Company Bankruptcy Analysis and Prediction

by

[redacted]

Artificial Intelligence Fundamentals, CMP7247

Module Coordinator: Dr. Debashish Das

Date: [redacted]

Table of Contents

Abstract

In the dynamic business landscape, understanding and predicting company bankruptcy is crucial for investors, creditors, and managers. This project explores bankruptcy prediction using machine learning models; Random Forest, Multilayer Perceptron (MLP), and Long Short-Term Memory (LSTM) models. Like most bankruptcy datasets, the dataset used in this project is imbalanced and this is addressed using a resampling techniques Synthetic Minority Over-sampling Technique (SMOTE) and Edited Nearest Neighbors (ENN). The dataset used for this project was gotten from Kaggle and it comprises of Public U.S companies.

Introduction

In the dynamic landscape of today's business environment, the ability to predict company bankruptcies holds great significance for investors and policy makers including; banks, regulators and organizational leaders (Narvekar and Guha, 2021). Following Altman's pioneering work on the Z-score model in 1968, the field of Bankruptcy prediction has gained increasing interests from researchers worldwide. In most cases, the research centers around distinguishing failed and non-failed companies using bankruptcy as a defining factor making it a binary classification problem (Shi and Li, 2019). Correctly predicting when a company might go bankrupt has many practical uses in fields like accounting, economics and finance. For instance, it's crucial for auditors to evaluate if a business can continue. So, improving how we predict bankruptcy can have a big impact on everyone involved in financial markets (Garcia, 2022).

Bankruptcy prediction models fall into distinct categories: statistical and machine learning models (Zelenkov and Volodarskiy, 2021). Advancements made in artificial intelligence coupled with the availability of computational resources have enabled the transition from statistical approaches to modern machine learning techniques (Zelenkov and Volodarskiy, 2021). Neural networks, Deep learning models and ensemble machine learning models such as Random Forests have been found to be particularly effective for bankruptcy prediction (Wang and Liu, 2021).

Despite remarkable progress and efforts, the practical application of existing bankruptcy prediction models remains a challenge. (Zelenkov and Volodarskiy, 2021). Company bankruptcy rates are inherently low, bankrupt companies make up a small fraction of the total observations, and as such bankruptcy datasets are typically imbalanced. This makes it challenging to develop accurate bankruptcy prediction models due to the scarcity of bankruptcy data points available for training and validation (Garcia, 2022). A widely used method for addressing class imbalance involves data preprocessing techniques like resampling (over sampling and under sampling). A prominent technique is the synthetic minority oversampling technique (SMOTE), which generates synthetic instances from the minority class to create a more balanced the dataset (Garcia, 2022).

Background

Predicting if a company will go bankrupt is a critical concern for various stakeholders like regulators, investors, and the companies themselves (Garcia, 2022). The implications and significant cost that bankruptcy can cause for the economy has led to intensified interest from researchers and financial institutions (Lombardo et al., 2022).

Emergence of Statistical Models

Traditionally, the focus of bankruptcy prediction was on accounting-based financial ratios and statistical models (Garcia, 2022). Research evolved from univariate ratio analysis to multivariate approaches like discriminant analysis and logistic regression. However, these methods are limited by their reliance on linear relationships and independence of dataset features (Garcia, 2022). This traditional approach, while providing insights, has struggled in terms of generalization across industries and time periods (Lombardo et al., 2022).

Advancements in Machine Learning

The development of advanced machine learning techniques has propelled the field of bankruptcy prediction into new territories. Machine learning models, known for their adaptability and capacity to handle large datasets, have replaced traditional statistical methods as the go-to tools predicting bankruptcy (Lombardo et al., 2022). Ensemble methods like Random Forests have emerged as formidable contenders (Smiti and Soui, 2020). While Neural Networks and deep learning techniques have outperformed traditional statistical techniques due to them being able to capture the nonlinear relationships in bankruptcy data. (Garcia, 2022). However, this journey has not been without its challenges, including the need to address class imbalance which is present in most bankruptcy datasets (Lombardo et al., 2022).

Aim and Objectives

The aim of this study is to evaluate and compare the results of Random Forest, neural network and deep learning models to predict bankruptcy in American public companies listed on the New York Stock Exchange and NASDAQ. The objectives of this project are:

- To provide a comprehensive description of the dataset used for bankruptcy prediction.
- To address the problem of bankruptcy prediction and its significance.
- To implement and assess multiple machine learning models, including Random Forest, Neural Network, and Deep Learning, for bankruptcy prediction.
- To summarize the approach employed for model development.
- To conduct data pre-processing, visualization, and feature selection, with references to Python code
- To perform model training, evaluation, and testing using appropriate performance metrics and Python code
- To compare the results achieved by different AI models, supported by tables, figures.

Dataset Description

This dataset used in this study is gotten from Kaggle and it contains accounting data related to American public companies listed on the New York Stock Exchange and NASDAQ collected from 1999 to 2018. The dataset contains a total of 78,682 rows and 18 distinct features with the target column representing the company's bankruptcy status. The values in the target column are binary in nature; a value of "alive" means the company is not bankrupt while a value of "failed" means the company is bankrupt.

To use the dataset, I had to load it into my notebook environment using the code below:

```
data = pd.read_csv("american_bankruptcy.csv")
data.head()
```

Figure 1: Code used for loading in the datatset

	company_name	status_label	year	X1	X2	X3	X4	X5	X6	X7	...	X9	X10	X11	X12	X13	X14	
0	C_1	alive	1999	511.267	833.107	18.373	89.031	336.018	35.163	128.348	...	1024.333	740.998	180.447	70.658	191.226	163.816	201
1	C_1	alive	2000	485.856	713.811	18.577	64.367	320.590	18.531	115.187	...	874.255	701.854	179.987	45.790	160.444	125.392	204
2	C_1	alive	2001	436.656	526.477	22.496	27.207	286.588	-58.939	77.528	...	638.721	710.199	217.699	4.711	112.244	150.464	139
3	C_1	alive	2002	396.412	496.747	27.172	30.745	259.954	-12.410	66.322	...	606.337	686.621	164.658	3.573	109.590	203.575	124
4	C_1	alive	2003	432.204	523.302	26.680	47.491	247.245	3.504	104.661	...	651.958	709.292	248.666	20.811	128.656	131.261	131

Figure 2: The first five rows of the dataset after it has been loaded

The column names of that came with the dataset had to be changed to be more insightful. The code below was used to change the column names:

```
# Changing column names
column_name_mapping = { 'X1': 'Current Assets', 'X2': 'Cost of Goods Sold', 'X3': 'Depreciation and Amortization',
'X4': 'EBITDA', 'X5': 'Inventory', 'X6': 'Net Income', 'X7': 'Total Receivables', 'X8': 'Market Value',
'X9': 'Net Sales', 'X10': 'Total Assets', 'X11': 'Total Long-term Debt',
'X12': 'EBIT', 'X13': 'Gross Profit', 'X14': 'Total Current Liabilities', 'X15': 'Retained Earnings',
'X16': 'Total Revenue', 'X17': 'Total Liabilities', 'X18': 'Total Operating Expenses'
}
data.rename(columns=column_name_mapping, inplace=True)
data.head()
```

Figure 3: Code to change the column names

	company_name	status_label	year	Current Assets	Cost of Goods Sold	Depreciation and Amortization	EBITDA	Inventory	Net Income	Total Receivables	...	Net Sales	Total Assets	Total Long-term Debt	EBIT	Gross Profit
0	C_1	alive	1999	511.267	833.107	18.373	89.031	336.018	35.163	128.348	...	1024.333	740.998	180.447	70.658	191.226
1	C_1	alive	2000	485.856	713.811	18.577	64.367	320.590	18.531	115.187	...	874.255	701.854	179.987	45.790	160.444
2	C_1	alive	2001	436.656	526.477	22.496	27.207	286.588	-58.939	77.528	...	638.721	710.199	217.699	4.711	112.244
3	C_1	alive	2002	396.412	496.747	27.172	30.745	259.954	-12.410	66.322	...	606.337	686.621	164.658	3.573	109.590
4	C_1	alive	2003	432.204	523.302	26.680	47.491	247.245	3.504	104.661	...	651.958	709.292	248.666	20.811	128.656

Figure 4: The first five rows of the dataset after changing the column names

Basic information including the size of the dataset, the datatypes of each column, and the amount the null values was gotten with the code below:

```
# Display basic information about the dataset
print(data.info())
```

Figure 5: Code to display basic information about the dataset

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 78682 entries, 0 to 78681
Data columns (total 21 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   company_name                    78682 non-null  object
 1   status_label                    78682 non-null  object
 2   year                            78682 non-null  int64
 3   Current Assets                  78682 non-null  float64
 4   Cost of Goods Sold              78682 non-null  float64
 5   Depreciation and Amortization   78682 non-null  float64
 6   EBITDA                          78682 non-null  float64
 7   Inventory                       78682 non-null  float64
 8   Net Income                      78682 non-null  float64
 9   Total Receivables               78682 non-null  float64
 10  Market Value                    78682 non-null  float64
 11  Net Sales                       78682 non-null  float64
 12  Total Assets                    78682 non-null  float64
 13  Total Long-term Debt            78682 non-null  float64
 14  EBIT                            78682 non-null  float64
 15  Gross Profit                    78682 non-null  float64
 16  Total Current Liabilities       78682 non-null  float64
 17  Retained Earnings               78682 non-null  float64
 18  Total Revenue                   78682 non-null  float64
 19  Total Liabilities               78682 non-null  float64
 20  Total Operating Expenses        78682 non-null  float64
dtypes: float64(18), int64(1), object(2)
memory usage: 12.6+ MB
None
```

There are no null values present in the dataset and all features have the correct datatype.

Figure 6: Basic information about the dataset

Problem to be Addressed

The accurate prediction of company bankruptcy holds pivotal importance for various stakeholders in today's dynamic and competitive business environment. The challenge lies in identifying early warning signs and constructing reliable prediction models (Zelenkov and Volodarskiy, 2021). The problem addressed in this project is to develop artificial intelligence models that can predict company. By doing so, I aim to provide investors, creditors, and managers with actionable insights for making informed decisions.

Artificial Intelligence models

In this project, three supervised machine learning classification models were trained on the bankruptcy dataset namely Random Forest, Multi-layer Perceptron (MLP) and Long Short-Term Memory (LSTM) models.

Summary of the approach

The main aim of this project to predict company bankruptcies using a machine learning approach, leveraging the power of Random Forest, MLP and LSTM algorithms. The workflow is made up of data preprocessing, feature engineering, resampling techniques, model training, and evaluation.

Random Forest

Random Forest is a prominent ensemble machine learning method introduced by Breiman in 2001. It comprises of a collection of decision trees that aggregate their predictions, offering improved accuracy compared to individual decision trees, making it particularly effective for complex datasets (Speiser et al., 2019). Alam et al., (2020)'s research was based on the application of the following machine learning models: support vector machine, decision tree, logistic model tree and random forest, toward predicting company bankruptcy ahead of its occurrence. The random forest outperformed the other models and previous techniques.

Multi-layer Perceptron (MLP)

A Multi-Layer Perceptron (MLP) is a neural network model that emulates the interconnected neurons in the human brain to predict outcomes from complex input variables. It consists of layers, including input, hidden, and output layers, where neurons communicate through activation functions to process input data and produce predictions (Marso and Merouani, 2020). The study by Ahmadpour Kasgari et al., (2012) on bankruptcy prediction using MLP demonstrates that the predictive MLP models outperformed alternative algorithms, yielding superior outcomes.

Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) is an advanced Recurrent Neural Network (RNN) architecture that overcomes the limitations of standard RNNs by effectively capturing long-

range dependencies in sequential data. It achieves this through specialized memory cells and gating mechanisms, enabling dynamic modeling of sequences with varying time spans (Staudemeyer and Morris, 2019).

Kim, Cho and Ryu (2021) compared the performance of LSTM models and found out that it outperformed in other classification models like Support Vector Machines and Logistic Regression in terms of accuracy, precision, recall

Data analysis and visualization

The first step is to import the necessary libraries needed for the project. They are shown in the diagram below:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from imblearn.combine import SMOTEENN
from imblearn.under_sampling import EditedNearestNeighbours
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
from sklearn.neural_network import MLPClassifier
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
```

Figure 7: Import libraries needed

After checking for null values(Figure 6), we need to view the class distribution of the dataset, i.e. check the amount of rows for the two values of the target column.

```
# To count the number of alive and failed in the status_label

class_percentage = data['status_label'].value_counts(normalize = True)
print(class_percentage)
```

```
status_label
alive     0.933657
failed    0.066343
Name: proportion, dtype: float64
```

Figure 8: Percentage of class distribution

```
# plotting the distribution of the status label
plt.figure(figsize=(8, 6))
sns.countplot(x='status_label', data=data)
plt.title('Class Distribution')
plt.xlabel('Status Label')
plt.ylabel('Count')
plt.show()
```

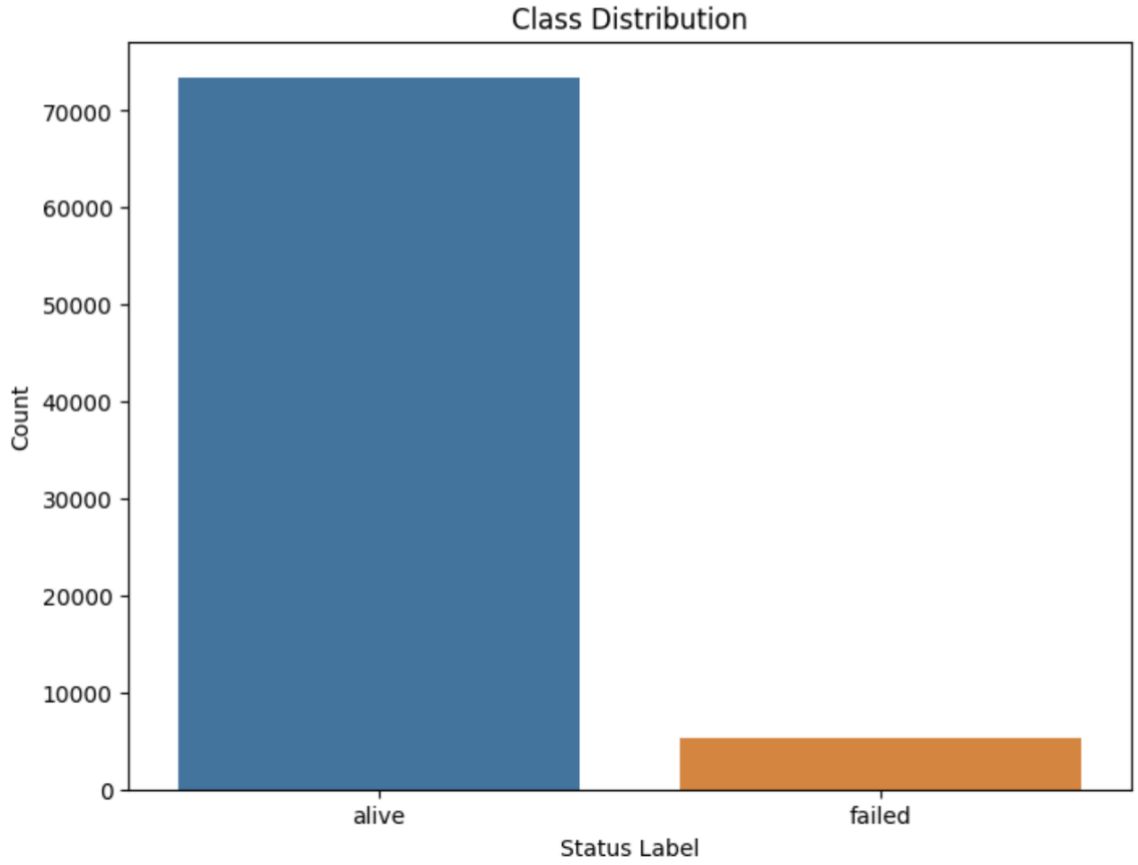

Figure 8: Percentage of class distribution

From the figure above, we can see that the data is heavily imbalanced, with the minority class (failed), accounting for just 6% of the population.

Exploratory Data Analysis

```
data.describe().round()
```

	year	Current Assets	Cost of Goods Sold	Depreciation and Amortization	EBITDA	Inventory	Net Income	Total Receivables	Market Value	Net Sales	Total Assets	Total Long-term Debt	EBIT	Gross Profit	Cur Liabil
count	78682.0	78682.0	78682.0	78682.0	78682.0	78682.0	78682.0	78682.0	78682.0	78682.0	78682.0	78682.0	78682.0	78682.0	786
mean	2008.0	880.0	1595.0	121.0	377.0	202.0	129.0	287.0	3414.0	2364.0	2867.0	722.0	256.0	769.0	6
std	6.0	3929.0	8930.0	652.0	2012.0	1061.0	1266.0	1336.0	18414.0	11950.0	12918.0	3242.0	1495.0	3775.0	29
min	1999.0	-8.0	-367.0	0.0	-21913.0	0.0	-98696.0	-0.0	0.0	-1965.0	0.0	-0.0	-25913.0	-21536.0	
25%	2002.0	19.0	17.0	1.0	-1.0	0.0	-7.0	3.0	35.0	28.0	37.0	0.0	-3.0	9.0	
50%	2007.0	100.0	104.0	8.0	15.0	7.0	2.0	23.0	228.0	187.0	213.0	8.0	7.0	64.0	
75%	2012.0	432.0	635.0	48.0	140.0	75.0	40.0	132.0	1245.0	1046.0	1171.0	249.0	88.0	344.0	2
max	2018.0	169662.0	374623.0	28430.0	81730.0	62567.0	104821.0	65812.0	1073391.0	511729.0	531864.0	166250.0	71230.0	137106.0	1168

Figure 9: Numerical data related to the dataset

From the figure above, one can observe that the mean values of all continuous columns in dataset is higher than the median value, this suggests that the continuous columns are positively skewed.

The mean values of all continuous columns in data set is higher than the median value, this suggests that the continuous columns are positively skewed.

I observed that most companies have lower current assets, with a few companies having significantly higher values. Companies with higher current assets might have better liquidity.

The positive skewness implies that a few companies have significantly higher total assets compared to others and that companies with higher long-term debt might have more financing obligations.

Positive skewness also implies that some companies have high EBIT and EBITDA values, high gross profit values, high current liabilities, high retained earnings values, high total revenue values, high total liabilities, high total operating expenses, high costs of goods sold, high positive net sales, high total receivables, high depreciation and amortization values, large inventory values, very high market values compared to the majority and high positive net income values.

Since the continuous data is positively skewed, we need to transform the data before training the model. I used MinMaxScaler and Cube root for this.

```
def drawsub_plots(data_to_draw):
    num_features = data_to_draw.columns
    num_plots = len(num_features)
    num_rows = 5

    fig, axes = plt.subplots(num_rows, 4, figsize=(12, 3*num_rows))
    fig.suptitle('Histograms of Numerical Features by Class', y=0.88)

    plt.subplots_adjust(top=0.9 - 0.02*num_rows, hspace=0.5)
    for i, feature in enumerate(num_features):
        row = i // 4
        col = i % 4
        ax = axes[row, col]


        sns.histplot(data=data_to_draw, x=feature, hue=data["status_label"], element='step', bins=100, ax=ax)
        ax.set_title(f'{feature}')
        ax.set_xlabel(feature)
        ax.set_ylabel('Count')

        if col > 0:
            ax.set_yticks([])
            ax.set_ylabel('')

    fig.delaxes(axes[num_rows-1, 2])
    fig.delaxes(axes[num_rows-1, 3])

    plt.show()
```

Figure 10: A function made for drawing subplots

```
# Use minmax sclaer to scale the features
min_max_scaler = MinMaxScaler()
min_max_scaled_data = pd.DataFrame(min_max_scaler.fit_transform(data.drop(columns=["status_label", "company_name", "yea
#Draw histograms after transformation
drawsub_plots(min_max_scaled_data)
```

Figure 11: Code for MinMaxScaling

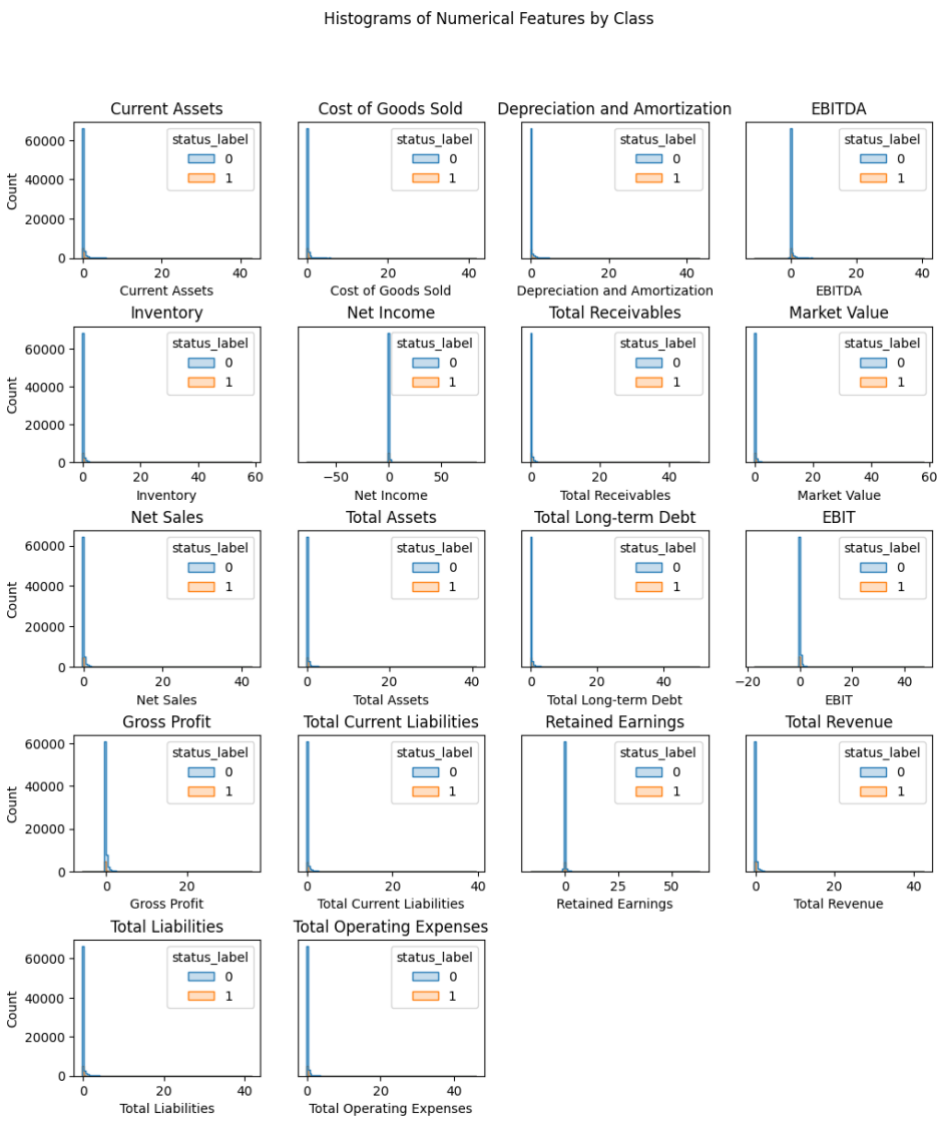

Figure 12: Plots of each feature after MinMaxScaling

```
# Cube root Transformation
log_cbrt = np.cbrt(data.drop(columns=["status_label", "company_name", "year"]))

#Draw histograms after transformation
drawsub_plots(log_cbrt)
```

Figure 13: Code for cube root transformation

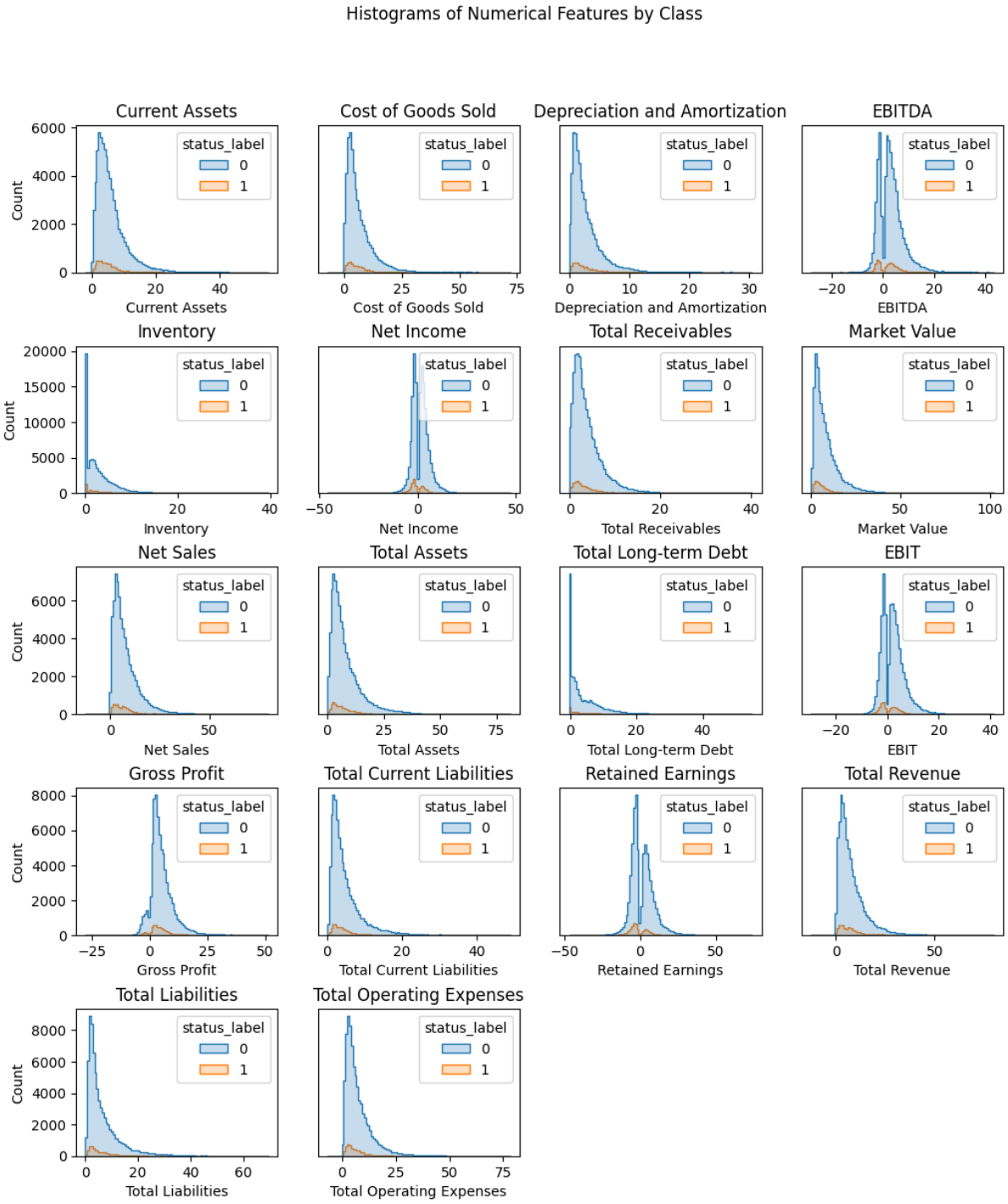

Figure 14: Plots of each feature after Cube root Transformation

I observed that the positive skewness was still present in Minmax Scaled Data. However, the Cube root Transformed Data did not have as much positive skewness. Hence, I used Cube root transform for my dataset.

Feature Selection

Before performing feature selection, I encoded the non-float/integer values. I used the Label encoder library provided by sklearn. The columns affected were the status_label and the company name.

```
# Change the status_label to number
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
data["status_label"] = le.fit_transform(data.status_label.values)
data.head()
```

	company_name	status_label	year	Current Assets	Cost of Goods Sold	Depreciation and Amortization	EBITDA	Inventory	Net Income	Total Receivables	...	Net Sales	Total Assets	Total Long-term Debt	EBIT	Gross Profit
0	C_1	0	1999	511.267	833.107	18.373	89.031	336.018	35.163	128.348	...	1024.333	740.998	180.447	70.658	191.226
1	C_1	0	2000	485.856	713.811	18.577	64.367	320.590	18.531	115.187	...	874.255	701.854	179.987	45.790	160.444
2	C_1	0	2001	436.656	526.477	22.496	27.207	286.588	-58.939	77.528	...	638.721	710.199	217.699	4.711	112.244
3	C_1	0	2002	396.412	496.747	27.172	30.745	259.954	-12.410	66.322	...	606.337	686.621	164.658	3.573	109.590
4	C_1	0	2003	432.204	523.302	26.680	47.491	247.245	3.504	104.661	...	651.958	709.292	248.666	20.811	128.656

Figure 15: Code for encoding the target variable

```
def custom_encoder(columns):
    for column in columns:
        le = LabelEncoder()
        data[column] = le.fit_transform(data[column].values)

custom_encoder(["company_name"])
```

Figure 16: Code for encoding the company name

I made use of the correlation heatmap to check correlation amongst the features present in the dataset.

```
# Correlation heatmap
# Checking the correlation amongst the features
plt.figure(figsize=(16,10))
features = data.drop('status_label', axis=1)
sns.heatmap(features.corr(),annot=True,cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```

Figure 17: Code for plotting Correlation heatmap

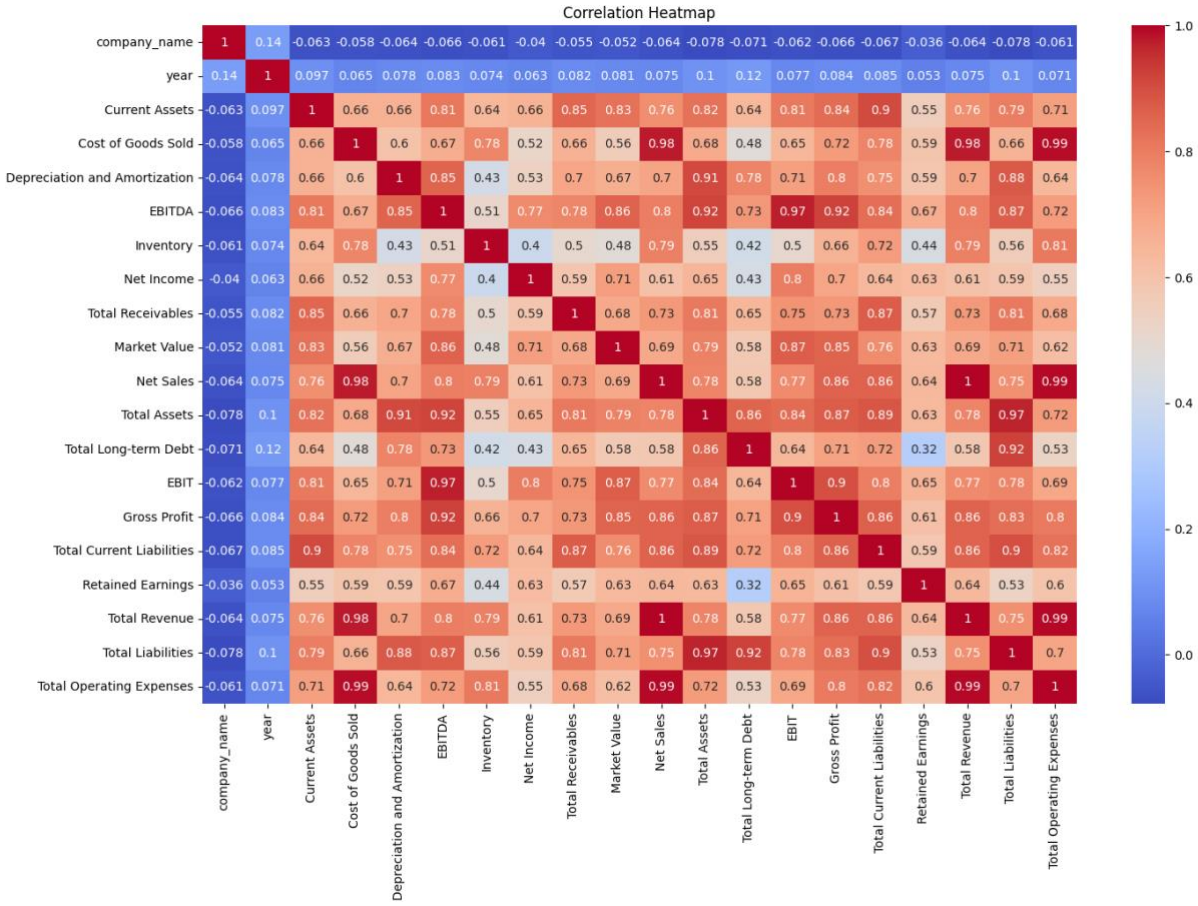

Figure 18: Correlation heatmap with all the features

I observed that the company name and year columns have very weak correlation with the rest of the features. Hence, I dropped them.

```
# company_name and year have weak correlation and can be dropped
data.drop(columns=["company_name", "year"], inplace=True)
features.drop(columns=["company_name", "year"], inplace=True)
data.head()
```

	status_label	Current Assets	Cost of Goods Sold	Depreciation and Amortization	EBITDA	Inventory	Net Income	Total Receivables	Market Value	Net Sales	Total Assets	Total Long-term Debt	EBIT	Gross Profit	Total Current Liabilities	Retai Earni
0	0	511.267	833.107	18.373	89.031	336.018	35.163	128.348	372.7519	1024.333	740.998	180.447	70.658	191.226	163.816	201
1	0	485.856	713.811	18.577	64.367	320.590	18.531	115.187	377.1180	874.255	701.854	179.987	45.790	160.444	125.392	204
2	0	436.656	526.477	22.496	27.207	286.588	-58.939	77.528	364.5928	638.721	710.199	217.699	4.711	112.244	150.464	139
3	0	396.412	496.747	27.172	30.745	259.954	-12.410	66.322	143.3295	606.337	686.621	164.658	3.573	109.590	203.575	124
4	0	432.204	523.302	26.680	47.491	247.245	3.504	104.661	308.9071	651.958	709.292	248.666	20.811	128.656	131.261	131

Figure 19: Code for dropping weak correlation features

Data Preparation

Before moving on to building the machine learning models, we need to perform some data preparation tasks. The first is to sperate the features from the target column.

```
# separate the features and the target column

from sklearn.model_selection import train_test_split
X = data.drop(columns=['status_label'])
Y = data['status_label']

X.head()
```

	Current Assets	Cost of Goods Sold	Depreciation and Amortization	EBITDA	Inventory	Net Income	Total Receivables	Market Value	Net Sales	Total Assets	Total Long-term Debt	EBIT	Gross Profit	Total Current Liabilities	Retained Earnings	Tota Revenu
0	511.267	833.107	18.373	89.031	336.018	35.163	128.348	372.7519	1024.333	740.998	180.447	70.658	191.226	163.816	201.026	1024.33
1	485.856	713.811	18.577	64.367	320.590	18.531	115.187	377.1180	874.255	701.854	179.987	45.790	160.444	125.392	204.065	874.25
2	436.656	526.477	22.496	27.207	286.588	-58.939	77.528	364.5928	638.721	710.199	217.699	4.711	112.244	150.464	139.603	638.72
3	396.412	496.747	27.172	30.745	259.954	-12.410	66.322	143.3295	606.337	686.621	164.658	3.573	109.590	203.575	124.106	606.33
4	432.204	523.302	26.680	47.491	247.245	3.504	104.661	308.9071	651.958	709.292	248.666	20.811	128.656	131.261	131.884	651.95

Figure 20: Code for separating features from the target column

From the earlier analysis, cube root was found to be ideal for data transformation for this dataset. The next step is performing cube root transformation on the features.

```
# Tranform features using Cube root transform
X_cbrt = np.cbrt(X)
X_cbrt.head()
```

	Current Assets	Cost of Goods Sold	Depreciation and Amortization	EBITDA	Inventory	Net Income	Total Receivables	Market Value	Net Sales	Total Assets	Total Long-term Debt	EBIT	Gross Profit	Total Current Liabilities	Retain Earnin
0	7.996180	9.409508	2.638720	4.465263	6.952177	3.276136	5.044247	7.196809	10.080461	9.049106	5.650886	4.134158	5.761236	5.471656	5.8580
1	7.861448	8.937055	2.648451	4.007631	6.844105	2.646263	4.865579	7.224799	9.561941	8.886872	5.646080	3.577587	5.433852	5.005221	5.8873
2	7.586588	8.074701	2.822941	3.007647	6.593044	-3.891654	4.264023	7.143911	8.611994	8.921955	6.015690	1.676374	4.823782	5.318766	5.1875
3	7.345966	7.919755	3.006357	3.132743	6.382128	-2.315211	4.047801	5.233335	8.463916	8.822108	5.481014	1.528778	4.785459	5.882674	4.9880
4	7.560716	8.058437	2.988101	3.621349	6.276379	1.518873	4.712611	6.759937	8.671080	8.918155	6.288380	2.750622	5.048279	5.082124	5.0901

Figure 21: Code for transforming features

Model training, evaluation and testing

Now that the features and target variable have been separated, the next step is to split the data into training and testing sets. The training set is fed into the model, this enables the model to be able to make predictions. The testing set is used to validate how accurate the predictions are. The split ratio used for this dataset is 70-30, where 70% of the data is for training and the remaining 30% for testing.

```
# Split the scaled data into training and testing sets
X_train_cbrt, X_test_cbrt, Y_train_cbrt, Y_test_cbrt = train_test_split(X_cbrt, Y,
                                                                        test_size=0.3,
                                                                        random_state=42)
```

Figure 22: Code for splitting dataset

Adressing Class Imbalance

The bankruptcy dataset used in this study is very imbalanced. With the minority class (failed), accounting for just 6% of the population. The training set contained 51,471 cases of "alive" companies and 3,606 cases of "failed companies".

```
#Class distrubution before resampling
unique, counts = np.unique(Y_train_cbrt, return_counts=True)
print(dict(zip(unique, counts)))
```

```
{0: 51471, 1: 3606}
```

Figure 23: Class distribution in the training set

A combination of resampling techniques was used to address the issue of class imbalance within the dataset.

The two techniques used are Synthetic Minority Over-sampling Technique (SMOTE) and Edited Nearest Neighbors (ENN). These techniques were chosen to ensure that both the minority class ("failed") and the majority class ("alive" companies) are well represented in the training set.

```
# #Application and of SMOTE and EditedNearestNeighbours to the training set
sm = SMOTEENN(
    sampling_strategy='auto',
    smote=SMOTE(sampling_strategy=0.3, k_neighbors=5),
    enn=EditedNearestNeighbours(sampling_strategy='auto', n_neighbors=3)
)

X_train_resampled, Y_train_resampled = sm.fit_resample(X_test_cbrt,Y_test_cbrt)
```

Figure 24: Code for resampling the training set

The resampling process resulted in a much more balanced training set.

```
#Class distrubution after resampling
unique, counts = np.unique(Y_train_resampled, return_counts=True)
print(dict(zip(unique, counts)))
```

```
{0: 17494, 1: 6597}
```

Figure 25: Class distribution in the training set after resampling

```
# Plot training set class distrubiton after resampling

plt.figure(figsize=(8, 6))
sns.countplot(x='status_label', data=pd.DataFrame({'status_label': Y_train_resampled}))
plt.title('Class Distribution')
plt.xlabel('Status Label')
plt.ylabel('Count')
plt.show()
```

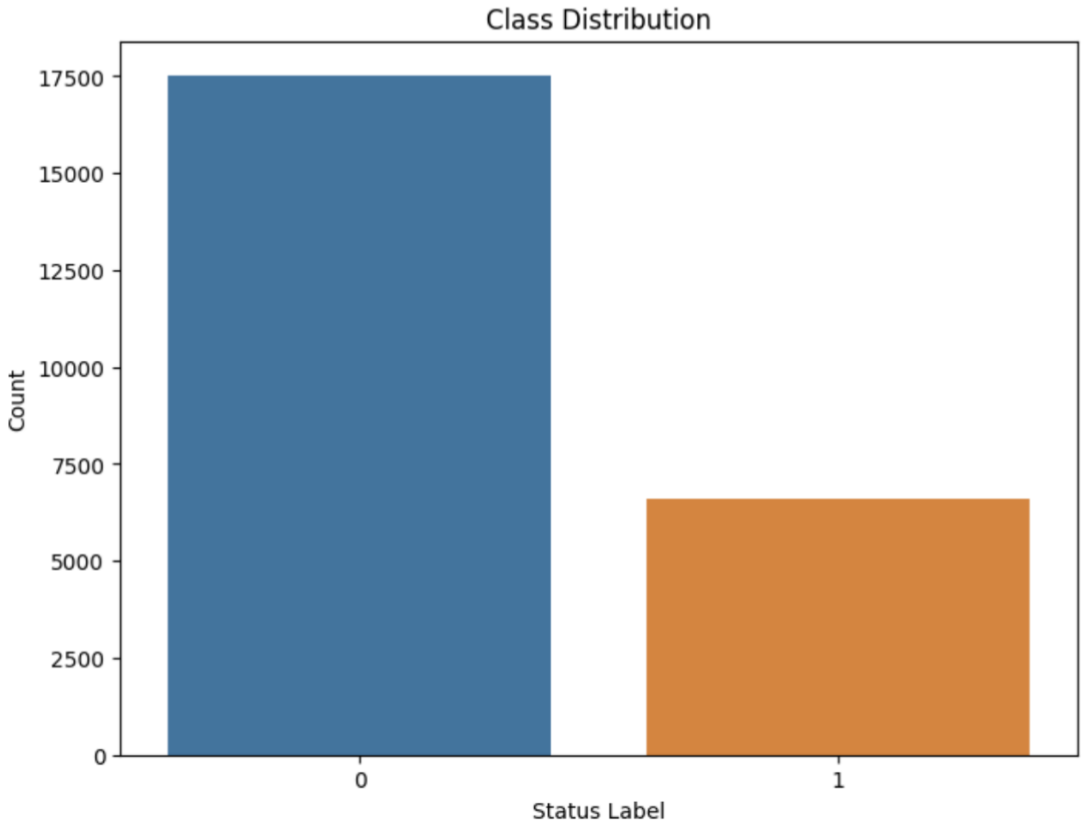

Figure 26: Plot of Class distribution in the training set after resampling

Random Forest

The first machine learning model the bankruptcy dataset was trained on is the Random Forest model, known for its ensemble of decision trees. Before training the model, I performed hyper parameter tuning to get the optimal configuration.

I used the RandomizedSearchCV function provided by scikit-learn to explore a range of hyperparameters like n_estimators, max_depth, min_samples_split and classification socring metrics. The randomized search yielded optimal hyperparameters which was used to build the final Random Forest model.

```
#specify the hyperparameters and their values
param_grid = {
    'n_estimators': [100, 200, 500],
    'max_depth': [50, 100, 200],
    'min_samples_split': [2, 3, 5]
}
scoring = ['f1', 'recall', 'precision', 'roc_auc']

forestclass = RandomForestClassifier(random_state=42)
```

```
from sklearn.model_selection import RandomizedSearchCV

# we'll use 10-fold cross-validation, so the model does not over fit
rand_grid_search_RF = RandomizedSearchCV(forestclass, param_grid, cv=10, n_iter=6,
                                         scoring=scoring,  random_state=5,
                                         return_train_score=True, refit='precision')

rand_grid_search_RF.fit( X_train_resampled ,Y_train_resampled)
```

```
▸ RandomizedSearchCV
▾ estimator: RandomForestClassifier
RandomForestClassifier(random_state=42)
▾ RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
best_forest = rand_grid_search_RF.best_estimator_
best_forest
```

```
▾ RandomForestClassifier
RandomForestClassifier(max_depth=200, n_estimators=200, random_state=42)
```

Figure 27: Hyper parameter tuning for Random Forest

The hyperparameters were used to initialize and train the Random Forest model.

```
# Initialize and train the Random Forest Classifier
random_forest_model = RandomForestClassifier(n_estimators=200, max_depth=200, min_samples_split=5)
random_forest_model.fit(X_train_resampled, Y_train_resampled)

Y_pred_rf = random_forest_model.predict(X_test_cbrt)
```

Figure 28: Training of the Random Forest model

```
# Plot the confusion matrix and other scoring parameters for the random forest model
sns.heatmap(confusion_matrix(Y_test_cbrt, Y_pred_rf),annot=True,cmap='Blues', fmt="g")
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')

print("Accuracy:", accuracy_score(Y_test_cbrt, Y_pred_rf))
print("Precision:", precision_score(Y_test_cbrt, Y_pred_rf))
print("Recall:", recall_score(Y_test_cbrt, Y_pred_rf))
print("F1 Score:", f1_score(Y_test_cbrt, Y_pred_rf))
```

```
Accuracy: 0.9693708959966109
Precision: 0.6914482165878814
Recall: 0.9969021065675341
F1 Score: 0.8165440243592995
```

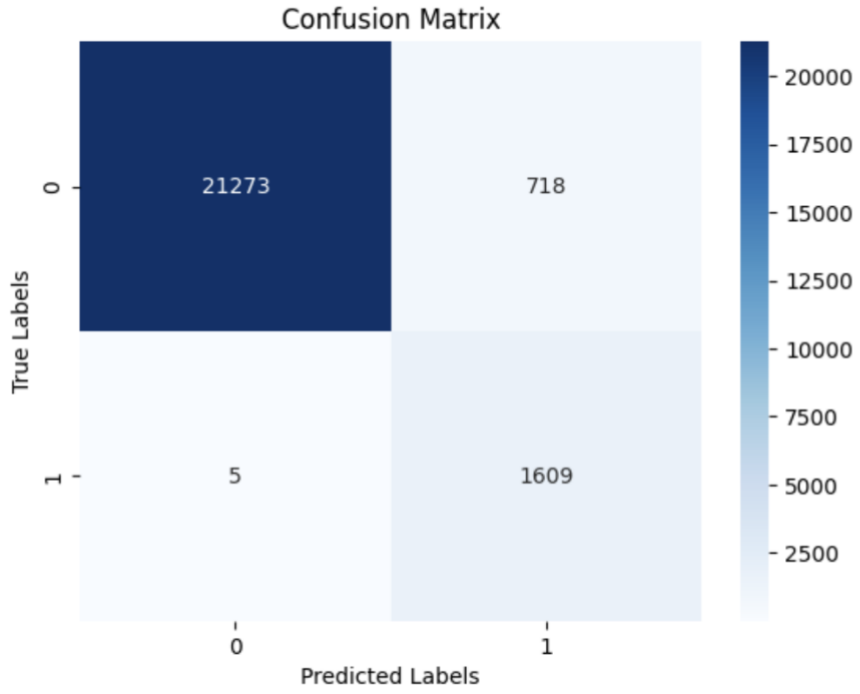

Figure 29: Classification scoring for Random Forest model

Multi-layer Perceptron (MLP) Model

The second machine learning model used is a type of artificial Neural Network called the Multilayer Perceptron (MLP) model. The hyperparameter tuning for the MLP model was done manually and it involved iteratively adjusting hyperparameters, including the number of hidden layers, units per layer, and activation functions. This resulted in an optimized MLP model for the bankruptcy dataset.

```
# Initialize and train the MLP Classifier

mlp = MLPClassifier(hidden_layer_sizes=(100,100,100), activation="relu", solver="adam", max_iter=600)

mlp.fit(X_train_resampled, Y_train_resampled)

Y_pred_mlp = mlp.predict(X_test_cbrt)
```

Figure 30: Training of the MLP model

```
: # Plot the confusion matrix and other scoring parameters for the MLP model
sns.heatmap(confusion_matrix(Y_test_cbrt, Y_pred_mlp),annot=True,cmap='Blues', fmt="g")
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')

print("Accuracy:", accuracy_score(Y_test_cbrt, Y_pred_mlp))
print("Precision:", precision_score(Y_test_cbrt, Y_pred_mlp))
print("Recall:", recall_score(Y_test_cbrt, Y_pred_mlp))
print("F1 Score:", f1_score(Y_test_cbrt, Y_pred_mlp))
```

```
Accuracy: 0.949671679728871
Precision: 0.5777372262773722
Recall: 0.9807930607187113
F1 Score: 0.7271474506201194
```

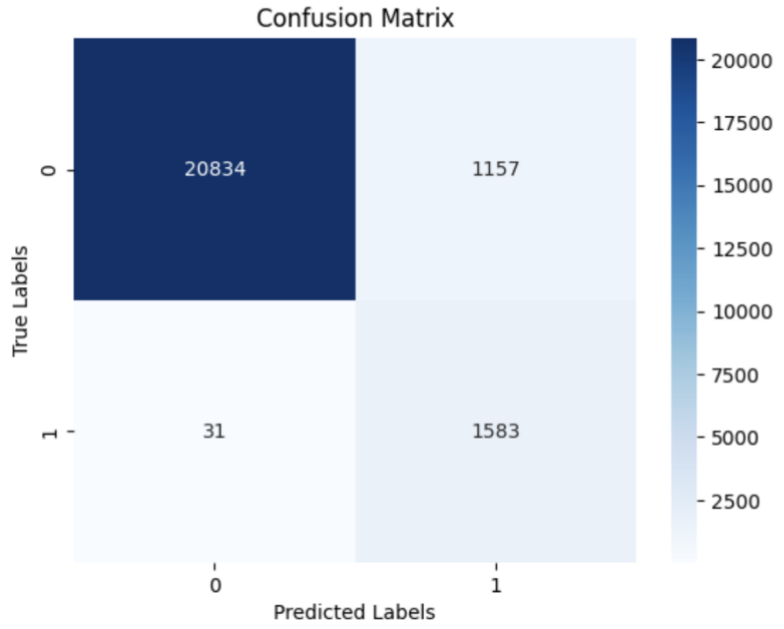

Figure 31: Classification scoring for the MLP model

Long Short-Term Memory (LSTM)

The third and final machine learning model used in this project was the Long Short-Term Memory (LSTM) model. Although LSTM models are mostly used for sequence-based data, I adapted the architecture to the format of my dataset. This involved converting my testing and training data frames to arrays.

```
# Convert DataFrame to NumPy array
X_train_array = X_train_resampled.values
X_test_array = X_test_cbrt.values

y_train_array = Y_train_resampled.values
y_test_array = Y_test_cbrt.values

# Reshape the data to include a time step dimension of 1
X_train_lstm = X_train_array.reshape((X_train_array.shape[0], 1, X_train_array.shape[1]))
X_test_lstm = X_test_array.reshape((X_test_array.shape[0], 1, X_test_array.shape[1]))
```

Figure 32: Reshaping data for LSTM

The hyperparameter tuning for the LSTM model was done manually and it involved iteratively adjusting hyperparameters like the number of epochs. This resulted in an optimized LSTM model for the bankruptcy dataset.

```
# Build LSTM model
model = Sequential()
model.add(LSTM(units=50, input_shape=(X_train_lstm.shape[1], X_train_lstm.shape[2])))
model.add(Dense(units=1, activation='sigmoid'))

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(X_train_lstm, y_train_array, epochs=120, batch_size=32)

# Make predictions on the test data
Y_pred_lstm = model.predict(X_test_lstm)
```

```
Epoch 1/120
753/753 [==============================] - 6s 3ms/step - loss: 0.5135 - accuracy: 0.7573
Epoch 2/120
753/753 [==============================] - 2s 3ms/step - loss: 0.4903 - accuracy: 0.7690
Epoch 3/120
753/753 [==============================] - 2s 2ms/step - loss: 0.4802 - accuracy: 0.7737
Epoch 4/120
753/753 [==============================] - 2s 2ms/step - loss: 0.4733 - accuracy: 0.7776
Epoch 5/120
753/753 [==============================] - 2s 3ms/step - loss: 0.4676 - accuracy: 0.7805
Epoch 6/120
753/753 [==============================] - 2s 3ms/step - loss: 0.4619 - accuracy: 0.7827
Epoch 7/120
753/753 [==============================] - 2s 2ms/step - loss: 0.4571 - accuracy: 0.7833
Epoch 8/120
753/753 [==============================] - 1s 2ms/step - loss: 0.4531 - accuracy: 0.7873
Epoch 9/120
753/753 [==============================] - 2s 2ms/step - loss: 0.4487 - accuracy: 0.7891
Epoch 10/120
```

Figure 33: LSTM model training

```
Y_pred_lstm = (Y_pred_lstm > 0.5).astype(int)  # Convert probabilities to binary classes

# Plot the confusion matrix and other scoring parameters for the MLP model
sns.heatmap(confusion_matrix(Y_test_cbrt, predicted_classes),annot=True,cmap='Blues', fmt="g")
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')

print("Accuracy:", accuracy_score(Y_test_cbrt, Y_pred_lstm))
print("Precision:", precision_score(Y_test_cbrt, Y_pred_lstm))
print("Recall:", recall_score(Y_test_cbrt, Y_pred_lstm))
print("F1 Score:", f1_score(Y_test_cbrt, Y_pred_lstm))
```

```
Accuracy: 0.8687142554543529
Precision: 0.3022636484687084
Recall: 0.7032218091697645
F1 Score: 0.42279754144160925
```

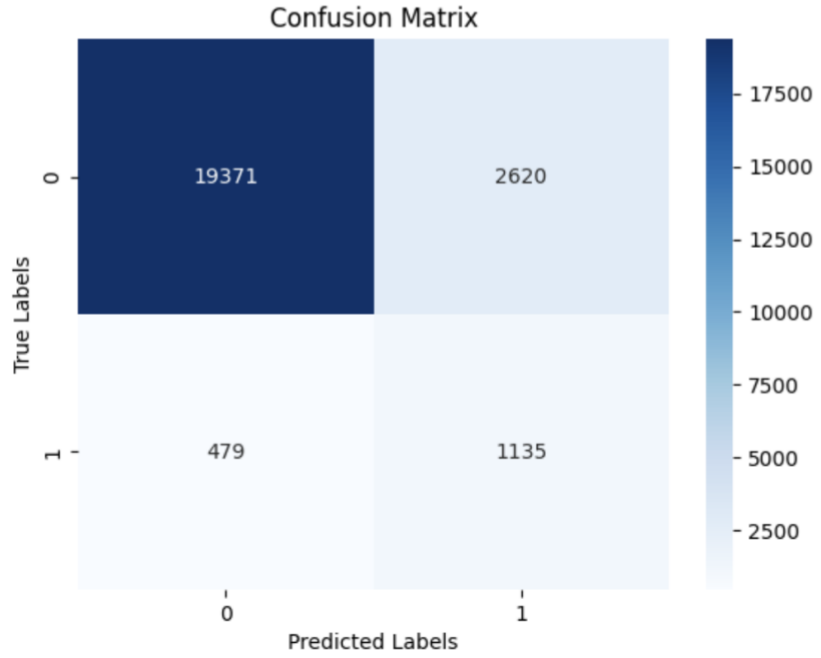

Figure 34: Classification scoring for the LSTM model

Results and discussion

The performance of the Random Forest, MLP, LSTM models on the bankruptcy are evaluated using classification scoring metrics like Accuracy, Precision, Recall and F1 Score. The results are discussed below:

Random Forest Results

The Random Forest model had an accuracy of 96.9%. The model correctly classified bankrupt companies 69.1% of the time. The model scored 99.7% in being able to identify actual bankruptcies. The F1 Score, which balances precision and recall, 81.7%.

MLP Results

The Multilayer Perceptron (MLP) model had an accuracy of 95%. The model correctly classified bankrupt companies 57.8% of the time. The model scored 98.1% in being able to identify actual bankruptcies. The F1 Score, which balances precision and recall, 72.7%.

LSTM Results

The Long Short-Term Memory (LSTM) model, while adapted for non-sequential data, demonstrated reasonable performance. With an accuracy of 86.9% and a precision of 30.2%. The model's recall was 70.3% while its F1 Score was 42.3%.

```
def create_score_barplot(metric, metric_name):
    models = ['Random Forest', 'MLP', 'LSTM']
    # Create a bar plot
    plt.figure(figsize=(10, 6))
    plt.bar(models, metric, color=['blue', 'green', 'orange'])
    plt.xlabel('Machine Learning Models')
    plt.ylabel(f'{metric_name} (%)')
    plt.title(f'Comparison of {metric_name} Percentages')
    plt.ylim(0, 100)
    plt.tight_layout()

    plt.show()
```

Figure 35: Function to create bar plots for classification metrics

```
# Corresponding accuracy percentages
accuracies = [accuracy_score(Y_test_cbrt, Y_pred_rf)*100, accuracy_score(Y_test_cbrt, Y_pred_mlp)*100, accuracy_score(Y
precisions = [precision_score(Y_test_cbrt, Y_pred_rf)*100, precision_score(Y_test_cbrt, Y_pred_mlp)*100, precision_scor
recalls = [recall_score(Y_test_cbrt, Y_pred_rf)*100, recall_score(Y_test_cbrt, Y_pred_mlp)*100, recall_score(Y_test_cbr
f1_scores = [f1_score(Y_test_cbrt, Y_pred_rf)*100, f1_score(Y_test_cbrt, Y_pred_mlp)*100, f1_score(Y_test_cbrt, Y_pred_

create_score_barplot(accuracies, "Accuracy")
create_score_barplot(precisions, "Precision")
create_score_barplot(recalls, "Recall")
create_score_barplot(f1_scores, "F1 Score")
```

Figure 36: Code to display the comparison of metrics

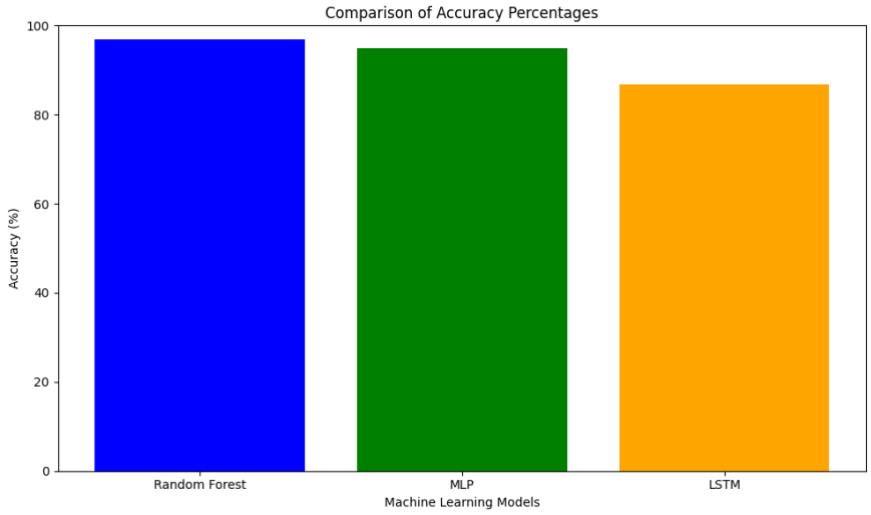

Figure 37: Plot of the comparison of Accuracy Percentages

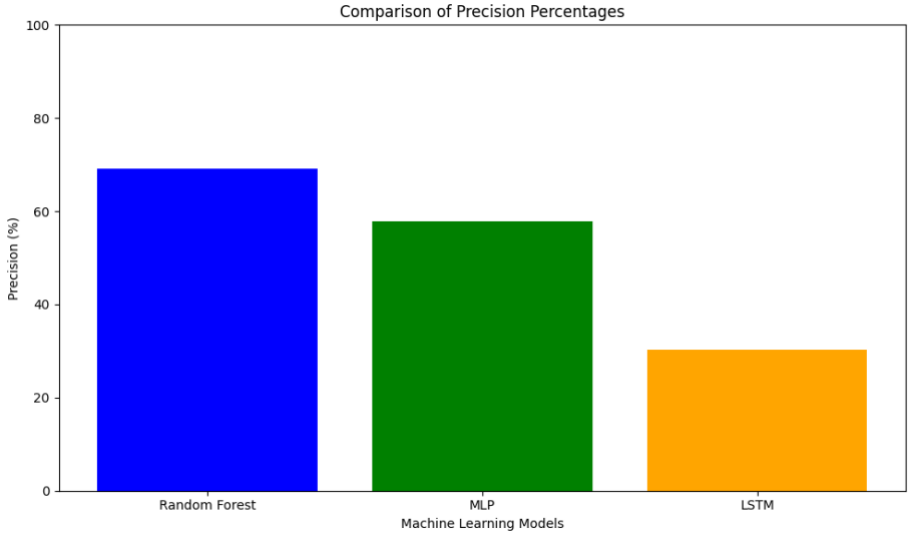

Figure 38: Plot of the comparison of Precision Percentages

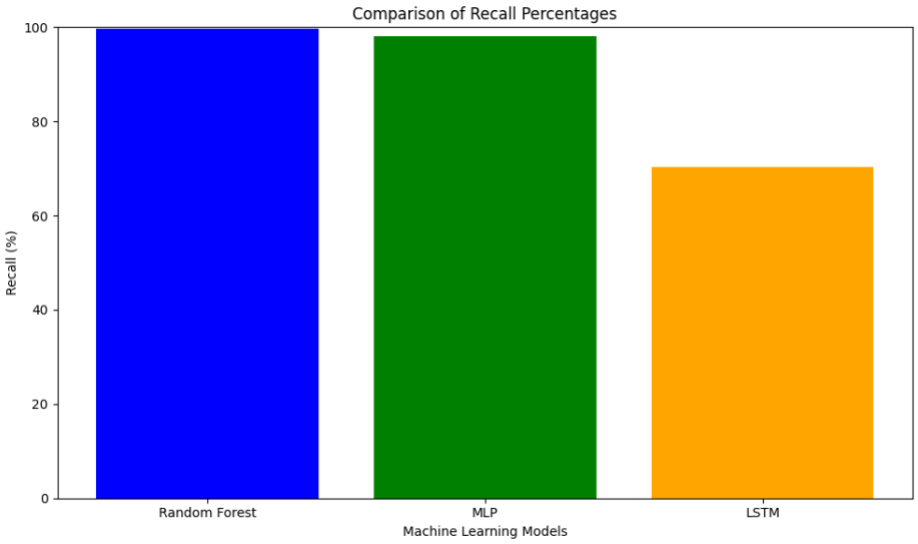

Figure 39: Plot of the comparison of Recall Percentages

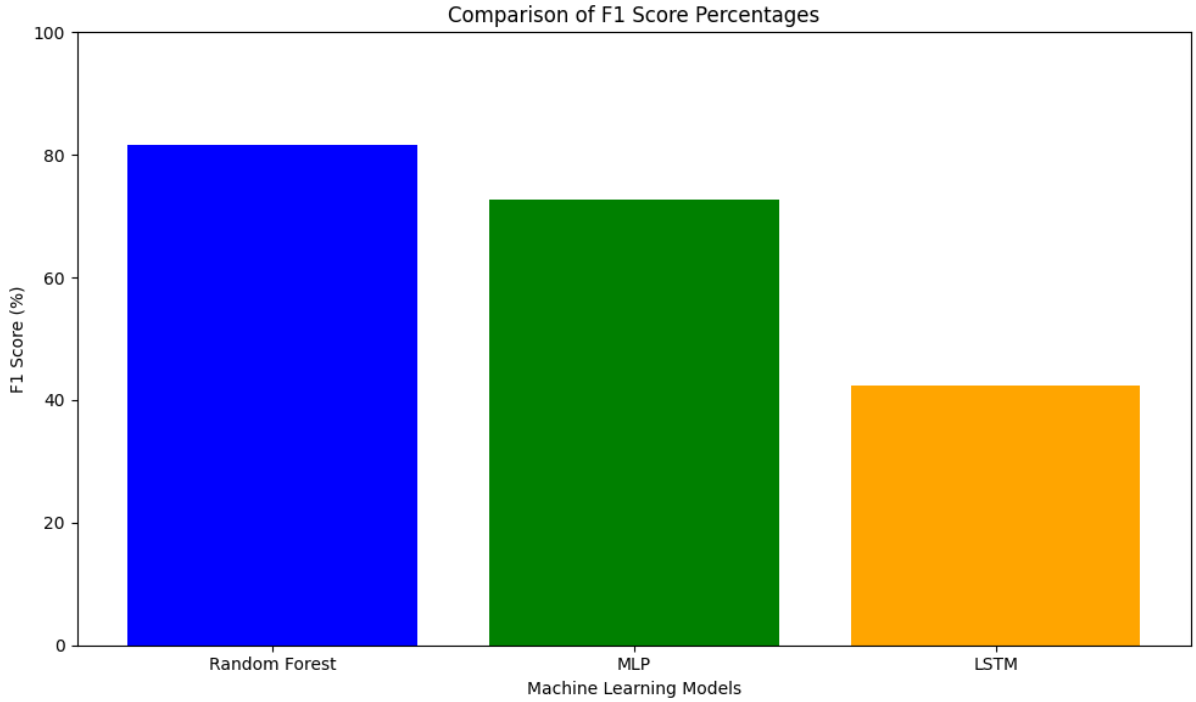

Figure 40: Plot of the comparison of F1 Score Percentages

Results Comparison

The Random Forest model excelled in overall accuracy, making it an optimal choice for bankruptcy choice for identifying both bankruptcy and non-bankruptcy instances. The MLP model demonstrated balanced performance, though the possibility of false positive predictions should be considered. The LSTM model, despite its adaptation for non-sequential data, contributed a novel perspective to bankruptcy prediction, but its performance was moderate due to the unconventional nature of the data.

Models	Accuracy	Precision	Recall	F1 Score
Random Forest	96.9%	69.1%	99.7%	81.7%
MLP	95%	57.8%	98.1%	72.7%
LSTM	86.9%	30.2%	0.3%	42.3%

Table 1: Classification scoring for the LSTM model

Overall, the Random Forest model's high precision and recall make it well-suited for accurate bankruptcy classification.

Analysis of the AI project life cycle compliance with the AI ethics

In recent times, the field artificial intelligence has experienced rapid growth with newer technologies being developed constantly (Hagendorff, 2020). AI ethics is a direct response to this growth and has become a huge focus within the field of digital ethics (Kazim and Koshiyama, 2021). Many ethical guidelines have been made in recent years, serving as guides for technology developers (Hagendorff, 2020). AI ethics was taken into consideration at every step of the development of this project.

Conception and Problem Definition

I took into consideration ethical principles such as fairness, transparency, and accountability during the conception and problem definition phase of this project. I identified potential societal impacts of bankruptcy prediction and ensured to assess the ethical implications of using AI for such a task.

Data Collection and Privacy

The data collection phase was given great attention as data privacy is a great concern nowadays. I got the dataset from Kaggle and ensured compliance with their data usage and privacy regulations. I ensured that I used a dataset where sensitive information like company names were shielded thus upholding privacy rights.

Model Development

Avoid discriminatory outcomes was a key consideration of mine when I started the model building phase. Special attention was given to addressing biases in training data to ensure nondiscriminatory outcomes. Resampling techniques were used to address class imbalance making sure each class is well represented in the dataset.

Conclusion, Recommendations and Future work

Conclusion

In this project, I carried out bankruptcy prediction using the Random Forest, Multilayer Perceptron (MLP), and Long Short-Term Memory (LSTM) models.

The Random Forest model had the best accuracy, precision, and recall of all the Machine learning models used in this project. It effectively captured the relationships within the dataset and it a dependable choice for bankruptcy prediction.

The MLP model demonstrated competitive performance while the LSTM model, although adapted for the bankruptcy dataset had moderate performance.

Recommendations

The results of this project can be useful for and researchers seeking to use machine learning models bankruptcy prediction. The Random Forest Model outperformed the other models used in this study and is an excellent choice for bankruptcy prediction.

Future works

Although the Random Forest model was the optimal model for bankruptcy prediction in this topic, it was too slow when learning from a large dataset. This poses serious concerns for its use in real-word situations where faster predictions are required. Leveraging boosting techniques could improve the training time and make it more efficient.

References

1. Narvekar, A. and Guha, D. (2021). Bankruptcy prediction using machine learning and an application to the case of the COVID-19 recession. Data Science in Finance and Economics, 1(2), pp.180–195. doi:https://doi.org/10.3934/dsfe.2021010.

2. Garcia, J. (2022). Bankruptcy prediction using synthetic sampling. Machine Learning with Applications, p.100343. doi:https://doi.org/10.1016/j.mlwa.2022.100343.

3. Shi, Y. and Li, X. (2019). A bibliometric study on intelligent techniques of bankruptcy prediction for corporate firms. Heliyon, 5(12), p.e02997. doi:https://doi.org/10.1016/j.heliyon.2019.e02997

4. Zelenkov, Y. and Volodarskiy, N. (2021). Bankruptcy prediction on the base of the unbalanced data using multi-objective selection of classifiers. Expert Systems with Applications, 185, p.115559. doi:https://doi.org/10.1016/j.eswa.2021.115559.

5. Wang, H. and Liu, X. (2021). Undersampling bankruptcy prediction: Taiwan bankruptcy data. PLOS ONE, 16(7), p.e0254030. doi:https://doi.org/10.1371/journal.pone.0254030.

6. Lombardo, G., Pellegrino, M., Adosoglou, G., Cagnoni, S., Pardalos, P.M. and Poggi, A. (2022). Machine Learning for Bankruptcy Prediction in the American Stock Market: Dataset and Benchmarks. Future Internet, 14(8), p.244. doi:https://doi.org/10.3390/fi14080244.

7. Smiti, S. and Soui, M. (2020). Bankruptcy Prediction Using Deep Learning Approach Based on Borderline SMOTE. Information Systems Frontiers. doi:https://doi.org/10.1007/s10796-020-10031-6.

8. Alam, T.M., Shaukat, K., Mushtaq, M., Ali, Y., Khushi, M., Luo, S. and Wahab, A. (2020). Corporate Bankruptcy Prediction: An Approach Towards Better Corporate World. The Computer Journal, 64(11). doi:https://doi.org/10.1093/comjnl/bxaa056.

9. Speiser, J.L., Miller, M.E., Tooze, J. and Ip, E. (2019). A comparison of random forest variable selection methods for classification prediction modeling. Expert Systems with Applications, 134, pp.93–101. doi:https://doi.org/10.1016/j.eswa.2019.05.028.

10. Marso, S. and Merouani, M. (2020). Bankruptcy Prediction using Hybrid Neural Networks with Artificial Bee Colony. [online] Available at: https://www.engineeringletters.com/issues_v28/issue_4/EL_28_4_26.pdf.

11. Ahmadpour Kasgari, A., Divsalar, M., Javid, M.R. and Ebrahimian, S.J. (2012). Prediction of bankruptcy Iranian corporations through artificial neural network and Probit-based analyses. Neural Computing and Applications, 23(3-4), pp.927–936. doi:https://doi.org/10.1007/s00521-012-1017-z.

12. Kim, H., Cho, H. and Ryu, D. (2021). Corporate Bankruptcy Prediction Using Machine Learning Methodologies with a Focus on Sequential Data. Computational Economics. doi:https://doi.org/10.1007/s10614-021-10126-5.

13. Staudemeyer, R.C. and Morris, E.R. (2019). Understanding LSTM -- a tutorial into Long Short-Term Memory Recurrent Neural Networks. arXiv:1909.09586 [cs]. [online] Available at: https://arxiv.org/abs/1909.09586.

14. Kazim, E. and Koshiyama, A.S. (2021). A high-level overview of AI ethics. Patterns, 2(9), p.100314. doi:https://doi.org/10.1016/j.patter.2021.100314.

15. Hagendorff, T. (2020). The Ethics of AI Ethics: An Evaluation of Guidelines. Minds and Machines, [online] 30(1), pp.99–120. doi:https://doi.org/10.1007/s11023-020-09517-8.