Final Report: Evaluating Machine Learning Techniques in the Field of Bankruptcy Prediction

By: Oliver Hancock

Supervised By: Walter Colombo

BSc Computer Science with Year in Industry

School of Computer Science and Informatics, Cardiff University

Completed: 09/05/2024

I. Abstract

The merging of the fields of finance and machine learning has given rise to a new set of tools to conduct financial analysis on tradable securities, generating new opportunities to profit in the market. Financial distress analysis is a sub field of this growing sector that has not received as much attention as price or broader economic forecasting. Through this project several machine learning techniques as well as a proprietary neural network are implemented alongside a full preprocessing pipeline to conduct classification analysis on individual institutions financial health. Data from the Taiwanese Economic Journal, including 96 financial metrics and 6819 companies is used to train and evaluate the performance of each of these models. Furthermore, a command line interface is developed to allow a broad range of potential stakeholders to conduct their own analysis.

II. Acknowledgements

I would like to thank my moderator Walter Colombo for giving great insights to my project and helping steer it to the form it has taken and motivating me to keep going. I would also like to thank my friends and family, particularly Bryn and Fraser for supporting me and keeping me driven to complete my work to the best of my ability.

Contents

List of Figures

List of Abbreviations

AI	Artificial Intelligence
NN	Neural Network
ML	Machine Learning
RF	Random Forest
LR	Logistic Regression
SVM	Support Vector Machine
FCA	Financial Conduct Authority
PRA	Prudential Regulation Authority
GCF	Government Commercial Function
LTSM	Long Short-Term Memory
EDA	Exploratory Data Analysis
RNN	Recurrent Neural Network
MLP	Multilayer Perceptron
GCF	Government Commercial Function
SMOTE	Synthetic Minority Over-Sampling Technique
ANOVA	Analysis of Variance
DFNN	Deep Feedforward Neural Network
MSE	Mean Squared Error
CART	Classification and Regression Tree
GAN	Generative Adversarial Network
GNN	Graph Neural Network

III. Introduction

The ability to predict the movements of financial markets has historically led much innovation in both financial and scientific fields. It creates insights from data and behaviour that can be directly leveraged to generate wealth, attracting large attention from a variety of stakeholders including, investors, creditors, auditors, and regulatory bodies.

This project aims to assess the accuracy and reliability of various existing models and a create a proprietary model bringing together the most valuable elements of these technologies to create a tool that could be used by stakeholders. The aim is not to create a fully complete product but one that could be used in conjunction with other financial analysis techniques to aid a rigorous analysis of a company's financial standing in regard to financial distress.

Financial distress analysis was chosen specifically as it is a sub field within computational financial analysis that has not received as much attention and time in developing potential solutions compared to price prediction. It also allows me to further my understanding of artificial intelligence, building on the theoretical fundamentals completed as part of my university syllabus.

Introduction to Financial Distress Analysis

Given the financial nature of the input data, effort has been made to understand at a basic level the importance of holistic nature of the underlying financial data. This includes calculating key financial ratios from raw data gathered from financial reports such as balance sheets, cashflow statements and income statements. These aides the project in the understanding of the target user of this technology, companies, or persons, that may not have the most understanding of the underlying computational workings of these methods but understand financial markets and are looking to leverage the predictions for financial gain or for future financial planning.

Introduction to Machine Learning

Over the years, statistical methods such as Z-Score analysis built on Multiple Discriminant Analysis (MDA) in the 1960s (Altman, 1968) to Logistic Regression in the 1980s (Ohlson, 1980) and AdaBoost implementations of Decision Trees in the 2000s (Coşkun & Turanli, 2023) have been used to gain insights. All these have tried to gain understandings of future movements from financial ratios. Whilst there is a range of use cases within the financial sector for these technologies, bankruptcy prediction is a case that has an abnormally high risk to reward ratio, has a wide range of potential stakeholders and has significant economics with cascading effects (Alam, et al., 2021). With the great strides that the wider field of Artificial

Intelligence has made in recent years, there is some catching up to do in terms of applying these new models and optimization features to specific use cases. The key aim of this project is to assess and design a robust model to differentiate between financially distressed and financially health companies.

The aim of this project is to provide a comprehensive comparison of a selection of traditional statistical analysis techniques before designing a proprietary neural network to gain insights in the use of more modern techniques to differentiate between financially distressed and financially health companies. Whilst some research has been done on the application of neural networks to predict financial distress (Wiklund, 2022), there is a lack of comprehensive review of model tuning techniques such as hyperparameters selection, selection of activation and error functions as well as higher level techniques such as auto encoders and advanced networks such as Graph Neural Networks (GNNs) and Generative Adversarial Networks (GANs). This project aims to understand how these technologies can be applied and collated within a tangible setting.

Objectives

The primary aim is to create a system that uses various forms of ML and AI models to predict with upmost certainty the level of financial distress within a company. This prediction is intended to be used in conjunction with more rigorous financial analysis of a company. These predictions will be derived from the results of models that have been evaluated within the earlier stages of this project to find the most suitable approach to the task. These objectives can be broken down as follows:

1) Review literature.
   a) Understand the current standing of research in the field and decide on a more concise direction of research and development towards the solution.

2) Data acquisition and pre-processing.
   a) Find main dataset and alternative datasets to form a more varied testing suite.
   b) Complete basic data analysis on the raw dataset to gain an overview understanding.
   c) Undertake data cleaning procedures to prepare the data for applications in models.
   d) Create functions to generate financial ratios found from literature to generate more descriptive parameters.

3) Implement traditional ML models and processes.
   a) Models proposed to implement are RF, LR, SVM and Gradient Boosting
   b) Within these models, further procedures such as feature selection, feature scaling, hyperparameter optimization and cross validation.

4) Evaluate ML models Traditional ML techniques.

   a) Evaluation will be conducted on analysis done on the dataset; this is available from community code on Kaggle.
   b) Decide on optimum preprocessing steps and hyper parameter selection within each model.
   c) Also conclude which models can be implemented within ensemble learning.

5) Implement NN models.
   a) Make decisions towards the architecture and structure of the NN model.
   b) Create the model and train it from the test data.
   c) Implement a system to generate predictions from external/untrained data.

6) Evaluate NN models.
   a) Develop a test suite with passable criteria for the NN model.
   b) Use this information to further tune and improve the model iteratively.
   c) Retrain and finalise the model.

7) Amalgamate most performant models to usable system.
   a) Create an algorithm to generate an ensemble of techniques and models.
   b) Implement a system to create a final prediction from this ensemble of methods.

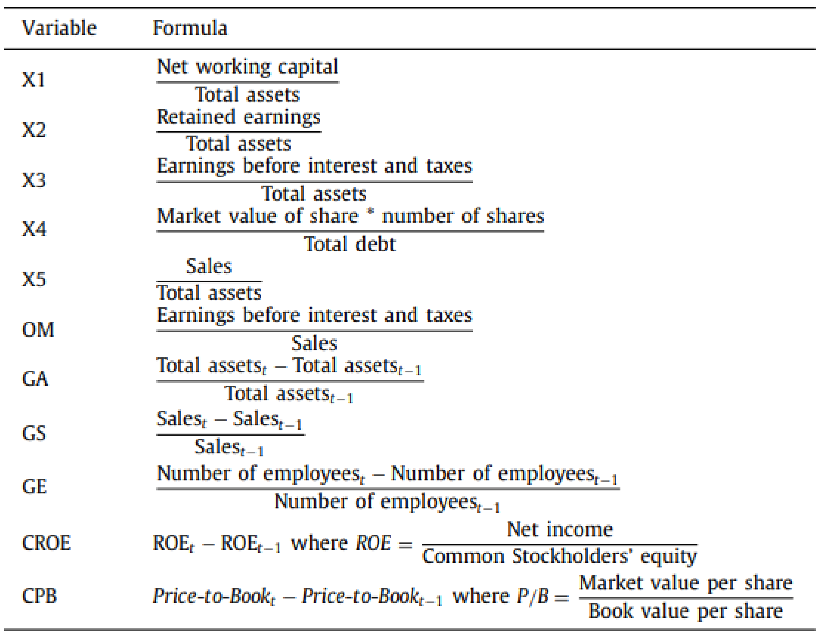
IV. Background

Financial Distress Analysis

Financial analysis has been at the forefront of business decision making for centuries, allowing for deep insight into the health and performance of companies. Financial distress analysis has become a key subcategory of the field. This is due to the critical nature of the research and findings that this provides. It is traditionally conducted by parties in all areas interested in either investing or the general running's of a business.

The origins of financial distress analysis go back to the 19th century (Richardson & Sablik, 2015) and built upon the work conducted in broader financial analysis. This involved the key discovery of gaining information through data through the calculation of financial ratios, helping to simply understand the goings on of businesses, in the aim to create codified systems to evaluate risk within a company. Some common early ratios included, working capital / current assets, net worth / total debt, and current assets / current liabilities. These ratios would be considered primitive by today's standards but are still considered fundamental in the early stages of some analysis. Into the 20th century the discipline started relying much more on statistical methods and became far more rigorous. Some key improvements were the development of a wider pool of ratios to conduct analysis through including the understanding of the key role of cashflow based ratios (Beaver, 1966). Improvements in the field came from a wider adoption of said techniques from institutions and created beneficial guidance for corporations themselves, enabling businesses to understand a state of risk and create routines and business decisions to improve their financial standing. This accompanied improvements in the statistical processes used to gain insights from these ratios including Z-Score a form of MDA (Altman, 1968).

Moving into the 21st century the power of computation was widely applied to the field allowing for more complex analysis to occur. Further and more granular financial ratios were calculated and implemented, continuing the trends of previous work. Much of these ratios were based around analysing change and rate of change of ratios and those related to growth. Given the abundance of data around businesses from technological advances, metrics of the structure of businesses were also implemented in the analysis through datapoints such as number of employees, salaries, and governance structures (Barboza, Kimura, & Altman, 2017). An example of metrics used within the referenced study is seen below in Figure IV-1: Ratios and metrics used for modern financial distress analysis .



Variable	Formula
X1	$\frac{\text{Net working capital}}{\text{Total assets}}$
X2	$\frac{\text{Retained earnings}}{\text{Total assets}}$
X3	$\frac{\text{Earnings before interest and taxes}}{\text{Total assets}}$
X4	$\frac{\text{Market value of share * number of shares}}{\text{Total debt}}$
X5	$\frac{\text{Sales}}{\text{Total assets}}$
OM	$\frac{\text{Earnings before interest and taxes}}{\text{Sales}}$
GA	$\frac{\text{Total assets}_t - \text{Total assets}_{t-1}}{\text{Total assets}_{t-1}}$
GS	$\frac{\text{Sales}_t - \text{Sales}_{t-1}}{\text{Sales}_{t-1}}$
GE	$\frac{\text{Number of employees}_t - \text{Number of employees}_{t-1}}{\text{Number of employees}_{t-1}}$
CROE	$\text{ROE}_t - \text{ROE}_{t-1}$ where $ROE = \frac{\text{Net income}}{\text{Common Stockholders' equity}}$
CPB	$\textit{Price-to-Book}_t - \textit{Price-to-Book}_{t-1}$ where $P/B = \frac{\text{Market value per share}}{\text{Book value per share}}$

Figure IV-1: Ratios and metrics used for modern financial distress analysis (Barboza, Kimura, & Altman, 2017)

These newfound datapoints and metrics were used in conjunction with far more complex statistical methods with the development of machine learning techniques. Now any financial distress analysis relies heavily if not solely on this method. From this a separate movement pushing for more understanding of qualitative factors within a company such as corporate governance, organizational culture and managment quality.

<u>Machine Learning</u>

A subset of Artificial Intelligence, Machine Learning has evolved significantly from its conception in the 1950s. The original problem sought to be solved, was that of creating algorithms and models that could infer knowledge or make predictions from data without explicit programming. It is the search to create automated systems to discover patters and learn from data, allowing machine to improve performance through iterations when applied to specific tasks.

Early Works

Some of the earliest work in the field related to Arthur Sameul, who led much of the early work in computer gaming and artificial intelligence. He is the one who first referred to the work in this field as “machine learning”. The early applications of his work were implemented in a system to automate a game of checkers (Sameul, 1959). This was implemented using a search tree system, a simple graph-based model. Scoring was calculated through the use of a minimax strategy, this strategy would later be researched further and formalised as the minmax algorithm. The novel element of this work was the implementation of alpha-beta pruning, allowing for possible end cases that are known to not affect the decision making of earlier turns to be excluded from consideration, making it viable in terms of computation time. This work demonstrated the potential for machines to be able to learn and adapt to changing systems whilst making consistent accurate decisions.

Artificial Neural Networks (ANNs)

So much of modern technology has been inspired by what is seen in nature and implementing manmade systems of natural behaviour. ANNs are no different and the work conducted by Donald Hebb sought to use the process to create abstractions of the biological neural networks that make up our brain through a system name Hebbian Learning (Hebb, 1949). It built up from the foundational elements of neurons, connected by synapses, using an electrical stimuli input, to generate electrical impulses to form our every conscious and sub-conscious thought. Within an ANN the synapses are modelled as weights where negative values represent biological inhibitory connections with positive values constitute excitatory connections.

There are a huge number of different types of ANNs, each suited to solving different problems, and each with flexibility within the model itself for tuning. The main types are:

- Perceptron (and multi-layer Perceptrons)
- Feed Forward Neural Network
- Convolutional Neural Network
- Recurrent Neural Network
- LTSM – Long Short-Term Memory
- Radial Basis Functional Neural Network

Existing Work

There has been previous research in the applications of Machine Learning in the field of financial distress analysis. However, this research seems to lack discussion or do not include key optimization features that are available to MLP. A notable exception is (Wiklund, 2022) “predicting company bankruptcy using artificial neural networks” who’s work included that of a similar neural network structure to that which is planned within my project. However, this paper lacked the implementation of ensemble learning and omitted any use of time series data allowing for the use of other key deep learning technologies. This study used a much smaller number of input variables for its training which I hope to improve on in my work.

Another notable piece of research is that of (Barboza, Kimura, & Altman, 2017) “Machine learning models and bankruptcy prediction”. This work is often cited as being a highly comprehensive overview of traditional machine learning methods applied to the field and served as part of the inspiration to undertake this project. Its use of variety financial ratios and descriptive metrics of a company structure, such as number employees creates insightful input variables, however there is scope for a much greater number of variable as well as input companies to improve the accuracy of the model. Its findings showed the possibility for successful results in the field, especially in the use boosting and bagging techniques with random forests, producing accuracy of ~86%.

Finally, a review of studies on bankruptcy prediction models (Alaka, et al., 2018) gives a broad overview of how the field as a whole has performed in their studies. It gives recommendations on where future research should be conducted and what is to be expected. This paper gives a strong indication as to whether the results I will be analysing from my own work is in line with that of our peers. Its recommendations are to continue focus on ANNs and SVMs as they were the most performant in the review of studies. It also makes comment on the implementation of hybrid and ensemble techniques that I am hoping to capture in this work.

Some implementations of this work have been completed and been made commercially available. These systems, namely Risk Management solutions by SAS (SAS, n.d.) however are packaged as part of a broader management toolset. These tools are also solely focused for a corporate audience and would be considered unobtainable for a small or lone investor looking to conduct their own research. As such there is a large population who are less technologically adept and able to implement libraries such as those used in the implementation section of the project.

Stakeholders

The broad range of stakeholders arises from the fact that financial distress can create investment opportunities for some stakeholders, such as short sellers looking to bet on negative trends in a company's stock price. On the other side executives within a business can use the findings of such analysis to make decisions within the business itself. Finally, and often most commonly, creditors rely on such analysis to determine the suitability of companies and separately individuals, for financial credit products such as loans, risk management services and asset backed financing solutions. Regulatory bodies also conduct deep analysis on in the field both in a macro and micro scale. GFC use this to help conduct due diligence on individual companies for suitability of rewarding government contracts (Government Commercial Function, 2023) whereas a body such as the Bank of England conduct such analysis of large numbers of corporate bodies to gauge the stability of individual sectors and the economy as a whole.

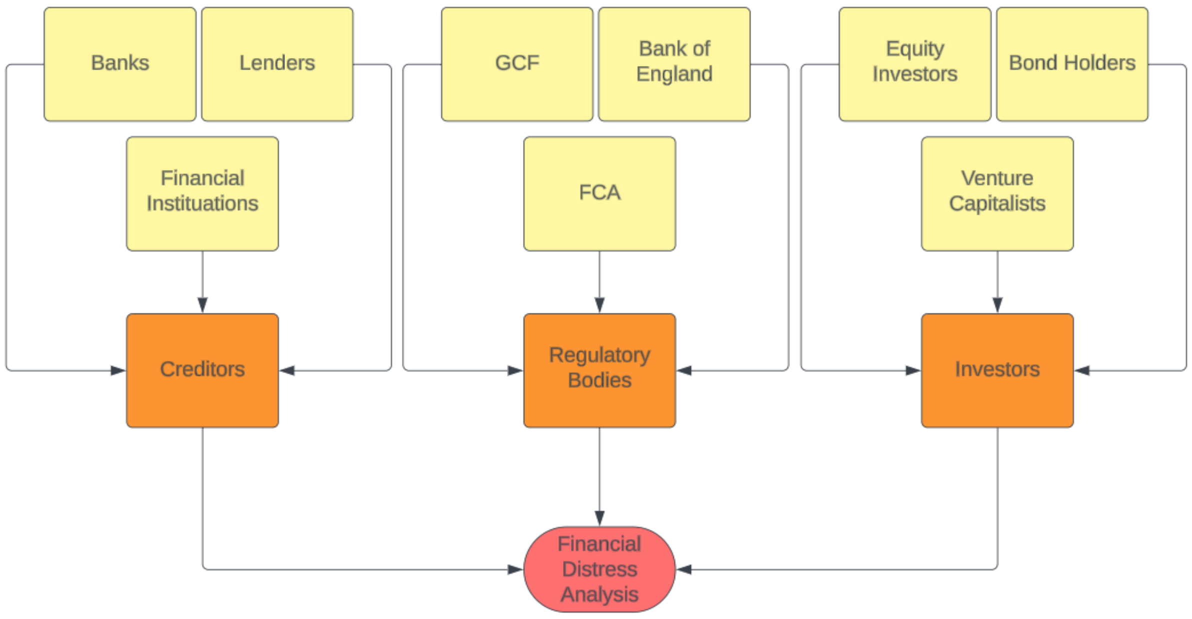


Figure IV-2:Key Stakeholders in the Financial Distress sector.

The implementation of this specific project is intended to be used by those wishing to conduct their own statistical and algorithmic research in regard to a specific company's financial distress status. There is an expectation of a certain degree of understanding of basic computer science concepts, but a very low level. This would involve the ability to install pip packages for python use, as well as understanding and possibly generating .csv files. This also allows for a broader pool of stakeholders as these are but the minimum requirements and those working in a professional setting within the financial sector could also see use for a tool such as this.

V. Specification and Design

Requirements

As described this project has two primary objectives, finding the optimum usage and implementation of various machine learning models, and subsequently creating a simple terminal-based interface for use by stakeholders. To this end, requirements can be broken down into:

Data Preparation

Developing a system for data entry and data preprocessing to train the models on. These data preprocessing steps include but are not limited to data cleaning, adding missing data or removing values that are duplicate entries, outlier removal and feature variable title cleaning. From this any data transformation steps are required such as creating numerical representations of categorical variables, feature scaling and normalization. Finally, the data is then split into training, testing and validation subsets for implementation in the training stage of the model.

Model Integration

This is the implementation of each model itself. This should be done in using a modular structure for ease of expiation and implementation of other models later in the development cycle of after the completion of this project. Each model should be simple to initialise and train whilst also allowing for functionality to make predictions without having to retrain the model each time this functionality is used. This phase also includes creating and training the NN model and saving said model to reduce the time spent waiting for the model to train, given NN typically take much longer to train than traditional models. In this phase hyper parameters and optimization techniques should also be implemented.

Prediction Generation

The developed solution is not really of any great use if it can't be used to predict the future prospects of companies outside of the training dataset. To reach this goal a template document will need to be created so that the financial information of the company can be input in a simple way and in the correct format for the program to infer. There then needs to be a system to apply the same preprocessing steps as was completed on the training data. Finally, the models need to be tested using this data and the results from each model parsed through an ensemble function to conclude on a majority decision on the predicted health of the company. All of this information needs to be collated and returned to the user in a simple readable format with some appropriate visualisations for greater understanding. This can be done either through the command line or by generating a file for later inspection.

Design Methodology

As this project is being developed by a single developer, there is not requirement for a broad and detailed methodology such as Agile or Waterfall as traditional projects would start with. The overall philosophy of this project is one similar to Agile involving making development decisions and deciding on the best course of action through understandings found during writing the code. This could be as based on the results of the performance of a model or insights that are found in the data or its limitations. To this end, the primary goals and timeline of they will be tackle has been written and can be seen below. However, this is not concrete and is open for change and updates as progress it made.

Machine Learning

Logistic Regression (LR)

Logistic regression is a statistical modelling technique in the field of binary classification. It is similar to the more primitive linear regression, implementing a simple straight line to differentiate values based on one or more predictor variable. Logistic regression makes uses of a sigmoid function defined as:

$$S(x) = \frac{1}{1 + e^{-x}}$$

Which when viewed graphically creates a line:

The model iterates through coefficient values that are estimated through a separate algorithm such as Maximum Likelihood Estimation (MLE), with the aim of maximizing the likelihood of returning the observed value. This produces a likelihood function derived from the product of the probabilities for each datapoint. Once this is done, the sigmoid function is used to calculate the prediction probability ready to be applied to unseen data by using the

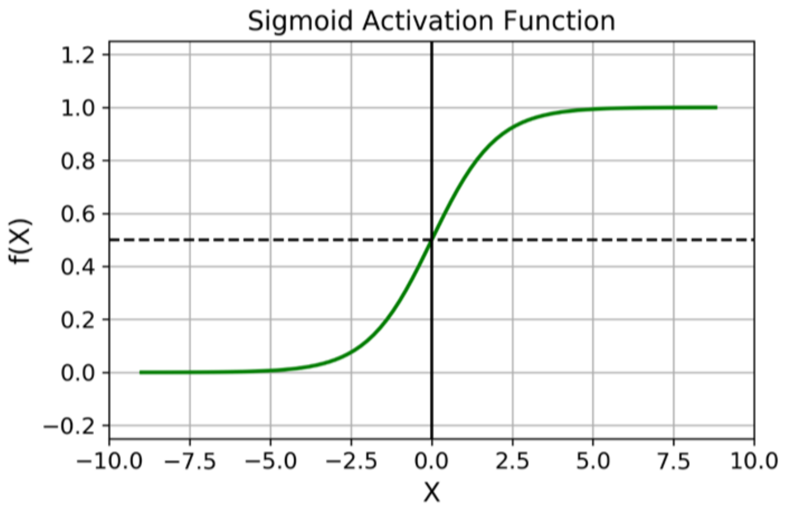


Figure V-1: Graphical representation of sigmoid curve (Shanu, 2022)

unseen input parameters and the known probability parameters. Given the binary nature of the output of the model it is ideally suited to binary classification problems, such as the one discussed in this project.

Support Vector Machine (SVM)

A SVM is another supervised learning algorithm that can be used for both classification and regression predictions. The fundamental process underpinning the algorithm is that of finding a hyperplane that clearly separates input variables from the different decision variable classes. The hyperplane is calculated such that it separates the classes by the widest possible margin. This can be seen in Figure V-2: Visualization of Support Vector Machine (SVM), as H3 produces the greatest margin, whilst H2 fully separates the classes, it does not leave the greatest margin for error.

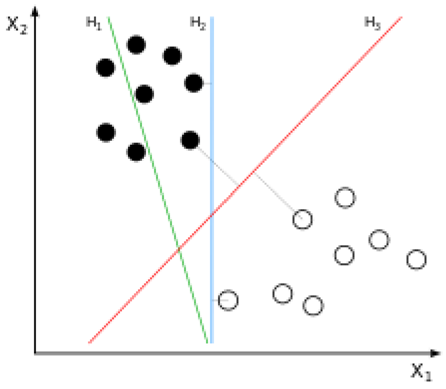


Figure V-2: Visualization of Support Vector Machine (SVM) (Wikipedia contributors, 2024)

There are different types of SVMs such as linear (simple) or non-linear (kernel) SVMs which perform better in different use cases. Kernal SVMs typically involve higher dimensional data that can be used for more complex classification and optimization tasks. Figure V-3: Graphical representation of linear vs non-linear SVM, shows this difference within a 2D plane. Kernal SVMs can further advanced using polynomial, radial basis functions or sigmoid functions, all implementing the basic notion of calculating the similar between datapoints within a space.

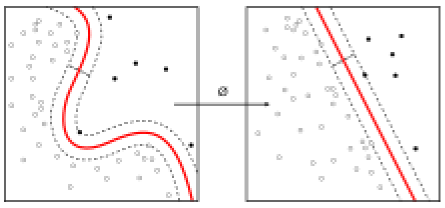


Figure V-3: Graphical representation of linear vs non-linear SVM (Wikipedia contributors, 2024)

SVMs seek to minimize this classification error by mapping input data into higher dimensional spaces through the described kernel functions.

Random Forest

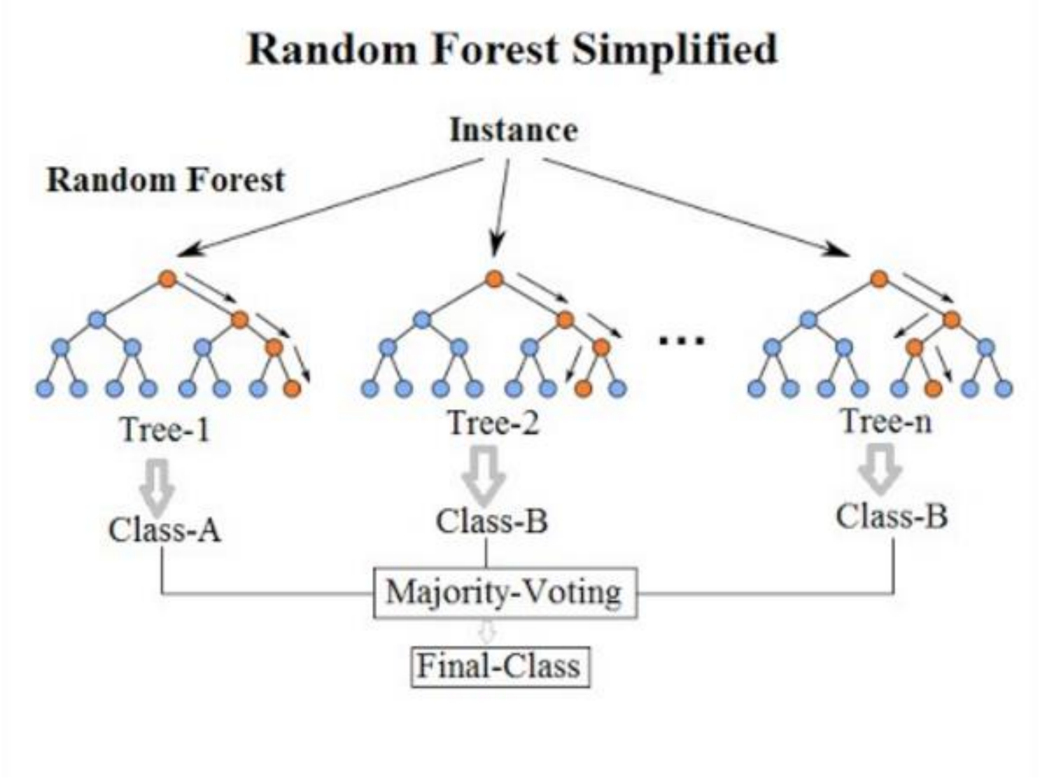


Figure V-4: Visualization of Random Forest Algorithm (Koehrsen, 2017)

A technique often implemented in ML is Ensemble Learning, bringing together the predictive power of multiple models to provide a single predictive result. Random forests use this strategy through running multiple decision tree models. A decision tree is a graph-based structure that conducts a question like decision on each node, splitting the data further to arrive at a final decision. The end result of a decision tree is to find an optimal subset of data that minimizes an error score. The Classification and Regression Tree (CART) algorithm is commonly used to train the tree, whilst Gini impurity, Mean Squared Error (MSE) or Information Gained are common scoring algorithms. Decision trees are simple to implement and are therefore quite common but can lead to damaging problems including bias and overfitting, from attempting to incorporate the unrelated noise within the data into the prediction of the structure.

This is why ensemble techniques are so important to random forest. Multiple decision trees are brought together using both bootstrap aggregating (bagging) and feature randomness to improve the results of the model. Bagging is a common technique that splits the input data into multiple subsets, replacing elements of the original dataset, applying the model before aggregating the results from each of the models, in this case decision trees. Feature randomness occurs at each split of the decision tree, reducing the number of feature variables to be considered in the child nodes, further reduces noise, and allows for higher dimensional data. The results of each of these decision trees are then parsed to a majority voting function that selects the most commonly returned value from the ensemble of trees.

Gradient boosting

Similarly to the Random Forest model, Gradient Boosting is an ensemble algorithm. Within gradient boosting the underlying techniques are required to be weak learners, which overtime create a knowledge base for further iterations to learn from. Gradient boosting is an alternative way to implement ensemble learning, in this case it is sequential rather than parallel as was the case for random forest. Gradient boosting can be used with many different underlying learners such as linear models, or K-Nearest Neighbour classifiers.

This weak learner can for example be as simple as a mean average of the feature variables. Each iteration a loss function is used to calculate the error of the previous iteration and a new weak learner is then built from the resulting loss result. Each complete iteration, the previous learners are summed to a total ensemble classifier. Figure V-5: Visualization of the iteration within the gradient boosting algorithm, shows how the regression from each loss function is summed to the total to make it progressively more accurate. After a certain stopping

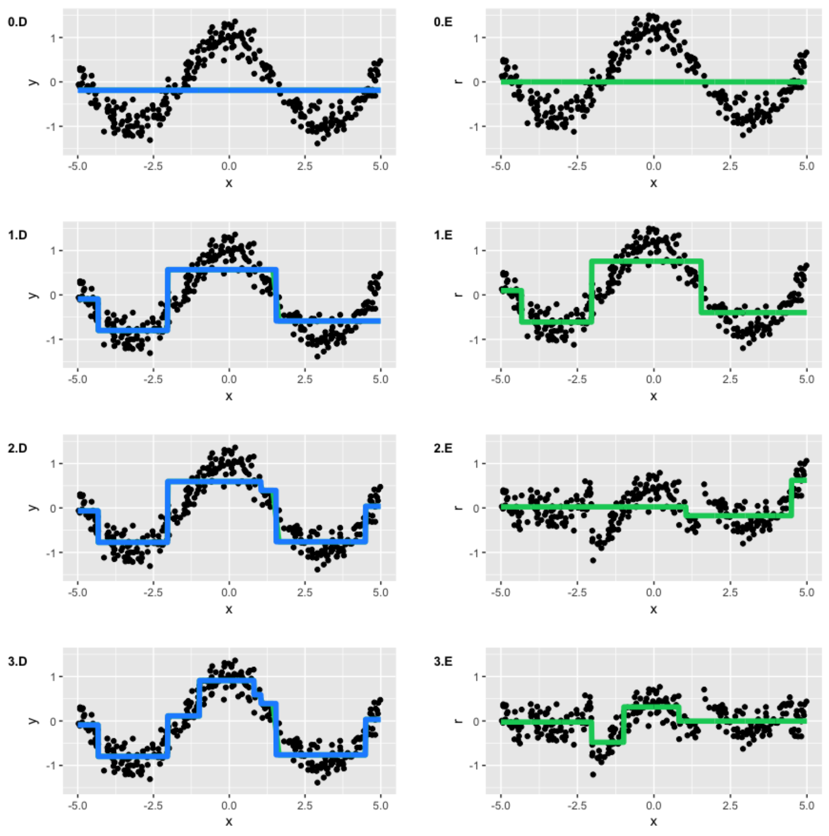


Figure V-5: Visualization of the iteration within the gradient boosting algorithm (Manokhina, 2018)

criterion is reach the algorithm is halted and the final model is produced, reading for testing data to be parsed to generate a prediction.

Gradient boosting is a common choice for real world applications as it provides a lot of flexibility in how the algorithm is implemented and allows for a lot of expandability. One of the issues that can be seen from the model is that of overfitting, usually a result of an excess of weak learners, though this can be minimized through the use of regularizes and depth constraints.

Artificial Neural Networks

Architecture

Perceptron are the simplest and oldest forms of ANNs and provide a simple way to understand the fundamentals that larger models build from. All ANNs are built from a series of inputs, weights, activation functions, and an output. Inputs can consist of both variable inputs as well as bias inputs, a constant input that allows for better fitting of a model. The activation function is a key part of the model and is the element that maps the linear inputs of input layers to a non-linear output, without this extra computation, the model would be a simple linear regression model. Activation functions will be discussed further in Activation Functions. A perceptron is built using only a single neuron, its key characteristic making is so simple. These are the most basic building blocks of a model; however, the power comes from changing the

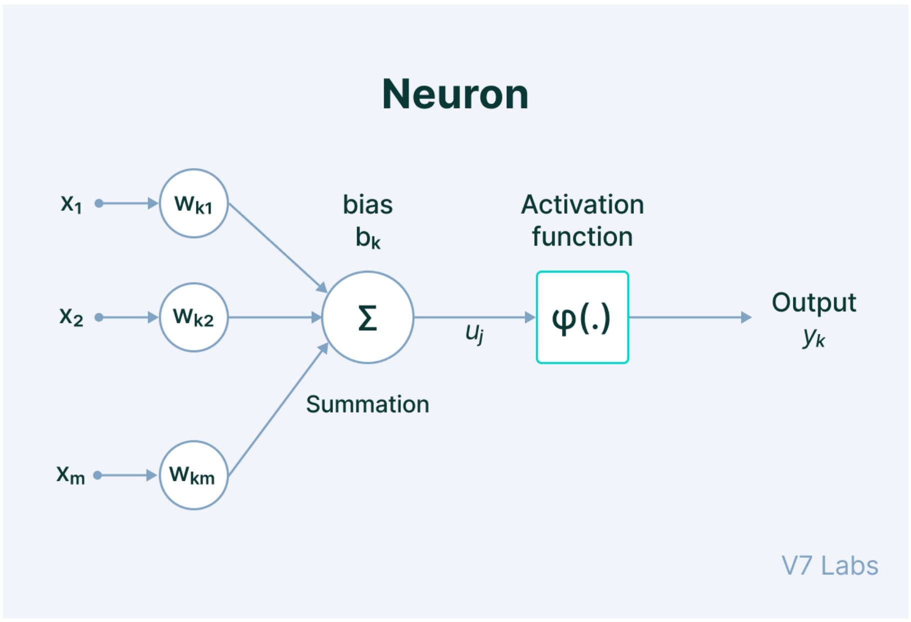


Figure V-6: Diagram of a basic Perceptron ANN (Baheti, 2021)

architecture and flow of data within the model. A simple view of this perceptron is shown by Figure V-6: Diagram of a basic Perceptron ANN below.

The principal elements of the architecture of an ANN are that of its layers. The layers are distinguished as input, hidden and output. The most important of which being the hidden layers, which are the defining feature of a given architecture. These hidden layers can be designed to provide different functionality and are classified by such. Dense layers are the simplest of these, performing simple linear transformations making them popular for simple classification problems. Feed-Forward models are easily conceptualised by considering multiple perceptrons fed into each other in rows and then stacked to create broader layers. All data flows through the system are a single forward direction, deriving the name. These feed forward neural networks can be further advanced by increasing the number of layers that comprise them, resulting in a Deep feedforward neural network DFNN.

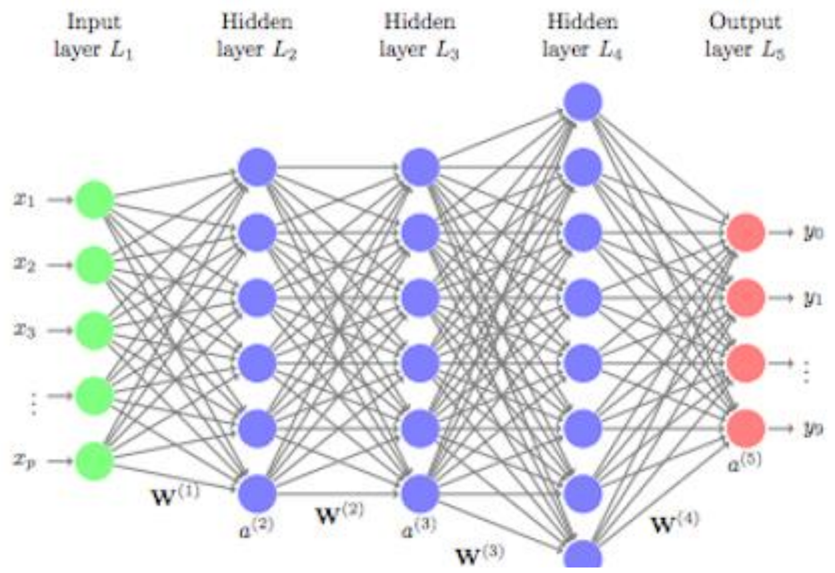


Figure V-7: Representation of layers within a feed forward NN (University of Cincinnati, n.d.)

Backpropagation

It is important to understand the processes that enables these models to become knowledgeable from simple data. The key operation is backpropagation. When the model initially runs, it is known as the forward pass, where weights are assigned at random across each node, and a prediction is made. This prediction is completely random and therefore useless in real terms. The second pass, however, is what makes these NNs powerful, the performance of each node for each training instance is assessed and the then weights and biases are automatically adjusted to improve the predictions accuracy accordingly. This is done by backpropagation. Backpropagation relies on a loss or cost function, which we will go in depth in later. Each forward iteration calculates the accuracy in terms of the loss function and then working backwards sequentially through layers, the gradient or derivative of the loss in respect to the weight of the network. This process is known as gradient descent backpropagation.

This can be very computationally intensive and so a technique known as mini batch gradient descent can be used to make the processing less computationally intensive. This involves taking a randomized subset of the data and doing both a forward and back pass to arrive at a cost function before updating the model's weights. This means that each pass does not see the entire training data, but iterations are much quicker and in a sufficiently large and varied dataset, the approximation can be very close to the true value that standard gradient boosting would have provided. There is a third gradient boosting technique, Stochastic gradient boosting however this is out of the scope for this project.

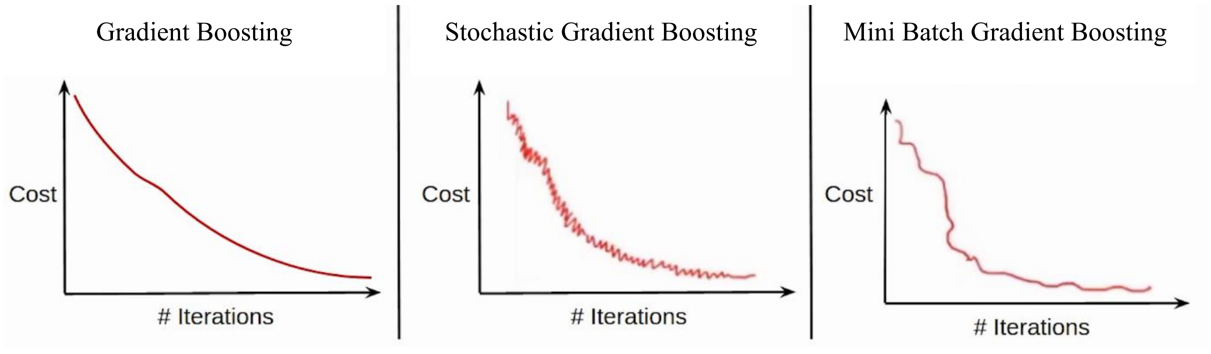


Figure V-11: Comparison of cost function for different gradient boosting techniques (Singh, 2023)

Activation Functions

Activation functions are one of the most important elements of a neural network as they are the final decider of the state of a neuron at each layer whilst adding non-linearity to the model. There are a variety of activation functions that can be selected, and each have their advantages and disadvantages. Sigmoid functions are a common implementation and are similar to the logistic regression discussed earlier in the Logistic Regression (LR) section, implementing the same curve. Sigmoid curves are generally used in the output layer of a NN as they can easily be defined to return binary outputs, aiding in classification problems. Another common activation function is that of the Tanh function.

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

The tanh activation function is generally preferred to the sigmoid function as the range produced maps to -1 to 1 allowing for data to be centred from its use. It is normally implemented in the hidden layers of a network.

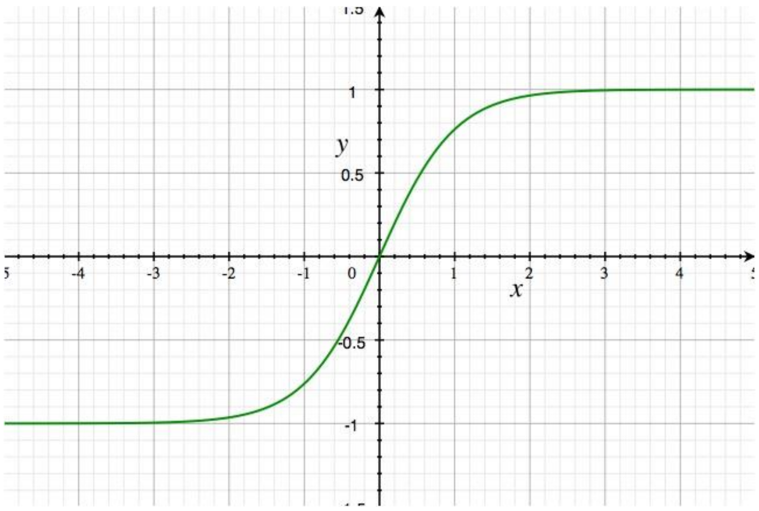


Figure V-12:tanh graph (Cowan, n.d.)

The most common activation function is the Rectified Linear Unit (RELU) function. It is defined as:

$$f(x) = \begin{cases} 0, & for\ \ x < 0 \\ x, & for\ x \geq 0 \end{cases}$$

The function is so popular given it is less computationally intensive that the activation functions previously discussed due to only a fewer number of neurons being activated during its use. However, there are some issues with the function, notably the problem of Dying ReLU which results in a gradient value of 0, leading to some neurons being returned as 0 immediately, reducing the overall efficacy of the training of the model. This can however be fixed using an activation function known as Leaky ReLU defined as:

$$f(x) = \begin{cases} 0.1x, & for\ \ x < 0 \\ \ x, & for\ x \geq 0 \end{cases}$$

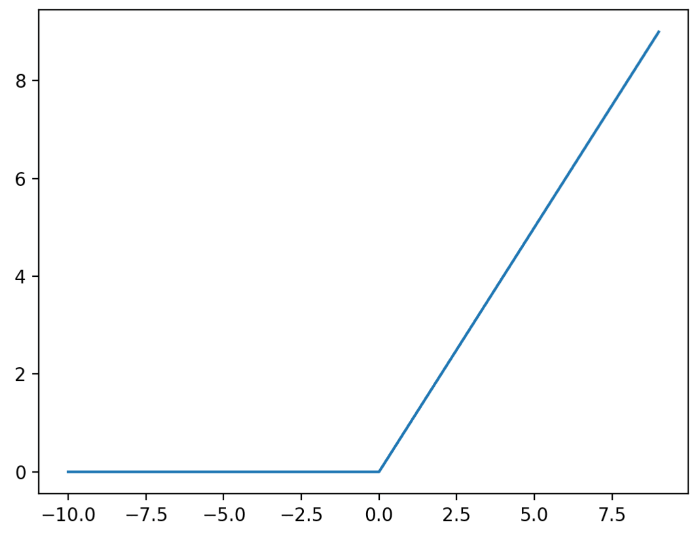


Figure V-13: ReLU activation function graph (Brownlee, 2020)

This change allows for values that would be 0, on the left side of the graph, during standard ReLU to be mapped to a lower value, but not to 0. It also allows for backpropagation with negative numbers.

Loss Function

Earlier the concept of loss functions was introduced, the process of calculating the accuracy of a model at the end of each iteration through the data, or a batch of the data. It is the measure of how much the prediction provided by the model and the expected outcome differ. The overarching goal of a NN model is to minimize this loss function. There are many different possible losses function that one can implement, however in this project we will look at two of the most common ones that are best suited to be implemented in a classification, specifically binary classification problem. The first of these is Binary Cross-Entropy, also known as log loss, defined by:

$$Log\ loss = \frac{1}{n}\sum_{i=1}^{n} -(y_i \times log(p_i) + (1 - y_i) \times log(1 - p_i))$$

Such that the probability of class 1 is (p_i) and class 0 is $(1 - p_i)$. It is a subset of the broader cross-entropy, which finds the probability distribution of losses. Therefore, it scales the negative score to how confident the model is in its incorrect predictions. It also penalizes the model for providing predictions that are correct but with low confidence. This loss function is best suited for high precision tasks, suiting it well for this project.

The flexibility of NNs mean there are many ways of optimizing the model in very specific ways. Given the exploratory nature of this project, the optimization techniques implemented will be discovered during the programming process and will be discussed in the Implementation section.

Tools and Technologies

This project is going to be written in Python as it is the industry standard for solving data science problems. It is also the language I feel most comfortable in and has extensive documentation and resources to aid the development process.

The other major benefit of using Python is that of its broad family of libraries that can be used to streamline much of the complexity of the models described. There are many options in this category but scikit-learn will be used throughout this module for both the machine learning models themselves and many of the utility and preprocessing libraries included. This is the most popular library of its kind and most tutorials on the subject implement their solutions with it.

For the Artificial Neural Network, scikit-learn could have been used as it has some specialised classes to implement such technologies. However, the learning curve and customization options of the models seemed to be lacking. Instead TensorFlow through Keras seemed like a much better option. For deep neural networks it seems to be the industry standard (Simplilearn, 2023). The addition of Keras as a high-level library running on top makes it very easy to implement within python. Whilst out of the scope of this project TensorFlow has many more options for creating very powerful highly performant systems which could be helpful for me to understand the library currently if the opportunity to use it professionally later became available.

There is also a whole list of standard libraries such as NumPy and Pandas that have been used but will not be discussed. A virtual environment was used whilst developing this project and therefore a copy of all installed libraries is available in the appendix.

VI. Implementation

Implementing the designed system to create a usable interface on trained models is the main objective of this project. The following chapter chronicles the entire process from data collection, pre-processing, the application of each model and building a user interface for stakeholders. Whilst 3rd-party libraries make this process much simpler than building from the ground up, there are notable roadblocks and logical problems encountered along the way.

Data Collection

Arguably the most important part of being successful when implementing an AI solution is having relevant and correct data. Having both a wide range of feature variables representing different financial metrics as well as a sufficiently deep database with enough companies as datapoints for accuracy to be built. To this end, the most exhausting free resource fitting these criteria that could be found was a dataset of Taiwanese companies from 1999 – 2009 discovered on Kaggle but owned and maintained by UCI Machine Learning Repository. This data is provided by the Taiwan Economy Journal, a reputable source for such information. This dataset is sufficiently broad as it contains 95 feature variables alongside a classification variable stating whether the company has gone bankrupt, as defined by the Taiwanese Stock Exchange. A list of all variables can be found below. The dataset contains a record for 6819 individual companies, which is broad enough to train a model of some accuracy with. Another benefit of this dataset is that it is licenced under Creative Commons 4.0 allowing for it to be used for any purpose. Whilst this data set is very detailed for the free se licence it has, an issue that will be discussed later in the evaluation section is that of the dataset just being too small compared to those being used in professional applications. This is due to there being a huge industry around data brokerages and financial data is one of the most difficult and expensive data to acquire and so is hidden behind very large paywalls. A link to the public download page for the dataset is available here, (UCI Machine Learning Repository, 2020).

Data Analysis

Now that the data has been acquired the process of Exploratory Data Analysis (EDA) begins, with the goal of better understanding key characteristics of the data and creating description statistics, visualisations, understanding correlation vs causation and data quality assessments.

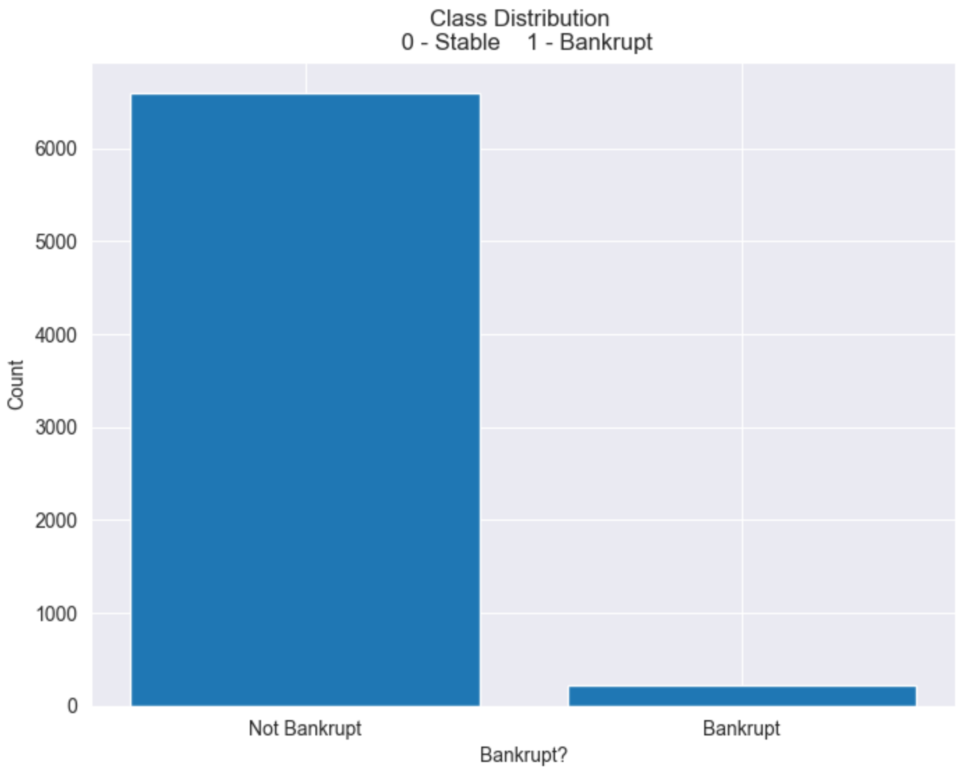


Figure VI-1: Class distribution of Bankrupt Companies

The first stage of this is checking that the data is fully present and clean. This is done by checking for null values, duplicate rows, and other erroneous data artifacts. From this the visualizations can be created. The first important information is that of the class distribution of our classifying variable "Bankrupt?" we can see a clear class imbalance here with only being 220 confirmed bankruptcies to 6599 being stable, a 96.77% to 3.23% ratio. This is something that during the feature engineering late could need to be corrected for to create a more accurate model at predicting both upside and downside potential. From this an understanding of the characteristics of the feature variables is good to understand, this was done using the inbuilt functionality that the pandas library offer, namely DataFrame.describe(), a table version of this can be seen at. From this we can see that there are many outlier values within the data, this will have to be rectified later in the implementation process.

For a better understanding of these outliers a function was created to use a simple quartile outlier algorithm to find the 15 variables with the most outliers and visualize these variables with a box plot. This can be seen in Figure VI-2: Box plots of broadest outlier variables and these outliers can clearly be seen to affect the data very negatable, with many variables having many values set to 0, when there is clearly an average they should be grouped around. There are many ways that outliers like this can be improved with, and for this project an isolation forest algorithm will be used to select and remove the outliers of the model. This is due to isolation forest algorithms do not rely on a normal distribution of data and use multi-dimensional analysis to look across multiple variables to decide on outlier variables and is often used in financial applications (Hyder & Naaz, 2019).

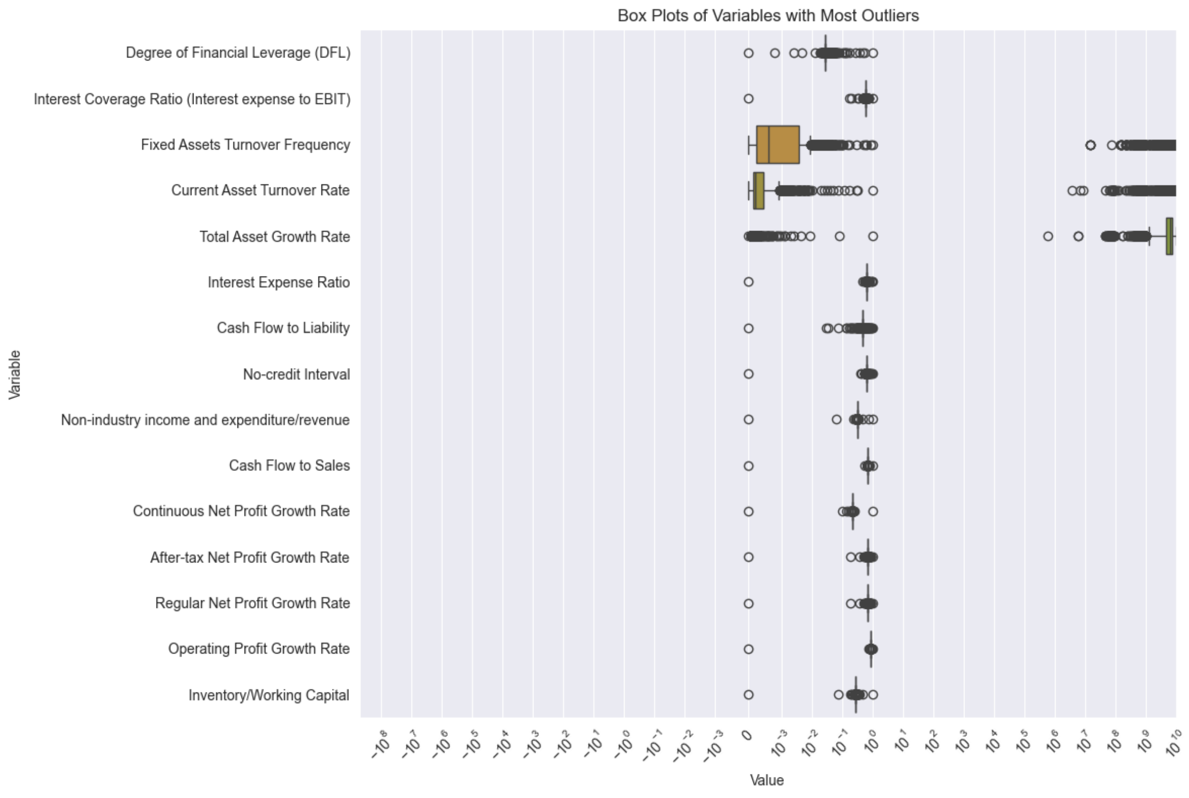


Figure VI-2: Box plots of broadest outlier variables

Figure VI-3: Pearson Correlation ranking for feature variables

Figure VI-4: Histogram for highest correlation variablesFigure VI-5: Pearson Correlation ranking for feature variables that no one variable is causing a huge amount of sway in the outcome of a company. This suggests that more complex models such as deep learning techniques might be better equipped to make predictions due to there being a simple correlation between variables and a more hidden relationship between variables will have impact. It is also worth noting that the majority of variables have an extremely low correlation value. Given there are sufficient variables in the dataset this could provide the opportunity to implement some feature selection. This would involve the removal of a portion of the variable that are being fed to the model to reduce the computation time required to train a model.

The final metric to understand within the data is that of its skew, which can lead to bias within a model and misleading evaluation metrics. We will look at the skew of the variables which have the highest correlation with the classification variable to get at a better understanding for those variables that will likely have the highest impact, this is seen in Figure VI-6: Histogram for highest correlation .

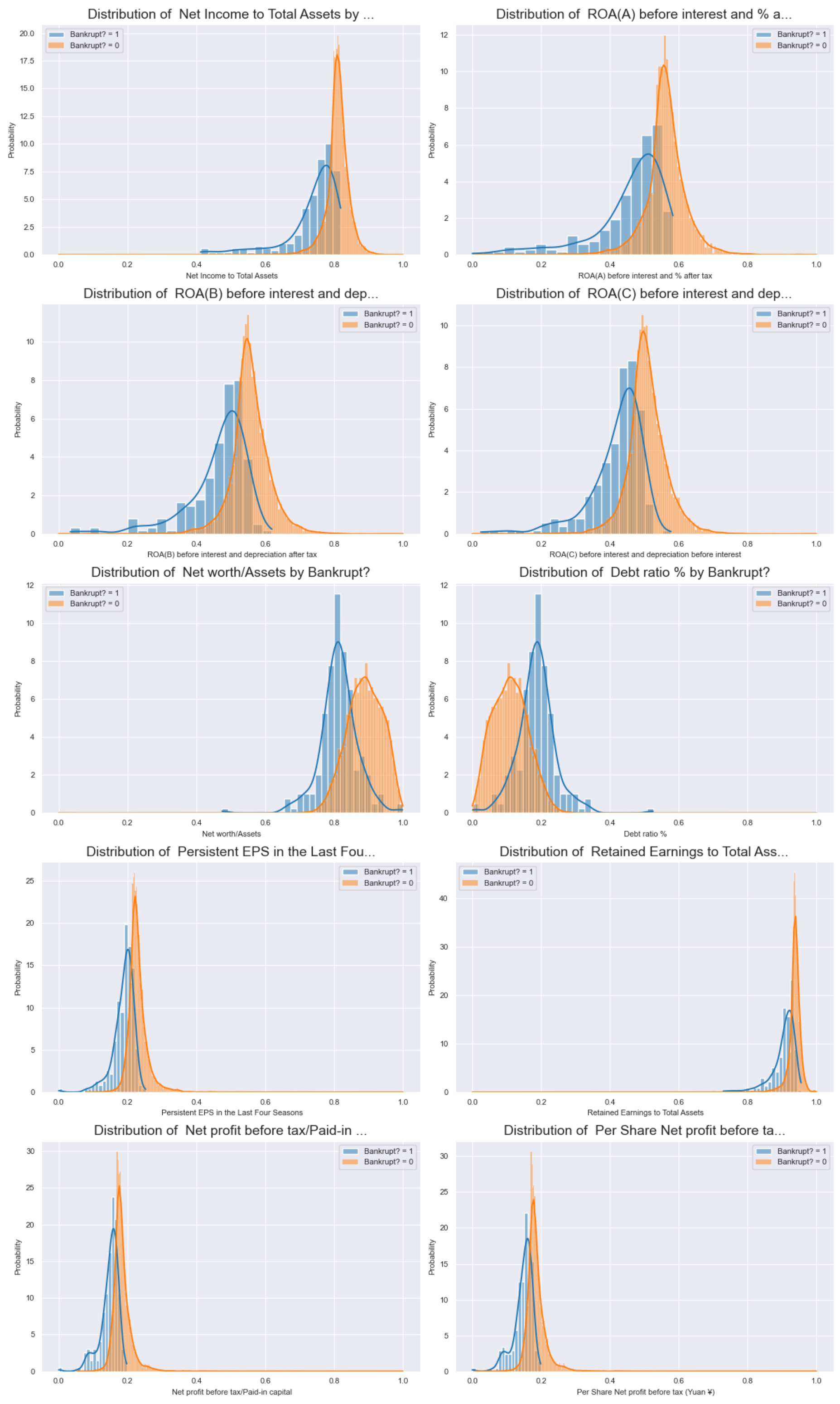


Figure VI-3: Histogram for highest correlation variables.

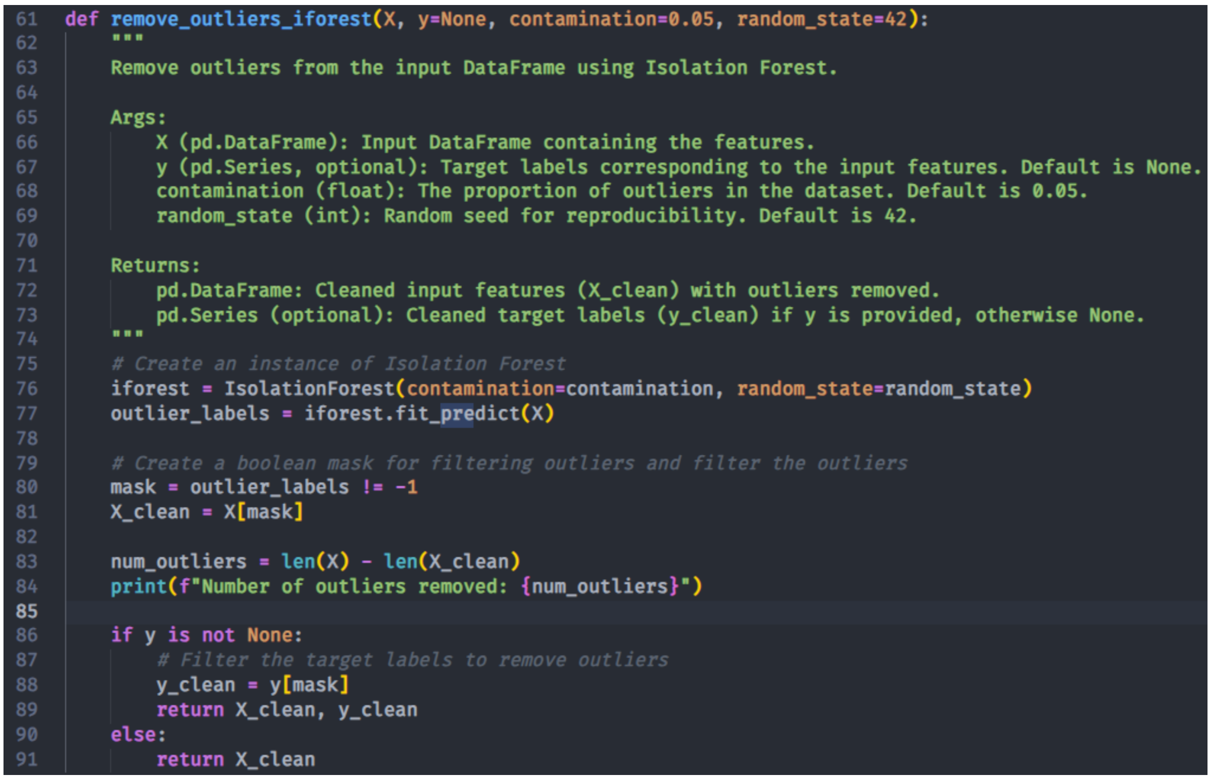
We can see from this that the majority of variables have a slightly more negative skew when the company is bankrupt, we can infer that this is generally due to the feature variables being a metric or ratio of finance, where a higher value represents more money that company can use for operations in some way. As none of the models that have been selected implement a linear selection skew should be less of an issue than if it was implemented in say a basic linear regression model. However, given the goal of making the model as performant as possible, from this knowledge I will be moving forward by applying an algorithm for skew correction.

Data Preparation

Now that the shape and characteristics of the data we have are known, manipulations on it to better the model are required. This data preparation is broken down into outlier removal, skew reduction, feature selection and class imbalance adjustment.

Outlier Removal

There are many approaches that can be used to implement outlier removal, from simple quartile-based removal to complex proprietary algorithms such as the Box Cox transformation. For this project the Isolation Forest is best suited due to it being regarded as the strongest algorithm for anomaly detection. It itself is actually an implementation of a machine learning algorithm, utilizing tree structures to randomly partition each data record, an ensemble of these trees finally being used to remove nodes which contain values of a high anomaly score. The key parameters for this function are that of contamination and n_estimators. Contamination refers to the expected number of results that are anomalous, it allows for background domain knowledge to be leveraged to better predict the removal rate. For this project given companies have very broad financial data that would like to be included in the models training, a value of 0.05 has been used, expecting 5% of values to be erroneous. The n_estimators value represents the number of trees in the ensemble and has been set to 50 for this project to create sufficient trees that are not too deep. With this configuration on average 350 records are removed due to outliers.

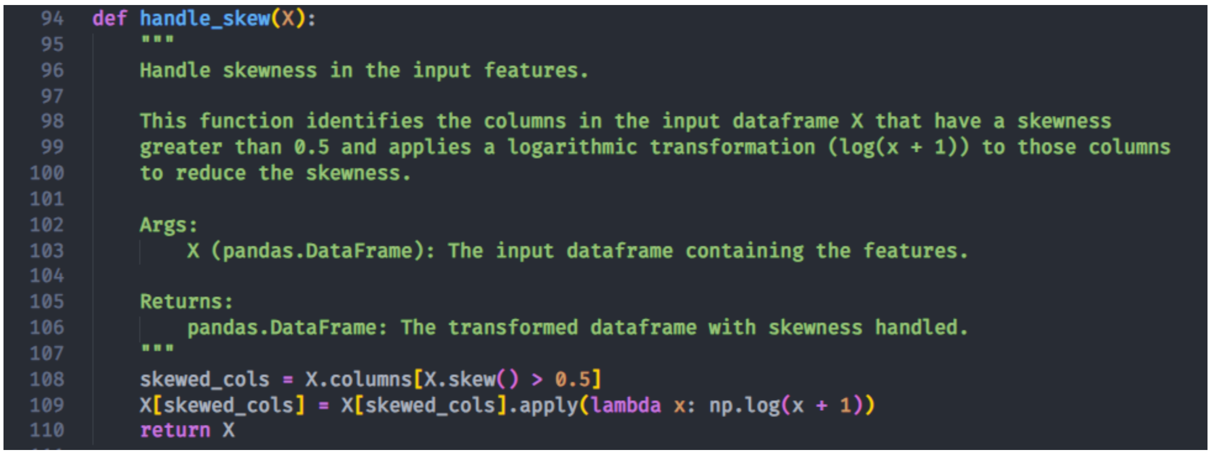


```python
def remove_outliers_iforest(X, y=None, contamination=0.05, random_state=42):
    """
    Remove outliers from the input DataFrame using Isolation Forest.

    Args:
        X (pd.DataFrame): Input DataFrame containing the features.
        y (pd.Series, optional): Target labels corresponding to the input features. Default is None.
        contamination (float): The proportion of outliers in the dataset. Default is 0.05.
        random_state (int): Random seed for reproducibility. Default is 42.

    Returns:
        pd.DataFrame: Cleaned input features (X_clean) with outliers removed.
        pd.Series (optional): Cleaned target labels (y_clean) if y is provided, otherwise None.
    """
    # Create an instance of Isolation Forest
    iforest = IsolationForest(contamination=contamination, random_state=random_state)
    outlier_labels = iforest.fit_predict(X)

    # Create a boolean mask for filtering outliers and filter the outliers
    mask = outlier_labels != -1
    X_clean = X[mask]

    num_outliers = len(X) - len(X_clean)
    print(f"Number of outliers removed: {num_outliers}")

    if y is not None:
        # Filter the target labels to remove outliers
        y_clean = y[mask]
        return X_clean, y_clean
    else:
        return X_clean
```

Figure VI-4: Function containing outlier removal code.

Skew Reduction

Skew reduction is quite simple to implement as there is built-in functionality in the pandas library to check for it. We collect those columns that contain skew and then use a lambda function to iteratively apply a log (x +1) function, applying a skew reduction.



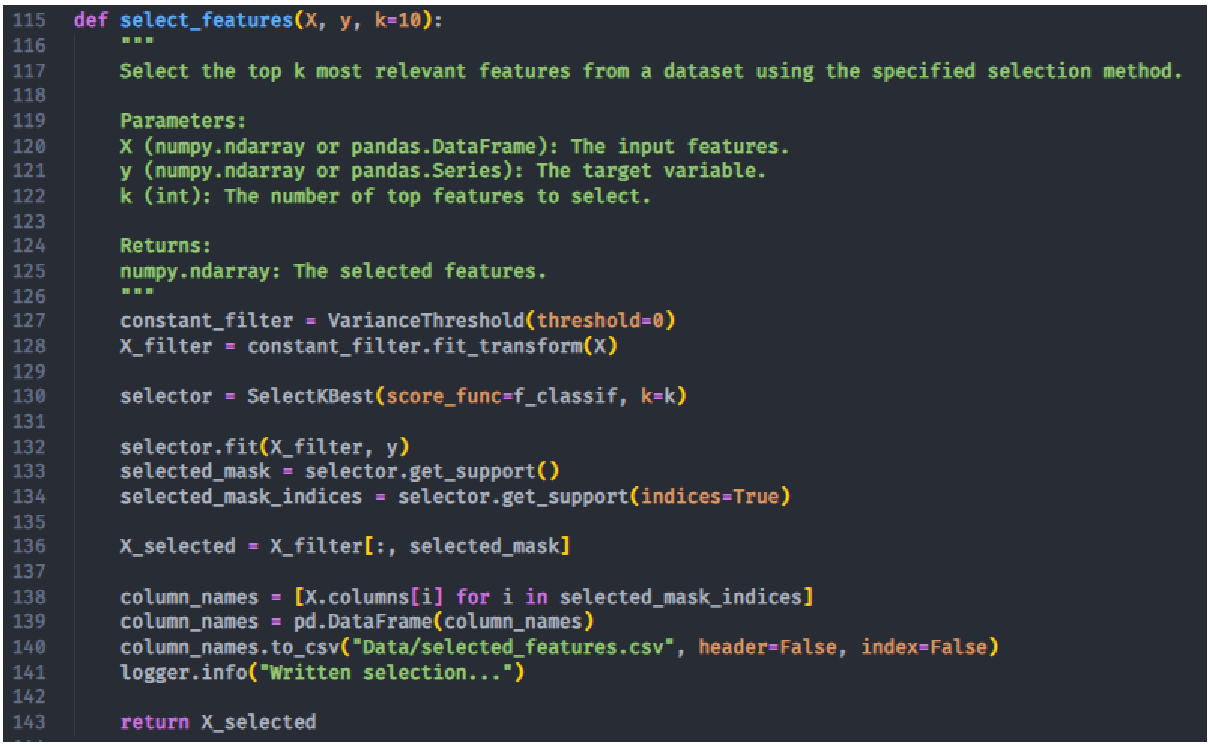
```python
def handle_skew(X):
    """
    Handle skewness in the input features.

    This function identifies the columns in the input dataframe X that have a skewness
    greater than 0.5 and applies a logarithmic transformation (log(x + 1)) to those columns
    to reduce the skewness.

    Args:
        X (pandas.DataFrame): The input dataframe containing the features.

    Returns:
        pandas.DataFrame: The transformed dataframe with skewness handled.
    """
    skewed_cols = X.columns[X.skew() > 0.5]
    X[skewed_cols] = X[skewed_cols].apply(lambda x: np.log(x + 1))
    return X
```

Figure VI-5: Function for skew reduction implementation.

Feature Selection

Selecting features can reduce the computation time whilst keeping the most important feature variables within a model. This is easily accomplished by using the SKLearn library and its SelectKBest and requires little configuration for the benefits it provides. It selects which variables are most appropriate from a configurable scoring algorithm and returns a mask to select the chosen feature variables with. The scoring function used is the Analysis of Variance (ANOVA) F-score, one of the most popular functions to use. This is due to it checking multiple groups concurrently to analysis if there is a relationship between them, reducing computation time.

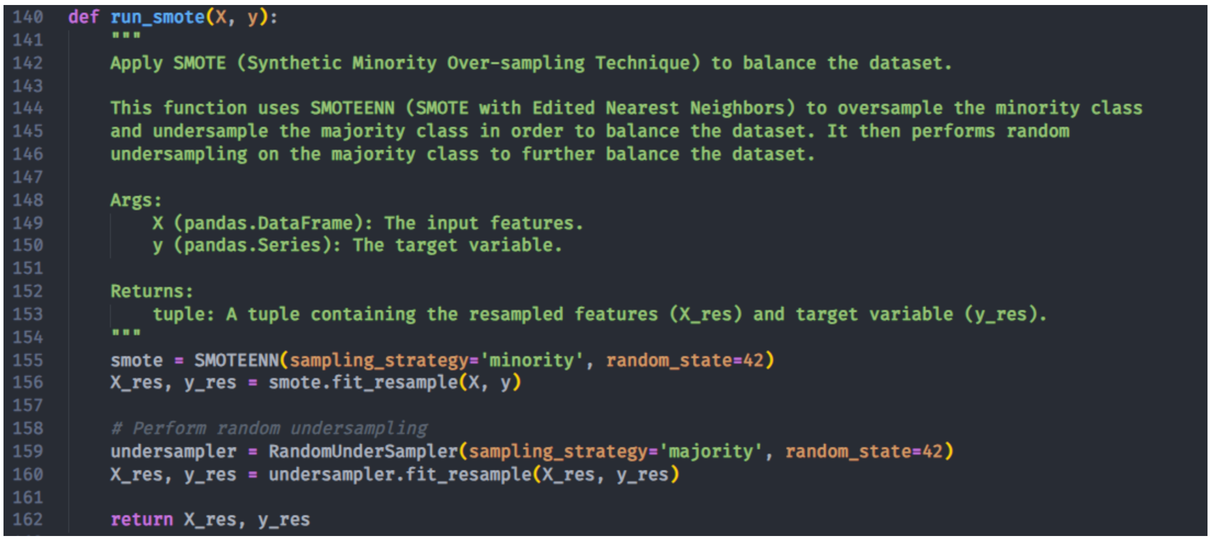
Later on, will be discussed making predictions on data that is input by the user, and this will require using the same selected feature variables as decided during the training data preprocessing. Therefore, a way of saving the feature variables selected was created. A csv file is written with the selected features as a list. When a new company is being tested this data is read and is used as a filter to remove the unnecessary variables in the tests data. This issue took longer than anticipated due to having to access each variable name from the original SelectKBest selector object.



```python
def select_features(X, y, k=10):
    """
    Select the top k most relevant features from a dataset using the specified selection method.

    Parameters:
    X (numpy.ndarray or pandas.DataFrame): The input features.
    y (numpy.ndarray or pandas.Series): The target variable.
    k (int): The number of top features to select.

    Returns:
    numpy.ndarray: The selected features.
    """
    constant_filter = VarianceThreshold(threshold=0)
    X_filter = constant_filter.fit_transform(X)

    selector = SelectKBest(score_func=f_classif, k=k)

    selector.fit(X_filter, y)
    selected_mask = selector.get_support()
    selected_mask_indices = selector.get_support(indices=True)

    X_selected = X_filter[:, selected_mask]

    column_names = [X.columns[i] for i in selected_mask_indices]
    column_names = pd.DataFrame(column_names)
    column_names.to_csv("Data/selected_features.csv", header=False, index=False)
    logger.info("Written selection...")

    return X_selected
```

Figure VI-6: Function to implement ANOVA feature selection.

Class Imbalance Reduction



```python
def run_smote(X, y):
    """
    Apply SMOTE (Synthetic Minority Over-sampling Technique) to balance the dataset.

    This function uses SMOTEENN (SMOTE with Edited Nearest Neighbors) to oversample the minority class
    and undersample the majority class in order to balance the dataset. It then performs random
    undersampling on the majority class to further balance the dataset.

    Args:
        X (pandas.DataFrame): The input features.
        y (pandas.Series): The target variable.

    Returns:
        tuple: A tuple containing the resampled features (X_res) and target variable (y_res).
    """
    smote = SMOTEENN(sampling_strategy='minority', random_state=42)
    X_res, y_res = smote.fit_resample(X, y)

    # Perform random undersampling
    undersampler = RandomUnderSampler(sampling_strategy='majority', random_state=42)
    X_res, y_res = undersampler.fit_resample(X_res, y_res)

    return X_res, y_res
```

Figure VI-7: Function to implement SMOTE for class imbalance.

One issue initially was that of implementing Synthetic Minority Over Sampling Technique (SMOTE). This procedure was chosen as it is the industry standard and was also implemented by some sources for which is being used for, allowing for a like to like comparison. However, it was soon found that when SMOTE was implemented, because the class imbalance in the classifier variable was so strong, there was not enough data to accurate make the synthetic data required to balance the classes. This resulted in the newly balanced classes having much lower quality data and the predictive ability of the models dropped significantly. To improve this, SMOTE with Edited Nearest Neighbors (SMOTEEN) was chosen as it completes majority under sampling to reduce the amount of oversampling that is required by the traditional SMOTE algorithm. This improved the model's performance whilst having a more balanced classification variable, however there was space for a separate call of standard random under sampling to create the ideal balance in classes.

Calculating Financial Ratios

As was discussed in Financial Distress Analysis background, the use of specific ratios in financial analysis has historically improved the findings of the work. To implement this in the project I have decided to calculate ratios that have historically been used in credit analysis and are regarded by many as the standard starting point for this analysis.

$$BV/TA = Book\ Value\ /\ Total\ Assets$$

$$CF/TA = Cash\ Flow\ /\ Total\ Assets$$

$$GOI/TA = Gross\ Operating\ Income\ /\ Total\ Assets$$

$$P/CF = Price\ /\ Cash\ Flow$$

Within the dataset that we are working with not all of these exact metrics are available as features. However, there are metrics that are very similar and so for this exercise it should be inconsequential, for example Gross Operating Income is unavailable, however, Operating Profit Per Share is available. These ratios are then added to the dataset and have the previous feature selection process applied.

Data Splitting

The final stage of data preparation is that of splitting it. In this case data is being split into 3 distinct sections: training, validation, and testing. Standard practice is to have a training/testing split of around 75/25. In this project as a validation stage is also being used a split of 70/20/10 has been chosen.

Machine Learning Models

Each of the models will be called from a run_models with each of the functions parsing taking in the splits of the data and then returning back the model object. This allows us to create a dictionary of models where the trained model can be called from a simple dictionary lookup. The implementation of each of these models is quite simple to implement, given the high-level nature of the SKLearn library and simply involve calling a single function. However, for each of these methods additional techniques to improve the performance of the models have been applied.

One of these techniques is hyperparameter tuning. For each of these models, research has been conducted into the optimum range of values for the most effective parameters to result in optimum performance by each. This was implemented throughout by SKLearn GridSearchCV allowing for multiple parameters and a range of values for each to be parsed, the model is then run using a smaller subset of the dataset with the different hyperparameter values combinations. The most performative model is then selected, and the entire dataset is applied to train and test the model. The pseudocode for how each model is run is as follows:

```
function run_single_model(dataset)
    parameter_grid = {p1:[v1,v2,v3], p2:[v1,v2,v3]}
    model = initialize_model()

    grid_search = gridSearchCV(parameter_grid, model)
    grid_search.fit_model(dataset)
    model_best = grid_search.best_model
    prediction = model_best.predict(dataset)

    print(prediction.results)
    return model_best
endfunction
```

The results of the optimum parameters from conduction hyperparameter tuning are seen below in a table for each model.

Logistic Regression (LR)

C	solver
0.1	Newton-cg
0.5	Lbfgs
1	liblinear
5	
10	

Gradient Boosting (GB)

Learning rate	Max depth	Max features	N estimators	Subsample
0.01	3	Sqrt	100	0.8
0.1	5	log	200	1.0
1	7		300	

Support Vector Machine (SVM)

C	Kernel	Gamma	Class weight
0.1	Linear	Scale	none
1	Rbf	auto	Balanced
10	sigmoid		

Random Forest (RF)

Class weight	Max depth	Min samples split	N estimators
None	None	2	100
Balance	5	5	200
	10	10	300

Neural Network

Given how broad the scope is for ways you can implement a MLP NN the first challenge was to decide on an architecture, namely how many layers, what type of layers and how many neurons per layer. A sequential model was chosen as it creates a simple stack of layers each in a simple feed-forward layout, one of the simpler options but suited well to MLP models.

One of the initial challenges faced in the development was setting up the input layer. Due to the feature selection and skew reduction the structure of the input dataset was subject to change. To counter this, a flexible input dimensions was required and numpy.shape() allowed for the input structure to dynamically change to the dimensions of the dataset on entry. The activation function used is PreLu, a modification of the LeakyReLu function discussed in Activation Functions that has a trainable $x < 0$ component allowing for a trainable gradient within that range. This is implemented so there is another trainable component of the model (Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification, 2015).

Dense layers were selected for the hidden layers of the model as their full connectedness allows for better understanding of hierarchical representations of data. We have 4 of these dense layers including input and output and 2 hidden layers. The dimensions of these layers increase then decrease, which means the computational load is not as high. Resulting in the number of parameters as seen in Figure VI-12: Parameters in in the model being very manageable:

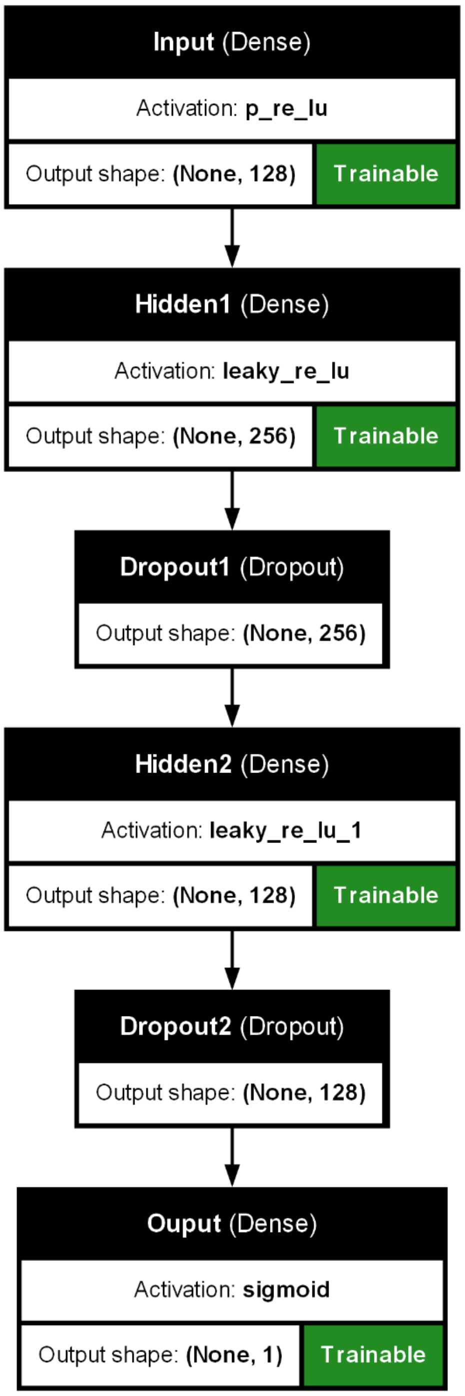
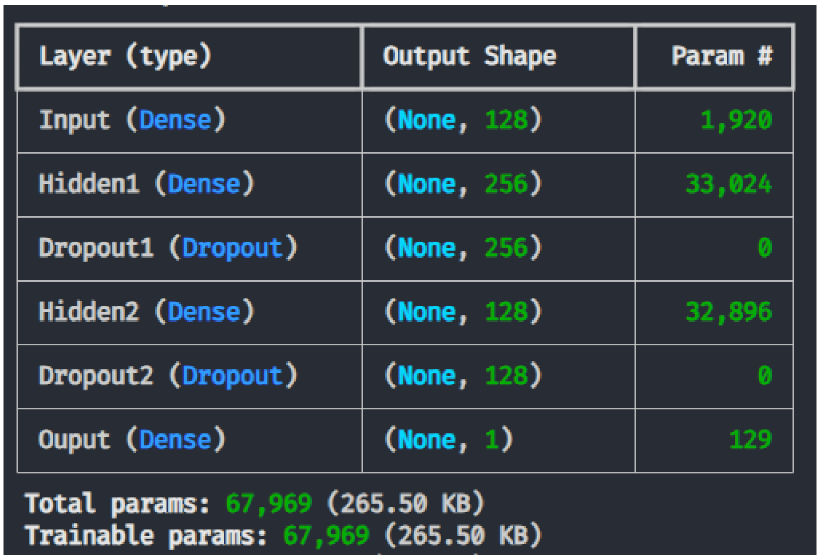


Figure VI-8: Chosen MLP architecture.



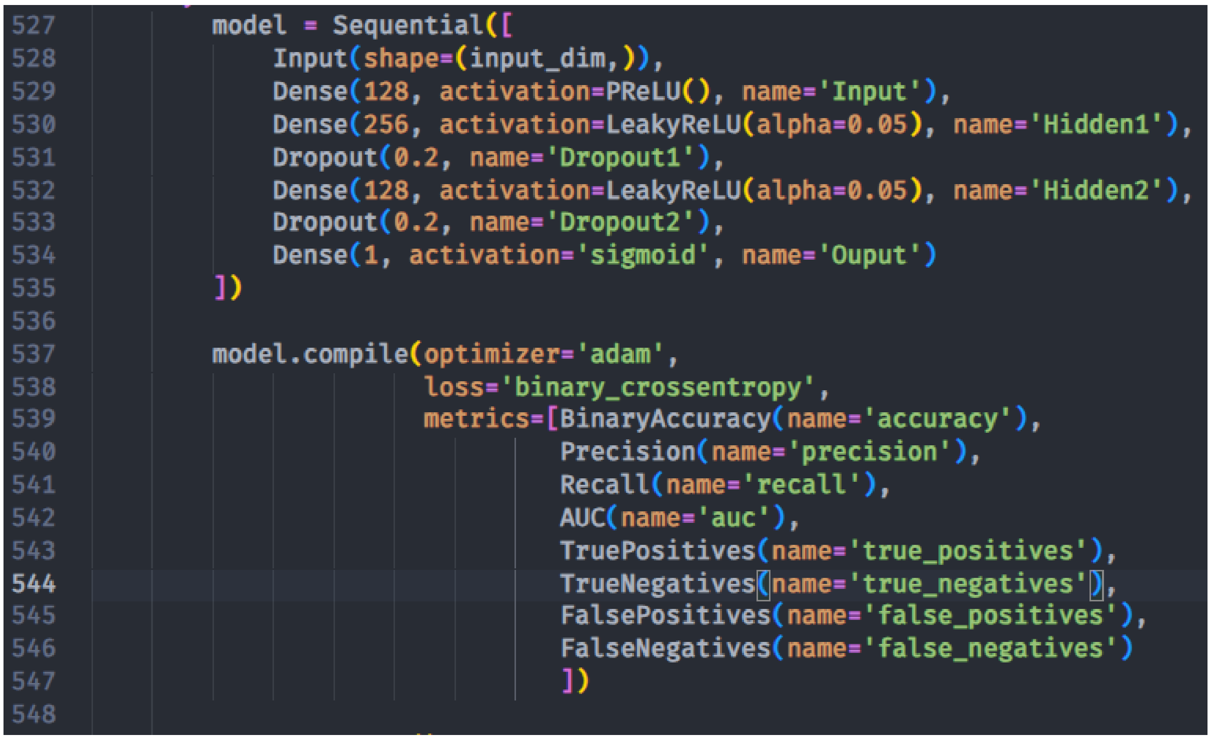
Layer (type)	Output Shape	Param #
Input (Dense)	(None, 128)	1,920
Hidden1 (Dense)	(None, 256)	33,024
Dropout1 (Dropout)	(None, 256)	0
Hidden2 (Dense)	(None, 128)	32,896
Dropout2 (Dropout)	(None, 128)	0
Ouput (Dense)	(None, 1)	129

```
Total params: 67,969 (265.50 KB)
Trainable params: 67,969 (265.50 KB)
```

Figure VI-9: Parameters in model.

These layers each implement the Leaky ReLu activation function as it has a faster computation time than both Sigmoid and PreLu. 2 dropout layers are used, at each layer 20% of the value is set to 0 to reduce overfitting. Some experimentation was done on the effectiveness of this technique and will be discussed in the evaluation. Finally, the output layer uses a Sigmoid activation function, this nicely maps the prediction for the model to a binary output representing a prediction of bankruptcy or not.

Throughout the model a loss function of binary cross entropy was used as discussed in Loss Function. This was a sensible choice as the algorithm is quite efficient whilst penalizing



```
model = Sequential([
    Input(shape=(input_dim,)),
    Dense(128, activation=PReLU(), name='Input'),
    Dense(256, activation=LeakyReLU(alpha=0.05), name='Hidden1'),
    Dropout(0.2, name='Dropout1'),
    Dense(128, activation=LeakyReLU(alpha=0.05), name='Hidden2'),
    Dropout(0.2, name='Dropout2'),
    Dense(1, activation='sigmoid', name='Ouput')
])

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=[BinaryAccuracy(name='accuracy'),
                       Precision(name='precision'),
                       Recall(name='recall'),
                       AUC(name='auc'),
                       TruePositives(name='true_positives'),
                       TrueNegatives(name='true_negatives'),
                       FalsePositives(name='false_positives'),
                       FalseNegatives(name='false_negatives')
                       ])
```

Figure VI-10: Code used to define Neural Network

miss classifications using its logarithmic nature. The code used to define the model is shown below in Figure VI-13: Code used to define Neural Network.

The model is then trained on the training data and a value of 5 for the number of epochs was selected as it prevents memorization of noise and additional computation time, whilst also seeing the data enough to gain learning insights. A custom callback function to print the progress of the models training was created for a nice user interface implementing lambda functions.

Making Predictions

After all models were implemented, it was time to create the user interface to allow stakeholders to use the trained models to gain predictive insights of companies outside of the initial database. This was done solely through a terminal interface for simplicity and relies on basic single character commands and filenames for simplicity. A company is initially loaded using the given file path from a user and the data is loaded into a DataFrame. For ease of use for the end user, a template csv file has been made with all of the headings that can be input.

An error kept occurring during testing that I was unsure of how to remedy, caused by numbers being infinite or too large for float32/64 types. Through more in depth testing it was found that the log function used to reduce the skew of the data during preprocessing was the cause of the issue. This meant that the function had to implement some kind of cap to stop the values reaching this limit. Luckily this was simple enough to remedy once the source of the issue was confirmed, as NumPy has a built in clip() function which creates an upper/lower limit that is assigned to the variable is it is seen to go about a defined threshold. This did also require changing the flow of the function as a lambda function could no longer be used. Instead, a simple loop was created to iterate over each of the columns and apply this this function. The rectified code can be seen below.

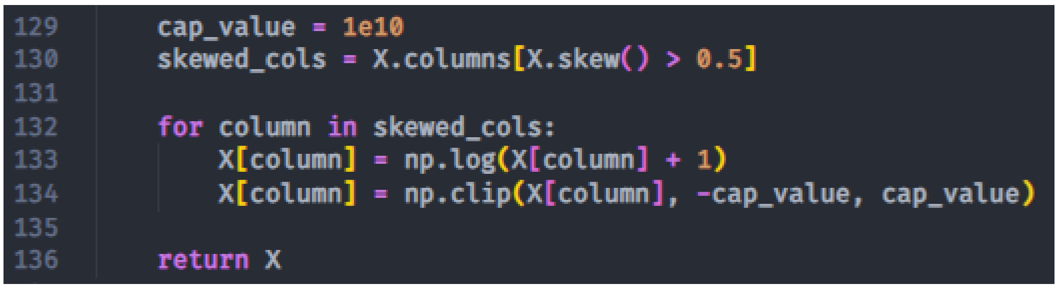
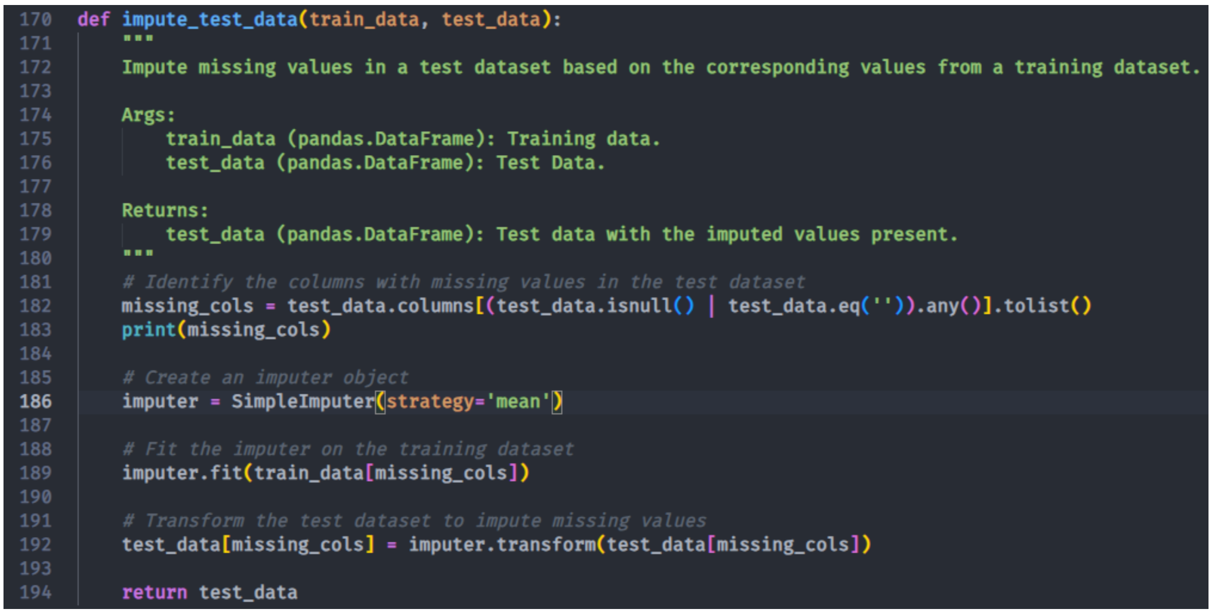

```
    cap_value = 1e10
    skewed_cols = X.columns[X.skew() > 0.5]

    for column in skewed_cols:
        X[column] = np.log(X[column] + 1)
        X[column] = np.clip(X[column], -cap_value, cap_value)

    return X
```

Figure VI-11: Update skew code to implement a limit.

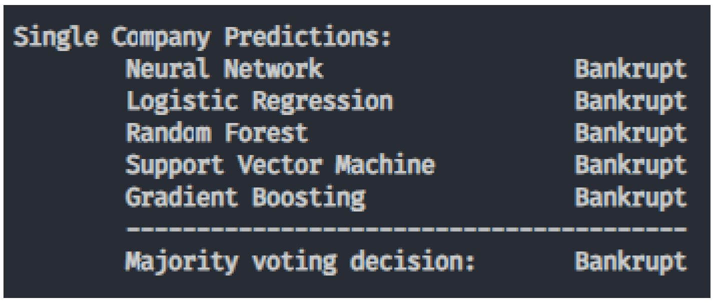
The user is then prompted if all values in the file have been added as it is understandable that some due diligence may not result in all company financial metrics being obtained. If this is true, then the value must be imputed so that there is not a missing feature variable when making a prediction with the trained models. This is done fairly simply by using the SKLearn SimpleImputer and parsing all empty or null columns in the test data. From this the mean average of each of those columns are used as values in the dataset. Whilst it may not be the most accurate, using a mean value of the dataset means that the variable should have little affect on the outcome of the model.



```python
def impute_test_data(train_data, test_data):
    """
    Impute missing values in a test dataset based on the corresponding values from a training dataset.

    Args:
        train_data (pandas.DataFrame): Training data.
        test_data (pandas.DataFrame): Test Data.

    Returns:
        test_data (pandas.DataFrame): Test data with the imputed values present.
    """
    # Identify the columns with missing values in the test dataset
    missing_cols = test_data.columns[(test_data.isnull() | test_data.eq('')).any()].tolist()
    print(missing_cols)

    # Create an imputer object
    imputer = SimpleImputer(strategy='mean')

    # Fit the imputer on the training dataset
    imputer.fit(train_data[missing_cols])

    # Transform the test dataset to impute missing values
    test_data[missing_cols] = imputer.transform(test_data[missing_cols])

    return test_data
```

Figure VI-12: Code of impute values function.

Once this is complete, a dictionary containing each of the trained models including the NN is iterated through. A model.predict() call is made and the result of each prediction is added to a new predictions dictionary. The result of each prediction is printed, and then majority voting is implemented to return the consensus decision of the model through finding the mode value in the predictions dictionary values. The whole terminal interface is running in a while True loop so that multiple individual companies can be analyzed if stored in separate files, however at this time only one company at a time is able to be parsed to the models from a file.



```
Single Company Predictions:
        Neural Network                  Bankrupt
        Logistic Regression             Bankrupt
        Random Forest                   Bankrupt
        Support Vector Machine          Bankrupt
        Gradient Boosting               Bankrupt
        ----------------------------------------
        Majority voting decision:       Bankrupt
```

Figure VI-13: Output from test prediction

VII. Results and Evaluation

The evaluation of the solution created here will be broken into two sections; a traditional analysis of the performance of the models created and how it performs given real world examples. This mirrors the joint aim of this project to better understand how different models perform in the given application of financial distress analysis and also the more product-based side of this, creating a tool for real analysis to be conducted with.

There are a well-documented set of standard practices on how to evaluate machine learning models. This is made easier by the nature of the models testing themselves during the training process, in order to train themselves. The metrics being used are as follows:

- Accuracy: Percentage of correct predictions.
- Precision: Accuracy in terms of true positives amongst all positive predictions, a quality metrics.
- Recall: Proportion of True positives within the whole system, a quantity metric.
- F1 Score: A mean average of precision and recall balancing the metric category.
- ROC/AUC: Measures across various threshold to have a rounded evaluation metric.
- Confusion Matrices: Summarised True/False Positive/Negatives in a simple matrix format.

These metrics were gathered but an interesting trend became apparent very early in the process. Various preprocessing steps had a massive impact on the performance of all models, affecting different metrics in different ways. The two key offenders were applying class rebalancing through SMOTE and feature selection. Notably when feature selection is not used and class rebalancing is applied, the models in general perform much worse than other combinations of these preprocessing steps. Analysis will be conducted on how the results differ before deciding on a preprocessing pipeline to be applied and then in more in-depth analysis will be conducted on the models using the standardized practice.

	No Smote No Feature	No Smote Yes Feature	Yes Smote No Feature	Yes Smote Yes Feature
Accuracy	0.96852	0.96868	0.71292	0.94362
Precision	0.96922	0.96922	0.80092	0.94116
Recall	0.9992	0.99936	0.81726	0.94732
F1	0.98396	0.98406	0.70854	0.94374
ROC AUC	0.7077	0.8643	0.82618	0.9835

Table 1: Average Model metrics by preprocessing technique

	No Smote No Feature	No Smote Yes Feature	Yes Smote No Feature	Yes Smote Yes Feature
Accuracy	0.96852	0.96868	0.71292	0.94362
Precision	0.96922	0.96922	0.80092	0.94116
Recall	0.9992	0.99936	0.81726	0.94732
F1	0.98396	0.98406	0.70854	0.94374
ROC AUC	0.7077	0.8643	0.82618	0.9835

Table 1: Average Model metrics by preprocessing technique shows the large differences in performance depending on the chosen preprocessing pipeline. It is important to note that accuracy is not always the best metric to analyse the performance with. This is the benefit of ROC/AUC, displaying how the model can distinguish between both positive and negative classes. This important especially in this dataset without the class rebalancing as the positive (bankrupt) class is severely underrepresented. This means the models could obtain a strong accuracy score by simply predicting every company will not go bankrupt. This can be seemed most obviously from the confusion matrix of the Neural Network with neither of the preprocessing steps applied. We can see that while it attempted to make a few predictions that showed it was bankrupt none of them were correct and the vast majority of tis guesses were that it is not bankrupt.

	True Positive	True Negative
Predicted Positive	0	42
Predicted Negative	0	1254

This issue seems to be remedied when we implement feature selection without class imbalance. Though when this is implemented the overall accuracy of the model struggles and there is still room for improvement in terms of predicting both positives and negatives. Moving on to implementing class rebalancing without feature selection we see this is the worst performing preprocessing pipeline by far. With an average accuracy of 71.3% and an ROC/AUC of 82.6% it is struggling to make predictions overall with any reasonable accuracy of both classification outcomes. This is likely caused by the SMOTE algorithm oversampling and under sampling the respective classes too much, however, to have a balanced dataset it is almost impossible to avoid this from happening.

Before moving on to analysing the best performing results from using both preprocessing steps is it important to note the recall value of most models across all previous preprocessing pipelines has been exactly one. This is important to note as it implies that the model never makes a false negative. A false negative being that it never guesses a company is bankrupt when it is actually financially stable. This shows very clearly the main issue being described that little attempt it made to guess that companies are bankrupt as the blanket assumption that they are financial stable.

	NN	LR	RF	SVM	GB
Accuracy	0.9031	0.9152	0.9833	0.9286	0.9879
Precision	0.8642	0.9202	0.9924	0.9319	0.9971
Recall	0.9541	0.9072	0.9738	0.9231	0.9784
F1	0.9069	0.9136	0.983	0.9275	0.9877
F2	0.9346	0.9098	0.9774	0.9249	0.9821
ROC AUC	0.9744	0.9724	0.9974	0.9752	0.9981

Table 2: Performance of models with full pre-processing pipeline

Table 2: Performance of models with full pre-processing pipeline, shows us the importance of using the preprocessing steps in the pipeline. Performance in all metrics is the strongest overall across all models. This eliminates the issue that using each preprocessing step individually brought up. Using feature selecting and class rebalancing allows the model to ignore those parameters that may have much less accuracy when they are oversampled which caused the issues discussed earlier. We can see that the accuracy of all of the models is superb with none of them falling below 90% accuracy and the ROC/AUC being extremely strong not falling below 97% on any of them and the confusion matrices of each of these models shows the accuracy of the model at predicting both positive and negative values. This is further

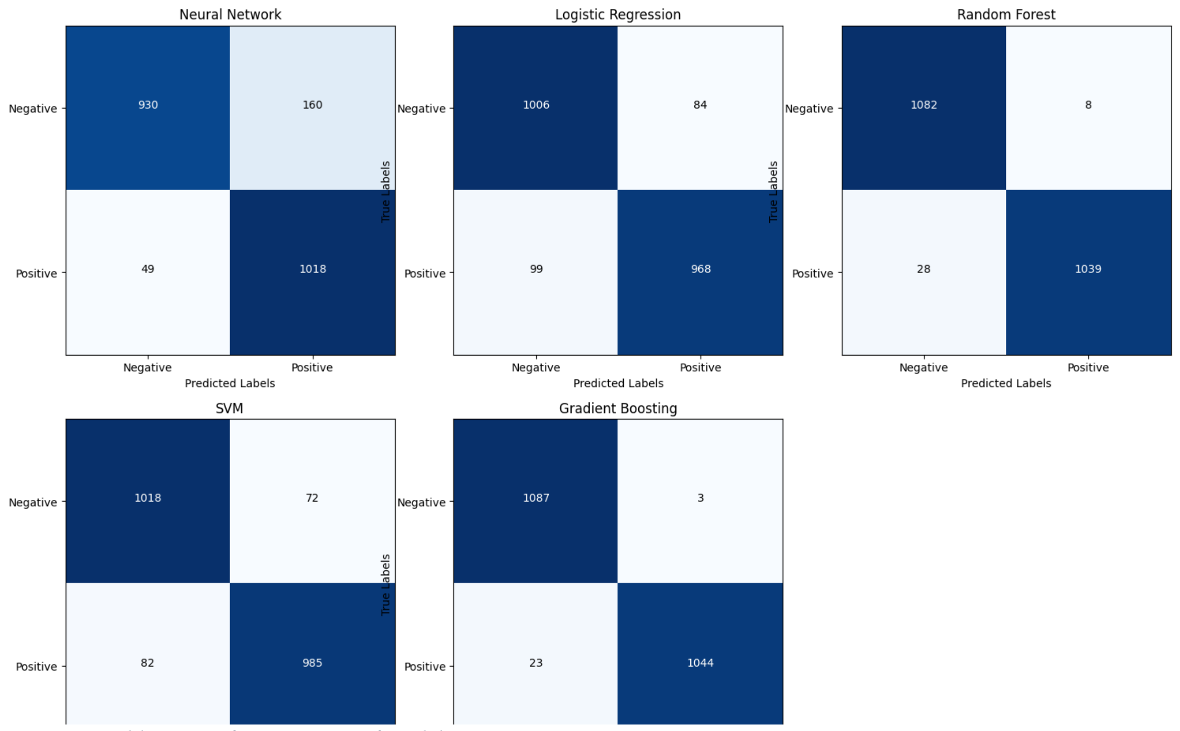


Table 3: Confusion matrix of models

supported by the fact that none of the models are showing a precision value of 1. Whilst the reduction would at first indicate that the performance is reduced, like in earlier discussion we know that this means that the model is attempting to make more predictions that companies are going to go bankrupt. From this we will be moving on with analysis from the results of the full preprocessing pipeline as it has clearly indicated that it performs the best across all metrics.

Now we can analyse the performance of each individual model. This will be conducted by analysing models across both the optimum preprocessing steps and with combinations of both. This is done as it provides the models with a more stressed environment so can better show how the models respond to less-than-optimal results. This is shown in the table below as the average metric for each model across the 4 possible preprocessing pipelines selected. The first point to note is how well both Random Forest and Gradient boosting perform across all pipelines. This could be due to a few factors, namely that both these models are ensemble methods, using an aggregate of simpler models to find the optimum solution. These models also inherently conduct feature scaling through splitting the features being used each time a branch is created and scoring the performance of that tree, due to this only the most effective features are used, just like implementing feature selection. Due to this the poor performance experience without feature selecting without SMOTE that brought down the average performance for all other models do not apply. These averaged results are seen in Table 4: Model metrics averaged across preprocessing .

	NN	LR	RF	SVM	GB
Accuracy	0.840475	0.8381	0.977725	0.8553	0.980575
Precision	0.828025	0.839325	0.98155	0.96675	0.985
Recall	0.987375	0.9768	0.988375	0.760725	0.99065
F1	0.887675	0.887175	0.984825	0.777975	0.987725
F2	0.939725	0.93265	0.9869	0.764525	0.989425
ROC AUC	0.692275	0.785425	0.962225	0.826475	0.9607

Table 4: Model metrics averaged across preprocessing pipelines.

To better understand how each model is performing in each metric, a ranked score was given to each of them, (lower being better) so their performance in relation to each other can be better understood. It is immediately obvious that both Random Forest and Gradient Boosting remain the most performant models, holding the best performance in every metric score. Then SVM, Neural Networks and Logistic Regression come in behind. This can be seen in Table 5: Ranked performance metrics.

	NN	LR	RF	SVM	GB
Accuracy	5	4	2	3	1
Precision	5	4	2	3	1
Recall	3	5	2	4	1
F1	5	4	2	3	1
ROC AUC	3	5	2	4	1
Summed Performance	21	22	10	17	5

Table 5: Ranked performance metrics

Overall, the models have performed very well considering the dataset that has been used for this project. Accuracy of over 90% in all models is extremely strong and shows the power of these techniques. It is also worth remembering that the models are being implemented with a majority voting system. Given this I believe that confidence can be placed on the collection of models even if one of them is less performant in specific circumstances.

The performance of the Neural Network being lower than the average of the other models was a surprising insight. Given it is seen as the most technical and sophisticated method there was an assumption in the development process that this would result in it being the most performant of the models. The opposite is true for Gradient Boosting and Random Forest as they are often seen as one of the simplest models in terms of the algorithm behind it.

One point of contention within the system that seems to be untestable in the current system is that of the impact of inflation and currency within the system. Given that the data is a snapshot in time of the company's financial metrics at the time of collection external economic factors such as exchange rates and inflation are not built into the system. This could lead to some issues if a user tried to conduct analysis on a company that is dollar denominated as some ratios could have different relationships. This is also true of inflation, as it impacts varying metrics in different ways leading to changes in the truth of the relationships explored by the models.

Initially a set of tests involving the use of external data from a source such as Yahoo Finance, a popular website for acquiring financial data, was planning to validate the models in a more holistic sense. This was found to be more difficult than initially anticipated as so many of the financial ratios and metrics that are used to test the data are not in fact available within the system. This makes it incredibly difficult to test the data with truly external and unseen data, separate from the test data from the original dataset split.

Sadly, given the time sensitive nature of this project there was not enough time to design a full user testing suite and gain approval for human testing. This would have been used to make sure that the user interface was strong enough to complete all the goals of the program. However, as this system is still very simple the user interface is not the limiting factor in my opinions and anecdotal opinions have backed up this case.

VIII. Conclusion

This project set out to better understand and implement Machine Learning techniques and models to predict the level of financial distress within companies and subsequently forecast the chances of bankruptcy. The objectives of building a preprocessing pipeline, designing, and training models and finally creating a system for users to interface with the trained models have all successfully been accomplished.

Implementations of Logistic Regression, Gradient Boosting, Random Forest, Support Vector Machines and a Neural Network were all successfully applied to the problem. Insight was gained on the differences in each of these models and how they perform for this given task. The results themselves were unexpected as it was discovered that the simpler models such as LR and GB are more performative than some more modern techniques, namely Neural Networks. The impact of the preprocessing stages of the project were discovered to have a major impact on the overall performance of some models but less so on others, leading to a learning outcome not expected from the inception of this idea. These steps were the key element of the system that allowed the models to differentiate between positive and negative classes. Without this the majority of the models defaulted to predicting the negative class due to class imbalance inherent in the dataset.

The performance of these models has shown to be very strong. A thorough test suite was used to analyse their This allows users to conduct their own research into companies to help aid their financial analysis. This was the primary goal of the project and shows the power of these technologies. The models averaged an accuracy of 94.3% whilst maintaining an ROC/AUC score of 98.6% showing a profound ability to make predictions accurately for companies of both binary outcomes. Whilst it was noted earlier that Neural Networks were not the strongest model, this could be due to the static, non-time series data that was used. The ability to recognise complex patterns within data could have been further leveraged by implementing a different architecture such as LTSM allowing for better predictive characteristics over different time horizons.

Finally, a user interface was developed to allow for others to use the system without a previous high-level knowledge of programming and computer science. This was achieved but the UI is not the standout feature of this program. There is much scope for improving this system and the lack of testing in this area means that could be an unknown weakness.

In summary this project accomplishes what it set out to achieve and created learning insights that can be applied to other areas of work and built upon to further improve the system.

Whilst there is space to improve the model a more detailed test suite involving better data sources and human analysis would be required to pinpoint those areas that are lacking and time restrictions within the project made this unfeasible.

IX. Future Work

Whilst almost all of the goals initially set out in this project have been accomplished there is still much room for improvement. The main issue would be that of data availability, a superior model could have been achieved by having a broader dataset. A dataset that contained records of companies from multiple countries could also provide a more rounded understanding of financial distress and could result in multiple trained models that could be selected to test data against the model that is most similar in geography and context to the data being evaluated. To further this, qualitive data could be implemented in some way and there has been much research in the benefit of this (van Auken, García-Pérez-de-Lema, & Madrid-Guijarro, 2011). This could be the quality of the senior management team which would be made available through some kind of web scraper of a website such as LinkedIn.

Another element that could receive more attention in a following iteration of the program is that of the imputer used to generate data from missing fields when prompting the models with test data. There would be two ways I suggest this is done. The simpler method would be to use a higher-level Imputer, such as one based on a K-Nearest Neighbour algorithm. This would generate values derived from multivariate data rather than simply finding the mean of each feature. This would allow for context of that specific company's financial performance to be used rather than basing the decision on the dataset as a whole. The more complex improvement that could be made would be to create an automated system to retrain the models using on the features that are available from the test data. It is unknown which of these methods would produce a better outcome, so more research is required in this space.

The voting system used to collate the predictions of each of these models. Firstly, a decision early in the development process to only product binary classifications from the models could be altered to create a stronger voting system. A value representing the confidence of the prediction returned from each model would allow for nuance on each prediction. As it currently stands the binary prediction of each model is a limiting factor to the ensemble system. A further improvement would be to using a weighted voting system based on test metrics on how accurate each model is. This would stop weaker models from having to large an impact on the final returned system.

Overall, I think that the best improvement to the system would be to use a time series dataset that could be constantly updated to consider the most modern relationships between financial ratios and financial distress. This would also allow for more complex neural network architectures such as RNNs, LTSMs and Encoder-Decoder models. These models are more powerful than the MLP implemented in this system. A further benefit of this would be to allow for users to make predictions across multiple time horizons to gain a rounded understanding of the future health of the company.

X. <u>Reflection</u>

Having completed this project, it has been an incredible opportunity to learn about a field I had little experience in before, applied artificial intelligence. Having taken an optional module on Artificial Intelligence I was looking forward to expanding on this knowledge. This started from day one, from researching which models would be most appropriate to understanding the background on the contextual knowledge needed to produce a strong real-world system. This was important as all of my previous education on the topics had been from a theoretical perspective. Being able to apply what I had learned was a great privilege.

It was also a difficult learning experience as this field of computer science is constantly changing. Finding sources that are trustworthy but also sufficiently detailed to conduct my own learning took time. I found the theoretical background allowed me to understand the logical reasoning of why the models worked, but when it came time to write the code putting them into action I struggled at time. The time taken to find these informative resources and applying myself to learn the techniques sadly took much more time than expected and reduced the overall amount of time available to implement the system being planned. If I were to complete this or a similar project, I would feel comfortable moving into the development stage much faster, ideally allowing for a much broader and more refined system to be created.

The libraries of Scikit-Learn and TensorFlow definitely made this easier as they are both high level implementations, however techniques such as hyperparameter tuning and understanding what architecture to implement for the Neural Network itself was far more difficult. The architecture of the system itself and the decision that went along with that were by far the most difficult elements. Which preprocessing steps would be most suitable, what activation functions for each layer in the NN and how to summarize the results from each model were difficult endeavours.

It also dawned on me the difficulty of building a system around financial data. The real-world implication of the decision that the system makes puts more pressure on the developer to strive for the highest accuracy possible as the downside is known to be far more impactful than other real-world implementations. This was balanced by the fascinating opportunity I learned this field has. I believe this area of applied data science is going to be at the forefront of the improvement across many industries as the possibilities are more widely understood. I feel very appreciative that I have had the time and space to understand this space with reasonable detailed and it has motivated me to pursue further research and potentially a career in the field.

I think the main aspect that let down this project on the whole is that of its testing system. This was partly due to the dataset used and the time constrains of the project. If this project was to be re done, the testing suite would be one of the first things considered and planned in detail. This would allow me to have much more confidence in what has been produced and make it more trustworthy to the stakeholders it was built for. That being said, there were great insights to be gained from the limited tests conducted. I was very surprised by the performance of some of the simpler models that were developed, and it made me reevaluate how I see the pros and cons of all forms of algorithm and model.

XI. Appendices

Full table produced from exploratory data analysis.

	count	mean	std	min	25%	50%	75%	max
Bankrupt?	6819	0.032263	0.17671	0	0	0	0	1
ROA(C) before interest and depreciation before interest	6819	0.50518	0.060686	0	0.476527	0.502706	0.535563	1
ROA(A) before interest and % after tax	6819	0.558625	0.06562	0	0.535543	0.559802	0.589157	1
ROA(B) before interest and depreciation after tax	6819	0.553589	0.061595	0	0.527277	0.552278	0.584105	1
Operating Gross Margin	6819	0.607948	0.016934	0	0.600445	0.605997	0.613914	1
Realized Sales Gross Margin	6819	0.607929	0.016916	0	0.600434	0.605976	0.613842	1
Operating Profit Rate	6819	0.998755	0.01301	0	0.998969	0.999022	0.999095	1
Pre-tax net Interest Rate	6819	0.79719	0.012869	0	0.797386	0.797464	0.797579	1
After-tax net Interest Rate	6819	0.809084	0.013601	0	0.809312	0.809375	0.809469	1
Non-industry income and expenditure/revenue	6819	0.303623	0.011163	0	0.303466	0.303525	0.303585	1
Continuous interest rate (after tax)	6819	0.781381	0.012679	0	0.781567	0.781635	0.781735	1
Operating Expense Rate	6819	2E+09	3.24E+09	0	0.000157	0.000278	4.15E+09	9.99E+09
Research and development expense rate	6819	1.95E+09	2.6E+09	0	0.000128	5.09E+08	3.45E+09	9.98E+09
Cash flow rate	6819	0.467431	0.017036	0	0.461558	0.46508	0.471004	1
Interest-bearing debt interest rate	6819	16448013	1.08E+08	0	0.000203	0.000321	0.000533	9.9E+08
Tax rate (A)	6819	0.115001	0.138667	0	0	0.073489	0.205841	1
Net Value Per Share (B)	6819	0.190661	0.03339	0	0.173613	0.1844	0.19957	1
Net Value Per Share (A)	6819	0.190633	0.033474	0	0.173613	0.1844	0.19957	1
Net Value Per Share (C)	6819	0.190672	0.03348	0	0.173676	0.1844	0.199612	1
Persistent EPS in the Last Four Seasons	6819	0.228813	0.033263	0	0.214711	0.224544	0.23882	1

Cash Flow Per Share	6819	0.323482	0.017611	0	0.317748	0.322487	0.328623	1
Revenue Per Share (Yuan Â¥)	6819	1328641	51707090	0	0.015631	0.027376	0.046357	3.02E+09
Operating Profit Per Share (Yuan Â¥)	6819	0.109091	0.027942	0	0.096083	0.104226	0.116155	1
Per Share Net profit before tax (Yuan Â¥)	6819	0.184361	0.03318	0	0.17037	0.179709	0.193493	1
Realized Sales Gross Profit Growth Rate	6819	0.022408	0.012079	0	0.022065	0.022102	0.022153	1
Operating Profit Growth Rate	6819	0.84798	0.010752	0	0.847984	0.848044	0.848123	1
After-tax Net Profit Growth Rate	6819	0.689146	0.013853	0	0.68927	0.689439	0.689647	1
Regular Net Profit Growth Rate	6819	0.68915	0.01391	0	0.68927	0.689439	0.689647	1
Continuous Net Profit Growth Rate	6819	0.217639	0.010063	0	0.21758	0.217598	0.217622	1
Total Asset Growth Rate	6819	5.51E+09	2.9E+09	0	4.86E+09	6.4E+09	7.39E+09	9.99E+09
Net Value Growth Rate	6819	1566212	1.14E+08	0	0.000441	0.000462	0.000499	9.33E+09
Total Asset Return Growth Rate Ratio	6819	0.264248	0.009634	0	0.263759	0.26405	0.264388	1
Cash Reinvestment %	6819	0.379677	0.020737	0	0.374749	0.380425	0.386731	1
Current Ratio	6819	403285	33302156	0	0.007555	0.010587	0.01627	2.75E+09
Quick Ratio	6819	8376595	2.45E+08	0	0.004726	0.007412	0.012249	9.23E+09
Interest Expense Ratio	6819	0.630991	0.011238	0	0.630612	0.630698	0.631125	1
Total debt/Total net worth	6819	4416337	1.68E+08	0	0.003007	0.005546	0.009273	9.94E+09
Debt ratio %	6819	0.113177	0.05392	0	0.072891	0.111407	0.148804	1
Net worth/Assets	6819	0.886823	0.05392	0	0.851196	0.888593	0.927109	1
Long-term fund suitability ratio (A)	6819	0.008783	0.028153	0	0.005244	0.005665	0.006847	1
Borrowing dependency	6819	0.374654	0.016286	0	0.370168	0.372624	0.376271	1
Contingent liabilities/Net worth	6819	0.005968	0.012188	0	0.005366	0.005366	0.005764	1
Operating profit/Paid-in capital	6819	0.108977	0.027782	0	0.096105	0.104133	0.115927	1
Net profit before tax/Paid-in capital	6819	0.182715	0.030785	0	0.169376	0.178456	0.191607	1
Inventory and accounts receivable/Net value	6819	0.402459	0.013324	0	0.397403	0.400131	0.404551	1
Total Asset Turnover	6819	0.141606	0.101145	0	0.076462	0.118441	0.176912	1
Accounts Receivable Turnover	6819	12789705	2.78E+08	0	0.00071	0.000968	0.001455	9.74E+09

Average Collection Days	6819	9826221	2.56E+08	0	0.004387	0.006573	0.008973	9.73E+09
Inventory Turnover Rate (times)	6819	2.15E+09	3.25E+09	0	0.000173	0.000765	4.62E+09	9.99E+09
Fixed Assets Turnover Frequency	6819	1.01E+09	2.48E+09	0	0.000233	0.000593	0.003652	9.99E+09
Net Worth Turnover Rate (times)	6819	0.038595	0.03668	0	0.021774	0.029516	0.042903	1
Revenue per person	6819	2325854	1.37E+08	0	0.010433	0.018616	0.035855	8.81E+09
Operating profit per person	6819	0.400671	0.03272	0	0.392438	0.395898	0.401851	1
Allocation rate per person	6819	11255785	2.95E+08	0	0.004121	0.007844	0.01502	9.57E+09
Working Capital to Total Assets	6819	0.814125	0.059054	0	0.774309	0.810275	0.850383	1
Quick Assets/Total Assets	6819	0.400132	0.201998	0	0.241973	0.386451	0.540594	1
Current Assets/Total Assets	6819	0.522273	0.218112	0	0.352845	0.51483	0.689051	1
Cash/Total Assets	6819	0.124095	0.139251	0	0.033543	0.074887	0.161073	1
Quick Assets/Current Liability	6819	3592902	1.72E+08	0	0.00524	0.007909	0.012951	8.82E+09
Cash/Current Liability	6819	37159994	5.1E+08	0	0.001973	0.004904	0.012806	9.65E+09
Current Liability to Assets	6819	0.090673	0.05029	0	0.053301	0.082705	0.119523	1
Operating Funds to Liability	6819	0.353828	0.035147	0	0.341023	0.348597	0.360915	1
Inventory/Working Capital	6819	0.277395	0.010469	0	0.277034	0.277178	0.277429	1
Inventory/Current Liability	6819	55806805	5.82E+08	0	0.003163	0.006497	0.011147	9.91E+09
Current Liabilities/Liability	6819	0.761599	0.206677	0	0.626981	0.806881	0.942027	1
Working Capital/Equity	6819	0.735817	0.011678	0	0.733612	0.736013	0.73856	1
Current Liabilities/Equity	6819	0.33141	0.013488	0	0.328096	0.329685	0.332322	1
Long-term Liability to Current Assets	6819	54160038	5.7E+08	0	0	0.001975	0.009006	9.54E+09
Retained Earnings to Total Assets	6819	0.934733	0.025564	0	0.931097	0.937672	0.944811	1
Total income/Total expense	6819	0.002549	0.012093	0	0.002236	0.002336	0.002492	1
Total expense/Assets	6819	0.029184	0.027149	0	0.014567	0.022674	0.03593	1
Current Asset Turnover Rate	6819	1.2E+09	2.82E+09	0	0.000146	0.000199	0.000453	1E+10
Quick Asset Turnover Rate	6819	2.16E+09	3.37E+09	0	0.000142	0.000225	4.9E+09	1E+10

Working capital Turnover Rate	6819	0.594006	0.008959	0	0.593934	0.593963	0.594002	1
Cash Turnover Rate	6819	2.47E+09	2.94E+09	0	0.000274	1.08E+09	4.51E+09	1E+10
Cash Flow to Sales	6819	0.671531	0.009341	0	0.671565	0.671574	0.671587	1
Fixed Assets to Assets	6819	1220121	1.01E+08	0	0.08536	0.196881	0.3722	8.32E+09
Current Liability to Liability	6819	0.761599	0.206677	0	0.626981	0.806881	0.942027	1
Current Liability to Equity	6819	0.33141	0.013488	0	0.328096	0.329685	0.332322	1
Equity to Long-term Liability	6819	0.115645	0.019529	0	0.110933	0.11234	0.117106	1
Cash Flow to Total Assets	6819	0.649731	0.047372	0	0.633265	0.645366	0.663062	1
Cash Flow to Liability	6819	0.461849	0.029943	0	0.457116	0.45975	0.464236	1
CFO to Assets	6819	0.593415	0.058561	0	0.565987	0.593266	0.624769	1
Cash Flow to Equity	6819	0.315582	0.012961	0	0.312995	0.314953	0.317707	1
Current Liability to Current Assets	6819	0.031506	0.030845	0	0.018034	0.027597	0.038375	1
Liability-Assets Flag	6819	0.001173	0.034234	0	0	0	0	1
Net Income to Total Assets	6819	0.80776	0.040332	0	0.79675	0.810619	0.826455	1
Total assets to GNP price	6819	18629418	3.76E+08	0	0.000904	0.002085	0.00527	9.82E+09
No-credit Interval	6819	0.623915	0.01229	0	0.623636	0.623879	0.624168	1
Gross Profit to Sales	6819	0.607946	0.016934	0	0.600443	0.605998	0.613913	1
Net Income to Stockholder's Equity	6819	0.840402	0.014523	0	0.840115	0.841179	0.842357	1
Liability to Equity	6819	0.280365	0.014463	0	0.276944	0.278778	0.281449	1
Degree of Financial Leverage (DFL)	6819	0.027541	0.015668	0	0.026791	0.026808	0.026913	1
Interest Coverage Ratio (Interest expense to EBIT)	6819	0.565358	0.013214	0	0.565158	0.565252	0.565725	1
Net Income Flag	6819	1	0	1	1	1	1	1
Equity to Liability	6819	0.047578	0.050014	0	0.024477	0.033798	0.052838	1

Variable labels and names

	Variable description and name
Y1	Bankrupt? Class label
X1	ROA(C) before interest and depreciation before interest: Return On Total Assets(C)
X2	ROA(A) before interest and % after tax: Return On Total Assets(A)
X3	ROA(B) before interest and depreciation after tax: Return On Total Assets(B)
X4	Operating Gross Margin: Gross Profit/Net Sales
X5	Realized Sales Gross Margin: Realized Gross Profit/Net Sales
X6	Operating Profit Rate: Operating Income/Net Sales
X7	Pre-tax net Interest Rate: Pre-Tax Income/Net Sales
X8	After-tax net Interest Rate: Net Income/Net Sales
X9	Non-industry income and expenditure/revenue: Net Non-operating Income Ratio
X10	Continuous interest rate (after tax): Net Income-Exclude Disposal Gain or Loss/Net Sales
X11	Operating Expense Rate: Operating Expenses/Net Sales
X12	Research and development expense rate: (Research and Development Expenses)/Net Sales
X13	Cash flow rate: Cash Flow from Operating/Current Liabilities
X14	Interest-bearing debt interest rate: Interest bearing Debt/Equity
X15	Tax rate (A): Effective Tax Rate
X16	Net Value Per Share (B): Book Value Per Share(B)
X17	Net Value Per Share (A): Book Value Per Share(A)
X18	Net Value Per Share (C): Book Value Per Share(C)
X19	Persistent EPS in the Last Four Seasons: EPS-Net Income
X20	Cash Flow Per Share
X21	Revenue Per Share (Yuan ¥): Sales Per Share
X22	Operating Profit Per Share (Yuan ¥): Operating Income Per Share
X23	Per Share Net profit before tax (Yuan ¥): Pretax Income Per Share
X24	Realized Sales Gross Profit Growth Rate
X25	Operating Profit Growth Rate: Operating Income Growth
X26	After-tax Net Profit Growth Rate: Net Income Growth
X27	Regular Net Profit Growth Rate: Continuing Operating Income after Tax Growth
X28	Continuous Net Profit Growth Rate: Net Income-Excluding Disposal Gain or Loss Growth
X29	Total Asset Growth Rate: Total Asset Growth
X30	Net Value Growth Rate: Total Equity Growth
X31	Total Asset Return Growth Rate Ratio: Return on Total Asset Growth
X32	Cash Reinvestment %: Cash Reinvestment Ratio
X33	Current Ratio
X34	Quick Ratio: Acid Test
X35	Interest Expense Ratio: Interest Expenses/Total Revenue
X36	Total debt/Total net worth: Total Liability/Equity Ratio

X37	Debt ratio %: Liability/Total Assets
X38	Net worth/Assets: Equity/Total Assets
X39	Long-term fund suitability ratio (A): (Long-term Liability+Equity)/Fixed Assets
X40	Borrowing dependency: Cost of Interest-bearing Debt
X41	Contingent liabilities/Net worth: Contingent Liability/Equity
X42	Operating profit/Paid-in capital: Operating Income/Capital
X43	Net profit before tax/Paid-in capital: Pretax Income/Capital
X44	Inventory and accounts receivable/Net value: (Inventory+Accounts Receivables)/Equity
X45	Total Asset Turnover
X46	Accounts Receivable Turnover
X47	Average Collection Days: Days Receivable Outstanding
X48	Inventory Turnover Rate (times)
X49	Fixed Assets Turnover Frequency
X50	Net Worth Turnover Rate (times): Equity Turnover
X51	Revenue per person: Sales Per Employee
X52	Operating profit per person: Operation Income Per Employee
X53	Allocation rate per person: Fixed Assets Per Employee
X54	Working Capital to Total Assets
X55	Quick Assets/Total Assets
X56	Current Assets/Total Assets
X57	Cash/Total Assets
X58	Quick Assets/Current Liability
X59	Cash/Current Liability
X60	Current Liability to Assets
X61	Operating Funds to Liability
X62	Inventory/Working Capital
X63	Inventory/Current Liability
X64	Current Liabilities/Liability
X65	Working Capital/Equity
X66	Current Liabilities/Equity
X67	Long-term Liability to Current Assets
X68	Retained Earnings to Total Assets
X69	Total income/Total expense
X70	Total expense/Assets
X71	Current Asset Turnover Rate: Current Assets to Sales
X72	Quick Asset Turnover Rate: Quick Assets to Sales
X73	Working capitcal Turnover Rate: Working Capital to Sales
X74	Cash Turnover Rate: Cash to Sales
X75	Cash Flow to Sales
X76	Fixed Assets to Assets
X77	Current Liability to Liability

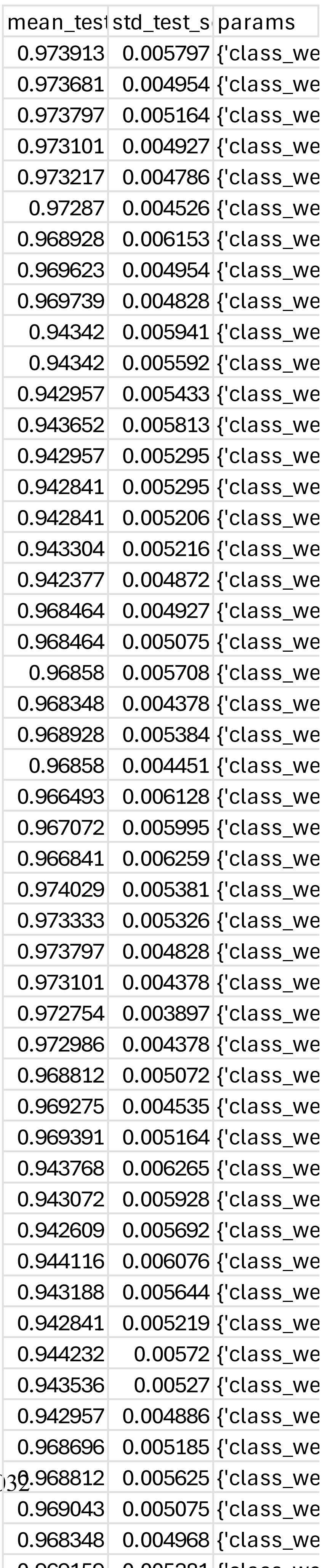
X78	Current Liability to Equity
X79	Equity to Long-term Liability
X80	Cash Flow to Total Assets
X81	Cash Flow to Liability
X82	CFO to Assets
X83	Cash Flow to Equity
X84	Current Liability to Current Assets
X85	Liability-Assets Flag: 1 if Total Liability exceeds Total Assets, 0 otherwise
X86	Net Income to Total Assets
X87	Total assets to GNP price
X88	No-credit Interval
X89	Gross Profit to Sales
X90	Net Income to Stockholder's Equity
X91	Liability to Equity
X92	Degree of Financial Leverage (DFL)
X93	Interest Coverage Ratio (Interest expense to EBIT)
X94	Net Income Flag: 1 if Net Income is Negative for the last two years, 0 otherwise
X95	Equity to Liability

Hyperparameter tuning full outputs

LR

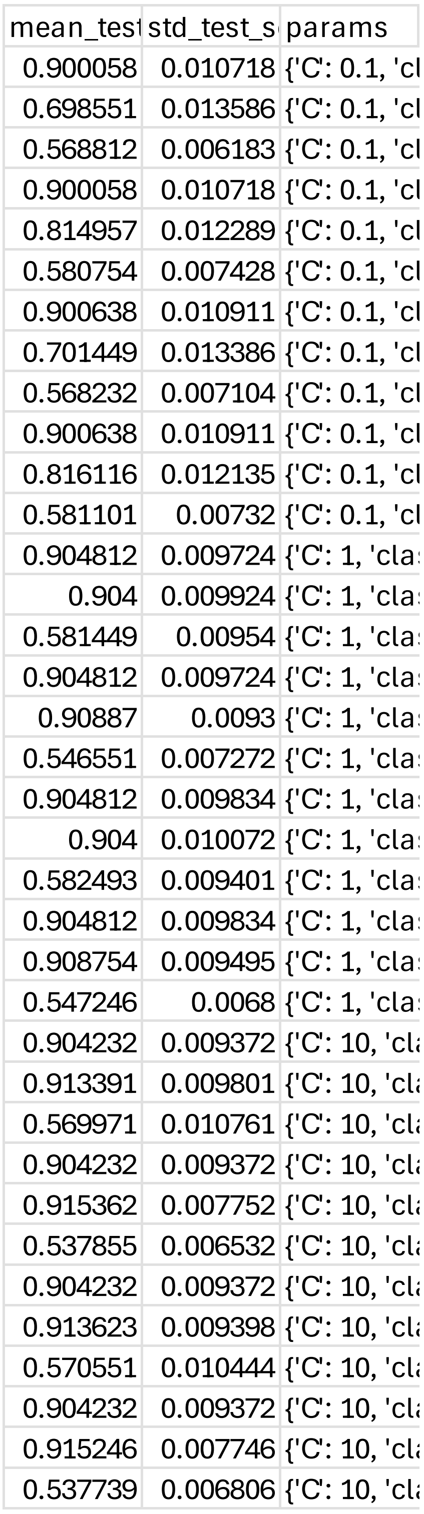
mean_test_score	std_test_score	params
0.891014493	0.009121919	{'C': 0.1, 'solver': 'newton-cg'}
0.891362319	0.008879979	{'C': 0.1, 'solver': 'lbfgs'}
0.875594203	0.011208662	{'C': 0.1, 'solver': 'liblinear'}
0.899014493	0.007755126	{'C': 0.5, 'solver': 'newton-cg'}
0.898898551	0.007338315	{'C': 0.5, 'solver': 'lbfgs'}
0.894724638	0.009828438	{'C': 0.5, 'solver': 'liblinear'}
0.901217391	0.009022644	{'C': 1, 'solver': 'newton-cg'}
0.901333333	0.009146937	{'C': 1, 'solver': 'lbfgs'}
0.900521739	0.009780451	{'C': 1, 'solver': 'liblinear'}
0.901449275	0.008239229	{'C': 5, 'solver': 'newton-cg'}
0.90284058	0.007808676	{'C': 5, 'solver': 'lbfgs'}
0.902492754	0.007935044	{'C': 5, 'solver': 'liblinear'}
0.901797101	0.008365521	{'C': 10, 'solver': 'newton-cg'}
0.902144928	0.008395996	{'C': 10, 'solver': 'lbfgs'}
0.903188406	0.008049417	{'C': 10, 'solver': 'liblinear'}

RF



mean_test	std_test_s	params
0.973913	0.005797	{'class_we
0.973681	0.004954	{'class_we
0.973797	0.005164	{'class_we
0.973101	0.004927	{'class_we
0.973217	0.004786	{'class_we
0.97287	0.004526	{'class_we
0.968928	0.006153	{'class_we
0.969623	0.004954	{'class_we
0.969739	0.004828	{'class_we
0.94342	0.005941	{'class_we
0.94342	0.005592	{'class_we
0.942957	0.005433	{'class_we
0.943652	0.005813	{'class_we
0.942957	0.005295	{'class_we
0.942841	0.005295	{'class_we
0.942841	0.005206	{'class_we
0.943304	0.005216	{'class_we
0.942377	0.004872	{'class_we
0.968464	0.004927	{'class_we
0.968464	0.005075	{'class_we
0.96858	0.005708	{'class_we
0.968348	0.004378	{'class_we
0.968928	0.005384	{'class_we
0.96858	0.004451	{'class_we
0.966493	0.006128	{'class_we
0.967072	0.005995	{'class_we
0.966841	0.006259	{'class_we
0.974029	0.005381	{'class_we
0.973333	0.005326	{'class_we
0.973797	0.004828	{'class_we
0.973101	0.004378	{'class_we
0.972754	0.003897	{'class_we
0.972986	0.004378	{'class_we
0.968812	0.005072	{'class_we
0.969275	0.004535	{'class_we
0.969391	0.005164	{'class_we
0.943768	0.006265	{'class_we
0.943072	0.005928	{'class_we
0.942609	0.005692	{'class_we
0.944116	0.006076	{'class_we
0.943188	0.005644	{'class_we
0.942841	0.005219	{'class_we
0.944232	0.00572	{'class_we
0.943536	0.00527	{'class_we
0.942957	0.004886	{'class_we
0.968696	0.005185	{'class_we
0.968812	0.005625	{'class_we
0.969043	0.005075	{'class_we
0.968348	0.004968	{'class_we
0.969159	0.005281	{'class_we

SVM



mean_tes	std_test_s	params
0.900058	0.010718	{'C': 0.1, 'cl
0.698551	0.013586	{'C': 0.1, 'cl
0.568812	0.006183	{'C': 0.1, 'cl
0.900058	0.010718	{'C': 0.1, 'cl
0.814957	0.012289	{'C': 0.1, 'cl
0.580754	0.007428	{'C': 0.1, 'cl
0.900638	0.010911	{'C': 0.1, 'cl
0.701449	0.013386	{'C': 0.1, 'cl
0.568232	0.007104	{'C': 0.1, 'cl
0.900638	0.010911	{'C': 0.1, 'cl
0.816116	0.012135	{'C': 0.1, 'cl
0.581101	0.00732	{'C': 0.1, 'cl
0.904812	0.009724	{'C': 1, 'cla
0.904	0.009924	{'C': 1, 'cla
0.581449	0.00954	{'C': 1, 'cla
0.904812	0.009724	{'C': 1, 'cla
0.90887	0.0093	{'C': 1, 'cla
0.546551	0.007272	{'C': 1, 'cla
0.904812	0.009834	{'C': 1, 'cla
0.904	0.010072	{'C': 1, 'cla
0.582493	0.009401	{'C': 1, 'cla
0.904812	0.009834	{'C': 1, 'cla
0.908754	0.009495	{'C': 1, 'cla
0.547246	0.0068	{'C': 1, 'cla
0.904232	0.009372	{'C': 10, 'cl
0.913391	0.009801	{'C': 10, 'cl
0.569971	0.010761	{'C': 10, 'cl
0.904232	0.009372	{'C': 10, 'cl
0.915362	0.007752	{'C': 10, 'cl
0.537855	0.006532	{'C': 10, 'cl
0.904232	0.009372	{'C': 10, 'cl
0.913623	0.009398	{'C': 10, 'cl
0.570551	0.010444	{'C': 10, 'cl
0.904232	0.009372	{'C': 10, 'cl
0.915246	0.007746	{'C': 10, 'cl
0.537739	0.006806	{'C': 10, 'cl

GB

mean_test	std_test_s	params
0.930087	0.002901	{'learning_
0.927072	0.003447	{'learning_
0.933217	0.004235	{'learning_
0.932058	0.00387	{'learning_
0.937623	0.006687	{'learning_
0.937159	0.007162	{'learning_
0.930087	0.002901	{'learning_
0.927072	0.003447	{'learning_
0.933217	0.004235	{'learning_
0.932058	0.00387	{'learning_
0.937623	0.006687	{'learning_
0.937159	0.007162	{'learning_
0.947942	0.005356	{'learning_
0.947246	0.005067	{'learning_
0.950377	0.006555	{'learning_
0.95142	0.005882	{'learning_
0.954899	0.004883	{'learning_
0.955362	0.005912	{'learning_
0.947942	0.005356	{'learning_
0.947246	0.005067	{'learning_
0.950377	0.006555	{'learning_
0.95142	0.005882	{'learning_
0.954899	0.004883	{'learning_
0.955362	0.005912	{'learning_
0.962087	0.005482	{'learning_
0.964522	0.005125	{'learning_
0.965101	0.004924	{'learning_
0.966145	0.004775	{'learning_
0.966609	0.004927	{'learning_
0.968116	0.005935	{'learning_
0.962087	0.005482	{'learning_
0.964522	0.005125	{'learning_
0.965101	0.004924	{'learning_
0.966145	0.004775	{'learning_
0.966609	0.004927	{'learning_
0.968116	0.005935	{'learning_
0.950841	0.006725	{'learning_
0.951536	0.007246	{'learning_
0.961855	0.005369	{'learning_
0.962319	0.006435	{'learning_
0.967652	0.00572	{'learning_
0.965333	0.007018	{'learning_
0.950841	0.006725	{'learning_
0.951536	0.007246	{'learning_
0.961855	0.005369	{'learning_
0.962319	0.006435	{'learning_
0.967652	0.00572	{'learning_
0.965333	0.007018	{'learning_
0.967304	0.006098	{'learning_
0.968232	0.00604	{'learning_
0.974145	0.005953	{'learning_
0.974609	0.005203	{'learning_
0.97658	0.005075	{'learning_
0.977391	0.00426	{'learning_
0.967304	0.006098	{'learning_
0.968232	0.00604	{'learning_
0.974145	0.005953	{'learning_
0.974609	0.005203	{'learning_
0.97658	0.005075	{'learning_
0.977391	0.00426	{'learning_
0.974145	0.005519	{'learning_
0.972638	0.004266	{'learning_
0.978087	0.00528	{'learning_
0.97658	0.005758	{'learning_
0.980058	0.004995	{'learning_
0.979478	0.005371	{'learning_
0.974145	0.005519	{'learning_
0.972638	0.004266	{'learning_
0.978087	0.00528	{'learning_
0.97658	0.005758	{'learning_
0.980058	0.004995	{'learning_
0.979478	0.005371	{'learning_
0.959304	0.00971	{'learning_
0.967652	0.002603	{'learning_
0.910841	0.058306	{'learning_
0.971942	0.002831	{'learning_
0.900174	0.041439	{'learning_
0.964406	0.022511	{'learning_
0.959304	0.00971	{'learning_
0.967652	0.002603	{'learning_
0.910841	0.058306	{'learning_
0.971942	0.002831	{'learning_
0.900174	0.041439	{'learning_
0.964406	0.022511	{'learning_
0.948174	0.029388	{'learning_
0.974841	0.005141	{'learning_
0.892986	0.079337	{'learning_
0.97913	0.003721	{'learning_
0.847536	0.116124	{'learning_
0.98029	0.003793	{'learning_
0.948174	0.029388	{'learning_
0.974841	0.005141	{'learning_
0.892986	0.079337	{'learning_
0.97913	0.003721	{'learning_
0.847536	0.116124	{'learning_
0.98029	0.003793	{'learning_
0.941681	0.058761	{'learning_
0.978899	0.005953	{'learning_
0.944812	0.053659	{'learning_
0.97913	0.005866	{'learning_

Full list of required libraries

Package	Version
absl-py	2.1.0
accelerate	0.29.2
aggdraw	1.3.18.post0
alembic	1.13.1
aniso8601	9.0.1
astunparse	1.6.3
blinker	1.7.0
certifi	2024.2.2
charset-normalizer	3.3.2
click	8.1.7
cloudpickle	3.0.0
colorama	0.4.6
contourpy	1.2.0
cycler	0.12.1
docker	7.0.0
entrypoints	0.4
filelock	3.13.4
Flask	3.0.2
flatbuffers	24.3.7
fonttools	4.48.1
fsspec	2024.3.1
gast	0.5.4
gitdb	4.0.11
GitPython	3.1.43
google-pasta	0.2.0
graphene	3.3
graphql-core	3.2.3
graphql-relay	3.2.0
greenlet	3.0.3
grpcio	1.62.1
h5py	3.10.0
huggingface-hub	0.22.2
idna	3.6
imbalanced-learn	0.12.0
imblearn	0.0
importlib_metadata	7.1.0
itsdangerous	2.1.2
Jinja2	3.1.3
joblib	1.3.2
keras	3.1.1
kiwisolver	1.4.5
libclang	18.1.1
Mako	1.3.2
Markdown	3.6
markdown-it-py	3.0.0
MarkupSafe	2.1.5
matplotlib	3.8.2
mdurl	0.1.2
ml-dtypes	0.3.2
mlflow	2.11.3
mlxtend	0.23.1
mpmath	1.3.0
namex	0.0.7
networkx	3.3
numpy	1.26.4
opt-einsum	3.3.0
optree	0.10.0
packaging	23.2
pandas	2.2.0
pillow	10.2.0
pip	24.0
protobuf	4.25.3
pydot	2.0.0
Pygments	2.17.2
pyparsing	3.1.1
python-dateutil	2.8.2
pytz	2024.1
pywin32	306
PyYAML	6.0.1
querystring-parser	1.2.4
regex	2023.12.25
requests	2.31.0
rich	13.7.1
safetensors	0.4.3
scikit-learn	1.4.1.post1
scipy	1.12.0
seaborn	0.13.2
setuptools	69.2.0
six	1.16.0
smmap	5.0.1
SQLAlchemy	2.0.29
sqlparse	0.4.4
sympy	1.12
tensorboard	2.16.2
tensorboard-data-server	0.7.2
tensorflow	2.16.1
tensorflow-intel	2.16.1
termcolor	2.4.0
threadpoolctl	3.3.0
tokenizers	0.15.2
torch	2.2.2
tqdm	4.66.2
transformers	4.39.3
typing_extensions	4.10.0
tzdata	2024.1
urllib3	2.2.1
visualkeras	0.0.2
waitress	3.0.0
Werkzeug	3.0.1
wheel	0.43.0
wrapt	1.16.0
xgboost	2.0.3
zipp	3.18.1

XII. References

Alaka, H. A., Oyedele, L. O., Owolabi, H. A., Vikas, K., Ajayi, S. O., Akinade, O. O., & Bilal, M. (2018). Systematic review of bankruptcy prediction models: Towards a framework for tool selection. *Expert Systems with Applications*, 164-184.

Alam, T., Shaukat, K., Mushtaq, M., Ali, Y., Khushi, M., Luo, S., & Abdul, W. (2021). Corporate Bankruptcy Prediction: An Approach Towards Better Corporate World. *The Computer Journal*, 1731–1746.

Altman, E. (1968). Financial Ratios, Discriminant Analysis and the Prediction of Corporate Bankruptcy. *The Journal of Finance*, 589-609.

Baheti, P. (2021, 7 8). *The Essential Guide to Neural Network Architectures*. Retrieved from v7labs: https://www.v7labs.com/blog/neural-network-architectures-guide

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

Beaver, W. H. (1966). Financial Ratios As Predictors of Failure. *Journal of Accounting Research*, 71-111.

Brownlee, a. (2020, 8 20). *A Gentle Introduction to the Rectified Linear Unit (ReLU)*. Retrieved from Machine Learning Masteru: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

Coşkun, S. B., & Turanli, M. (2023). Credit risk analysis using boosting methods . *Journal of Applied Mathematics Statistics and Informatics*, 5-18.

Cowan, D. (n.d.). *Tanh Activation Function*. Retrieved from The Science of Machine Learning & AI: https://www.ml-science.com/tanh-activation-function

Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. (2015).

Government Commercial Function. (2023). *Coporate FInancial Distress - Guidance Note.* London: UK Government.

Hebb, D. (1949). *The Organization of Behaviour.* New York: John Wiley& Sons.

Hyder, J., & Naaz, S. (2019). Credit Card Fraud Detection using Local Outlier Factor and Isolation. *International Journal of Computer Sciences and Engineering* , 1060-1064.

Jeong, C., Min, J. H., & Kim, M. S. (2012). A tuning method for the architecture of neural network models incorporating GAM and GA as applied to bankruptcy prediction. *Expert Systems with Applications*, 3650-3658.

Koehrsen, W. (2017, 12 27). *Random Forest Simple Explanation*. Retrieved from Medium: https://williamkoehrsen.medium.com/random-forest-simple-explanation-377895a60d2d

Manokhina, A. (2018, 4 16). *Open Machine Learning Course. Topic 10. Gradient Boosting*. Retrieved from medium: https://medium.com/open-machine-learning-course/open-machine-learning-course-topic-10-gradient-boosting-c751538131ac

Ohlson, J. A. (1980). Financial Ratios and the Probabilistic Prediction of Bankruptcy. *Journal of Accounting Research*, 109-131.

Richardson, G., & Sablik, T. (2015, 12 4). *Banking Panics of the Gilded Age*. Retrieved from Federal Reserve History: https://www.federalreservehistory.org/essays/banking-panics-of-the-gilded-age

Sameul, A. L. (1959). Some studies in machine learning using the game of checkers. . *IBM Journal of Research and Development*, 210-229.

SAS. (n.d.). *Risk Management*. Retrieved from SAS: https://www.sas.com/en_gb/solutions/risk-management.html

Shanu, S. (2022). *Sigmoid Activation Function*. Retrieved from insideaiml: https://insideaiml.com/blog/Sigmoid-Activation-Function-1031

Simplilearn. (2023, 05 24). *Scikit-learn vs TensorFlow: A Detailed Comparison*. Retrieved from Simplilearn: https://www.simplilearn.com/scikit-learn-vs-tensorflow-article

Singh, H. (2023, 11 7). *Variants of Gradient Descent Algorithm*. Retrieved from Analytics Vidhya: https://www.analyticsvidhya.com/blog/2021/03/variants-of-gradient-descent-algorithm/#:~:text=In%20the%20case%20of%20Stochastic,parameters%20based%20on%20every%20subset.

smaddikonda. (2018). *Bankruptcy-Prediction*. Retrieved from GitHub: https://github.com/smaddikonda/Bankruptcy-Prediction

Son, H., Hyun, C., Phan, D., & Hwang, H. J. (2019). Data analytic approach for bankruptcy prediction. *Expert Systems with Applications*, 138:112816.

UCI Machine Learning Repository. (2020). *Taiwanese Bankruptcy Prediction*. Retrieved from UCI Machine Learning Repository: https://doi.org/10.24432/C5004D

University of Cincinnati. (n.d.). *Feedforward Deep Learning Models*. Retrieved from UC Business Analytics R Programming Guide: https://uc-r.github.io/feedforward_DNN

van Auken, H., García-Pérez-de-Lema, D., & Madrid-Guijarro, A. (2011). An analysis of non-financial factors associated with financial distress. *Entrepreneurship & Regional Development*, 159–186.

Wikipedia contributors. (2024, 04 24). *Support vector machine*. Retrieved from Wikipedia, The Free Encyclopedia: https://en.wikipedia.org/w/index.php?title=Support_vector_machine&oldid=1215385461

Wiklund, J. W. (2022). *Predicting company bankruptcy using artificial neural networks.* Stockholm: KTH Royal Institute of Technology.