ERASMUS UNIVERSITY ROTTERDAM

Erasmus School of Economics

Master Thesis - Econometrics: Quantitative Finance

A Multimodal Deep Learning Approach to Default Prediction

Date final version: September 19, 2021

Supervisor:
Prof. Dr. R.L. Lumsdaine

Second Assessor:
S.H.L.C.G. Vermeulen

Author:
Yassine Bouzbiba

Student Number:
432260

Abstract

In this paper, I investigate the application of multimodal machine learning methods to bankruptcy prediction. In previous literature, the prediction of bankruptcy is limited to machine learning methods applied to either accounting or text data. Several papers have attempted to combine the insights derived from these two separate data modalities by using a weighted combination of the predictions obtained. However, none of these papers consider the dependencies in and the structure of this multimodal data. I propose a methodology in which accounting and text data are merged at the stage prior to applying bankruptcy prediction methods using a multimodal deep Boltzmann machine. The results indicate that taking the dependency and data structure across modalities into account does not offer improvements in precision and recall relative to the use of traditional machine learning approaches applied to standard accounting ratios. Furthermore, the use of Support Vector Machines using a single data modality has shown the best overall performance defined as the recall and precision of the predictions.

Contents

1 Introduction

Default prediction has been a longstanding area of research with the focal point being the prediction of the probability of default (PD). The problem of PD modelling has seen several approaches which have largely been based on a stochastic analysis (Merton (1974), Vasicek (1984), McQuown (1993), and Kealhofer (1995)). Other approaches revolve around the use of firm characteristics for predicting PD. Beaver (1966) uses a selection of accounting ratios to predict defaults. Altman (1968) applies Multiple Discriminant Analysis (MDA) for predicting defaults using accounting ratios. Ohlson (1980) improves upon both these approaches by applying a conditional logistic regression as a more econometrically plausible methodology.

Subsequent literature has incorporated machine learning approaches in predicting PD, as these techniques allow for the use of more data while not being dependent on predefined model specifications (Odom and Sharda (1990), Tam and Kiang (1992), and Barboza et al. (2017)). Apart from the use of accounting data, the use of text data has become more popular leading to new methods for predicting defaults. An example of this is sentiment analysis, which investigates the tone of a company's financial publications and what implications this could have for its financial risk (Cerchiello et al. (2017) and Gentzkow et al. (2019)). Other approaches make use of techniques related to word frequency (Stevenson et al. (2021)).

The most recent research in default prediction makes use of a combination of text and accounting data (Mai et al. (2019)), also referred to as combining modalities. The prevalent way of combining data is by applying methods to each modality separately and then fusing individual outputs to obtain a combined output. A disadvantage of this method is that it neglects the structure of multimodal data as it does not correct for potential dependencies that might exist between data from different modalities.

This paper considers the combination of data modalities in a machine learning framework for predicting defaults prior to the use of inferential techniques. The methodology comprises of a combination of a dimensionality reduction technique and a neural network. The dimensionality reduction technique used is known as the multimodal Deep Boltzmann Machine (DBM) (Srivastava et al. (2012)). The multimodal DBM is used to create a reduced representation of the combination of accounting data and text data. This reduced representation is then inputted into a Deep Neural Network (DNN) in order to make a prediction for the bankruptcy status of a company. This methodology will be benchmarked against a support vector machine (SVM).

The analysis in this paper is based on data extracted from the Eikon Datastream database by Thomson Reuters and the Securities and Exhange Comission's Electronic Data Gathering

Analysis and Retrieval system (EDGAR). Eikon Datastream is used for retrieving accounting data and EDGAR is used to retrieve Management Disclosure and Analysis (MDA) sections of the annual reports of the companies considered. By matching observations for which both the data in Eikon and EDGAR is available, I am able to build a panel data frame for analysis which shows the development of a company throughout the sample. Several adjustments have been made including oversampling of the minority class, removing duplicates, KNN imputation, and concatenating adjusted data frames. Text embedding using the Doc2Vec methodology is also implemented to convert the text data into numeric vectors which are used as input for the machine learning methods that are implemented.

The results show that even when the dependencies in the multimodal data are taken into consideration, performance is still inferior to methodologies that do not take these dependencies into consideration. Furthermore, using only accounting data offers the most accurate default predictions. The reason for this inferior performance could be related to a misspecification in the text conversion methodology. As the methodology used in this paper comprises of a complex architecture, the exact reason for this inferior performance remains subject to speculation.

To my knowledge, this paper is the first to take the data structure of combined modalities into consideration in the field of default prediction using machine learning techniques. With this paper, I wish to motivate further research into issues related to data structure in combined modalities for default prediction.

This paper is organized as follows. In section 2, I provide a review of the literature where I place emphasis on the application of machine learning methods in default prediction. I also touch on methods that are used for the combination of text and accounting data as well as other research related to multi-modal machine learning methods. Section 3 describes the methodology that is used for the pre-processing of text data, the combination of data modalities, and the prediction of bankruptcies. Section 4 gives a description of the data. This mainly revolves around the process of parsing text from the EDGAR database and issues related to the data. The selection of what firms for what years will be used in my analysis and the labeling of the data is also mentioned here. Section 5 provides a discussion of the results along with a discussion on the hyper parameters used and the adjustments that are made. Section 6 and 7 provide concluding remarks, limitations of this paper, and recommendations for future research.

2 Literature Review

The prediction of default has been the subject of a great strand of literature. A divergence exists between the literature that places emphasis on default prediction related to consumer loans and the literature related to corporate loans. This thesis will only consider the latter. First, an overview of classic approaches to default prediction are discussed. I then describe the current state of knowledge on machine learning applied to default prediction by highlighting the most important outcomes and trends in the field using the existing literature. The most important findings are used as a guide for creating a structured perspective of existing research for this thesis. I also highlight the gap in the existing research which this thesis intends to fill. Lastly, I review the literature behind the methodology I intend to use to show its relevance from a technical perspective.

Generally, two approaches can be taken in predicting defaults. One approach is based on stochastic analysis and makes use of the probability of default based on the Merton model. Several extensions of this model exist in which the probability of default is considered in a portfolio framework incorporating market based information (Merton (1974), Vasicek (1984), McQuown (1993), and Kealhofer (1995)).

Another approach utilizes econometric analysis and makes use of methodologies such as multiple regression analysis and multiple discriminant analysis (Beaver (1966) and Altman (1968)). Some of the earliest methods make use of accounting ratios in a classical regression setting. There are however several shortcomings to this way of prediction. One of them has to do with the assumptions of normality and independence of the residuals. This leads to incorrect statistical inference such as over stating the power of predictions. An improvement can be offered by making use of a conditional logistic regression analysis which improves on the these issues (Ohlson (1980)).

One commonality that the above methods share is that they all use accounting ratios. The advantage of using accounting ratios has to do with comparability and intuitiveness. Another advantage has to with the availability of accounting data. Apart from accounting data, another useful source of data is market data. This relates to information such as price-to-earnings ratios and EV/EBITDA multiples (Atiya (2001)).

This begs the question of what accounting ratios are most useful in predicting bankruptcies. The majority of papers use the ratios as proposed by Beaver (1966) and Altman (1968). Some papers supplement these ratios by making use of market data to derive insights that are more up to date with the latest developments in the market. Accounting ratios namely reflect accounting

statements which have a publication lag (e.g. Hubbard and Vetter (1996)). The selection of accounting ratios in this thesis follows the literature in this regard.

The use of accounting data has been extended to incorporate higher volumes of data in more recent times. To cope with this large amounts of data, the use of parametric approaches might pose problems related to misspecification and omitted variables. In this case it is more useful to refer to methodologies that do not have a fixed model specification. These are so-called non-parametric methods among which machine learning methods are the most widely used. These methods are also proven to perform better than MDA and MRA for default prediction with the NN being particularly competitive (Ohlson (1980), Odom and Sharda (1990), and Tam and Kiang (1992)).

One machine learning method that deserves special attention is SVM. Some empirical evidence shows the superiority of SVM to NN. This has mainly to do with SVM guaranteeing a unique solution to optimization problems. NN on the other hand uses an optimization method known as gradient descent which is dependent on an intitialization, This means that NN could give different solutions depending on the initialization used. Another advantage that SVM has comapared to NN is the small number of hyper parameters of the model which decreases the amount of tuning required. One should note however that the performance of SVM improves as the evaluated data is more linearly separable. These advantages and disadvantages highlight the usefulness of SVM as a benchmark (Shin et al. (2005)).

The above approaches are based on financial data. An increasing strand of literature, however considers the use of text data as it is more prevalent in the world of finance, e.g. annual reports, news articles, and research reports. Especially the use of social media by large financial institutions and agents has forced much attention onto these kinds of data causing an increase in their popularity (Cerchiello et al. (2017)). The question then remains as to what the best way is to incorporate this kind of data for practical applications. An often cited paper is Gentzkow et al. (2019) which discusses the practicalities around the use of text as data in economic research. It also offers a practical guide for preprocessing text data to make it useful in economic research as the use of text data could suffer from shortcomings such as high dimensionality. Because of this high-dimensionality, machine learning methods (i.e. non-parametric methods) are preferred as this thesis investigates. The recommendations in Gentzkow et al. (2019) are used in the pre-processing of the text data in this thesis. Furthermore, the findings in Cerchiello et al. (2017) and Gentzkow et al. (2019) suggest that the use of text data could offer additional explanatory power in explaining financial phenomena.

Some papers show that there is a difference between the information that is captured by text

data as opposed to accounting data. This suggests that the use of both data modalities could add to the predictive power of these methods. What remains unclear is what kind of machine learning method does the best job of combining data modalities. Different papers suggest different approaches (see for instance Mai et al. (2019) and Stevenson et al. (2021)).

These conflicting results suggest that the combination of text data and accounting data is promising but undecided on the exact methodology to use. Furthermore, it also suggests that the architecture of the methodology used could have something to do with the performance of default predictions when using multimodal data. A shortcoming of the existing literature is that it fails to discuss how the approach of combining the modalities of data affects prediction results.

Technically, the approach to modality combination falls within the scope of model architecture, an area that has not gotten a lot of attention. This is the gap that this thesis intends to fill. This thesis aims to contribute to the literature by filling the void created by the lack of focus on the methods used for combining data modalities. The application of default prediction is especially useful because of the complexity of the dependencies and structure of the data (Zhao et al. (2016)).

Multimodal machine learning applied to default prediction has mainly made use of artificial neural networks. A traditional artificial neural network contains an input layer, a hidden layer and an output layer which are all connected to each other. The objective of a neural network is to approximate a function $f(x)$ that maps a certain input to an output. A neural network with at least one hidden layer could approximate any Borel measurable function[1] with any level of accuracy as long as the input space has finite dimensions. This is possible due to the Universal Approximation Theorem (Hornik et al. (1989)). Non-linear relationships between the input vector and the output of the neural network can be introduced in this layer architecture through activation functions. These functions offer a way to transform linear output from a node in the network into a non-linear output. In a DNN there are multiple hidden layers (as can be seen in figure 3). DNNs are essentially an extension of the traditional artificial neural network. This extension is in the depth of the network meaning the number of layers and the amount of neurons in the network. Essentially, this increase in hidden layers allows for more complexity in fitting the input data to the output data.

In the context of deep neural networks, data is combined by simply inputting different data modalities into a separate neural network. These neural networks are individually trained upon which the output variables are combined. This can be done in different manners such as an equal

[1]Simply put, a Borel Measurable function can be seen as any function which maps any finite dimensional space to another finite dimensional space.

weighting of the output from network 1 and the output from network 2 (D. Sun et al. (2018) and Mai et al. (2019)). As straightforward as combining modalities in the above methodology might sound, it does not take the dependency of different variables with respect to each other into account. Another method that does take this dependency into account is based on Boltzmann machines. Boltzmann Machines are a network architecture that consists of one hidden layer and one visible layer in which all the nodes in the network are connected to each other. Boltzmann machines are a stochastic approach to neural networks and basically maximize the likelihood of the input features given in the visible layer conditional on the latent variables given in the hidden layer. Essentially, a forward pass and a backward pass are implemented in which the input data is "copied" as accurately as possible. This means that the input data from the visible layer reaches the hidden layer and then gets passed back through the corresponding weights in the weight matrix. The goal is to minimize the error between the input that was passed back from the hidden layer (the copy) and the original input data. In this way the network is able to discover the dependency and the structure of the input data. However, the practical usefulness of Boltzmann machines does not present a competitive contestant in the machine learning literature. The performance of Boltzmann machines could be improved tremendously by constraining some of the connections in the network (Osborn (1990)).

Restricted Boltzmann machines (RBM) are an example of a different layout for the Boltzmann Machine in which these constraints on the connections are imposed. The model can be seen as an undirected graphical model and is called restricted because of the fact that the nodes within a layer are not connected to each other which restrics intra-layer communication. RBM is an unsupervised learning model and is mainly used for feature extraction. The model is mechanically similar to Boltzmann machines. The probabilities inside the network are based on energy functions and make use of softmax normalization. The training of the network is based on the so-called contrastive divergence algorithm. By use of a combination of Gibbs sampling (see appendix) and increasing and decreasing probabilities based on the input data, the model is able to fit the weights accurately. The idea is to decrease the energy of features that resemble the training set while increasing the energy of features that are sampled by the model using Gibbs sampling (see Hinton (2002) for more on contrastive divergence). The RBM is very similar to an autoencoder (for more on auto encoders I refer the reader to Goodfellow et al. (2016)) because of the way it encodes and decodes the input data to detect a pattern in the data. The original use of these Boltzmann machines is intended with input data that is binary. However, several efforts have been made to train models using real valued data. This ignited the emergence of several methods such as the Gaussian-Bernoulli Boltzmann machines.

Furthermore, the concept of a Deep Boltzmann machine was introduced, which simply increases the amount of hidden layers in the network (Welling et al. (2004)).

DBM's in a multimodal context work similar to the DNN approach. Consider an example with two input vectors of different modalities. The vectors are first modelled separately using a DBM. The two seperate DBM's are then combined using a joint layer. The output of this joint layer is a reduced version of the multimodal input data. This data can then be used for prediction. Moreover, the multimodal DBM has proven to be practically useful in financial applications unlike other multimodal methodologies that consider data dependence structures. Apart from this, lots of information is available on the use of multimodal DBMs. Because of its practicality and the wide range of information that can be found on this model, it presents a good candidate for combining the accounting and text data in this thesis. A disadvantage of the multimodal DBM however, has to do with training. The softmax function used to determine the appropriate probabilities in the different nodes of the network could suffer from imbalance. It might be the case that one modality has a very high dimension compared to the other modality. This is usually resolved by using oversampling. Nonetheless, this methodology[2] presents a promising avenue for the applications in this thesis (Srivastava et al. (2012) and Assis et al. (2018)).

[2]A full explanation of the various Boltzmann machine architectures can be found in the appendix combined with a full derivation of the statistical properties and training methodologies of these methods.

3 Methodology

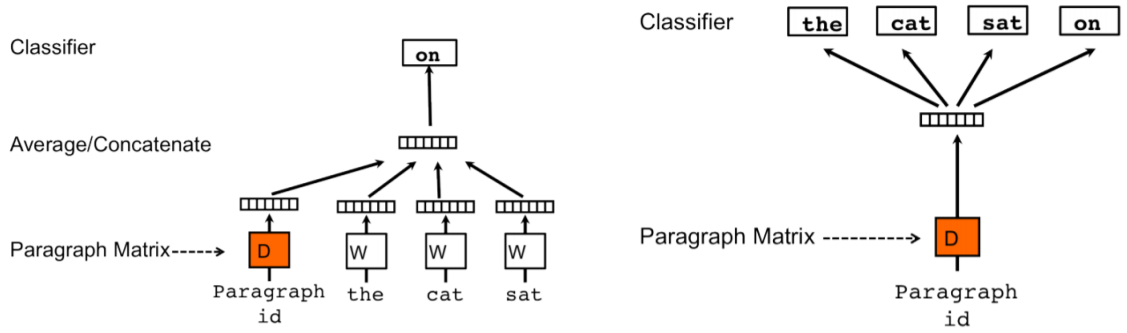
In this section the methods are discussed that are inspired by the reviewed literature. Firstly, the methodology for transforming raw text into useful data is described. I then discuss the method used for combining the different modalities of data. Thirdly, the classification methodologies are described which are used in mapping the input data to forecast the future state of a company, i.e. bankrupt or not bankrupt. Because the methods that are discussed are meant for classification, an adjustment is made to make sure that the methods can be used in a forecasting context. This adjustment is explained in detail. Furthermore, the procedure by which the methods are fitted to the data is discussed. Several shortcomings and pitfalls in fitting machine learning models to data are also highlighted. To evaluate the proposed methods, a certain measure of their performance is needed. This follows in the discussion on performance metrics.

3.1 Using text as data

The raw text that is used in the conversion must first be separated into documents, denoted D. This separation is dependent on the level on which the prediction variable is defined. The text used for the purposes of this thesis is extracted from the MDA section of the annual reports (form 10-Ks) of the companies in the evaluated sample. In this thesis the objective is to predict bankruptcy on a company level for a given year. Therefore the documents D are defined as the text of each MDA section for a given company in a given year. The documents can be further split into tokens. A token can be defined as any predefined subset of a string of text such as a word or phrase. Using words to tokenize the sentence "I hope you are doing well." would then result in the tokens $\{I\}, \{hope\}, \{you\}, \{are\}, \{doing\}, \{well\}, \{.\}$. Note that any punctuation also gets its own token. An often-raised concern in natural language processing has to do with the high dimensionality of text data. Therefore, raw text data is first pre processed by removing punctuation, numbers, HTML tags, and symbols to reduce dimensionality. Additionally, the dimensionality of the text data can be reduced by removing stop words, i.e. "but", "from", "for", "until", "too", and "just". Finally, a procedure named stemming is often applied. Stemming is the process of reducing a word to its basic form such that for example "swimming" and "flying" become "swim" and "fly". By using stemming, a reduction in the number of unique words is realized (Gentzkow et al. (2019)).

A word embedding gives a word a representation such that words with similar meaning have similar representations. The objective of any word embedding method is therefore to create a representation that shows high similarity for words that are semantically similar. Semantic

similarity can be formalized as the relation between words on a contextual basis for instance. A mathematical definition of semantic similarity is given by the conditional probability $p(w_{t+j}|w_t)$, which denotes the probability that the vector representation of a word w emerges in j positions from t given that this word occurs in position t. To illustrate semantic similarity, if the word "metro" is mapped to a certain representation such as a numerical vector, then its difference to the representation of the word "bus" (i.e. the numerical vector for the word "bus") would be smaller than to the representation of the word "pigeon" (i.e. the numerical vector for the word "pigeon"). In a text, the words "metro" and "bus" will be used in similar positions in the grammatical construction of a sentence and both represent a mode of public transport which is not the case for the words "metro" and "pigeon". To maximize the semantic similarity, the goal is to maximize the probability that a certain word appears within a specific range of words in a text, that is, $\frac{1}{T}\sum_{t=k}^{T-k} p(w_{t+j}|w_t)$, where T denotes the length of this context. One such algorithm that uses this approach is the Doc2Vec algorithm by Le and Mikolov (2014) which is an extension of the Word2Vec algorithm introduced by Mikolov et al. (2013). The Doc2Vec algorithm uses a neural network approach for mapping D to a numerical vector using two different configurations. The first configuration uses the word vectors w together with a vector representation of D to predict the next word in a sentence from the document. The algorithm then improves the vector representation of D until convergence which is reached when the prediction of the next word in a sentence is done correctly for a specified percentage of cases. This is called the distributed memory model and is shown in figure 1a.



(a) Paragraph Vector: A distributed memory model

(b) Paragraph Vector without word ordering: Distributed bag of words

Figure 1: Two architectures of the Paragraph Vector model Le and Mikolov (2014)

The other configuration does not consider the word order in a sentence sampled from the document D but rather uses a random sample of words from the document D, which is called a text frame. The neural network then intends to predict the words in this text frame from the vector representation of D. Throughout its iterations the neural network learns a vector

representation of D. This configuration is called the distributed bag of words approach and is shown in figure 1b. Doc2Vec is recommended to be implemented using a combination of the distributed memory model approach and the bag of words approach Le and Mikolov (2014). The reason for this is because this combination offers consistency across the majority of tasks. Another important consideration is the size of the vector for the embedding of the documents which is a trade-off between the accuracy of the embedding, the amount of data available, and the eventual use of the embedding. Usually, algorithms can be used to determine a lower bound on the amount of elements in the text embedding (Patel and Bhattacharyya (2017)). However, in this thesis there are several factors that force a deviation from this strategy. Since the objective is to use the embeddings in a classification/prediction task in which the embedding is combined with the accounting data, the dimensions of the accounting data and the embedding must not differ too much. Furthermore, given the small amount of data available, it is best to not increase the elements of the embedding by a substantial amount since there is not enough data to reach the accuracy needed for creating a embedding of size 100 for instance. Therefore I choose an embedding size of 50 for the text data of the MDA sections. The size of the context (also called window size) used to train the algorithm is dependent upon how many surrounding words are needed to find out the meaning of a word. Generally, the larger the window size, the better Doc2Vec is able to find topic and domain specific information (Levy and Goldberg (2014)). Because I intend to deduce whether a company is going bankrupt, i.e. topic specific information I use a window size of 5 instead of the usual 3. One other specification has to do with the algorithm used for determining the probabilities $p(w_{t+j}|w_t)$. In this thesis I use hierarchical softmax[3].

3.2 Combining data modalities

This section describes the methodology related to combining text data and accounting data. This is done using the multimodal Boltzmann machine. The multimodal deep Boltzmann machine consists of standard Boltzmann machines and RBMs.

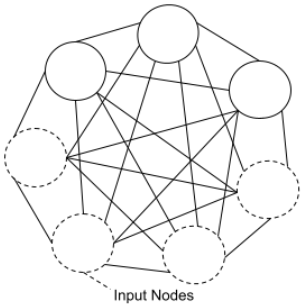
The objective of the Boltzmann machine is to approximate the distribution of the data in the input layers, which is denoted $P_{input}(V)$, by the distribution that results from the Boltzmann machine, which is denoted $P_{hidden}(V)$. The loss function to determine the similarity between $P_{input}(V)$ and $P_{hidden}(V)$ is the Kullback-Leibler divergence given in equation 9.

[3]More on this can be found in the technical explanation and derivation of the Doc2Vec algorithm in the appendix.

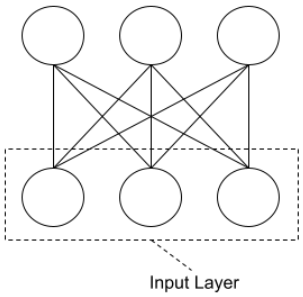
$$G = \sum_{v} P_{input}(v) \ln \left(\frac{P_{input}(v)}{P_{hidden}(v)} \right) \tag{1}$$

A drawback of the Boltzmann machine is that the complexity of the model's structure increases exponentially as the number of input nodes increases. For this reason RBMs are used which are Boltzmann Machines in which the interconnected structure is limited to adjacent layers meaning that there are no connections between nodes in the same layer (see figure 2).

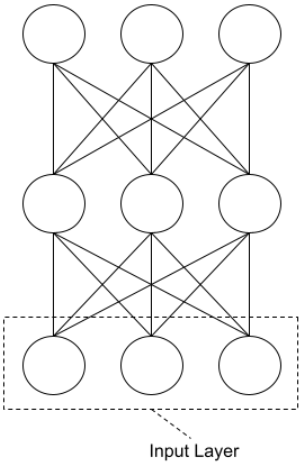
A more accurate representation of the input data can be found through use of the Deep Boltzmann Machine (DBMs) (see figure 2). It is possible to combine two separate DBMs[4] for different modalities of data to generate one type of output of data by connecting them to a joint layer. Although this might seem similar to the approach seen in Mai et al. (2019), the difference is that the Boltzmann architecture is used to obtain a data encoding whereas a DNN is used for predicting a dependent variable.



(a) Boltzmann Machine



(b) RBM



(c) DBM

Figure 2: Illustration of the difference between a Boltzmann Machines, a RBM, and a DBM

[4]A detailed technical description and derivation of the probability distributions and training algorithms of the various Boltzmann machine configuration can be found in the appendix.

An adjustment is needed when applying the multimodal Boltzmann machine to modalities that are measured using a real valued vector. This adjustment is the replacement of the input layer in the DBMs by a Gaussian RBM (Srivastava et al. (2012)). As accounting data and text embeddings are also expressed in real valued vectors the first layer in the multimodal deep Boltzmann machine in this thesis is a Gaussian RBM.
The multimodal DBM using a Gaussian RBM input layer can be implemented using the TensorFlow library for Python. The first step is to create a DBM for the accounting data and one for the text data. Then the individual DBMs are connected using another DBM which outputs a reduced representation of the combined modalities.

3.3 Classification

This section describes the methods used to perform the classification of the observed companies in the data set. In the literature review, the contributions of Mai et al. (2019) are highlighted as a seminal paper for this thesis. Furthermore, the competitiveness of SVMs, as argued by Shin et al. (2005), qualified this model as a useful benchmark. I elaborate on NNs and SVMs and how these methods are applied to this thesis.

3.3.1 Neural Networks

This subsection describes classification using NNs. In the literature review, the contributions of Barboza et al. (2017) with regards to the use of neural networks are highlighted. DNNs consist of multiple layers of neurons. In essence, the objective is to find a mapping $\hat{f}(x)$ from the input vector to a given output. This is achieved by finding weights for the nodes such that this mapping can be realized. The DNN consists of several parts. The activation represents the product of the values of a given layer and the weights leading to the adjacent layer. The activation then serves as input to an activation function. The activation function is a function that transforms the activation to generate a non-linear output. The activation function yields the output of one node. The procedure can be repeated for all nodes in all layers until we reach the output layer which is then mapped to the output $\hat{f}(x) = \hat{y}$. The loss function shows the difference between the output as given by the neural network versus the actual value that should be generated. Different loss functions exist which are shown in the appendix. Based on the loss function, the weights and biases in the network are adjusted. In order to do this, the sensitivity of the loss function to the weights and biases is determined which is essentially the partial derivative of the loss function with respect to the weights and biases. The algorithm that is used to then adjust the weights and biases using this partial derivative is called backpropagation. Passing the

entire data through the neural network once is called an epoch. For training a neural network multiple epochs are used. The result of training the algorithm using the data could lead to several issues. Overfitting describes the situation in which the fit obtained cannot be properly generalized to new data. Underfitting represents the situation in which the obtained fit cannot be used because it is close to a straight line. Overfitting can be resolved using regularization. An often used method is called dropout Srivastava et al. (2014) where nodes are dropped out of the neural network. This makes the neural network more robust and hence can solve overfitting. Technical details of the DNN are given in the appendix.

In this thesis the objective is to predict bankruptcies for our sample of companies using text data and accounting data. In the literature this is known as binary classification. The configurations of the DNNs are based on the recommendations for binary classification problems. Because Mai et al. (2019) forms the foundation for this thesis, I make use of several of their recommendations to implement the deep learning network used. One recommendation they make is the use of the ADAM activation function (Kingma and Ba (2014)). Another recommendation Mai et al. (2019) make has to do with the architecture of the neural network they use. They recommend the nodes in the network to have four nodes per layer with four layers in the network. However, the network they use has an input layer of four nodes which means that the exact specification for the purposes of this thesis has to be found through a structured search by experimenting with different parameters called hyperparameter tuning. For the loss function for binary classification problems there are two options namely the Hinge Loss and the Binary Cross Entropy Loss. The Binary Cross Entropy Loss usually provides better probabilistic estimates in binary problems. Table 1 shows an overview of all the parameters along which I perform this search for the optimal parameters.

	DNN Text Data	DNN Accounting Data	DNN Multimodal Data
Layers	2, 4, 6, 8	2, 4, 6	2, 4, 6, 8
Neurons per Layer	4, 8, 10, 12, 20, 25, 30	4, 8, 10, 12, 20, 30	4, 8, 10, 12, 20, 25, 30
Activation Function	ReLu, LeakyReLu, Tanh, Softmax	ReLu, LeakyReLu, Tanh, Softmax	ReLu, LeakyReLu, Tanh, Softmax
Loss Function	Binary Cross Entropy, Hinge Loss	Binary Cross Entropy, Hinge Loss	Binary Cross Entropy, Hinge Loss
Optimization Method	SGD, ADAM	SGD, ADAM	SGD, ADAM
Learning Rate	0.1, 0.01, 0.001, 0.0001	0.1, 0.01, 0.001, 0.0001	0.1, 0.01, 0.001, 0.0001
Batch Size	16, 32, 64, 128, 256	16, 32, 64, 128, 256	16, 32, 64. 128, 256
Source	Liu et al. (2016)	Odom and Sharda (1990)	Mai et al. (2019)

Table 1: Parameter values considered for GridSearchCV. The values of these parameters are evaluated using the below mentioned sources.

The neural networks are used as a method for evaluating the additional performance of a

classification using the combined modality which is the output of the deep Boltzmann machine. This is illustrated in figure 3. The neural network can however also be implemented using single modalities (meaning using only the accounting data or the text data) for benchmarking. In this thesis I do both. Additionally, I also benchmark the implementation using the output of the deep Boltzmann machine by implementing a neural network using all available features as input (meaning both the accounting and text data).

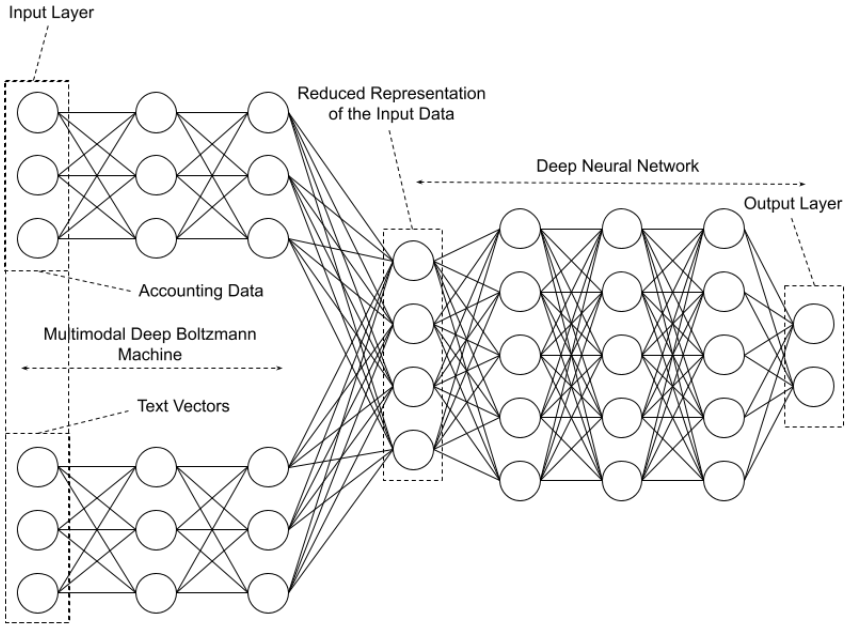


Figure 3: Multi-modal Architecture

3.3.2 Support Vector Machines

In this subsection I describe bankruptcy classification using SVMs. In principle, a set of data points can be separated into groups using a straight line. In this case SVM is a binary classification method that separates data points using a straight line or a hyper plane. But in practice there are cases in which data points cannot be separated in this simple manner. This occurs in the case where a certain data point belonging to a certain class, say class A, might be closer to data points of another class, say class B. Drawing a straight line between data points might not be a good plan whereas allowing some points to be misclassified might give a better overall result. The line that separates the data points in this case is called a soft margin or Support Vector Classifier. In some cases, the data points can be separated only in higher dimensions. Raising the dimensions of data is a computationally expensive task but by using kernel tricks the process can be made more efficient. The classification namely only requires a measure of distance between the data points. Using a kernel function can allow us to measure the distance

in higher dimensions. Several kernel functions exist for SVMs. For non-linear classification the polynomial kernel and radial basis function kernel are the most popular.
The application in this thesis requires such a non-linear classification. As is the case for neural networks, finding the optimal parameters for SVM requires hyper parameter tuning. A preliminary selection of hyper parameters can be based on the literature. As the objective in this thesis is the classification of bankruptcies I follow some of the recommendations made in the literature (Shin et al. (2005), Min and Lee (2005), and Erdogan (2013)). An overview of the hyper parameters along which we perform an exhaustive search is given in table 2. Additionally, a full derivation of the SVM methodology is given in the appendix.

	SVM Text Data	SVM Accounting Data	SVM Multimodal Data
Kernel	Polynomial, Rbf	Linear, Polynomial, Rbf	Linear, Polynomial, Rbf
C	$2^7, 2^9, 2^{11}$	1, 10	1, 10, $2^7, 2^9, 2^{11}$
Degree	1, 2, 3	2, 3	1, 2, 3
Coefficient	0.6, 0.06	0.6, 0.06	0.6, 0.06
Gamma	$2^{-7}, 2^{-5}, 2^{-3}$	0.5, 1.0	$2^{-7}, 2^{-5}, 2^{-3}, 0.5, 1.0$
Source	A. Sun et al. (2009)	Shin et al. (2005)	

Table 2: Parameter values considered for GridSearchCV. The values of these parameters are evaluated using the below mentioned sources.

The SVM is implemented in similar manner to the neural network. I implement the SVM for the accounting data and text embeddings separately and I implement the SVM on the combination of accounting data and text embeddings. This serves as another benchmark for the neural network implementation using the multimodal deep Boltzmann machines. Another point relates to the predictions made by the SVM classifier. Since I implement SVM using the Scikit learn package for python, the predictions generated are the probability of the observation belonging to either the bankrupt or non bankrupt class. However, the exact boundary that determines when an observation is classified as bankrupt or non-bankrupt can be treated as a hyper parameter and can therefore be tuned where the intention is to maximize the performance measures over the test data (the distinction between train and test data is given in the next section).

3.4 Adjustments for Forecasting

Given that this paper uses forecasting in a neural network, the way in which training is executed is slightly adjusted. This section describes this adjustment. Considering the description of the methodology thus far, a static neural network will namely only provide benefits for classification problems. By preparing the samples that are used for training in such a way that classification

is performed for labels in the future, forecasting can be made possible. The most common time frame to do this in bankruptcy studies is a one-step out prediction (Bellovary et al. (2007)). In this thesis, I do this by predicting the dependent variable in time $t+1$ using the data of time t. Then I continue by using $t+1$ for predicting $t+2$ and $t+3$ for predicting $t+4$ and so forth.

3.5 Training and Testing

In this section I describe the procedure that is used to train the machine learning methods applied. As Goodfellow et al. (2016) describe, the exact optimal parameters for a machine learning model must be determined using experimentation. For this experimentation, several widely used techniques are available. One of the best known is k-fold cross validation. This technique splits the data that is evaluated by a model into k folds. Of these k-folds one fold is kept as a test set while the remaining $k-1$ folds are used as training data. In the next iteration, the next fold is used as the test data set while again the other $k-1$ folds are used as the training data.

Often times, the data is split prior to k-fold cross validation. When this is done, k-fold cross validation contains one part of this split (the majority of data points) which is then split into the training part and a validation part. The other part as a result of the initial split (the minority of data points) becomes the test data. For clarity, k-fold cross validation with prior split into a train and test set is illustrated in figure 4.

In making a selection for the exact parameters that are evaluated, grid search is often used. This method evaluates all combinations of a range of pre-specified parameters. Grid search and k-fold cross validation can be combined using GridSearchCV from the sci-kit learn package for python (Pedregosa et al. (2011)).
In performing the above method several properties of the data must be taken into consideration. For example, in obtaining a subset from the data that is used, this subset must be a good representation of the complete data. Often times this is done by dividing the data into subgroups based on a certain property or feature of the data which is called stratification. The exact proportions of the subgroups in the original data must then also be used in the subsets used for determining the parameters. Another consideration is randomizing the observations when determining the best parameters. This is called shuffling of the observations. Both stratification and shuffling can be implemented simultaneously using the StratifiedShuffleSplit from the scikit-learn package for python (Pedregosa et al. (2011)).

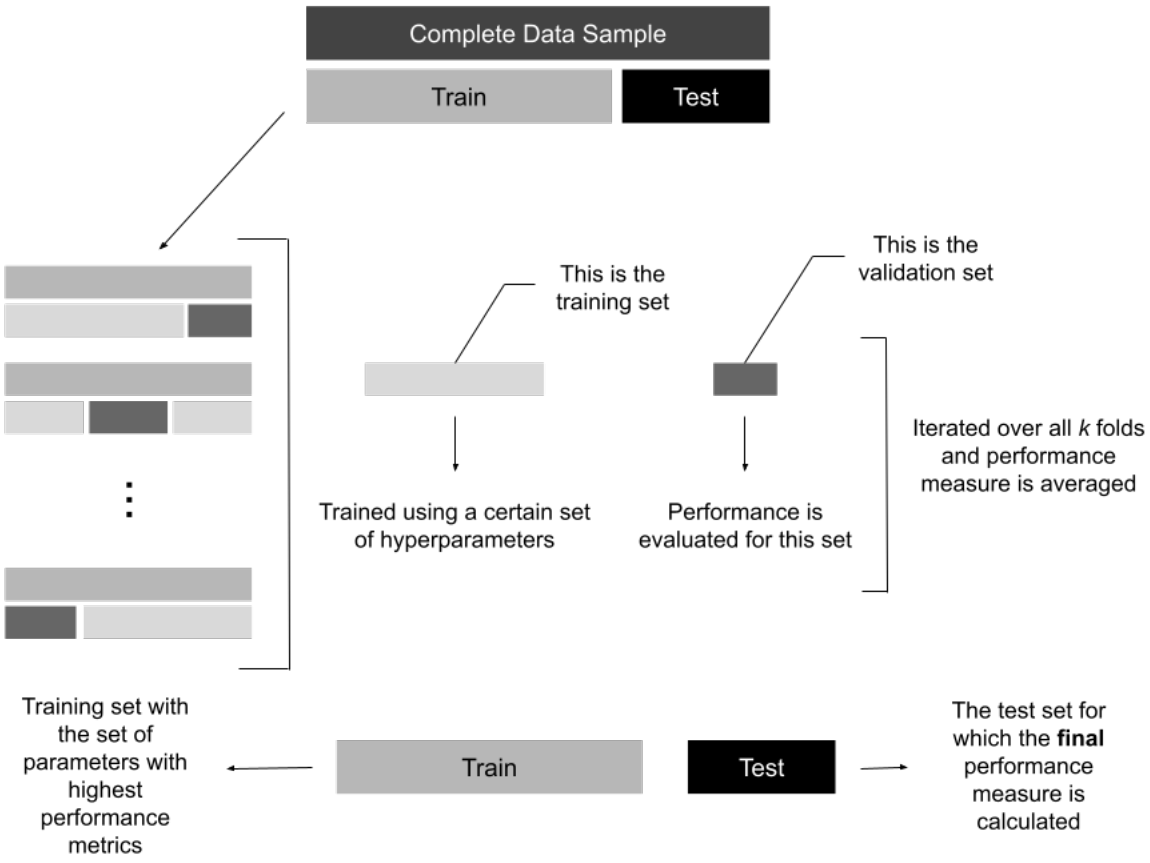


Figure 4: k-fold cross validation illustrated.

3.6 Performance Metrics

In order to evaluate the performance of the models used, some sort of metric or measure is required. This section describes the metrics used in this thesis. Several performance metrics offer advantages and disadvantages depending on the context of use. In the case of this research, performance metrics related to classification problems are required. One often cited disadvantage related to the performance measurement in classification methods is the class imbalance problem. This has to do with the fact that in case of imbalances, naive methods, meaning simply picking the majority class as a prediction, will seem to perform well in terms of accuracy. This means that accuracy will show a bias. As mentioned in the part on testing and training of this methodology, oversampling procedures are used on the training set. The performance is evaluated on the test set. Since accuracy will likely suffer from the aforementioned bias, the performance metrics used will have to take this into consideration. I choose to use precision, recall, and the F1 score which are explained next.

3.6.1 Confusion Matrix, precision, recall and F_1-Score

A confusion matrix is a matrix that describes the classifications made and not made by a model (see figure 5). Here, TP stands for true positive and indicates the situation in which the predicted value and the actual value are both 1. TN stands for true negative indicating the case where the prediction is 0 and the actual value is also 0. FN stands for false negative indicating that the model predicted a 0 while the actual value is a 1. FP stands for false positive and is the exact opposite of FN.

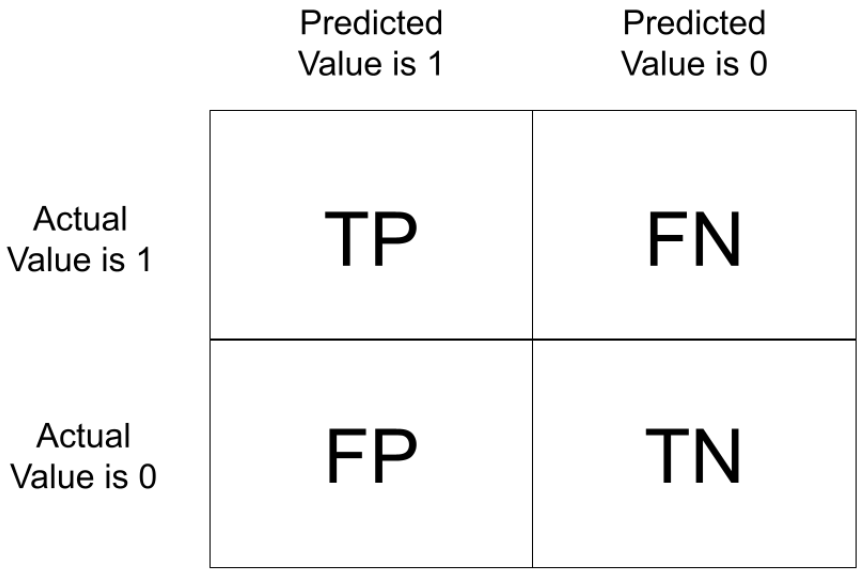


Figure 5: Confusion matrix.

From the confusion matrix, the chosen performance measures can be calculated given in equation 2, 3, and 4. To illustrate these metrics, an example is used. Say that we are predicting bankruptcies. In that case predicting that a company will not go bankrupt while it is actually going bankrupt (false negative) has far more severe consequences than predicting that a company will go bankrupt while the company actually proves to not be bankrupt (false positive). Precision then shows how many times a correct positive prediction is made relative to the number of favourable outcomes. A low precision thus indicates the situation in which the model predicts lots of false positives. Recall shows the number of correct predictions made relative to every positive prediction made. A low recall thus means that the model predicts lots of false negatives. What can be seen by this example is that having one type of prediction error can be more favourable as in the bankruptcy example higher recall would be more favourable, but

this would simultaneously mean that the model would have lower precision. The F_1-Score is essentially the harmonic mean of the precision and recall. This means that if the F_1-Score is low, either the precision or recall are low. When the F_1-Score is high then both precision and recall are high. For the sake of completeness, accuracy is defined in equation 5.

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

$$F_1\text{-}Score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{4}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5}$$

4 Data

In this section I describe the retrieval of the text and accounting data. I then discuss the selection of the sample which is analyzed in this thesis. A description is given of the reconciliation of the data to apply the research methods in this thesis. Furthermore, some issues and solutions relating to the data are discussed, which largely revolve around missing values, inconsistencies in scale and imbalancedness.

4.1 Text Data

The raw text data for each company is retrieved using the requests library for Python (*pub:55741* (n.d.)). These data are the MDA sections of the form 10-K's in a given year of the sample period (2010 to 2020). The raw text data is cleaned by removing symbols, URLs, and upper case letters. Also, stop words are removed using the Gensim package for Python (Řehůřek and Sojka (2010)). Stop words represent commonly used words that do not provide additional information such that the text data can be represented more sparsely without loss of information if these words are removed. Examples of stop words are 'or', 'who', 'as', 'from', 'him', 'each', 'the', etc. I tokenize the cleaned text into sentences after which I separate the sentence tokens into word tokens. In order to assign the right set of tokens[5] to the right company's MDA section, I attach labels to the tokens (also known as tagging). The labeled list of tokens is used as the input to the Doc2Vec model which is implemented using the Gensim package.

4.2 Accounting Data

For retrieving the accounting data I use the Eikon Datastream database by Thomson Reuters. The Central Index Key (CIK), which is the identifier for the text data, can be used to retrieve the accounting data. This procedure is done for the years 2010 to 2020. The accounting features are ratios based on both financial statements and market information. The ratios are calculated using a company's annual report and latest available market data for a given sample date. For instance, the P/E ratio for Apple in 2015 is based on Apple's share price on the last trading day of 2015 and its earnings per share as shown in its 2015 annual report. A comprehensive list of the features used is included in the appendix together with a description of these features. The appendix also shows a correlation matrix and summary statistics of the accounting features[6].

[5] A set of tokens is also known as a document.

[6] The text embeddings are excluded from the summary statistics and correlation matrix as these are not interpretable.

4.3 Sample Selection

The list of companies that is considered for this analysis is selected based on the availability of matching data, i.e. data for which both accounting information and annual reports for the text data is available. This means that all text data for available companies in the SEC's EDGAR database are extracted. Since this information is organized based on an identification number the accounting data for each observed company in the Thomson Reuter's Eikon Database can be easily obtained. Observations for which both text and accounting data is available are used in the research. To clarify, first the text data from the EDGAR database is retrieved through the SEC website using the indexation of the SEC which allows us to pull all form 10-K's that are available for all public companies subject to SEC oversight in a given year. The URLs for all these form 10-K's are saved in a dataframe together with their Central Index Key (CIK), a number that is used by the SEC to index the companies it oversees. This procedure is done for the years 2010 to 2020. Secondly, it is necessary that the text data has corresponding accounting data and that these data are available. These identifiers are used to retrieve the accounting data. The companies for which either modality of the data is not available are dropped.

4.4 Data Architecture

The accounting data and the text data have to be matched appropriately. Firstly, the accounting data and text embeddings are concatenated for each corresponding observation. The dependent variables of the data set are binary variables where $y_{i,t} = 1$ represents a company that is bankrupt and $y_{i,t} = 0$ represents a company that is not bankrupt in year t. As described in the methodology section, the goal is to make an one-step out forecast which allows us to implement a time-series approach to a static machine learning model. This means that the observations that will be used for predicting a default one year from now must still be active in the current year. Furthermore, in case a company has gone bankrupt, I leave out this company from the data in the following years. The reason for this is that otherwise an observation with the same identifier will emerge in the data multiple times making it difficult to merge the data of various years together. The final data frame is a long list of observations for which every company in every year presents one row and the final column indicates the status of the company a year from that specific year. Because I consider a one-step out prediction only, this approach prevents the need for tracking the same companies throughout multiple years and with that the need to create a panel data which would only add complexity to the models proposed.

4.5 Data Related Issues

The data requires several adjustments in order to make sure that it is compatible with the intended methodology. Some of these problems are related to the data being imbalanced, handling missing values in the data set, and the right scaling of the data. I describe how I solve these issues.

4.5.1 Missing values

A concern following the retrieval of the accounting data is the large number of missing values for some features which run up to 50% of the values missing. This is usually resolved by using KNN Imputation, which replaces the missing values in the data set by the average for that value in its K nearest neighbours of observations based on the euclidean distance. However, the use of a KNN Imputation is not a productive endeavour for variables with more than 50% of the values missing. Because this is the case for the feature "Liquidity Coverage Ratio" (see appendix for a full list of the features used), the column for this variable is dropped from the data set.

Another concern related to KNN imputation that is researched has to do with the potential disturbance of the data structure when using KNN imputation (Beretta and Santaniello (2016)). This means that the distribution of a feature could be altered following the implementation of KNN imputation as observations that were previously empty are now filled. Depending on these new values, the distribution of a feature could be altered from a normal distribution to a distribution that has fatter tails, such as a t-distribution. Using K nearest neighbours where $K > 1$ instead of just using the adjacent value as replacement provides lower errors in inferential statistics (Beretta and Santaniello (2016)). However, using the adjacent value does a better job of preserving the structure of the data. Comparing the distribution of the columns in the data before and after imputation indicates that this does not pose a problem and thus the standard KNN imputation using $K = 5$ is used in this thesis.

4.5.2 Inconsistent Scales

A requirement for the machine learning methods I apply in this thesis is that the data should be similar in scale. This is already the case for the text embeddings as the Doc2Vec algorithm provides an output that ranges between 1 and -1. The accounting data however could show some discrepancies as the range for this data type is wider. I therefore use min-max scaling to make sure that the results of the applied methods show reliable estimates. The min-max scaler is given in equation 6.

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{6}$$

4.5.3 Imbalanced Data

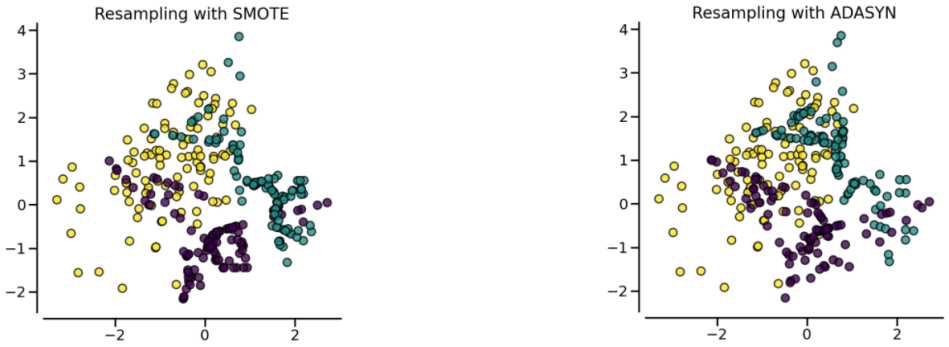
Another problem that occurs in the data set is the smaller proportion of bankrupt observations compared to the non-bankrupt observations throughout the sample period. This points at an imbalanced data set which presents problems in classification tasks. The problem is that a large imbalance in the data could result in a naive method, meaning simply predicting everything as non-bankrupt, to achieve a high accuracy. The classification algorithm hence becomes biased towards the majority class such that an adjustment of the data is needed. Two approaches can be taken to solve the imbalance in a data set for this which are under sampling and oversampling. Under sampling takes a subset of the majority class equal in amount to the number of samples in the minority class. This is seldom used because the overall amount of data is massively reduced, meaning that implementation of machine learning techniques will not produce accurate results. Over sampling is a technique which creates copies of the data in the minority class. In practice, combining under- and over sampling achieves the best results (Johnson and Khoshgoftaar (2019)).

Several algorithms exist for oversampling such as SMOTE and ADASYN. SMOTE uses a linear interpolation of the K nearest neighbors to the data points that belong to the minority class (Chawla et al. (2002)). This means that for the set of observations in the minority class A, the K nearest neighbours are found for each data point $x_i \in A$ using the euclidean distance between the x_i's. The proportion of the data points in the minority class to the data point in the majority class then determines how many new data points will have to be generated. The algorithm then selects a random neighbor out of the set of the K nearest neighbours to the data points in set A which are denoted B. The new data point then becomes $x_{\text{new}} = x + rand(0, 1)|x - x_k|$ (Chawla et al. (2002)). The pseudo code for SMOTE can be found in the appendix.

The ADASYN algorithm only differs slightly. Since ADASYN uses a probability distribution along the euclidean distance it offers a non linear interpolation. This resolves the issue around linear correlation between data points such that the applied methods are more robust (He et al. (2008)).

In this thesis, I choose a combination of under- and oversampling. I first, under sample the majority class to achieve a unbiased balance of bankruptcies in the data. I then oversample using

the ADASYN algorithm. The difference between SMOTE and ADASYN is illustrated in figure 6 where The green dots represent the data points belonging to the minority class and the yellow dots represent the data points belonging to the majority class. The purple points illustrate the newly generated data points using the SMOTE and ADASYN algorithms respectively. The difference is that the purple dots in the ADASYN algorithm are more concentrated around specific areas than is the case in SMOTE.



(a) SMOTE (b) ADASYN

Figure 6: Illustration of the difference between oversampling using the SMOTE algorithm and oversampling using the ADASYN algorithm (Lemaître et al. (2017))

5 Results

In this section the empirical findings of the applied methods are presented. Firstly, the performance metrics of the various methods are discussed. I highlight the implications of the results of the multimodal approach by contrasting its performance against that of the other methods used. Secondly, I show the optimal hyperparameters of the various methods with a discussion of the observed discrepancies. The reader is also reminded of the purpose of the different hyperparameters. The optimal hyperparameters of the DNN are elaborated upon together with a discussion on the training procedure.

5.1 Performance Results

The performance metrics that are used in this thesis are the recall, precision, and F_1-score. The accuracy, which is often used in the machine learning literature, will not be considered as it is not a reliable measure of performance in this setting because of the possibility of naive predictions. In the case of naive predictions the accuracy simply shows the number of non-bankruptcies in the test set. The accuracy is therefore not meaningful unless it is higher than the proportion of samples belonging to the majority class in the test data. A naive model would therefore only reach a maximum accuracy of $\frac{1330}{210+1330+20+10} = 0.847$. This number seems high and would normally indicate good performance but does not give an accurate idea of the algorithm's actual reliability. A naive prediction is mostly the result of a massive data imbalance. However, since I used undersampling in combination with ADASYN oversampling, the amount of bankrupt to non-bankrupt companies is not as skewed as in the original data.

The confusion matrix in table 3 shows the classifications and misclassifications made by the SVM model. The model does not predict any of the bankrupt cases, regardless of whether this is a wrong or a right prediction. This is seen in the confusion matrix by the 0 values for bankrupt predictions. The predictions made by the model hint at a naive prediction meaning our model simply assigns a majority class prediction for every observation. The performance measures are also given in the table and somewhat confirm the observation made in the confusion matrix.

The confusion matrix of the bankruptcy classifications using the accounting data shows that the majority of observations are classified correctly, meaning that SVM using accounting data performs well. This is supported by the performance measures in the table. The recall of 0.955 shows a good performance given the total amount of observations that are classified. A SVM using accounting data performs much better than the naive prediction seen when using only text data as the recall and precision of 0.955 and 0.913 indicate. In the case of bankruptcy

prediction, a false negative, meaning that the model predicts a company will not go bankrupt while it does, has far more severe consequences than a false positive, predicting a company will go bankrupt while it does not. The reason for this is that when a bankruptcy is not expected, the costs and risk a financial institution is exposed to are higher than when a client is denied a loan because he was falsely being predicted to be at high risk of bankruptcy. In these situations where the costs associated with false negatives are higher, recall is the key performance metric. The significantly better performance of the SVM using accounting data compared to the SVM using text data could be the result of several underlying reasons. Firstly, the text vectors as obtained using the Doc2Vec could be a result of a misspecified model. This means that the parameters used for Doc2Vec in this thesis are not suited for classification using SVM. Specifying the Doc2Vec model correctly for the data used in this thesis however, remains a difficult exercise. The reason for this is that the MDA sections in the form 10-K's of the different companies relate to a different business context. As an example, the vocabulary, jargon, synonyms and context of language used by a company in the healthcare industry will be very different from a company that operates in the oil and gas industry. Similarly, a company that operates internationally will have a different use of language from a company that solely operates in its domicile. Secondly, the vectors that result from the Doc2Vec algorithm have a relatively small range compared to the accounting ratios. The use of min-max scaling in the case of the accounting data could therefore lead to a transformation that shows larger deviations and would therefore be more easily separable than would be the case for the text data. Apart from issues related to the retrieval of the text data, the literature does show that bankruptcy classification based on accounting ratios performs better than bankruptcy classification based on text data (see Zhou et al. (2014) and Ravula (2021)).

Using both accounting and text data simultaneously in a SVM framework could also lead to interesting insights. The performance metrics for the SVM using accounting data shows a better performance than that of the SVM using text data. This begs the question of whether the combination of data can offer any improvements when SVM is applied. Of course, this would only work in case the combination of the modalities would allow for a pattern in the data that is better separable using a hyperplane. If this is not the case, the performance will most likely fall somewhere between the use of the SVM applied using the text data and the SVM applied using the accounting data separately as the SVM will pick up on the most pronounced pattern of data points that can be separated by a hyperplane.

The confusion matrix shows that the majority of the classifications are performed correctly. However, it is striking that the majority of misclassifications fall within the false positives,

meaning that the majority of misclassifications are for companies the model predicted not to go bankrupt while they actually did go bankrupt. From the performance of the SVM for using the text data and accounting data separately one would suppose that a similar pattern would persist. As could be derived from the confusion matrix, the recall, precision, and F1-score are lower than the ones found for the SVM applied to the accounting data separately.

Methodology	**Confusion Matrix**				**Performance Metric**		
		Predicted					
		Bankrupt	Non-Bankrupt	Total	Recall	Precision	F1 Score
SVM using Text Data	Bankrupt	0	220	220	0.000	0.000	0.000
	Non-Bankrupt	0	1350	1350			
	Total	0	1570	1570			
SVM using Accounting Data	Bankrupt	210	10	220	0.955	0.913	0.933
	Non-Bankrupt	20	1330	1350			
	Total	230	1340	1570			
SVM using Multimodal Data	Bankrupt	142	78	220	0.645	0.540	0.541
	Non-Bankrupt	121	1229	1350			
	Total	263	1307	1570			
Deep Learning using Accounting Data	Bankrupt	204	16	220	0.927	0.903	0.915
	Non-Bankrupt	22	1328	1350			
	Total	226	1344	1570			
Deep Learning using Text Data	Bankrupt	198	22	220	0.900	0.835	0.867
	Non-Bankrupt	39	1311	1350			
	Total	237	1333	1570			
Multimodal Method	Bankrupt	0	220	220	0.000	0.000	0.000
	Non-Bankrupt	0	1350	1350			
	Total	0	1570	1570			

Table 3: Confusion matrices and performance metric for the methods used. The confusion matrices show the predicted values horizontally and the actual values vertically as well as the column totals and row totals. The performance metrics column consists of the recall, precision and F_1 Score.

One would assume that the accounting data are distributed in such a way that data points are linearly separable meaning a one layer neural network would be sufficient. The reason for this is that accounting data are relative numbers, making them easy to compare. When one company performs worse compared to others this is expressed as an accounting ratio that lies farther away than that of the other companies observed. Additionally, given that a one-step out prediction is used, one would assume that the difference between bankrupt and non-bankrupt companies would be well-highlighted in the accounting ratios of these companies. The reason for this is because companies close to bankruptcy would show signs of unhealthy financials expressed through the accounting ratios. But this is not the case which could be the result of a number of possible characteristics of the applied methodology. One reason could be the additional non-linearity introduced because of the use of KNN Imputation for treating missing values. However, KNN imputation is a method that implements a form of interpolation. A number of data points close to a missing value are selected and the average of these neighbours replaces the missing value. Because of this, the newly formed data points always remain within

the current outer most data points. The effect this has is that disturbance of the data structure is negligible. Another reason for this non-linearity is the oversampling technique used. Since I used the ADASYN oversampling algorithm the newly generated data points belonging to the bankrupt class are non-linear transformations of each other which makes it difficult to apply models that make use of linear separability. However, this additional complexity is a better proxy for the distribution of accounting data in reality and therefore gives a better idea of the practical usefulness of the applied techniques.

The main method that is applied in this thesis is a DNN with feature reduction using a multimodal Boltzmann machine. The merged representation obtained through the multimodal Boltzmann machine is used as input for a DNN. This becomes especially apparent when looking at table 3. This table shows that the model makes a naive prediction. An explanation for this naive prediction is that the reduction in dimensionality of the multimodal data using the deep Boltzmann machine might reduce the input vectors in such a way that only explanatory factors are left that predict the majority class. This is especially problematic when the minority class is oversampled. One would assume that this would be difficult given the non-linearity in the data (as a consequence of oversamping using ADASYN). However, the reduction in dimensionality is based on the probability structure of the occurrence of different features for the various observations. When this is considered it becomes more apparent that the multimodal machine might not only reduce the dimension but also neglect observations of the minority class.

When comparing the various models I implemented, overall the methods using the accounting data seem to be performing best. A possible reason for this is the more robust structure of the accounting data which allows both SVM and DNN to detect patterns accurately. For the text data, the underlying reason for a failure to detect a pattern could have several underlying reasons.

Because of the bad performance of the text data compared to the accounting data, it is difficult to obtain additional information from the combination of this data. This was especially apparent in the case of the SVM implemented on combined modality (without the deep Boltzmann machine). There seemed to be no interaction effects between the features of the different modalities that could increase performance.

5.2 Hyperparameter Tuning

Table 4 shows the optimal hyperparameters for the various methods applied. The hyperparameter are found using grid search[7]. Optimality is defined as the highest performance metrics obtained for the concerning methodology. The support vector machines require tuning of the kernel, the C parameter, the degree of the polynomial, the tuning coefficient, and the gamma parameter. The kernel function determines the distance of the data points with respect to each other. Several kernel functions exist (see appendix) most notably the radial basis function kernel (Rbf) which uses a Taylor approximation to calculate distance between points up to an infinite dimension (Scholkopf et al. (1997)). The C parameter controls the penalty for data points that are misclassified by the soft margin. A larger C penalizes misclassified points more heavily and causes the hyperplane to separate data with a smaller error rate whereas a smaller C penalizes misclassification less heavily (see appendix). The degree parameter is related to the polynomial kernel and represents the dimension that the distance between data points are measured on. The tuning coefficient (Coefficient in the table) serves as an intercept in the polynomial and Rbf kernel. The gamma parameter controls the variance of the decision boundary where a larger gamma yields a model with high bias and low variance (i.e. more strict classifications) and vice versa.

For the SVM using text data the highest performance metrics are achieved with a Rbf kernel with a penalty C of 128, degree of 1, tuning coefficient of 0.6, and a variance parameter, gamma, of 2^{-7}. It is notable that the optimal kernel is not the polynomial kernel as is usually the case for NLP applications (Chang et al. (2010)). However, the large dimensionality of the text data used in this thesis requires a kernel which can facilitate distance measures in a higher dimensional space, which can be achieved by the Rbf kernel.

For the SVM using accounting data, table 4 shows optimality with a Rbf kernel, a penalty C of 10, degree of 1, tuning coefficient of 0.6, and a variance parameter gamma of 1.0. Compared to the SVM for the text data, the SVM for the accounting data imposes a lower C but a higher gamma. This means that the penalties for misclassified points are lower such that a less strict separation by the hyperplane is imposed. However, the effect of gamma weighs heavier than the effect of the C parameter since the behaviour of SVM is more sensitive to gamma (Goodfellow et al. (2016)). The model thus picks up on the complex boundary to separate data points. This specification of the model with a higher gamma is however more susceptible to overfitting.

The SVM using the multimodal data shows that the optimal hyperparameters are a Rbf kernel,

[7]For a complete overview of the hyperparameters used in GridSearchCV I refer the reader to the methodology section of this thesis.

a misclassification penalty C of 1, a degree of 1, tuning coefficient of 0.6, and a variance parameter gamma of 2^{-7}. Note that the objective of the SVM is to separate data points as well as possible according to some sort of boundary imposed by the specifications of the SVM model. The more similar the distribution of different data points are, the more similar the specifications for the SVM model will be. The hyperparameters for the SVM using multimodal data fall relatively close to the hyperparameters of the SVM using accounting data. However there are differences between the penalty parameter C and the noise parameter gamma for the SVM models. As the effect of gamma outweighs the effect of the C parameter, it becomes apparent that the complexity of the structure of the text data is dominant in the multimodal data. This is because the gamma parameter is closer to the gamma parameter of the SVM using text data.

Method	Hyperparameter	Value
SVM using Text Data	Kernel	Rbf
	C	128
	Degree	1
	Coefficient	0.6
	Gamma	2^{-7}
SVM using Accounting Data	Kernel	Rbf
	C	10
	Degree	1
	Coefficient	0.6
	Gamma	1.0
SVM using Multimodal Data	Kernel	Rbf
	C	1
	Degree	1
	Coefficient	0.6
	Gamma	2^{-7}
Deep Learning using Accounting Data	Number of Layers	4
	Neurons per Layer	30
	Activation Function	ReLU
	Loss Function	BCE
	Optimization Algorithm	ADAM
	Learning Rate	0.01
	Batch Size	64
Deep Learning using Text Data	Number of Layers	4
	Neurons per Layer	20
	Activation Function	ReLU
	Loss Function	BCE
	Optimization Algorithm	ADAM
	Learning Rate	0.01
	Batch Size	64
Multimodal Method	Number of Layers	5
	Neurons per Layer	64
	Activation Function	ReLU
	Loss Function	BCE
	Optimization Algorithm	ADAM
	Learning Rate	0.01
	Batch Size	256

Table 4: Optimal hyperparameters for the methods applied. Optimality is defined as the highest performance metrics obtained across different combinations of hyperparameters (i.e. recall, precision, F_1 score). The first five methdologies serve as a benchmark for the main method which is the multimodal method as described in the methodology.

The deep neural networks require tuning of the number of layers in the network, the neurons per layer, the activation function, the loss function, the optimization algorithm, the learning rate, and the batch size. The number of layers and the neurons per layer form the foundation of the architecture of the model and determine the complexity of the model. The activation function determines the values that are outputted by the nodes in the neural network and determine the proportion of non-linearity. The loss function determines the difference between the estimated output and the actual output (also called the loss). The optimization algorithm is the method used to adjust the weights in order to improve the loss. The learning rate determines the size of the steps in which this optmization is done. The batch size shows how many samples are used

to perform the calculations of the optimization algorithm[8].

Noticeable is that the DNN using the accounting data and the DNN using the text data have similar hyperparameters. For all models I make use of the ReLU activation function, BCE loss function, and ADAM optimization algorithm with a learning rate of 0.01. The ReLU activation function is used as it provides more efficient calculation of the activation and prevents a vanishing gradient (Talathi and Vartak (2015)). An often occurring disadvantage of this method however, is that it does not cope well in situations where activations are negative. Since the used data in this thesis is normalized using min-max scaling, all values that occur in the neural network are positive mitigating this disadvantage. The BCE loss function allows for well-behaved probabilistic outcomes of classification models. The use of the ADAM optimization algorithm is recommended for the majority of tasks in machine learning. Given that our methods make use of different modalities of data, the best results will be achieved using more generalizable optimization techniques (Ruder (2016)).

For the deep neural network using the accounting data, the optimal number of hidden layers is 4 and the number of neurons per layer is 30. For the deep neural network using the text data, the optimal number of hidden layers is 4 and the number of neurons per layer is 20. This difference has to do with the higher number of features in the text data. Because there are more features in the text data a more complex configuration is needed to predict bankruptcy.

The difference in batch size between the methods is attributable to the amount of samples needed for minimizing the gradient and is dependent on the sample size of the data used. The batch size is equal for deep learning using text and accounting data which has to do with the number of data points being approximately equal.

The multimodal method combines a DNN architecture with a multimodal Boltzmann machine. For the multimodal Boltzmann machine no hyperparameter tuning is required. However, the DNN does require hyperparameter tuning to find the minimum loss. The activation function, loss function, and optimization function follow from the configuration for the other deep neural networks. The higher number of batch size compared to the DNN using accounting data and text data has to do with the increased number of data points. Because of this increase, a larger batch size is needed for converging to find the minimum of the optimization function.

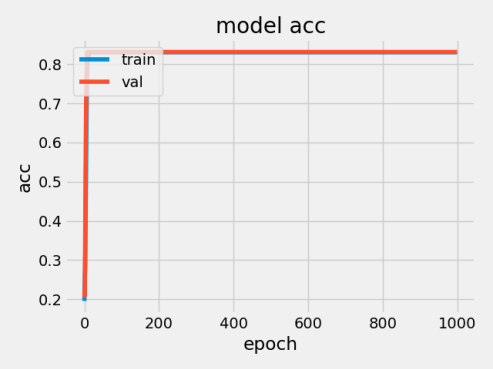
[8]For a technical explanation of the mechanics and hyperparameters of the artificial neural network I refer the reader to the appendix

5.3 Training Neural Networks

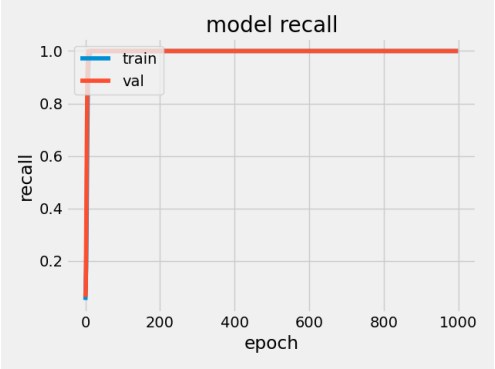
For the optimization of the hyperparameters for the deep neural network a more elaborate explanation is given. This has to do with the multiple epochs used in training which means that the data sample is used multiple times in training the DNN.

Figure 7 shows the performance metrics plotted against the epochs in training. One epoch represents the use of the entire data sample once. The various performance metrics as well as the loss of the model are plotted against the epoch to observe the improvement that is made by the training of the algorithm. The most striking graph is the one showing the precision during training, which seems to obtain a score of 1.000 around the first epoch. This is most likely due to the model having no false negatives during the first epoch as a result of its initialization. However, as the loss decreases the DNN seems to behave more like a naive predictor.

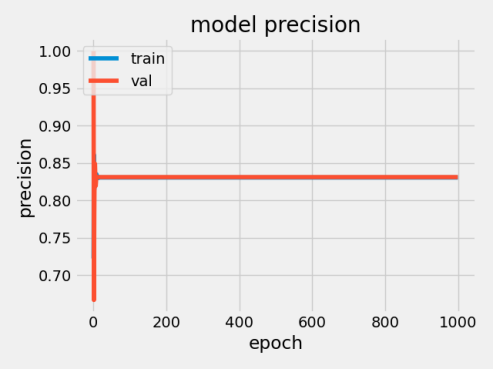
(a) Loss Function performance



(b) Accuracy performance



(c) Recall performance

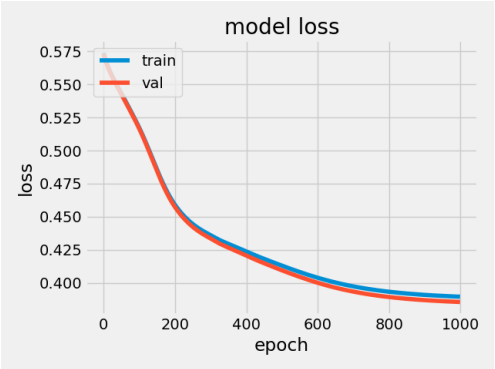


(d) Precision performance

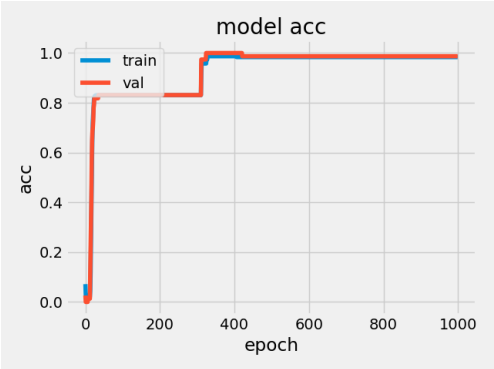
Figure 7: Performance of the deep neural nework using only text data.

Figure 8 shows the loss, accuracy, recall, and precision plotted against the epochs used in training. The improvement of the loss function seems to take place over a larger number of epochs. From the steepness of the graphs in figure 8b, 8c, and 8d however, a relatively fast convergence in accuracy, recall, and precision is observed. This means that the improvement in the loss of the model is not proportional to the marginal improvement in accuracy, recall, and precision.

Figure 9 shows the minimization of the loss function. Convergence seems to be achieved in a relatively smooth manner using a small number of epochs. Recall however that the multimodal method made naive predictions using the test data. The fast convergence that can be seen in figure 9a is therefore not an actual indication of quick convergence but rather the algorithm minimizing its loss function as a result of different initialization values for the optimization algorithm. This can be confirmed by graph 9b, 9c, and 9d since these graphs for the accuracy, recall, and precision instantly jump to their final values.

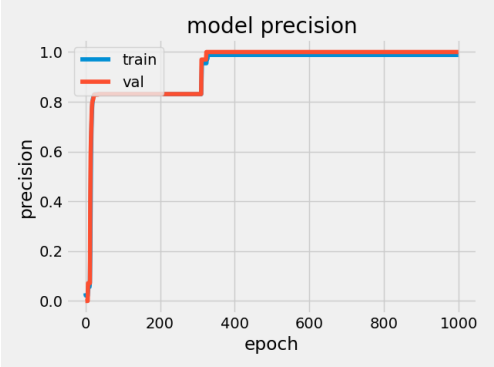


(a) Loss Function performance



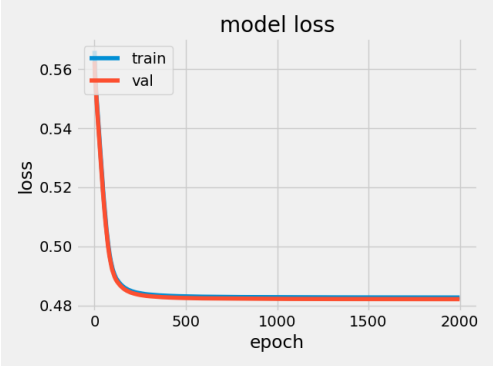
(b) Accuracy performance

(c) Recall performance

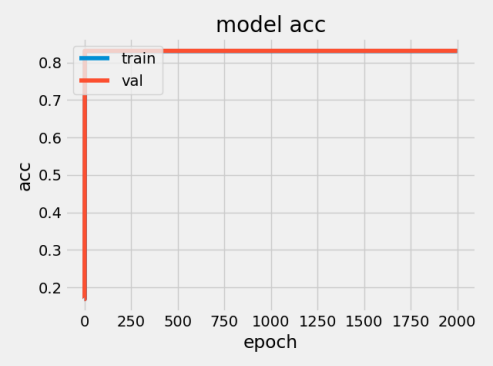


(d) Precision performance

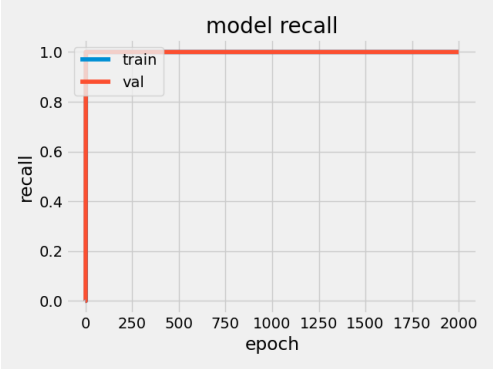
Figure 8: Training and validation performance of the deep neural nework using only accounting data plotted against epochs.



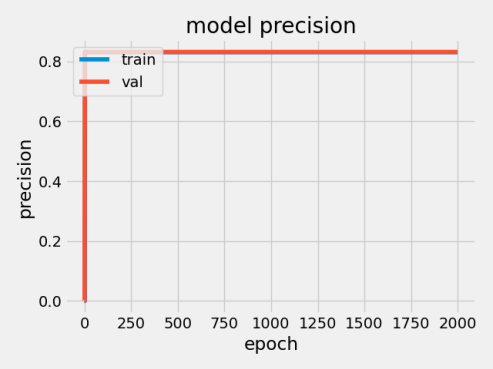
(a) Loss Function performance



(b) Accuracy performance



(c) Recall performance



(d) Precision performance

Figure 9: Performance of the deep neural network using only accounting data.

6 Conclusion & Discussion

In this paper, I investigate the application of multimodal machine learning methods to bankruptcy prediction. I predict the bankruptcy status of observed companies over the sample period 2010 to 2020 using accounting data retrieved from Thomson Reuters' Eikon Datastream and text data in the form 10K's retrieved from the SECs EDGAR database. The data is modified by using KNN imputation to resolve for missing data. Furthermore, normalization is applied in the form of min-max scaling to ensure equal contribution to the fitting of the applied models. To observe the competitiveness of multimodal machine learning methods, a comparison is made between SVM, multimodal Boltzmann machines, and neural networks.

Traditionally, the prediction of bankruptcy has been based on stochastic analysis (Merton (1974),Vasicek (1984), McQuown (1993), Kealhofer (1995)) and linear models applied to accounting ratios (Beaver (1966), Altman (1968)). The use of machine learning methods has slowly gained popularity due to its ability to cope with larger amounts of data available to derive more accurate bankruptcy predictions (Ohlson (1980), Odom and Sharda (1990)). Other methodologies have used text data in a machine learning context to predict bankruptcies (Cerchiello et al. (2017), Gentzkow et al. (2019)). Combining accounting and text data might help in detecting patterns that would otherwise be unobservable. Methodologies that use this so-called multimodal data have largely made use of separately trained models for which the individual scored are combined (Mai et al. (2019), Stevenson et al. (2021)). One issue with this approach is that the structure of the data and the dependencies within and across modalities are not taken into consideration. I improve on this by applying methods that do take these dependencies and the structure of the data into consideration.

I find that predictions using solely accounting data offer the best performance due to the linear separability of this data modality. Methodologies that use text data have performed worse than methodologies that use combined modalities and accounting data in terms of their respective performance metrics. This has to do with the non-linearity of this data which makes it hard for the applied methodologies to classify correctly. The performance of SVM and neural networks using multimodal data show a performance that lies between the performance of SVM and NN using accounting data and SVM and NN using text data. Additionally, I find that the SVM methodologies perform best for all types of data, highlighting their competitiveness in bankruptcy prediction. The model proposed in this thesis, using the multimodal Boltzmann machine had the worst performance. A reason for this could be that the reduction in dimensionality is based on the probability structure of the occurrence of different features for the various

observations. This means that the multimodal machine might not only reduce the dimension but also neglect the features that are most important for the observations of the minority class. Some limitations of this research can be highlighted to provide motivation for future research. One of these limitations has to do with the numerous specifications for the Doc2Vec algorithm. The number of specifications increases more rapidly when one considers that an additional hyperparameter tuning can be performed when the parameters of the applied methods are combined. For instance, the hyperparameter tuning for the Doc2vec and SVM combination will require tuning for all combinations of hyperparameters that belong to both Doc2Vec and SVM which is simply infeasible.

Another limitation has to do with overfitting. The inclusion of additional regressors can always improve in-sample performance to a certain extent but this will be at the cost of the accuracy of predictions. However, it would be interesting to apply methodology such as general-to-specific regressions to determine the right features out of a large selection.

In researching inferences from multi-modal data to predict bankruptcies, possibilities in model specifications and methodologies are numerous. Some of the limitations in this research mainly have to do with the dimensionality reduction technique used. In this thesis I chose to use DBMs given the literature available on the methodology. Furthermore, the use of other methodologies such as multi-modal autoencoders do not have any relevant literature hinting at their applicability for a variety of data but could nonetheless be interesting to consider for future research.

References

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

Assis, C. A., Pereira, A. C., Carrano, E. G., Ramos, R., & Dias, W. (2018). Restricted boltzmann machines for the prediction of trends in financial time series. In *2018 international joint conference on neural networks (ijcnn)* (pp. 1–8).

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

Barboza, F., Kimura, H., & Altman, E. (2017). Machine learning models and bankruptcy prediction. *Expert Systems with Applications*, *83*, 405–417.

Beaver, W. H. (1966). Financial ratios as predictors of failure. *Journal of accounting research*, 71–111.

Bellovary, J. L., Giacomino, D. E., & Akers, M. D. (2007). A review of bankruptcy prediction studies: 1930 to present. *Journal of Financial education*, 1–42.

Beretta, L., & Santaniello, A. (2016). Nearest neighbor imputation algorithms: a critical evaluation. *BMC medical informatics and decision making*, *16*(3), 197–208.

Carreira-Perpinan, M. A., & Hinton, G. E. (2005). On contrastive divergence learning. In *Aistats* (Vol. 10, pp. 33–40).

Cerchiello, P., Giudici, P., & Nicola, G. (2017). Twitter data models for bank risk contagion. *Neurocomputing*, *264*, 50–56.

Chang, Y.-W., Hsieh, C.-J., Chang, K.-W., Ringgaard, M., & Lin, C.-J. (2010). Training and testing low-degree polynomial data mappings via linear svm. *Journal of Machine Learning Research*, *11*(4).

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

Cho, K. H., Raiko, T., & Ilin, A. (2013). Gaussian-bernoulli deep boltzmann machine. In *The 2013 international joint conference on neural networks (ijcnn)* (pp. 1–7).

Erdogan, B. E. (2013). Prediction of bankruptcy using support vector machines: an application to bank bankruptcy. *Journal of Statistical Computation and Simulation*, *83*(8), 1543–1555.

Gentzkow, M., Kelly, B., & Taddy, M. (2019). Text as data. *Journal of Economic Literature*, *57*(3), 535–74.

Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). *Deep learning* (Vol. 1) (No. 2). MIT press Cambridge.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: data mining, inference, and prediction.* Springer Science & Business Media.

He, H., Bai, Y., Garcia, E. A., & Li, S. (2008). Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In *2008 ieee international joint conference on neural networks (ieee world congress on computational intelligence)* (pp. 1322–1328).

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural computation*, *14*(8), 1771–1800.

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

Hubbard, R., & Vetter, D. E. (1996). An empirical comparison of published replication research in accounting, economics, finance, management, and marketing. *Journal of Business Research*, *35*(2), 153–164.

Johnson, J. M., & Khoshgoftaar, T. M. (2019). Survey on deep learning with class imbalance. *Journal of Big Data*, *6*(1), 1–54.

Kealhofer, S. (1995). *Portfolio management of default risk, proprietary documentation.* KMV Corporation, San Francisco.

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

Le, Q., & Mikolov, T. (2014). Distributed representations of sentences and documents. In *International conference on machine learning* (pp. 1188–1196).

Lemaître, G., Nogueira, F., & Aridas, C. K. (2017). Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, *18*(17), 1-5. Retrieved from `http://jmlr.org/papers/v18/16-365.html`

Levy, O., & Goldberg, Y. (2014). Dependency-based word embeddings. In *Proceedings of the 52nd annual meeting of the association for computational linguistics (volume 2: Short papers)* (pp. 302–308).

Liu, P., Qiu, X., & Huang, X. (2016). Recurrent neural network for text classification with multi-task learning. *arXiv preprint arXiv:1605.05101*.

Mai, F., Tian, S., Lee, C., & Ma, L. (2019). Deep learning models for bankruptcy prediction using textual disclosures. *European journal of operational research*, *274*(2), 743–758.

McQuown, J. A. (1993). *A comment on market vs. accounting based measures of default risk* (Tech. Rep.). mimeo, KMV Corporation.

Merton, R. C. (1974). On the pricing of corporate debt: The risk structure of interest rates. *The Journal of finance*, *29*(2), 449–470.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Min, J. H., & Lee, Y.-C. (2005). Bankruptcy prediction using support vector machine with optimal choice of kernel function parameters. *Expert systems with applications*, *28*(4), 603–614.

Odom, M. D., & Sharda, R. (1990). A neural network model for bankruptcy prediction. In *1990 ijcnn international joint conference on neural networks* (pp. 163–168).

Ohlson, J. A. (1980). Financial ratios and the probabilistic prediction of bankruptcy. *Journal of accounting research*, 109–131.

Osborn, T. R. (1990). Fast teaching of boltzmann machines with local inhibition. In *International neural network conference* (pp. 785–785).

Patel, K., & Bhattacharyya, P. (2017). Towards lower bounds on number of dimensions for word embeddings. In *Proceedings of the eighth international joint conference on natural language processing (volume 2: Short papers)* (pp. 31–36).

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Ravula, S. (2021). Bankruptcy prediction using disclosure text features. *arXiv preprint arXiv:2101.00719*.

Řehůřek, R., & Sojka, P. (2010, May 22). Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks* (pp. 45–50). Valletta, Malta: ELRA. (`http://is.muni.cz/publication/884893/en`)

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Scholkopf, B., Sung, K.-K., Burges, C. J., Girosi, F., Niyogi, P., Poggio, T., & Vapnik, V. (1997). Comparing support vector machines with gaussian kernels to radial basis function classifiers. *IEEE transactions on Signal Processing*, *45*(11), 2758–2765.

Shin, K.-S., Lee, T. S., & Kim, H.-j. (2005). An application of support vector machines in bankruptcy prediction model. *Expert systems with applications*, *28*(1), 127–135.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning*

research, *15*(1), 1929–1958.

Srivastava, N., Salakhutdinov, R., et al. (2012). Multimodal learning with deep boltzmann machines. In *Nips* (Vol. 1, p. 2).

Stevenson, M., Mues, C., & Bravo, C. (2021). The value of text for small business default prediction: A deep learning approach. *European Journal of Operational Research*.

Sun, A., Lim, E.-P., & Liu, Y. (2009). On strategies for imbalanced text classification using svm: A comparative study. *Decision Support Systems*, *48*(1), 191–201.

Sun, D., Wang, M., & Li, A. (2018). A multimodal deep neural network for human breast cancer prognosis prediction by integrating multi-dimensional data. *IEEE/ACM transactions on computational biology and bioinformatics*, *16*(3), 841–850.

Talathi, S. S., & Vartak, A. (2015). Improving performance of recurrent neural network with relu nonlinearity. *arXiv preprint arXiv:1511.03771*.

Tam, K. Y., & Kiang, M. Y. (1992). Managerial applications of neural networks: the case of bank failure predictions. *Management science*, *38*(7), 926–947.

Vasicek, O. A. (1984). *Credit valuation.* KMV Corporation, March.

Welling, M., Rosen-Zvi, M., & Hinton, G. E. (2004). Exponential family harmoniums with an application to information retrieval. In *Nips* (Vol. 4, pp. 1481–1488).

Zhao, Y., Shrivastava, A. K., & Tsui, K. L. (2016). Imbalanced classification by learning hidden data structure. *IIE Transactions*, *48*(7), 614–628.

Zhou, L., Lai, K. K., & Yen, J. (2014). Bankruptcy prediction using svm models with a new approach to combine features selection and parameter optimisation. *International Journal of Systems Science*, *45*(3), 241–253.

A Descriptive Statistics

A.1 Overview of Features

Identifier	Feature	Description
0	Dividend Yield	The amount of dividend a company pays out as a percentage of the stock price.
1	P/E Ratio	Ratio of the price of the stock of a company divided by the earnings a company made during the calendar year.
2	P/B Ratio	Ratio of the price of the stock of a company divided by the book value per share.
3	Current EV / 12M Forward EBIT	Multiple of the current enterprise value over the 12 month forward Earning Before Interest and Tax. This is a forecasted measure based on analyst consensus.
4	Current EV / 12M Forward EBITDA	Multiple of the current enterprise value over the 12 month forward Earning Before Interest, Tax, Depreciation and Amortization. This is a forecasted measure based on analyst consensus.
5	Current EV / 12M Forward Sales	Multiple of the current enterprise value over the 12 month forward Sales. This is a forecasted measure based on analyst consensus.
6	12M Forward Net Debt / EV	Ratio of the 12 month forward net debt to enterprise value. Net debt is defined as total debt - cash. This is a forecasted measure based on analyst consensus.
7	Price / Cash Flow Ratio	Ratio of the price of the stock of a company divided by the book value per share.
8	P/S Ratio	Ratio of the price of the stock of a company divided by the book value per share.
9	Gross Profit Margin	Gross profit divided by the revenue made in a year.
10	Operating Profit Margin	Operating profit divided by the revenue made in a year.
11	ROA	Return on the total assets on the balance sheet.
12	ROIC	Return on invested capital.
13	ROE	Return on total equity.
14	D/V	Debt to total value of the firm.
15	E/V	Equity to total value of the firm.
16	Total Asset Turnover	Ratio of total sales to the average of the assets at the beginning of the year and the end of the year.
17	Accounts Payable / Sales	Ratio of accounts payable (found on the balance sheet) divided by sales.
18	Inventory Turnover	Illustrates how many times a company's inventory has been sold and replenished during a specific time period. Ratio of COGS to average value of inventory.
19	Net Sales / Working Capital	Ratio between sales adjusted for allowances, returns, and discounts divided by the working capital.
20	Quick Ratio	Ratio of current assets - inventory - prepaid expenses divided by current liabilities.
21	Current Ratio	Ratio of current assets divided by current liabilities.
22	Long Term D/V	Long term debt to total value of the firm. This is a forecasted measure based on analyst consensus.
23	12M Forward Net Debt / EBITDA	Ratio of net debt divided by the Earnings before Interest, Tax, Depreciation, and Amortization. This is a forecasted measure based on analyst consensus.

A.2 Summary Statistics

	Mean	Standard Deviation
Dividend Yield	3.987	245.100
P/E Ratio	60.154	385.240
P/B Ratio	5.272	313.588
Current EV / 12M Forward EBIT	18.479	890.272
Current EV / 12M Forward EBITDA	-7.411e+05	1.481e+07
Current EV / 12M Forward Sales	46.146	3254.287
12M Forward Net Debt / EV	-0.139	50.028
Price / Cash Flow Ratio	7.298745	535.993
P/S Ratio	51.008	985.470
Gross Profit Margin	-0.341	1232.668
Operating Profit Margin	-704.207	13352.900
ROA	-3.545	1079.722
ROIC	-10.371	488.067
ROE	-128.912	1.123e+04
D/V	32.668	515.784
E/V	57.839	193.693
Total Asset Turnover	0.779	1.034
Accounts Payable / Sales	79.227	1601.435
Inventory Turnover	28.938	524.713
Net Sales / Working Capital	-5.291	1659.814
Quick Ratio	2.677	29.222
Current Ratio	3.423	29.260
Long Term D/V	33.854	229.029
12M Forward Net Debt / EBITDA	2.025	88.441

A.3 Distribution of Features

	min	25%	50%	75%	max
Dividend Yield	0.000	0.000	0.600	2.622	4.726e+04
P/E Ratio	0.000	15.200	22.680	42.558	3.830e+04
P/B Ratio	-4140.350	1.150	2.120	3.610	6.300e+04
Current EV / 12M Forward EBIT	-4.616e+04	7.550	12.286	18.769	1.535e+05
Current EV / 12M Forward EBITDA	-1.282e+09	5.416	8.241	11.628	7.817e+03
Current EV / 12M Forward Sales	-7.985e+03	1.217	2.792	6.671	6.214e+05
12M Forward Net Debt / EV	-9.723e+04	0.091	0.248	0.418	499.615
Price / Cash Flow Ratio	-7.578e+05	2.470	8.990	14.545	4.650e+04
P/S Ratio	-657.970	0.970	2.310	6.344	1.259e+05
Gross Profit Margin	-1.176+e05	21.025	35.500	52.880	107.500
Operating Profit Margin	-1.493e+06	-25.665	5.620	16.092	1.999e+04
ROA	-4.334e+04	-8.311	1.670	6.556	1.542e+05
ROIC	-5.910e+04	-6.410	4.000	8.900	3.120e+05
ROE	-2.245e+06	-21.297	3.700	12.310	4.940e+04
D/V	-9.793e+04	12.300	35.074	51.392	1.715e+04
E/V	-2.229e+04	49.760	64.300	89.435	401.240
Total Asset Turnover	-0.560	0.230	0.680	1.000	138.590
Accounts Payable / Sales	-5.126	4.933	8.128	15.363	2.669e+05
Inventory Turnover	-5.240	4.120	8.300	21.256	9.855e+04
Net Sales / Working Capital	-3.330e+05	0.102	2.741	7.010	3.007e+04
Quick Ratio	-0.080	0.890	1.500	2.473	4.979e+03
Current Ratio	0.000	1.380	2.244	3.320	4.980e+03
Long Term D/V	-1.171e+04	7.170	30.720	46.320	3.4130e+04
12M Forward Net Debt / EBITDA	-1.419e+04	0.716	1.916	3.436	4.240e+03

B Technical Explanation Multimodal Boltzmann Machine

In order to develop an understanding of the multimodal DBMs in Srivastava et al. (2012), an understanding of regular Boltzmann machines and RBMs is required as the multimodal DBM methodology is a generalization of the RBM methodology. The RBM methodology on its turn is an extension of the regular Boltzmann Machine. The differences between the architecture for the different Boltzmann machines can be seen in figure 10.

Boltzmann Machines are a type of unsupervised network that generate a representation of the data that is inputted to the visible layers (or input nodes) of the network. The input is a vector $\mathbf{v}$ consisting of binary elements $[0, 1]^D$ where D denotes the elements of vector $\mathbf{v}$. The model makes use of the Boltzmann distribution which is traditionally used in statistical mechanics to describe the state of a system as a function of its energy and temperature. The probability distribution function of the Boltzmann distribution is given in equation 7 where M denotes the number of states i that the system can be in, ε is the energy of a certain state i, T is the temperature of the system and k represents a parameter known as the Boltzmann constant. The optimal state of the model in the machine learning application is that in which the energy is the lowest. The energy function ε_i for the model is given in equation 8 where $w_{i,j}$ is the weight between node i, and j, s_i represents the state of node i, and θ_i is a bias term for node i. In the Boltzmann machine the temperature T serves as a hyper parameter. The training of the Boltzmann machine intends to approximate the distribution of the data in the input layers, which is denoted $P_{input}(V)$, by the distribution that results from the Boltzmann machine, which is denoted $P_{hidden}(V)$. The loss function to determine the similarity between $P_{input}(V)$ and $P_{hidden}(V)$ is the Kullback-Leibler divergence given in equation 9. Important to understand is that the training of Boltzmann Machines is separated into two so-called phases. One is the positive phase in which the states of the visible units are determined by a binary state vector sampled from data in the training set. The other is the negative phase, where the state of the input vector is random. The partial derivative of the Kullback-Leibler divergence with respect to the weight is given in equation 10. Here p_{ij}^{+} denotes the probability that nodes i and j are both in state 1 given the occurrence of an equilibrium in the positive phase and p_{ij}^{-} denotes the probability that nodes i and j are both in state 1 given the occurrence of an equilibrium in the negative phase. α represents a learning rate. During training the weights are adjusted in such a manner to minimize the Kullback-Leibler divergence.

$$p_i = \frac{e^{-\varepsilon_i/kT}}{\sum_{j=1}^{M} e^{-\varepsilon_j/kT}} \tag{7}$$

$$\varepsilon = -\left(\sum_{i<j} w_{ij} s_i s_j + \sum_i \theta_i s_i\right) \tag{8}$$

$$G = \sum_v P_{input}(v) \ln\left(\frac{P_{input}(v)}{P_{hidden}(v)}\right) \tag{9}$$

$$\frac{\partial G}{\partial w_{ij}} = -\frac{1}{\alpha}\left[p_{ij}^{+} - p_{ij}^{-}\right] \tag{10}$$

Boltzmann machines however, suffer from certain limitations making them difficult to use in practice. One such limitation is that the complexity of the model's structure grows exponentially as the number of input nodes grows. A solution for this limitation is presented by RBMs. RBMs are essentially Boltzmann Machines in which the interconnected structure is limited to adjacent layers meaning that there are no connections between nodes in the same layer (see figure 10). A change to the probability distribution as given in equation 7 is required in the case of this altered structure. The probabilities as given in equation 7 are namely based on the interdependence of all nodes in the network whereas in the case of RBMs an independence exists. These altered probabilities are shown in equation 11 and equation 12.

$$P(v \mid h) = \prod_{i=1}^{m} P\left(v_i \mid h\right) \tag{11}$$

$$P(h \mid v) = \prod_{j=1}^{n} P\left(h_j \mid v\right) \tag{12}$$

RBMs use a training algorithm known as contrastive divergence (CD) (Carreira-Perpinan and Hinton (2005)). The goal is to maximize the likelihood over a set of input vectors which is denoted V with respect to the weights $w_{i,j}$ in the graph (see equation 14). The algorithm works as follows. Given a sample for the input $\mathbf{v} \in [0,1]^D$ an activation function is used, such as the sigmoid function, to calculate a set of probabilities for the hidden layer (for more on activation functions see section 4.3). The next step is to generate a representation for the input vector $\mathbf{v}$ denoted by $\hat{\mathbf{v}}$ by sampling from the hidden layer $\mathbf{h}$. Additionally, from this new vector $\hat{\mathbf{v}}$ the vector of probabilities $\hat{\mathbf{h}}$ is generated. This gives us a total of four vectors to consider namely $\mathbf{v}$, $\mathbf{h}$, $\hat{\mathbf{v}}$, and $\hat{\mathbf{h}}$. Using the finding shown for Boltzmann machines in equation 10 and generalizing

this to the case for the RBM structure, this results in equation 13. This equation is thus used to update weights of the RBM where α is the learning rate.

$$\Delta W = \alpha \left(v h^\top - \hat{v}' \hat{h}^\top \right) \tag{13}$$

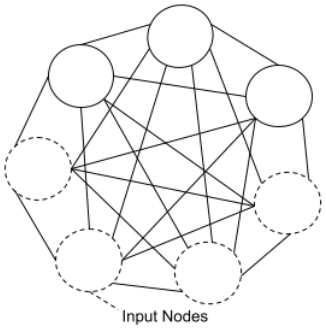
A problem occurs when considering the amount of elements in the vectors $\mathbf{v}$, $\mathbf{h}$, $\hat{\mathbf{v}}$, and $\hat{\mathbf{h}}$. The sampling procedure for generating $\hat{\mathbf{v}}$ and $\hat{\mathbf{h}}$ for every iteration in the algorithm and the calculation of the outer products $v' h^\top$ and $\hat{v}' \hat{h}^\top$ becomes a computationally heavy endeavour. In fact, the amount of samples increases exponentially as the number of input elements increases. A solution to this problem is offered by using Gibbs sampling to sample from the outer products $v' h^\top$ and $\hat{v}' \hat{h}^\top$ (the pseudocode for the Gibbs sampler can be found in the appendix).

$$\arg\max_W \prod_{v \in V} P(v) \tag{14}$$

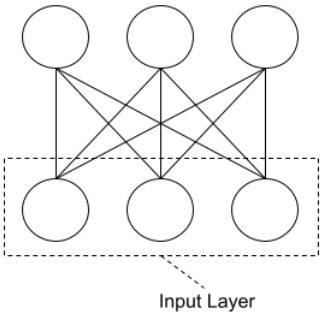
An extension of the standard RBMs is an algorithm which allows for inputs that have a continuous distribution such that now the input vector $\mathbf{v} \in \mathbb{R}^D$. This algorithm is known as the Gaussian Bernoulli Restricted Boltzmann Machine which is often cited in research that applies feature extraction to image data for example (for more on this methodology see Cho et al. (2013)).

Some data types require a more complex configuration than the RBM structure to obtain a more accurate representation of the input data. This could be the case for the structure as found in data related to speech recognition for instance. Additional complexity can be offered by Deep Boltzmann Machines (DBMs) (see figure 10). The input vector remains the same as the earlier defined $\mathbf{v} \in [0,1]^D$ but the hidden layers and the probabilities associated with $\mathbf{v}$ are different. These probabilities are now given by equation 15. The hidden layers can be written as $\boldsymbol{h}^{(1)} \in \{0,1\}^{F_1}, \boldsymbol{h}^{(2)} \in \{0,1\}^{F_2}, \ldots, \boldsymbol{h}^{(L)} \in \{0,1\}^{F_L}$ where F_L stands for the number of nodes in a specific hidden layer $\mathbf{h}$. In equation 15, W_{ij}^L are the weights associated with the network in layer L, Z is the sum of $e^{\sum_{ij} W_{ij}^{(1)} \nu_i h_j^{(1)} + \sum_{jl} W_{jl}^{(2)} h_j^{(1)} h_l^{(2)} + \sum_{lm} W_{lm}^{(3)} h_l^{(2)} h_m^{(3)}}$ for all possible layers and nodes in the network. The structure of a DBM can be seen as multiple RBMs stacked on top of each other.

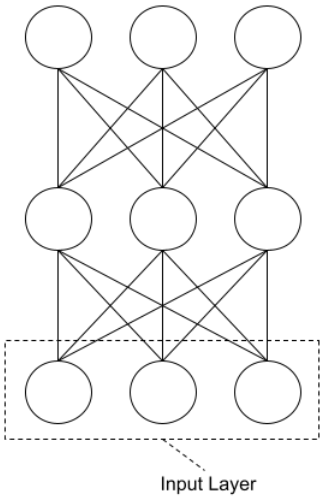
$$p(\boldsymbol{\nu}) = \frac{1}{Z} \sum_h e^{\sum_{ij} W_{ij}^{(1)} \nu_i h_j^{(1)} + \sum_{jl} W_{jl}^{(2)} h_j^{(1)} h_l^{(2)} + \sum_{lm} W_{lm}^{(3)} h_l^{(2)} h_m^{(3)}} \tag{15}$$



(a) Boltzmann Machine



(b) RBM



(c) DBM

Figure 10: Illustration of the difference between a Boltzmann Machines, a RBM, and a DBM

In the same way that the Gaussian RBM are used to make sure that a real valued vector $\mathbf{v} \in \mathbb{R}^D$ can be used as input vector for an RBM, this can also be implemented for a DBM.

For learning multimodal data representations, Srivastava et al. (2012) suggest a Deep Boltzmann Machine model that can successfully use a large amount of unlabeled data. The suggested model combines numerous data modalities into a uniform representation that captures useful information for classification. This is demonstrated using an experiment in order to find out how multimodal DBMs perform in classification tasks. Finding the right connections between different data sources to get a uniform representation out of a combination of data modalities poses a difficult problem as the correlational and probabilistic structure of two different modalities can be very different. The task becomes even more difficult when the data has missing values or is very noisy.

Previous attempts for multimodal classification include Huiskes et al. (2010) who show an improvement in performance of SVMs and Linear Discriminant Analysis for image classification when text is added to images. Guillaumin et al. (2010) use a multiple kernel learning approach

to SVM to show the improvement in object identification tasks when a text modality is added to images.

As is shown previously the input data for Boltzmann Machines, RBMs and DBMs is traditionally limited to binary vectors. In the same way that Gaussian RBMs offer a solution to use real valued vectors as input for RBMs, this adjustment can also be made for multimodal DBMs.

C Deep Neural Network Technical Explanation

The architecture of a DNN is described by the number of hidden layers the network has and the number of nodes each hidden layer has. In figure 3 the distinction can be seen between the input layer, the hidden layers, and the output layer. In the case of this thesis, the input layer equates to the reduced output as generated by the Deep Boltzmann Machine (see figure 3). The goal of the neural network is to approximate a function $f(x)$ that maps the reduced input vectors to the bankruptcy indicator. This was possible due to the Universal Approximation Theorem (Hornik et al. (1989)) with any level of accuracy as long as the input space has finite dimensions. Non-linear relationships between the input vector and the output of the neural network can be introduced in this layer architecture through activation functions. These functions offer a way to transform the linear output of a node into a non-linear output.

An activation function determines the output of a node in a specific layer in a neural network. Some of the most well-known activation functions include the sigmoid function (equation 16), ReLU (equation 17) and softmax (equation 18). In these function a represent the input to a layer, i.e. the product of the output of the previous nodes and their corresponding weights. Activation functions have properties that make them useful for specific tasks and specific network architectures. For instance, in the case of binary classification problems with two classes, the sigmoid activation function is often used. For classification problems that have multiple classes as its output, the softmax function is used. The softmax function outputs probabilities that sum up to 1 given a set of values such that the output of a node can be interpreted as a probability. The ReLu function has the ability to promote sparsity as it pushes certain values to zero which improves the acceleration of learning. ReLu is also computationally less expensive since it only seeks the maximum between two values to produce an output from a node. The various activation functions also have disadvantages. For instance, ReLu could fall in a situation where certain nodes are dropping out of the network unintendedly because of a negative a. Vanishing gradient problems arise in the case of the sigmoid function for DNNs. The sigmoid activation function namely transforms a relatively large input space to a input space between 0 and 1. Any large or small change made to the input data is not reflected proportionally by the sigmoid function. When evaluating the gradient, this means that the gradient gets close to 0. Taking the chain rule (see below) into consideration, this small gradient is multiplied many times depending on the depth of the network. This causes inaccuracy in training. For this reason, I use the ReLu activation function in the hidden layers but a sigmoid activation function in the last layer of the network as the output is required to be a 0 or 1.

$$S(a) = \frac{1}{1+e^{-a}} \qquad (16) \qquad f(x) = max(0,a) \qquad (17) \qquad \sigma(\mathbf{a})_i = \frac{e^{a_i}}{\sum_{j=1}^{K} e^{a_j}} \qquad (18)$$

As mentioned, to find the intended approximation for our objective function, the neural network has to be fitted to the data set. This is done using training which describes the adjustments of the weights in a neural network such that classifications can be made for new data points up to a certain accuracy. This adjustment of the weights is done using an algorithm known as backward propagation of errors, or backprop for short. Backprop uses the gradient with respect to the weights in the neural network given the architecture of a neural network and an error function. The architecture of the network is taken into consideration because the gradient will have to be calculated with respect to the amount of layers and nodes. To illustrate this, the case of a neural network with one hidden layers is used. In equation 19, x_{ij} denotes the input data where $i \in N$ denotes the observation and $j \in P$ denotes the features, $\beta_{mj}^{(1)}$ denotes the weights for the features $j \in P$ and nodes in the adjacent layer $m \in M$. $h(\cdot)$ denotes an activation function, and $S(\cdot)$ denotes the sigmoid activation function given in equation 16. In equation 20, which shows the gradient of the loss function with respect to the weights, R denotes a certain loss function. This result is derived using the chain rule.

$$\begin{aligned} a_{mi}^{(1)} &= \sum_{j=0}^{p} x_{ij}\beta_{mj}^{(1)} \\ z_{mi} &= h\left(a_{mi}^{(1)}\right) \\ a_{ki}^{(2)} &= \sum_{m=0}^{M} z_{mi}\beta_{km}^{(2)} \\ \widehat{y}_{ki} &= S\left(a_{ki}^{(2)}\right) \end{aligned} \qquad (19)$$

$$\frac{\partial R_i}{\partial \beta_{km}^{(2)}} = \frac{\partial R_i}{\partial \widehat{y}_{ki}} \cdot \frac{\partial \widehat{y}_{ki}}{\partial a_{ki}^{(2)}} \cdot \frac{\partial a_{ki}^{(2)}}{\partial \beta_{km}^{(2)}} \qquad (20)$$

Suppose the loss function R_i is given by the squared error such that $R_i = \sum_{k=1}^{K}(y_{ik} - \hat{y}_{ik})$. The terms in equation 20 can then be written as equation 21.

$$\begin{aligned} \partial R_i/\partial \widehat{y}_{ki} &= 2\left(\widehat{y}_{ki} - y_{ik}\right) \\ \partial \hat{y}_{ki}/\partial a_{ki}^{(2)} &= \sigma'\left(a_{ki}^{(2)}\right) \\ \partial a_{ki}^{(2)}/\partial \beta_{km}^{(2)} &= z_{mi} \end{aligned} \tag{21}$$

Consider $\frac{\partial R_i}{\partial \beta_{km}^{(2)}} = \delta_{ki}^{(2)} z_{mi}$ for the first layer in the neural network where $\delta_{ki}^{(2)} = \partial R_i/\partial a_{ki}^{(2)}$, which can be written as $\delta_{ki}^{(2)} = 2\left(\widehat{y}_{ki} - y_{ik}\right)\sigma'\left(a_{ki}^{(2)}\right)$ and $z_{mi} = h\left(a_{mi}^{(1)}\right)$. Equation 22 then presents a step backward to consider the partial derivative with respect to the weights in the first layer of the network. Given the simplifications in equation 23, the gradient of the loss function R_i with respect to $\beta_{mj}^{(1)}$ can then be written as equation 24.

$$\frac{\partial R_i}{\partial \beta_{mj}^{(1)}} = \sum_{k=1}^{K} \frac{\partial R_i}{\partial \widehat{y}_{ki}} \cdot \frac{\partial \widehat{y}_{ki}}{\partial a_{ki}^{(2)}} \cdot \frac{\partial a_{ki}^{(2)}}{\partial z_{mi}} \cdot \frac{\partial z_{mi}}{\partial a_{mi}^{(1)}} \cdot \frac{\partial a_{mi}^{(1)}}{\partial \beta_{mj}^{(1)}} \tag{22}$$

$$\begin{aligned} \partial a_{ki}^{(2)}/\partial z_{mi} &= \beta_{km}^{(2)} \\ \partial z_{mi}/\partial a_{mi}^{(1)} &= h'\left(a_{mi}^{(1)}\right) \\ \partial a_{mi}^{(1)}/\partial \beta_{mj}^{(1)} &= x_{ij}. \end{aligned} \tag{23}$$

Equation 24 can be further simplified by writing $\delta_{mi}^{(1)} = \partial R_i/\partial a_{mi}^{(1)} = h'\left(a_{mi}^{(1)}\right)\sum_{k=1}^{K}\delta_{ki}^{(2)}\beta_{km}^{(2)}$. In this case equation 24 can be written as $\frac{\partial R_i}{\partial \beta_{mj}^{(1)}} = \delta_{mi}^{(1)} x_{ij}$.

$$\frac{\partial R_i}{\partial \beta_{mj}^{(1)}} = \sum_{k=1}^{K} 2\left(\widehat{y}_{ki} - y_{ik}\right)\sigma'\left(a_{ki}^{(2)}\right)\beta_{km}^{(2)} h'\left(a_{mi}^{(1)}\right) x_{ij} = h'\left(a_{mi}^{(1)}\right)\sum_{k=1}^{K}\delta_{ki}^{(2)}\beta_{km}^{(2)} x_{ij} \tag{24}$$

Loss functions are used to determine how far off the neural network is from a correct classification or a correct prediction. In this paper we limit ourselves to the ones that are used for binary classification problems. The best known loss functions for these kinds of problems are the Hinge Loss function and the Binary Cross-Entropy Loss function (BCE) given in equation 25 and 26, respectively. The Hinge Loss however, is designed around a SVM model whereas the BCE loss is used for logistic regressions. Therefore, I use the BCE loss function in this paper.

$$R = \max(0, 1 - y * f(x)) \tag{25}$$

$$R = -y * \log(p) - (1-y) * \log(1-p) = \begin{cases} -\log(1-p), & \text{if } y = 0 \\ -\log(p), & \text{if } y = 1 \end{cases} \tag{26}$$

The adjustment of the gradients as derived in equation 24 takes place using gradient descent algorithms such as the ADAM algorithm, ADAGRAD, and RMSProp. Generally, the exact algorithm used does not necessarily depend on the task at hand but is more so a consequence of the hyperparameter configuration of the model used. However, some guidance can be offered by thinking of the form of the gradient of the loss function. Since this is a convex function, the use of methods that involve a momentum term can be considered. Also, considering the amount of data, optimizers can be used that make use of sampling strategies, also called batch learning. Therefore, I decide to evaluate the performance of the neural network using SGD with an ADAM extension.

Stochastic gradient descent is an extension of the basic gradient descent algorithm given in equation 27. The basic gradient descent algorithm averages the gradient ∇R_i over the entire data sample N. When the data sample N is large, this optimization could take very long. SGD, which is shown in equation 28 where α is the learning rate, offers a quicker way by improving the weights with every single observation in the data sample. However, the drawback of the SGD optimization method is that because the steps are based on single observations convergence to a minimum might take longer or might never be reached due to the effect of noise. It might namely be the case that adjacent data points send the optimizer in two different directions. A compromise between the two above optimizers is given by mini-batch SGD given in equation 29 where $\mathcal{I}$ is a subset of the data sample N. Here, the exact amount of observations per mini-batch is found through experimentation. Another role is played by epochs in training a neural network. An epoch is the amount of times the entire data sample is passed through the neural network. An epoch contains a number of batches. By increasing the amount of epochs the neural network will have the opportunity to better fit the weights to the data. By plotting the amount of epochs against the error rate, an optimum in the amount of epochs can be found based on the minimum error rate.

$$\widehat{\boldsymbol{\beta}}^{(\tau+1)} = \widehat{\boldsymbol{\beta}}^{(\tau)} - \alpha \frac{1}{N} \sum_{i=1}^{N} \nabla R_i \left(\widehat{\boldsymbol{\beta}}^{(\tau)} \right) \tag{27}$$

$$\widehat{\boldsymbol{\beta}}^{(\tau+1)} = \widehat{\boldsymbol{\beta}}^{(\tau)} - \alpha \nabla R_i \left(\widehat{\boldsymbol{\beta}}^{(\tau)} \right) \tag{28}$$

$$\widehat{\boldsymbol{\beta}}^{(\tau+1)} = \widehat{\boldsymbol{\beta}}^{(\tau)} - \alpha \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} \nabla R_i \left(\widehat{\boldsymbol{\beta}}^{(\tau)} \right) \tag{29}$$

All named optimization methods require learning rate α which determines how large the improvement steps will be that gradient descent takes. If α is too large, the optimizer might overshoot and thus not converge but if α is too small then convergence might take very long. In ADAM, the optimizer used in this research, the learning rate is adapted throughout the optimization. ADAM makes use of momentum-like terms, namely an exponentially decaying average of past gradients and past squared gradients, denoted m_t and v_t respectively. How the algorithm adapts m_t and v_t with every iteration is given in equation 30 where g_t is the gradient. The terms θ_1 and θ_2 are hyperparemeters for which Kingma and Ba (2014) suggest $\theta_1 = 0.9$ and $\theta_2 = 0.999$ as default values. ADAM has one peculiarity which is the bias towards zero that the momentum-like terms m_t and v_t have when they are initialized at zero. This drawback is mitigated by introducing the terms $\hat{m}_t$ and $\hat{v}_t$ (see equation 31) which correct m_t and v_t for this bias.

$$\begin{aligned} m_t &= \theta_1 m_{t-1} + (1 - \theta_1)\, g_t \\ v_t &= \theta_2 v_{t-1} + (1 - \theta_2)\, g_t^2 \end{aligned} \tag{30}$$

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1 - \theta_1^t} \\ \hat{v}_t &= \frac{v_t}{1 - \theta_2^t} \end{aligned} \tag{31}$$

Taking all the terms together, equation 32 shows how the weights of the neural network are then updated with every iteration. ϵ is a hyper parameter for which Kingma and Ba (2014) suggest default setting $\gamma = 10^{-8}$.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \gamma} \hat{m}_t \tag{32}$$

In finalizing the training of a neural network, one could run into several obstacles relating to the generalization of the model. This is known as underfitting and overfitting. Underfitting describes the situation in which the fit of a model does not perform well on the training set nor

on the test set and can thus not be generalized. This can be due to misspecification of a model for instance. Underfitting is quite simple to detect using a good performance measure for the task at hand. Overfitting describes the situation in which the general fit of the model is too specific to the data, meaning that the model is not able to generalize adequately. A solution to this obstacle is presented by dropout and regularization. In dropout, nodes in a given layer are randomly dropped out of the neural network, which alters the configuration of the model. The effect the on the remaining nodes in a layer is that their activation obtains a larger influence on the overall outcome of the model. This prevents correlating nodes and causes a stronger contrast between active and non-active nodes leading to an increase in the robustness of the model which then reduces overfitting.

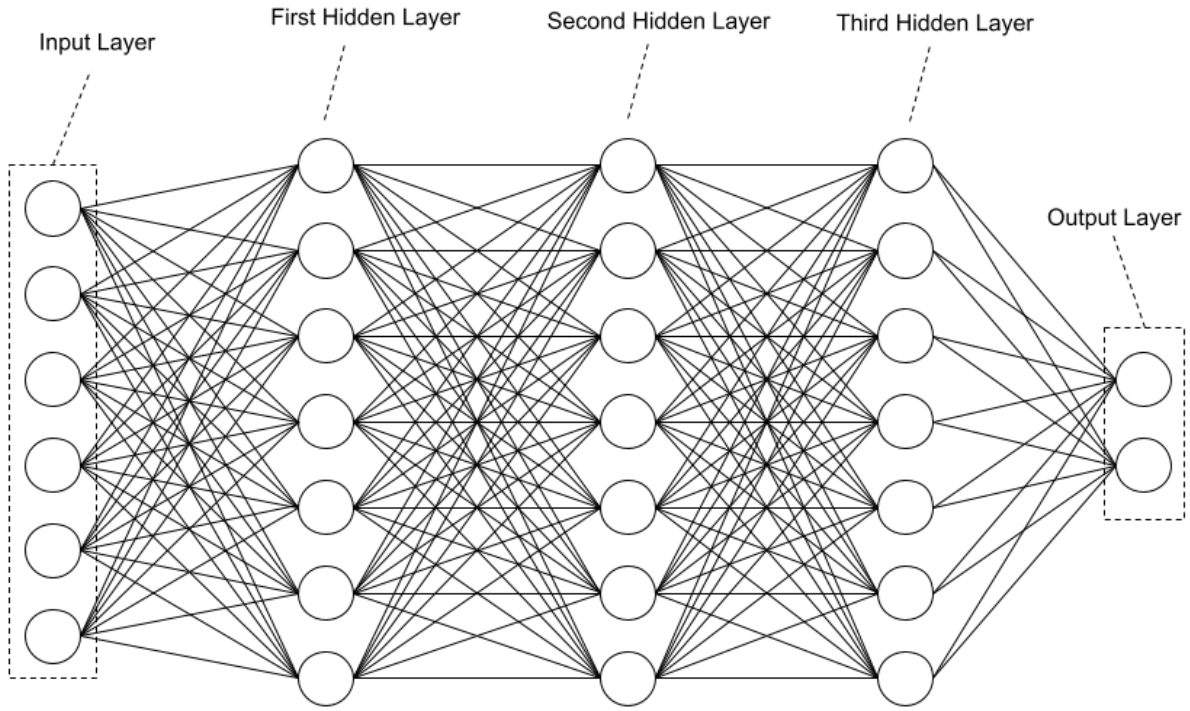


Figure 11: Deep Neural Network. Note that this is not the architecture design used in this research but simply a representation of a neural network.

The exact configuration of the DNN, meaning the number of hidden layers and the amount of nodes per hidden layer, also requires attention. Given these hyper parameters, a tuning is required to find the optimal setup for our problem as it is practically impossible to know what configuration will perform best given certain performance measures. Regardless of the impossibility of saying exactly what can be used certain rules of thumbs are generally followed. One of these rules is that the amount of nodes in the hidden layers should be between the amount of nodes in the input layer and the amount of nodes in the output layer.

D Support Vector Machines

Our goal is to classify data points and indicate to which class they belong. Figure 14 shows the different data points indicated by the color blue and the color purple. Logically one would draw a line exactly between the data points to separate them into two groups. Consider however the case where outliers are present in the data such that some of the blue data points are mixed with the purple data points. This will make it harder to separate the data points using a simple line. Naturally in that case, one should allow for some data points to be misclassified. In this case the distance between the outer most data points of each class and the line indicated by the black arrow is called a soft margin. The use of a soft margin for determining where exactly the data points are split is also called the Support Vector Classifier.

$$\{x : f(x) = x^T\beta + \beta_0 = 0\} \tag{33}$$

$$G(x) = \text{sign}\left[x^T\beta + \beta_0\right] \tag{34}$$

Another situation occurs when the data is only separable in a larger dimensional space. This is where SVMs come into play. SVMs is a classification technique with the objective to find a hyper plane (given in equation 39) that separates data points belonging to a certain class while resting on the outer most data points belonging to those classes. These data points can be seen as pairs of $\mathbf{x}_i \in \mathbb{R}^D$ and $y_i \in \{-1, 1\}$. A decision rule for classifying these data points is given in equation 40 where the parameters β and β_0 are to be optimized. A full derivation of this optimization problem is given in the appendix.

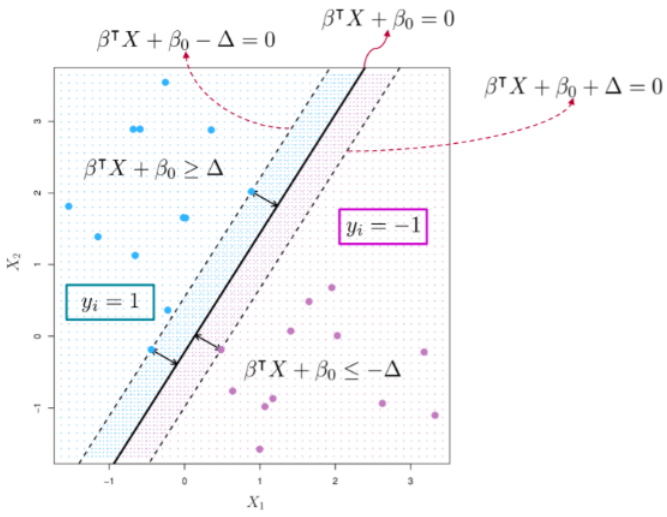


Figure 12: Illustration of a Support Vector Machine in the case of perfectly separable data (Hastie et al. (2009)).

One thing that must be highlighted relates to the separability of data in higher dimensions. This will thus require the evaluation of the distance of data points relative to each other in higher dimensions. This is done using kernel functions. Given the derivation of the decision rule for the SVM classifier in the appendix, the Lagrange dual function of the problem is repeated in equation 35 and 36. Equation 36 represents the same function but with data vector $\mathbf{x}_i$ transformed in the inner product of the data vectors.

$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{i'=1}^{N} \alpha_i \alpha_{i'} y_i y_{i'} x_i^T x_{i'} \tag{35}$$

$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{i'=1}^{N} \alpha_i \alpha_{i'} y_i y_{i'} \left\langle h\left(x_i\right), h\left(x_{i'}\right) \right\rangle \tag{36}$$

Since only the inner product of the data vectors is of importance, the kernel function in equation 37 is used which calculates the inner product in the transformed space. Various kernel functions are given in equation 38 namely the linear kernel, polynomial kernel, and radial basis kernel. Exact knowledge of the transformation $h(x)$ can thus be left aside. The output of these kernel functions then serve as a distance measure that can be replaced back in equation 35.

$$K\left(x, x'\right) = \left\langle h(x), h\left(x'\right) \right\rangle \tag{37}$$

$$\begin{aligned} K\left(x_i, x_j\right) &= x_i^\top x_j \\ K\left(x_i, x_j\right) &= \left(1 + x_i^\top x_j\right)^d \\ K\left(x_i, x_j\right) &= e^{-\gamma \| x_i - x_j \|^2}, \gamma > 0 \end{aligned} \tag{38}$$

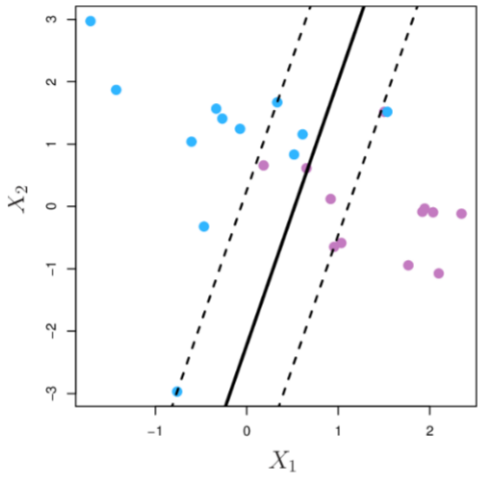


Figure 13: Illustration of a Support Vector Machine in the case of data that is not perfectly separable (Hastie et al. (2009)).

E Derivation Support Vector Machine classifier

$$\left\{x : f(x) = x^T\beta + \beta_0 = 0\right\} \tag{39}$$

$$G(x) = \text{sign}\left[x^T\beta + \beta_0\right] \tag{40}$$

$$\max_{\beta,\beta_0,\|\beta\|=1} M \text{ subject to } y_i\left(x_i^T\beta + \beta_0\right) \geq M, i = 1, \ldots, N, \tag{41}$$

This problem can also be written in the following manner:

$$\min_{\beta,\beta_0} \|\beta\| \text{ subject to } y_i\left(x_i^T\beta + \beta_0\right) \geq 1, i = 1, \ldots, N, \tag{42}$$

Allowing for misclassifications using ξ gives:

$$y_i\left(x_i^T\beta + \beta_0\right) \geq M - \xi_i \text{ or } y_i\left(x_i^T\beta + \beta_0\right) \geq M\left(1 - \xi_i\right), \tag{43}$$

$$\min_{\beta,\beta_0} \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{N}\xi_i \text{ subject to } \xi_i \geq 0, y_i\left(x_i^T\beta + \beta_0\right) \geq 1 - \xi_i \forall i \tag{44}$$

The Lagrange (primal) function is given by

$$L_P = \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i\left[y_i\left(x_i^T\beta + \beta_0\right) - (1 - \xi_i)\right] - \sum_{i=1}^{N}\mu_i\xi_i, \tag{45}$$

This is minimized w.r.t β, β_0 and ξ_i. First order conditions give:

$$\beta = \sum_{i=1}^{N}\alpha_i y_i x_i, 0 = \sum_{i=1}^{N}\alpha_i y_i, \alpha_i = C - \mu_i, \forall i, \tag{46}$$

Furthermore, we impose positivity constraints $\alpha_i, \mu_i, \xi_i \geq 0 \forall i$. The Lagrangian dual objective function

$$L_D = \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{i'=1}^{N}\alpha_i\alpha_i y_i y_{i'} x_i^T x_{i'}, \tag{47}$$

The objective becomes to maximize L_D subject to $0 \leq \alpha_i \leq C$ and $\sum_{i=1}^{N}\alpha_i y_i = 0$. KKT conditions include the following constraints:

$$\alpha_i\left[y_i\left(x_i^T\beta + \beta_0\right) - (1 - \xi_i)\right] = 0 \mu_i\xi_i = 0 y_i\left(x_i^T\beta + \beta_0\right) - (1 - \xi_i) \geq 0 \tag{48}$$

Above equation uniquely characterizes the solution to the primal and dual problem.

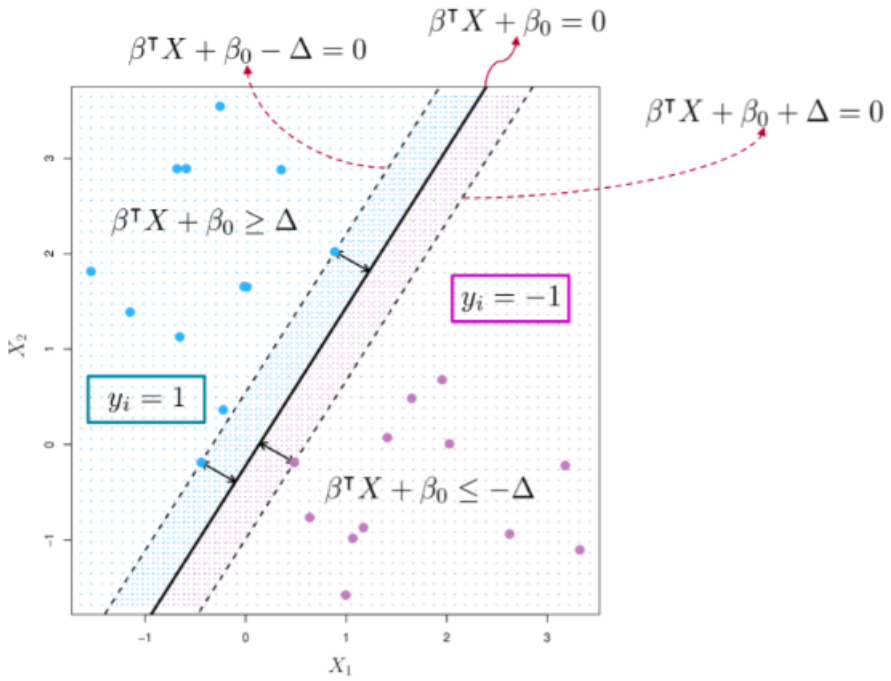


Figure 14: Illustration of a Support Vector Machine in the case of perfectly separable data (Hastie et al. (2009)).

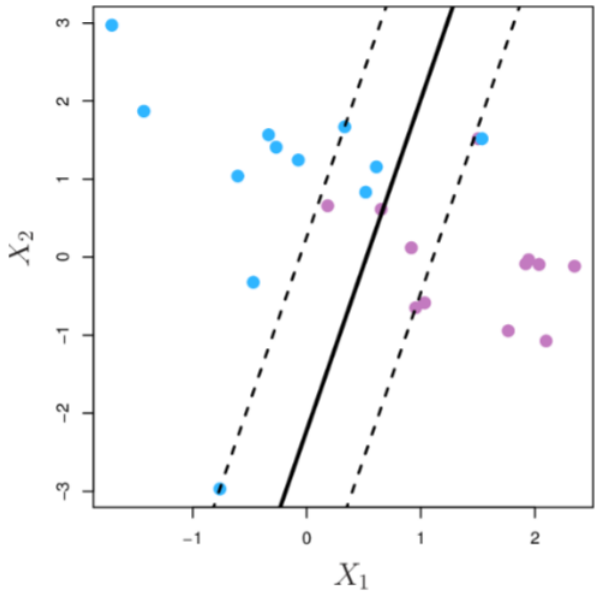


Figure 15: Illustration of a Support Vector Machine in the case of data that is not perfectly separable (Hastie et al. (2009)).

F Pseudocode Gibbs Sampler

Algorithm 1 Gibbs sampler

Initialize $x^{(0)} \sim q(x)$
for iteration $i = 1, 2, \ldots$ **do**
$x_1^{(i)} \sim p(X_1 = x_1 | X_2 = x_2^{(i-1)}, X_3 = x_3^{(i-1)}, \ldots, X_D = x_D^{(i-1)})$
$x_2^{(i)} \sim p(X_2 = x_2 | X_1 = x_1^{(i)}, X_3 = x_3^{(i-1)}, \ldots, X_D = x_D^{(i-1)})$
$\vdots$
$x_D^{(i)} \sim p(X_D = x_D | X_1 = x_1^{(i)}, X_2 = x_2^{(i)}, \ldots, X_D = x_{D-1}^{(i)})$
end for

Figure 16: Pseudocode for Gibbs Sampling

G Pseudocode SMOTE

Algorithm *SMOTE*(T, N, k)
Input: Number of minority class samples T; Amount of SMOTE $N\%$; Number of nearest neighbors k
Output: $(N/100)$ * T synthetic minority class samples
1. (∗ *If N is less than 100%, randomize the minority class samples as only a random percent of them will be SMOTEd.* ∗)
2. **if** $N < 100$
3. **then** Randomize the T minority class samples
4. $T = (N/100) * T$
5. $N = 100$
6. **endif**
7. $N = (int)(N/100)$ (∗ *The amount of SMOTE is assumed to be in integral multiples of 100.* ∗)
8. k = Number of nearest neighbors
9. *numattrs* = Number of attributes
10. *Sample*[ ][ ]: array for original minority class samples
11. *newindex*: keeps a count of number of synthetic samples generated, initialized to 0
12. *Synthetic*[ ][ ]: array for synthetic samples
(∗ *Compute k nearest neighbors for each minority class sample only.* ∗)
13. **for** $i \leftarrow 1$ **to** T
14. Compute k nearest neighbors for i, and save the indices in the *nnarray*
15. Populate(N, i, *nnarray*)
16. **endfor**

Populate(N, i, *nnarray*) (∗ *Function to generate the synthetic samples.* ∗)
17. **while** $N \neq 0$
18. Choose a random number between 1 and k, call it nn. This step chooses one of the k nearest neighbors of i.
19. **for** $attr \leftarrow 1$ **to** *numattrs*
20. Compute: $dif = Sample[nnarray[nn]][attr] - Sample[i][attr]$
21. Compute: gap = random number between 0 and 1
22. $Synthetic[newindex][attr] = Sample[i][attr] + gap * dif$
23. **endfor**
24. $newindex{+}{+}$
25. $N = N - 1$
26. **endwhile**
27. **return** (∗ *End of Populate.* ∗)
End of Pseudo-Code.

Figure 17: Pseudocode for the SMOTE Algorithm (Chawla et al. (2002))

H Pseudocode ADASYN

Procedure

(1) Calculate the degree of class imbalance:

$$d = m_s/m_l \tag{1}$$

where $d \in (0,\ 1]$.

(2) If $d < d_{th}$ then (d_{th} is a preset threshold for the maximum tolerated degree of class imbalance ratio):

(a) Calculate the number of synthetic data examples that need to be generated for the minority class:

$$G = (m_l - m_s) \times \beta \tag{2}$$

Where $\beta \in [0,\ 1]$ is a parameter used to specify the desired balance level after generation of the synthetic data. $\beta = 1$ means a fully balanced data set is created after the generalization process.

(b) For each example $\boldsymbol{x_i} \in minorityclass$, find K nearest neighbors based on the Euclidean distance in n dimensional space, and calculate the ratio r_i defined as:

$$r_i = \Delta_i/K, \quad i = 1, ..., m_s \tag{3}$$

where Δ_i is the number of examples in the K nearest neighbors of $\boldsymbol{x}_i$ that belong to the majority class, therefore $r_i \in [0,\ 1]$;

(c) Normalize r_i according to $\hat{r}_i = r_i / \sum_{i=1}^{m_s} r_i$, so that $\hat{r}_i$ is a density distribution ($\sum_i \hat{r}_i = 1$)

(d) Calculate the number of synthetic data examples that need to be generated for each minority example $\boldsymbol{x}_i$:

$$g_i = \hat{r}_i \times G \tag{4}$$

where G is the total number of synthetic data examples that need to be generated for the minority class as defined in Equation (2).

(e) For each minority class data example $\boldsymbol{x}_i$, generate g_i synthetic data examples according to the following steps:

Do the **Loop** from 1 to g_i:

(i) Randomly choose one minority data example, $\boldsymbol{x}_{zi}$, from the K nearest neighbors for data $\boldsymbol{x}_i$.

(ii) Generate the synthetic data example:

$$\boldsymbol{s_i} = \boldsymbol{x_i} + (\boldsymbol{x_{zi}} - \boldsymbol{x_i}) \times \lambda \tag{5}$$

where $(\boldsymbol{x_{zi}} - \boldsymbol{x_i})$ is the difference vector in n dimensional spaces, and λ is a random number: $\lambda \in [0,\ 1]$.

End **Loop**

Figure 18: ADASYN Pseudocode He et al. (2008)

I Pseudocode ADAM optimizer

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

Figure 19: ADAM Pseudocode (Kingma and Ba (2014)).