Introduction and Background

Financial stability has always been a critical concern for businesses and investors. One of the important elements of financial stability is being able to predict potential risks, such as bankruptcy. A comprehensive understanding of financial records is essential for making informed business decisions. The inability to predict company bankruptcies results in a substantial financial loss. Machine learning has become a useful tool for making predictions. The main objective of this project is to develop a machine-learning model that predicts the company's likelihood of going bankrupt based on historical financial records. The project began with data collection and preprocessing using a Kaggle dataset comprising 78,682 observations. The data was cleaned, missing values were handled, and feature selection was performed to identify key attributes essential for predicting bankruptcy. This model aims to assist banks, investors, and corporations in spotting potential financial losses and managing risks more effectively.

Literature Review

Our literature review included examining existing research on bankruptcy prediction models and their methodologies. We found that ensemble methods, particularly XGBoost, have shown high efficacy in classification tasks similar to ours. These findings have reinforced our decision to use XGBoost as our primary model. In addition, the team reviewed the Kaggle US Company Bankruptcy Prediction Dataset description to enhance understanding of the general dataset's structure.

Dataset Description

The dataset contains financial data for companies available on the New York Stock Exchange and NASDAQ from 1999 through 2018, including revenues, profit margins, debt ratios, and operational costs. The target variable, "status_label" indicates if a company is operational or bankrupt, making it suitable for binary classification. Bankruptcy status is assigned under two conditions: management filing for Chapter 11 to reorganize the business or Chapter 7 to cease operations entirely. These records provide a comprehensive basis for predicting a company's bankruptcy likelihood using machine-learning models.

The columns of the data set are split into “company_name”, “status_label”, the year, and then the x1-x18 features. The meaning of the X features are as follows: X1 - Current assets, X2 - Cost of goods sold, X3 - Depreciation and amortization, X4 - Earnings before interest, taxes, depreciation, & amortization, X5 - Inventory, X6 - Net Income, X7 - Total Receivables, X8 - Market Value, X9 - Net Sales, X10 - Total assets, X11 - Total Long-term debt, X12 - Earning before interest & taxes, X13 - Gross profit, X14 - Total current liabilities, X15 - Retained earnings, X16 - Total revenue, X17 - Total liabilities, X18 - Total operating expenses.

Experimental Data Analysis

Data Cleaning

No data cleaning was necessary for our dataset. All of the columns were filled with appropriate values, with no values missing, and hence, no need for data imputation. There were also no duplicate observations in the dataset that needed to be considered.

Feature Encoding

We used feature encoding on our target column to represent the class labels (which were originally strings) as numbers before feeding them into the models. For the companies that didn’t go bankrupt (i.e. the “alive” class), we encoded them to 0. For the companies that did go bankrupt (i.e. the “failed” class), we encoded them to 1. We discarded the original target column with the strings of “alive” and “failed” afterwards.

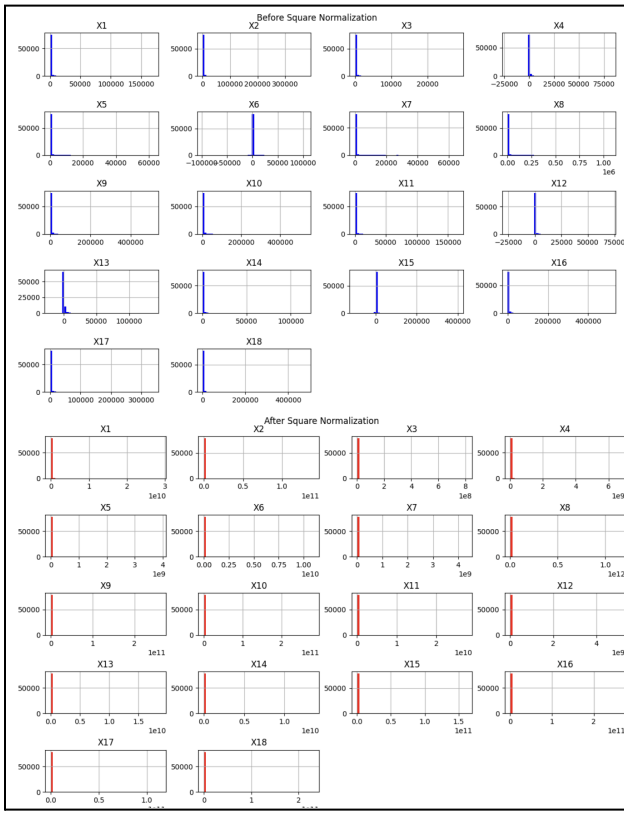
This simple method of binary encoding made sense for our binary classification problem, since there are only two labels, and they can easily be represented with one target column, since there

are only two encoded labels, 0 and 1. Using one-hot encoding would lead to two columns being used, one for the “alive” class, and one for the “failed” class, with sparse vectors (i.e. vectors with 0s) that would take up unnecessary storage.

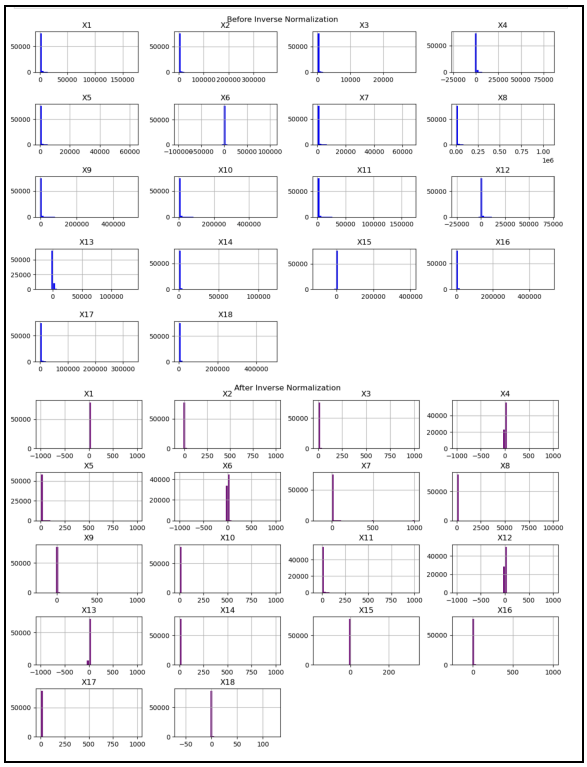
Data Normalization

We explored various normalization methods to prepare the data for modeling. To evaluate the performance of each method, we visualized the data both before and after applying normalization. These visualizations allowed us to compare the effects of the transformations on the data distribution and identify the method that best suited our dataset. Specifically, we tried 3 methods:

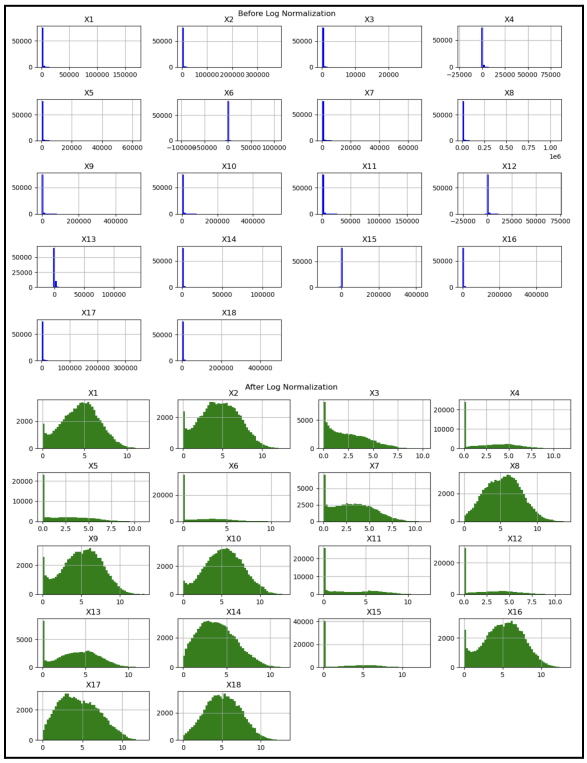
1. Square Normalization



2. Inverse Normalization



3. Log Normalization



We chose log normalization as our best technique since it proved to be the most effective

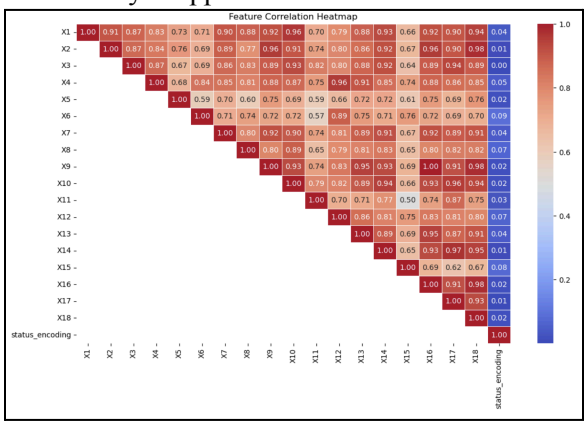
method for transforming the data. The histograms after log normalization demonstrate a more uniform and well-distributed structure compared to the original data, which exhibited extreme skewness and outliers.

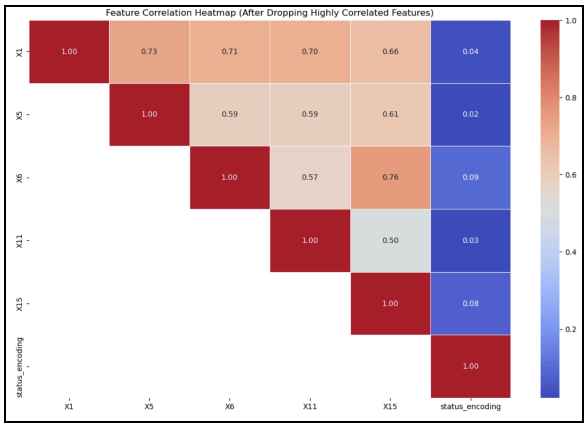
Outlier Detection

To better utilize our data and minimize the impact of outliers on our model we decided to set upper and lower bounds to the data, and set outliers past those ranges to be the max of those ranges. The ranges were determined using box plots with the lowerBound = Q1 - 1.5 * IQR and the upperBound = Q3 + 1.5 * IQR. All

Feature Selection

We used a heatmap to visualize the correlations and understand the relationships between the features. Strong positive correlations are represented in shades of red, while negative correlations are shown in shades of blue. By analyzing these relationships, we identified and removed highly correlated features to reduce redundancy and avoid making our model unnecessarily complex. After removing the redundant features, we generated a new heatmap to confirm that the redundant features were successfully dropped.





Data Scaling

We used z-score standardization to scale all of the feature columns to within a similar range, with the aim of preventing feature columns with larger values from dominating the other feature columns during the training process, which would lead to biased results on the testing data if fed into the wrong model (i.e. a model that requires scaled data).

We applied the following formula to each feature column, leading to each feature column having a mean of 0 and a standard deviation of 1:

$$X_{norm} = \frac{X-\mu}{\sigma} \quad (1).$$

Equation (1): Here, X is our dataset, and μ and σ are the mean and standard deviation (of each individual column), respectively. The result of the operation is a standardized dataset.

A subtlety in applying z-score standardization to the dataset was that it needed to be partitioned into the training and testing datasets using an appropriate split first. (In our case, we used an 80:20 split, with 80% of the original dataset comprising the training dataset, and 20% comprising the testing dataset.) This is so that we could calculate the mean and standard deviation (using the above formula) based on the training dataset only, and then apply the standardization (based on these values) to the training and testing dataset afterwards. This is so

that no data leakage could be introduced into our model, through indirectly looking into the testing dataset with our mean and standard deviation calculations (which would be based off of values in the testing dataset), potentially leading to overly-optimistic results on the testing dataset that would give us an inaccurate representation of our models' performance.

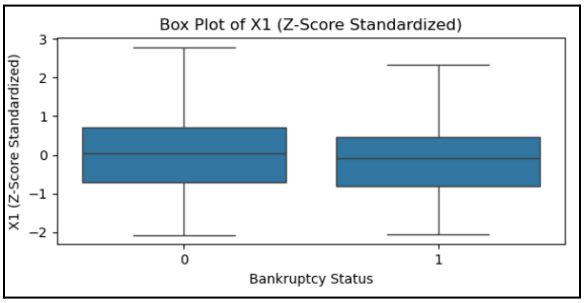


Figure 1: Here, a box plot visualization shows how the data of the first feature column in the training dataset is distributed for both class labels (0 being the "alive" companies, and 1 being the "failed" companies) after applying data scaling using z-score standardization.

In the above plot, the data ranges from about -2 to 3, with a very small difference in the distribution between the class labels. The median is about 0 for both class labels, which is also the mean of the entire feature column's values, indicating that the feature column more or less follows a normal distribution.

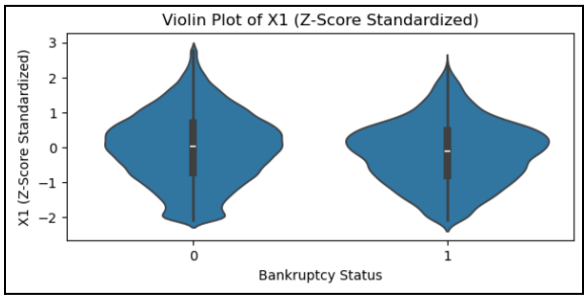


Figure 2: Here, a similar violin plot shows the first feature column's median and interquartile range, similar to the box plot. Additionally, we can observe that most of the data falls within the range of -1 to 1, as can be seen from the width of the "violin" in that range.

Data Sampling

Something inherent in our dataset was that it had much more 0 samples than 1 samples. In our training dataset, initially, there were 58742 samples for the 0 class, and 4203 samples for the 1 class. Since classification models like logistic regression are poor with imbalanced datasets, we needed to deal with the issue accordingly.

To put the issue into perspective, initially, when testing the logistic regression model that had been trained on the imbalanced testing dataset, the model yielded an accuracy of about 93%, giving a false impression of good performance. In reality, the model was classifying all of the 0 samples well, and the 1 samples badly, and due to the imbalance in the dataset, the penalty of misclassifying the minority class was low, and hence, the accuracy was good, despite the precision and recall being poor.

Oversampling is a way to deal with imbalanced datasets. "The simplest oversampling method involves randomly duplicating examples from the minority class in the training dataset, referred to as Random Oversampling" (Brownlee). According to Brownlee, one popular algorithm for random oversampling is SMOTE, or Synthetic Minority Oversampling Technique (which is also available for use in Python). We applied this oversampling algorithm on the testing dataset, the result being a new testing dataset with 58472 samples for both classes (i.e. a balanced dataset) that could be fed into our models.

We also used undersampling, which involves randomly deleting samples of the majority class from the testing dataset (Brownlee). We removed samples of the 0 class until we reached equilibrium, with 4203 samples for both classes in the new testing dataset. Unfortunately, we did significantly reduce the testing dataset, but the data was now balanced.

Another method we used was SMOTE and Edited Nearest Neighbors Rule, or SMOTEENN, which is a combination of oversampling and undersampling: "Experiments have shown that applying both types of techniques together can often result in better overall performance of a model fit on the

resulting transformed dataset" (Brownlee). After applying the SMOTEENN algorithm (which is also available for use in Python) on the testing dataset, the result was a new dataset with a slight imbalance in the samples between both classes, with 40,693 samples from the 0 class, and 48876 samples from the 1 class (which was the original minority class).

All of these testing datasets (including the original, imbalance dataset) could be used in our models.

Proposed Methodology

Our proposed methodology was to use three different classification models and gauge the results of each, selecting the optimal model for use. Our chosen models were logistic regression, random forest, and XGBoost, each model varying in degree of complexity, with their own advantages.

Logistic Regression Methodology

Logistic regression is a simple model that is commonly used for binary classification, and was hence a good starting point for training and testing our model, especially with the availability of easy-to-use Python libraries for conducting the task.

All different types of sampling methods were chosen to be trained in conjunction with logistic regression to see which would result in the best model, including the original scaled testing dataset with an imbalance in the amount of samples with either class, the oversampled testing dataset with a balance between both classes, the hybrid-sampled testing dataset, with a close balance between both classes, and the undersampled testing dataset, with both classes being evenly represented, but the number of samples having shrunk significantly.

Hyperparameter tuning was performed with three selected hyperparameters: the choice of the penalty term, which was either an L1 penalty term representing lasso regularization, or an L2 penalty term representing ridge regression, to prevent any overfitting from occurring in the model, through penalizing large weights and encouraging smaller weights and a more robust, generalizable model that could perform well on unseen data. C, which is also more well-known as the inverse of λ, is the inverse of the regularization parameter, which controls the strength of the regularization or the penalization applied to the model. Since C is the inverse of the regularization parameter λ, lower values of C indicate higher values of λ, meaning a stronger penalization applied to the weights, which could potentially cause underfitting. A high value of C indicates the opposite: a low value of λ, and a very small amount of penalization applied to the weights, which could lead to overfitting if too small. Some different values of C were provided to see which was the best, including 0.01, 0.1, 1, 10, and 100. Lastly, the number of max iterations was also selected as a hyperparameter, which can be thought of as slightly similar to the term of epochs in stochastic and batch gradient descent, in the sense that the model would either terminate with the max number of iterations, or converge to an optimal model before then if appropriate conditions were met (Scikit Learn). Values for the maximum number of iterations chosen were 100, 200, 300, and 400.

To help determine the optimal combination of hyperparameters that would result in the best model of logistic regression, grid search was used (which is available in Python) to simulate the process and go through each combination of hyperparameters, selecting the optimal one.

Random Forest Methodology

Random Forest is an ensemble learning technique designed to enhance predictive accuracy and reduce overfitting by aggregating the outputs of multiple decision trees. Due to having a hyperparameter that controls the number of features, we could guarantee more uniqueness and correlation reduction between the trees. We utilize a Random Forest model consisting of 100 decision trees. The dataset is split into 80% for training and 20% for testing. During training, the model builds trees based on randomly selected subsets of data and features, ensuring variability and reducing overfitting.

Feature important analysis of this algorithm helps identify the most significant predictors of bankruptcy for our classification problem. The results of the feature importance analysis are visualized to provide a clearer understanding of the model's decision-making process and the contribution of each feature.

XGBoost Methodology

XgBoost is a machine learning library that uses gradient boosted decision trees, a supervised learning library that uses gradient descent. It is good for speed, efficiency and ability to scale well. It works by splitting the data between a training and testing set and converting all the data into a DMatrix format. This is an internal data structure optimized for memory efficiency and training speed. The "object" hyperparameter is used to determine what kind of classification will be done (ie. binary, multiclass;softmax).

Experimental Results and Evaluation

Logistic Regression Results and Evaluation

When logistic regression was conducted on the original scaled testing dataset that had a major imbalance between the 0 samples (i.e. the "alive" companies that didn't go bankrupt) and the 1 samples (i.e. the companies that "failed" or went bankrupt), it led to misleading results based on the original accuracy result, which was around 93%, which would be a wonderful sight at first glance. After more careful prying, it was learned that the model was simply cateogirizng the majority class (i.e. the 0 samples) correctly very well, and horribly failing for the minority class (i.e. the 1 samples). In general, these situations could arise probably partly due to a given model learning well on the majority class samples due to their number and disregarding the minority class since their relative size is so small in comparison, and a misclassification in that category isn't as penalizing. Our model was performing well on the testing dataset, which was imbalanced, with thousands of samples belonging to the majority, and around a thousand for the minority. Since the majority was so large, classifying all of the majority samples correctly and the minority samples incorrectly had no impact on the accuracy, since the ratio between the two was so skewed. Since the most important part of bankruptcy prediction is the minority, and being able to figure out if a company is on the brink of going bankrupt, this was not good enough as a first step.

Oversampling was used as a next step, but that too wasn't good, as the model was performing very poorly on the testing dataset, possibly due to overfitting on the testing dataset, as oversampling with SMOTE can involve duplicating random samples of the minority class, which could possibly lead to the model overfitting on those particular samples.

With hybrid-sampling, the same issue persisted, with a very low performance on the testing dataset.

Out of all the sampling methods, undersampling was the best, as it only involved cutting out the majority class samples, without any forms of duplication, which made it perform better relatively with the other sampling methods, but still not great. The original accuracy was around 93%, but with this sampling method, was now down to around 61% on the grid search, and 57% for the testing dataset, which is not good at all, but the classification for the minority class had gotten much better.

In terms of the optimal hyperparameters, the best ones were a C value of 0.1, max iterations of 100, and a penalty term of L2, or ridge regression.

	precision	recall	f1-score	support
0	0.96	0.57	0.71	14720
1	0.09	0.64	0.16	1017
accuracy			0.57	15737
macro avg	0.53	0.60	0.44	15737
weighted avg	0.90	0.57	0.68	15737

Table 1: This shows the different evaluation metrics for the logistic regression model on the testing dataset.

The 0 class was still performing okay after greatly changing the training dataset, with an

f1-score of 71%, but the minority class still hadn't improved that much. This could indicate that more data preprocessing would be needed, through better feature selection through other methods, and possibly engineering new features that could separate between both classes more effectively.

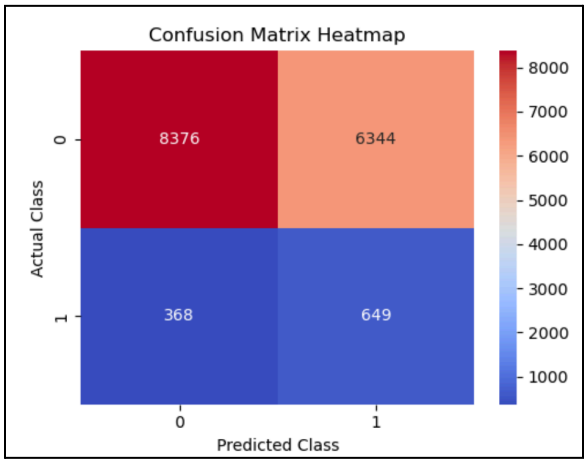


Figure 3: Here, a heatmap of the confusion matrix is depicted, showing the results of the model on the testing dataset.

Note again, that the 0 class indicates the majority class or the "alive" companies, and 1 indicates the minority class or the "failed" companies. What can be seen from the above matrix, is that the classification for the minority class, which had previously been much worse, with about a few samples being classified correctly, was now at 649 correctly classified samples, but at the unfortunate cost of classifying the majority class, which was originally only misclassifying a few samples, but had now increased to 6344 misclassified samples.

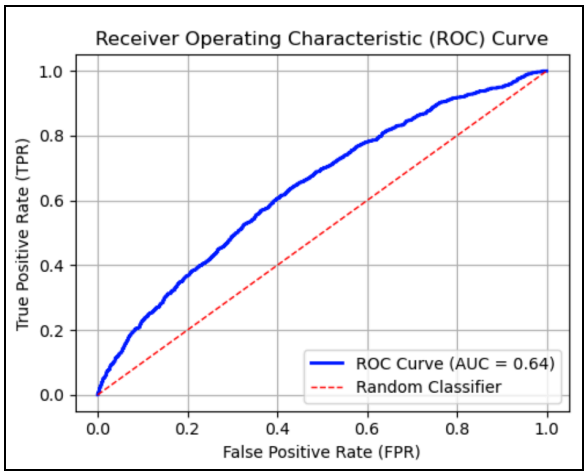


Figure 4: Here is an ROC curve showing a comparison of the TPR to the FPR.

Our imbalanced data set was further reflected in our ROC curve while comparing the true positive rate and the false positive rate. The area under the curve is at 0.64, which shows that the model is better than 50/50 when making predictions but not significantly so.

Conclusion and Discussion

This project aimed to develop a machine-learning model to predict company bankruptcy by leveraging historical financial data. We explored various data preprocessing techniques, sampling methods, and models to enhance the accuracy and reliability of predictions. The results indicate several critical takeaways regarding the challenges and methodologies involved in bankruptcy prediction.

The inherent imbalance in the dataset presented significant obstacles to model performance. Initial attempts using logistic regression highlighted the limitations of traditional models when applied to skewed data. Oversampling and hybrid sampling approaches failed to address the imbalance effectively, often leading to overfitting. Undersampling proved slightly more promising, improving performance on minority class predictions, though at the cost of overall accuracy. The findings underscore the importance of carefully managing class imbalance, particularly in datasets where the

minority class carries critical decision-making value.

The evaluation metrics and results revealed the limitations of logistic regression for this task. Despite its simplicity and interpretability, the model struggled to provide robust predictions for the minority class. Alternative models such as random forest and XGBoost—known for their ability to handle non-linear relationships and higher-dimensional data—show greater potential in this domain. Further experiments should focus on implementing these models with advanced hyperparameter tuning and feature engineering to maximize performance. Z-score standardization improved the comparability of features and reduced bias, while log normalization and outlier treatment contributed to stabilizing data distributions. However, further exploration of feature engineering techniques and domain-specific insights could enhance model interpretability and predictive power.
Our work serves as a foundation for future research aimed at improving bankruptcy prediction models. Recommendations for future work include testing additional ensemble methods, such as gradient boosting and stacking, and integrating external datasets to improve model generalizability. Collaboration with financial experts to refine feature selection and interpret model outputs can also provide actionable insights for stakeholders.

This project has demonstrated the complexities and opportunities inherent in using machine learning for financial risk prediction. While the results indicate room for improvement, the methodologies and findings provide valuable direction for developing more effective solutions, ultimately benefiting banks, investors, and corporations in managing financial risks.

Additional Information

GitHub Link
Link:
https://github.com/dovelover145/Bankruptcy-Prediction

Project Roadmap

1. **Phase 1:** Data Collection & Preprocessing
   a. Collect the dataset
   b. Clean the data
   c. Address missing data
   d. Feature selection
2. **Phase 2:** Model Development
   a. Train the different models
   b. Conduct analysis
3. **Phase 3:** Model Evaluation
   a. Evaluate the model using cross-validation
   b. Record performance (accuracy, precision)
4. **Phase 4:** Hyperparameter Tuning
   a. Optimize hyperparameters
5. **Phase 5:** Web Development
   a. Front-End
   b. Back-End
   c. Database
6. **Phase 6:** Final Testing & Report Preparation
   a. Test the final model
   b. Prepare project report
7. **Phase 7:** Project Report
   a. Final project report

Assignments

1. **Artem Tikhonov:** Collected data, performed feature selection (heatmap), worked on normalization techniques (square, inverse, and log), researched the best-fit models as alternatives for future model development (like XGBoost), worked with Seyed Omid Zahiri on the Random Forest model (its evaluation and hyperparameter tuning), wrote the entire code for front-end & back-end parts of web development, developed the team's timeline, and recorded the project demo. The team member also attended all the team's weekly meetings, and contributed to 1-Pager Project Report, Mid-Quarter Project Progress Report, and Final Report.
2. **Jasper Fadden:** Worked on the XGBoost model (its evaluation and hyperparameter tuning). The team member also attended all the team's weekly meetings, and contributed

to 1-Pager Project Report and Mid-Quarter Project Progress Report.

3. **Lucas Kjellberg:** Addressed the outliers, and worked with Ibrahim Siddiqui on the Logistic Regression model (its evaluation and hyperparameter tuning), and implemented the ROC curve. The team member also attended all the team's weekly meetings, and contributed to 1-Pager Project Report, Mid-Quarter Project Progress Report, and Final Report.
4. **Ibrahim Siddiqui:** Found the dataset, set up all the coding environments, involved in the feature encoding and Z-score scaling, and worked with Lucas Kjellberg on the Logistic Regression model (its evaluation and hyperparameter tuning) as well as data sampling. The team member also attended all the team's weekly meetings (partially for one), and contributed to the Final Report.
5. **Seyed Omid Zahiri:** Helped with feature selection code (heatmap), worked on normalization techniques (square, inverse, and log), researched the XGBoost algorithm, and worked with Artem Tikhonov on the Random Forest model (its evaluation and hyperparameter tuning). The team member also attended all the team's weekly meetings, and contributed to 1-Pager Project Report, Mid-Quarter Project Progress Report, and Final Report.

References

Barboza, Flavio, et al. “Machine Learning Models and Bankruptcy Prediction.” Expert Systems with Applications, vol. 83, Oct. 2017, pp. 405–417, https://doi.org/10.1016/j.eswa.2017.04.006.

Brownlee, Jason. “Tour of Data Sampling Methods for Imbalanced Classification.” *Machine Learning Mastery*, 23 Jan. 2020, machinelearningmastery.com/data-sampling-methods-for-imbalanced-classification/.

IBM. “What Is XGBoost? | IBM.” Www.ibm.com, 5 June 2024, www.ibm.com/topics/xgboost.

Narvekar, Aditya, and Debashis Guha. “Bankruptcy Prediction Using Machine Learning and an Application to the Case of the COVID-19 Recession.” Data Science in Finance and Economics, vol. 1, no. 2, 2021, pp. 180–195, https://doi.org/10.3934/dsfe.2021010.

Scikit-Learn.“LogisticRegression.” *Scikit-Learn*, 2024, scikit-learn.org/1.5/modules/generated/sklearn.linear_model.LogisticRegression.html.

Scikit-Learn.“RandomForestClassifier.” *Scikit-Learn*, 2024, scikit-learn.org/1.5/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

“US Company Bankruptcy Prediction Dataset.” Www.kaggle.com, www.kaggle.com/datasets/utkarshx27/american-companies-bankruptcy-prediction-dataset.