Predicting Bankruptcy Using Classification Models

Team 4 - 12.11.2024

Group members: Bjorn Moxnes, Jobin Lakeh, Johannes Kvale & Maran Shanmugathas

1 Introduction

This report evaluates classification models for predicting bankruptcy using logistic regression, feature selection techniques, ensemble methods and neural networks. The analysis is based on financial data from Taiwanese companies between 1999 and 2009.

To predict bankruptcy, a binary outcome, classification is a natural and effective method for assigning data points to one of two categories. Since classification models rely on input features to create decision boundaries, selecting the most relevant financial metrics is critical for improving their performance. Feature selection helps reduce noise, improve accuracy, and make models more interpretable by focusing on attributes that contribute most to predicting bankruptcy.

We will evaluate various feature selection techniques, including Correlation-based Feature Selection, Variance Thresholding, Forward-, Backward Selection and Lasso and Ridge regularization to pinpoint the most relevant features. Additionally, we will explore models with built-in feature selection capabilities. These include Decision Trees and Random Forest, Gradient Boosted Trees, and AdaBoost. Finally, we construct a neural network to compare its performance with the simpler models.j

Lastly, we will test all the models on unseen test data. The results will be presented in a table for easy comparison, followed by a thorough analysis to highlight the best performing models and feature selection techniques.

2 Exploratory Data Analysis

2.1 Dataset Description

Diving into the dataset, we use `data.head()` in part 2 of the notebook to show the first five rows of the table. Each company (row) in the dataset has 96 attributes (columns). The first column, `"bankruptcy?"`, contains either 0 (not bankrupt) or 1 (bankrupt) and is what we are going to try to predict. The other 95 columns (features) are financial metrics such as return on assets, operating gross margin and current ratio.

In Part 2.1, we use the `.info()` method to gain a comprehensive overview of the dataset's structure, including feature names, data types, and the number of entries. This helps us assess the data for preprocessing requirements. The dataset contains 6,819 observations, and all features are numerical (int64 or float64), which is essential for our models to operate effectively, as they require numerical input for computation.

In Part 2.2 of the notebook, we analyze the distribution of the target variable and find that the dataset is highly imbalanced (Figure 1). Approximately 96.8% of companies are labeled as "alive," while only 3.2% are marked as bankrupt. This imbalance poses challenges for predictive modeling, as models often favor the majority class. While techniques such as oversampling or undersampling could address this, we choose not to use them to stay within the scope of this class curriculum and retain the dataset's original distribution.

Preprocessing is performed in Part 2.3 to prepare the data for modeling. In Part 2.3.1 and 2.3.2, we use `.isnull()` and `.duplicated()` to check for missing values and duplicates. We confirm the dataset is clean, with no NaN values or duplicates. Ensuring data quality at this stage is crucial because missing values or duplicates can introduce biases or errors in the model, negatively impacting its performance and reliability. By verifying the dataset's integrity, we create a solid foundation for subsequent analysis and modeling steps.

In Part 2.3.3, we split the data into training and testing sets, allocating 20% of the data to the test set. We then normalize the training data by subtracting its mean and dividing each column by its standard deviation through `.fit` and `.transform(Xtrain)`. Standardization can have beneficial effects on the convergence of the optimization solvers and is necessary in the training of the neural network. To ensure Xtrain is still a DataFrame after transforming, we use `pd.DataFrame`, `columns=Xtrain.columns` and `index=Xtrain.index`.

Finally, we use `.describe()` to observe that the data has been scaled correctly. Specifically, that the mean and standard deviation is 0 and 1, respectively.

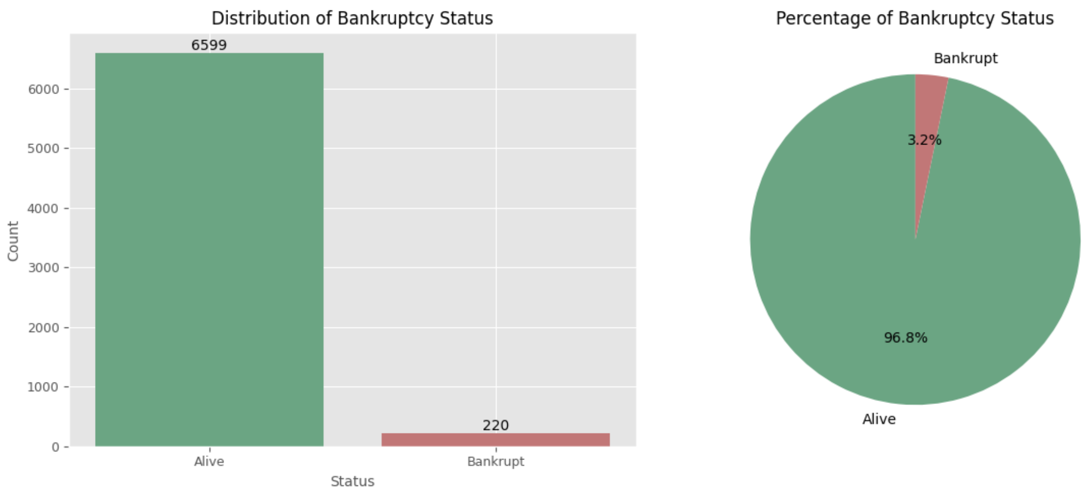


Figure 1: Distribution of data in the dataset

3 Methodology

3.1 Simple Models

In the beginning of section 3 of the notebook, we focus on developing simple classification models for the bankruptcy prediction task. These models serve as benchmarks for evaluating the performance of more optimized and computationally intensive models. By using these straightforward models, which require less computational effort, we establish a foundation to assess whether the added complexity of sophisticated models is justified in terms of improving prediction accuracy for the bankruptcy problem.

3.1.1 Logistic Regression with all features

The first baseline model is a simple logistic regression model trained on the full set of features. While this approach does not involve additional computational effort or coding for feature selection, it requires higher computational resources during training due to the inclusion of all available features. This model provides a reference point to evaluate the potential benefits of feature selection techniques, which aim to reduce dimensionality and enhance computational efficiency without compromising predictive performance.

3.1.2 Logistic Regression with the most correlated features

The second baseline model is a logistic regression model trained on the 10 features most correlated with the target variable. Identifying these features is computationally inexpensive and reduces the dimensionality of the data, making the training process more efficient. By focusing on a smaller subset of relevant features, this model also highlights the potential trade-off between computational efficiency and predictive performance.

3.1.3 Variance Threshold

The third baseline model employs a simple variance threshold technique to select features. This method identifies features with a variance above a specified threshold (variance greater than 1), ensuring that only attributes with sufficient variability are included in the model. Features with low variance—those that show little to no variation across the companies—are unlikely to hold meaningful information for distinguishing between classes, as they do not provide significant differentiation. It is crucial to perform this process on an unscaled dataset, as scaling standardizes all variances to 1, making the Variance Threshold technique ineffective. This approach is computationally efficient and helps streamline the training process by excluding features with minimal variability. By using these selected features, the variance threshold model demonstrates how basic feature selection techniques can enhance computational efficiency while maintaining model relevance.

3.2 Forward and Backwards Selection

Next, we move on to Forward and Backward selection of features. Both algorithms selects a subset of features greedy; Forward stepwise selection starts with no features and iteratively adds the one that most improves model performance until no further significant improvement is possible while Backward stepwise selection begins with all features, removing the least important one at each step. Both methods simplify models by selecting relevant features while maintaining performance. We use 5-fold cross (`cvfolds=5`) validation in both algorithms to determine which feature should be added/removed.

3.3 Regularization

In this section, we introduce two regularization-based logistic regression models: Lasso (L1 regularization) and Ridge (L2 regularization). Regularization techniques are designed to prevent overfitting by penalizing large coefficients, effectively encouraging the model to prioritize the most relevant features. Lasso performs feature selection by driving some coefficients to zero, thereby excluding less significant features, while Ridge shrinks coefficients towards zero without fully eliminating them, promoting stability and robustness. These models, by considering both the loss term $\sum_{i=1}^{N} L(y_i, \hat{y}_i)$ to ensure predictive accuracy and the regularization term $R(\underline{\theta})$ to penalize overfitting, not only enhance generalization to unseen data but also provide insight into feature importance. This makes them invaluable tools for analyzing the bankruptcy prediction problem, as they balance model complexity and relevance.

$$\hat{\underline{\theta}} = \arg\min_{\underline{\theta}} \left(\sum_{i=1}^{N} L(y_i, \hat{y}_i) \ + \ \lambda \, R(\underline{\theta}) \right)$$

The underlying code structure for both models is identical, differing only in the choice of the regularization penalty $R(\theta)$, which is `L1` for Lasso and `L2` for Ridge in the code.

Addressing Class Imbalance with `class_weight`

To account for the highly imbalanced nature of the dataset, the `class_weight` parameter is included and set to `'balanced'` when defining both models. This configuration automatically adjusts the class weights inversely proportional to their frequencies in the dataset. By giving greater importance to the minority class during training, this approach helps the model handle imbalanced data more effectively, improving its ability to generalize across both classes.

Hyperparameter Tuning with GridSearchCV

To optimize the regularization strength, we perform hyperparameter tuning using `GridSearchCV`. Instead of directly tuning the regularization parameter λ as done in some settings, scikit-learn uses the parameter C, which is the inverse of λ ($C = \frac{1}{\lambda}$). A smaller C corresponds to stronger regularization. The `param_grid` is defined to iterate through a logarithmically spaced array of C values, allowing the model to explore a range of regularization strengths systematically.

In `GridSearchCV`, we use `scoring='recall'` and `refit='recall'`. Recall (which is the same as TPR) is particularly suitable for imbalanced datasets because it ensures that the model prioritizes minimizing false negatives. This is critical in scenarios where identifying the minority class is more important than overall accuracy.

The parameter `n_jobs=-1` is set in `GridSearchCV` to utilize all available CPU cores, significantly speeding up the computation for hyperparameter tuning when running locally. Additionally, the cross-validation (`cv`) is set to 3 folds to ensure robust model evaluation without excessive computational cost.

3.3.1 Lasso Regularization

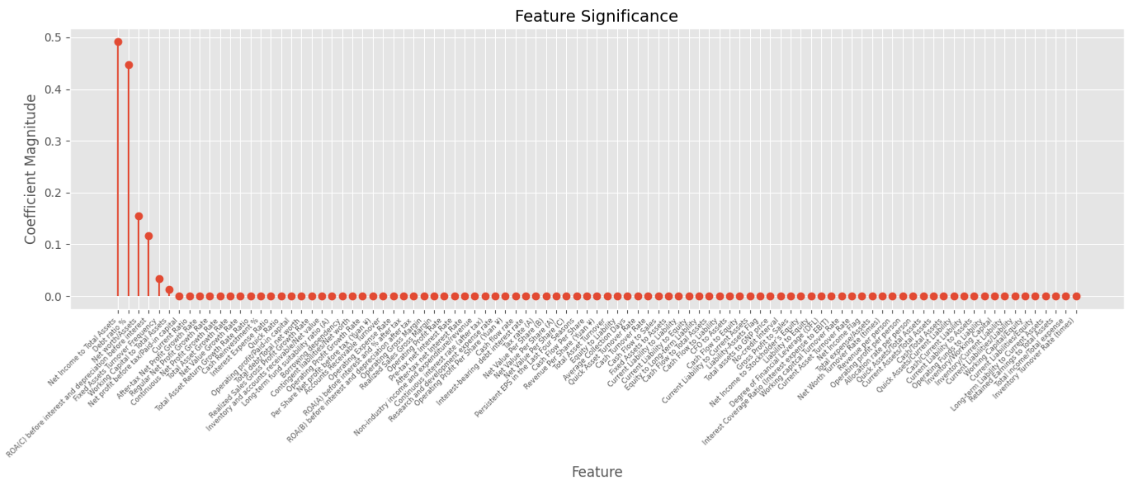


Figure 2: Distribution of data in the dataset

The plot in Figure 2[1] illustrates the coefficients of the optimal Lasso-regularized logistic regression model identified earlier. Notably, many coefficients have been reduced to zero, demonstrating the model's ability to perform feature selection by excluding less relevant variables. This is due to the L1 penalty, defined as $\sum_{j=1}^{P} |\theta_j|$ which shrinks some coefficients to exactly zero. By penalizing the absolute values of the coefficients, the L1 regularization encourages sparsity in the model, effectively selecting only the most relevant features. This approach simplifies the model, enhances interpretability, and helps prevent overfitting by focusing on the most important features for predicting bankruptcy.

3.3.2 Ridge Regularization

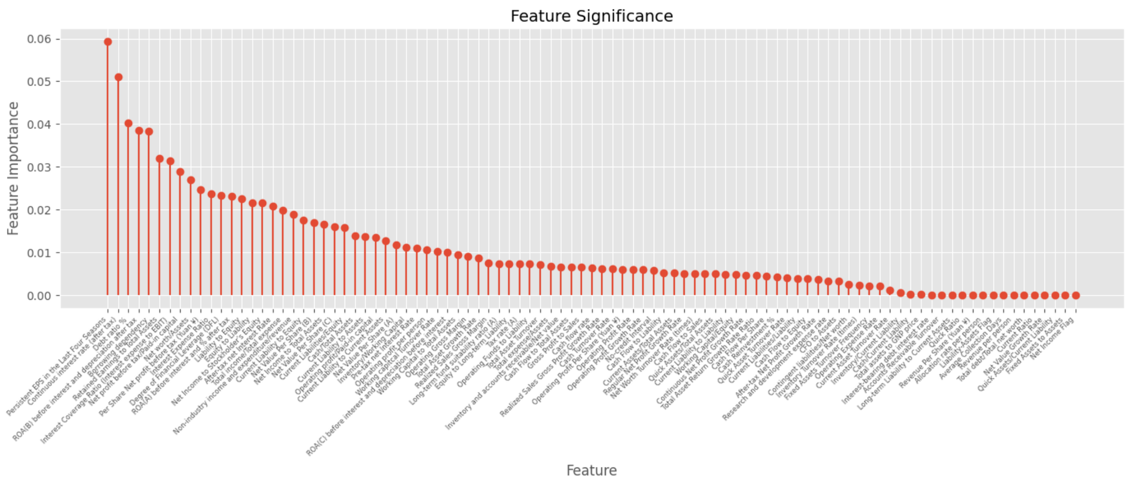


Figure 3: Distribution of data in the dataset

The plot in Figure 3 illustrates the coefficients of the optimal Ridge-regularized logistic regression model identified earlier. Unlike Lasso, Ridge regularization does not reduce coefficients to exactly zero but instead shrinks them towards zero while retaining all features. This is due to the

[1]The purpose with these plots is to show that features are assigned different weights depending on the model, not what the important features are.

L2 penalty, defined as $\lambda \sum_{j=1}^{P} \theta_j^2$ which penalizes the squared magnitude of the coefficients. By applying this penalty, Ridge reduces the impact of less relevant variables, stabilizing the model and improving generalization. This approach ensures that all features contribute to the prediction while controlling for overfitting, making it particularly effective in high-dimensional datasets. The plot highlights how the L2 penalty balances feature importance across the dataset.

3.4 Decision Tree & Ensemble Methods

3.4.1 Decision Tree

In decision tree models, Gini and Entropy are hyperparameters that define the splitting criterion. Gini focuses on minimizing misclassification, while Entropy emphasizes maximizing information gain. By testing both on validation data, one can compare their performance and select the one that yields the best results for a specific dataset. However, to narrow the scope of the report, we chose to work exclusively with Gini. Additionally, the tree depth is a crucial hyperparameter that balances overfitting (deep trees) and underfitting (shallow trees). We explicitly set `max_depth=10` for this model, though it could also be determined through hyperparameter tuning. To narrow the scope of the report, we chose to fix this parameter, allowing it to serve as a baseline for comparison with the ensemble methods discussed later in the report, which inherently handle depth tuning by combining multiple trees of varying depths to balance complexity and generalization.

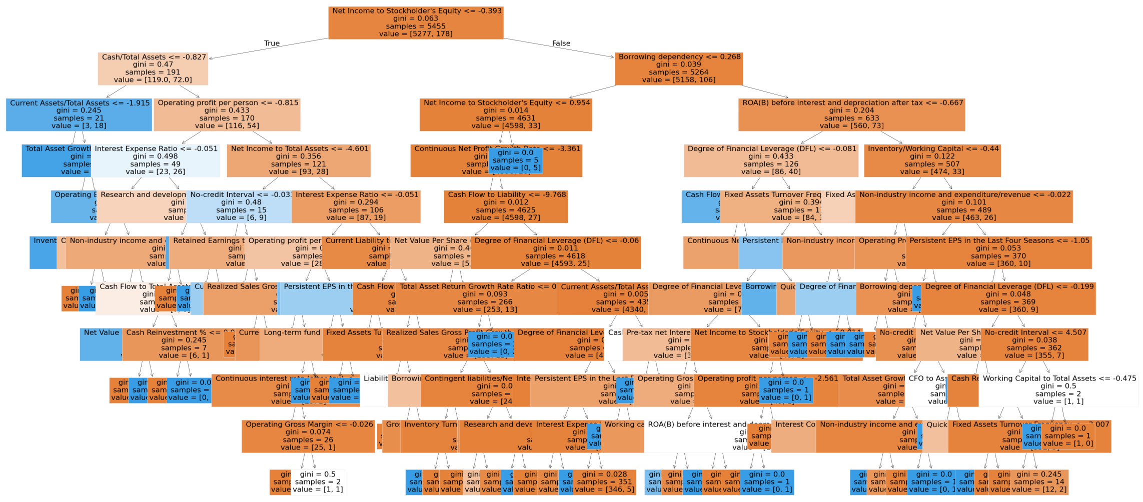


Figure 4: Distribution of data in the dataset

The plot in Figure 4 shows a visualization of the decision tree classifier of the decision tree model in 3.4.1 using the `plot_tree` function. This demonstrates the model's great ability of being easily interpretable. Additionally, decision trees provide feature significance scores, derived from the structure of the tree itself. These scores are not displayed here, as the plots closely resemble the feature importance plots shown in previous subsections and are therefore not repeated here. However, they are available for viewing in the notebook.

Ensemble Methods

The three primary ensemble techniques explored in this analysis are Random Forest, Gradient Boosting, and AdaBoost. These methods enhance predictive performance by combining multiple decision trees to create a more robust model, and they differ in the techniques the employ for tree selection, weighting, and aggregation, ultimately producing more stable and accurate predictive models than a single decision tree.

Unlike a single decision tree, which is inherently interpretable and can be visualized using the `plot_tree` function, ensemble methods generate multiple trees that work collectively to make predictions. This collective approach improves performance by averaging or combining the predictions of multiple trees, reducing variance and enhancing accuracy. However, this gain in predictive power comes at the cost of interpretability, as the ensemble model cannot be visualized as a single tree. Despite this limitation, ensemble methods provide feature importance scores, which offer valuable insights into the relative contributions of different features to the model's predictions.

3.4.2 Random Forest

Random Forest is an ensemble method that operates on the high-complexity high-variance side of the spectrum by constructing multiple deep decision trees and combining their predictions through averaging. Each tree in the forest is trained on a random subset of the data and a random selection of features, ensuring diversity among the trees. This randomness helps reduce overfitting by ensuring that individual trees do not overly rely on specific patterns in the training data, resulting in a more robust and generalized model.

The `max_features` parameter in `param_grid_rf` explores two feature selection strategies at each tree split: `sqrt` (square root of total features) and `log2` (log base 2 of features). These methods differ subtly in how many features are considered during each tree's node split: `sqrt` typically selects more features for larger datasets, while `log2` tends to be more restrictive, especially in high-dimensional spaces. By including both strategies, `GridSearchCV` will systematically test which provides better model performance, allowing the algorithm to determine the most effective feature sampling approach that balances model complexity and predictive power.

Simultaneously, the `n_estimators` parameter uses `np.linspace(2, 100, 5, dtype=int)` to create an array of 5 evenly spaced tree counts, allowing the grid search to explore different ensemble sizes and find the optimal configuration that maximizes the model's performance, particularly focusing on recall for handling class imbalance.

3.4.3 Gradient Boosting Classifier

Gradient Boosting Classifier moves to the other side of the complexity spectrum by focusing on low-complexity, low variance trees and combining them iteratively through boosting. Each tree in the ensemble is trained to correct the errors of its predecessor, gradually improving the model's performance. By using simple models with low depth, gradient boosting maintains low variance and avoids overfitting. However, through the boosting process, it builds a strong predictive model by emphasizing difficult-to-predict examples. This approach allows gradient boosting to achieve high accuracy while retaining the ability to generalize well, making it highly effective for a wide range of tasks.

In contrast to some other models, `GradientBoostingClassifier` does not have a direct `class_weight` parameter. Instead, it requires manual weight adjustment through `sample_weight`, which is a flexible mechanism that allows individual samples to be weighted differently during model training. We therefore need to explicitly compute sample weights using `compute_sample_weight()` and pass them during the model fitting process in `gbc_gs.fit(Xtrain, ytrain, sample_weight=sample_weights)`.

The `learning_rate` parameter in `GridSearchCV` for the Gradient Boosting Classifier controls the step size for updating model parameters during training. We chose the values [0.1, 1.0] where a smaller value allows for conservative, gradual improvements, which can enhance robustness and accuracy. In contrast, a larger value (e.g., 1.0) enables faster, more aggressive updates. By evaluating both rates, `GridSearchCV` identifies whether a cautious or bold update strategy better captures patterns in the imbalanced dataset.

3.4.4 AdaBoost

AdaBoost is another ensemble method that enhances model accuracy by combining multiple weak classifiers, typically decision stumps. It uses an exponential loss function to iteratively increase the weights of misclassified instances, ensuring subsequent models focus on classifying these correctly. Each weak learner is weighted based on its accuracy, contributing to a strong overall model. AdaBoost effectively reduces bias while maintaining low variance and is robust against overfitting, making it effective in handling noisy data.

For the implementation we used `DecisionTreeClassifier` with `max_depth=1` to create decision stumps which ensures simplicity in the weak classifiers. Further, we used the parameter `class_weight='balanced'` to address class imbalance by adjusting weights inversely to class frequencies, ensuring equal class representation during training. For the learning rate, we used values of `[0.01, 0.1]` to test different levels of weak learner contribution, balancing speed and stability.

3.5 Neural Network

Lastly, we construct a sequential neural network with 2 hidden `Dense` layers. The input layer has 95 neurons to accept the input size of 95 features. The first hidden layer has 64 neurons, each using the ReLU activation function to introduce non-linearity and handle complex patterns in the data. The second hidden layer contains 32 neurons with the same activation function. The output layer has a single neuron with a sigmoid activation function, which outputs probabilities between 0 and 1, making it suitable for binary classification tasks. The choice of sigmoid over softmax in this case is due to the binary nature of the problem, which simplifies computation while achieving the same result.

In total, this network has 8,257 parameters. To prepare the model for training, we `compile` the network using `loss='crossentropy'` as our loss-function. This measures the difference between the predicted probabilities and the true binary labels. For training, we use the default `batch_size=32` and 100 epochs. Figure 5 represents the training history to the neural network.

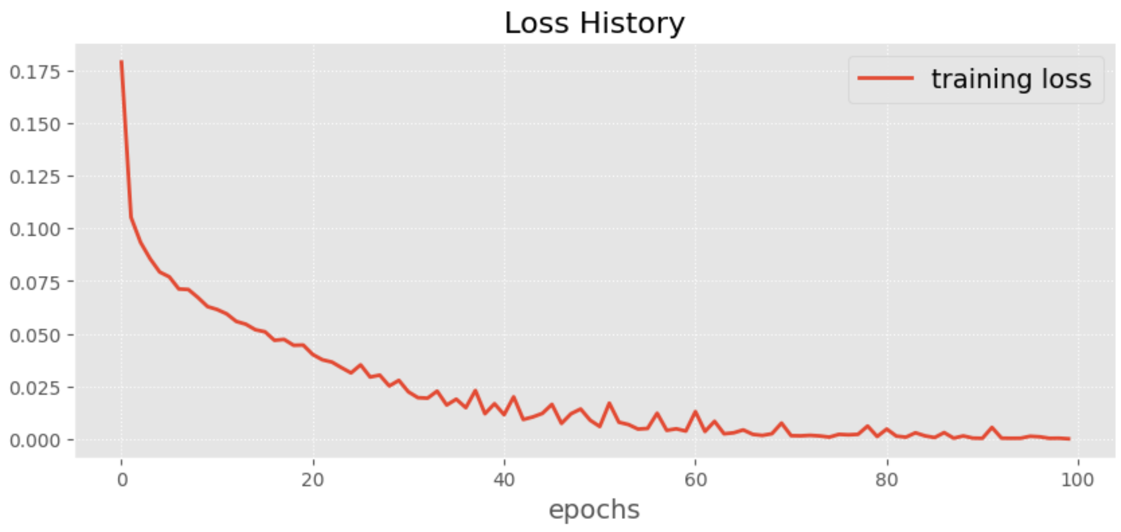
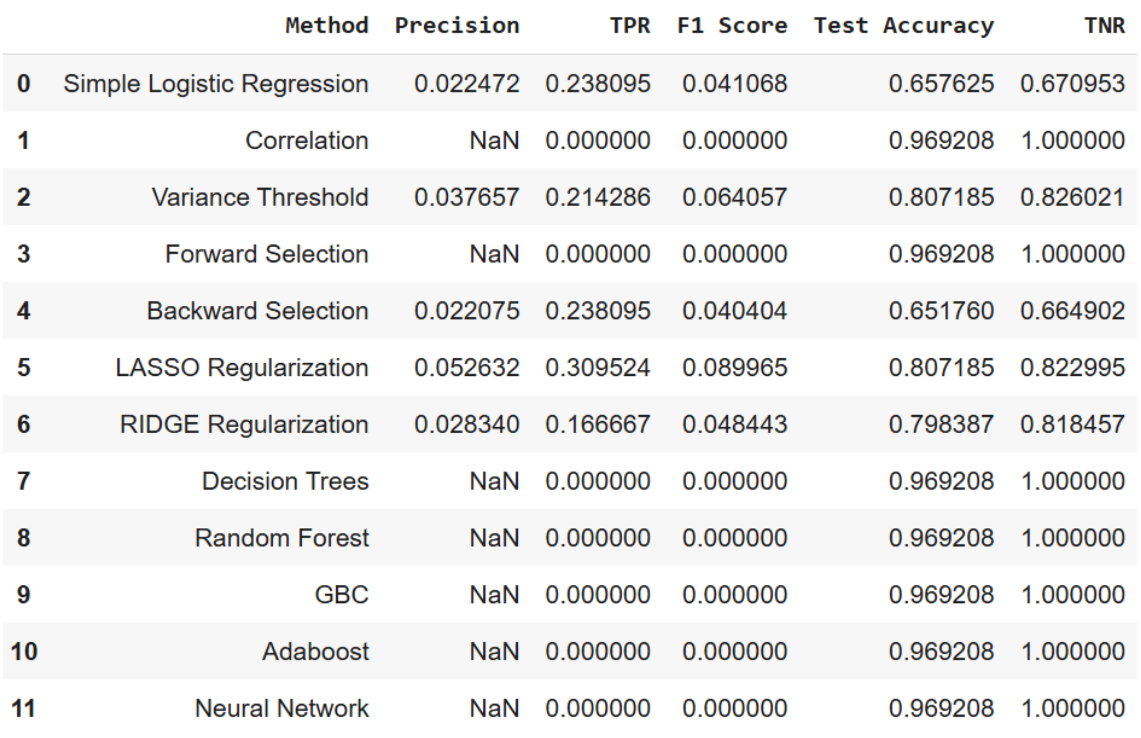


Figure 5: Loss history in neural network training

4 Results

Figure 6 shows the results we got for each model tested on unseen (test) data. The metrics shown are precision, True Positve Rate (TRP), F1 score, accuracy and True Negative Rate (TNR).



	Method	Precision	TPR	F1 Score	Test Accuracy	TNR
0	Simple Logistic Regression	0.022472	0.238095	0.041068	0.657625	0.670953
1	Correlation	NaN	0.000000	0.000000	0.969208	1.000000
2	Variance Threshold	0.037657	0.214286	0.064057	0.807185	0.826021
3	Forward Selection	NaN	0.000000	0.000000	0.969208	1.000000
4	Backward Selection	0.022075	0.238095	0.040404	0.651760	0.664902
5	LASSO Regularization	0.052632	0.309524	0.089965	0.807185	0.822995
6	RIDGE Regularization	0.028340	0.166667	0.048443	0.798387	0.818457
7	Decision Trees	NaN	0.000000	0.000000	0.969208	1.000000
8	Random Forest	NaN	0.000000	0.000000	0.969208	1.000000
9	GBC	NaN	0.000000	0.000000	0.969208	1.000000
10	Adaboost	NaN	0.000000	0.000000	0.969208	1.000000
11	Neural Network	NaN	0.000000	0.000000	0.969208	1.000000

Figure 6: Distribution of data in the dataset

5 Analysis

To evaluate the performance of our machine learning models in predicting bankruptcy, we leveraged multiple metrics that account for the severe class imbalance in the dataset, where 96.8% of instances represent non-bankrupt companies and only 3.2% represent bankrupt ones. Below is a detailed analysis of the key metrics and the performance of each model.

5.1 Precision

Precision measures the proportion of companies correctly identified as bankrupt (true positives) out of all companies predicted as bankrupt. Among the models tested:

- The LASSO Regularization model achieved the highest precision at 0.0526, followed by the Variance Threshold model at 0.038 and RIDGE regularization at 0.0283.
- Simple Logistic Regression and Backward Selection both showed low precision, at approximately 0.022.
- Correlation, Forward Selection, Random Forest, GBC, Adaboost, Decision Trees, and Neural Networks all achieved NA values in precision due to no bankrupt companies being predicted.

In theory, high precision indicates that the model minimizes false positives, i.e., it avoids flagging non-bankrupt companies as bankrupt. However, in this dataset, the precision scores are low because of the severe class imbalance and the models' inability to separate the classes effectively. The bankrupt (minority) class lacks sufficient representation, leading the models to over-predict the non-bankrupt (majority) class.

5.2 True Positive Rate (TPR)

TPR measures the proportion of actual bankrupt companies that are correctly predicted as bankrupt. Since the goal is to identify bankrupt companies accurately, TPR is highly relevant.

- LASSO Regularization had the highest TPR at 0.3095, followed by Simple Logistic Regression and Backward Selection at 0.2381.
- Variance Threshold achieved 0.2143, while RIDGE Regularization got 0.1667
- Neural Network, Correlation, Forward Selection, Random Forest, GBC, Decision Trees, and Adaboost had a TPR of 0.0, failing to detect any bankruptcies.

While a high TPR is essential for minimizing false negatives, these results reflect the difficulty of achieving a strong TPR in this dataset. The highest TPR of 0.3095 (LASSO) still leaves nearly 70% of bankrupt companies undetected, highlighting significant limitations. Achieving higher TPR typically increases false positives, which underscores the importance of evaluating TPR in conjunction with precision to strike a balance.

5.3 F1 Score

The F1 Score is a metric that combines both Precision and TPR into a single metric. A great aspect of the F1 Score is that neither precision nor TPR are disproportionately prioritized, since high TPR alone can lead to excessive false positives and high precision can lead to excessive false negatives. The F1 Score balances these two concerns.

- LASSO Regularization achieved the highest F1 Score at 0.0900, indicating the best balance between precision and TPR among the models.
- Variance Threshold followed with an F1 Score of 0.0641, showing moderate balance.
- RIDGE Regularization, Simple Logistic Regression and Backward Selection had lower F1 Scores of 0.0484, 0.0411 and 0.0419, respectively.
- Neural Network, Correlation, Forward Selection, Random Forest, GBC, Decision Trees and Adaboost had an F1 Score of 0.0, as a result of both precision and TPR being zero.

Given the low precision and TPR scores across most models, it is expected that the F1 Score is also low, as it reflects the balance between these two metrics.

5.4 Accuracy

Accuracy is the proportion of correctly classified instances (both positives and negatives) out of the total instances in the dataset. It measures how often the model is correct overall. However, it does not differentiate between classes, which can be misleading in imbalanced datasets.

- Correlation-Based Selection, Forward Selection, Decision Trees, Random Forest, GBC, Adaboost and Neural Networks achieved the highest accuracy of 96.92%, as they overwhelmingly predicted the majority class (non-bankrupt) correctly but failed to detect any bankruptcies.
- LASSO Regularization and Variance Threshold both showed lower accuracies of 80.72%, with RIDGE Regularization slightly lower at 79.84%, as they traded accuracy for modest improvements in identifying bankruptcies (higher TPR).
- Simple Logistic Regression and Backward Selection performed poorly, with accuracies of 65.76% and 65.18%, as they struggled with correctly classifying both bankrupt and non-bankrupt companies.

The high accuracy values for some models, like Correlation-Based Selection and Neural Networks, are misleading as they fail to detect any bankruptcies. Even models like LASSO Regularization, which attempt to improve TPR and precision, perform poorly overall. Given that a naive model predicting all companies as non-bankrupt would still achieve 97% accuracy, these results highlight the limitations of accuracy as a meaningful metric in imbalanced datasets.

5.5 True Negative Rate (TNR)

TNR measures the proportion of correctly classified non-bankrupt companies. It reflects how well the model avoids false positives. However, like accuracy, it does not differentiate between the classes, which can be misleading in imbalanced datasets.

- Correlation-Based Selection, Forward Selection, Decision Trees, Random Forest, GBC, Adaboost and Neural Networks achieved the highest TNR of 1.0, as they predicted all companies as non-bankrupt but failed to detect any bankruptcies.
- Variance Threshold, LASSO Regularization, and RIDGE Regularization had moderate TNR values of 0.8260, 0.8230, and 0.8185, as they sacrificed some TNR to achieve slight improvements in TPR (identifying bankruptcies).
- Simple Logistic Regression and Backward Selection performed poorly in TNR, with values of 0.6710 and 0.6649, as they produced more false positives.

The high TNR values for some models, like Correlation-Based Selection and Neural Networks, are misleading because they fail to predict any bankruptcies, focusing solely on the majority class. Even models like LASSO Regularization, which attempt to balance TPR and precision, still fall short overall. Given that a naive model predicting all companies as non-bankrupt would also achieve nearly perfect TNR, these results underscore the limitations of TNR as a meaningful metric in this imbalanced dataset.

5.6 Reflections

In the analysis we wanted to evaluate the trade-offs between computational efficiency and model performance. Reducing the number of features can improve processing speed but may compromise predictive accuracy. However, since all models made poor predictions, it appears that running time was not the primary issue. It is worth noting that increasing model complexity, which often leads to longer training times, could also introduce the risk of overfitting.

We also considered evaluating model interpretability against accuracy. While interpretable models like logistic regression and decision trees provide direct insights, complex models such as neural networks require additional techniques to explain their predictions. However, given the poor accuracy of both interpretable models and models that are more complex to interpret, this aspect seems less relevant in our current context.

6 Conclusion

The severe class imbalance in the dataset heavily influenced model performance, with high accuracy and true negative rate masking the failure to detect bankrupt companies. Despite employing various machine learning techniques, from simple logistic regression to advanced neural networks, most models defaulted to predicting the majority class, rendering their predictions nearly useless.

Metrics like precision, true positive rate, and F1 score remained low across most models, highlighting the limitations of traditional evaluation metrics for imbalanced datasets. This research underscores that effective minority class prediction requires sophisticated, targeted strategies for handling severely skewed data. Moreover, a fundamental issue in addressing imbalanced datasets is that data quality and distribution are often more critical than model complexity.