Northeastern University
College *of* Engineering

Polish Companies Bankruptcy Prediction

FINAL PROJECT REPORT

BY

Aryaman Kakad

FOR

IE 7300 - Statistical Learning for Engineers

Under the Guidance of

Prof. Ramin Mohammadi

1. ABSTRACT

Forecasting bankruptcy involves the anticipation of financial insolvency and different indicators of economic hardship for businesses. This field constitutes a broad scope within finance and accounting investigations, owing to its significance to creditors and investors seeking insights into the potential bankruptcy risks of a company. The objective of predicting financial instability is to construct a prognostic framework integrating diverse econometric factors for an advanced understanding of a firm's financial well-being. Within this realm, diverse approaches have been suggested, leveraging statistical hypothesis testing, statistical modeling (such as generalized linear models), and more recently, artificial intelligence techniques like neural networks, Support Vector Machines, and decision trees.

In our project, we record our findings while constructing, investigating, and evaluating various prevalent classification models commonly employed in bankruptcy prediction. These models include Logistic Regression, Gaussian Naive Bayes, Decision Tree, and Support Vector Machines.

We opted for the dataset on bankruptcies among Polish Companies, incorporating synthetic attributes to represent more complex statistical aspects. A synthetic attribute involves amalgamating econometric measures through basic arithmetic operations (addition, subtraction, multiplication, division). Our initial steps involve conducting data preprocessing and exploratory data analysis. During this phase, we address missing data by employing well-established imputation techniques such as Mean and k-nearest Neighbors, adjusting our approach based on the percentage of missing values within each feature. To tackle the imbalance in data distribution, we employ the Synthetic Minority Oversampling Technique (SMOTE) for augmenting instances belonging to the minority class labels.

The feature extraction phase of our project involves a comprehensive approach to enhance model performance. Utilizing Backward Elimination through Logistic Regression helps streamline features. Additionally, Logistic Regression with L1 regularization optimizes the sub-library for improved accuracy. Correlation matrices aid in understanding feature interdependencies, while Box Plots assess data distribution and scaling. Lastly, Box Plots are employed to identify outliers and understand feature data distribution, contributing to a robust feature extraction methodology.

2. INTRODUCTION

Predicting the insolvency of a business holds significant weight in economic decision-making. The financial health of a company, regardless of its size, not only impacts the local community, industry stakeholders, and investors but also casts ripples on policy-making and the global economy. Hence, the substantial social and economic repercussions stemming from corporate insolvencies have garnered the attention of scholars aiming to gain a deeper understanding of the causes behind business failures and, ultimately, to enhance the ability to predict financial distress. The extent of research in this domain is intricately linked to data availability. With a plethora of data on both bankrupt and non-bankrupt public companies, various accounting ratios signaling potential risks can be computed, accompanied by numerous other plausible explanatory factors. As a result, this field proves conducive to testing increasingly sophisticated, data-driven forecasting methodologies.

The aim of predicting bankruptcy is to evaluate a company's financial health and future outlook for sustained market operations. This domain, spanning finance and econometrics, integrates human expertise with historical data from thriving and unsuccessful businesses. Businesses are commonly evaluated through various metrics depicting their operational status, which are then employed to construct a mathematical model based on previous instances.

Various challenges are linked to bankruptcy prediction. Two primary concerns arise in this context: Initially, domain experts suggest econometric indicators to portray the firm's status. However, the effective integration of these indicators into a cohesive model remains ambiguous. Secondly, training models with historical data often grapple with imbalanced data issues, as successful companies significantly outnumber bankrupt ones. Consequently, the resulting model tends to classify companies as successful (the majority class), even when some may be distressed firms. These challenges significantly impact the ultimate predictive performance of the model.

The evolution of bankruptcy prediction has seen the utilization of various statistical techniques that gradually became accessible. This journey involves an increasing awareness of potential pitfalls in initial analyses. Notably, there is still ongoing publication of research that falls into these pitfalls despite their long-standing recognition. The examination of bankruptcy prediction dates back to at least 1932, marked by FitzPatrick's study in The Certified Public Accountant, which

analyzed 20 pairs of firms— one that failed and one that survived. These pairs were matched based on date, size, and industry. Unlike contemporary statistical analyses, FitzPatrick did not employ such methods but instead meticulously interpreted ratios and their trends. This interpretation essentially amounted to a sophisticated analysis involving multiple variables.

Turning to contemporary approaches in the realm of bankruptcy prediction, it's pertinent to mention the current application of survival methods. Recent developments include option valuation approaches that factor in stock price variability. Structural models, on the other hand, consider a default event for a firm when its assets significantly diminish compared to its liabilities. The testing of bankruptcy prediction has extended to encompass neural network models and other intricate methodologies. Business information companies now employ advanced techniques that go beyond the content of annual accounts, taking into account current events such as age, legal judgments, negative publicity, payment incidents, and creditor payment experiences.

Recent studies in the domain of Bankruptcy Prediction explore diverse approaches, modeling techniques, and individual models to determine if any particular method outperforms others. In a study by Zhang, Wang, and Ji (2013), an innovative rule-based system was introduced to address the bankruptcy prediction challenge. The entire process involves four stages: firstly, sequential forward selection identified the most crucial features; secondly, a rule-based model, chosen for its ability to convey tangible meaning, was tailored to the dataset; thirdly, a genetic ant colony algorithm (GACA) was incorporated. To enhance the algorithm, the fitness scaling strategy and the chaotic operator were integrated, resulting in a novel approach called fitness-scaling chaotic GACA (FSCGACA), utilized to optimize the parameters of the rule-based model. Lastly, the model's generalization was improved through the application of the stratified K-fold cross-validation technique.

3. DATASET DESCRIPTION

We have examined a dataset focused on predicting bankruptcies, sourced from Polish bankruptcy data available on the University of California Irvine (UCI) Machine Learning Repository. This repository is a valuable resource for the Machine Learning/Data Science community, offering openly accessible datasets for research and learning purposes. Specifically, our dataset revolves around forecasting bankruptcy for Polish companies and was sourced from the Emerging Markets Information Service (EMIS), a global database covering information on emerging markets. The analysis spans 2000-2012 for bankrupt companies and 2007-2013 for those still in operation. This dataset proves highly pertinent to our research on bankruptcy prediction, featuring valuable econometric indicators as attributes and a substantial number of samples from Polish companies across five different timeframes. For our project, we focus on the 3rd-year data, encompassing financial rates from the 3rd year of the forecasting period and corresponding class labels indicating bankruptcy status three years later. In the 3rd year subset of the dataset, there are a total of 10,503 instances, with 495 being classified as bankrupt and 10,008 as non-bankrupt. This multivariate dataset comprises 64 features, each consisting of real values. Notably, there is some missing data in this subset. The dataset, donated on April 11, 2016, is particularly suited for classification tasks.

Dataset Link: Polish companies bankruptcy data - UCI Machine Learning Repository

The 64 features within our dataset are as follows:

X1 net profit / total assets

X2 total liabilities / total assets

X3 working capital / total assets

X4 current assets / short-term liabilities

X5 [(cash + short-term securities + receivables - short-term liabilities) / (operating expenses - depreciation)] * 365

X6 retained earnings / total assets

X7 EBIT / total assets

X8 book value of equity / total liabilities

X9	sales / total assets
X10	equity / total assets
X11	(gross profit + extraordinary items + financial expenses) / total assets
X12	gross profit / short-term liabilities
X13	(gross profit + depreciation) / sales
X14	(gross profit + interest) / total assets
X15	(total liabilities * 365) / (gross profit + depreciation)
X16	(gross profit + depreciation) / total liabilities
X17	total assets / total liabilities
X18	gross profit / total assets
X19	gross profit/sales
X20	(inventory * 365) / sales
X21	sales (n) / sales (n-1)
X22	profit on operating activities / total assets
X23	net profit/sales
X24	gross profit (in 3 years) / total assets
X25	(equity - share capital) / total assets
X26	(net profit + depreciation) / total liabilities
X27	profit on operating activities / financial expenses
X28	working capital / fixed assets
X29	logarithm of total assets
X30	(total liabilities - cash) / sales
X31	(gross profit + interest) / sales
X32	(current liabilities * 365) / cost of products sold
X33	operating expenses / short-term liabilities
X34	operating expenses / total liabilities
X35	profit on sales / total assets
X36	total sales / total assets

X37	(current assets - inventories) / long-term liabilities
X38	constant capital / total assets
X39	profit on sales/sales
X40	(current assets - inventory - receivables) / short-term liabilities
X41	total liabilities / ((profit on operating activities + depreciation) * (12/365))
X42	profit on operating activities/sales
X43	rotation receivables + inventory turnover in days
X44	(receivables * 365) / sales
X45	net profit/inventory
X46	(current assets - inventory) / short-term liabilities
X47	(inventory * 365) / cost of products sold
X48	EBITDA (profit on operating activities - depreciation) / total assets
X49	EBITDA (profit on operating activities - depreciation) / sales
X50	current assets / total liabilities
X51	short-term liabilities / total assets
X52	(short-term liabilities * 365) / cost of products sold)
X53	equity / fixed assets
X54	constant capital / fixed assets
X55	working capital
X56	(sales - cost of products sold) / sales
X57	(current assets - inventory - short-term liabilities) / (sales - gross profit - depreciation)
X58	total costs /total sales
X59	long-term liabilities/equity
X60	sales/inventory
X61	sales/receivables
X62	(short-term liabilities *365) / sales
X63	sales / short-term liabilities
X64	sales / fixed assets

4. EXPLORATORY DATA ANALYSIS

4.1 - FEATURE ENGINEERING

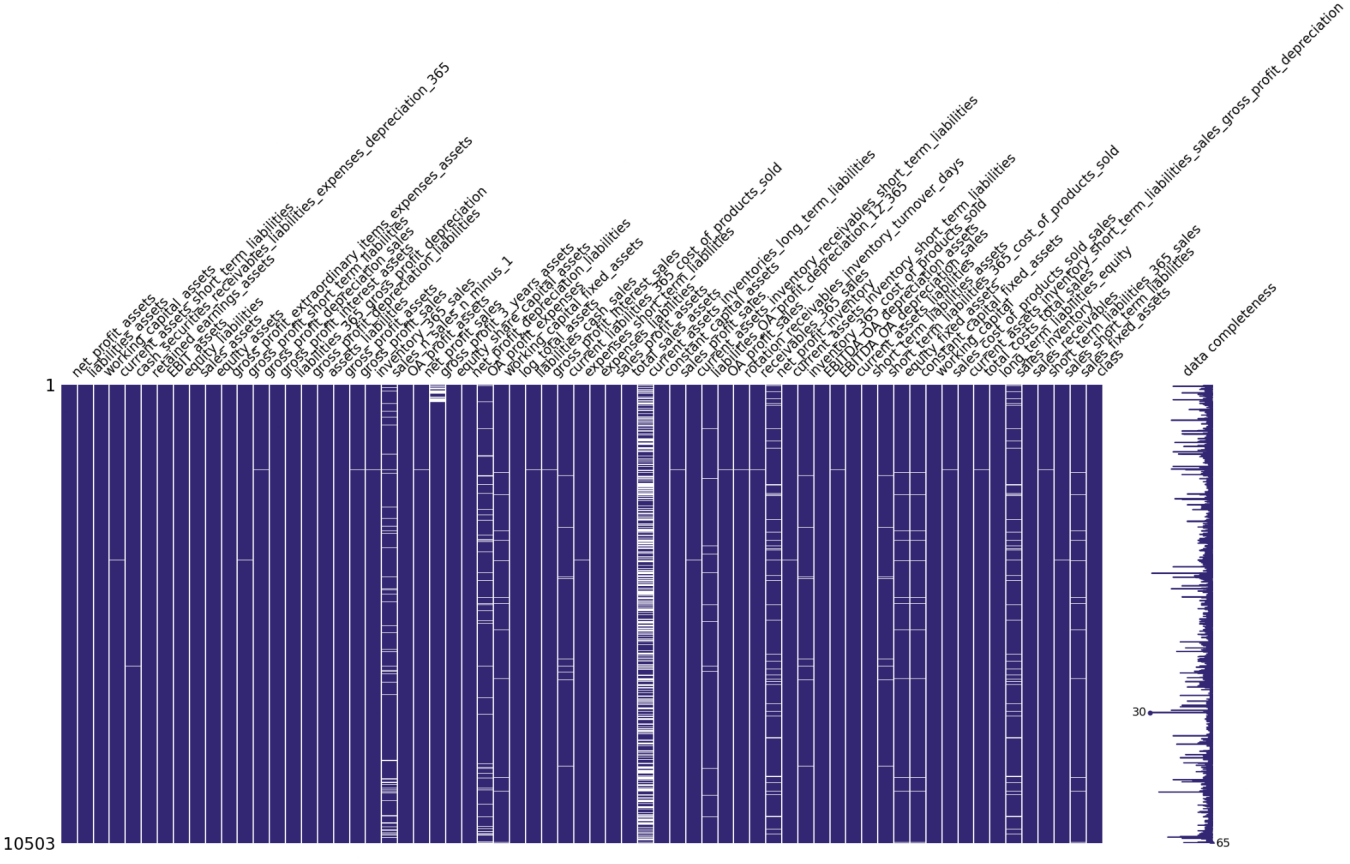
The absence of data poses three challenges:

- Incomplete data may lead to a significant amount of distortion.
- It complicates the handling and analysis of the data.
- Results in decreased efficiency.

Eliminating rows containing missing values, known as Listwise deletion, introduces bias and impacts the representativeness of the findings. The sole practical alternative to Listwise deletion is Imputation. Imputation involves substituting missing data with estimated values and preserving all cases by replacing missing data based on available information. In our project, we investigated two imputation techniques, which will be discussed in the subsequent sections:

1) Imputation using the Mean
2) Imputation using Missing Forest

During our process of feature engineering, we employed the ***missingno*** library to visually depict the absence of values in our dataset. The matrix plot, created using missingno, offers a lucid insight into the patterns of missing data across each feature.



Following this visualization, we performed a systematic evaluation of missing values by calculating the percentage of absence for each feature in our dataset. For features with a missing value percentage of less than 10, we implemented ***Mean Imputation*****.** On the other hand, if the missing value percentage exceeded 10, we opted for ***Missing Forest Imputation***, a technique that leverages the information from neighboring data points to impute the missing values.

Deciding to employ Mean Imputation for missing values below 10% and adopting Missing Forest Imputation for values exceeding this threshold involves carefully weighing the trade-offs between simplicity and complexity inherent in imputation techniques. Mean Imputation, known for its simplicity and directness, is applied when dealing with a relatively small proportion of missing values. It is computationally efficient and maintains the overall simplicity of the dataset, providing a reasonable estimation for missing data without unnecessary complexities. Conversely, in cases where missing values exceed 10%, choosing Missing Forest Imputation proves advantageous. This more sophisticated approach leverages ensemble learning with decision trees to capture complex relationships and patterns in the data. Using a forest of trees enables it to handle non-linear dependencies and offer more precise imputations, especially in situations where basic imputation methods may be insufficient. This decision reflects a strategic balance between computational efficiency for minor data gaps and the enhanced predictive capabilities of ensemble methods for more significant missing data scenarios.

4.1.1 - Mean Imputation

The mean imputation method involves substituting missing values in the dataset with the average of the corresponding variable. In our data set, we employed this technique by replacing missing values in a feature with the mean derived from the other available non-missing values for that specific feature. The utilization of mean imputation has the effect of diminishing correlations related to the imputed variable(s). This occurs because, in instances of imputation, there is an assurance of no correlation between the imputed variable and any other variables that have been measured. While mean imputation possesses favorable attributes for univariate analysis, it presents challenges for multivariate analysis. Therefore, we selected mean imputation as a foundational approach, implementing it through the ***SimpleImputer*** class in scikit-learn.

4.1.2 - Missing Forest Imputation

Missing Forest Imputation for missing data is an advanced methodology designed to tackle the issue of missing data within a dataset. This innovative approach involves the creation of a "forest" composed of decision trees. Each tree is trained on a subset of the data that contains complete information, enabling them to capture the intricate relationships and patterns inherent in the dataset. In the presence of missing values, the collective action of the forest is employed to predict and fill in the absent data points based on the acquired patterns. This technique proves particularly effective in managing missing data within a dataset that is non-linear and complex, offering a robust strategy for imputation.

4.2 - FEATURE SELECTION

In the domain of projects centered on predicting bankruptcy classification, the process of feature extraction assumes a crucial role in elevating both the effectiveness and interpretability of the model. The features chosen exert substantial influence on the model's predictive capabilities, and diverse methodologies are employed to extract pertinent information from the dataset. In this context, we explore five distinctive approaches to feature extraction:

4.2-1 - **Logistic Regression with Backward Elimination:**

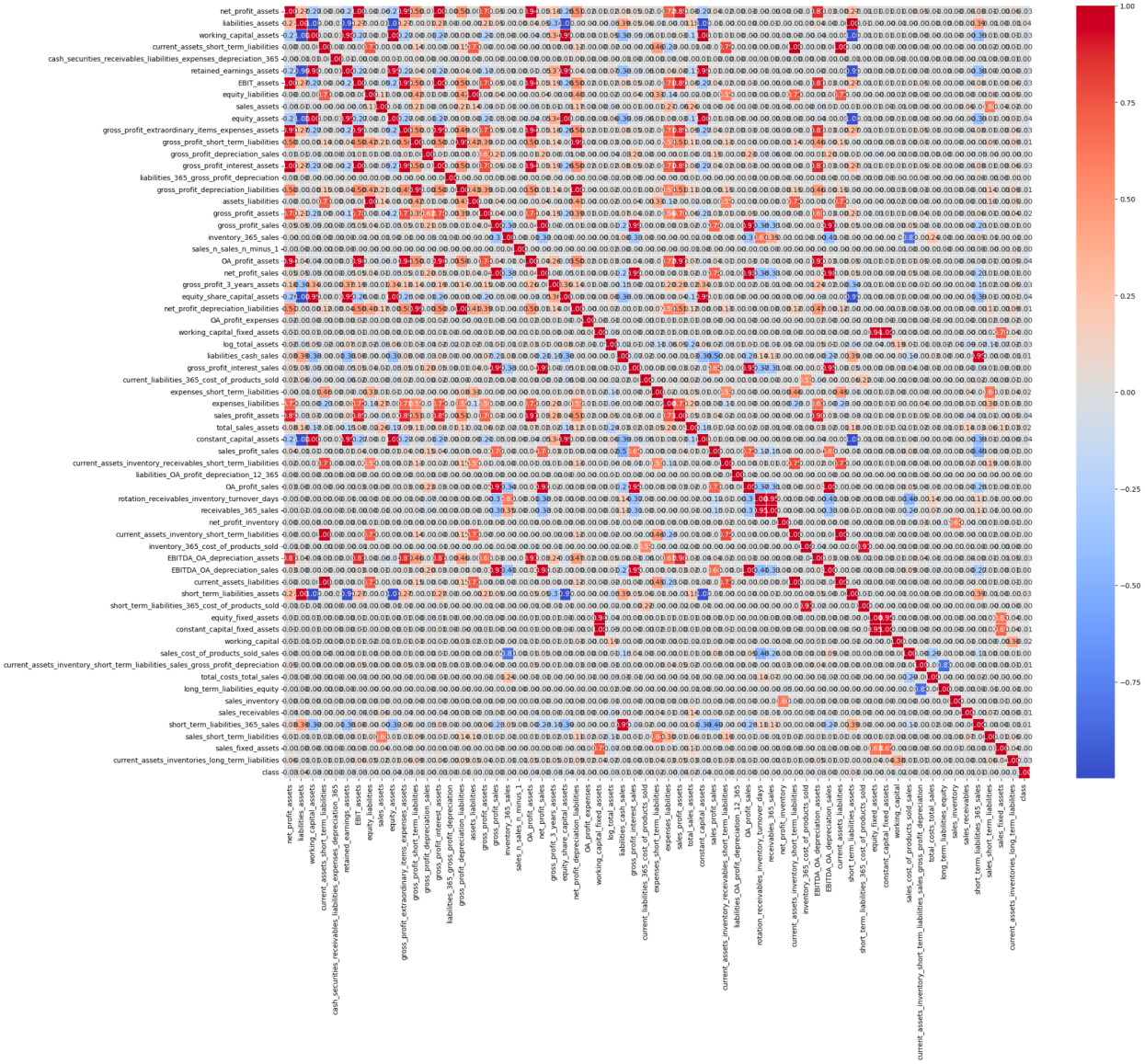
Employing logistic regression in tandem with a backward elimination process proves to be a valuable technique for feature extraction. This involves systematically removing variables with the least significance, based on statistical criteria, until the model achieves optimal performance. This approach assists in pinpointing and retaining the most influential features that significantly contribute to the task of predicting bankruptcy. After applying this method, we reduced the number of features in our dataset from 64 to 60.

4.2.2 - **Logistic regression sub-module employing L1 regularization:**

The integration of this regularization method serves to encourage sparsity in the feature set. This particular regularization approach promotes the occurrence of precisely zero coefficients for certain features, essentially pinpointing a subset of the most pertinent attributes. This not only streamlines the model but also eases the discernment of critical predictors for bankruptcy. The number of features after using this method of feature extraction remained the same that is 60 features.

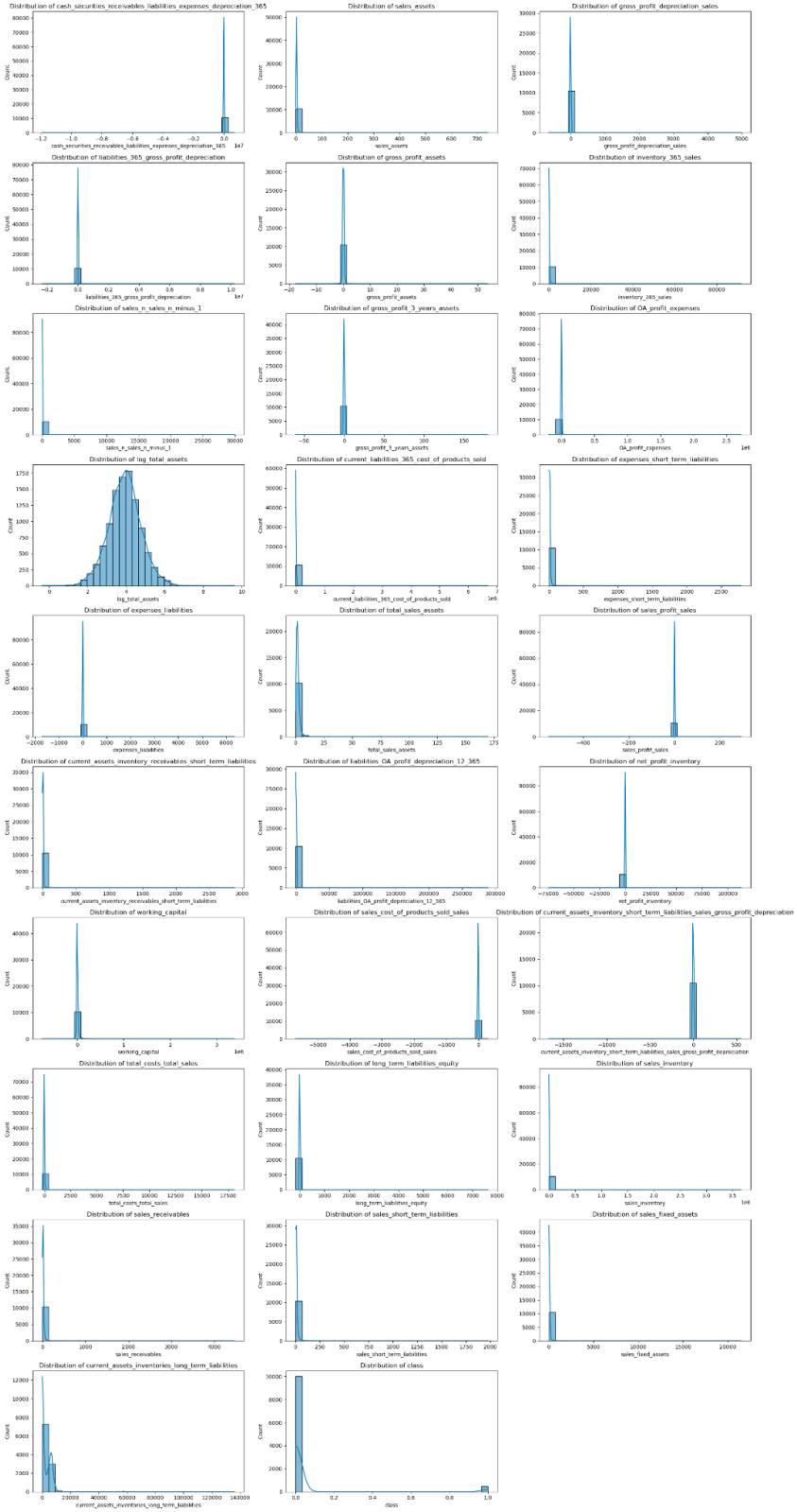
4.2.3 - **Correlation Matrix**

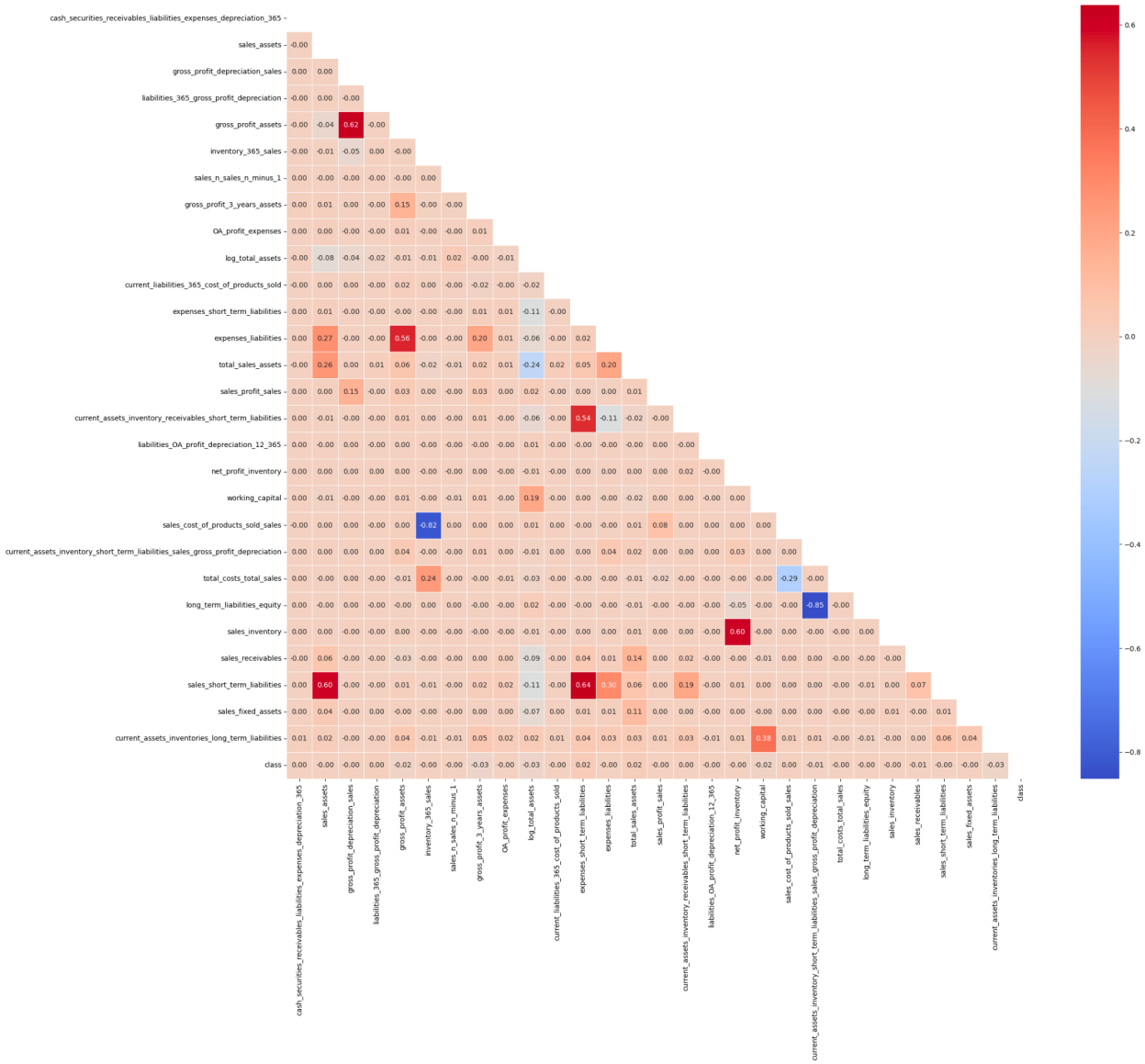
Examination of the correlation matrix facilitates the detection of associations and interdependencies within features. When two features exhibit a significant correlation, it may suggest redundancy, providing an opportunity to eliminate one of them. This elimination process serves to improve the efficiency of the model and mitigate issues related to multicollinearity. Utilizing this approach assists in the curation of a varied feature set, ensuring that the selected features collectively contribute to the overall predictive accuracy of the model. We identify highly correlated features in a data frame and drop one feature from each highly correlated pair to reduce redundancy in the dataset. The threshold for identifying highly correlated features is set at 0.9, meaning that if the correlation between two features is greater than or equal to 0.9, they are considered highly correlated. After using the correlation matrix we reduced the number of features significantly in our dataset from 60 to 29.



4.2.4 - **Correlation Matrix and Data Distribution of Selected 29 Features**

Analyzing the feature distribution among various categories (bankruptcy and non-bankruptcy) unveils valuable insights into their ability to distinguish. Features demonstrating unique distributions for both categories are informative and play a substantial role in the classification process. This methodology guarantees the preservation of features that genuinely differentiate between different financial health conditions. Based on the distribution of the data for our features within the dataset we found that some of the features have left-skewed data distribution, some of them have right-skewed data distribution and some of the features have normal data distribution.


Distribution of cash_securities_receivables_liabilities_expenses_depreciation_365
Distribution of sales_assets
Distribution of gross_profit_depreciation_sales
Distribution of liabilities_365_gross_profit_depreciation
Distribution of gross_profit_assets
Distribution of inventory_365_sales
Distribution of sales_n_sales_n_minus_1
Distribution of gross_profit_3_years_assets
Distribution of OA_profit_expenses
Distribution of log_total_assets
Distribution of current_liabilities_365_cost_of_products_sold
Distribution of expenses_short_term_liabilities
Distribution of expenses_liabilities
Distribution of total_sales_assets
Distribution of sales_profit_sales
Distribution of current_assets_inventory_receivables_short_term_liabilities
Distribution of liabilities_OA_profit_depreciation_12_365
Distribution of net_profit_inventory
Distribution of working_capital
Distribution of sales_cost_of_products_sold_sales
Distribution of current_assets_inventory_short_term_liabilities_sales_gross_profit_depreciation
Distribution of total_costs_total_sales
Distribution of long_term_liabilities_equity
Distribution of sales_inventory
Distribution of sales_receivables
Distribution of sales_short_term_liabilities
Distribution of sales_fixed_assets
Distribution of current_assets_inventories_long_term_liabilities
Distribution of class
Count



4.2.5 - **Box Plot for Outlier Detection**

Utilizing box plots as a means of identifying outliers proves valuable for pinpointing extreme values that may negatively impact model performance. The presence of outliers can introduce distortions during the training process, and addressing them through removal or transformation can bolster the model's resilience. The application of box plots assists in the identification of features less susceptible to outlier influence, ultimately reinforcing the reliability of the bankruptcy prediction model. The box plot below clearly shows that the features have different scales and there are quite some outliers in certain features. We will need to scale the data and maybe drop the extreme outliers.

In essence, the incorporation of these techniques for feature extraction plays a pivotal role in crafting a resilient and precise bankruptcy prediction model by strategically refining the feature set for classification. Each technique contributes a distinct perspective to the feature selection process, enhancing the model's interpretability and ability to generalize effectively.

4.3 - DATA PREPROCESSING

4.3.1 - **Train/Test Split**

Train/Test Split is a crucial phase in the process of building a model. It involves dividing the dataset into separate training and testing sets using methodologies like the train_test_split approach. This division guarantees that the model undergoes training on one specific subset and is subsequently assessed on another. The significance of this segregation lies in its role in evaluating the model's overall ability to generalize, providing valuable observations into its capacity for making precise predictions on novel, undisclosed data.

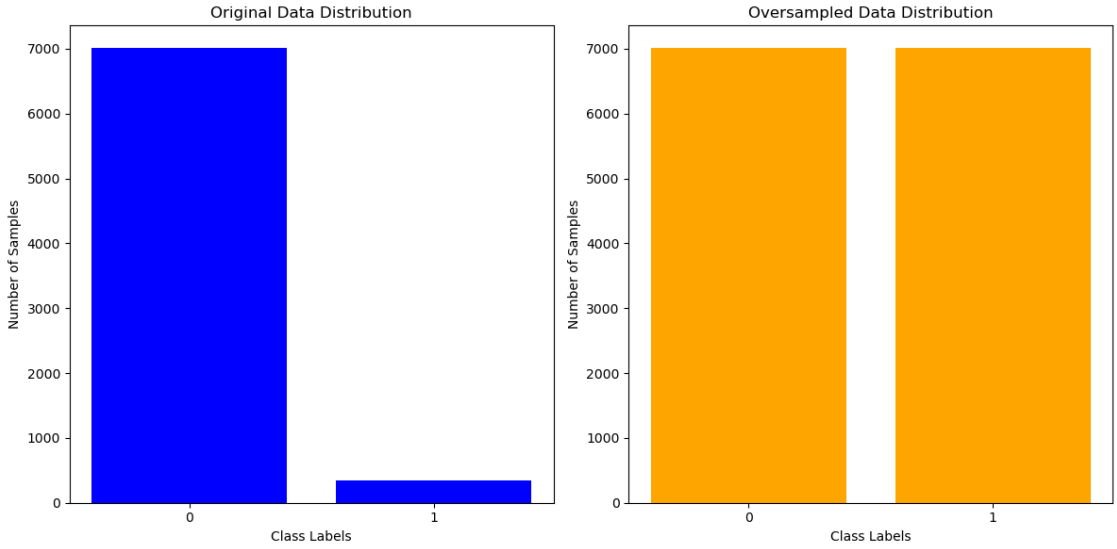
4.3.2 - **Normalization**

It is essential to normalize the features to counteract the impact of diverse scales present in the dataset. In the context of predicting bankruptcy, where financial metrics may exhibit considerable differences in scale, employing normalization techniques such as Z-score normalization is crucial. This approach guarantees that each feature makes a proportional contribution to the training of the model, fostering stability and convergence throughout the learning phase. By doing so, the model becomes adept at identifying meaningful patterns without succumbing to the disproportionate influence stemming from variations in the scales of individual features.

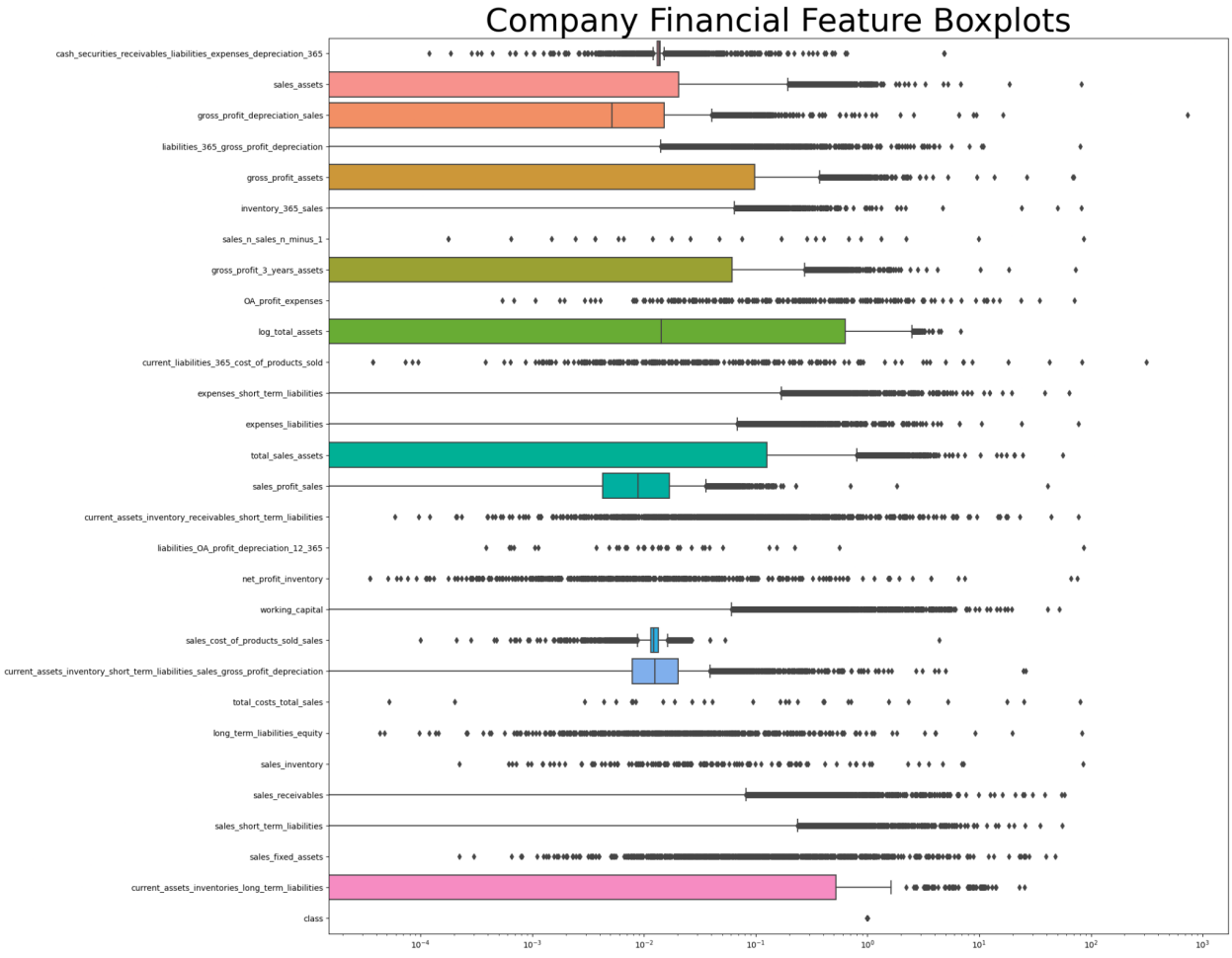
$$Z = \frac{x - \mu}{\sigma}$$

4.3.3 - **Class Imbalance**

Effectively addressing the issue of class imbalance, where the instances of one class significantly outnumber the other, necessitates a strategic approach. An oversampling method, specifically the Synthetic Minority Over-sampling Technique (SMOTE), is employed to artificially generate instances of the minority class. SMOTE, a widely utilized technique, is exemplified in the context of training data with s samples and f features within the feature space, assuming continuous features for simplicity. To illustrate, consider a dataset focused on birds, where the feature space for the minority class includes attributes like beak length, wingspan, and weight. During oversampling, a sample is taken, and its k nearest neighbors are considered in the feature space. A synthetic data point is then created by multiplying the vector between one of these neighbors and the current data point by a random number x between 0 and 1 and adding it to the current data point. The implementation of SMOTE from the imbalanced-learn library enhances the model's exposure to underrepresented cases, preventing biased learning and improving its ability to identify subtle patterns associated with potential bankruptcies.



After applying the above three processes, we created a box plot to see the impact of data scaling and the subsequent result on the outliers.



5. IMPLEMENTED MODELS

The basic flow in implementing the classification models with our project is we create the model from scratch, and we perform hyperparameter tuning in selected models, for the rest of the model, instead of tuning the hyperparameters we use a different number of records (subset sample) for each of those specific model trained to see the impact of the size of the dataset. For all four models, we are going to run the model on Class class-balanced data and class-imbalanced data to visualize the effect of SMOTE. We also are going to perform a Bias-Variance trade-off for all 4 models. The four models that we have created and implemented from scratch are

5.1 - LOGISTIC REGRESSION

The assumptions for the data to apply Logistic regression were also evaluated like No High Multicollinearity was handled by dropping features in the Feature Selection module. We also took care of extreme outliers when applying Scaling to the dataset so that they do have a significant impact on the model. Logistic regression serves as a linear classification model, commonly referred to as logit regression, maximum-entropy classification (MaxEnt), or the log-linear classifier. Within this framework, the logistic function is employed to model the probabilities associated with potential outcomes in a singular trial. Logistic Regression is going to be our baseline model.

The following equation is going to be the sigmoid function used to create our Logistic Regression model.

$$\phi(z) = \frac{1}{1 + e^{-z}}$$

The below equation is the gradient descent formula for updating the weights of the Logistic Regression model:

$$\theta_j := \theta_j - \frac{\alpha}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

5.2 - GAUSSIAN NAIVE BAYES CLASSIFIER

The assumptions for the data before applying Gaussian Naive Bayes are fulfilled like Independent Features within the dataset are handled by dropping highly collinear features. Some of the features within the dataset follow a normal Gaussian distribution while some have left-skewed and right-skewed data distribution. The Naive Bayes classifier stands as a supervised learning algorithm, rooted in the application of Bayes' theorem under the "naive" presumption of independence between all pairs of features. In the context of a class variable y and a dependent feature vector $x1$ to xn, the theorem articulates a specific relationship:

$$P(y \mid x_1, \ \ldots, \ x_n) = \frac{P(y)P(x_1, \ \ldots, \ x_n \mid y)}{P(x_1, \ \ldots, \ x_n)}$$

After using the naive independence assumption we get

$$P(x_i \mid y, \ x_1, \ \ldots, \ x_{i-1}, \ x_{i+1}, \ \ldots x_n) = P(x_i \mid y)$$

this value is simplified to the following for all i:

$$P(y \mid x_1, \ \ldots, \ x_n) = \frac{P(y) \prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \ \ldots, \ x_n)}$$

As $P(x1, \ldots, xn)$ remains constant concerning the input, we can employ the subsequent classification guideline:

$$P(y \mid x_1, \ \ldots, \ x_n) \propto P(y) \prod_{i=1}^{n} P(x_i \mid y)$$

The Gaussian Naive Bayes application utilizes the Gaussian Naive Bayes algorithm for classification purposes. The concept of Maximum Likelihood Estimation is used to find the values of uy (mean) and σy^2 (variance). The underlying assumption in this approach is that the probability distribution of the features follows a Gaussian distribution:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

5.3 - <u>DECISION TREE CLASSIFIER</u>

The two basic assumptions before applying the Decision Tree classifier are Independent Features and Data Balance which both have been dealt with within the Feature Selection and Data Pre-Processing section of this report. Decision Trees (DTs) represent a non-parametric approach to supervised learning, serving purposes in both classification and regression tasks. In our specific classification objective, we aim to construct a model predicting the outcome of a target variable (denoted as y, indicating the likelihood of a firm experiencing bankruptcy). This prediction is based on discerning straightforward decision rules derived from various data features (specifically, $x1$ through $x29$, encompassing the financial distress variables of a firm). As the decision tree is constructed, the data is structured in records in the format (x, Y), with x representing the financial distress variables and Y indicating the target variable. Our model assigns equal importance to all features, treating them with equal weight during the decision tree's construction. Throughout this process, we utilize the 'Entropy' index as a criterion for assessing the quality of each split.

$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

5.4 - SUPPORT VECTOR MACHINES

We have already dealt with the assumptions of the data before applying SVM like Outliers Detection and Data Imbalance when working on the Feature Engineering and Data Pre-Processing section. Support Vector Machines (SVM) emerge as a potent and adaptable tool in the domain of machine learning, particularly when applied to the prediction of bankruptcy classification. SVMs showcase proficiency in managing both linear and non-linear associations within data, rendering them apt for uncovering complex patterns intrinsic to financial datasets. This discussion delves into the fundamentals of SVM and its utilization in the crucial task of forecasting bankruptcies. We are creating a Soft Margin SVM model based on this equation:

$$\begin{aligned} min \quad & \frac{1}{2} w^T w + C \sum_{i=1}^{m} \xi_i \\ s.t. \quad & y_i (x_i^T w + b) \geq 1 - \xi_i \qquad \xi_i \geq 0 \end{aligned}$$

Essentially, SVM functions by discovering a hyperplane that optimally segregates the data into distinct classes while maximizing the margin, defined as the separation between the hyperplane and the closest data points of each class. In the realm of bankruptcy prediction, SVM aims to pinpoint a hyperplane that effectively distinguishes between financially distressed and stable companies based on pertinent features.

The SVM decision function can be expressed as

$$f(x) = sign(w \cdot x + b)$$

In this equation, w denotes the weight vector, x signifies the input feature vector, and b represents the bias term. The sign function plays a pivotal role in determining the predicted class, with positive values indicating one class and negative values denoting the other. To summarize, Support Vector Machines, with their capacity to adapt to both linear and non-linear patterns and address class imbalances, emerge as a formidable asset for predicting bankruptcy classifications. The meticulous selection of suitable kernels and parameters empowers SVM to navigate the complexities of financial data, contributing to resilient and precise predictions in identifying companies susceptible to bankruptcy risks.

5.5 BIAS-VARIANCE TRADE-OFF

Understanding the intricate interaction between bias and variance is pivotal in the domain of machine learning, directly influencing the efficacy of classification models. This discussion delves

into the concept of the Bias-Variance Trade-Off and its repercussions on four prominent classification models: Logistic Regression, Gaussian Naive Bayes, Decision Tree Classifier, and Support Vector Machines (SVM).

$$\sigma^2 + \underbrace{\left(\mathbb{E}\left[\hat{f}(x_0)\right] - f(x_0)\right)^2}_{\text{Bias}^2} + \underbrace{\mathbb{E}\left[\left(\hat{f}(x_0) - \mathbb{E}\left[\hat{f}(x_0)\right]\right)^2\right]}_{\text{Variance}}$$

Logistic Regression is a simple and interpretable linear model known for its reduced variance but potential bias in the face of complex, non-linear data. It is suitable for straightforward datasets where interpretability is crucial.

Gaussian Naive Bayes, based on Bayes' theorem and assuming feature independence, is efficient for tasks like text classification. It demonstrates minimal variance but introduces bias due to the assumption of naive independence, making it favorable for specific applications.

The **Decision Tree Classifier** captures intricate data relationships but may exhibit elevated variance and overfitting. Striking a balance, such as controlling tree depth and implementing pruning, is crucial. Decision trees are valuable for scenarios prioritizing interpretability and identifying non-linear patterns.

Support Vector Machines (SVM) strike a balance between bias and variance by maximizing the margin between classes. Robust against overfitting, SVMs excel in high-dimensional spaces. However, the introduction of a kernel function adds a bias-variance trade-off, requiring careful selection for optimal performance.

To enhance model performance, Principal Component Analysis (PCA) was applied to a class-balanced dataset to explore potential feature reduction based on cumulative explained variance.

5.6 PRINCIPAL COMPONENT ANALYSIS

Principal Component Analysis (PCA) emerges as a potent statistical method extensively utilized in the examination of data and machine learning to streamline and distill crucial insights from datasets with high dimensionality. By converting the initial variables into a fresh set of independent

variables termed principal components, PCA facilitates a more streamlined representation of the data. This discussion delves into the fundamental principles of PCA, its practical applications, and the mathematical underpinnings that constitute its foundation.

Reduction of Dimensionality: PCA stands out in diminishing the dimensionality of datasets while conserving the majority of vital information. This proves pivotal in situations where managing an extensive array of variables becomes computationally burdensome or susceptible to the risk of overfitting.

Eigenvalues and Eigenvectors: At the heart of PCA lie eigenvalues and eigenvectors. In the context of PCA, an analysis of the covariance matrix of the original data unveils these eigenvalues and eigenvectors. These eigenvalues signify the variance of the data along the corresponding eigenvectors, which function as the principal components.

The covariance matrix C is calculated as

$$C = \frac{1}{n-1} \sum_{i=1}^{n} (X_i - \bar{X})(X_i - \bar{X})^T$$

where Xi represents the data points and X(bar) is the mean vector.

The eigenvalue equation is given by:

$$\mathbf{C}\boldsymbol{v} = \boldsymbol{\lambda}\boldsymbol{v}$$

where λ is the eigenvalue and v is the corresponding eigenvector.

Principal components are derived by organizing eigenvectors in descending order based on their corresponding eigenvalues. These components establish a fresh basis for the data, and opting for a subset among them facilitates reducing dimensionality while conserving the utmost variance within the dataset.

In summary, Principal Component Analysis stands as an essential instrument for data scientists and analysts aiming to grasp and simplify intricate datasets. The mathematical basis involving eigenvalues and eigenvectors establishes a robust groundwork for comprehending the transformation process. By integrating PCA into analytical workflows, practitioners can unveil valuable insights, optimize computational efficiency, and augment the interpretability of their findings.

6. RESULTS

After creating multiple scenarios for each model and selecting the most suitable one, the provided table presents a concise summary of our test set results. To facilitate the comparison of algorithms, we have consolidated accuracy metrics from various models into a single table.

Logistic regression with Hyperparameter Tuning on Class Imbalanced Dataset:

	Learning Rate	Epsilon	Training F1 Score	Training Recall	Training Precision	Training Accuracy	Testing F1 Score	Testing Recall	Testing Precision	Testing Accuracy
0	0.01000	0.05000	0.015748	0.008671	0.085714	0.948993	NaN	0.0	0.0	0.946684
1	0.01000	0.00500	0.022039	0.011561	0.235294	0.951714	NaN	0.0	0.0	0.950175
2	0.01000	0.00050	0.022039	0.011561	0.235294	0.951714	NaN	0.0	0.0	0.950175
3	0.01000	0.00005	0.022039	0.011561	0.235294	0.951714	NaN	0.0	0.0	0.950175
4	0.01000	0.00050	0.022039	0.011561	0.235294	0.951714	NaN	0.0	0.0	0.950175
5	0.00100	0.05000	0.011050	0.005780	0.125000	0.951306	NaN	0.0	0.0	0.950492
6	0.00100	0.00500	0.011236	0.005780	0.200000	0.952122	NaN	0.0	0.0	0.951761
7	0.00100	0.00050	0.005650	0.002890	0.125000	0.952122	NaN	0.0	0.0	0.951761
8	0.00100	0.00005	0.005698	0.002890	0.200000	0.952530	NaN	0.0	0.0	0.952396
9	0.00100	0.00050	0.005650	0.002890	0.125000	0.952122	NaN	0.0	0.0	0.951761
10	0.00010	0.05000	NaN	0.000000	0.000000	0.950898	NaN	0.0	0.0	0.950175
11	0.00010	0.00500	NaN	0.000000	0.000000	0.950898	NaN	0.0	0.0	0.950175
12	0.00010	0.00050	0.005571	0.002890	0.076923	0.951442	NaN	0.0	0.0	0.951127
13	0.00010	0.00005	0.005634	0.002890	0.111111	0.951986	NaN	0.0	0.0	0.951761
14	0.00010	0.00050	0.005571	0.002890	0.076923	0.951442	NaN	0.0	0.0	0.951127
15	0.00001	0.05000	NaN	0.000000	0.000000	0.950762	NaN	0.0	0.0	0.950175
16	0.00001	0.00500	NaN	0.000000	0.000000	0.950762	NaN	0.0	0.0	0.950175
17	0.00001	0.00050	NaN	0.000000	0.000000	0.950762	NaN	0.0	0.0	0.950175
18	0.00001	0.00005	0.005571	0.002890	0.076923	0.951442	NaN	0.0	0.0	0.951127
19	0.00001	0.00050	NaN	0.000000	0.000000	0.950762	NaN	0.0	0.0	0.950175

Support Vector Machines - Hyperparameter Tuning on Class Imbalance Dataset:

	Sample Size	C	Training F1 Score	Training Recall	Training Precision	Training Accuracy	Testing F1 Score	Testing Recall	Testing Precision	Testing Accuracy
0	200.0	0.5	0.880000	0.444444	0.173913	0.250000	0.859410	0.107383	0.049080	0.067368
1	200.0	1.0	0.945000	0.222222	0.333333	0.266667	0.925421	0.020134	0.032609	0.024896
2	200.0	1.5	0.935000	0.000000	0.000000	NaN	0.933354	0.013423	0.030769	0.018692
3	200.0	2.0	0.940000	0.111111	0.200000	0.142857	0.929546	0.006711	0.013333	0.008929
4	400.0	0.5	0.957500	0.055556	1.000000	0.105263	0.949540	0.000000	0.000000	NaN
5	400.0	1.0	0.940000	0.055556	0.125000	0.076923	0.937480	0.006711	0.020000	0.010050
6	400.0	1.5	0.952500	0.055556	0.333333	0.095238	0.946049	0.006711	0.043478	0.011628
7	400.0	2.0	0.957500	0.166667	0.600000	0.260870	0.944462	0.020134	0.093750	0.033149
8	600.0	0.5	0.946667	0.066667	0.333333	0.111111	0.947001	0.013423	0.090909	0.023392
9	600.0	1.0	0.951667	0.100000	0.600000	0.171429	0.949222	0.013423	0.133333	0.024390
10	600.0	1.5	0.951667	0.100000	0.600000	0.171429	0.947953	0.006711	0.058824	0.012048
11	600.0	2.0	0.950000	0.133333	0.500000	0.210526	0.937798	0.013423	0.039216	0.020000
12	800.0	0.5	0.953750	0.085714	0.375000	0.139535	0.944779	0.006711	0.037037	0.011364
13	800.0	1.0	0.953750	0.085714	0.375000	0.139535	0.944145	0.020134	0.090909	0.032967
14	800.0	1.5	0.956250	0.085714	0.500000	0.146341	0.945732	0.013423	0.076923	0.022857
15	800.0	2.0	0.948750	0.085714	0.250000	0.127660	0.936528	0.013423	0.036364	0.019608

Decision Tree - Subsample class imbalance dataset of 500 records - Hyperparameter Tuning

	Max Depth	Training F1 Score	Training Recall	Training Precision	Training Accuracy	Testing F1 Score	Testing Recall	Testing Precision	Testing Accuracy
0	2.0	NaN	0.000000	NaN	0.958	NaN	0.000000	NaN	0.952713
1	3.0	0.625000	0.476190	0.909091	0.976	0.390698	0.281879	0.636364	0.958426
2	4.0	0.645161	0.476190	1.000000	0.978	0.386473	0.268456	0.689655	0.959695
3	5.0	0.800000	0.761905	0.842105	0.984	0.298013	0.302013	0.294118	0.932720
4	6.0	0.923077	0.857143	1.000000	0.994	0.280255	0.295302	0.266667	0.928277

Logistic regression with Hyperparameter Tuning on Class Balanced Dataset:

	Learning Rate	Epsilon	Training F1 Score	Training Recall	Training Precision	Training Accuracy	Testing F1 Score	Testing Recall	Testing Precision	Testing Accuracy
0	0.01000	0.05000	0.682951	0.973737	0.525902	0.547959	0.098160	0.966443	0.051706	0.160267
1	0.01000	0.00500	0.682951	0.973737	0.525902	0.547959	0.098160	0.966443	0.051706	0.160267
2	0.01000	0.00050	0.682951	0.973737	0.525902	0.547959	0.098160	0.966443	0.051706	0.160267
3	0.01000	0.00005	0.682951	0.973737	0.525902	0.547959	0.098160	0.966443	0.051706	0.160267
4	0.01000	0.00050	0.682951	0.973737	0.525902	0.547959	0.098160	0.966443	0.051706	0.160267
5	0.00100	0.05000	0.682817	0.971596	0.526369	0.548673	0.097845	0.959732	0.051550	0.163123
6	0.00100	0.00500	0.682817	0.971596	0.526369	0.548673	0.097845	0.959732	0.051550	0.163123
7	0.00100	0.00050	0.682817	0.971596	0.526369	0.548673	0.097845	0.959732	0.051550	0.163123
8	0.00100	0.00005	0.682817	0.971596	0.526369	0.548673	0.097845	0.959732	0.051550	0.163123
9	0.00100	0.00050	0.682817	0.971596	0.526369	0.548673	0.097845	0.959732	0.051550	0.163123
10	0.00010	0.05000	0.657811	0.682130	0.635167	0.645161	0.154795	0.758389	0.086194	0.608378
11	0.00010	0.00500	0.657811	0.682130	0.635167	0.645161	0.154795	0.758389	0.086194	0.608378
12	0.00010	0.00050	0.663744	0.691550	0.638088	0.649657	0.156057	0.765101	0.086890	0.608696
13	0.00010	0.00005	0.681886	0.718384	0.648917	0.664859	0.157133	0.765101	0.087558	0.611869
14	0.00010	0.00050	0.663744	0.691550	0.638088	0.649657	0.156057	0.765101	0.086890	0.608696
15	0.00001	0.05000	0.656291	0.680131	0.634065	0.643805	0.154477	0.758389	0.085997	0.607426
16	0.00001	0.00500	0.656291	0.680131	0.634065	0.643805	0.154477	0.758389	0.085997	0.607426
17	0.00001	0.00050	0.656291	0.680131	0.634065	0.643805	0.154477	0.758389	0.085997	0.607426
18	0.00001	0.00005	0.656291	0.680131	0.634065	0.643805	0.154477	0.758389	0.085997	0.607426
19	0.00001	0.00050	0.656291	0.680131	0.634065	0.643805	0.154477	0.758389	0.085997	0.607426

Support Vector Machines - Hyperparameter Tuning on Class Imbalance Dataset:

	Sample Size	C	Training F1 Score	Training Recall	Training Precision	Training Accuracy	Testing F1 Score	Testing Recall	Testing Precision	Testing Accuracy
0	200.0	0.5	0.730000	0.826923	0.704918	0.761062	0.549667	0.744966	0.074397	0.135283
1	200.0	1.0	0.690000	0.798077	0.669355	0.728070	0.503332	0.812081	0.072979	0.133924
2	200.0	1.5	0.665000	0.625000	0.698925	0.659898	0.670898	0.704698	0.095628	0.168404
3	200.0	2.0	0.705000	0.730769	0.710280	0.720379	0.605839	0.738255	0.083778	0.150479
4	400.0	0.5	0.697500	0.802885	0.676113	0.734066	0.535703	0.818792	0.078306	0.142941
5	400.0	1.0	0.707500	0.793269	0.690377	0.738255	0.512218	0.825503	0.075275	0.137970
6	400.0	1.5	0.717500	0.802885	0.698745	0.747204	0.562044	0.778523	0.079289	0.143921
7	400.0	2.0	0.735000	0.802885	0.719828	0.759091	0.553475	0.791946	0.078983	0.143640
8	600.0	0.5	0.706667	0.824104	0.674667	0.741935	0.545858	0.785235	0.077177	0.140541
9	600.0	1.0	0.698333	0.791531	0.675000	0.728636	0.579181	0.785235	0.082920	0.150000
10	600.0	1.5	0.711667	0.817590	0.682065	0.743704	0.578864	0.805369	0.084626	0.153159
11	600.0	2.0	0.703333	0.794788	0.679666	0.732733	0.585528	0.765101	0.082310	0.148631
12	800.0	0.5	0.693750	0.750600	0.689427	0.718714	0.608696	0.778523	0.088146	0.158362
13	800.0	1.0	0.702500	0.772182	0.692473	0.730159	0.598858	0.812081	0.089167	0.160691
14	800.0	1.5	0.697500	0.772182	0.686567	0.726862	0.601714	0.818792	0.090370	0.162775
15	800.0	2.0	0.697500	0.738609	0.698413	0.717949	0.640114	0.765101	0.093982	0.167401

Decision Tree - Subsample class imbalance dataset of 500 records - Hyperparameter Tuning

	Max Depth	Training F1 Score	Training Recall	Training Precision	Training Accuracy	Testing F1 Score	Testing Recall	Testing Precision	Testing Accuracy
0	2.0	NaN	0.000000	NaN	0.958	NaN	0.000000	NaN	0.952713
1	3.0	0.625000	0.476190	0.909091	0.976	0.390698	0.281879	0.636364	0.958426
2	4.0	0.645161	0.476190	1.000000	0.978	0.386473	0.268456	0.689655	0.959695
3	5.0	0.800000	0.761905	0.842105	0.984	0.298013	0.302013	0.294118	0.932720
4	6.0	0.923077	0.857143	1.000000	0.994	0.280255	0.295302	0.266667	0.928277

		Training			
Model	**Dataset**	**F1-Score**	**Precision**	**Recall**	**Accuracy**
Logistic Regression	Class Imbalanced	0.011236	0.200000	0.005780	0.952122
	Class Balanced	0.681886	0.648917	0.718384	0.664859
Support Vector Machines	Full Data - Class Imbalance	0.944	0.4	0.074	0.125
	Full Data - Class Balance	0.718	0.7167	0.7835	0.7486

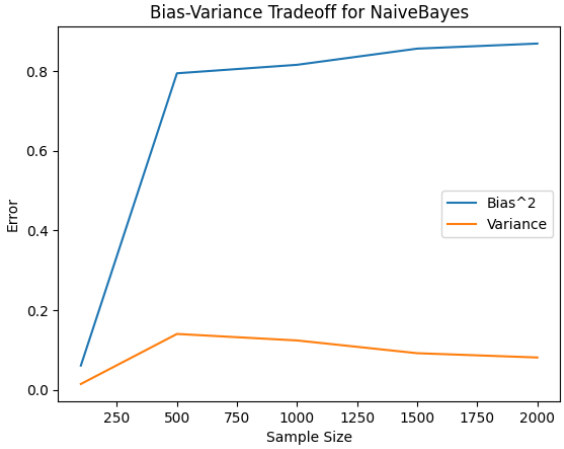
		Testing			
Model	**Dataset**	**F1-Score**	**Precision**	**Recall**	**Accuracy**
Logistic Regression	Class Imbalanced	Nan	0.000000	0.000000	0.951761
	Class Balanced	0.157133	0.087558	0.765101	0.611869
Naive Bayes	Class Imbalanced	0.0925	0.048	0.973	0.098
	Class Balanced	0.092377	0.048524	0.959732	0.108220
Support Vector Machines	Full Data - Class Imbalance	0.943	0.03	0.0067	0.011
	Full Data - Class Balance	0.5982	0.090	0.8322	0.1669
Decision Tree Classifier	500 Sample - Class Imbalance	0.386	0.689	0.2684	0.959
	500 Sample - Class Balance	0.2571	0.175	0.4832	0.8679

Bias Variance Trade-Off Analysis

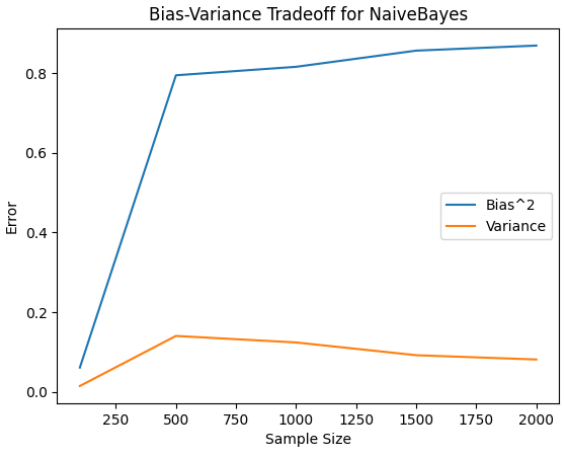
Logistic Regression with Class Imbalance

Logistic Regression with Class Balance

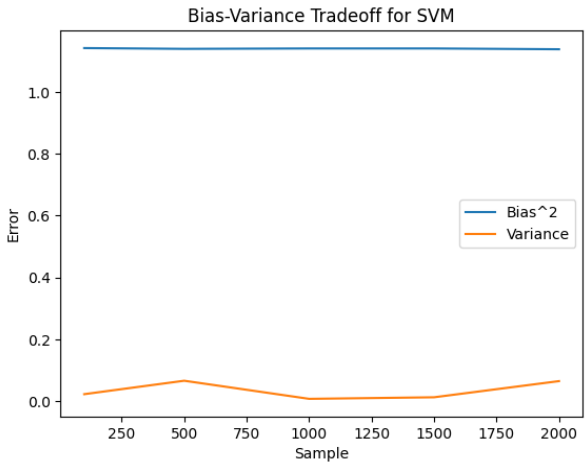
Naive Bayes with Class Imbalance



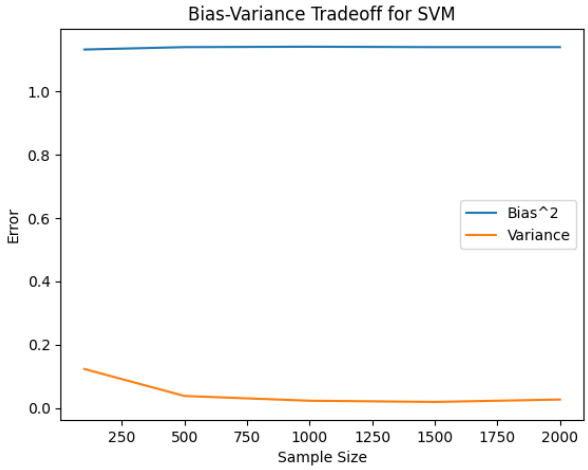
Naive Bayes with Class Balance



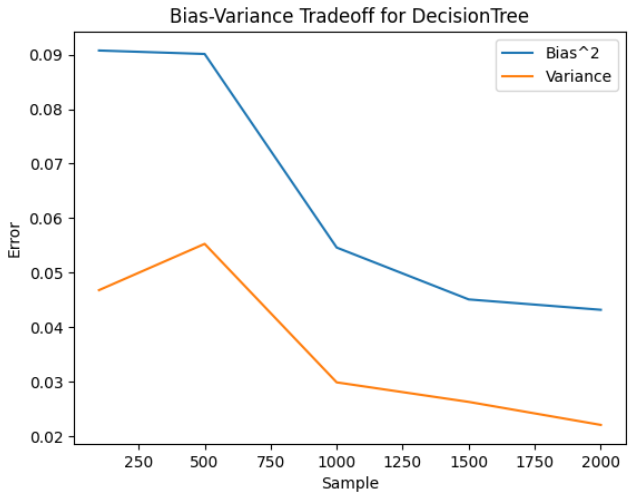
Support Vector Machine with Class Imbalance



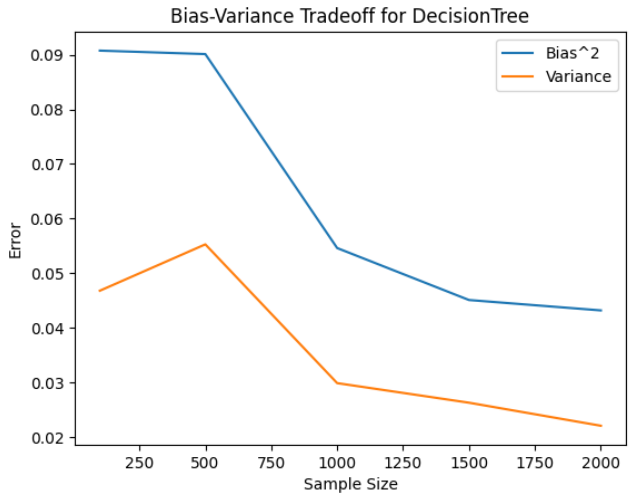
Support Vector Machine with Class Balance



Decision Tree with Class Imbalance



Decision Tree with Class Balance



To evaluate the expenses associated with each model, we have aggregated the duration for training the model and executing the for loop, along with the time required to fit the optimal model.

Model	Dataset	Time Taken to tune the model	Time Taken to fit the best model
Logistic Regression	Class Imbalanced	10.2 s	615 ms
	Class Balanced	42.5 s	757 ms
Naive Bayes	Class Imbalanced	NaN	16 s
	Class Balanced	NaN	15.9 s
Support Vector Machines	500 Sample - Class Imbalanced	11.9 µs	849 ms
	500 Sample - Class Balanced	16.607 s	2.5 s
Decision Tree Classifier	500 Sample - Class Imbalanced	208.14 s	41.7 s
	500 Sample - Class Balanced	3min 25s	42.7 s

Bias Variance Trade-Off Analysis on Class Imbalanced Data - 12 min 59s

Bias Variance Trade-Off Analysis on Class Balanced Data - 13 min 15s

7. DISCUSSION

Logistic Regression:

1) Class Imbalanced (Training):
    - Low F1-Score (0.011236): Indicates poor performance in capturing the minority class (bankrupt companies).
    - Low Recall (0.005780): Fails to identify a significant number of actual bankrupt companies.
    - High Accuracy (0.952122): Misleadingly high due to the dominance of the non-bankrupt class.

2) Class Imbalanced (Testing):
    - Undefined F1-Score (Nan): Likely due to zero true positives, indicating a failure to predict bankrupt companies.
    - Low Precision and Recall (0, 0): Fails to predict any bankrupt companies.

The dataset displays a considerable imbalance between classes, with '0' appearing 7000 times, while '1' is present only 245 times. The existence of "NaN" values in specific metrics, particularly in the identification of '1', indicates difficulties in precisely capturing and categorizing the minority class due to its infrequent instances. The model demonstrates strong accuracy, especially in effectively categorizing occurrences of '0'. This success is linked to the dataset's skewed distribution, resulting in a dominant representation and a substantial number of accurate predictions for this class. The uneven distribution of data poses challenges for the model in generalizing well to instances of '1'. The scarcity of '1' instances may lead to the model struggling to identify meaningful patterns for this class, resulting in the appearance of "NaN" values. To sum up, the logistic regression model encounters obstacles in accurately identifying instances of the minority class ('1') on the imbalanced dataset, leading to the presence of "NaN" values in specific metrics. The model's high accuracy primarily arises from its effectiveness in correctly categorizing the majority class ('0') due to its prevalent presence in the dataset.

3) Class Balanced (Training):
    - Significantly Improved F1-Score (0.681886): A balanced dataset allows the model to capture both classes effectively.
    - Balanced Precision and Recall (0.648917, 0.718384): Indicates better overall performance.
    - Reduced Accuracy (0.664859): Reflects the trade-off for better balance.

4) Class Balanced (Testing):
    - Improved F1-Score (0.157133): Better performance compared to the imbalanced test set.
    - Balanced Precision and Recall (0.087558, 0.765101): Indicates improved ability to capture bankrupt companies.
    - Reduced Accuracy (0.611869): Reflects the trade-off for improved balance.

In brief, the logistic regression model, when utilized on well-balanced data with fine-tuned hyperparameters, exhibits strong effectiveness in attaining a synergistic equilibrium between precision and recall, crucial for proficient classification assignments. In general, it enhances outcomes when contrasted with the performance of the logistic regression model on imbalanced datasets.

Support Vector Machines:

1) Full Data - Class Imbalance (Training):
    - High F1-Score (0.944): Indicates effective performance on the imbalanced training data.
    - Balanced Precision and Recall (0.4, 0.074): Shows challenges in precision and recall due to class imbalance.
    - Low Accuracy (0.125): Skewed by the majority class.

2) Full Data - Class Imbalance (Testing):
    - Maintained High F1-Score (0.943): Indicates good generalization to the test set.
    - Low Precision and Recall (0.03, 0.0067): Reflects challenges in predicting the minority class.

3) Full Data - Class Balance (Training):
    - Slightly Reduced F1-Score (0.718): Reflects the trade-off for better balance.
    - Balanced Precision and Recall (0.7167, 0.7835): Indicates effective classification across both classes.
    - Balanced Accuracy (0.7486): This represents a trade-off for improved balance.

4) Full Data - Class Balance (Testing):
    - Maintained High F1-Score (0.5982): Decent performance, but slightly reduced compared to the imbalanced case.
    - Balanced Precision and Recall (0.09, 0.8322): Indicates a better trade-off between precision and recall.

When assessed on the fully populated dataset with class imbalance during training, the Support Vector Machines (SVM) exhibited a commendable F1-Score of 0.944, indicating effective performance despite the skewed class distribution. However, the balanced precision and recall values of 0.4 and 0.074, respectively, underscored the inherent challenges associated with precision and recall due to the uneven class distribution. The low accuracy of 0.125 further emphasized the impact of the majority class on the model's overall accuracy.

In the testing phase on the imbalanced dataset, the SVM maintained a high F1-Score of 0.943, suggesting good generalization to the test set. Nevertheless, the low precision and recall values of 0.03 and 0.0067 highlighted the model's struggles in predicting the minority class.

Upon transitioning to a balanced dataset for training, the SVM demonstrated a slightly reduced F1-Score of 0.718, indicative of the trade-off made for achieving a better balance between the classes. The balanced precision and recall values of 0.7167 and 0.7835, respectively, underscored

the model's effectiveness in classification across both classes, striking a balance between precision and recall. The balanced accuracy of 0.7486 represented a compromise for improved class balance.

In the testing phase with a balanced dataset, the SVM maintained a relatively high F1-Score of 0.5982, showcasing decent performance despite a slight reduction compared to the imbalanced case. The balanced precision and recall values of 0.09 and 0.8322 suggested a better trade-off between precision and recall, highlighting the model's improved performance on both classes in a balanced setting.

Naive Bayes:

1) Class Imbalanced (Testing):
    - Low F1-Score (0.0925): Struggles with the imbalanced test data.
    - Low Precision (0.048): Many false positives.
    - High Recall (0.973): Captures most of the true positives.

When applying the Naive Bayes technique to our dataset comprising 10,500 records, a substantial disparity in the frequency of occurrences for the labels '1' and '0' was observed. Specifically, '1' exhibited a significantly higher frequency, appearing 7,000 times, whereas '0' was much less frequent, appearing only 245 times. Additionally, our data deviated from a normal distribution pattern, a characteristic we acknowledged during the preprocessing phase. Despite the departure from a Gaussian pattern, we opted to employ Naive Bayes to explore its performance. Our findings revealed that Naive Bayes excelled in identifying the less common instances of '1,' achieving a detection rate of 97.32%. However, when considering the broader context, its overall accuracy struggled. The metrics of F1 score, precision, and accuracy displayed notably lower values. This indicates that while Naive Bayes demonstrates proficiency in detecting rare cases, it encounters challenges in achieving accurate results across both types of cases. Consequently, this underscores the significance of the dataset's structure and the imbalance in occurrence frequencies, influencing the efficacy of models like Naive Bayes.

2) Class Balanced (Testing):
    - Similar F1-Score (0.092377): Limited improvement over the imbalanced case.
    - Balanced Precision and Recall (0.048524, 0.959732): Better trade-off between precision and recall.
    - Reduced Accuracy (0.108220): Still affected by the imbalance.

When evaluating the performance of Gaussian Naive Bayes on a dataset containing an equal proportion of positive and negative examples, it was observed that the model demonstrated effectiveness in identifying positive instances. However, its overall performance was suboptimal, exhibiting challenges in achieving accuracy and precision for both positive and negative classifications. This limitation arises from the non-conformity of the features, or observed elements, to a Gaussian distribution pattern. Consequently, the model's efficacy is compromised even when dealing with a balanced dataset.

Decision Tree Classifier:

1) 500 Sample - Class Imbalance (Training):
    - Moderate F1-Score (0.386): Struggles with capturing the minority class.
    - High Precision (0.689) and Recall (0.2684): Indicates biased predictions towards the majority class.
    - High Accuracy (0.959): Dominated by the majority class.

2) 500 Sample - Class Balance (Training):
    - Reduced F1-Score (0.2571): Due to better balance.
    - Balanced Precision and Recall (0.175, 0.4832): Better trade-off, but still challenges in capturing minority instances.
    - Balanced Accuracy (0.8679): This represents a trade-off for improved balance.

Upon assessing the performance of the Decision Tree Classifier on a dataset of 500 samples with class imbalance during training, it exhibited a moderate F1-Score of 0.386, indicating difficulties in accurately identifying the minority class. The classifier displayed high Precision (0.689) and Recall (0.2684), suggesting a tendency toward biased predictions in favor of the majority class. Despite achieving a high overall Accuracy of 0.959, this was primarily attributed to the dominance of the majority class.

In contrast, when applied to a balanced dataset of 500 samples during training, the Decision Tree Classifier showed a reduced F1-Score of 0.2571, attributed to the improved balance between classes. The model achieved a balanced Precision and Recall of 0.175 and 0.4832, respectively, signifying a better trade-off but still facing challenges in capturing minority instances. The Balanced Accuracy, recorded at 0.8679, reflects a compromise for enhanced balance in predictions.

8. CONCLUSION

Among the models evaluated, the Decision Tree Classifier trained on a dataset of 500 samples with class imbalance exhibits the highest accuracy, recording an impressive 95.9%. This model, despite struggling with a moderate F1-Score and imbalanced precision and recall values, achieves a high accuracy primarily due to the dominance of the majority class in the dataset. The accuracy metric, while showcasing the model's overall correctness, can be misleading in the presence of imbalanced data, as demonstrated by the Decision Tree Classifier. In scenarios where the majority class significantly outnumbers the minority class, a model may achieve high accuracy by predominantly predicting the majority class. Therefore, while the Decision Tree Classifier excels in overall accuracy, careful consideration of other metrics, such as precision, recall, and F1-Score, is essential to comprehensively assess its performance, especially in the context of imbalanced datasets.

The Decision Tree Classifier trained on a class-balanced dataset of 500 samples exhibits a reduced accuracy compared to its performance on the imbalanced dataset. This can be attributed to the trade-off made in achieving better balance between the classes. When the dataset is balanced, the model faces the challenge of accurately classifying both the majority and minority classes, leading to a more equitable representation of instances from each class. In the imbalanced dataset, the model likely heavily relies on predicting the majority class, which contributes to a high accuracy score. However, when transitioning to a balanced dataset, where both classes are more evenly represented, the model's predictions must adapt to capture instances of both classes. This adjustment may result in more errors in predicting both the majority and minority classes, causing a reduction in accuracy.

The decision tree model might struggle to generalize well to the balanced dataset, as it needs to navigate between the characteristics of both classes. This compromise in accuracy is often seen as a positive trade-off when considering the model's ability to make more balanced predictions across both classes, as reflected in improved precision and recall values. Therefore, the reduced accuracy in the class-balanced scenario is indicative of the model's efforts to achieve a better equilibrium between the two classes rather than favoring the majority class, which was the case in the imbalanced setting.