \in A$ to a single numerical feature, $g_{ts}(x) : A \to \mathbb{R}$, where the encoding aims to map each category a_k to an estimate of the probability or expected value of the target value y conditioned on the category:

$$g_{ts}(a_k) \approx \mathbb{E}(y|x = a_k). \tag{3.18}$$

A straight forward approach to estimating this expectation is using the average value of y over the training sample, given that the corresponding $x = a_k$. As this estimate is noisy for low-frequency categories, Micci-Barreca (2001) suggests smoothing it by some prior s:

$$g_{ts}(a_l) = \frac{\sum_{j=1}^{n} \mathbb{1}_{\{x=a_l\}} y_j + \gamma s}{\sum_{j=1}^{n} \mathbb{1}_{\{x=a_l\}} y_j + \gamma}, \tag{3.19}$$

where $\gamma > 0$ is a tuning parameter. Micci-Barreca (2001) also suggests using the average target value across the whole dataset for the prior s.

An additional feature of the CatBoost framework specifically, is the concept of *ordered target encoding*. While a detailed description will be left to (Prokhorenkova et al., 2018), it builds on the same concept of ordered boosting, in that rather than computing target statistics on the whole dataset (which results in the same shift discussed previously in the section), it uses a similar ordered procedure to compute the target statistics during training.

3.4 Neural networks

Neural networks are a family of learning methods inspired by the neural circuitry of the brain, modelled as a sequence of linear combinations of input features, with non-linear functions applied in between. While the concept of using neural networks for function estimation was first introduced in Rosenblatt (1958), the methods typically require both large amounts of data and are computationally intensive, which has resulted in a surge of popularity only in recent years, as the availability of both of these resources has greatly increased.

In this section, we will describe a basic, although commonly used type of neural networks, called fully connected neural networks. This will also serve as a foundation for the recurrent neural networks we will describe in Section 3.5.

3.4.1 Fully connected neural networks

The most basic type of neural networks are fully connected, feed-forward neural networks. These networks consist of an input layer, a sequence of M "hidden" layers, and an output layer. Each hidden layer consists of h_m nodes, each of which represents a linear combination of every node in the previous layer, to which a nonlinear *activation function* f_m is applied. At the final layer (a single node if the output is one dimensional), an output function g is applied to transform the output to the desired target form. In other words, values are "fed forward" in the network as linear combinations. Formally, this can be represented by the following process:

$$\begin{aligned} \mathbf{z}_0 &= \mathbf{x}, \\ \mathbf{z}_m &= f_m(\mathbf{W}_{m-1}\mathbf{z}_{m-1}), \quad m = 1, \dots, M \\ \hat{y} := F(\mathbf{x}) &= g(\mathbf{W}_M \mathbf{z}_M), \end{aligned} \tag{3.20}$$

which can be written in a compact composite form,:

$$F(\mathbf{x}) = g(\mathbf{W}_M f_M(\mathbf{W}_{M-1} f_{M-1}(\mathbf{W}_{M-2} f_{M-2}(\cdots \mathbf{W}_1 f_1(\mathbf{W}_0 \mathbf{x}))))). \tag{3.21}$$

The coefficients $\mathbf{W}_m \in \mathbb{R}^{h_m \times h_{m-1}}$ in the linear combinations are often referred to as *weights*, and make up the trainable parameter set of the neural network. They also typically include a bias term that is not multiplied with the previous input, which we for convenience will drop in our description (essentially this means that we pad the $\mathbf{z}_m$ with 1 at the first element). The weights are typically represented as edges in the network (see Figure 3.3). The activation functions f_m, the number of hidden layers M, and the number of nodes in each layer h_m are hyperparameters, defining what we will refer to as the *architecture* of the network.

Intuitively, the parameters M and h_m control the complexity of the model: Higher values of these parameters allows the model to learn more complicated relationships in the data (with $M = 0$, the neural network essentially becomes a regression model). At the same time, increasing the network size both makes it harder to optimize (as we will see in Section 3.4.2), as well as more susceptible to overfitting, due to the increased flexibility.

One way to view a neural network is as a two-part model, consisting of a *feature extractor* and an estimator model (the final layer). In this sense, the feature extractor is made up of the M hidden layers, which are optimized to find the best intermediary feature representations Z_M of the input vector $\mathbf{x}$ for use as inputs in the estimator model, which in the fully connected case can be viewed as a regression model.

Activation functions

There are multiple popular choices for activation functions f_m. They are required to be differentiable (as we will see in Section 3.4.2), and simple, easy to compute functions are often preferable due to how frequently they will be applied. They are also often nonlinear to allow the network to model nonlinear relationships. Historically, the sigmoid $f(Z) = 1/(1 + e^{-Z})$ has often been deployed (Rosenblatt, 1958). An alternative to the sigmoid is the hyperbolic tangent $f(Z) = \tanh(Z)$, which models similar behaviour as the sigmoid, but has range $(-1, 1)$, as opposed to $(0, 1)$. Other popular activation functions include the rectified linear unit ("ReLU") $f(Z) = \max(0, Z)$ (where one sets the gradient equal to 0 at $Z = 0$) and variations thereof (Maas et al., 2013).

The final transformation function g depends on the characteristics of y; in a regression setting, one typically deploys the identify function $h(Z) = Z$. For binary prediction, the sigmoid is often used, as the range $(0, 1)$ allows it to be interpreted as a "probability" for $y = 1$.

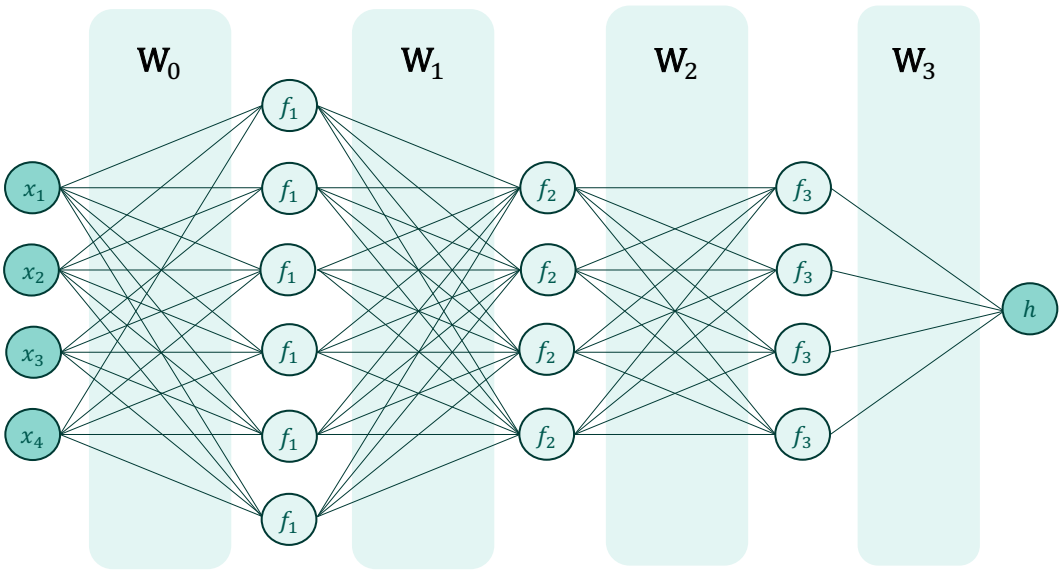


Figure 3.3: An illustration of a fully connected neural network with $M = 3$ hidden layers, with $h_1 = 6$, $h_2 = 4$, $h_3 = 4$ nodes. Note that the constant weight terms at each layer are omitted in this illustration.

3.4.2 Back-propagation

The question then becomes how to estimate the optimal parameter set $\{\mathbf{W}_m\}_0^M$ in 3.20. These parameters are learned iteratively from the data using gradient-based numerical optimization on a loss function L, as discussed in Section 3.1.1.

In order to update the weights $\mathbf{W}_m$ at each layer, we need the corresponding gradients $\nabla_{\mathbf{W}_m}$ of L with respect to the weights $\mathbf{W}_m$, for a sample $\mathbf{x}_i$. The procedure of obtaining these gradients are split into two processes, called the *forward pass* and the *backwards pass*.

In the forward pass, the network is evaluated in the (ordinary) feed-forward manner at a sample $\mathbf{x_i}$, holding the weights fixed. Let $\mathbf{z}_m$ denote the output of layer m (as in (3.20)), and $\mathbf{a}_m = \mathbf{W}_{m-1}\mathbf{z}_{m-1}$ the weighted input to layer m, both evaluated at $\mathbf{x}_i$.

Because of the composite form of 3.21, we can then expand the derivative of the loss function using the chain rule from calculus, now holding the other values (computed during the forward pass) fixed. After evaluating the forward pass, we get

$$\partial_{\mathbf{x}} L = \partial_{\mathbf{z}_g} L \cdot \partial_{\mathbf{a}_g} \mathbf{z}_g \cdot \partial_{\mathbf{z}_M} \mathbf{a}_g \cdot \partial_{\mathbf{a}_M} \mathbf{z}_M \cdot \partial_{\mathbf{z}_{M-1}} \mathbf{a}_M \cdots \partial_{\mathbf{a}_1} \mathbf{z}_1 \cdot \partial_{\mathbf{x}} d\mathbf{a}_1,$$

where $L = L(y, F(\mathbf{x}_i))$, and $\mathbf{z}_g$ and $\mathbf{a}_g$ refers to the output and input to the output function g, respectively. The g, f_m, $\mathbf{z}_m$ and w_m are as defined in (3.20). The $\partial_{(\cdot)}$ denotes the partial derivative with respect to its subscript.

As $\mathbf{a}_m = \mathbf{W}_{m-1}\mathbf{z}_{m-1}$ are matrix multiplications, their derivative with respect to $\mathbf{z}_{m-1}$ are just the weights $\mathbf{W}_{m-1}$. From the definitions in (3.20), the derivatives of $\mathbf{z}_m$ with respect to the input $\mathbf{a}_m$ are the derivatives of the activation function f_m. We then get

$$\partial_{\mathbf{x}} L = \partial_{\mathbf{z}_g} L \cdot h' \cdot \mathbf{W}_M \cdot f'_M \cdot \mathbf{W}_{M-1} \cdots f'_1 \cdot \mathbf{W}_1. \tag{3.22}$$

The gradient ∇ is the transpose of the derivative of the outputs, meaning we transpose the matrices and reverse the order of the terms to obtain the gradient:

$$\nabla_{\mathbf{x}} L = \mathbf{W}_1^\intercal \cdot f'_1 \cdot \cdots \cdot \mathbf{W}_{M-1}^\intercal \cdot f'_M \cdot \mathbf{W}_M^\intercal \cdot h' \cdot \nabla_{\mathbf{z}_g} L. \tag{3.23}$$

The *backward pass* of back-propagation then consists of sequentially evaluating this expression from right to left, as each term in (3.23) depends on the terms to the right of it, and $\nabla_{\mathbf{z}_g} L$ can be readily computed after the forward pass. We introduce the quantity δ_m, defined as the gradient of the input at layer m:

$$\delta_m = f'_m \cdot \mathbf{W}_{m+1}^\intercal \cdot \cdots \cdot \mathbf{W}_{M-1}^\intercal \cdot f'_M \cdot \mathbf{W}_M^\intercal \cdot h' \cdot \nabla_{\mathbf{z}_g} L. \tag{3.24}$$

Each δ_m can then be calculated recursively as

$$\delta_{m-1} = f'_{m-1} \cdot \mathbf{W}_m^\intercal \cdot \delta_m, \quad m = M, M-1, \ldots, 0.$$

Finally, we can compute the gradient of the weights W_m at layer m as

$$\begin{aligned} \nabla_{\mathbf{W}_m} &= \delta_m \cdot \frac{\delta \mathbf{W}_m \mathbf{z}_{m-1}}{\delta \mathbf{W}_m} \\ &= \delta_m \cdot \mathbf{z}_{m-1}^\intercal. \end{aligned}$$

Intuitively, the loss gradients δ_m are computed and propagated in a backwards manner through the network, giving the name *back-propagation*.

With the gradients available, we can make use of the gradient-based techniques described in Section 3.1.1. An ordinary gradient descent step at iteration $t+1$ over all training samples becomes:

$$\mathbf{W}_m^{t+1} = \mathbf{W}_m^t = -\alpha_{t+1} \sum_{i=1}^{n} \delta_m (\mathbf{z}_m^{t,i})^\intercal, \tag{3.25}$$

where the superscript in $\mathbf{z}_m^{t,i}$ indicates that it is evaluated at $\mathbf{x}_i$ using parameters at iteration t. Again, the learning rate α_t is a tuning parameter, and may be fixed or chosen adaptively.

3.4.3 Categorical feature embedding

As in the other methods discussed in this chapter, neural networks require numerical inputs, meaning transformations are required for categorical features. Again, one-hot encoding the categorical variables as discussed in Section 3.3.4 is a popular and straight forward option, but suffers from the same drawbacks of dimensionality increase discussed in its respective section.

One interesting solution to categorical features exclusive to neural networks is *categorical feature embeddings*. The approach is inspired by and very similar to techniques often deployed in natural language processing, where the idea is to map a vast vocabulary of words (a discrete "space" of words where one can view each word as a nominal value) to some numerical vector space called an *embedding space* (Zhang et al., 2016). A desired property of this space is then that words that are close to each other (typically in terms of Euclidian distance) have semantically similar meaning.

For categorical variable embedding, the principles are the same. The categorical variables are first encoded to a numerical vector $\hat{\mathbf{x}}$ where each category is assigned an integer, which is then encoded to the embedding space using a set of learnable weights $\mathbf{W}_{emb} \in \mathbb{R}^{d \times s}$, where s is the cardinality of the categorical variable space and the embedding space dimension d being a hyperparameter (Guo and Berkhahn, 2016).

The embedding space is then given by the matrix multiplication $\mathbf{W}_{emb}\hat{\mathbf{x}}$, which is then concatenated to the numerical feature vector $\mathbf{x}$ in (3.20), treated as ordinary numerical features. The weights $\mathbf{W}_{emb}$ can be trained and updated using ordinary back-propagation, resulting in optimal embedding parameters being learned according to the specified target values and loss.

In this sense, one-hot encoding can actually be viewed as an embedding where D equals the cardinality of the category set and all categories have a fixed mapping, equally distant. Similarly, the target encoding discussed in Section 3.3.4 can be viewed as an embedding with $D = 1$, where the embedding weights maps the categories to the corresponding expectation of the target variable y.

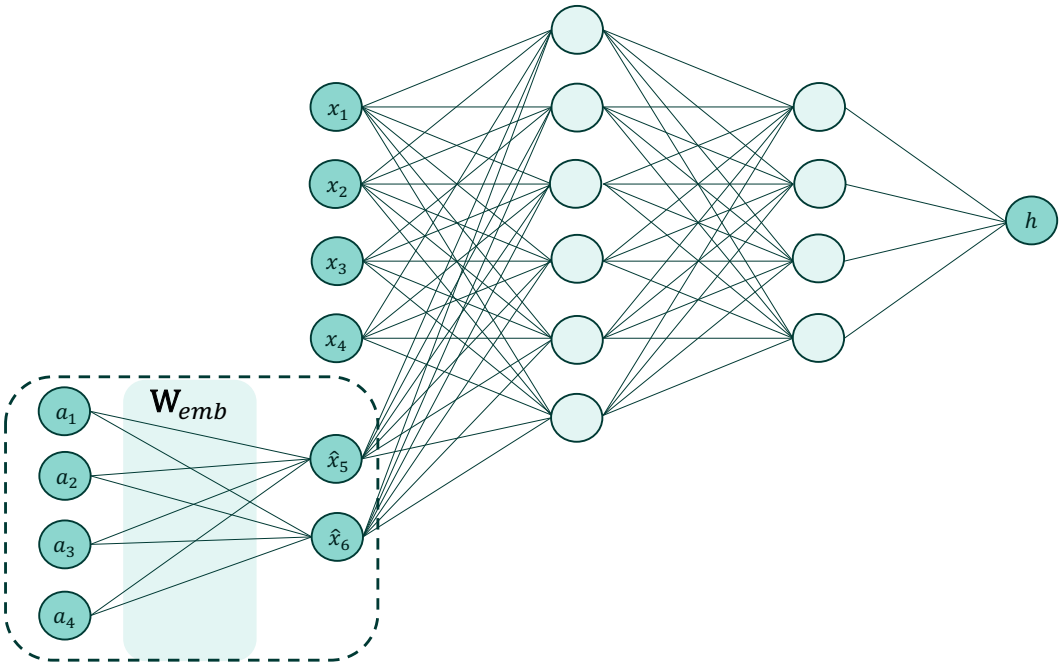


Figure 3.4: An illustration of a categorical feature embedding for a categorical variable with cardinaliy $s = 4$, and output dimension $d = 2$. The embedding layer produces a set of "new" features $\hat{x}_5, \hat{x}_6$, which are concatenated with the other 4 features.

3.4.4 Issues with neural networks

While the inherent capacity of neural networks allow for approximation of almost any underlying function, there are a number of issues related to building, training, and tuning neural networks. In most applications they are inherently overparameterized, with a non-convex and unstable optimization (Hastie et al., 2001). They are also notoriously data-hungry, requiring large amounts of data to be able to generalize well (Pasini, 2015). In this section, we will highlight some of the challenges and how they may be addressed in a qualitative manner, referencing other sources for more details.

Vanishing/exploding gradients

Because of the composite form of the gradients and their sequential dependencies, the training of neural networks may become unstable, especially so for deeper networks. In the case of unbounded activation functions, the gradients may accumulate and become very large as the depth of the network increases, resulting in *exploding gradients*, which in turn makes the weight updates unstable and training infeasible. Bounded activation functions, such as the sigmoid or hyperbolic tangent may help with this (Glorot and Bengio, 2010).

On the other hand, if the gradients are very small, they may cause all depending gradients to shrink as well, resulting in *vanishing gradients*. This may occur when asymptotic activation functions, such as the sigmoid and hyperbolic tangent, becomes saturated, resulting in near-0 gradients, which in turn results in no training (Glorot and Bengio, 2010).

Generally, the two following properties should approximately hold in order to prevent both exploding and vanishing gradients (Goodfellow et al., 2016):

- The mean of the activations should be zero.
- The variance of the activations should stay the same across every layer.

Both of these issues are typically addressed by tuning either the network architecture, the activation functions or properly initializing the network's weights (Goodfellow et al., 2016).

Weight initialization

One challenge of neural networks is generating initial values of the weight set $\{\mathbf{W}_m\}_0^M$, often referred to as *weight initialization*. Weight initialization greatly affects how the training progresses, and badly initialized weights may induce either vanishing or exploding gradients, described above. Glorot and Bengio (2010) shows that the outcome of different initialization schemes depend on both the architecture of the neural network, as well as the particular choices for activation functions f_m.

Generally, equally initialized weights will result in all gradients across a layer being equal, meaning all nodes in the layer learn the same behaviour. Typically, small (near zero) random initializations are preferable as they avoid both of these problems.

As mentioned in the previous section, we generally want activations such that the variance of the activations in each layer are the same. Because the activations are propagated forwards sequentially, this means that the initialized weights should not have the same variance in each layer (Goodfellow et al., 2016). Glorot and Bengio (2010) propose an initialization scheme that normalizes the weights in a layer to a region where the gradient updates behave well called *glorot initialization*, by ensuring the variance is scaled such that the variance of the output is approximately equal to 1 in every layer.

Regularization

Because of the overparameterized nature of neural networks, they are very prone to overfitting, motivating the application of one or more regularization methods. As before, introducing penalizing terms are a standard approach. Hastie et al. (2001) suggests *weight decay*, which analogously to L_2 regularization in Section 3.2 adds a penalty term to the loss function L, proportional to total magnitude of the weights.

Another option is *dropout*, where one eliminates the signal of a random subset of nodes at each iteration, which is thought to motivate smaller dependence on individual, "stronger" nodes which are more likely to be overfit (Srivastava et al., 2014).

A more pragmatic approach involves randomly splitting the dataset into a training set and a *validation set*, where the latter is used to evaluate the model's generalization ability throughout the training process, where the updates happens based exclusively on the training set. When the loss on the validation set starts to increase, the model has started to overfit the training dataset, and the training process is stopped. This technique is often referred to as *early stopping*, and is often deployed in other machine learning methods as well (such as gradient boosting machines, described in Section 3.3.3).

Model interpretation

One of the major criticisms of neural networks is the difficulty of properly interpreting the output of a trained model. Due to the complex and intertwined nature of the models, their major strength in complex estimation capabilities also become a major weakness, especially in applications where model accountability is important, as tracing what motivated a particular decision or output quickly becomes infeasible (Gilpin et al., 2018). Neural network explainability neural network interpretability remains an active area of research (Gilpin et al. 2018; Ribeiro et al. 2016).

3.5 Timeseries methods

In predictive timeseries modelling, we extend the predictive problem to account for time-distributed data, then including another temporal dimension to the data, where we assume some unique relationship between the data points and/or target variable.

Although many methods for modelling time dependent behaviour exist, we will limit our focus to methods building on the ideas of neural networks. With these methods, we can leverage the flexibility of the neural networks combined with large datasets to model complex, time dependent behaviour. Specifically, we will consider a family of neural network called *recurrent neural networks*, or RNNs.

3.5.1 Recurrent neural networks

Traditional recurrent neural networks consist of an input layer, a recurrent layer, and an output layer. The recurrent layers can be thought of as intermediary internal states $\mathbf{h}_t$ ("hidden" states) at each timestep, which acts as a link between the temporal elements in the timeseries. These states are often referred to as *recurrent cells* in the network. States consisting of h hidden units are given by a recurrence formula

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t | \mathbf{W}_h, \mathbf{W}_x), \tag{3.26}$$

where $\mathbf{h}_t^\intercal \in \mathbb{R}^h$, and f is some function parameterized by the weights $\mathbf{W}_t \in \mathbb{R}^{h \times h}$, $\mathbf{W}_x \in \mathbb{R}^{h \times p}$ (with $\mathbf{x}^\intercal \in \mathbb{R}^p$, as before). Note that f, $\mathbf{W}_x$, and $\mathbf{W}_h$ are not dependent on t, i.e. the weights and functions are shared across the timesteps. In the ordinary recurrent neural network, the prediction process for an output $\hat{y}_t$ can be described as

$$\begin{aligned} \mathbf{h}_t &= f(\mathbf{W}_h \mathbf{h}_{t-1} \mathbf{W}_x \mathbf{x}_t), \\ \hat{y}_t := F(\mathbf{x}_t) &= g(\mathbf{W}_{h_t}). \end{aligned}$$

where $\mathbf{W}_y^\intercal \in \mathbb{R}^h$ are the output weights.

Intuitively, this formulation allows the network to learn optimal dependence of all previous timesteps through the recursive equation (3.26) and weights $\mathbf{W}_h$, in addition to the input vector $\mathbf{x}_t$ as in the neural network framework in Section 3.4.

In the case of recurrent neural networks, the loss L of all time steps is simply computed as sum of losses at every timestep:

$$L(\hat{y}, y) = \sum_{t=1}^{T} L(\hat{y}_t, y_t). \tag{3.27}$$

3.5.2 Back-propagation through time

For training RNNs, we can use the principles of back-propagation described in Section 3.4.2, with some additional considerations, accommodating for the sequential dependence imposed by the hidden states $\mathbf{h}_t$. This process is called *back-propagation through time*, or BPTT.

For our purposes, we will present a simplified description of BPPT as in Zhang et al. (2020), where we let the functions f and g be the identify function. This simplification reduces a lot of notational clutter, while still bringing to light the key ideas, and the following complications. A detailed description can be found in Zhang et al. (2020).

To use the back-propagation scheme in Section 3.4.2, we need to find the derivatives of the loss L with respect to the network weights $\mathbf{W}_y$, $\mathbf{W}_h$, and $\mathbf{W}_x$. The first is easily obtained from (3.27):

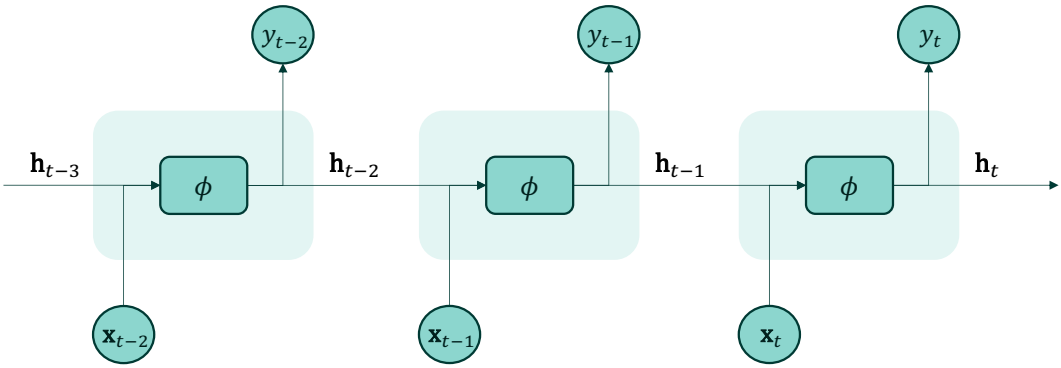


Figure 3.5: A simplified illustration of a recurrent neural network, showing how the previous hidden states are used to produce both an output y_t and the next hidden state $\mathbf{h}_t$. The edges represent sets of trainable weights, and the ϕ denotes a fully connected layer with a corresponding activation function.

$$\partial_{\mathbf{W}_h} L = \sum_{t=1}^{T} \partial_{\hat{y}_t} L(\hat{y}_t, y_t) \mathbf{h}_t,$$

where $\partial_{(\cdot)}$ denotes the partial derivative with respect to its subscript. For the latter two weight sets, we again use the chain rule, on (3.27):

$$\begin{aligned}
\partial_{\mathbf{W}_h} L &= \sum_{t=1}^{T} \partial_{\hat{y}_t} L\left(\hat{y}_t, y_t\right) \mathbf{W}_y \partial_{\mathbf{W}_h} \mathbf{h}_t, \\
\partial_{\mathbf{W}_x} L &= \sum_{t=1}^{T} \partial_{\hat{y}_t} L\left(\hat{y}_t, y_t\right) \mathbf{W}_y \partial_{\mathbf{W}_x} \mathbf{h}_t.
\end{aligned}$$

The first two factors of these equations are simple to compute. In order to compute the last factor, we use that from the recursive nature of the hidden states, we have that

$$\begin{aligned}
\partial_{\mathbf{h}_t} \mathbf{h}_{t+1} &= \mathbf{W}_h^\intercal, \\
\partial_{\mathbf{h}_t} \mathbf{h}_T &= (\mathbf{W}_h^\intercal)^{T-t},
\end{aligned}$$

using that we assume f to be the identity function. This gives the partial derivatives

$$\begin{aligned}
\partial_{\mathbf{W}_h} \mathbf{h}_t &= \sum_{j=1}^{t} (\mathbf{W}_h^\intercal)^{t-j} \mathbf{h}_j, \\
\partial_{\mathbf{W}_x} \mathbf{h}_t &= \sum_{j=1}^{t} (\mathbf{W}_h^\intercal)^{t-j} \mathbf{x}_j.
\end{aligned} \tag{3.28}$$

In the general case, the exponential in the sum of (3.28) is replaced with a sequential product of the gradients, with some additional terms (Zhang et al., 2020).

While this allows for the derivatives to be computed recursively and used in weight updating, the potentially large powers of $\mathbf{W}_h$ causes some problems that has to be considered. For large values of t, if the eigenvalues of $\mathbf{W}_h$ are greater than one, the gradients will quickly

explode. Similarly, if they are smaller than 1, the gradients vanish, resulting in amplified cases of the issues described in Section 3.4.4

To counteract this, one typically approximates the gradients by truncating the sum after τ timesteps, ignoring any terms in (3.28) after $t - \tau$. This will, however, come at the cost of the long term influences of inputs, making the model focus mainly on short term influences (which may or may not be desirable) (Zhang et al., 2020).

3.5.3 LSTM network

One recurrent framework that addresses the key issues of traditional RNNs without the loss of long-term dependencies, are the *long short term memory network*, or LSTM, introduced by Hochreiter and Schmidhuber (1997). It builds on the traditional recurrent architecture, but introduces a *memory cell*, which is used in addition to the hidden state to control the flow of information across different timesteps. In this section, we will give a conceptual description of the LSTMs, similar to that given in Zhang et al. (2020), while referring a more in-depth description to the original work of Hochreiter and Schmidhuber (1997).

Conceptually, we want to use the memory cell $\mathbf{c}_t$ to scale the hidden state $\mathbf{h}_t$, to either increase or decrease its influence at every timestep t. The information flow into and out of the memory cell is controlled by a set of three *gates*; the output gate $\mathbf{o}_t$, the input gate $\mathbf{i}_t$, and the forget gate $\mathbf{f}_t$.

Intuitively, the output gate controls what data is sent as output of the cell, the input gate controls the data flow into the cell, and the forget gate controls how data is deleted from the cell. Each gate has its own set of weights, both for the previous hidden state $\mathbf{h}_{t-1}$, and the input vector $\mathbf{x}_t$.

The outputs of each of the gates at each timestep t are simply linear combinations of their respective set of trainable weights and the previous hidden state, sent through a sigmoid activation function, denoted by σ:

$$\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{W}_{xi}\mathbf{x}_t + \mathbf{W}_{hi}\mathbf{h}_{t-1}),\\
\mathbf{f}_t &= \sigma(\mathbf{W}_{xf}\mathbf{x}_t + \mathbf{W}_{hf}\mathbf{h}_{t-1}),\\
\mathbf{o}_t &= \sigma(\mathbf{W}_{xo}\mathbf{x}_t + \mathbf{W}_{ho}\mathbf{h}_{t-1}),
\end{aligned}$$

where the weight parameters are $\mathbf{W}_{xi}, \mathbf{W}_{xf}, \mathbf{W}_{xo} \in \mathbb{R}^{h\times p}$ and $\mathbf{W}_{hi}, \mathbf{W}_{hf}, \mathbf{W}_{ho} \in \mathbb{R}^{n\times h}$, for $\mathbf{x} \in \mathbb{R}^p$ and h hidden units. As before, we omit the bias terms in this description. The sigmoid constrains the output to the range $(0, 1)$, allowing them to be thought of as gates.

A candidate memory cell $\widetilde{\mathbf{c}}_t$ is also constructed based on the previous hidden state, and sent through a hyperbolic tangent activation function:

$$\widetilde{\mathbf{c}}_t = \tanh(\mathbf{W}_{xc}\mathbf{x}_t + \mathbf{W}_{hc}\mathbf{h}_{t-1}),$$

where $\mathbf{W}_{xc} \in \mathbb{R}^{h\times n}$ and $\mathbf{W}_{hc} \in \mathbb{R}^{h\times h}$. The forget gate and input gate are then used to scale the previous and candidate memory cells through element-wise multiplication:

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \widetilde{\mathbf{c}}_t,$$

producing the memory cell at timestep t. Finally, the hidden state at timestep t is computed by scaling scaling the hyperbolic tangent of the memory cell with the output gate:

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t), \tag{3.29}$$

which in turn is used to produce the output $\hat{y}$, as in (3.26), completing the LSTM process.

Intuitively, the series of gates allows the memory cell to "remember" long term dependencies, separately from h_t, and allowing them to influence the output when relevant. This is all made possible through the framework of learnable weights central to neural networks, naturally adding more complexity compared to e.g. the traditional RNNs described in Section 3.5.1. However, this additional complexity has proven to be effective, making LSTMs perform very well on a number of timeseries-related tasks (Shewalkar et al., 2019).

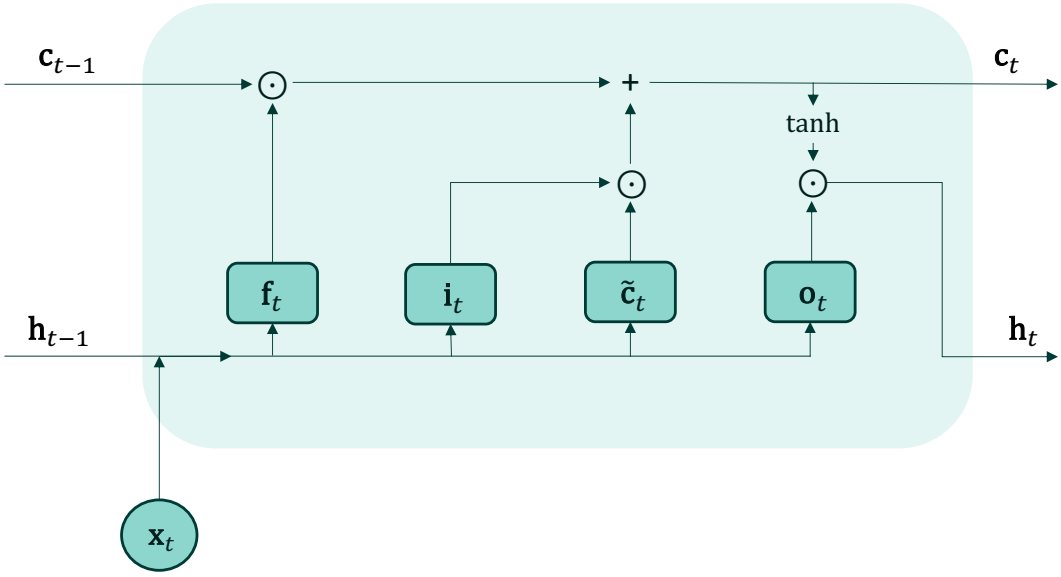


Figure 3.6: A simplified illustration of the interaction between the input $\mathbf{x}_t$, the previous hidden state $\mathbf{h}_{t-1}$, the previous memory cell $\mathbf{c}_{t-1}$, and the set of gates that make up an LSTM cell. The edges represent weights, and the gates are shown as rectangular boxes. These cells are then sequentially aligned in the t dimension, as the RNN in Figure 3.5.

Back-propagation through time for LSTMs

The introduction of gates to the RNN framework helps solve the issue of vanishing and exploding gradients faced in traditional RNNs. We will leave the exact mathematical motivation behind this to Hochreiter and Schmidhuber (1997), giving only a conceptual explanation.

In the LSTM framework, the exponential product in the gradient (3.28) contains instead a sum of the derivatives for the weights of each of the separate gates, rather than a single term, in addition to the hidden state. This additional flexibility allows the network to keep or remove gradients at certain timesteps t, should it be relevant. This has been shown to greatly help stabilize training, as well as increase performance on many tasks where both long and short term dependence is relevant (Hochreiter and Schmidhuber, 1997), (Shewalkar et al., 2019), (Anani and Samarabandu, 2018).

Chapter 4

Methods for model evaluation

In this chapter, we will define and discuss some methods that will be used for evaluating the bankruptcy prediction models presented in Chapter 6.

In Section 4.1, we will define some key binary classification metrics that will be used to analyze and evaluate the models, discussing their properties, strengths and weaknesses. In Section 4.2, we then introduce a model interpretation framework that can be used to explain and analyze many machine learning models, namely SHAP values, and then detail how it can be applied to each of our considered models.

4.1 Binary classification evaluation metrics

In order to evaluate and compare the performance of a predictive model, one ideally wants to formulate the overall quality of the model in terms of one or more *evaluation metrics*, which encapsulates some or all of the model's desirable properties in an intuitively appreciable manner. This is typically formulated as a single score value, facilitating easy comparison and ranking of models.

Because of the inherent tendency of many machine learning methods to overfit the training dataset - discussed throughout Chapter 3 - the evaluation metrics are typically computed on a separate *test set*, that has not been used during training. This allows for comparing of separate models' generalization ability, which is generally of most interest. Note that the test set should also be different from the validation set discussed in Section 3.4.4, as the validation set is used to optimally train the model.

In this section we then present and briefly discuss some popular metrics used for binary prediction tasks, which we will apply to evaluate and compare the bankruptcy prediction models in Chapter 6.

4.1.1 Scoring rules

In order to motivate our particular choices of metrics, we will first define the notion of scoring rules. Assume some model producing a set of probabilistic predictions $\widetilde{y}_i = \hat{P}(y_i = 1) \in (0, 1)$ for a set of binary target variable $y_i \in \{0, 1\}$ with a "true" probability $P(y_i = 1)$, $i = 1, \ldots, n$. A scoring rule s is then a mapping between the prediction $\hat{y}$ and a numeric loss:

$$s : (\hat{y}, y) \to s(\hat{y}, y).$$

A *proper scoring rule* is a scoring rule that is optimized (minimized or maximized depending on its formulation) in expectation by $\widetilde{y}_i = P(y_i = 1)$. It is said to be *strictly proper* if it is optimized *only* by $\widetilde{y}_i = P(y_i = 1)$. These are desirable properties, as they guarantee that the desired model behaviour (i.e. good approximation of the true probability $P(y_i = 1)$) will be reflected in better scores s.

Some popular metrics, such as accuracy, require the model predictions to be binary, meaning they are not, strictly speaking, scoring rules. In order to evaluate these metrics, we will apply a *decision threshold* τ, giving the binary predictions $\hat{y}_i = \mathbb{1}_{(\tilde{y}_i > \tau)}$. In practice (and in all of our future applications), this threshold is typically set to 0.5.

4.1.2 Accuracy

In a binary classification setting, the overall classification accuracy of the model is often the most natural metric to consider. It is simply defined as the ratio of correct predictions to total number of samples. For a set of binary predictions $\hat{y}_i$ following from a decision threshold τ, it is defined as

$$\text{accuracy} = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{(\hat{y}_i = y_i)}.$$

While perhaps the most intuitively appreciable metric, accuracy has some major drawbacks that has to be taken into consideration. In the case of probability predictions, all informativeness of the probabilities are lost in the decision rule $\hat{y}_i = \mathbb{1}_{(\tilde{y}_i > \tau)}$, and accuracy is consequently agnostic to any predictions in the range $(\tau, 1)$. One would typically prefer a model with probability estimates (correctly) distributed closer to 0 or 1, for which the accuracy metric provides no information. Consequently, the metric is also highly sensitive to the particular choice of τ.

The accuracy metric also becomes harder to interpret in the case of unevenly distributed target variables (unbalanced data). The accuracy of any naive model predicting exclusively positive (or negative) classifications is equal to the proportion of positive targets y_i, which in the case of highly unbalanced data problems may produce a very high accuracy, for a model with no predictive power. Therefore, it is important to consider the target data distribution when using the accuracy score as a metric.

4.1.3 F_1-score

A metric that addresses some of the issues with accuracy, while still requiring a decision threshold τ, is the F_1-score. It considers both the *precision* and *recall* of the model, providing information about the model's predictive power in the case of highly unbalanced data. It is simply defined as the harmonic mean of these quantities:

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}},$$

where

$$\begin{aligned} \text{precision} &= \frac{\sum_{i=1}^{n} \mathbb{1}_{(\hat{y}_i = y_i)} \mathbb{1}_{(y_i = 1)}}{\sum_{i=1}^{n} \mathbb{1}_{(\hat{y}_i = 1)}} = \frac{TP}{TP + FP}, \\ \text{recall} &= \frac{\sum_{i=1}^{n} \mathbb{1}_{(\hat{y}_i = y_i)} \mathbb{1}_{(y_i = 1)}}{\sum_{i=1}^{n} \mathbb{1}_{(y_i = 1)}} = \frac{TP}{TP + FN}. \end{aligned}$$

Note that the numerator in the middle expressions are simply the number of correct positive predictions. In the alternative formulation, TP denotes the number of true positives, FP the number of false positives and FN the number of false negatives (see Figure 4.1). Consequently, precision can be interpreted as the estimated probability of a randomly selected positive prediction being correct, while recall is the probability of a random true positive being amongst the positive predictions.

While allowing for a more balanced evaluation of a model's performance in the case of unbalanced data, the F_1-score still suffers from the same drawbacks as accuracy in the information loss in the decision process, and inherent sensitivity to decision threshold τ.

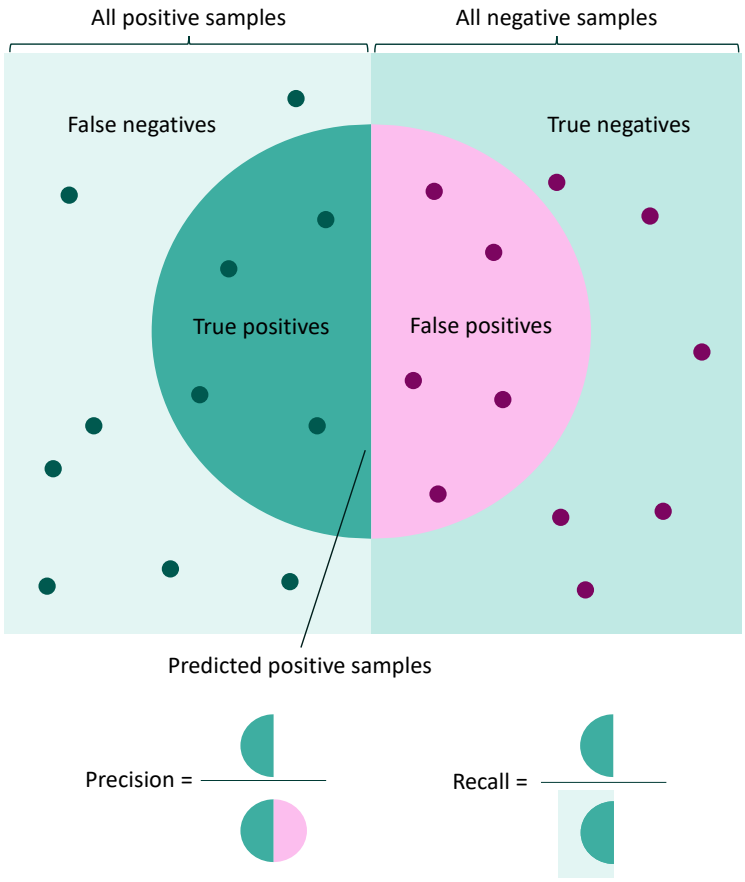


Figure 4.1: An illustration of true positives, false positives, false negatives and true negatives. The outer rectangles represent the space of all samples, with the left rectangle being all positive labelled samples (dark green dots), the right all negative labelled samples (purple dots). The circle represents the subset of samples predicted as positive by some model, and an illustration of the corresponding precision and recall metrics is given below. The figure is adapted from Wikimedia Commons (2014).

4.1.4 Brier score

A popular scoring rule that can be deployed beyond the task of binary classification is the *Brier score*, first described in Brier (1950). It is most commonly formulated as

$$bs = \frac{1}{n}\sum_{i=1}^{n}(\widetilde{y}_i - y_i)^2, \tag{4.1}$$

essentially the mean-squared error of the probabilistic prediction $\widetilde{y}_i$. It can be shown to be strictly proper, offering insight into the average quality of the probability estimates $\widetilde{y}_i$ that is lacking in both accuracy and the F_1-score.

A model consistently predicting probabilities close to 0.5 (e.g. 0.49 for negative labels and 0.51 for positive labels), will receive a relatively poor Brier score, while the accuracy may be good, if the classifications are correct according to the decision threshold. Thus, the Brier score will give a good indication of the quality of the individual predictions of the model, i.e. if the model is (correctly) confident in its probability estimates, independent of any decision threshold.

4.1.5 AUC score

A scoring metric that is often used for both evaluation and optimization, is the *area under receiver operator characteristic curve*, or AUC score, for short (Cortes and Mohri, 2003). It builds on the concepts of precision and recall discussed in Section 4.1.3, but varies the decision threshold τ across the range $[0, 1]$ to produce the *receiver operating characteristic* curve, or ROC curve, defined by the resulting points $(precision, recall)$. As $\tau = 0$ implies $precision = 0$ and $recall = 1$, and $\tau = 1$ gives $precision = 1$ and $recall = 0$, the ROC curve will always intercept these points, and then typically have a concave shape.

The ROC curve is visualized with $1 - precision$ on the x-axis and *recall* on the y-axis, as shown in Figure 4.2. The AUC score is then computed as the total area under this curve, and is consequently bounded to the range $[0, 1]$.

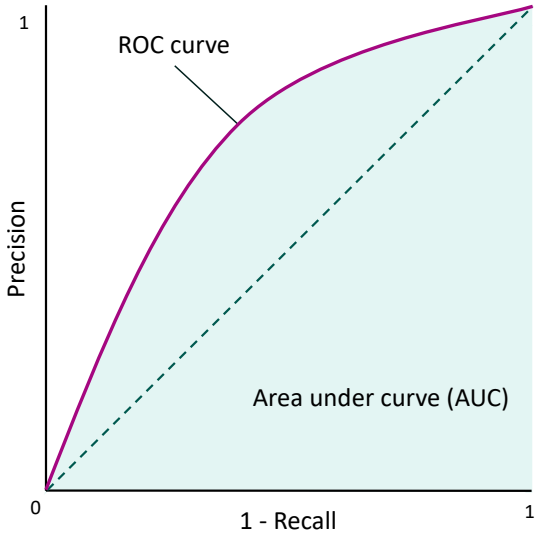


Figure 4.2: An illustration of the ROC curve with corresponding area under curve (AUC), in this case greater than 0.5. An arbitrary model will have a a linear curve and AUC score of 0.5, here showed as a dotted line.

The ROC curve describes the trade-off between precision and recall for different threshold values τ, meaning it encapsulates both information of the quality of the probability predictions $\widetilde{y}_i$, while also being somewhat robust to unbalanced data, as the F_1-score. A random classifier will on average have a linear ROC curve with slope 1, and thus an AUC score of 0.5, providing a baseline for the score. A perfect model will have an AUC score of 1, meaning any AUC score in the range $(0.5, 1]$ has predictive power. The AUC score can also be shown to be semi-proper (Cortes and Mohri, 2003).

Due to it encapsulating the total trade-off between precision and recall for all decision thresholds τ, as well as relevance even for unbalanced data, the AUC score will be our main metric for evaluating and comparing the bankruptcy prediction models.

4.1.6 Note on scoring metrics for bankruptcy prediction

When using metrics to evaluate models that are to be deployed in practical applications, it is important to consider the relevant contexts and ideally incorporate them into the metrics in some sense. For bankruptcy prediction specifically, there will often be different economical impacts for false positive and false negative predictions. An optimal evaluation would then incorporate such practical implications into their respective metrics, via e.g. weighting in the F_1-score (rather than using the harmonic mean), or simulation studies such as in Wahlstrøm et al. (2020). This would also have implications for the unbalanced data problem. We will, however, leave such considerations outside the scope of this thesis, always conforming to the previous "fair" metrics for consistency.

4.2 Machine learning model interpretability

A common criticism of advanced machine learning models involve the complexity of interpreting a model's output, and its learned behaviour. We will refer to this as *model interpretability*. Often dubbed the "black-box" problem, the complexity of models such as neural networks and large tree ensembles makes it difficult to dissect which features contributed to a specific prediction, and to what extent. Furthermore, precise analysis of the overall learned behaviour of a model may be equally complicated, which may cause problems such as data leakage and model bias to go unnoticed (Yu and Alì, 2019).

In many real-world applications (such as those where a bankruptcy prediction model may be applied) a key reason to apply such models may be to analyze the underlying reasons for the predicted behaviour, in order react to them. Therefore, model interpretability becomes a very desirable property, and often even a necessary requirement (especially in the light of new legal requirements), for a model to be applied.

While machine learning methods have developed greatly in terms of predictive performance capabilities, a lot is yet to be understood in terms of how they learn and perform predictions (Yu and Alì, 2019). However, this topic has gotten increased attention, motivating works such as Ribeiro et al. (2016) and Gilpin et al. (2018).

In this section we will focus on a framework that attempts to unify many of these ideas into one coherent framework, built on a set of desirable properties for a model explainability process, namely *Shapley Additve Explanations* - or *SHAP* - compiled in Lundberg and Lee (2017).

4.2.1 Shapley values

As the name implies, SHAP builds on the concept of *Shapley values*, which was first introduced as a game theoretic concept in Shapley (1951). In this context, Shapley-values address how to fairly distribute the gains of a coalition game where a set of players cooperate (unevenly) to achieve some outcome. The central idea is to assign each player with a value representing the gain of their contribution, relative to the expected gain should the player not have participated. Thus, the sum of the contribution values for all players should equal the total value of the gain, relative to no cooperation occurring.

The analogy to a predictive model is then to consider each feature as a player and the model prediction as the outcome, which we then want to "distribute" amongst the features. Thus, Shapley values provide *local explanations*, meaning they can be used to explain individual model predictions.

For a general prediction function $F(\mathbf{x})$, let $S \subset J$ denote a subset of the set of all features J, and let F_S denote a model trained only on the feature subset S. The Shapley value for a feature $j \in J$ for a particular example $\mathbf{x}$ is then computed as the weighted contribution of the feature, summed over all possible feature subsets S (Lundberg and Lee, 2017):

$$\phi_j(F) = \sum_{S \subset J \setminus \{j\}} \frac{|S|!(p - |S| - 1)!}{p!} (F_{S \cup j}(\mathbf{x}_{S \cup j}) - F_S(\mathbf{x}_S)), \tag{4.2}$$

where $\mathbf{x}_S$ denotes the feature vector $\mathbf{x}$ with only features in S, and p is the total number of features. In the case of non-independent features in $\mathbf{x}$, this requires training a separate model F_S for every feature subset S, which quickly becomes infeasible. One therefore often resorts to approximate (4.2) in some sense by for instance Monte-Carlo sampling (Štrumbelj and Kononenko, 2014), or assume feature independence (Lundberg and Lee, 2017).

4.2.2 Axiomatic properties of Shapley values

It can be shown that the Shapley value (4.2) is the only feature attribution method that satisfies four axiomatic properties, namely *efficiency*, *symmetry*, *dummy* and *additivity* (Lundberg and Lee, 2017).

Efficiency simply states that the feature contributions ϕ_j must sum to the prediction of a particular $\mathbf{x}$, less the average prediction:

$$\sum_{j=1}^{p} \phi_j = F(\mathbf{x}) - E_{\mathbf{x}\in\mathcal{D}}(F(\mathbf{x})),$$

ensuring the fair distribution of the contributions amongst the features.

The symmetry property states that the contribution of two features j and k should be the same if they contribute equally to all possible feature feature subsets. The dummy property states that a feature j whose influence on the prediction regardless of feature subset is always 0, should have a Shapley value of 0.

Finally, the additivity property states that for a model consisting of additive submodels, $F = f_1 + f_2 + \ldots$ (such as gradient boosting machines described in Section 3.3), the Shapley value of the full model should equal the sum of the Shapley values of each of the submodels:

$$\phi_j(F) = \phi_j(f_1) + \phi_j(f_2) + \ldots.$$

Conforming to these axiomatic properties provides a solid theoretical foundation for Shapley values to be used for model explanation.

4.2.3 SHAP (SHapley Additive exPlanations)

SHAP (SHapley Additive exPlanations) by Lundberg and Lee (2017) provide an extension of the Shapley values described above, connecting it to other explanation frameworks such as LIME (Ribeiro et al., 2016), by formulating model explanation as a separate additive *explanation model* g of binary variables:

$$g(\mathbf{x}') = \phi_0 + \sum_{j=1}^{p} \phi_i \mathbf{x}'_i, \tag{4.3}$$

where $\mathbf{x}' \in \{0,1\}^p$ is a *simplified input* that map to the original inputs $\mathbf{x}$ through a mapping function $\mathbf{x} = h_x(\mathbf{x}')$. In this context, (4.2) becomes

$$\phi_j(F) = \sum_{\mathbf{z}'\subset\mathbf{x}'} \frac{|\mathbf{z}'|!(p-|\mathbf{z}'|-1)!}{p!}(F_x(\mathbf{z}') - F_x(\mathbf{z}' \setminus j)), \tag{4.4}$$

where $|\mathbf{z}'|$ is the number of non-zero elements in $\mathbf{z}'$, $F_x(\mathbf{z}') = F(h_x(\mathbf{z}')) = E(F(\mathbf{z}|\mathbf{z}_S))$, S being the set of non-zero indices in $\mathbf{z}'$, and $\mathbf{z}' \setminus j$ denotes setting feature $z'_j = 0$.

Note that while using the mapping $h_x(\mathbf{z}') = \mathbf{z}_S$ requires training of separate models for each S, SHAP values instead approximate $F(\mathbf{z}_S|\mathbf{z}_S)$ with the expectation $E(F(\mathbf{z})|\mathbf{z}_S)$. By relaxing the assumption of feature independence, this can be further approximated noting that

$$E(F(\mathbf{z})|\mathbf{z}_S) = E_{\mathbf{z}_{\bar{S}}|\mathbf{z}_S}(F(\mathbf{z})) \approx E_{\mathbf{z}_{\bar{S}}}(F(\mathbf{z})), \tag{4.5}$$

where $\bar{S} = J \setminus S$ (Lundberg and Lee, 2017).

4.2.4 SHAP for logistic regression

Logistic (and linear) regression models are often used for its simple interpretation of feature attribution, simply by the coefficient weights β (following the notation in Section 3.2). Indeed, assuming feature independence, the SHAP value for feature j for a logistic regression model 3.4 (in logit space) becomes

$$\phi_j = \beta_j(x_j - E(x_j)), \tag{4.6}$$

aligning with the classical interpretation of feature attribution in logistic regression models (Lundberg and Lee, 2017).

When multicollinearity is present, however, the estimation is not as straightforward, and we run into the same problems estimating the expectation in (4.5) in an efficient manner. Lundberg and Lee (2017) propose an efficient estimation of this expectation by assuming $\mathbf{x}$ to be multivariate Gaussian.

The exact derivation of the method requires extensive computation, but in short, the Gaussian assumption allows for the approximation of a transformation matrix $\mathbf{T}$ such that the Shapley values $\phi = \beta\mathbf{T}\mathbf{x}$. This transformation matrix can be precomputed using random sampling, allowing for efficient estimation of SHAP values under multicollinearity.

4.2.5 KernelSHAP

Lundberg and Lee (2017) propose a kernel-based estimation approach for approximating (4.5) in a model-agnostic manner, inspired by the LIME framework (Ribeiro et al., 2016). In LIME, the explanation model g is assumed to be (locally) linear, and is found by minimizing

$$\epsilon = \arg\min_{g \in \mathcal{G}} L(f, g, \pi_{\mathbf{x}'}) + \Omega(g), \tag{4.7}$$

where the loss function L, regularization term Ω, and a *weighting kernel* $\pi_{\mathbf{x}'}$ are heuristic parameters, and $\mathcal{G}$ is the set of all linear functions. Lundberg and Lee (2017) then propose the following parameters such that the solution to (4.7) recover the Shapley values (4.4), thus adhering to the desired axioms in Section 4.2.2:

$$\begin{aligned}
\Omega(g) &= 0, \\
\pi_{\mathbf{x}'}(\mathbf{z}') &= \frac{p-1}{\binom{p}{|\mathbf{z}'|}|\mathbf{z}'|(p-|\mathbf{z}'|)}, \\
L(f, g, \pi_{\mathbf{x}'}) &= \sum_{\mathbf{z}' \in Z} (F(h_x^{-1}(\mathbf{z}')) - g(\mathbf{z}'))^2 \pi_{\mathbf{x}'}(\mathbf{z}').
\end{aligned}$$

With g assumed to be linear and L being the squared loss, the Shapley values can in this case be computed by weighted linear regression, resulting in a model-agnostic estimation of SHAP values under these assumptions.

4.2.6 TreeSHAP

The linear assumptions of KernelSHAP restricts the flexibility of the methods, especially so for more complex, non-linear models. Lundberg et al. (2020) propose a specialization for tree-based methods (such as decision trees described in Section 3.3.1 or gradient boosting trees in Section 3.3) for computing SHAP values in a computationally efficient manner. Specifically, it allows for efficient approximation of the conditional expectation $E(f\mathbf{x}|\mathbf{x}_S)$ for a single tree model f, which can then be applied to additive ensembles by the additivity property in Section 4.2.2. We will give a conceptual description of the estimation of the SHAP values in a tree model, leaving the description of the computational optimizations and exact algorithmic structure to Lundberg et al. (2020).

For a decision tree structure as described in Section 3.3.1 based on a set of features J, let S denote a subset of features. When $S = J$, the set of all features, the expectation $E(f\mathbf{x}|\mathbf{x}_S) = E(f\mathbf{x})$. When S contains only a subset of features, the expectation is computed by ignoring the predictions that depend on nodes with splitting features not present in S ("unreachable" nodes). For the remaining terminal nodes, the predictions are weighted by the number of training samples in each node, and the $E(f\mathbf{x}|\mathbf{x}_S)$ is then the recomputed (and then weighted) value c_j in (3.8) for the remaining terminal nodes.

Feature importance in gradient boosting trees

In many gradient boosting tree implementations, such as Ke et al. (2017) and Chen and Guestrin (2016), one can compute a global measure called *feature importance*, meant to capture the overall attribution of each feature. These methods generally rely on computing the total loss reduction for each feature (Ke et al., 2017), or the number of optimal splits found for each feature (see (3.9)).

However, Lundberg et al. (2020) argue that these are inconsistent, meaning that the model can be changed to rely more on a single feature, but result in a decrease in the attributed feature importance. These methods are therefore lacking in their explainable power, and does not adhere to the axiomatic properties of SHAP values, making SHAP values preferable for gradient boosting tree explanation.

4.2.7 Global SHAP values analysis

While SHAP values inherently produce local explanations, Lundberg et al. (2020) suggest simple extensions to allow for global explanations of model behaviour by using aggregated SHAP values, as they provide consistent and theoretically sound foundations for such analysis. The most natural extension then involves global importance of individual the individual features, simply as the mean magnitude of the SHAP value across all training samples.

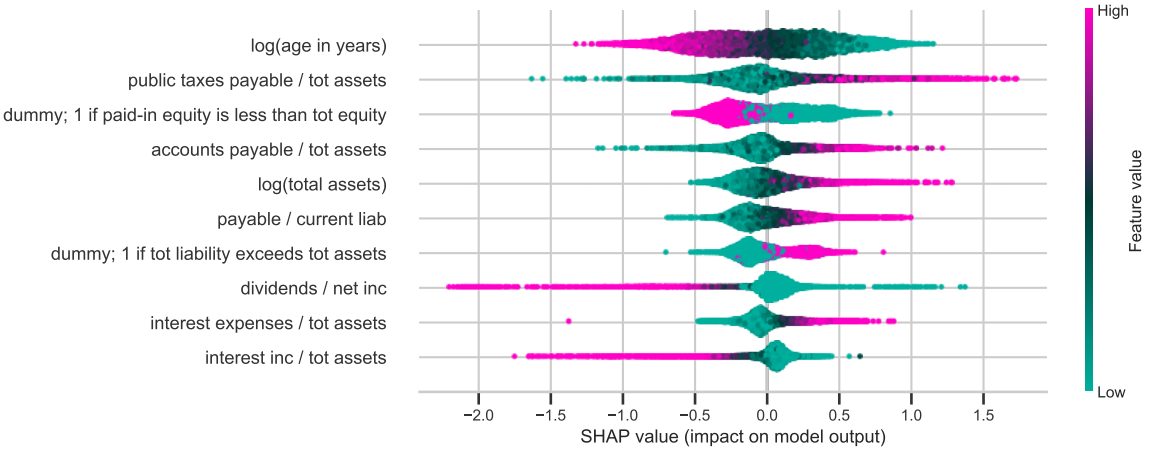


Figure 4.3: Example SHAP summary plot showing the SHAP values and corresponding feature values for 10 different features.

The `SHAP` package offers a global summary plot, which plots the n most important features (as the features with the greatest mean SHAP value magnitude), as horizontal scatter plots with color intensity representing the feature value. An example of a SHAP summary plot is shown in Figure 4.3. Note that non-uniform color distributions (e.g. differences in feature value distributions for the same SHAP value) implies that feature interactions caused the resulting SHAP value. These plots provide an intuitive view of both the most important features, as well as their feature value and corresponding SHAP value distributions across the whole dataset.

From Figure 4.3, we see for instance that high feature values of the feature $dividens\ /\ net\ inc$ has a large negative impact on the model output, and vice versa. The mixed feature intensity values for similar SHAP values for the feature $public\ taxes\ payable\ /\ tot\ assets$ implies that interaction effects are taking place.

While the `SHAP` package also offers views for analysis of specific interactions between feature value pairs and their corresponding SHAP values, we will not be considering them in our analysis, as we will focus on the more general model behaviour, rather than specific features.

Chapter 5

Experimental setup and design

Before we present and analyze the results of our bankruptcy methods, we will thoroughly describe how the procedures and experiments were deployed. The methodology was constructed to adhere to the highest standards of machine learning best practice, whenever possible. As we are working with time-distributed data, this involves avoiding *time-leakage*, meaning future data must be unavailable to a model that is considering data at a specific time. When other considerations had to be made, they will be clearly specified.

In this chapter we will describe in detail the data preprocessing process, as well as how the dataset was balanced, building on the dataset described in Section 2.3. We will then specify how the dataset was split into a train and test set, and then detail the temporal k-fold validation scheme deployed in order to reliably tune and optimize the models.

We will then briefly highlight some considerations made when performing model interpretability analysis and feature selection, before describing the special considerations that were made when formulating the problem as a timeseries problem.

Finally, the specific implementations of each of the models (and corresponding SHAP analysis) will be described. A general note is that all methods were implemented using the `Python` programming language, and that all packages used are open-source and available for free.

5.1 Data preprocessing

In statistical learning applications using large numerical datasets, one often applies some sort of data preprocessing processes in order to allow for better and more robust training of the algorithms. In general, we follow the same preprocessing that was applied in Wahlstrøm et al. (2020) on the same dataset, which we will describe in this section.

5.1.1 Quantile truncation

One technique typically applied to handle numerical data outliers is *quantile truncating*, where one truncates the values of each of the features to a specified quantile. For every sample $i = 1, \dots, n$ and feature $k = 1, \dots, p$, we set the corresponding value x_k^i as

$$(x_j^i)' = \begin{cases} Q_k^{\alpha_1}, & \text{if } x_k^i < Q_k^{\alpha_1}, \\ Q_k^{\alpha_2}, & \text{if } x_k^i > Q_k^{\alpha_2}, \\ x_j^i, & \text{otherwise,} \end{cases}$$

where Q_k^α denotes the sample α-quantile of feature k, $\alpha \in (0, 1)$. Such a truncating scheme allows for protection against outliers (which are often problematic, as discussed in Chapter 3), while not discarding any data samples. This is especially advantageous when the feature

set is large, as in our case, as this can cause many samples to be outliers in just a low number of features.

Wahlstrøm et al. (2020) found the model performance to be little sensitive to particular (reasonable) choices of α_1 and α_2. We will follow their optimal parameters, using symmetrical quantiles given by $\alpha_1 = 0.01$, $\alpha_2 = 0.99$.

5.1.2 Feature scaling

Some of the methods described in Chapter 3 require the numerical features to be on a similar scale, in order to avoid inherent bias or instability. For these methods - logistic regression (Section 3.2) and neural network methods (Sections 3.4 and 3.5) - we standardize the features by performing *Z-scaling*, normalizing the data based on the sample mean μ_k and standard deviation σ_k of feature k. For every sample i and numerical feature k:

$$(x_k^i)' = \frac{x_k^i - \mu_k}{\sigma_k}.$$

Note that we perform Z-scaling *after* quantile truncation.

5.1.3 Log transformation

For the features $company\_age$ and $total\_assets$, a log transformation is applied. As the $company\_age$ may be 0, all values for this feature were added with 1 before log transformation. The log transformations were applied before quantile truncation and feature scaling.

5.1.4 Categorical encoding

For the categorical variable described in 2.3, we will analyze results both for one-hot encoding, target statistic encoding and categorical embedding, discussed in Sections 3.3.4 and 3.4.3 (the latter only for neural networks). In the latter two cases, this involves performing some integer encoding of categories before the models are applied, where we arbitrarily assign each unique category with an integer.

Note that there are also two binary variables; $dummy; 1 if paid-inequity is less than to equity$ and $dummy;\ 1\ if\ tot\ liability\ exceeds\ tot\ assets$. These will always be kept in binary form, and not scaled and truncated.

5.2 Dataset balancing

In most real world classification or prediction applications, there will typically be an imbalance in the distribution of target label classes, referred to as *unbalanced data*. In our binary target case, we have a majority of positive labels (i.e. non bankruptcies), referred to as majority samples. We will in this section refer to r as the proportion of minority samples (meaning samples labelled with the minority target value, i.e. bankrupt companies) in the dataset.

In the case of very unbalanced data ($r < 0.1$), binary machine learning models may have trouble learning a predictive decision boundary, getting stuck in local minima as the gradient based optimization may tend towards predicting all samples as the majority class (Somasundaram and Reddy, 2016). In our case $r = 0.0169$, raising concerns for using a machine learning model to learn these rare occurrences.

A simple, yet effective way to negate this is to perform one or more sampling techniques in order to balance the distribution of target values in the training set. Note that this approach comes with drawbacks, which we will discuss at the end of this section.

5.2.1 Matched undersampling

We will reproduce the sampling method used by Wahlstrøm et al. (2020), which deploys a three-step matching process of samples, aimed towards matching each minority sample with a similar majority sample company. Although somewhat heuristic, this approach is thought to better allow for the model to learn a more optimal decision boundary between samples, as opposed to random undersampling, and is used throughout the bankruptcy prediction literature (Wahlstrøm et al. 2020; Härdle et al. 2009).

The process involves imposing three requirements for which a candidate majority sample needs to match the considered minority sample:

1. The majority sample needs to be of the same accounting year as the minority sample. This ensures no time-leakage in the balancing process.
2. The majority sample needs to be of the same industry as the minority sample. The industry is defined as the uppermost level (i.e. most general level) of the NACE-code, described in Section 2.3.5.
3. The $total\_assets$ of the majority sample needs to be within $\pm 10\%$ of the minority sample (before log transformation).

Note that for the timeseries methods, the average yearly $total\_assets$ over the timeseries were used in the third requirement.

When imposing these requirements, each minority sample were found to have at least one matching majority sample. In the case of multiple matched samples, the majority sample used in the balanced dataset was drawn randomly from these.

As results are typically compared using one or more of the metrics discussed in Section 4.1, and because these metrics are sensitive in various degrees to the target value distribution of the considered dataset, results are often reported on the balanced dataset, as in Wahlstrøm et al. (2020). We will conform to this, focusing most of the reported results on the balanced dataset. We will, however, also do a final evaluation of the models on the full dataset, and report the findings.

While there are more advanced sampling techniques available, such as synthetic minority oversampling (*SMOTE*, Fernández et al. (2018)) or Adaptive synthetic sampling (*ADASYN*, Haibo He et al. (2008)), initial experiments with these techniques showed little improvement. As we consider the unbalanced data problem, and consequently sampling techniques, beyond the scope of this thesis, we will only deploy the same technique as Wahlstrøm et al. (2020), for consistency.

5.2.2 Issues with sampling

Naturally, many considerations must be made when applying such sampling schemes on the training data. The most evident one is that of distribution shift, as one shifts the training data distribution from the true distribution one wants to approximate when performing such artificial sampling techniques. This will typically produce strong biases in the model, and one would need to thoroughly investigate the model's behaviour on the unbalanced dataset in order to learn these biases before any practical applications of the model can be considered. As mentioned before, we will consider any such investigation beyond the scope of this thesis.

Another apparent drawback is the inefficient usage of data, largely ignoring the majority of the (majority class) data, which naturally should hold at least some information relevant to the predictive power of the model. One approach alternative to dataset balancing that may make better use of the data is to use *sample weighting* during training (Johnson and Khoshgoftaar, 2019). By assigning larger weights to minority samples, the model can perform greater iterative updates for minority samples. This forces the model to "pay more attention" to the sparse minority samples, compared to the (numerically abundant) majority samples.

Initial experiments showed good promise for applying the sample weighting technique, especially when combined with fractional dataset balancing (e.g. balancing the dataset such that $r = 0.1$, as opposed to $r = 0.5$). This resulted in comparable performance on a fully

balanced dataset, while increasing performance on the full, unbalanced dataset. However, we will not be reporting these results as they are considered beyond the scope of this thesis, only noting the method's promise.

The unbalanced data problem is both evident and relevant, not only in bankruptcy prediction, but in machine learning in general (Somasundaram and Reddy, 2016). (Krawczyk, 2016), (Kotsiantis et al., 2005). We stress that this problem has to be handled, or at least considered, before applying any of these models to real-world cases. However, any such considerations should include economical or other practical factors in order to properly evaluate the trade-offs that has to be made. Thus, we consider the problem beyond the scope of this thesis, and while we will report results on the full, unbalanced dataset, no considerations has been made to optimize towards this case - only for the balanced data case. We acknowledge the limitations of this, and encourage further work to investigate how to improve upon this.

5.3 Train and test set splitting scheme

As described in Section 4.1, it is considered machine learning best practice to split the dataset into a separate training set and test set, where the former is used to train, tune and optimize models, and the latter exclusively to report the final results. This practice results in a realistic estimate of the model's expected performance on unseen data.

In the case of time dependent data, the test set should be comprised of the most recent data observations, in order for the results to most realistically reflect the expected future performance of the model. We will therefore use the accounting year as the index upon which to split data intro training and test set, using only all of one accounting year's data to train a model, or none of it.

In specific, we will set aside the final two years of data, 2013 and 2014, to use as a test set. The remaining years 2006-2012 will be used as either training or validation data (see Figure 5.1).

Also note that some of the preprocessing described in Section 5.1 depend on in-sample parameters. These parameters were estimated exclusively on the training set (or training fold, during validation), and all preprocessing used on the test set was performed using the training set parameters, avoiding any data leakage.

5.4 Temporal k-fold training and validation scheme

In order to adjust and optimize our models, we will split the training data into a set of *folds*, each consisting of one training set and one validation set. During optimization of the models, we will cycle through these folds, adjusting models and parameters to optimize the performance on the validation sets. This method is often referred to as k-fold cross-validation (Geisser, 1975) (where k refers to the number of folds), which is an extension to the leave-one-out cross-validation (Stone, 1974).

In the case of temporally distributed data, we would again prefer to avoid the problem of *time leakage*. We therefore deploy a temporal 4-fold scheme, where each fold consists of 3 years of training data, and 1 year of validation data (see Figure 5.1). By limiting each fold to always use 3 years of training data, this scheme will also allow to analyze the temporal stability of the models, a major topic in Wahlstrøm et al. (2020). Table 5.1 shows the resulting number of samples in each year.

This technique was used for tuning and optimizing all static models. Note that while the validation set performance may give an indication of the model's performance, it is not to be considered as expected out of sample-performance, as the sets were used during training to optimize the models.

5.4.1 Rolling window k-fold scheme for timeseries data

As will be discussed in Section 5.7, structuring the data as timeseries results in a reduction of the total number of samples, especially for samples with longer timeseries lengths . To ensure

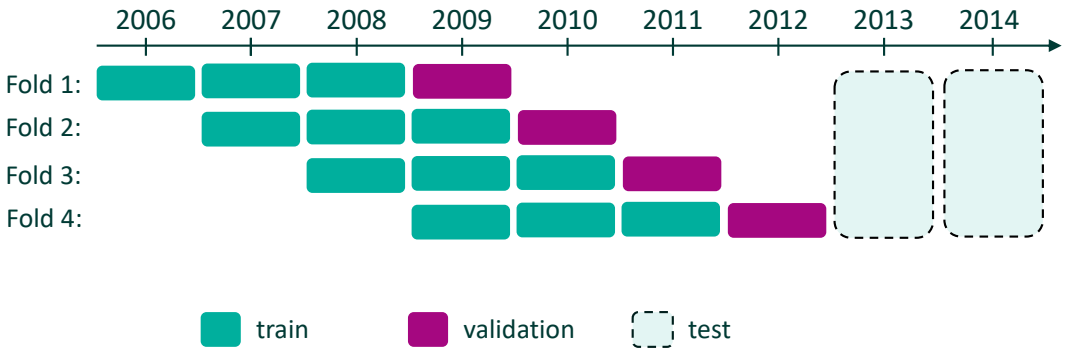


Figure 5.1: An illustration of temporal 4-fold validation scheme applied when training and and tuning the static models. Each box represents an accounting year of either training or validation data. Note that the years 2013 and 2014 are not used in this scheme, as they make up the test set.

Accounting year	Number of samples	Percentage of total data
2006	2668	8.929 %
2007	4380	14.66 %
2008	3836	12.84 %
2009	3528	11.81 %
2010	3082	10.31 %
2011	3210	10.74 %
2012	2790	9.337 %
2013	3258	10.90 %
2014	3128	10.47 %

Table 5.1: Number of samples in each accounting year (after dataset balancing), used to build training and validation sets for the fold schemes.

that we have a sufficient number of samples in each training fold for the timeseries methods, we adapt the fixed 4-year fold scheme to instead use a rolling window to select the folds.

Here, the number of years of training data is increased for each fold, starting with only 2006 as training data (validating on 2007), and then increasing the number of training years by one at each fold, using the following year as the validation year. At the final fold, all of the years $2006 - 2011$ make up the training set, with 2012 as the validation set (see Figure 5.2).

This scheme allows us to investigate the performance increase as more data becomes available at each fold, and we expect the performance to be worst for the first few folds, and increase as more data becomes available. This comes at the cost of investigating the time stability of the timeseries methods. This decision was made as we expect the fixed year fold scheme to provide too little data in each fold for the timeseries case.

5.5 Model interpretation

In all of our implementations, we will make use of the SHAP values, obtained by the methods described in Section 4.2. All methods are implemented in the `SHAP` package (Lundberg and Lee, 2017), and the specific methods used for each model is described in their respective implementation sections at the end of this chapter.

For the static methods, we will perform all SHAP analysis on models trained on data from the first fold, in order to reduce data leakage when we use the SHAP analysis for feature selection. For the timeseries methods, however, we want to to ensure that timeseries of length 4 are present in the data (see Section 5.7), and will therefore use the fourth fold to

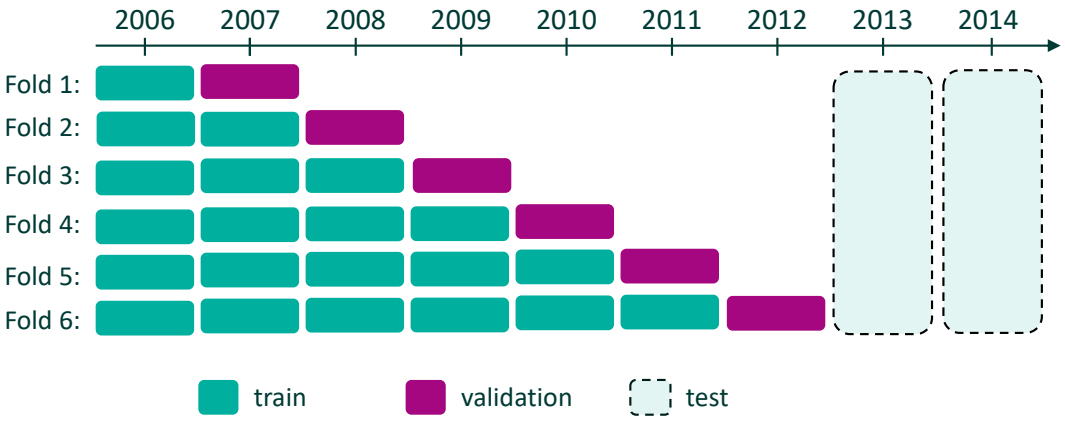


Figure 5.2: An illustration of the rolling k-fold scheme used for timeseries training, which allows us to investigate the performance increase as more data becomes available at each fold.

perform SHAP analysis, noting the potential data leakage when performing SHAP feature selection for this fold.

Note that since SHAP is used only to analyze the learned behaviour of a model *after* the model is trained, there is no inherent problem using the same training set to obtain the SHAP values (other than the considerations mentioned above). While multiple model variations will be analyzed in each of their respective sections, we will only select one model of each class to perform SHAP analysis.

5.6 Feature selection

Because the features in the training set are derived as different ratios and linear combinations of underlying accounting data, we expect a high degree of multicollinearity in at least some of the feature data. It is therefore reasonable to expect that the properties of a company is overrepresented by the original 156 features, and that a subset of features may be sufficient to represent the data, resulting in more compact and perhaps even better performing models. A lower number of features will also help practical management and quality insurance, should the models be deployed in practical contexts.

One of the goals of Wahlstrøm et al. (2020) was to compare different techniques for feature selection, as well as investigate after which number of features the model performance begins to plateau (or decrease). They implement several methods for feature selection and performed time stability analysis for each one. Extending this analysis and performing detailed feature selection analysis is considered beyond the scope of this thesis. We will, however, replicate one of the methods used in Wahlstrøm et al. (2020), as well as investigate the use of SHAP values for feature selection. We will also use their general finding in that the performance increased quickly around 15 features, and started to plateau as the number of features approached 30. We will therefore be considering feature subsets of 15 and 30 features, which should give accurate pictures of the performance of models with different degrees of compactness.

5.6.1 Wrapper method

The method from Wahlstrøm et al. (2020) that we will replicate is called the *wrapper method* (John et al. 1994; Kohavi and John 1997), which aims to select the best possible feature subset by sequentially considering adding (and removing) features, and then evaluating whether this offers improvements or not by fitting models to each of the different feature subsets. Conceptually, this method greedily adds (and removes) features in order to search for the best possible feature subset.

Due to its exhaustive search nature, this method is computationally expensive, requiring multiple models to be fitted at each step (Chandrashekar and Sahin, 2014). Therefore, we will only implement it for logistic regression, as it is the fastest model to fit.

5.6.2 SHAP importance feature selection

For the other models, we will make use of SHAP feature importance for feature selection, described in Section 4.2.7. Here, we simply select the d most important features, ranked by the mean of the absolute SHAP values of the features. This method has some drawbacks, as it does not always account for multicollinearity, meaning we risk subsetting features that may be heavily correlated. It is, however, deemed sufficient for our purposes.

As mentioned in Section 5.5, the SHAP values will be obtained on a model trained on the first fold (years $2006 - 2009$) for the static methods, and the fourth fold (years 2006-2010) for the timeseries methods. For even more robust results, one could perform separate SHAP analysis for each of the folds and perform feature selection based on the aggregation of these. However, using only one fold for feature selection was deemed sufficient for our purposes.

5.7 Timeseries modelling

In practical timeseries prediction applications, sufficient care has to be taken when structuring data into timeseries, making sure to avoid problems such as data leakage and bias resulting from the construction process of the timeseries. In this section, we will describe in detail the considerations made when structuring the financial statement data into timeseries, as well as highlighting the potential pitfalls.

The general objective and problem formulation remains the same as before; we are looking to predict the bankruptcy of a company, given the historically available accounting data. However, rather than using the only the single previous year of accounting data, we will now be constructing timeseries of multiple years, using the same set of financial ratios as features.

Because of the terminal nature of our timeseries data (accounting data generation ends when a company becomes bankrupt), some specific considerations has to be made when we construct our dataset of timeseries samples, especially as the age of the company is a feature in our dataset (an important one at that, as we will see from the SHAP analysis in Section 6.5). We will therefore want a model framework that can handle mixed timeseries lengths, which motivates our focus towards the RNN and LSTM models.

5.7.1 Maximum length of timeseries

As our dataset covers a time span of 8 years, this is an obvious upper ceiling for the length of the timeseries we can construct. To ensure sufficient data in both the training and test sets for our relatively data-hungry neural network timeseries methods, we will only investigate timeseries with a maximum length of 4, as data availability generally goes down as we require more (connected) accounting data points in the time dimension. This also allows for more flexibility in our temporal k-fold schemes, discussed in Section 5.4.

While greater lengths would be interesting to investigate, given both the data availability restrictions and practical context, we deem this number sufficient for both capturing the economic developments of companies, and investigating the gain in predictive power in adding such information.

5.7.2 Timeseries dataset balancing considerations

While it may be tempting to structure a timeseries sample as the maximum number of available years of accounting data for the company, this can cause problems in the training data distribution, especially in the context of the undersampling we apply during dataset balancing, discussed in Section 5.2.

As we will see in Section 6.5, one of the most predictive features for all models is the $log(age\ in\ years)$; younger companies are generally more likely to go bankrupt. When we

then match a bankrupt company (which is more likely to have shorter available timeseries) to a non-bankrupt one (which is likely to be older) during the undersampling process, we would end up with a strong bias in the connection between the length of the timeseries, the age of the company, and the target (bankrupt/non-bankrupt), induced by the artificial sampling and timeseries structuring process.

To counteract this tendency, we deploy two techniques when structuring our dataset as timeseries and slightly alter the dataset balancing process in Section 5.2.1:

1. When selecting which timeseries length to use for a (minority) sample, we select the length randomly between 1 and $\min(4, l_{max})$, where l_{max} is the maximum timeseries length available (i.e. number of available connected years of financial statements) for the minority sample.
2. We then select a majority timeseries sample using the same matching criterion outlined in Section 5.2.1, while also requiring it to have the same length as the chosen minority sample.

Other than these alterations, the balancing process is the same as in Section 5.2.1.

This process ensures both that each of the timeseries lengths are equally balanced in the target outcome (i.e. timeseries of length 1, 2, 3, and 4 each have the same number of bankrupt/non-bankrupt cases), and that there is no inherent bias between between the timeseries length and target variable. The resulting number of timeseries samples for each length within each respective account year is shown in Table 5.2.

Accounting year	Length 1	Length 2	Length 3	Length 4
2006	2668.0	-	-	-
2007	2710.0	1670.0	-	-
2008	1860.0	1140.0	836.0	-
2009	1540.0	868.0	672.0	448.0
2010	1360.0	796.0	540.0	386.0
2011	1414.0	800.0	556.0	440.0
2012	1054.0	760.0	512.0	464.0
2013	1570.0	814.0	510.0	364.0
2014	1520.0	736.0	488.0	384.0

Table 5.2: Number of samples with the different timeseries lengths in each accounting year, after our timeseries dataset balancing. Naturally, the timeseries lengths of the first three years are limited by the data availability.

5.8 Model implementations

In this section, we will detail all of the specific model implementations used in the experiments, to produce the results in Chapters 6 and 7. For the static methods, we will detail three different models with their respective implementations. The timeseries methods, specifically an RNN and an LSTM neural network, will be described collectively, as their implementations share many similarities.

5.8.1 Logistic regression implementation

All logistic regression models were implemented through the library `scikit-learn` (Pedregosa et al., 2011). They were trained until convergence using a stopping criteria tolerance of 0.001, using the `newton-cg` solver method (implementing Newton's method with the exact Hessian, (Hastie et al., 2001)) with a log loss function (Section 3.2).

While Wahlstrøm et al. (2020) used no regularization in most of their experiments (with the exception of one feature selection technique), our initial experiments found that some

L_2-regularization (see Section 3.2) gave improved results, especially in the case of models that used one-hot encoded features. This is reasonable, as regularization helps with over-parametrization caused by increasing the dimensions of the feature space. For consistency and to facilitate easy comparison, all of our experiments will use L_2-regularization with regularization parameter $\lambda = 1$, which was experimentally found to be a good value (although the results were little sensitive to particular choices of λ).

We will fit and test logistic regression both with and without the categorical variable $nace\_code$, both with one-hot encoding and ordinary target encoding, as described in Section 3.3.4).

Logistic regression feature selection

For feature selection, we use the wrapper feature selection method deployed in Wahlstrøm et al. (2020) (described briefly in Section 5.6), and fit logistic regression models on feature subsets of 15 and 30 features. We also perform a SHAP feature importance analysis (the SHAP analysis will be detailed in Section 6.5), and select feature subsets of the 15 and 30 top features, ranked by SHAP feature importance. Because the feature importance of one-hot encoded features becomes distributed across all the encoded features (in a not necessarily representative manner), we will use the target encoded case for estimating the SHAP values.

SHAP implementation for logistic regression

As stated in Section 4.2.4, applying model explanation for linear (or logistic) regression is simple, assuming feature independence. As we expect multicollinearity in the feature data, we instead use the `LinearExplainer` of the `SHAP` package, using the correlation dependent features setting (which implements the additional considerations outlined in Section 4.2.4.

5.8.2 Neural network implementation

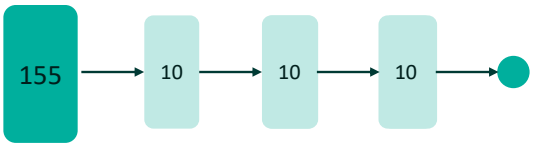
As outlined in Section 3.4, the performance and behaviour of neural networks depend on their architecture. For our experiments, we will replicate the general neural network architecture of Wahlstrøm et al. (2020), which used $M = 3$ hidden layers, with $k_1 = k_2 = k_3 = 10$ hidden nodes in each layer, and sigmoid activation functions. While we explored more advanced architectures, they showed no substantial improvements over the more simple architecture of Wahlstrøm et al. (2020), indicating that this architecture is able to capture most of the processes, given the available amount of data.

Although Wahlstrøm et al. (2020) used a linear activation in the final layer, we will use a sigmoid activation function here as well, to ensure the output is constrained to the $(0, 1)$ interval. We will test this architecture with and without one-hot encoding, and target encoding, of the $nace\_code$ feature.

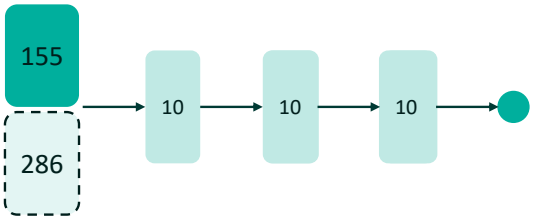
We then extend the architecture by introducing a categorical feature embedding of the $nace\_code$ feature. A good value for the embedding dimension was found experimentally to be $d = 3$; higher values were found to cause overfitting (see Figure 5.3c).

All neural networks were implemented using the `Keras` library (Chollet et al., 2015). Unless otherwise specified, the default parameters were used to train the network. In specific, the network was trained using stochastic gradient descent with a learning rate of 0.00025 (see Section 3.1.1), with a batch size of 32, which was found to be reasonable values. All weights are initialized with a glorot uniform distribution (see Section 3.4.4).

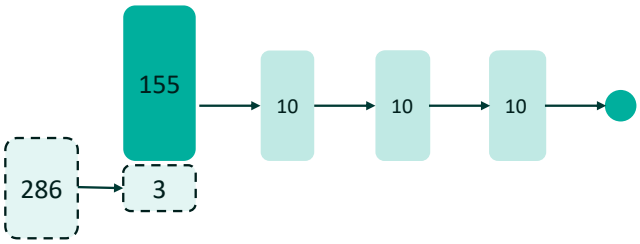
We train the networks for a maximum of 100 epochs (where 1 epoch corresponds to the number of batches required to go through the whole training dataset), although we monitor the training and validation losses, stopping the training using an early stopping criterion, described in Section 3.4.4. The criterion stops training if the validation loss has not improved for 5 epochs. Note that this may make the results reported on the validation folds slightly optimistic, as the validation sets are used during training.



(a) Neural network with only the 155 numerical features.



(b) Neural network with one-hot encoding for $nace\_code$, resulting in 286 additional input features.



(c) Neural network with an embedding layer for $nace\_code$, using embedding dimension 3.

Figure 5.3: The implemented neural network architectures, each consisting of 3 hidden layers with 10 nodes and sigmoid activation functions at every layer. The edges between the layers are represented by a single arrow. The left rectangles represent the inputs to the networks, which consists of 155 base (numerical features), and an additional 256 (the cardinality of $nace\_code$) features in the case of onehot encoding. For the embedding network, these features are embedded to just 3 embedding features. Not depicted is the target encoding, which is essentially an embedding layer with embedding dimension 1.

SHAP implementations for neural networks

For neural networks, we obtain approximate SHAP values using the `KernelExplainer` of the `SHAP` package, which is built on the KernelSHAP method described in Section 4.2.5. As the method greatly increases in sample time as the number of samples over which (4.7) is minimized, we use a sample of 1000 data points from our training set over which for every sample, we minimize this equation.

5.8.3 CatBoost implementation

We implement the gradient tree boosting algorithm using the `CatBoost` package (Prokhorenkova et al., 2018), which implements the additional concepts of ordered boosting described in Section 3.3.3.

The `CatBoost` package offers a wide variety of hyperparameters used for both ensemble and individual tree specification, and model regularization. For a set of these, we use the random search hyperparameter optimization implemented in the package, on the temporal validation structure. The obtained values are detailed in Table 5.3. For the parameters not listed here, the default values were used.

Parameter name	Explanation	Value
num_boost_round	Maximum number of trees to fit (iterations)	10000
learning_rate	See Section 3.1.1	0.005
bootstrap_type	Affects tree splitting (Prokhorenkova et al., 2018)	'Bernoulli'
max_depth	Maximum allowed tree depth (number of splits)	5
subsample	See Section 3.3	0.93
early_stopping_rounds	After which number of iterations without validation set improvement to stop training	250

Table 5.3: Parameters used for the models trained using the CatBoost package, obtained using the built in random search procedure. The bottom three parameters are regularization parameters. For parameters not listed here, the default parameters were used.

SHAP implementation for CatBoost

We obtain SHAP values for the CatBoost models using the TreeExplainer of the SHAP package, implementing the TreeSHAP method described in Section 4.2.6.

5.8.4 RNN and LSTM implementations

For the timeseries methods, we will implement and analyze the performance of both RNNs, described in Section 3.5.1, and LSTMs, described in 3.5.3. Due to their similar implementations, we will describe them collectively in this section.

The network architectures deployed are somewhat similar to those of the neural networks in 5.8.2, although they necessarily include RNN and LSTM layers, each with 20 nodes. These layers are used to learn the time dependent structure of the data, with two added layers of fully connected layers of top, each containing 10 nodes. In the context of "feature extractors" and "estimators" discussed in Section 3.4, the RNN/LSTM layers can be though of as timeseries feature extractors, while the two following layers combine the timeseries features to produce the final prediction.

Interestingly, tanh activation functions were found to work better than sigmoid activation functions, and are therefore used throughout the timeseries networks. This can perhaps be attributed to the exploding or vanishing gradient behaviour of recurrent types of neural networks, discussed in Section 3.5.

Rather than investigating multiple categorical feature encodings as for the static methods, we will build on the findings from these (as will be seen in Chapter 6), and use only a target encoding of the $nace\_code$ feature. This allows us to focus our analysis on the properties and behaviour specific to the timeseries formulation of the problem, as we do not expect the categorical variable behaviour to change significantly as we move to the timeseries problem.

As with the neural networks in Section 5.8.2, we implemented both the RNN and LSTM networks using the Keras library, using the same general parameters as in described in Section 5.8.2: We use stochastic gradient descent with a learning rate of 0.00025, a batch size of 32, and the glorot uniform distribution weight initialization. Again, we experimented with different values for these parameters, but as they offered little to no improvement, we used identical parameters, for consistency.

As before, we train the networks for a maximum of 100 epochs, using the same validation stopping criterion after 5 consecutive epochs without validation loss improvement. We note again that this means the validation set results in Section 7.1 may be slightly overfit, referring realistic out of sample performance to Section 7.2.

SHAP implementation for RNNs and LSTMs

Similarly as for the static neural network, we obtain SHAP values for the RNN and LSTM models using the `KernelExplainer` of the `SHAP` package. Again, we use a sample of 1000 points from the training set over which the expectation is minimized.

Chapter 6

Static method experiments and results

In this chapter, we will detail the implementations and variations of the static models we consider, and present and discuss the results. While we will generally report all the metrics described in Section 4.1, (using a classification threshold $\tau = 0.5$ where relevant), our main focus will be the AUC score, due to its desirable properties and consistent use in previous literature (Wahlstrøm et al., 2020; Bernhardsen and Larsen, 2007; Tian et al., 2015).

In Sections 6.1, 6.2, and 6.3, we will detail and present the results each of the considered static models separately, tested on the temporal validation fold setup described in Section 5.4, using balanced datasets (following the balancing procedure described in Section 5.2). Note that the specific parameters and architecture of the models were tuned to optimize performance on these validation sets, meaning the results are only indicative of the models' performance, while not necessarily reflecting true expected performance on unseen data.

For these methods, we will compare some variations of each of the models separately. We will mainly consider different choices of handling the novel $nace\_code$ categorical variable, described in Section 2.3.5, and different feature subsets of 15 and 30 features. The latter choices are motivated by the findings of Wahlstrøm et al. (2020), who generally found plateauing performance after around 20 features. Thus, we expect the 15 feature subsets to provide examples of more compact models, while the 30 feature subset variations can hopefully capture most of the relevant information and perform close to the full (156) feature versions.

For the reproduced methods (logistic regression and neural networks), we will also briefly compare them to the results of Wahlstrøm et al. (2020). However, they only report some specific metrics, often in the format of graphs, making exact comparisons difficult. Also note that as they only report validation fold performance, we will not be incorporating these results into our test set analysis in Section 6.4. As they are the only work that use the same dataset, it is the only relevant comparison we can make.

This validation set analysis is meant to motivate our particular choices of feature subsets and categorical variable encodings, to be used when we compare all of the model classes on the test set (comprised of 2013 and 2014 data) in Section 6.4. Here, we will compare results on both the balanced and unbalanced datasets, but note that as addressing the unbalanced data problem in any detail is beyond the scope of this thesis, we will only demonstrate the unbalanced dataset results, refraining from addressing in detail any of the issues that are revealed.

Finally, in Section 6.5, we will demonstrate how model interpretation can be performed, doing global SHAP value analysis of each of the models, and discuss and compare some of their learned behaviours. While SHAP analysis allows for individual feature explanation, we will rather focus on a global SHAP comparison of the models, described in Section 4.2.7,

as analysis of specific features would require more qualitative considerations and are thus considered beyond our scope.

A note on model variations

While we did perform multiple experiments for different model variations, hyperparameter setups and model architectures, we will not go into depth in describing and analyzing the individual experiments for each of these, as the amount of results would quickly blow up and it is not considered the main focus of this thesis. We will rather give a parsimonious presentation of each of the model types' performance and general behaviour, and focus on a comparison between feature subsets and ways of handling the novel categorical feature. We note that the specific variations we present are generally the best that we found for each of the respective model types.

A note on computation time

Although computation and training time are important aspects of machine learning methods, we will not be reporting the computation time of the experiments in this (and the following) chapter. This is partly because computation time depend heavily on the particular equipment used to perform the experiments, as well as the implementations used for the experiments, which may be unoptimized. Computational optimization is thus considered beyond the scope of this thesis.

6.1 Logistic Regression results

The first method we will analyze is logistic regression, for which theoretical foundations are detailed in Section 3.2, and implementations described in Section 5.8.1. While somewhat simple, it has proven to be effective for bankruptcy prediction, being both quick to optimize and easy to interpret.

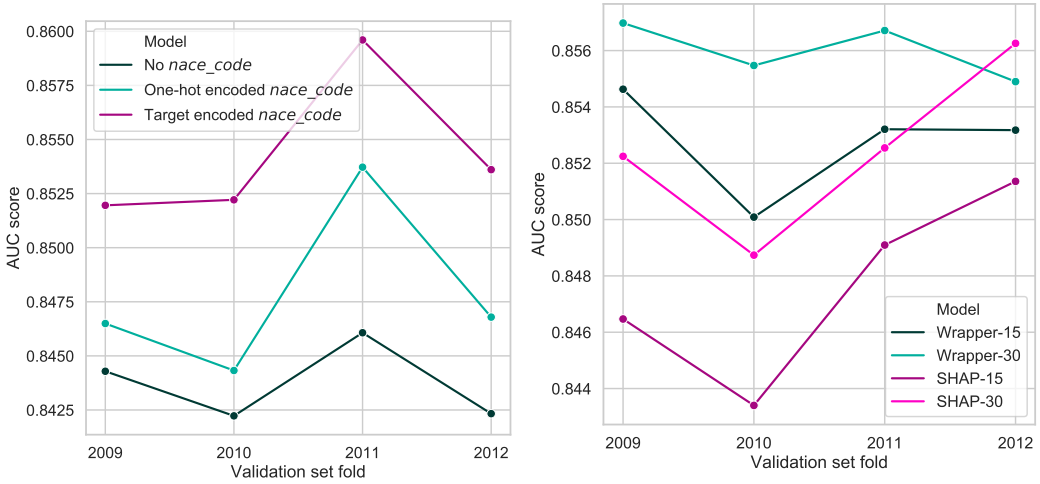
The experiments in this section will consider three full feature models with variations for the categorical variable, as well as four different feature subsets, described in Section 5.8.1. For the latter, a target encoding was used for the categorical feature. We will then study the temporal stability of these variations. Again note that these are results on the validation sets in the temporal k-fold scheme, described in Section 5.4.

	Accuracy score	Brier score	F_1-score	AUC score
No $nace\_code$	0.766212	0.160905	0.766906	0.843729
One-hot encoded $nace\_code$	0.770381	0.159070	0.770577	0.847833
Target encoded $nace\_code$	0.777890	0.155054	0.778798	0.854344
Wrapper-15	0.778920	0.156009	0.776897	0.852774
Wrapper-30	0.782399	0.154096	0.780907	0.856015
SHAP-15	0.773927	0.158525	0.775339	0.847581
SHAP-30	0.777411	0.155894	0.777341	0.852446
Wahlstrøm et al. (2020) (15)*	-	-	-	0.842

Table 6.1: Validation results for different implementations of logistic regression trained on different feature subsets. The scores are computed as the average scores on the validation sets over the 4 temporal folds. The first three models are trained on full feature subsets, while the last four are trained on different feature subset variations (with target encodings). For the former, using a target encoded $nace\_code$ outperforms the other full feature models, while the Wrapper-30 model gives the best overall performance. For the results of Wahlstrøm et al. (2020), only the AUC score (for 15 features) is available; the * indicates that is retrieved visually from a graph, and therefore is not exact.

6.1.1 Categorical variable encoding results

Table 6.1 shows that target encoding the $nace\_code$ feature gives comparatively better results than one-hot encoding, while leaving it out entirely worsens performance. The differences are, however, somewhat marginal, with the greatest relative AUC difference being 1.2581%. However, target encoding offers a more compact and more interpretable model, making it preferable over one-hot encoding. We also note that we see similar performance between the no $nace\_code$ model and the 15 feature model from Wahlstrøm et al. (2020).



(6.1a) Logistic regression models with different categorical variable encoding. Target encoding greatly outperforms the others in all folds.

(6.1b) Logistic regression models with trained on different feature subsets. Interestingly, the SHAP methods increase their performance in the last fold.

Figure 6.1: AUC scores for each of the folds for each of the implemented logistic regression models. All models tend to follow the same patterns, and there are not any apparent decrease in variation in the feature subset models.

Figure 6.1a shows the AUC score for each of the validation folds for the logistic regression models, indicating the time stability of the methods. The models generally behave the same across the validation folds, although the models that include the $nace\_code$ feature tend to have higher variation between the folds. This may indicate that increasing the number of features can result in greater variation across the folds. The variations are, however, relatively small, with the largest relative difference between folds being 0.648% in terms of AUC.

6.1.2 Feature subset results

In Table 6.1, we see that feature subsets only offer strict performance improvements in the case of 30 features selected by the wrapper algorithm, which gives a 0.234% increase in AUC score compared to the full feature models. The differences between the models are, however, marginal, and if compactness is preferable, any of the models could be considered over the full feature subset model. This shows that more compact models are indeed comparative to the full feature models, and that using the full feature set may make optimization more difficult, or include more noise.

Comparing our results to the 15 feature subset results of Wahlstrøm et al. (2020), we observe similar, although slightly higher performance. This may be attributed to the fact that Wahlstrøm et al. (2020) does not use a categorical variable, which is shown to have strong predictive power (see SHAP importance in Figure 6.10). Also note that this AUC score may be inaccurate, as it is retrieved visually from a graph.

Figure 6.1b shows the subset models' validation set performance in terms of AUC score, on all of the aforementioned methods. Generally, the Wrapper methods offer greater stability across the validation folds, especially so the Wrapper-30 method, making it preferable in terms of both performance and stability.

6.2 Neural network results

In this section, we will describe our replication and variations of the neural network models in Wahlstrøm et al. (2020). The theoretical foundation for these models was described in Section 3.4.

The 3-layer neural network is implemented as described in Section 5.8.2 and tested on different categorical encoding variations and feature subsets (again, with a target encoding used for the latter). While we explored more advanced architectures, they showed no substantial improvements over the more simple architecture of Wahlstrøm et al. (2020), and we will conform to this general architecture.

6.2.1 Categorical variable encoding results

Table 6.2 shows the average validation fold performance of the three different network variations, as well as the SHAP feature subset variations. We see that including the $nace\_code$ feature improves the performance in terms of all metrics, although the differences between the one-hot encoding and embedding are marginal. Again, the relative differences are small, the greatest AUC increase being only 0.5337%.

Figure 6.3a shows the time stability of the encoding variations across the different folds. While having worse performance, the network with no $nace\_code$ is by far the most stable. The embedding neural network tends to show the most variation, with a huge drop in performance between 2011 and 2012, where interestingly, the target encoding variation improves. This instability may imply that for some folds, the network is not able to properly learn useful embeddings.

Conceptually, the embedding network has the capacity to perform at least as well as the target encoded network. To see this, note that the target encoding can be thought of as an embedding with dimension 1 that performs a mapping between the categories and the corresponding mean target value in the training set. The fact that the embedding performs worse, highlights the difficulty of learning for neural networks, indicating either that the network converges towards other representations (that could e.g. be overfitted), or that the data in each fold is insufficient for the network to learn this relationship by itself.

	Accuracy score	Brier score	F_1-score	AUC score
No $nace\_code$	0.771885	0.159656	0.777536	0.847455
One-hot encoded $nace\_code$	0.773408	0.157548	0.778565	0.851275
Target encoded $nace\_code$	0.774292	0.157200	0.779236	0.851594
Embedding $nace\_code$	0.774955	0.158904	0.779404	0.850157
SHAP-15	0.770095	0.162182	0.773378	0.841026
SHAP-30	0.775957	0.158046	0.779595	0.849785
Wahlstrøm et al. (2020) (15)*	-	-	-	0.843

Table 6.2: Neural network model results. The target encoded neural network outperforms the other models in terms of AUC and Brier score, while the more compact SHAP-30 model performs best in terms of accuracy and F_1-score. For the results of Wahlstrøm et al. (2020), only the AUC score (for 15 features) is available - the * indicates that is retrieved visually from a graph, and therefore is not exact.

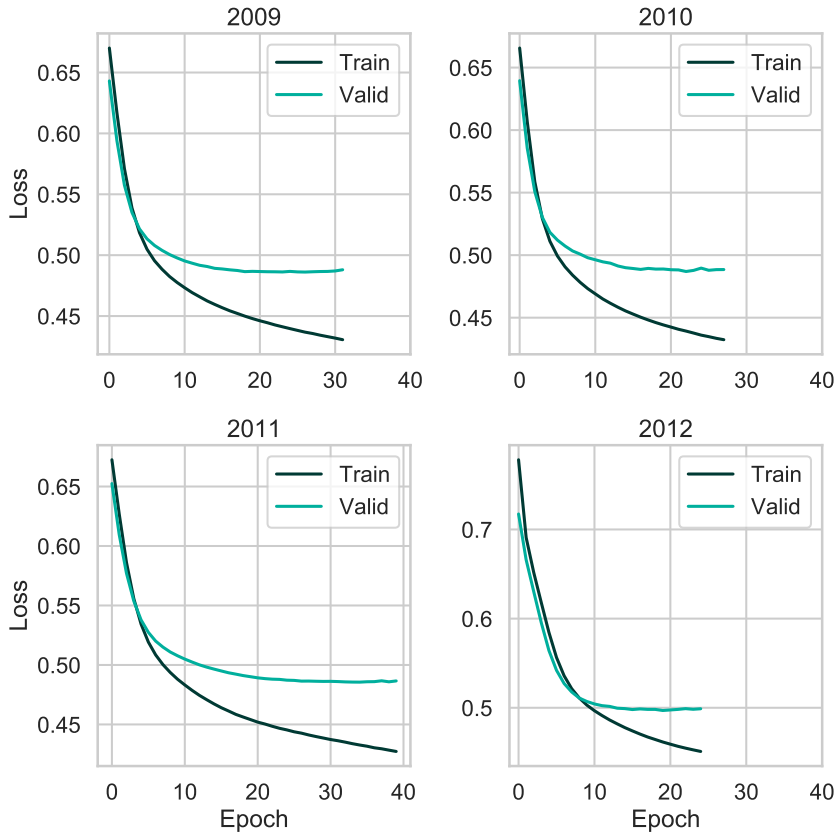


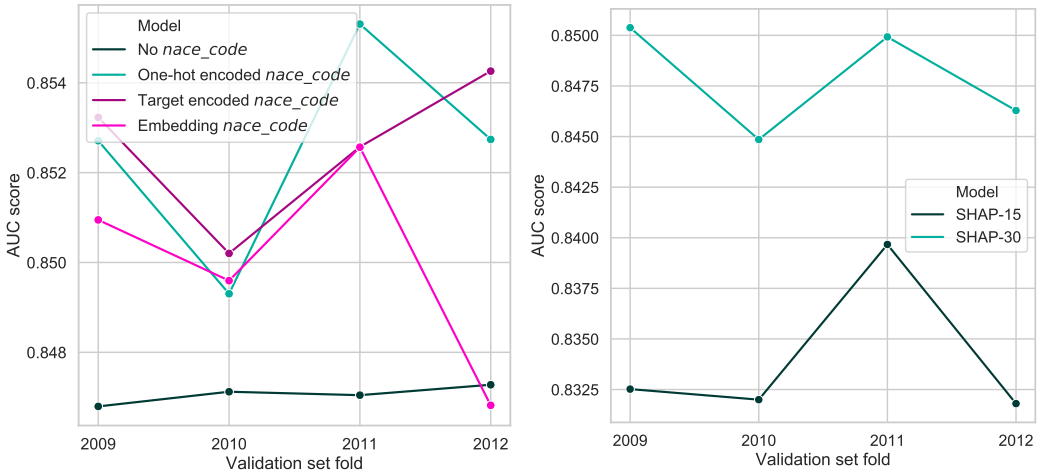
Figure 6.2: Plots of the training and validation loss progression for each of the validation folds, here examplified for the target encoded $nace\_code$ full feature neural network. Interestingly, the validation loss is lower than the training loss for the first few epochs, before the validation loss flattens out and the training is stopped as per the early stopping criterion, typically after 30-40 epochs.

6.2.2 Feature subset results

The lower part of Table 6.2 shows the performance of the feature subset neural network models. We see that while using a feature subset of 15 results in all metrics declining, using 30 features results in marginal improvements in the F_1-score and accuracy score, with slightly worse performance on the Brier score and AUC score, comparing to the target encoding full feature subset model. Again, this confirms the findings of Wahlstrøm et al. (2020) that using more than 15 features holds significant value, while the improvements starts to plateau after this point. As less complex models are generally to prefer over more complex ones, such marginal differences could cause one to prefer the 30 feature neural network model.

Comparing these results to the neural network results of Wahlstrøm et al. (2020), we again see similar - although slightly improved performance. Similarly as for the logistic regression, this may be attributed to the presence of the $nace\_code$ variable, or slight differences in architecture (for the final layer) or implementations.

Figure 6.3b shows the time stability of the neural network feature subset models. Again, we see the same general behaviour between folds, with a decline between 2011 and 2012, indicating some inherent differences in either the difficulty of predicting the year 2012 compared to 2011, or that perhaps the preceding training data is less representable of the following year in some of these folds.



(6.3a) Neural networks with different categorical variable encoding. No $nace\_code$ has the least variance, while target encoding performs best.

(6.3b) Neural networks with trained on different feature subsets. The behaviour is generally the same, with 30 features clearly performs better.

Figure 6.3: AUC scores for each of the folds for each of the implemented neural networks.

6.3 CatBoost results

One of the novel contributions of this thesis is the application of the CatBoost algorithm to bankruptcy predictions. While similar models such as random forests (Tian et al., 2015) and to some extent XGBoost (Zięba et al., 2016) has been used, no previous works have applied the more advanced gradient boosting tree models, especially so on a dataset of this magnitude. In this section, we will detail our results using the same framework as for logistic regression and neural networks in the preceding sections, comparing categorical variable encodings and SHAP feature subsets, following the implementation described in Section 5.8.3.

To demonstrate the efficiency of the native ordered target encoding of the CatBoost framework (described in Section 3.3.4), we will also include the results of an ordered target encoding in this section. For consistency with the other models, however, the feature subset models will still use the ordinary target encoding of the categorical feature.

6.3.1 Categorical variable encoding results

From Table 6.3, we see the same general behaviour seen in logistic regression and target encoding in that the model performs better in the presence of $nace\_code$, with AUC decreasing between $0.1763\% - 0.3131\%$. The ordered target encoding marginally outperforms ordinary target encoding in all metrics, indicating that the bias correction of ordered boosting improves generalization ability, albeit just slightly. Interestingly, ordered target encoding only outperforms one-hot encoding in AUC and Brier score, but not in accuracy and F_1-score. Again, the differences are, however, marginal.

Looking at Figure 6.4a, all models generally emit the same behaviour and show the same level of variation across the temporal folds, although the one-hot encoded variation shows perhaps the least variance. If temporal stability is important, this model should perhaps be considered over the target statistics encodings.

6.3.2 Feature subset results

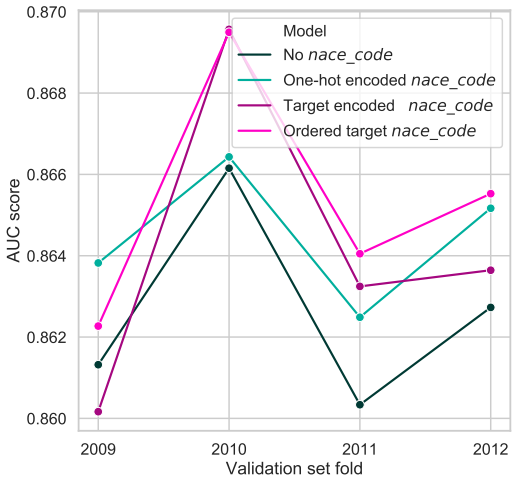
The results on the different feature subsets reflect the same behaviour as seen for the previous models, with significant declines in all metrics for the 15 feature subset model. For the 30 subset model, the performance is comparable to the full model, although very slightly worse

than the best full feature model (0.01756% decrease in AUC). Again, one may prefer the 30 feature model if a more compact model is desirable.

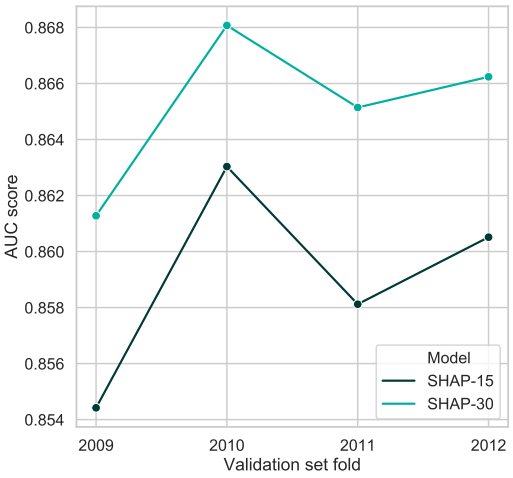
For temporal stability, shown in Figure 6.4b, the 30 feature subset model seems to show slightly less variance, while the general behaviour between the models are the same. As before, the 30 feature subset model is consistently better performing than the 15 feature subset model.

	Accuracy score	Brier score	F_1-score	AUC score
No $nace\_code$	0.781589	0.150423	0.786080	0.862634
One-hot encoded $nace\_code$	0.785318	0.149260	0.789093	0.864477
Target encoded $nace\_code$	0.783126	0.149695	0.786454	0.864155
Ordered target $nace\_code$	0.784727	0.148896	0.788601	0.865335
SHAP-15	0.779719	0.152709	0.784453	0.859022
SHAP-30	0.784196	0.149138	0.788654	0.865183

Table 6.3: Average validation fold metrics for all CatBoost model variations. Again, we see a clear improvement for all encoding variations with introduction of the $nace\_code$ feature. We also see comparative performance for the more compact SHAP-30 feature subset.



(6.4a) CatBoost models with different categorical variable encodings. As expected, ordinary and ordered target encoding are very similar, with the latter performing slightly better..



(6.4b) CatBoost models with trained on different feature subsets. As before, the SHAP-30 subset is consistently better performing.

Figure 6.4: AUC scores for each of the validation folds for each of the implemented CatBoost models. As before, all models generally emit the same patterns between the folds.

6.4 Static methods test set results

In this section, we will perform comparisons of each of the static methods described above, on the previously unseen test set of data from 2013 and 2014, reporting the results on both the balanced and unbalanced data set in respective sections. The models are trained on the remaining years $2006 - 2012$, with the neural network and CatBoost methods using the year 2012 as a validation year for the early stopping criterion.

Motivated by the findings in the previous sections, we will fit each of the models with a target encoding of $nace\_code$, as it generally gave the best results. The inclusion of target encoded $nace\_code$ in the feature set will therefore be implicit in the results that follow.

As seen in the previous sections, the 30 feature subset models selected by SHAP generally gives either the best, or close to best results, in terms of validation set performance, while using a significantly fewer features compared to the full feature models. Motivated by this, we will be using these models to report our test set results in the following sections, for consistency.

6.4.1 Balanced test set results

We will first report the results on the balanced test set, for which the models are optimized. While giving a useful comparison of the models performance, we stress again that these results do not reflect expected out of sample performance, as the undersampling shifts the data distribution significantly.

Evaluation metric performance

Figure 6.6 shows a comparison of the metrics for each of the models on the test set. These results reflect the findings of the previous sections, with CatBoost consistently outperforming the others in all metrics; CatBoost has a 2.571% and 3.532% greater AUC score than the neural network and logistic regression models, respectively

The lower Brier score also implies that CatBoost is more (correctly) confident in its probability predictions, meaning its probability estimates are generally more reliable (on the balanced dataset). This is reflected in Figure 6.5, where we see that CatBoost has generally more confident predictions (i.e. distributed close to 0 or 1) than logistic regression.

Model prediction probability distribution

While the undersampling used to obtain the balanced dataset distorts the interpretability of the model predictions as "probabilities", well distributed predictions (i.e. (correctly) distributed close to either 0 or 1) are generally a desirable property, as it increases the informativeness of the models' outputs. From Figure 6.5, we see that all models produce the same general, U-shaped distribution. This is generally more desirable than flat or inverted U-shaped distributions, as it implies more confident and therefore informative predictions.

We also see that the neural network and CatBoost models have even more similar prediction distributions, both seemingly reluctant to give very confident bankrupt predictions (predictions very close to 1), while having a comparatively larger amount of confident non-bankrupt predictions (predictions close to 0). The fact that both models have learned this behaviour may indicate that companies that are very unlikely to go bankrupt are easier to identify, while confidently predicting companies as bankrupt is more difficult. This can perhaps be attributed to the inherent noisiness of the bankruptcy target variable, discussed in Section 2.3.3.

The lack of this behaviour in the logistic regression model output may be attributed to its reduced model flexibility compared to the other models, making it unable to "learn" the same relative decrease in the highest prediction probability interval.

6.4.2 Full (unbalanced) test set results

In this section we will report each of the models' performance on the full test set. While the models have not been optimized or tuned for this (unbalanced) distribution of target samples, these results will give a realistic view of the models' expected out of sample performance.

Note that all minority (i.e. bankrupt) samples in the balanced dataset are also included in the full set - the only difference is a significant increase in the number of majority (non-bankrupt) samples.

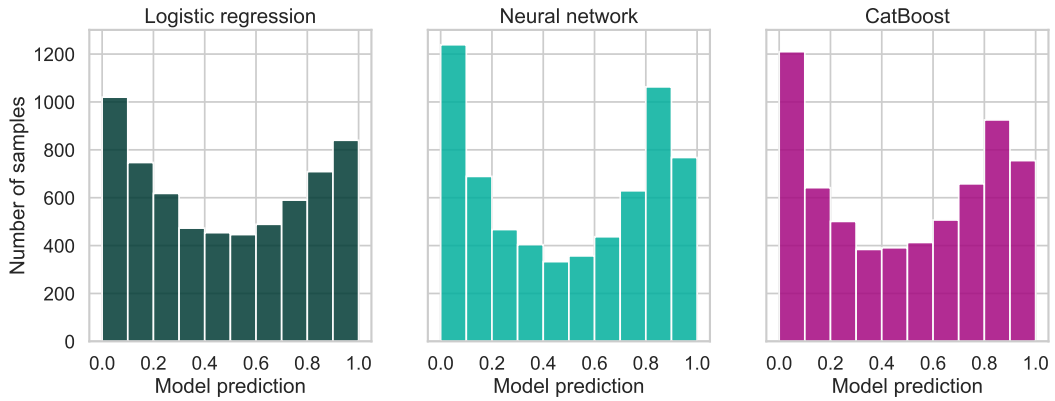


Figure 6.5: The prediction probability distributions of each of the models, with each bar representing an interval of length 0.1. Interestingly, the neural network and CatBoost models produce similar distributions, both confidently predicting near 0 probabilities for bankrupt companies while avoiding the near-1 interval.

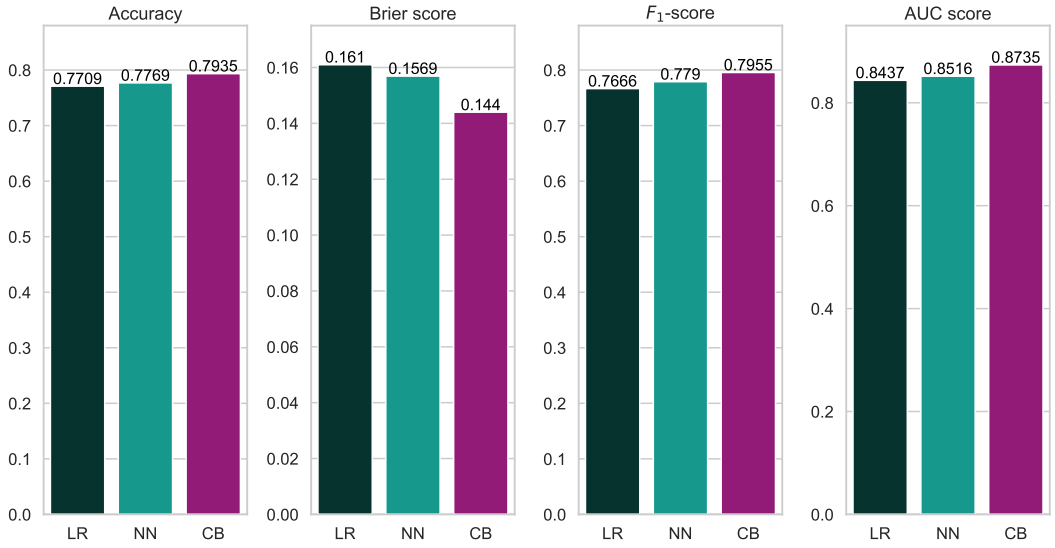


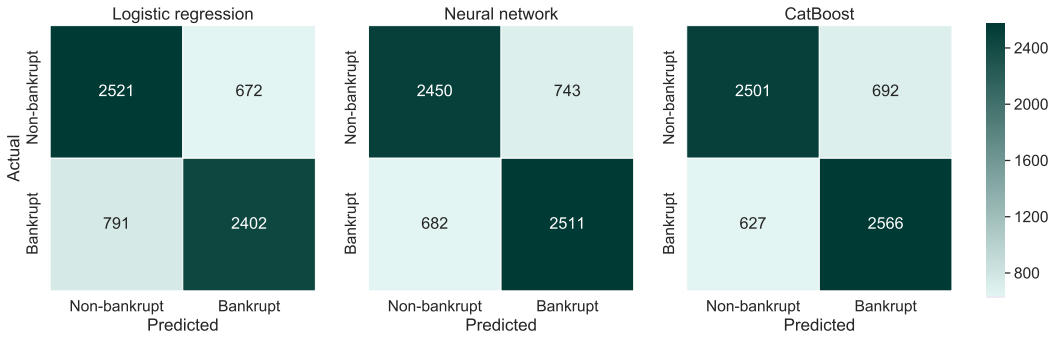
Figure 6.6: A comparison of the metrics of each of models on the balanced dataset. While the performance is similar, CatBoost (CB) consistently achieves better metrics, while logistic regression (LR) performs the worst of the compared models.

Evaluation metric performance

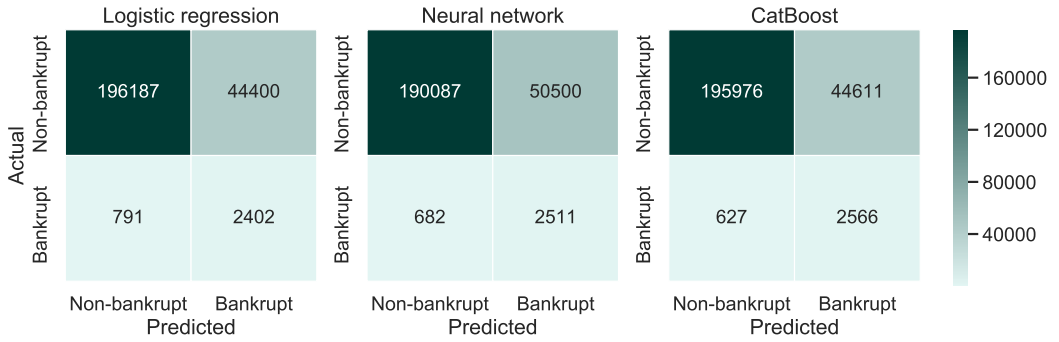
Figure 6.8 shows the resulting metrics for each of the models. We see the same general trend of CatBoost outperforming the other models, except for accuracy, for which logistic regression slightly outperforms CatBoost - confirming the earlier indications of CatBoost being the superior model of the ones considered here.

Interestingly, logistic regression now outperforms the neural network in all metrics, except for the AUC score. This may indicate that the neural network does in fact overfit for some of the minority samples, with logistic regression being able to better generalize these samples. However, these results are indecisive.

Note that while the accuracies seem to have improved, a naive model predicting only non-bankrupt samples will receive an accuracy score of 0.985. This illustrates the shortcomings of the accuracy metric, as beating this baseline will be very difficult for any model.



(a) Confusion matrices on the balanced test set. The predictions of all models generally behave well, with comparable (and relatively symmetrical) number of type 1 and type 2 errors.



(b) Confusion matrices on the full test set. The models tend to predict more non-bankrupt companies as bankrupt (type 2 error).

Figure 6.7: Confusion matrices of each of the methods on the balanced and full test set. We have included a color grading to amplify the differences between the two cases. In the balanced dataset, we see expected behaviour with most samples being correctly classified. In the unbalanced case, we see that the model tends to predict more non-bankrupt companies as bankrupt. Note that the lower rows of the plots (actual bankrupt) are the same, as all actual bankrupt samples in the full dataset, are also contained in the balanced dataset.

The perhaps most striking difference when comparing these results to those of the balanced dataset in Figure 6.6, is the difference in the F_1-scores, which is significantly lower for all models. This can be attributed to the large decrease in precision for all models (following the increase in false positive predictions, indicated by the confusion matrices in Figure 6.7b). This behaviour is somewhat expected, as the models have been trained on a dataset where bankrupt samples are much more frequent than in the unbalanced case, making the models more biased towards predicting bankruptcies. As the other metrics considered are comparable to those of the balanced dataset, this emphasizes the importance of using a nuanced and representative set of metrics to evaluate the models, as they all summarize potentially different behaviours.

Model prediction probability distribution

Figure 6.9 shows the model prediction distributions of the full test set. Comparing to the U-shaped distributions of Figure 6.5, all of these prediction distributions have high numbers of near-0 predictions, with strictly decreasing numbers of higher interval predictions, and very few predictions in the highest intervals. This is indeed the desired behaviour, and confirms that our models are able to produce meaningful predictions even on the full set.

However, the mean of the predictions is still high relatively high - 0.1578 for logistic regression, comparing to only the 1.514% of positive samples. This indicates that there are some bias in the models, as one would expect unbiased models to have prediction means close to 0.01514.

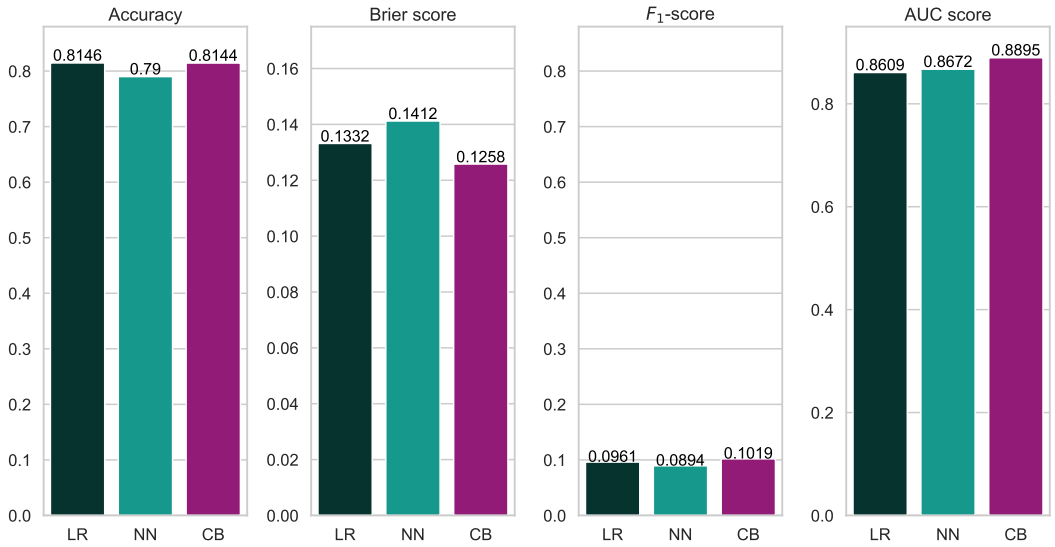


Figure 6.8: The model metric comparison on the full dataset. Most metrics give similar results to those obtained on the balanced dataset (Figure 6.6), except for in the F_1-score, which is the only metric that reflects the increases in false positive rates on the full test set. This emphasizes the need for a nuanced set of metrics in order to properly evaluate model behaviour.

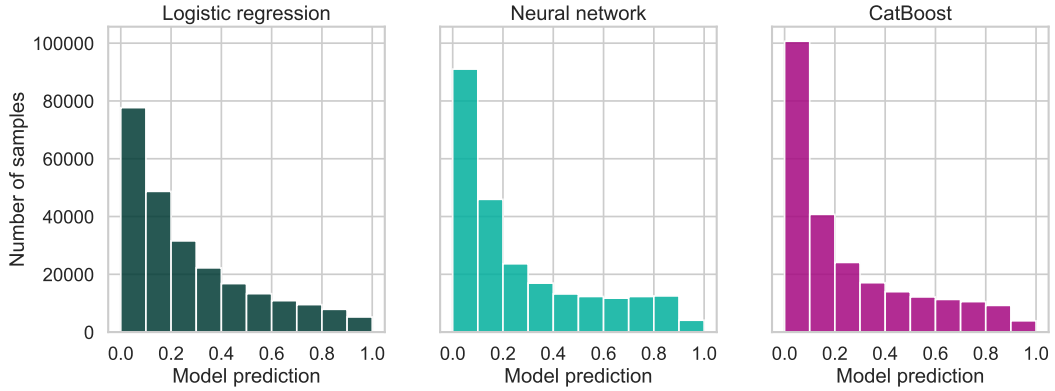


Figure 6.9: Full test set model prediction distributions. All models now produce very few predictions in the higher intervals, with the greatest bulk of predictions being close to 0, as is expected.

As mentioned before, this is the expected behaviour, although future work should seek to eliminate or minimize minimize this bias.

6.5 Static methods model interpretation

In this section, we will collect and compare our findings using SHAP to interpret the different models. While in no means an exhaustive analysis, we will demonstrate how SHAP values can be used to infer the learned behaviour of more complex models, specifically by analyzing the summary plots, explained in Section 4.2.7. The SHAP values are obtained using the implementations discussed in the respective model sections in Chapter 5.

The SHAP summary plot shows a vertical scatter plot with different features plotted horizontally, sorted by their total SHAP magnitude. The SHAP values are displayed vertically as dots (with similar SHAP values stacking horizontally, resulting in "thicker" scatter plots), and the

corresponding feature values for each of the dots are indicated by the color coding. Note that the SHAP values are shown before the final function transformation (sigmoid or logistic transformation, depending on the model), meaning negative values contribute towards less than model output 0.5, and positive towards greater than 0.5 model output.

We will show the summary plots for the 30 most important features trained on the first fold of the validation scheme, meaning they correspond to the SHAP feature subsets used throughout this chapter (the top 15 features for the SHAP-15 feature subsets). Also note that the $nace\_code$ variable is included as a target encoding.

As mentioned before, we will refrain from doing any specific feature or feature interaction analysis, as that would typically require more qualitative analysis of economical or other practical implications, leaving the economical analysis outside the scope of this thesis.

6.5.1 Logistic regression summary plot

Figure 6.10 shows the summary plot for the logistic regression model. We see the feature $log(age\,in\,years)$ being the most important one, with increasing company age resulting in lower bankruptcy probability estimation, and vice versa. This is perhaps not surprising, as we expect younger companies to be less stable and to be more likely to go bankrupt.

We also see from the $log(total\,assets)$ variable that companies that are large in terms of total assets are less likely to go bankrupt (this too, being expected). Also note that both of the dummy variables are present, and that the $nace\_code$ is 11th most important feature by the average SHAP magnitude. For the latter, the high importance combined fact that the model learns a strictly increasing relationship between the target encoding and the probability output, confirms that the target encoding is appropriate and holds significant predictive value.

An interesting example from the logistic regression summary plot is the leftmost point for the $Interest\,expenses\,/\,tot\,assets$, which is both an outlier in terms of SHAP value (being by far the most negative one), and does not seem to follow the trend of the rest of the samples for this feature, with the SHAP value strictly increasing with increasing feature values. This is most likely caused by feature interaction mechanisms, and would be an interesting sample to study in a practical context.

6.5.2 Neural network summary plot

Figure 6.11 shows the neural network SHAP summary plot. Comparing with the logistic regression model in Figure 6.10, we see a general accordance in terms of which features are considered most important. Notably, the $log(age\,in\,years)$ and $log(total\,assets)$ are both amongst the top features, with the same effects on model probability output. The most important feature found by the neural network, $public\,taxes\,payable\,/\,tot\,assets$, is the second most important in the logistic regression model. The $nace\_code$ is the 12th most important feature in the neural network, similar to the logistic regression model.

We also see the that the outlier found for the logistic regression model in $Interest\,expenses\,/\,tot\,assets$ is not present for the neural network, indicating that it could be a symptom of misfitting in the logistic regression model, or that the neural network model was unable to identify it as a reliable outlier. Due to the general restrictiveness of the logistic regression model, we deem the former more likely, as logistic regression is typically more susceptible to negative effects from outliers.

6.5.3 CatBoost summary plot

Figure 6.12 shows the CatBoost SHAP summary plot. Generally, this confirms the most important features found in Figures 6.10 and 6.11, with different permutations of $log(age\,in\,years)$, $log(total\,assets)$, $public\,taxes\,payable\,/\,tot\,assets$, and $accounts\,payable\,/\,sales$ being the most important features (with the same general learned relationships).

We also see that the CatBoost model finds the $nace\_code$ feature to be even more important than the other models, ranking it as the 6th most important feature. Interestingly, the learned

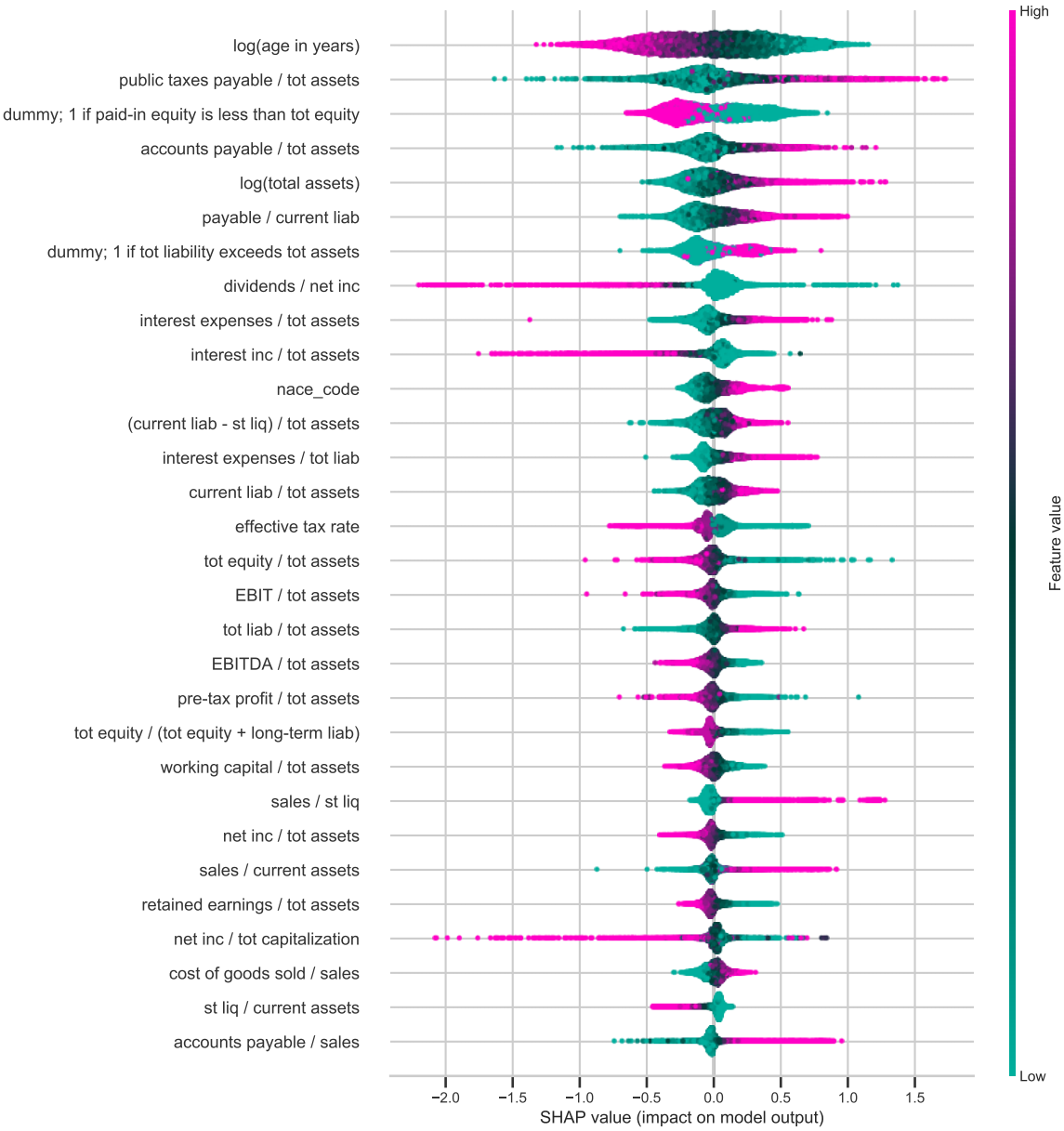


Figure 6.10: SHAP summary plot for the 30 most important features learned by the logistic regression model. The outlier for the $Interest\ expenses\ /\ tot\ assets$ feature is interesting, and could be subject for more qualitative analysis.

relationship between feature value and SHAP value is not as clear as for the logistic regression and neural network models. The fact that it is also more important for the CatBoost model shows that the CatBoost model has perhaps learned a more complex relationship including other feature interactions for this feature, which in turn has resulted in more predictive power.

Another interesting tendency in the CatBoost plot, is the apparent "grouping" of SHAP values for some features (such as for instance the right tail of $dividens\ /\ net\ inc$ or the left tail of $interest\ expenses\ /\ tot\ liab$, compared to the more smooth behaviours found in Figures 6.10 and 6.11 for the other models. This can perhaps be attributed to the tree structure of the CatBoost model, as output values are determined by (combinations of) disjoint regions. This results in less smooth prediction surfaces than for the logistic regression and neural network

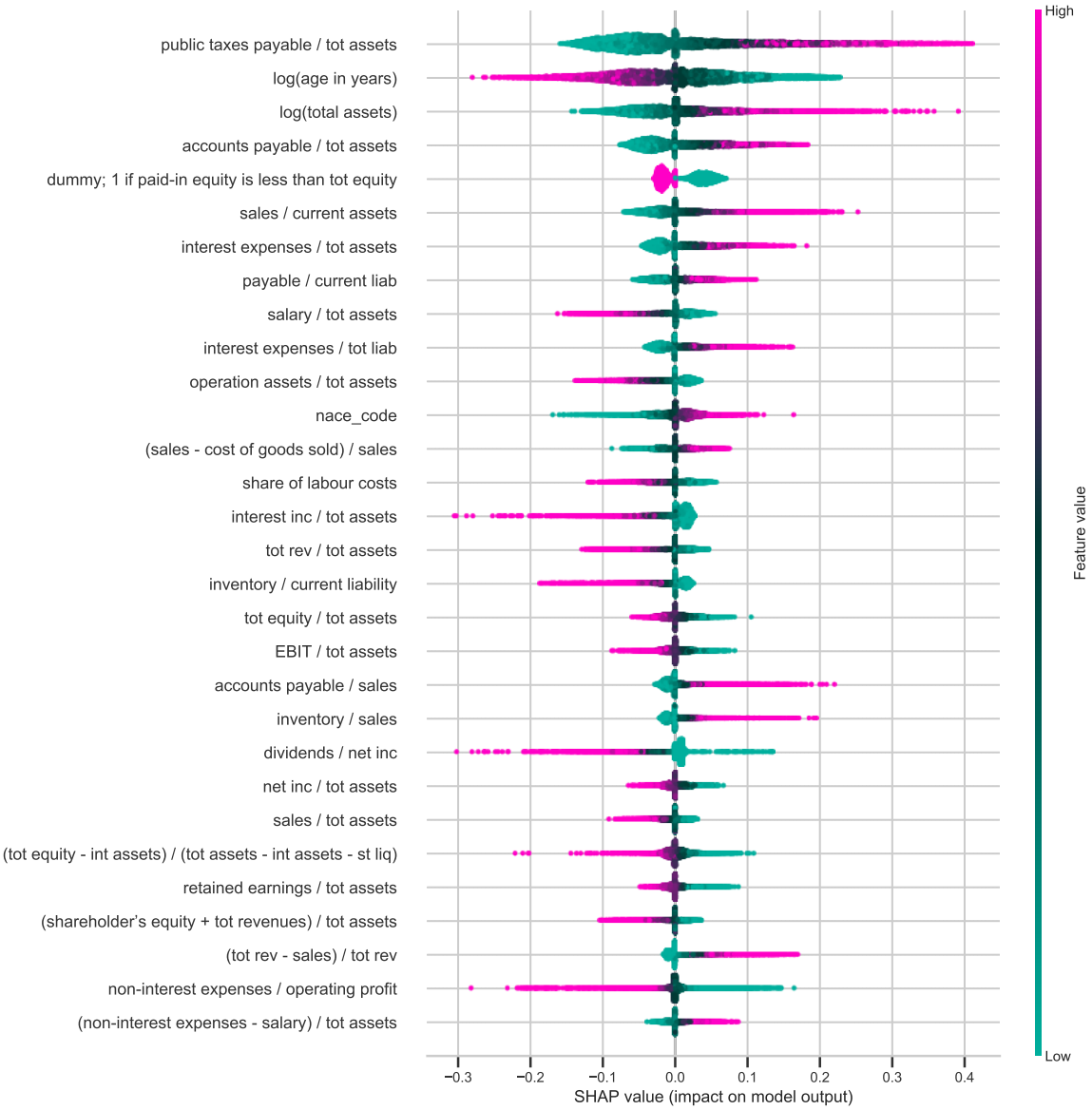


Figure 6.11: SHAP summary plot for the 30 most important features learned by the neural network model.

models, and corresponding "grouping" of SHAP values for the CatBoost model, should a lot of values fall in these specific regions.

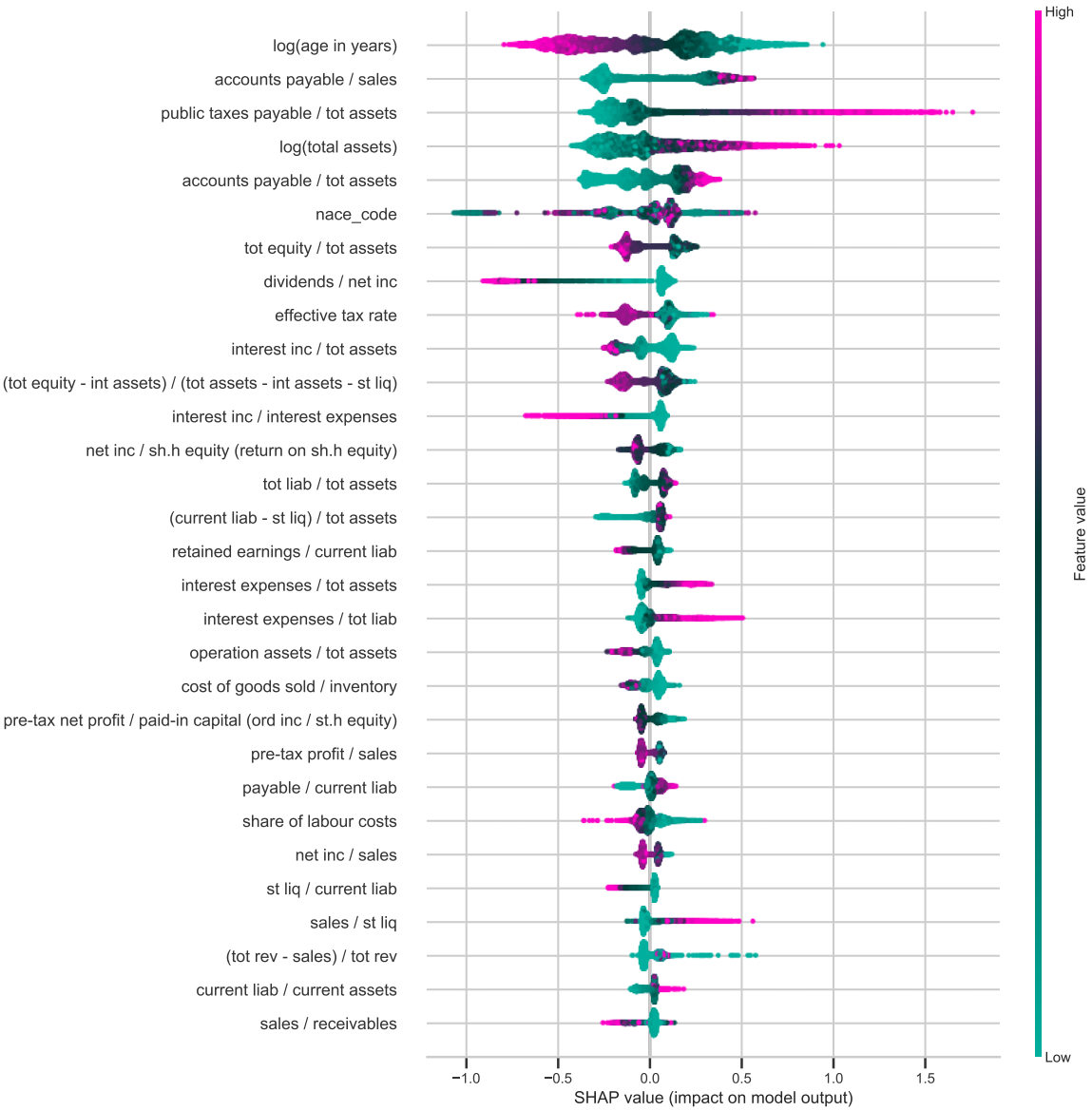


Figure 6.12: SHAP summary plot for the 30 most important features learned by the CatBoost model. Note the non-uniform relationships learned for the *nace_code* feature.

Chapter 7

Timeseries modelling experiments and results

As we have seen throughout Chapter 6, the previous year's accounting data holds significant predictive power for bankruptcy, yielding accurate results even for the simpler logistic regression models. However, one could assume that including multiple years of accounting data would provide the model with even more predictive information, revealing trends in the economical development of the company. Modelling bankruptcy prediction as a timeseries prediction problem then becomes a natural extension, using features from sequential financial statements to construct the timeseries.

In this chapter, we will introduce the application of modern timeseries methods to the bankruptcy prediction problem, namely the recurrent neural networks (RNNs) and long short term memory neural networks (LSTMs), both described in detail in Section 3.5, following a timeseries structuring of the problem, described in Section 5.7. As this space is significantly under-explored in the literature, this becomes a major part of the contribution of our works.

We will generally use the same experimental structure as in Chapter 5, and structure the analysis similarly to the static methods in Chapter 6. Rather than separating the analysis of the RNNs and LSTMs, however, we will consider both the RNN and LSTM networks collectively, due to their similarity in implementation and architecture. Indeed, the only difference in their implementations is the first layers in the networks.

In order to keep the analysis focused towards the contributions of the timeseries formulation of the problem, we will not consider different categorical encodings for the $nace\_code$ feature, rather using the general finding throughout Chapter 6 in that the target encoding performed the best or close to best for each of the considered methods (especially for the neural network, which is the most similar model). Another alteration to the experimental setup is that the models in this chapter are tuned and validated using training data from the rolling k-fold structure described in Section 5.4.1, rather than the fixed size fold structure used throughout Chapter 6. As described in Section 5.4.1, this allows us to investigate the relative increase in performance as the data availability increases.

We will also make comparisons with the static model results from Chapter 6 where relevant. Note, however, the general point that the timeseries formulation necessarily adds a significant complexity to the problem, both in terms of data requirements, model construction and other considerations (see for instance the additional considerations we made in Section 5.7). For deploying models in practical contexts, this may become problematic, and has to be taken into consideration.

7.1 RNN and LSTM validation set results results

In this section we will be reporting the timeseries results on the validation folds after training using the rolling k-fold scheme. As before, this means that the results will not be entirely

representative of expected out of sample performance, due to the earlystopping criterion using the validation set during training. However, the rolling k-fold scheme will allow for interesting analysis of the performance increase when more data is available. Keep also in mind the findings on the validation folds throughout Chapter 6, which generally showed the same trends of varying performance between folds, implying that some years might be more difficult to predict than others.

Analogously to Chapter 6, we will first fit LSTM and RNN models on the full feature set, before doing feature subset analysis on 15 and 30 features based on SHAP feature importance (the SHAP analysis will be performed in Section 7.3). We will leave the most detailed analysis (and static model comparisons) to the test set analysis in Section 7.2, focusing on the relative model performance and performance between folds (and feature subsets) in this section.

7.1.1 Average metrics across validation folds

Table 7.1 shows the performance for the LSTM and RNN models on the different feature subsets, computed as the average across the validation folds, excluding the first two. As these folds only have 1 and 2 years of training data, we expect the metrics for these folds to be particularly bad (see Figure 7.1). Excluding them allows for more informative and consistent comparison with the results of Chapter 6, as the amount of training data is more comparable.

	Accuracy score	Brier score	F_1-score	AUC score
LSTM full feature set	0.755545	0.167128	0.748564	0.837423
RNN full feature set	0.754898	0.167261	0.746886	0.836301
LSTM SHAP-15	0.757705	0.157506	0.756738	0.855408
LSTM SHAP-30	0.776405	0.151389	0.779623	0.862721
RNN SHAP-15	0.755708	0.158869	0.754714	0.852499
RNN SHAP-30	0.763348	0.157653	0.764549	0.854838

Table 7.1: Average validation fold results for the different timeseries models. These are computed as averages across all time folds, excluding the first two folds. We see that the LSTM models generally outperform the RNNs, especially for the SHAP feature subsets. Interestingly, the performance is significantly worse in the case of full feature subsets. This may be attributed to the increased model complexity, making more features more difficult to learn.

From Table 7.1, we see a clear trend of the LSTM outperforming the RNN, with better performance in all metrics for each respective feature subset. This implies that the added flexibility of the LSTM is indeed valuable, allowing it to better learn the time dependencies of the data. We also see the same trend as in Chapter 6, with the 30 feature subset models generally performing better than the 15 feature subset models.

An interesting observation is the large discrepancies between the full feature set models and the SHAP feature subset ones, with 3.02% and 2.22% higher AUC scores for the SHAP-30 LSTM and RNN models, respectively. Such large discrepancies were not seen in the static models in Chapter 6. This implies that the added complexity of the timeseries, combined with a large number of features, may make the models harder to train, as there are many more parameters to optimize. This emphasizes the importance of doing feature selection, especially as the models get more complex.

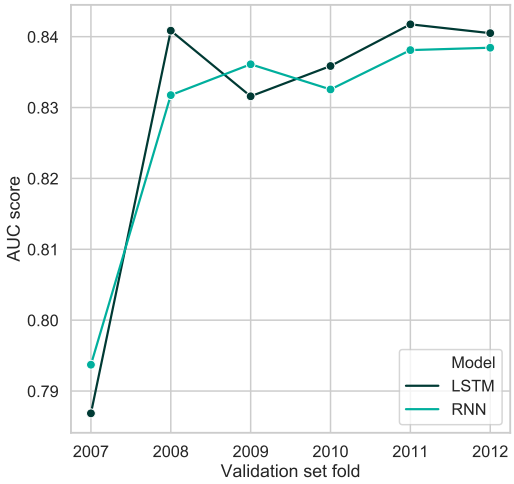
We also see that the relative difference between the LSTM and RNN models become greater for the feature subset models, especially for the SHAP-30 models. This may imply that the added advantages of the LSTM only becomes relevant when it is able to properly learn the data, as is the case when the feature subsets are smaller.

7.1.2 Behaviour across folds

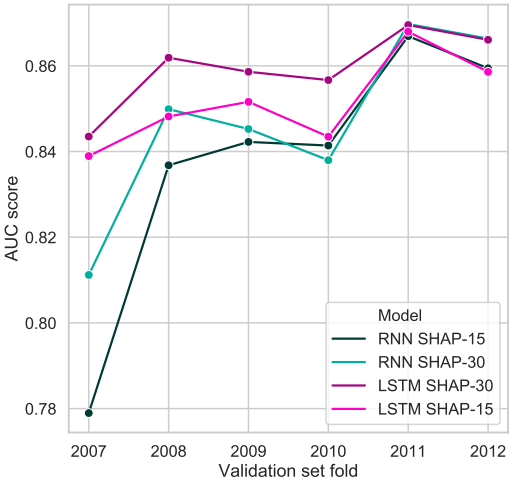
From Figure 7.1, we generally see a trend of slightly increasing AUC scores across the folds, as more data becomes available. This is especially apparent for the first fold, with only one

year of training data resulting in poor performance. Surprisingly, the second fold shows performance comparative to the other folds, especially for the full feature subset models. This may imply that only one additional year of timeseries data holds significant value, as the first fold will only have timeseries data of length 1.

Comparing the feature subset models in Figure 7.1b, the strengths of the LSTM models become more apparent. While the AUC for the last two folds is very similar for all models, the LSTM is more stable across the folds, especially so the SHAP-30 feature subset model. This implies that the LSTM model is more efficient in terms of available data, as well as being more robust to the variations in the data across the folds.



(7.1a) Logistic regression models with different categorical variable encoding. Target encoding greatly outperforms the others in all folds.



(7.1b) Logistic regression models with trained on different feature subsets. Interestingly, the SHAP methods increase their performance in the last fold.

Figure 7.1: AUC scores for each of the folds for each of the implemented logistic regression models. All models tend to follow the same patterns, and there is not any apparent decrease in variation in the feature subset models.

Comparing Figure 7.1 to the corresponding static models in Figures 6.1b, 6.3b, and 6.4b, we see a similar trend of decrease in the AUC score in the 2010 fold (except for the CatBoost model, interestingly). This could imply that this fold is particularly difficult to predict, and that the general trend of increasing performance with increasing data amount holds true.

7.2 Timeseries models test set results

In this section, we will present the results of the RNN and LSTM models on the test set, consisting of data from years 2013 and 2014. We will structure the analysis similarly as to that of Section 6.4 for the static models, reporting the results on both the balanced and the full (unbalanced) test sets. For the latter, we use the same timeseries length construction method described in Section 5.7.2. As in Chapter 6, the models will be trained on the data from 2006-2011, and use 2012 as the validation set for the earlystopping criterion.

Motivated by the findings in Section 7.1, we will only proceed with the SHAP-30 feature subsets for both the RNN and LSTM models, as these demonstrated the best performance. This is also consistent with the static method test set results in Section 6.4.

7.2.1 Balanced test set performance

As in Chapter 6, we will begin by reporting the results on the balanced test set. As well as comparing the average performance for each of the metrics, and the prediction probability distribution, we will also analyze the model behaviour across the different timeseries lengths, giving insights into the performance contribution of each additional year of accounting data.

To ease comparison between the static models, we will also include in our figures the corresponding metrics from the CatBoost model, as it was found to be the best performing of the static models. This allows us to more easily compare the relative increases (or decreases) in metrics which comes at the cost of the added complexity of the timeseries problem formulation.

Evaluation metrics performance

Figure 7.2 shows a comparison of the RNN and LSTM mode performances on our set of metrics on the balanced test set. We see the same general trend found in Section 7.1, with the LSTM generally outperforming the RNN in all metrics, further proving that the added flexibility of the model holds value in our bankruptcy prediction context.

Comparing these results with the static methods, we see that the LSTM ranks second in terms of AUC (and indeed most other metrics), behind the CatBoost model. Recall that the CatBoost significantly outperformed the static neural network, meaning that the timeseries methods does, however, show a significant increase in all metrics over the static neural network. This shows that the added timeseries information holds value in the context of neural network models.

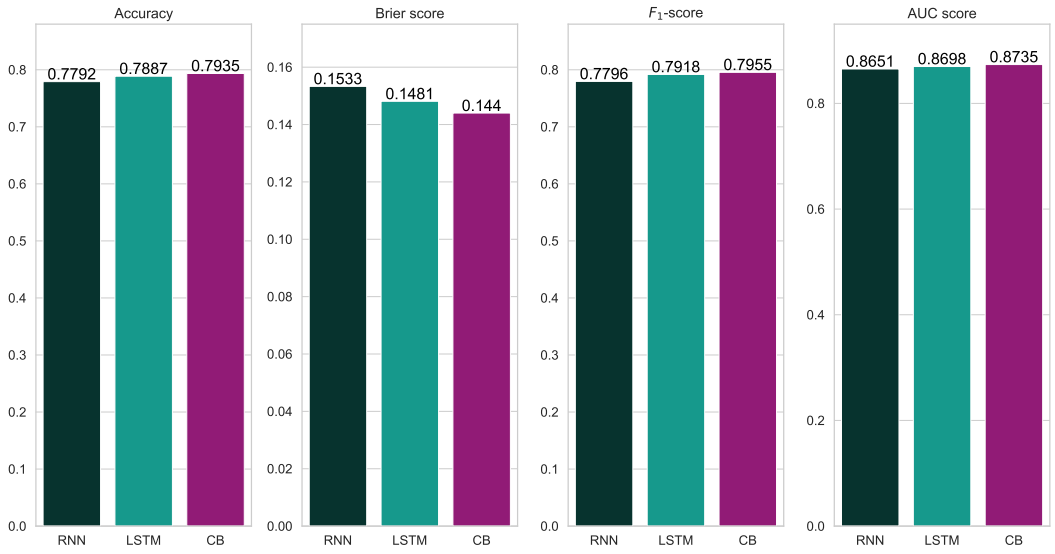


Figure 7.2: A comparison of the metrics of the RNN and LSTM models on the balanced dataset, along with the CatBoost (CB) model. While the performance is similar, CatBoost performs better on all considered metrics.

Results on different timeseries length

Another interesting topic of analysis is the timeseries model performance for samples grouped by their respective timeseries lengths. Grouping the results by the length of the timeseries allows us to learn more about the learned model behaviour, and how important the successive timeseries length are for predictive performance.

Figure 7.3 shows a breakdown of the AUC scores for each of the timeseries lengths, for the RNN and LSTM models. We see a clear general trend of increasing performance as more timeseries steps are made available, with the largest relative increase being from timeseries

length 1 to 2 (1.877% increase for the RNN and 1.537% for the LSTM). This confirms our suspicion of gradual increase in prediction performance as more time information becomes available.

Furthermore, we see that the relative differences between RNN and LSTM become more apparent as the timeseries length increases, with LSTM clearly outperforming the RNN at timeseries length 4. This shows the strength of the LSTM network, in terms of handling longer timeseries lengths (even a relatively short length of 4).

As could be expected, the AUC scores for both models at length 1 is similar to that of the static neural network (see Figure 6.6). Although the CatBoost model outperforms the timeseries methods in terms of average AUC (recall that length 1 timeseries samples dominate our balanced test set, due to our timeseries length selection method described in Section 5.7.2, see Table 5.2), the LSTM does produce better AUC scores at timeseries lengths 3 and 4, implying that these frameworks hold significant predictive value when allowed to process longer timeseries. If this inherent sampling bias was not present, timeseries lengths of 3 and 4 would be relatively more represented, making the timeseries methods outperform the static CatBoost models. This is an important point, but not one we can further address given the restrictions imposed by the dataset balancing.

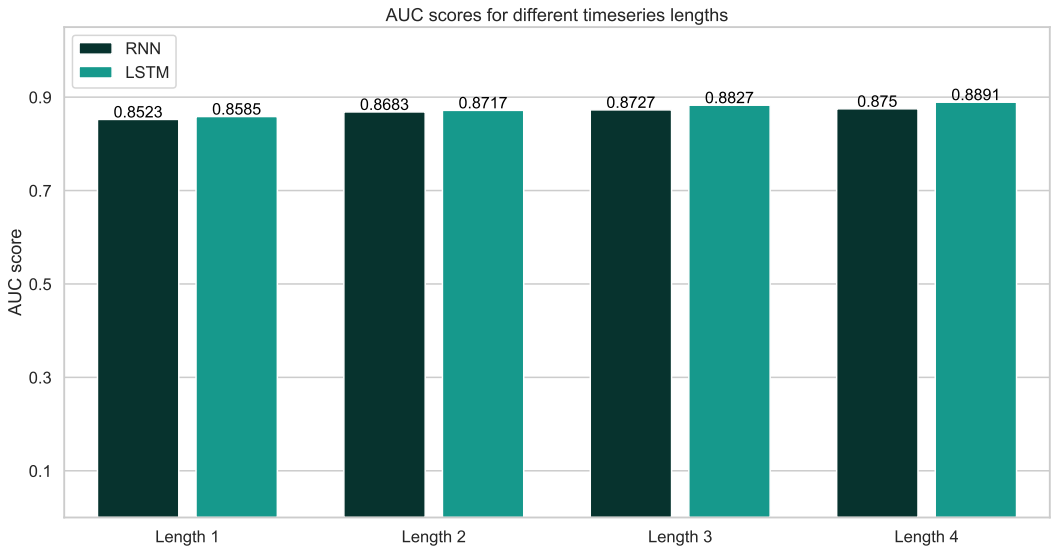


Figure 7.3: The average AUC scores for the RNN and LSTM models, grouped by the length of the timeseries samples. As expected, we see a general trend of increasing AUC scores as more timeseries information becomes available, with the greatest increase coming from length 1 to length 2.

Model prediction distribution

Figure 7.4 shows the model prediction distribution for each of the models. As for the models in Chapter 6, we see the same general U-shape, with the same reluctance towards the uppermost prediction interval as with the static neural network and CatBoost models. This behaviour was discussed in Section 6.4.1, and can be attributed to the same reasons discussed there.

An interesting difference in these distributions compared to those of Figure 6.5, is that the timeseries distributions are generally "flatter", in the sense that they seem to have more predictions around 0.5, and smaller spikes near the 0 prediction interval. One would perhaps expect this to be reflected in a lower Brier score, but comparing Figures 7.2 and 6.6 shows no clear difference. This could imply that while the timeseries predictions are generally less confident, they are often more correct in the predictions that are indeed confident.

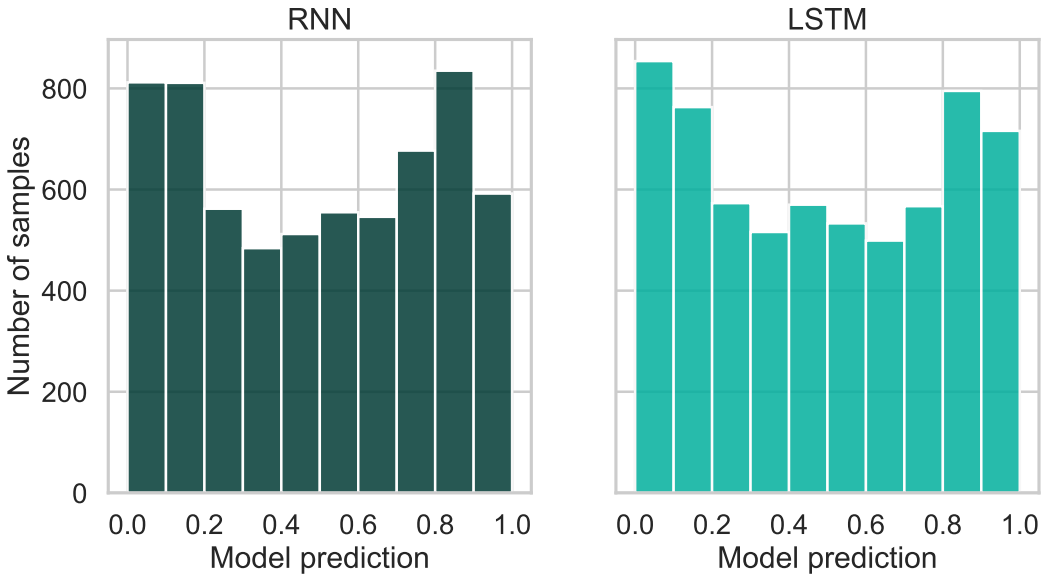


Figure 7.4: The prediction probability distributions of each of the RNN and LSTM models, with each bar representing an interval of length 0.1. Both models emit the general U-shape (with reluctance towards the highest interval) as for the static models in Figure 6.5, but the RNN and LSTM distributions generally has a flatter shape.

7.2.2 Full test set performance

In this section, we will analyze the test set performance on the full, unbalanced dataset. As before, we will follow the same overall analysis as for the static models in Section 6.4, with some additional points relevant to the timeseries problem.

Evaluation metrics performance

Figure 7.5 shows the full test set performance of both of the models, on all considered metrics. We again see LSTM outperforming the RNN in all metrics, except for accuracy, interestingly. As this is the first case of the RNN outperforming the LSTM in either of the considered metrics, this can perhaps be attributed to the weakness of accuracy as a metric, rather than the RNN model being superior (see discussion in Section 4.1.2).

Comparing these metrics with the metrics of the static models in Figure 6.8, we see generally comparative performance between the LSTM and CatBoost models, with both the LSTM and RNN models outperforming the other static models. The CatBoost model has slightly better AUC score (0.8895 vs. 0.8836) and Brier score (0.1258 vs. 0.1261) than the LSTM, while the LSTM slightly outperforms CatBoost in terms of accuracy (0.8144 vs 0.8230) and F_1-score (0.1019 vs 0.1030). While these results are indecisive, the added complexity of the timeseries methods could make one prefer the static CatBoost models.

Results on different timeseries lengths

Figure 7.6 shows the AUC score breakdown per timeseries length for each of the models, which shows a similar trend to that of Figure 7.3; the LSTM generally outperforms the RNN, with strictly increasing AUC scores as more timesteps become available. We again see the largest relative increase being from timeseries length 1 to 2, and see very good AUC scores for timeseries of lengths 3 and 4, especially for the LSTM models. This confirms the previous findings, in that these models perform best for longer timeseries.

The point of timeseries length bias discussed in Section 7.2.2 also still holds true, in that the timeseries lengths are unevenly represented, creating a stronger weight towards the (worse performing) 1 and 2 length timeseries. Again, we will not be addressing this any further,

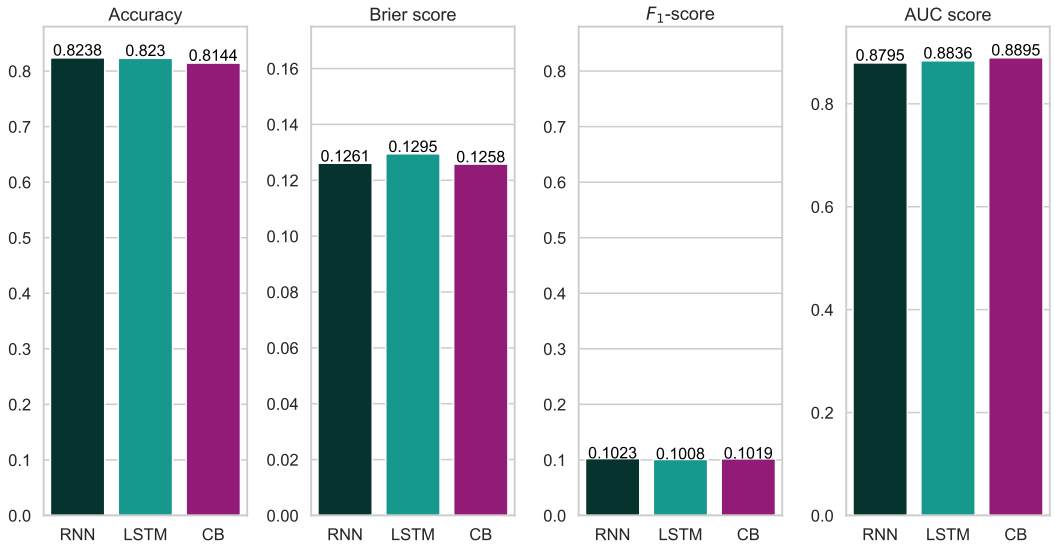


Figure 7.5: A comparison of the metrics of each of models on the full (unbalanced) dataset. In this case, the results are indecisive, with CatBoost and LSTM outperforming each other in different metrics.

simply noting its effect on the average metrics when comparing the timeseries models to the static methods.

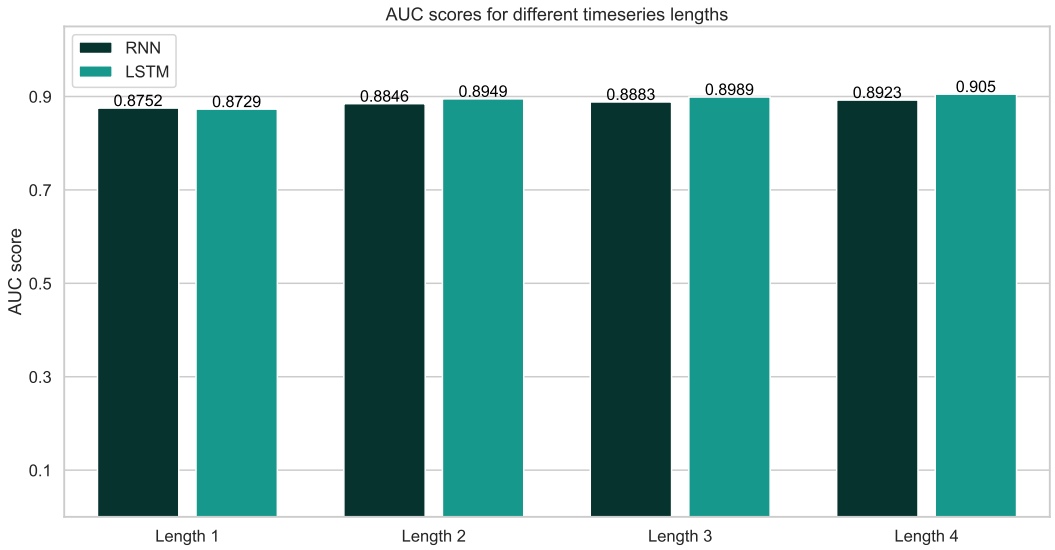


Figure 7.6: The average AUC scores for the RNN and LSTM models on the full dataset, grouped by the length of the timeseries samples. We see the same trend of increasing AUC as seen in Figure 7.3

Model prediction probability distribution

Figure 7.7 shows the prediction probability distributions on the full test set. Similarly as in Figure 6.9 for the static methods, the timeseries methods show heavy tails in the lowest intervals, with sequentially decreasing number of predictions in the increasing intervals - as is expected, given the low number of bankrupt samples in this set.

Similarly, the mean of the predictions is still relatively high - 0.1541 for the LSTM and 0.1618 for the RNN - which shows a clear bias towards predicting bankruptcy. As before, this is expected, and follows from the bias imposed in the dataset balancing process.

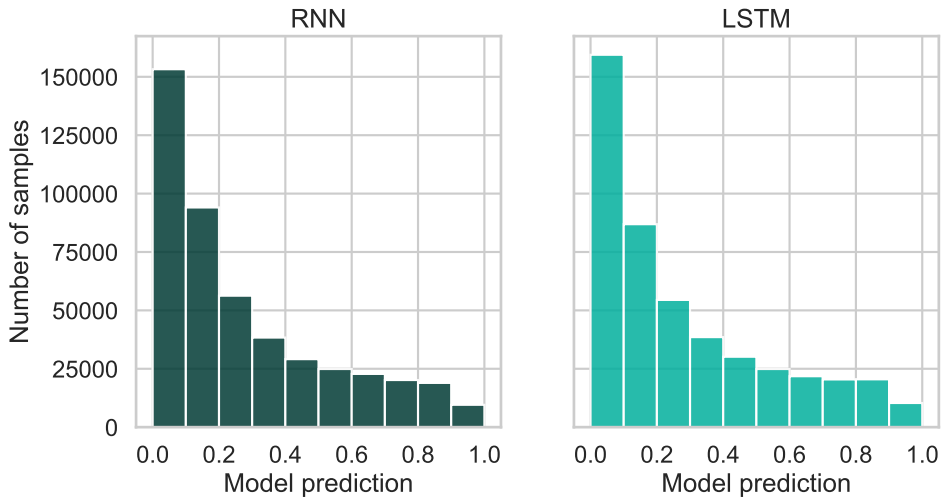


Figure 7.7: The prediction probability distributions of the RNN and LSTM of the models on the full test set, with each bar representing an interval of length 0.1. text.

7.3 Timeseries model interpretation

In this section, we will perform a similar global SHAP value interpretation of the RNN and LSTM models, as was done in Section 6.5. Because of the temporal nature of the timeseries models, however, the SHAP summary plots showed in this section cannot be analogously produced for these models. We will instead show horizontal bar plots of the mean SHAP magnitude for each of the features, for each of the timesteps. We would then expect a lower SHAP magnitude for each consecutive timestep, as the most recent information is, intuitively, the most relevant. While not as information dense as the summary plots, they still allow us to verify the models' learned behaviour and show the selected feature subsets.

As before, we will display the 30 most important features. In this case, they are selected based on the total SHAP magnitude for all timesteps for the feature. Note also that these SHAP values are computed on the fourth fold of the rolling k-fold scheme, as this is the first fold that contains data for all timeseries lengths.

From Figures 7.8 and 7.9 for the RNN and LSTM models, respectively, we generally see the expected behaviour of lower SHAP magnitude for consecutive timesteps. The differences are also generally big, with the first timestep having often more than twice the average SHAP magnitude as the second timestep. The consecutive timesteps are seemingly even smaller fractions of the previous timestep. This confirms our general findings in that most of the predictive power comes from the first timestep (indeed, this is confirmed by the comparative results of the static methods in Chapter 6), while consecutive timesteps offers less and less predictive performance increases.

While this is the general trend, there are some interesting deviations. For instance for the LSTM model in Figure 7.9, the feature $current\ assets/total\ liab$ (the fourth most important feature) shows a significantly higher average magnitude for the second timestep than for the first. This is indeed counter intuitive, and could point to the model learning a misrepresentation, perhaps due to overfitting. Indeed, the feature is not even amongst the top 30 features for the RNN model (Figure 7.8), providing further evidence towards this being a misrepresentation.

Comparing the learned relationships of the LSTM and RNN models with each other, we see that they generally conform to similar features, and also to those found for the static models.

The novel $nace\_code$ feature is still amongst the top features for both of the models, and features like $log(age\ in\ years)$ and $log(total\ assets)$ are also represented.

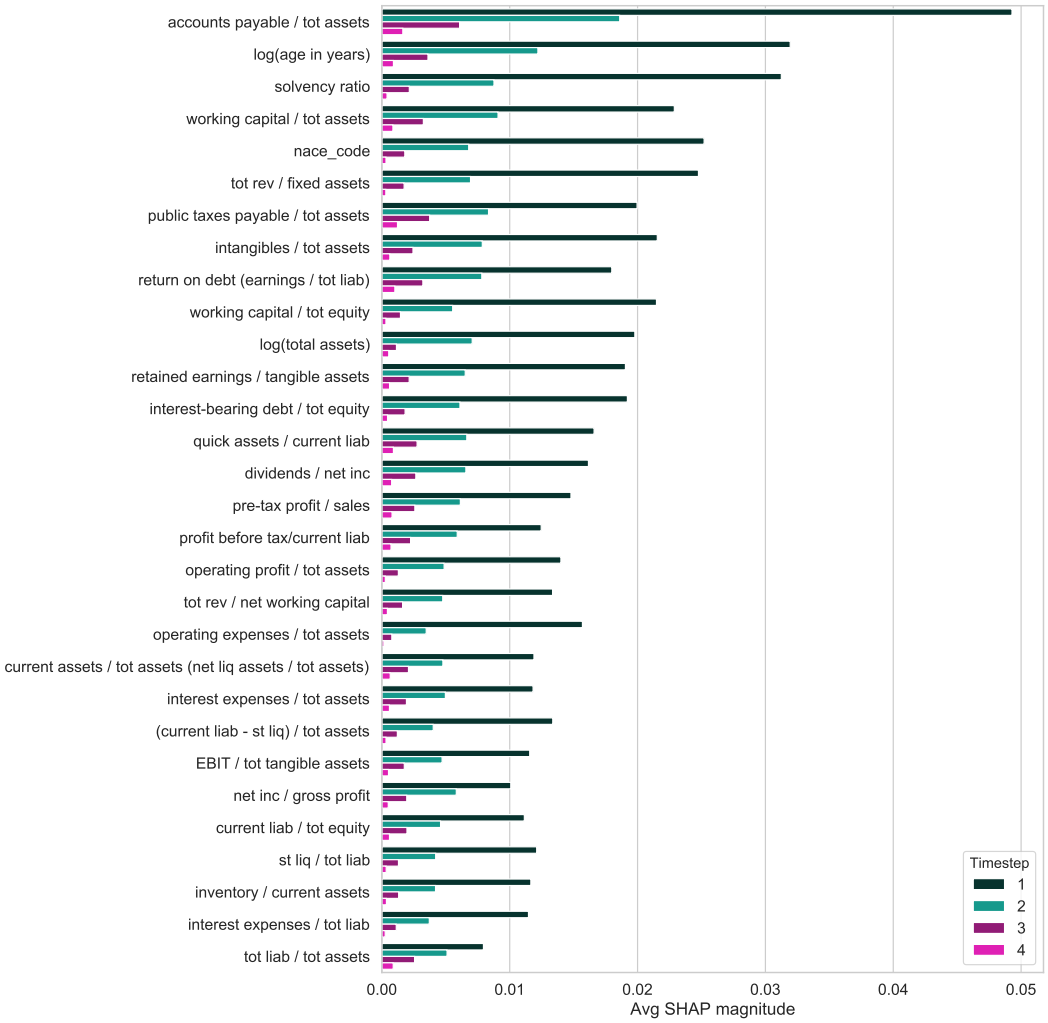


Figure 7.8: Average SHAP magnitudes for the 30 most important features learned by the RNN model.

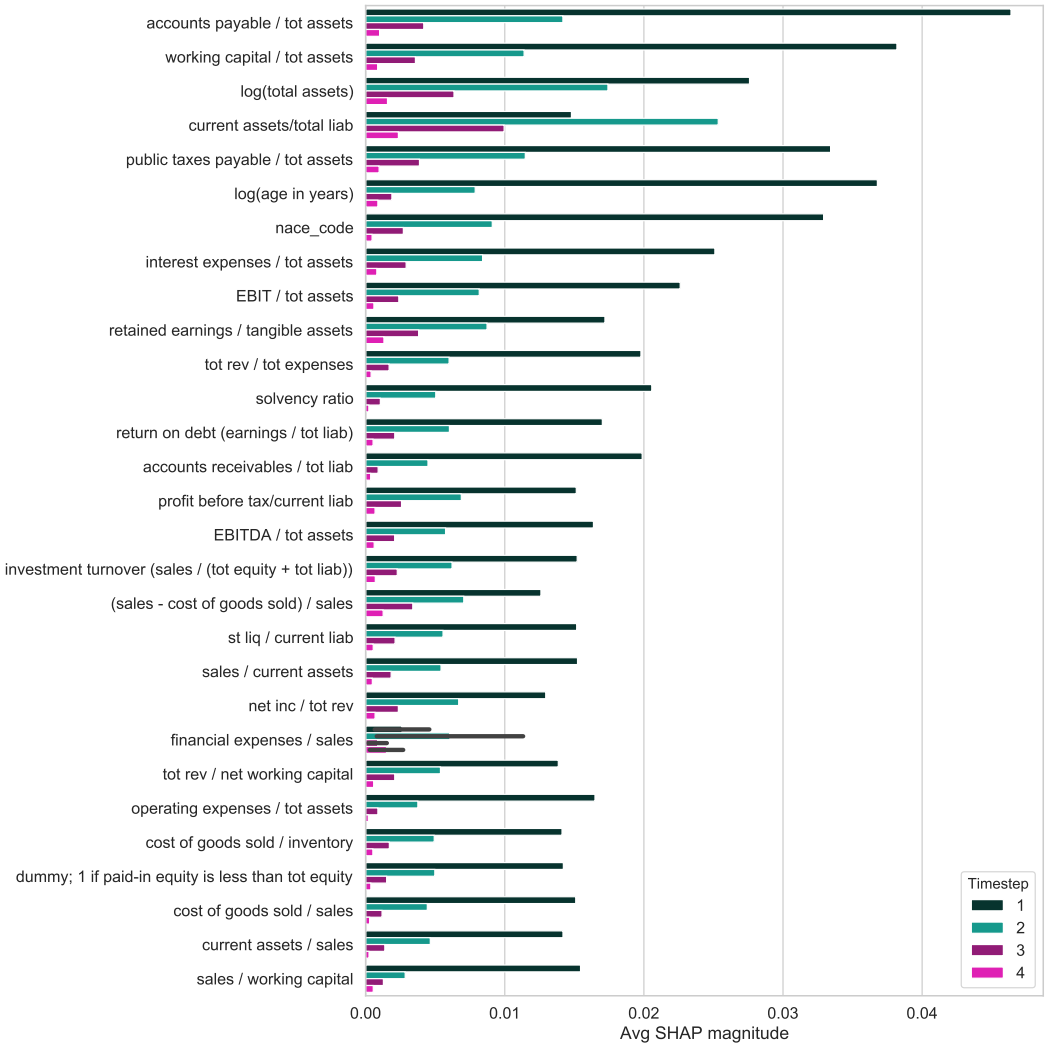


Figure 7.9: Average SHAP magnitudes for the 30 most important features learned by the LSTM model. Interestingly, the second timestep of the feature $current\ assets/total\ liab$ has a greater average magnitude than the first.

Chapter 8

Conclusion and further work

8.1 Conclusion

The goal of this thesis was to introduce modern statistical prediction models to the space of bankruptcy prediction, in order to drive valuable performance increases for a representative set of metrics, while also maintaining interpretability, which is often required for practical deployment of such models. Specifically, we targeted the economically significant SMEs, which has historically received less attention in the literature.

This was carried out by leveraging both an extensive dataset comprised of yearly financial statements from more than 175000 Norwegian SMEs ranging from the years $2006 - 2014$ (using as features different combinations and ratios of the entries), and modern machine learning prediction models, such as gradient boosting trees and LSTM neural networks. In order to achieve a realistic and fair picture of the expected model performances, our deployed experiments adheres to the highest possible machine learning standards (using temporal train, validation, and test set data splitting), analyzing the performance on a set of four relevant metrics meant to capture different properties of the models: Accuracy, Brier score, F_1-score, and AUC score, with our main focus being on the AUC score.

As the resulting model metrics are highly dependent on the specific dataset used, we reimplemented two models that are popular in the bankruptcy prediction literature, namely logistic regression and fully connected neural networks, to produce benchmark metrics. We then implemented the CatBoost gradient boosting tree algorithm using the `CatBoost` package (as well as RNNs and LSTMs, for performing timeseries bankruptcy prediction).

We also introduce the usage of a novel categorical feature, and consequent ways of integrating such categorical features into the aforementioned models. Our feature uses the NACE system to provide information about a company's industry, while ensuring that each NACE category had a sufficient number of samples (at least 500) by utilizing the hierarchical nature of the NACE system. We examined one-hot encoding, target encoding, and for neural networks, categorical embeddings, for integrating the categorical feature into the model frameworks. Of these, target encoding was found to produce the most reliable results, offering consistent relative performance increases (average validation set AUC increases ranging from $0.497 - 1.22\%$ compared to no categorical feature), while maintaining similar temporal stability (from visual inspection of the performance on the temporal validation folds).

For the static (non-time dependent) models, the CatBoost model was found to convincingly outperform both the logistic regression and neural network benchmarks, both on the balanced (i.e. evenly distributed target variables) and full (unbalanced) datasets, achieving AUC scores of 0.8735 and 0.8895, respectively. For the former, the CatBoost models produced AUC scores 2.571% and 3.532% greater than the neural network and logistic regression benchmark models, respectively. It also outperformed the benchmark models in the other considered

metrics, demonstrating the CatBoost framework's potential in the context of bankruptcy prediction.

In all cases, we found that using feature subsets of 30 features (selected by average SHAP magnitude) gave comparable or improved results to the full feature models, motivating the use for more compact and thus often more practical models.

By analyzing the model prediction probability distributions, an interesting trend was revealed: Both the neural network and CatBoost models seemed resistant towards predicting bankruptcy probabilities in the highest probability range $(0.9 - 1.0)$, indicating that confidently determining bankruptcies are comparatively harder than predicting non-bankruptcies, perhaps attributed to the practical complications involved in the definition of a binary bankruptcy target variable. The fact that logistic regression did not learn this relationship, can probably be attributed to the model's relative simplicity.

We then proposed a methodology for structuring bankruptcy prediction as a timeseries problem, by organizing subsequent financial ratio features into timeseries containing $1 - 4$ years of such data. Given the evident data restrictions, as well as potential practical considerations, we limited the timeseries to a maximum length of 4. We then implemented and analyzed a vanilla recurrent neural network (RNN), and the more advanced extension, the long short term memory network (LSTM).

The average validation results for the timeseries models proved that the added flexibility of the LSTM network made it consistently outperform the simpler RNN. Interestingly, we saw significantly worse performance for timeseries models trained on the full feature set, with the feature subset models resulting in performance more comparable to that of the static CatBoost model. This highlighted the difficulty in learning the more complex time dependent relationships.

Further analysis revealed major performance increases for the longer timeseries, with AUC scores similar to the static neural network for the timeseries with only 1 year of available data, and much stronger AUC scores for timeseries with 3 and 4 years of data. For the LSTM, AUC scores for length 3 and 4 timeseries reached 0.8827 and 0.8891 on the balanced test set. As the overall performance of the static CatBoost model and timeseries LSTM network was comparable, one would perhaps be inclined to prefer the static method, as the timeseries formulation involves significant added complexity. The increased performance for the longer timeseries may, however, be desirable, and should therefore be at least considered.

Conforming to the standard practice in the literature, the above results were produced on a dataset with an equal number of bankrupt and non-bankrupt companies. While not the strict focus of the thesis, we also demonstrated the model's performance on the unbalanced test dataset. While these experiments showed that the models had indeed learned strong predictive power even on the unbalanced set, they also revealed significant biases in the model outputs reflected in much lower F_1-scores, motivating further research into the handling of the unbalanced data problem.

Finally, we introduced a theoretically consistent framework for model interpretation, namely the SHAP framework, which we used to demonstrate how to analyze the learned behaviour of each of the models. This is a valuable tool for any practitioner looking to harness the increased performance demonstrated by the more complex models, while still maintaining the often important interpretability.

8.2 Further work

Throughout this thesis, we have made sure to highlight any problems, approximations or other considerations that may weaken the significance or applicability of our analysis. These are what we would consider the most relevant topics for further work within the space of bankruptcy prediction, or even machine learning modelling in general. We will summarize them in this section, as well as point to some other ideas for further work in the topic.

The perhaps greatest barrier for practitioners wanting to use the models analyzed in this thesis in practice, is concerned with the dataset balancing process used before model training

throughout. While our test set analysis showed some degree of generalization ability, the dataset sampling induced clear bias, which has to be addressed before deploying the models in practical contexts. While an active area of research in machine learning in general, our initial experiments pointed towards combinations of sample weighting and fractional dataset balancing as promising approaches. Other options include synthetic sampling techniques or even embedding domain specific knowledge or costs directly into the loss functions. Proper handling of this issue would provide very valuable tools for any practitioners looking to use these models in practice.

Having demonstrated the potential in a timeseries formulation of the bankruptcy prediction problem, we would also hope to see further research for this application. If datasets spanning greater periods become available, looking into the effects of longer timeseries could be very relevant, as well as exploring different timeseries model frameworks or variations. One could also think that the inclusion of external, time dependent data could be relevant, as the timeseries formulation allows for including potentially valuable macro economical developments over time, as additional features in the data.

Another potential for improvement lies in further tuning and optimization of the specific model configurations deployed. Our experiments were in no means exhaustive, and it is safe to assume that there is potential for at least marginal performance improvements in either hyperparameter optimization, different neural network architectures, or more optimal feature subsets.

Finally, we hope that our demonstration of the application of the SHAP interpretability framework to bankruptcy prediction can help motivate more analysis and applications in the field, and perhaps in finance in general. It is our hope that combining this framework with more qualitative economical analysis can pave the way for many new discoveries.

Bibliography

Agarwal, V. and Taffler, R. (2008). Comparing the performance of market-based and accounting-based bankruptcy prediction models. *Journal of Banking & Finance*, 32(8):1541–1551.

Alaka, H., Oyedele, L., Owolabi, H., Oyedele, A., Akinadé, O., Bilal, M., and Ajayi, S. (2017). Critical factors for insolvency prediction: towards a theoretical model for the construction industry. *International Journal of Construction Management*, 17:1–25.

Altman, E. I. (1968). Financial ratios, discriminant analysis and the prediction of corporate bankruptcy. *The Journal of Finance*, 23(4):589–609.

Anani, W. and Samarabandu, J. (2018). Comparison of recurrent neural network algorithms for intrusion detection based on predicting packet sequences. In *2018 IEEE Canadian Conference on Electrical Computer Engineering (CCECE)*, pages 1–4.

Arora, N. and Saini, J. R. (2013). Time series model for bankruptcy prediction via adaptive neuro-fuzzy inference system. *International Journal of Hybrid Information Technology*, 6:51–63.

Balcaen, S., O. H. (2006). 35 years of studies on business failure: an overview of the classic statistical methodologies and their related problems. *British Accounting Review*, 38.

Bank of International Settlements (2017). Basel III: Finalising post-crisis reforms. Report.

Beaver, W. (1966). Financial Ratios As Predictors Of Failure. *Journal of Accounting Research*, 4:71–111.

Bellovary, J., Giacomino, D., and Akers, M. (2006). A review of bankruptcy prediction studies: 1930 to present. *Accounting Faculty Research and Publications*, 33.

Bernhardsen, E. and Larsen, K. (2007). Modelling credit risk in the enterprise sector – further development of the sebra model. *Economic Bulletin*, 2(8).

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA.

Brier, G. W. (1950). Verification of Forecasts Expressed in Terms of Probability. *Monthly Weather Review*, 78(1):1.

Chandrashekar, G. and Sahin, F. (2014). A survey on feature selection methods. *Comput. Electr. Eng.*, 40(1):16–28.

Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794, New York, NY, USA. Association for Computing Machinery.

Chollet, F. et al. (2015). Keras. `https://keras.io`.

Cortes, C. and Mohri, M. (2003). Auc optimization vs. error rate minimization. In *Proceedings of the 16th International Conference on Neural Information Processing Systems*, NIPS'03, page 313–320, Cambridge, MA, USA. MIT Press.

Council of European Union (2006). Nace rev.2 statistical classification of economic activities in the european community.

Council of European Union (2014). Council regulation (EU) no 269/2014. http://eur-lex.europa.eu/legal-content/EN/TXT/?qid=1416170084502&uri=CELEX:32014R0269.

Fernández, A., Garcia, S., Herrera, F., and Chawla, N. (2018). Smote for learning from imbalanced data: Progress and challenges, marking the 15-year anniversary. *Journal of Artificial Intelligence Research*, 61:863–905.

Filipe, S. F., Grammatikos, T., and Michala, D. (2016). Forecasting distress in european sme portfolios. *Journal of Banking Finance*, 64:112 – 135.

FitzPatrick, P. J. (1932). A comparison of the ratios of successful industrial enterprises with those of failed companies. *The Certified Public Accountant*, pages 598–605.

Friedman, J. H. (2000). Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232.

Geisser, S. (1975). The predictive sample reuse method with applications. *Journal of the American Statistical Association*, 70(350):320–328.

Gilpin, L. H., Bau, D., Yuan, B. Z., Bajwa, A., Specter, M., and Kagal, L. (2018). Explaining explanations: An overview of interpretability of machine learning. In *2018 IEEE 5th International Conference on data science and advanced analytics (DSAA)*, pages 80–89. IEEE.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics*.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Guo, C. and Berkhahn, F. (2016). Entity embeddings of categorical variables.

Gupta, J., Barzotto, M., and Khorasgani, A. (2018). Does size matter in predicting smes failure? *International Journal of Finance Economics*, 23.

Haibo He, Yang Bai, Garcia, E. A., and Shutao Li (2008). Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pages 1322–1328.

Härdle, W., Lee, Y.-J., Schäfer, D., and Yeh, Y.-R. (2009). Variable selection and oversampling in the use of smooth support vector machines for predicting the default risk of companies. *Journal of Forecasting*, 28(6):512–534.

Hastie, T., Tibshirani, R., and Friedman, J. (2001). *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA.

Hernandez Tinoco, M. and Wilson, N. (2013). Financial distress and bankruptcy prediction among listed companies using accounting, market and macroeconomic variables. *International Review of Financial Analysis*, 30(C):394–419.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Hoffer, E., Hubara, I., and Soudry, D. (2017). Train longer, generalize better: Closing the generalization gap in large batch training of neural networks. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 1729–1739, Red Hook, NY, USA. Curran Associates Inc.

Härdle, W., Lee, Y.-J., Schäfer, D., and Yeh, Y.-R. (2009). Variable selection and oversampling in the use of smooth support vector machines for predicting the default risk of companies. *Journal of Forecasting*, 28(6):512–534.

John, G. H., Kohavi, R., and Pfleger, K. (1994). Irrelevant features and the subset selection problem. In *Proceedings of the Eleventh International Conference on International Conference on Machine Learning*, ICML'94, page 121–129, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Johnson, J. and Khoshgoftaar, T. (2019). Survey on deep learning with class imbalance. *Journal of Big Data*, 6:27.

Jones, S., Johnstone, D., and Wilson, R. (2017). Predicting corporate bankruptcy: An evaluation of alternative statistical frameworks. *Journal of Business Finance & Accounting*, 44(1-2):3–34.

Kahya, E. and Theodossiou, P. (1999). Predicting corporate financial distress: A time-series cusum methodology. *Review of Quantitative Finance and Accounting*, 13(4):323–345.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 3146–3154. Curran Associates, Inc.

Kohavi, R. and John, G. H. (1997). Wrappers for feature subset selection. *Artif. Intell.*, 97(1–2):273–324.

Kohavi, R. and Li, C.-H. (1995). Oblivious decision trees graphs and top down pruning. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'95, page 1071–1077, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Konkursrådet (2020). Innføring i konkurs.

Kotsiantis, S., Kanellopoulos, D., and Pintelas, P. (2005). Handling imbalanced datasets: A review. *GESTS International Transactions on Computer Science and Engineering*, 30:25–36.

Krawczyk, B. (2016). Learning from imbalanced data: Open challenges and future directions. *Progress in Artificial Intelligence*, 5.

Kumar, P. and Ravi, V. (2007). Bankruptcy prediction in banks and firms via statistical and intelligent techniques - a review. *European Journal of Operational Research*, 180:1–28.

Lundberg, S. M., Erion, G., Chen, H., DeGrave, A., Prutkin, J. M., Nair, B., Katz, R., Himmelfarb, J., Bansal, N., and Lee, S.-I. (2020). From local explanations to global understanding with explainable ai for trees. *Nature Machine Intelligence*, 2(1):2522–5839.

Lundberg, S. M. and Lee, S.-I. (2017). A unified approach to interpreting model predictions. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc.

Maas, A. L., Hannun, A. Y., and Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*.

Mansi, S., Maxwell, W., and Zhang, A. (2010). Bankruptcy prediction models and the cost of debt. *Journal of Fixed Income*, 21.

Martin, D. (1977). Early warning of bank failure: A logit regression approach. *Journal of Banking Finance*, 1(3):249 – 276.

Micci-Barreca, D. (2001). A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems. *SIGKDD Explorations*, 3:27–32.

Neumark, D., Wall, B., and Zhang, J. (2008). Do small businesses create more jobs? new evidence from the national establishment time series. Working Paper 13818, National Bureau of Economic Research.

Ohlson, J. A. (1980). Financial ratios and the probabilistic prediction of bankruptcy. *Journal of Accounting Research*, 18(1):109–131.

Papadopoulos, G., Rikama, S., Alajääskö, P., Salah-Eddine, Z., Airaksinen, A., and Luomaranta, H. (2015). Statistics on small and medium-sized enterprises.

Pasini, A. (2015). Artificial neural networks for small dataset analysis. *Journal of thoracic disease*, 7:953–60.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., and Gulin, A. (2018). Catboost: Unbiased boosting with categorical features. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 6639–6649, Red Hook, NY, USA. Curran Associates Inc.

Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). "why should i trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 1135–1144, New York, NY, USA. Association for Computing Machinery.

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

Shapley, L. S. (1951). Notes on the n-person game—ii: The value of an n-person game.

Shewalkar, A., Nyavanandi, D., and Ludwig, S. (2019). Performance evaluation of deep neural networks applied to speech recognition: Rnn, lstm and gru. *Journal of Artificial Intelligence and Soft Computing Research*, 9:235–245.

Somasundaram, A. and Reddy, U. S. (2016). Data imbalance: Effects and solutions for classification of large and highly imbalanced data.

Sra, S., Nowozin, S., and Wright, S. (2012). *Optimization for Machine Learning*. Neural information processing series. MIT Press.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958.

Stein, R. M. (2005). The relationship between default prediction and lending profits: Integrating ROC analysis and loan pricing. *Journal of Banking & Finance*, 29(5):1213–1236.

Stone, M. (1974). Cross-validatory choice and assessment of statistical predictions. *Journal of the royal statistical society. Series B (Methodological)*, pages 111–147.

Tian, S., Yu, Y., and Zhou, M. (2015). Data sample selection issues for bankruptcy prediction. *Risk, Hazards & Crisis in Public Policy*, 6(1):91–116.

Veganzones, D. and Séverin, E. (2020). Corporate failure prediction models in the twenty-first century: a review. *European Business Review*, ahead-of-print.

Štrumbelj, E. and Kononenko, I. (2014). Explaining prediction models and individual predictions with feature contributions. *Knowl. Inf. Syst.*, 41(3):647–665.

Wahlstrøm, R., Paraschiv, F., and Schmid, M. (2020). Bankruptcy prediction of privately held SMEs using feature selection methods. Submitted.

Wikimedia Commons (2014). Precision and recall. File: `Precisionrecall.jpg`.

Yu, R. and Alì, G. S. (2019). What's inside the black box? ai challenges for lawyers and researchers. *Legal Information Management*, 19(1):2–13.

Zhang, A., Lipton, Z. C., Li, M., and Smola, A. J. (2020). *Dive into Deep Learning*. `https://d2l.ai`.

Zhang, J. and Thomas, L. C. (2015). The effect of introducing economic variables into credit scorecards: an example from invoice discounting. *Journal of Risk Model Validation*, 9(1):57–78.

Zhang, Y., Rahman, M. M., Braylan, A., Dang, B., Chang, H.-L., Kim, H., McNamara, Q., Angert, A., Banner, E., Khetan, V., McDonnell, T., Nguyen, A., Xu, D., Wallace, B., and Lease, M. (2016). Neural information retrieval: A literature review.

Zięba, M., Tomczak, S., and Tomczak, J. (2016). Ensemble boosted trees with synthetic features generation in application to bankruptcy prediction. *Expert Systems with Applications*, 58.

Appendix A

Financial ratio features

A.1 List of all features

Table A.1 shows all 156 features used to perform bankruptcy prediction. All but the last one, $nace\_code$, are obtained as combinations of financial statement features, and are sourced from Wahlstrøm et al. (2020). The last feature is enginereed based on the NACE code, described in Section 2.3.5.

Number	Description
1	(inventory + accounts receivables) / total equity
2	(long-term liability + total equity) / fixed assets
3	account receivable / sales
4	quick assets / current liabilities
5	(quick assets / current liabilities) $\times interestearnedratio$
6	net income / total equity
7	EBITDA / total liabilities
8	total equity / total liabilities
9	short-term liquidity as a percentage of the capital employed
10	short-term liquidity / sales
11	short-term liquidity / current liabilities
12	short-term liquidity / total assets
13	sales / current assets
14	current assets / total equity
15	current assets / sales
16	current assets / total assets (net liquid assets / total assets)
17	current liabilities / current assets
18	current liabilities / total equity
19	current liabilities / total liabilities
20	current liabilities / sales
21	total liabilities / total assets
22	debtors / creditors (receivables/payables)
23	operating profit / (operating profit - interest expense)
24	EBIT / total assets
25	EBITDA / interest expense
26	effective tax rate
27	total equity / total assets
28	total equity / long-term liability
29	sales / total equity
30	pre-tax profit as a percentage of the capital employed
31	financial expenses / sales

Continued on next page

Number	Description
32	EBIT / sales
33	sales / fixed assets
34	fixed assets / total assets
35	fixed assets / total equity
36	intangibles / total assets
37	interest expenses / total revenues
38	interest-bearing debt / total equity
39	inventory / current liability
40	inventory / working capital
41	investment turnover (sales / (total equity + total liabilities))
42	total liabilities / total equity
43	long-term liability / current assets
44	net income / stockholders equity (return on shareholder's equity)
45	net income / sales
46	(total revenues - sales) / total revenues
47	total equity / fixed assets
48	total equity / sales
49	no-credit interval
50	dummy; one if total liability exceeds total assets
51	operating expenses / sales
52	short-term liquidity / total liabilities
53	operating profit / total revenues
54	operating profit / paid-in capital
55	operation asset / total asset
56	personnel costs / added value
57	pre-tax net profit / paid-in capital (ordinary income / stockholder's equity)
58	net income / total revenues
59	profits / net working capital
60	quick assets / sales
61	quick assets /total assets
62	earnings after tax and interest charge / net capital employed
63	current liabilities / earnings before tax and interest charge
64	retained earnings / sales
65	retained earnings / total assets
66	return on debt (earnings / total liabilities)
67	net income / total assets
68	total revenues / fixed assets
69	total revenues / total assets
70	total revenues / net working capital
71	sales / total assets
72	total assets / total revenues
73	total expenses / assets
74	total revenues / total expenses
75	working capital / current liabilities
76	working capital / sales
77	working capital / total assets
78	working capital / total equity
79	dummy; one if paid-in equity is less than total equity
80	working capital / total revenues
81	accounts payable / total assets
82	public taxes payable / total assets
83	EBIT / total liabilities
84	(non-interest expenses - salary) / total assets
85	(shareholder's equity + total revenues) / total assets
86	sales / working capital

Continued on next page

Number	Description
87	short-term liquidity / current assets
88	cost of goods sold / inventory
89	cost of goods sold / sales
90	(current assets - short-term liquidity) / total assets
91	current assets / common shareholder's equity
92	current liabilities / total assets
93	dividends / net income
94	working capital / long-term liabilities
95	working capital / operational expenditure
96	EBIT / total tangible assets
97	financial expenses / sales
98	fixed assets / (stockholder's equity + long-term liabilities)
99	(sales - cost of goods sold) / sales
100	income gearing
101	intangible assets / sales
102	interest expenses / total liabilities
103	interest expenses / total expenses
104	interest income / interest expenses
105	interest income / total assets
106	inventory / cost of goods sold
107	inventory / current assets
108	inventory / sales
109	long-term liabilities / total equity
110	long-term liabilities / total assets
111	sales / tangible assets
112	net income / gross profit
113	net income / total capitalization
114	net quick assets / inventory
115	total equity / (total equity + long-term liabilities)
116	non-interest expenses / operating profit
117	total revenues / sales
118	ordinary income / total equity
119	ordinary income / ordinary expenses
120	pre-tax profit / sales
121	pre-tax profit / total assets
122	owners equity / total assets
123	payable / current liabilities
124	payables / inventories
125	retained earnings / inventory
126	retained earnings / tangible assets
127	return on capital employed
128	return on net fixed assets
129	salary / total assets
130	sales / short-term liquidity
131	sales / inventories
132	sales / receivables
133	sales / total tangible assets
134	interest bearing debt / total liabilities
135	share of labour costs
136	(short-term assets - total liabilities) / total assets
137	solvency ratio
138	sales / stock holders equity
139	(total revenues + interest income) / total expenses
140	interest expenses / total assets
141	operating expenses / total assets

Continued on next page

Number	Description
142	tales / assets employed
143	EBITDA / total assets
144	operating profit / total assets
145	operating profit / sales
146	(current liabilities - short-term liquidity) / total assets
147	accounts payable / sales
148	retained earnings / current liabilities
149	(total equity - intangible assets) / (total assets - intangible assets - short-term liquidity)
150	EBIT / interest expense
151	accounts receivables / total liabilities
152	profit before tax/current liabilities
153	current assets/total liabilities
154	log(age in years)
155	log(total assets)
156	nace_code

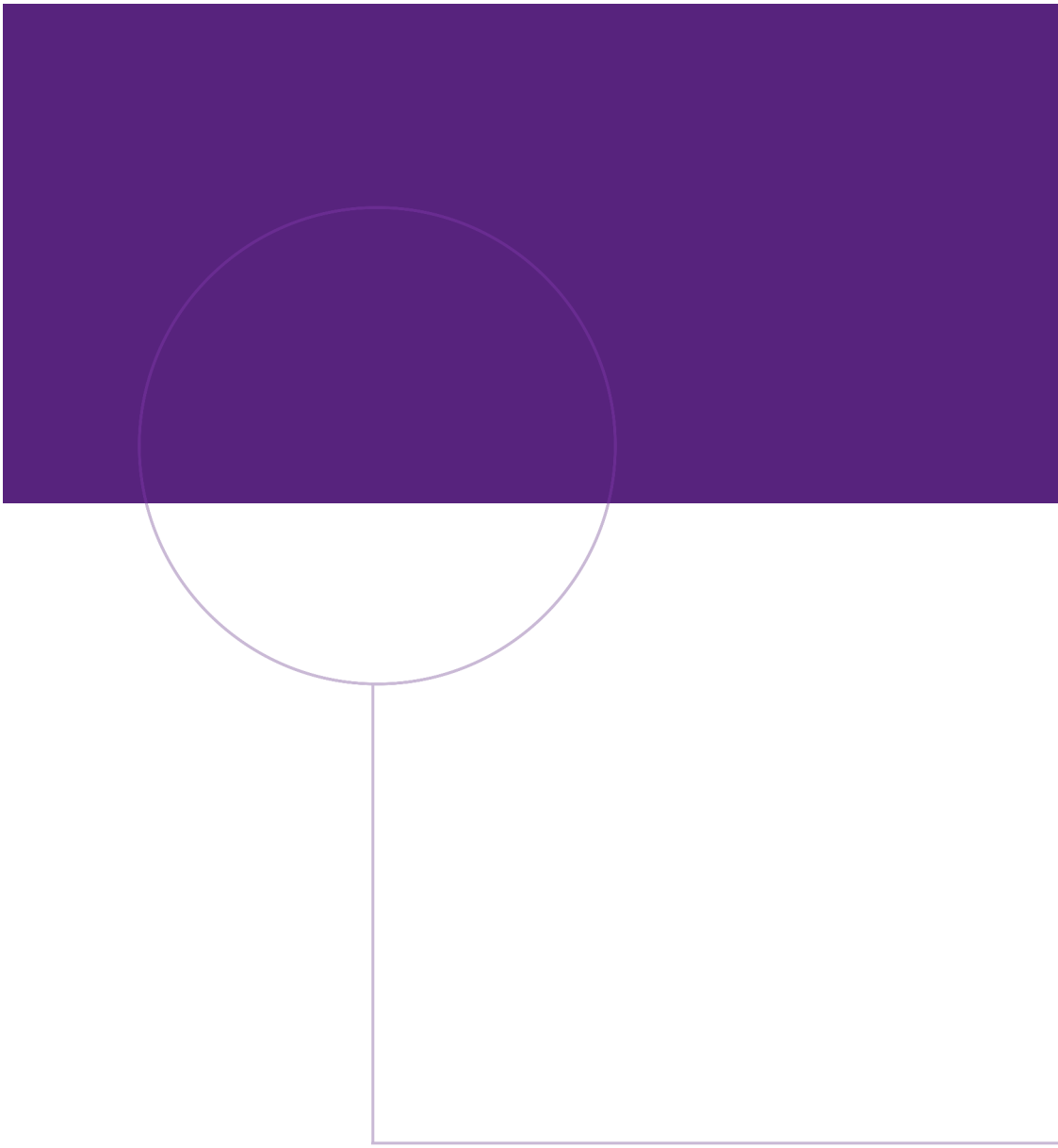
Table A.1: All 156 features used to perform bankruptcy prediction.

A.2 Financial abbreviations

In order to easier display the features in visualisations, we have used a number of abbreviations for the feature names in Table A.1, particularly in the SHAP plots in Sections 6.5 and 7.3. These are shown in Table A.2. While there is some ambiguity, the meaning should be clear from the context.

Original name	Abbreviation
intangible	int
liquidity	liq
liquid	liq
short-term	st
revenue	rev
ordinary	ord
liabilities	liab
total	tot
income	inc
shareholder's	sh.h
stockholder's	st.h

Table A.2: Abbreviations used when visualising features.



NTNU

Norwegian University of
Science and Technology