**FINANCIAL DISTRESS DETECTION USING ENSEMBLE LEARNING**

BY

PHOEBE WONG HUI LEI

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)

Faculty of Information and Communication Technology

(Kampar Campus)

FEB 2025

# COPYRIGHT STATEMENT



This Final Year Project report is submitted in partial fulfillment of the requirements for the degree of Bachelor of Computer Science (Honours) at Universiti Tunku Abdul Rahman (UTAR). This Final Year Project report represents the work of the author, except where due acknowledgment has been made in the text. No part of this Final Year Project report may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

# ACKNOWLEDGEMENTS

I would like to express thanks and appreciation to my supervisor, Dr Tong Dong Ling who has given me this bright opportunity to engage research in financial distress detection using ensemble learning. Besides that, she has given me a lot of guidance in order to complete this project. When I was facing problems in this project, the advice from them always assists me in overcoming the problems. Again, a million thanks to my supervisor.

# ABSTRACT

Financial distress prediction is a crucial role, as an “early warning” for a company to address with the financial risk including restructuring the financial strategies and managing the operating costs effectively. Over time, several approaches have been developed for financial distress predictions, which are methods based on the financial ratios, single classification model and ensemble learning. However, few challenges have been found out from the previous approaches such as the imbalance datasets, limitations on the financial ratios and the auditor biases on selecting financial ratios. In this thesis focuses on ensemble learning are known to capture large and complex datasets and provide more robust result. The aim of the project is to identify the optimal ensemble learning technique in detecting financial distress risk.

Area of Study: Financial distress detection, ensemble learning
Keywords: Financial distress detection, ensemble learning, financial ratios, bagging, stacking, boosting

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# Chapter 1
# Introduction

Financial distress is a scenario in which an individual or a company fails to generate the revenues to cover their financial responsibilities. There are few signs of financial distress, including declining sales which may be due to the low production quality, cash flow issue due to unresolve debts, increasing of operating costs and more [1]. When a company involves in financial stress, it could end up lead to bankruptcy and damage the creditworthiness. Hence, financial distress prediction acts as an "early warning" to the top management, stakeholders to control expenses effectively and perform strategies to improve the cash flow and reduce costs to maintain the financial stability. In the previous decades, the proposed approach for financial distress prediction have evolved into two main categories, market-based models and accounting-based models. Market based models depends on the stock market price to reflect the information exists in accounting statements and those not in the accounting statement. These marketing variables unlikely to be affected by the firm accounting policies. Accounting ratio-based models rely on a large number of accounting ratios with the ratio weightings determined by analyzing on a sample of failed and non-failed firms. Due the distribution of accounting ratios changes over time, it is recommended that such models be redeveloped periodically. The limitation of this approach, including information on accounting statements present past performance of a firm could be and could not be informative in predicting the future, and accounting numbers are subject to manipulation by management.[2]

## 7.2 Problem Statement and Motivation

Nowadays, it is increasingly common for companies, even well managed ones, to encounter a financial crisis or loses, which the worst case of might lead to bankruptcy. There are few causes contributed to these financial difficulties including poor economy, weak financial management, unexpected expenses or loss of revenues or income. However, previous approaches of financial distress predictions have struggled with issues like imbalanced data distribution, auditors' lack of experience leading to the

selection of incorrect financial ratios, and the limitations of single classification models, making accurate predictions difficult. To address these challenges, researchers have turned to machine learning approaches. In this thesis, ensemble learning as a method for classifying financial distress, which is expected to provide higher accuracy rate and handle the large datasets. [3]

## 1.2 Objectives

The aim of the project is to identify the optimal ensemble learning technique in detecting financial distress risk**.** To achieve the aim, there are 3 objectives that have been set as below:

1) Familiarize the architecture ensemble learning techniques

2) Compare and contrast three ensemble learning techniques (stacking, bagging and boosting) using classifiers like logistic regression and decision tree in classifying financial status of companies

3) Relate the findings to interpret business implications of financial distress

## 1.3 Project Scope and Direction

The scope of the project is conducting study on ensemble learning techniques which are bagging, boosting, stacking applied to financial distress classification problem. Additionally, it also studies how these ensemble methods can address the challenges from traditional approach such as manual auditing or calculating the Altman z-score, and the interpretation business implications.

## 1.4 Contributions

The project highlights how ensemble learning techniques improve the accuracy and robustness in detecting the financial risk compared to traditional approaches such as manual auditing or statistical computation. For example, manual auditing produce inconsistency results since it is depending on the auditors' knowledge, thoroughness and materiality levels applied. Additionally, both auditing or statistical computation process are time consuming. With the help of the ensemble learning, it mitigates the

challenges by improving time efficiency, consistency result and better quality on detecting the financial risk.

## 1.5 Report Organization

This report is 3rganizat into 6 chapters: Chapter 1 Introduction, Chapter 2 Literature Review, Chapter 3 System Design, Chapter 4 System Implementation and Testing, Chapter 5 System Outcome and Discussion, Chapter 6 Conclusion. The first chapter is the introduction of this project which includes problem statement, project background and motivation, project scope, project objectives, project contribution, highlights of project achievements, and report 3rganization. The second chapter is the literature review carried out on financial distress detection on the traditional approach, machine learning approach and ensemble learning approach. The third chapter is a proposed methodology of ensemble learning environment like bagging, stacking and boosting is presented . The fourth chapter is regarding the details on how to implement the design of the ensemble learning system. Furthermore, the fifth chapter reports the outcome of implementation of the ensemble learning system on detecting the financial distress

# Chapter 2
# Literature Review

## 2.1 Traditional approach

Traditional bankruptcy prediction models often rely on the manual computation of financial ratios to assess a company's financial condition and determine whether it is in a stable state or facing financial distress. Notable examples of these models include the Altman Z-Score, Springate S-Score, Zmijewski X-Score, and Grover G-Score. Each of these models utilizes different financial ratios to evaluate the likelihood of bankruptcy.

Altman Z-score formula is developed in 1967 by NYU Stern Finance Professor Edward Altman and was published in 1968. The model utilizes five financial ratios that can be obtained from a company's annual 10-K report, including profitability, leverage, liquidity, solvency, and activity to forecast the probability of an analyzed company under financial distress. The formula for the Altman Z-score is as follows:

Altman Z-Score = 1.2A + 1.4B + 3.3C + 0.6D + 1.0E
Where:

- A = working capital / total assets
- B = retained earnings / total assets
- C = earnings before interest and tax / total assets
- D = market value of equity / total liabilities
- E = sales / total assets

If the ratio obtained is below 1.8, the analyzed company classified as likely under financial distress, and the score obtained is more than 3, indicating the company not likely going bankrupt [4].

The Springate score is a bankruptcy prediction model derived from the Altman model. Initially, it considered 19 financial ratios, but ultimately only utilize four selected coefficients. The formula for the Springate score is as follows:

Springate score = 1.03A + 3.07B + .66C + .4D

Definitions:

A = Working capital / Total assets

B = EBIT / Total assets

C = Profit before tax / Current liabilities

D = Revenue / Total assets

If the score obtained is greater than 0.862, the analyzed company is in a safe state, otherwise it is classified as being in financial distress [5].

The Zmijewski score is another bankruptcy prediction model based on performance, leverage, and financial liquidity. Its formula is

Zmijewski score = -4.336 – 4.513 * (Net income / Total assets) + 5.679 * (Total liabilities / Total assets) + 0.004 * (Current assets/ Current liabilities)

In this model, a higher ratio obtained indicating a higher likelihood of the analyzed company to face bankruptcy [6].

Grover model is a model created by readapting Altman-Z score model. It consists of X1 and X3 variables from Altman Z Score and incorporate with profitability ratios indicated as ROA. The formula of the Grover model is:

G = 1.650X1 + 3.404X2 – 0.016ROA + 0.057

Description:

X1 = Working capital or Total assets

X2 = Earnings before interest and taxes or total assets

ROA = net income or total assets

If the score obtained is greater than 0.01 indicating the analyzed company is in a safe state, while score below of this threshold shows that it is under financial distress [7].

Apart from that, traditional approach for financial distress prediction is auditing. Few processes done by the auditor to access the fraud detection, including utilization of forensic techniques, integration of data analytics, professional judgement and skepticism, regulatory reforms and oversight mechanisms, collaborative efforts with regulatory authorities. By leveraging forensic techniques and data analytics to detect financial distress, auditors enhance their ability to identify underlying trends and unusual transactions that may indicate potential fraud [8]. A survey has conducted with the aim to study the purpose of hiring a financial auditor, the respondents agreed that 25% applied financial auditor as a desire to identify and prevent financial fraud or abuse [9]. It highlighted the usage of auditing in detecting the financial distress risk.

## 2.2 Machine Learning approach

| | Algorithms | Hyper-Parameter | AUC | Accuracy | Precision | Recall | F1 Score |
|---|---|---|---|---|---|---|---|
| 1 | Extreme Gradient Boosting | booster = “gbtree”, n_estimator = 100, max_depth = 1, random_state = 42 | 0.9702 | 0.9566 | 0.8726 | 0.8354 | 0.8536 |
| 2 | Random Forest | max_depth = 14,n_estimators = 100, random_state = 42 | 0.9788 | 0.9529 | 0.8535 | 0.8272 | 0.8401 |
| 3 | Logistic Regression | random_state = 42 | 0.9303 | 0.8623 | 0.8854 | 0.5148 | 0.6511 |
| 4 | Artificial Neural Network | n_hidden = 2, max_iter = 200, activations = relu, Optimizer = adam | 0.9034 | 0.9168 | 0.8025 | 0.6811 | 0.7368 |
| 5 | Decision Trees | Criterion = “gini”, max_depth = 14, random_state = 42 | 0.8848 | 0.9251 | 0.828 | 0.7065 | 0.7625 |
| 6 | Support Vector Machine | Kernel = “rbf”, probability = True, class_weight = “balanced”, random_state = 42 | 0.7889 | 0.8789 | 0.9427 | 0.4022 | 0.5815 |

*Figure 2.2.1 Comparison between machine learning models*

**Table 11**
AUC Results of the Financial Distress Prediction Models (16 Variables).

*Panel A: The AUC Results*

| Test Years | Train Periods | Train Samples | Test Samples | ST in Test Samples (%) | Non-ST in Test Samples (%) | Machine Learning Models | | | Z-Score Models | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | CART | AdaBoost | CUSBoost | AZM | SVZM | SM | ZMN |
| 2018 | 2012–2017 | 881 | 157 | 1.91 % | 98.09 % | 0.669 | 0.716 | 0.733 | 0.452 | 0.522 | 0.587 | 0.567 |
| 2019 | 2012–2018 | 1038 | 151 | 2.65 % | 97.35 % | 0.729 | 0.745 | 0.763 | 0.284 | 0.299 | 0.253 | 0.628 |
| 2020 | 2012–2019 | 1189 | 166 | 3.01 % | 96.99 % | 0.736 | 0.788 | 0.825 | 0.050 | 0.055 | 0.212 | 0.655 |
| 2021 | 2012–2020 | 1355 | 157 | 4.46 % | 95.54 % | 0.776 | 0.813 | 0.846 | 0.296 | 0.287 | 0.313 | 0.599 |
| **Average AUC** | | | | | | **0.728** | **0.766** | **0.792** | **0.271** | **0.291** | **0.341** | **0.612** |

*Panel B: The AUPR Results*

| Test Years | Train Periods | Train Samples | Test Samples | ST in Test Samples (%) | Non-ST in Test Samples (%) | Machine Learning Models | | | Z-Score Models | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | CART | AdaBoost | CUSBoost | AZM | SVZM | SM | ZMN |
| 2018 | 2012–2017 | 881 | 157 | 1.91 % | 98.09 % | 0.139 | 0.157 | 0.184 | 0.026 | 0.035 | 0.047 | 0.032 |
| 2019 | 2012–2018 | 1038 | 151 | 2.65 % | 97.35 % | 0.145 | 0.165 | 0.197 | 0.022 | 0.023 | 0.024 | 0.044 |
| 2020 | 2012–2019 | 1189 | 166 | 3.01 % | 96.99 % | 0.181 | 0.197 | 0.203 | 0.019 | 0.019 | 0.022 | 0.080 |
| 2021 | 2012–2020 | 1355 | 157 | 4.46 % | 95.54 % | 0.197 | 0.219 | 0.247 | 0.073 | 0.072 | 0.055 | 0.087 |
| **Average AUPR** | | | | | | **0.166** | **0.185** | **0.208** | **0.035** | **0.037** | **0.037** | **0.061** |

*Figure 2.2.2 Comparison of auc score between machine learning models and z- score models*

Due to the limitations of traditional approaches like linear relationships, homogeneity of variances and independence assumptions, machine learning methods have been introduced to mitigate the challenges. According to a study, it has applied six algorithms in predicting financial distress, which are extreme gradient boosting, random forest, logistic regression, ANN, decision tree and support vector machine, and make a comparison of performance as shown in figure 2.2.1. Extreme gradient boosting and random forest have outperformed than others with higher accuracy of 0.9566 and 0.9529 respectively. Besides that, F1 score for extreme gradient boosting is the highest F1 score of 0.8536, indicating that a good balance between precision and recall, have the capability to identify positive instances while minimizing false positives and false

negatives [10]. In addition, a study has conducted a comparison between machine learning models and z- score models. Based on what has shown in figure 2.2.2, machine learning models have outperformed the Z-score model in both AUC and APR result, indicating that it has better capabilities in detecting financial distress risk [11].

## 2.3 Ensemble Learning approach

Due to the limitations of machine learning methodology, people have switched from machine learning to ensemble learning approach with the aim of providing a better accuracy in detecting the financial distress. Ensemble learning is a combination of multiple learners with the aim to improve prediction performance than a single learner. The advantage of ensemble learning is bias variance tradeoff. Bias is referring to the difference between predicted and true values, whereas variance is referring to the differences between in predictions across multiple versions of a given model. If variance and bias increases, the more likely the model has lower accuracy. Thus, these two variables are closely related to the accuracy of the model on training and testing data. With the concept of aggregating two or more models, ensemble learning reduces the overall error rate and remains each model's own complexities and advantages. Parallel and sequential are the main categories in ensemble learning methods, and each of it have its differences. Parallel methods train each basic learner independently and parallelly. In contrast, sequential methods focus on training a new base learner to learn from the previous model and reducing the error made. Bagging, stacking and boosting are the most popular ensemble learning methods [12].

TABLE I. COMPARISON OF PREDICTION ERROR RATES (LOGISTIC REGRESSION AS BASIC CLASSIFIER)

| method | mean error rate (%) | std. (%) | paired-*t* test[a] |
|---|---|---|---|
| single classifier | 15.94 | 3.98 | 0.079* |
| ensemble | 15.11 | 2.93 | |

TABLE II. COMPARISON OF PREDICTION ERROR RATES (SVM AS BASIC CLASSIFIER)

| method | mean error rate (%) | std. (%) | paired-*t* test |
|---|---|---|---|
| single classifier | 28.33 | 9.97 | (0.00)** |
| ensemble | 19.67 | 4.66 | |

[a] * and ** indicates a statistically significant difference at the 0.1 and 0.05 level, respectively.

*Figure 2.3.1 Prediction Error Rate of SVM and Logistic Regression*

According to a study published, it has performed comparisons on the prediction error rate for single classifier and ensemble learning as shown in figure 2.3.1. It shows that the mean error rate for both SVM and logistic regression as basic classifier in ensemble method have lower mean error rate of 19.67% and 15.11% respectively than the single classifier method. Thus, it has proven that ensemble learning has better predicting performance specifically on logistics regression as the basic classifier.

Besides that, there is another study conducted using cost-sensitive stacking ensemble learning, with the aim of minimize total misclassification costs. Extreme Gradient Boosting (XGBoost) has been applied to remove the irrelevant features and remain ones. If the information obtained above the threshold of 0, retain the features, and otherwise eliminate it. Later, the data will be arranged in decreasing order of information gain score. Later, sequential forward selection technique (SFS) as wrapper method is applied to select the optimal feature subset with the highest balance accuracy (BACC). SFS generates candidate feature subsets by iteratively adding the feature with the highest information gain.

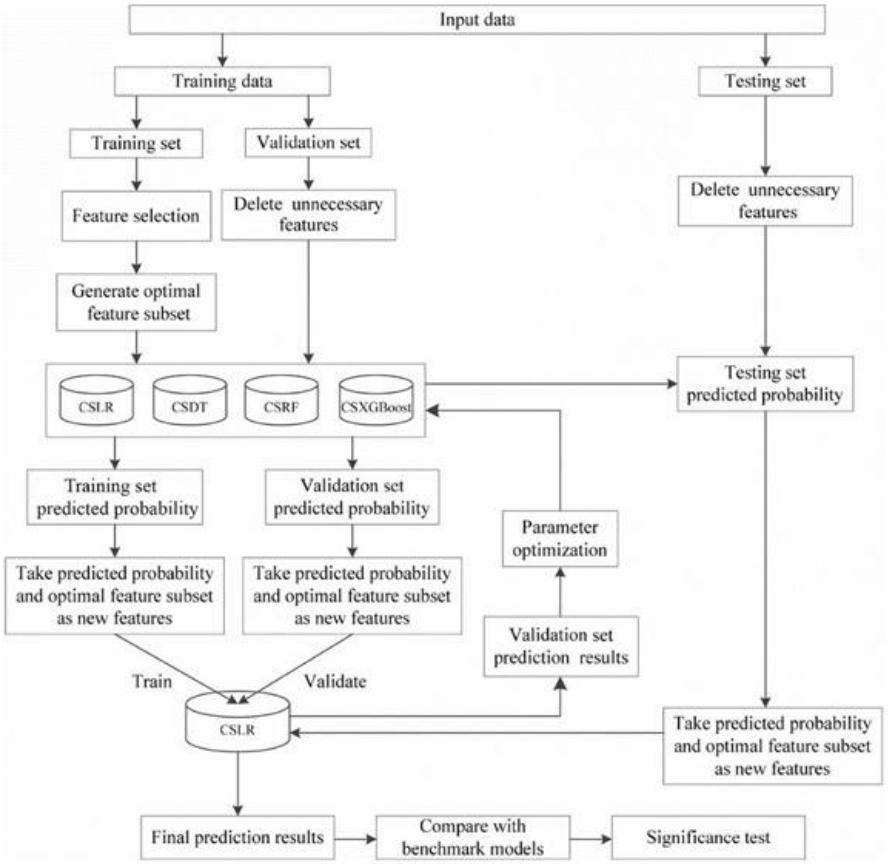



*Figure 2.3.2 Flow chart of cost-sensitive stacking ensemble learning*

| Data Year | TP | FN | TN | FP | ACC | Recall | F-measure | AUC | G-mean | Type I | Type II |
|---|---|---|---|---|---|---|---|---|---|---|---|
| *t*-2 | 109 | 11 | 550 | 15 | 0.9620 | 0.9083 | 0.8934 | 0.9409 | 0.9403 | 0.0265 | 0.0917 |
| *t*-3 | 122 | 6 | 523 | 34 | 0.9416 | 0.9531 | 0.8592 | 0.9460 | 0.9460 | 0.0610 | 0.0469 |
| *t*-4 | 118 | 10 | 521 | 36 | 0.9328 | 0.9219 | 0.8369 | 0.9286 | 0.9286 | 0.0646 | 0.0781 |
| *t*-5 | 115 | 13 | 528 | 29 | 0.9387 | 0.8984 | 0.8456 | 0.9232 | 0.9229 | 0.0521 | 0.1016 |

*Figure 2.3.3 Prediction performance CSStacking after feature selection for time periods t-m*

| Data Year | TP | FN | TN | FP | ACC | Recall | F-measure | AUC | G-mean | Type I | Type II |
|---|---|---|---|---|---|---|---|---|---|---|---|
| *t*-2 | 87 | 36 | 447 | 115 | 0.7796 | 0.7073 | 0.5354 | 0.7513 | 0.7501 | 0.2046 | 0.2927 |
| *t*-3 | 112 | 11 | 409 | 153 | 0.7606 | 0.9106 | 0.5773 | 0.8192 | 0.8140 | 0.2722 | 0.0894 |
| *t*-4 | 98 | 25 | 492 | 70 | 0.8613 | 0.7967 | 0.6735 | 0.8361 | 0.8352 | 0.1246 | 0.2033 |
| *t*-5 | 107 | 16 | 482 | 80 | 0.8599 | 0.8699 | 0.6903 | 0.8638 | 0.8638 | 0.1423 | 0.1301 |

*Figure 2.3.4 Prediction performance CSStacking without feature selection for time periods t-m*

| Data Year | TP | FN | TN | FP | ACC | Recall | F-measure | AUC | G-mean | Type I | Type II |
|---|---|---|---|---|---|---|---|---|---|---|---|
| *t*-2 | 108 | 15 | 517 | 45 | 0.9124 | 0.8780 | 0.7826 | 0.8990 | 0.8987 | 0.0801 | 0.1220 |
| *t*-3 | 109 | 9 | 537 | 30 | 0.9431 | 0.9237 | 0.8482 | 0.9354 | 0.9353 | 0.0529 | 0.0763 |
| *t*-4 | 107 | 21 | 532 | 25 | 0.9328 | 0.8359 | 0.8231 | 0.8955 | 0.8935 | 0.0449 | 0.1641 |
| *t*-5 | 86 | 15 | 537 | 47 | 0.9095 | 0.8515 | 0.7350 | 0.8855 | 0.8848 | 0.0805 | 0.1485 |

*Figure 2.3.5 Prediction performance of Stacking after feature selection for time periods t-m*

By comparing the result of CSStacking and Stacking after performed feature selection in between figure 2.3.3 and 2.3.5, it can be observed that CSStacking model has higher F measure, AUC, G- mean and Type II error than other model, indicating that combining Stacking and cost-sensitive learning can improve the model's predictive performance

# Chapter 3 System Methodology/Approach OR System Model

## 3.1 System Block Design

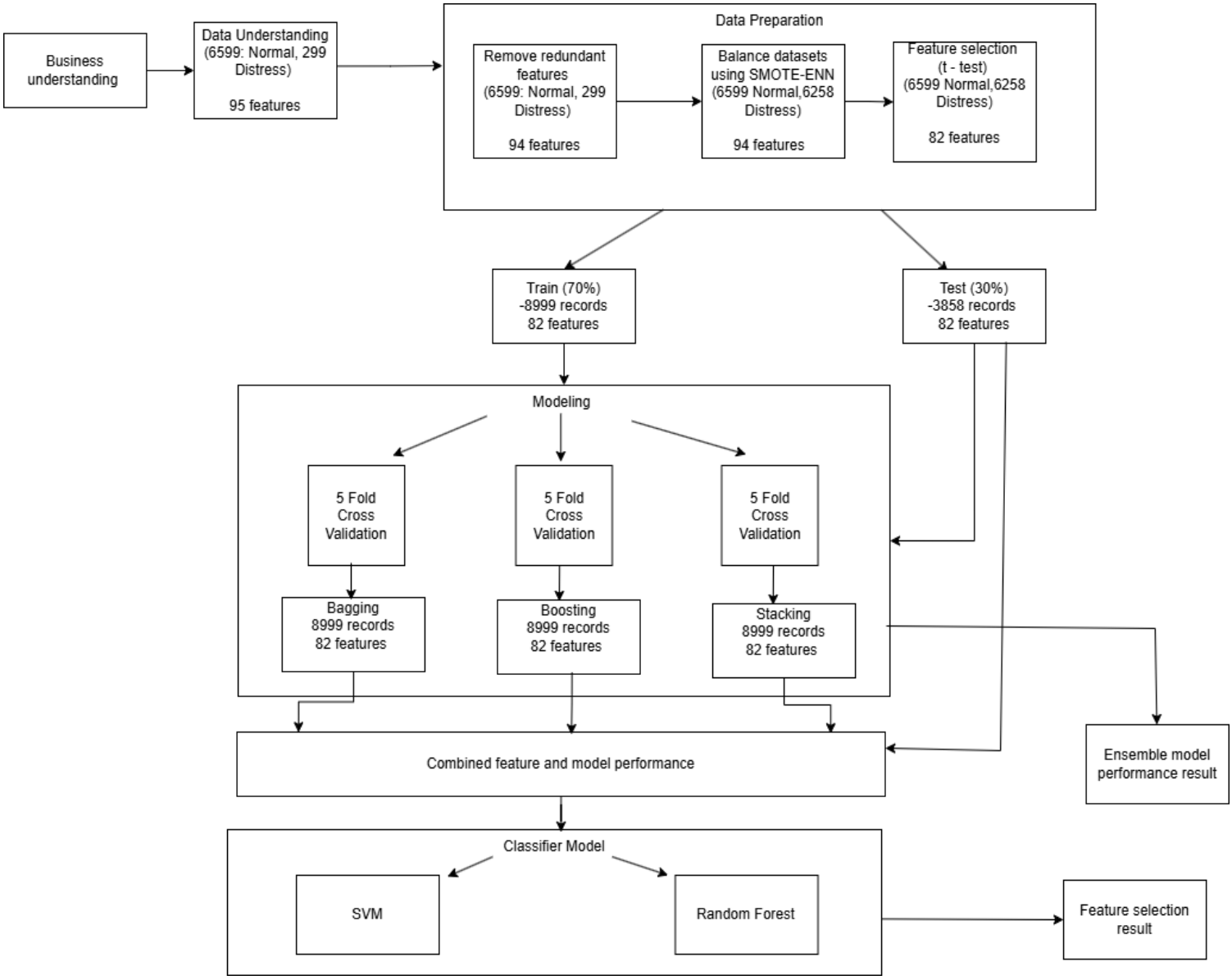



*Figure 3.1.1 Flow of the system methodology*

Business understanding

Before performing the project, some research studies have been conducted in the areas of financial distress, financial ratios including how the ratio is being derived, ensemble learning, basic classifier.

Data understanding

Dataset applied in this project is Taiwanese Bankruptcy Prediction dataset [10], which were collected from the Taiwan Economic Journal for the years 1999 to 2009 from financial ratios and corporate governance indicators, and the bankruptcy was defined according to business regulations of the Taiwan Stock Exchange. There are 6819 records with 95 features.

Data preparation

The redundant feature, "Net Income Flag" has been removed since it does not contribute much to the target value. In the dataset, there is an imbalance data distribution of two classes, which 6599 as non-bankrupt and 220 as the distressed one. It has utilized SMOTE – ENN to solve the problem. SMOTE verifies k nearest neighbours, and then generate the synthetic samples to reach the same size as the majority class [11]. After resampling, dataset contains a total of 12857 companies, which 6258 companies with bankruptcy status and 6599 companies with non-bankruptcy status.

Feature selection

T-test is applied for feature selection, since the target value is in binary form (0 and 1), and evaluating the relationship between means of the numerical features between the target value [18]. Variables with p value that is less than 0.05 will be selected. At the end, there is a total of 82 features being selected.

Data splitting

The data is initially split into 70% for training and 30% for testing. The training dataset contains 8,999 records with 82 features, while the test dataset contains 3,858 records, also with 82 features

Modeling

In the project, Logistic Regression and Decision Tree have been selected to apply in three ensemble learning environments, which are bagging, stacking and adaboosting (adaptive boosting).

Basic classifier selection

(i) Decision Tree

| Hasil: | precision | recall | f1-score | support |
|---|---|---|---|---|
| CUKUP SEHAT | 0.57 | 0.67 | 0.62 | 12 |
| DISTRESS | 0.69 | 0.71 | 0.70 | 35 |
| KURANG SEHAT | 0.75 | 0.72 | 0.74 | 54 |
| SANGAT SEHAT | 0.99 | 0.96 | 0.97 | 172 |
| SEHAT | 0.87 | 0.95 | 0.91 | 42 |
| accuracy | | | 0.88 | 315 |
| macro avg | 0.77 | 0.80 | 0.79 | 315 |
| weighted avg | 0.88 | 0.88 | 0.88 | 315 |

(a) Decision Tree f1-score accuracy results

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.36 | 0.33 | 0.35 | 12 |
| 1 | 0.81 | 0.63 | 0.71 | 35 |
| 2 | 0.68 | 0.80 | 0.74 | 54 |
| 3 | 0.95 | 0.90 | 0.92 | 172 |
| 4 | 0.65 | 0.81 | 0.72 | 42 |
| accuracy | | | 0.82 | 315 |
| macro avg | 0.69 | 0.69 | 0.69 | 315 |
| weighted avg | 0.83 | 0.82 | 0.82 | 315 |

(b) Naïve Bayes f1-score accuracy results

*Figure 3.1.2 Comparison of Decision Tree and Naïve Bayes*

| | LR | | | RF | | | DT | | | SVM | | | NB | | | kNN | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | T-1 | T-2 | T-3 | T-1 | T-2 | T-3 | T-1 | T-2 | T-3 | T-1 | T-2 | T-3 | T-1 | T-2 | T-3 | T-1 | T-2 | T-3 |
| Accuracy | 87 | 80 | 81 | 89 | 96 | 94 | 90 | 97 | 94 | 84 | 89 | 92 | 82 | 85 | 97 | 87 | 89 | 92 |
| Precision | 87 | 80 | 82 | 89 | 96 | 94 | 91 | 97 | 94 | 84 | 90 | 92 | 82 | 85 | 96 | 87 | 89 | 89 |
| Sensitivity | 87 | 79 | 81 | 88 | 97 | 94 | 90 | 97 | 94 | 84 | 89 | 92 | 82 | 85 | 97 | 87 | 89 | 88 |
| F-measure | 87 | 80 | 81 | 89 | 96 | 94 | 90 | 97 | 94 | 84 | 89 | 92 | 82 | 85 | 97 | 87 | 89 | 87 |

*Figure 3.1.3 Comparison of machine learning models in financial distress prediction*

Classifier that has been selected for this project is decision tree. According to research, it has made a comparison on the performance result in financial distress prediction at Rural Banks in Indonesia between decision tree and naïve bayes. Based on the result shown in figure 3.2.4.2, decision tree achieves a slightly better accuracy of 0.88 than naïve bayes with 0.82, indicating that it has the capability in predicting financial status of companies in different classes. The macro average precision for all classes in decision tree is 0.77 which is notably higher than naïve bayes with only 0.66. This shows the decision tree demonstrates better performance in classifying financial status including the minority class “Cukup Sehat” [15]. Apart from that, there is also another conducted in comparing the machine learning performance in financial

distress prediction on SME in Turkey in different time regions (t-1, t-2 and t-3). As a general overview, decision tree has a consistent performance in accuracy, precision, sensitivity and f-measure, by maintaining above 90% [24]. It indicates the robustness of the model by predicting distress and non-distress company in three-time regions. Therefore, it showed that decision tree has a better performance in financial distress detection.

(ii) Logistic Regression

| Metric | Logit Results | ANN Results | Difference |
|---|---|---|---|
| Accuracy | 98.00% | 82.50% | 15.00% |
| Sensitivity | 94.20% | 84,00% | 10,50% |
| Specificity | 99.30% | 82.00% | 17.00% |

*Figure 3.1.4 Comparison of Logistics Regression and ANN*

Another basic classifier chosen for this project is logistic regression. According to a study published, it has performed comparisons on the prediction error rate for single classifier and ensemble learning on the logistics regression and SVM, it has displayed that applying logistic regression as the base classifier in the ensemble learning has lower error rate of 15.11% as compared to SVM with 19.67% as show in figure 2.3.1[6]. In addition, a study has conducted to compare the performance result from Artificial Neural Networks (ANN) and logistic regression methodologies in financial distress prediction. Based on figure 3.4.1, it is observed that logistics regression has outperformed, achieving accuracy of 98% than ANN with 82.5%. Sensitivity has particularly emphasized, because it proves that logistic regression can classify 94.2% of the distress (positive) company than ANN with only 84% [14]. Therefore, it showed that logistic regression has a better performance in financial distress detection.

Ensemble Learning Environment

(i) Bagging

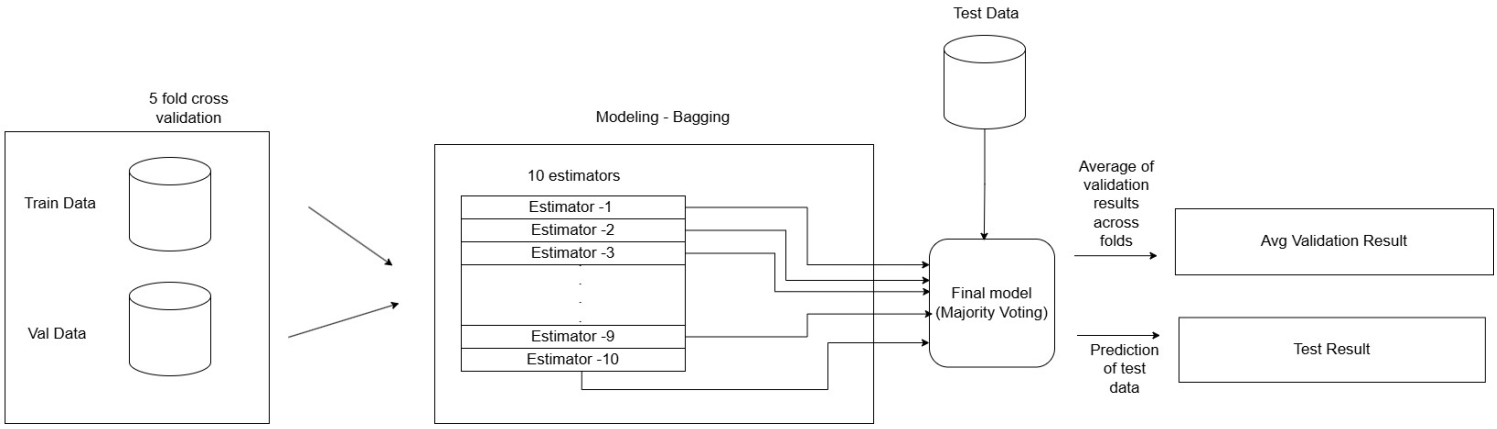



*Figure 3.1.5 Sample architecture of bagging ensemble learning*

Bagging is an ensemble learning method, with a combination of weak learners and become a strong learner. Each base model is trained independently on subset data and the predictions are aggregated through major voting to obtain the final prediction [19]. In this project, ten estimators were employed and trained using 5-fold cross-validation. For each fold's training samples, models are trained through bootstrap sampling method. After completing model training on one-fold, majority voting will apply to generate prediction on test and validation result. Upon finishing all 5 folds, evaluation was conducted by evaluation will be made by averaging the performance metrics across the folds. Finally, the model was fitted on the entire training set and tested on the test dataset to assess its generalization performance.In figure 3.1.5. has visualized the architecture of bagging environment.

(ii) Adaboosting

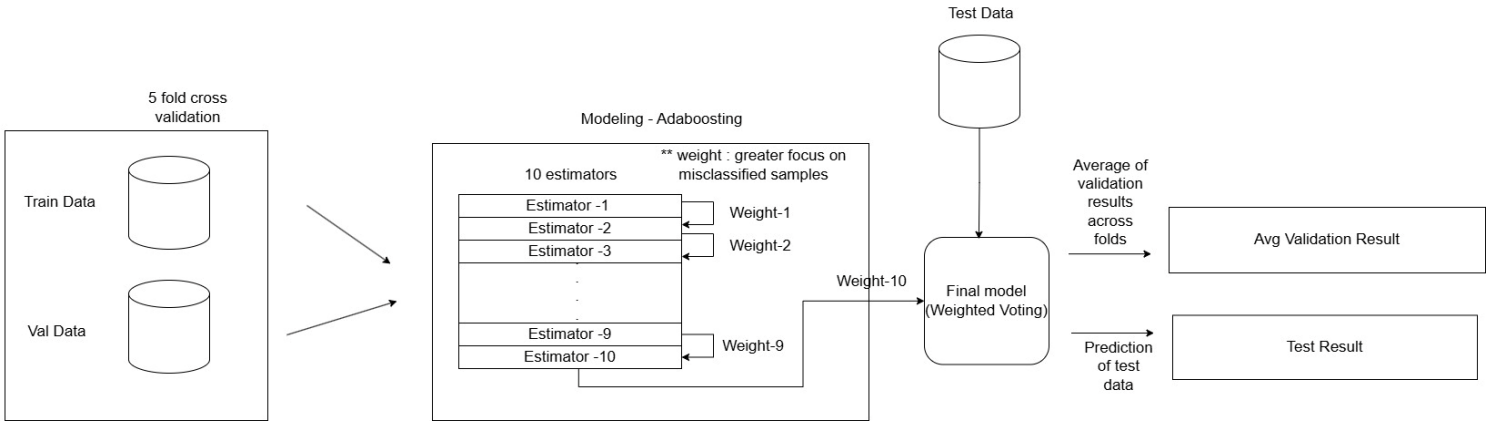



*Figure 3.1.6: Sample architecture of adaboosting (adaptive boosting) ensemble learning*

Boosting is an ensemble learning method that transforms multiple weak models into a single strong learner. It primarily focuses on sequential model training by gradually increasing the weights of misclassified instances until the errors are minimized and to achieve better accuracy performance [20]. In this project, AdaBoost was selected for its efficiency in handling financial distress detection. Ten estimators were employed and trained using 5-fold cross-validation. For each fold, models were trained through weighted sampling: after each model was trained, the weights of misclassified samples were increased, while those of correctly classified samples decreased. After completing model training on one-fold, weighted voting was applied to generate predictions for the validation set. Upon finishing all 5 folds, evaluation was conducted by averaging the performance metrics across the folds. Finally, the model was fitted on the entire training set and tested on the test dataset to assess its general performance. In figure 3.1.6 has visualized the architecture of adaboosting environment.

(iii) Stacking

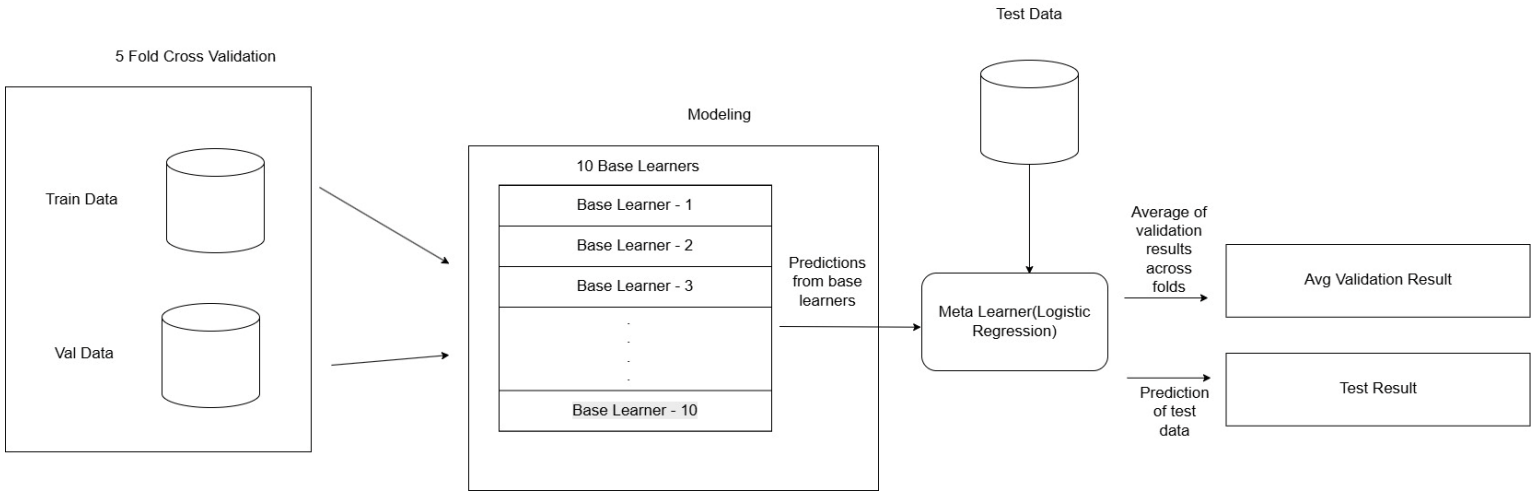



*Figure 3.1.7 Sample architecture of stacking ensemble learning*

Stacking is an ensemble learning method which new model is stacked up on top of the others. It emphasizes training multiple base models (level 0 models) parallelly and according to the combination of outputs to build a new model, known as meta model (level 1 model). The input of the meta model is the prediction from the individual base models [21]. For training samples, models are trained with datasets. Once completed trained, the results are stacked to form a new dataset and fed to the meta model to make the final predictions.

Performance Evaluation of Test Result

After training the base learners (logistic regression and decision tree) in three ensemble learning environments, evaluation has been performed on the test result and the computational time. The evaluation prioritized the false negative rate, as misclassifying financially distressed companies as healthy poses significant risk. Besides that, computational time was recorded with two conditions, either the model has been looping 300 times, or the model converged early by maintaining a constant result for consecutive 5 times.

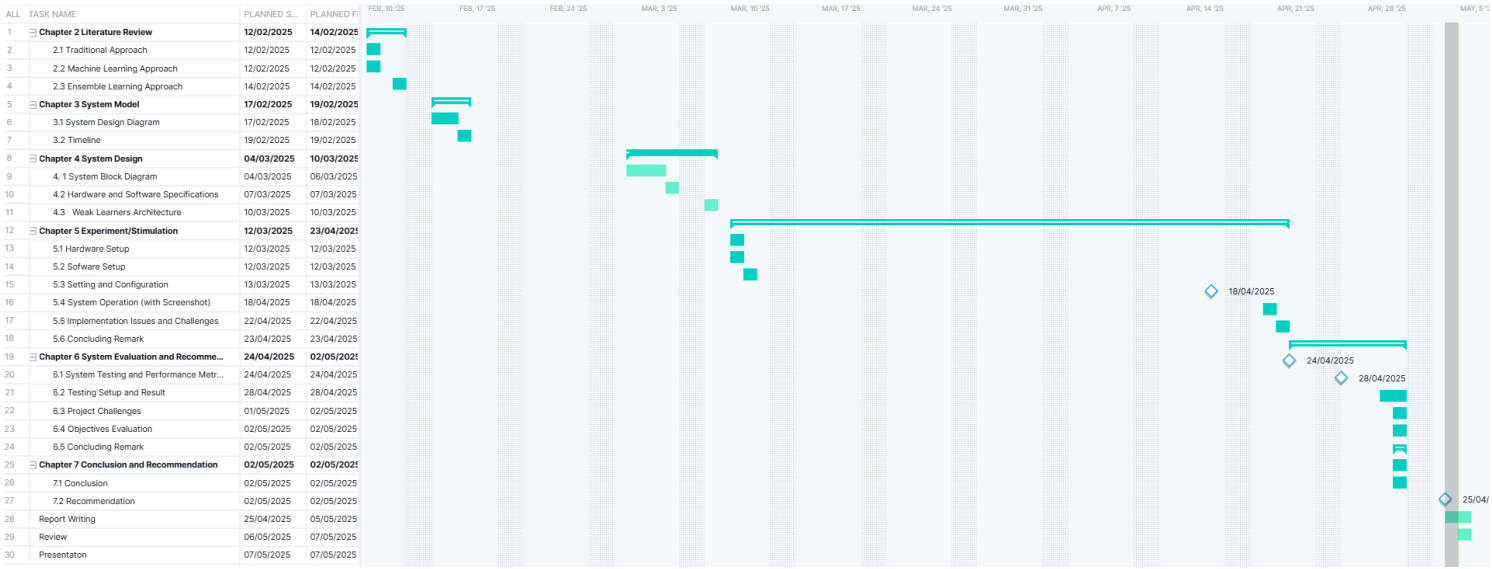
Performance Evaluation of Model Performance

Apart from that, significant features for each base learner in every ensemble learning environment have been identified by applying Permutation Importance Calculation and z-test. Features with a p-value below the threshold of 0.05 were considered statistically significant, with the purpose of exploring which features truly contribute to the model's predictive performance. Besides that, it also serves the purpose of proving whether the features selected from recommend ensemble learning techniques are robust across different classifiers by demonstrating strong predictive performance.

Model Training on Significant Feature

Two classifiers—Support Vector Machine (SVM) and Random Forest (RF)—were used to train models with significant features selected from both logistic regression and decision tree. In this process, various combinations of these features have been studied, including combined significant features from same base learner, overlapping features from same base learners and the combine features from recommended ensemble learning techniques. It is aimed at evaluating whether the selected features retain their predictive strength across different classifiers.

## 3.2 Timeline for FYP 2



| ALL | TASK NAME | PLANNED S... | PLANNED F... |
|---|---|---|---|
| 1 | **Chapter 2 Literature Review** | **12/02/2025** | **14/02/2025** |
| 2 | 2.1 Traditional Approach | 12/02/2025 | 12/02/2025 |
| 3 | 2.2 Machine Learning Approach | 12/02/2025 | 12/02/2025 |
| 4 | 2.3 Ensemble Learning Approach | 14/02/2025 | 14/02/2025 |
| 5 | **Chapter 3 System Model** | **17/02/2025** | **19/02/2025** |
| 6 | 3.1 System Design Diagram | 17/02/2025 | 18/02/2025 |
| 7 | 3.2 Timeline | 19/02/2025 | 19/02/2025 |
| 8 | **Chapter 4 System Design** | **04/03/2025** | **10/03/2025** |
| 9 | 4. 1 System Block Diagram | 04/03/2025 | 06/03/2025 |
| 10 | 4.2 Hardware and Software Specifications | 07/03/2025 | 07/03/2025 |
| 11 | 4.3 Weak Learners Architecture | 10/03/2025 | 10/03/2025 |
| 12 | **Chapter 5 Experiment/Stimulation** | **12/03/2025** | **23/04/2025** |
| 13 | 5.1 Hardware Setup | 12/03/2025 | 12/03/2025 |
| 14 | 5.2 Sofware Setup | 12/03/2025 | 12/03/2025 |
| 15 | 5.3 Setting and Configuration | 13/03/2025 | 13/03/2025 |
| 16 | 5.4 System Operation (with Screenshot) | 18/04/2025 | 18/04/2025 |
| 17 | 5.5 Implementation Issues and Challenges | 22/04/2025 | 22/04/2025 |
| 18 | 5.6 Concluding Remark | 23/04/2025 | 23/04/2025 |
| 19 | **Chapter 6 System Evaluation and Recomme...** | **24/04/2025** | **02/05/2025** |
| 20 | 6.1 System Testing and Performance Metr... | 24/04/2025 | 24/04/2025 |
| 21 | 6.2 Testing Setup and Result | 28/04/2025 | 28/04/2025 |
| 22 | 6.3 Project Challenges | 01/05/2025 | 02/05/2025 |
| 23 | 6.4 Objectives Evaluation | 02/05/2025 | 02/05/2025 |
| 24 | 6.5 Concluding Remark | 02/05/2025 | 02/05/2025 |
| 25 | **Chapter 7 Conclusion and Recommendation** | **02/05/2025** | **02/05/2025** |
| 26 | 7.1 Conclusion | 02/05/2025 | 02/05/2025 |
| 27 | 7.2 Recommendation | 02/05/2025 | 02/05/2025 |
| 28 | Report Writing | 25/04/2025 | 05/05/2025 |
| 29 | Review | 06/05/2025 | 07/05/2025 |
| 30 | Presentaton | 07/05/2025 | 07/05/2025 |



*Figure 3.2.1 Timeline for FYP2*

# Chapter 4 SYSTEM DESIGN

## 4.1 System Block Diagram

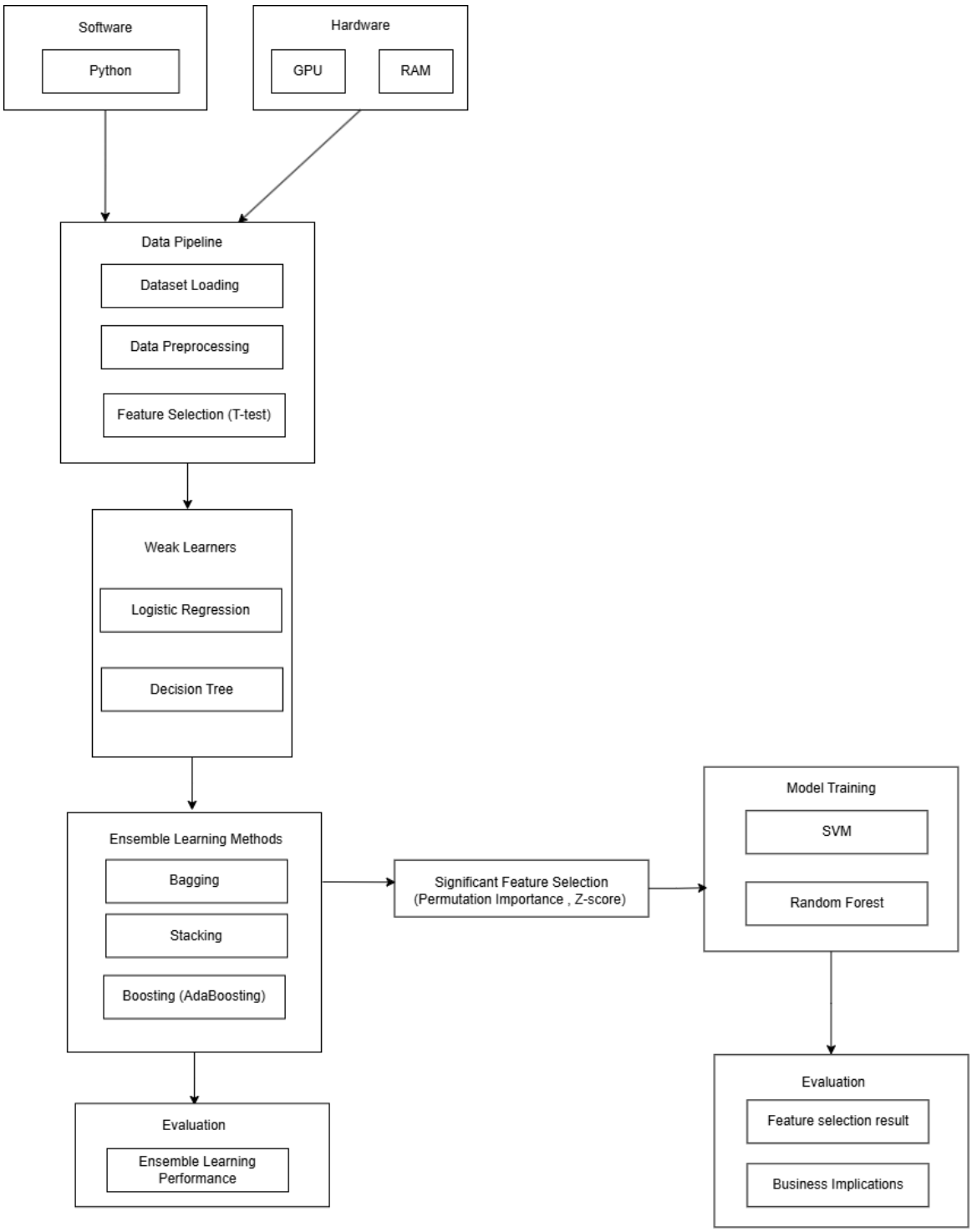



*Figure 4.1 System Block Diagram*

Figure 4.1 has presented an overview of the proposed financial distress detection framework. It visualizes the major components including hardware and software setup, data pipeline, weak learners, ensemble learning methods, and evaluation modules. The process begins with data loading and data preprocessing, followed by significant feature selection using t test. Later, weak learners such as logistic regression and decision tree are then fit into ensemble learning techniques like bagging, boosting, and stacking to evaluate which techniques has the best performance in financial distress detection. Significant features are extracted using permutation importance and z-score analysis and subsequently used to train final classifiers (SVM and Random Forest). Evaluation of the result not only based on the model performance but also relate to the business implications.

### 4.2 Hardware and Software Specifications

The hardware involved in this project is the computer. A computer issued for the purpose of training and testing the base model in different ensembles learning method in detecting financial distress. In table 4.2.1, it shows the specification of a laptop.

Table 4.2.1 Specifications of laptop

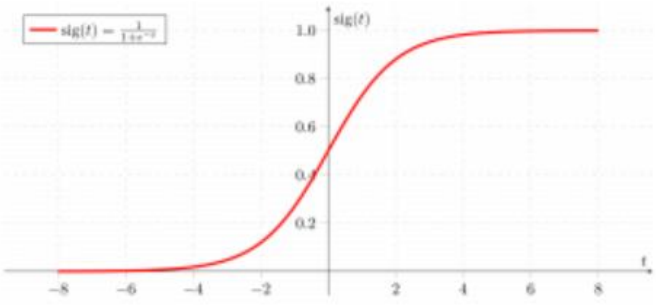
| **Description** | **Specifications** |
|---|---|
| Model | MateBook 13 |
| Processor | AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx 2.10 GHz |
| Operating System | Windows 10 |
| Graphic | NVIDIA GeForce GT 930MX 2GB DDR3 |
| Memory | 16GB DDR4 RAM |
| Storage | 461GB SATA HDD |

There are three software involved in the project. Python is the programming language used for developing an ensemble learning environment and perform performance evaluation. Google Colab served as the IDE for writing and executing the Python code. Excel is used as a preliminary in understanding on the datasets.

### 4.3 Weak Learners Architecture

#### 4.3.1 Logistic Regression

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




Sigmoid function

*Figure 4.3.1.1 Sigmoid function*

$$\frac{p(x)}{1-p(x)} = e^z$$

Applying natural log on odd. then log odd will be:

$$\log\left[\frac{p(x)}{1-p(x)}\right] = z$$

$$\log\left[\frac{p(x)}{1-p(x)}\right] = w \cdot X + b$$

$$\frac{p(x)}{1-p(x)} = e^{w \cdot X + b} \quad \cdots \text{Exponentiate both sides}$$

$$p(x) = e^{w \cdot X + b} \cdot (1 - p(x))$$

$$p(x) = e^{w \cdot X + b} - e^{w \cdot X + b} \cdot p(x))$$

$$p(x) + e^{w \cdot X + b} \cdot p(x)) = e^{w \cdot X + b}$$

$$p(x)(1 + e^{w \cdot X + b}) = e^{w \cdot X + b}$$

$$p(x) = \frac{e^{w \cdot X + b}}{1 + e^{w \cdot X + b}}$$

then the final logistic regression equation will be:

$$p(X; b, w) = \frac{e^{w \cdot X + b}}{1+e^{w \cdot X + b}} = \frac{1}{1+e^{-w \cdot X + b}}$$

*Figure 4.3.1.2 Equation of logistic regression*

Logistic regression is a supervised machine learning algorithm primarily used for binary classification task. It studies the linear relationship between independent variables and the log odds of the dependent variable. Sigmoid function that takes combination of input as independent variables and produces a probability value between 0 and 1. The sigmoid function converts the input variable into probability value ranging between 0 and 1. This enables the model to classify inputs into one of two classes. Figure 4.3.1.1 demonstrates how sigmoid function mapped continuous input data into the probabilistic space required for classification. In addition, Figure 4.3.1.2 shows how a logistic regression equation is structured, highlighting the relationship between input features, model weights, bias, and the final output probability [25]. Logistic regression is not only easy to implement and interpret as well as efficient in training but also interpretable—allowing model coefficients to be viewed as indicators of feature importance [26].

### 4.3.2 Decision Tree

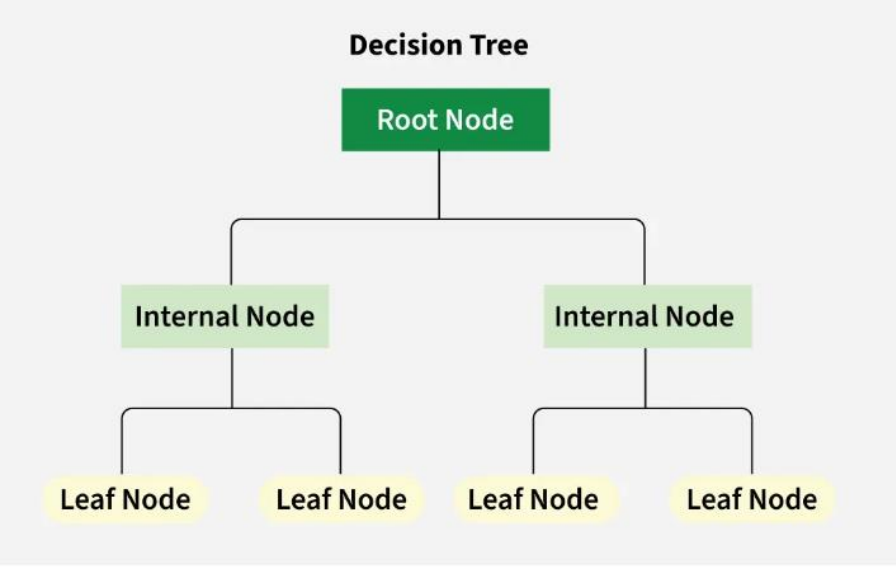



*Figure 4.3.2 Architecture of decision tree*

Decision Tree follows a hierarchical tree structure, beginning with one root node which is the starting point for decision making. From there, data is split through a sequence of conditions. Each decision node branches into further nodes, and the dataset continues to divide into smaller and more specific groups. This process breaks until further useful splits can be made or meets the predefined condition, can be referred to figure 4.3.2. There are two types of decisions tree which are classification trees used for predicting categorical outcomes prediction and

regression trees for predicting continuous variables like numerical values. One key advantage of Decision Trees is that they do not require feature scaling during the training process. In addition, it also demonstrates the ability to handle non-linear relationships, making them effective in capturing complex patterns between input features and target variables [27].

# Chapter 5

# EXPERIMENT/SIMULATION

## 5.1 Hardware Setup

The hardware involved in this project is the computer. A computer issued for the purpose of training and testing the base model in different ensembles learning method in detecting financial distress

## 5.2 Software Setup

- Programming language and environment
    - Python Version: Python 3.10 (default version)
    - Notebook environment: Google Colab

- Several Python libraries from the scikit-learn package and other standard libraries were utilized to implement the models and evaluate their performance:
    - sklearn.preprocessing:
        - StandardScaler : used to standardize the features for easier model training
    - sklearn.tree:
        - DecisionTreeClassifier: the base learner used in certain ensemble techniques.
    - sklearn.linear_model:
        - LogisticRegression: the base learner used in certain ensemble techniques.
    - sklearn.ensemble module:
        - AdaBoostClassifier: used to implement adaboosting ensemble technique.
        - StackingClassifier : used to implement stacking ensemble technique.
        - BaggingClassifier: used to implement bagging ensemble technique.
    - sklearn.metrics:
        - confusion_matrix,precision_score,recall_score,f1_score,accuracy_score:used for comprehensive evaluation of model performance.

- time module: used to record the computational time of a base learner within the ensemble learning framework
- scipy.stats:
  - stats: used to derive p value with the z score
- sklearn.inspection:
  - permutation_importance: used to derive the feature importance within an emsemble learning framework
- sklearn.model_selection：
  - GridSearchCV: used for hyperparameter tuning and model optimization
  - StratifiedKFold: perform cross validation

### 5.3 Setting and Configuration

This section outlines the configuration setup for conducting the ensemble learning experiments:

- Logistic Regression:
  - lbfgs solver
  - max_iter=200
  - class_weight = balance
  - random_state=42 was used to ensure reproducibility of results.
- Decision Tree (optimized using grid search)
  - criterion='entropy'
  - max_depth=7
  - min_samples_leaf=4
  - random_state=42 was used to ensure reproducibility of results.
- google.colab
  - drive.mount: used to mount the previous stored dataset

## 5.4 System Operation (with Screenshot)

Business Understanding

In this project, the scope of the dataset is focused on financial perspectives in terms of the financial indicators related to financial distress.

Data Understanding

```
print(df.info()) #numeric data -> in float, no categorical data
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6819 entries, 0 to 6818
Data columns (total 96 columns):
 #   Column                                                  Non-Null Count  Dtype  
---  ------                                                  --------------  -----  
 0   Bankrupt?                                               6819 non-null   int64  
 1    ROA(C) before interest and depreciation before interest  6819 non-null   float64
 2    ROA(A) before interest and % after tax                  6819 non-null   float64
 3    ROA(B) before interest and depreciation after tax       6819 non-null   float64
 4    Operating Gross Margin                                  6819 non-null   float64
 5    Realized Sales Gross Margin                             6819 non-null   float64
 6    Operating Profit Rate                                   6819 non-null   float64
 7    Pre-tax net Interest Rate                               6819 non-null   float64
 8    After-tax net Interest Rate                             6819 non-null   float64
 9    Non-industry income and expenditure/revenue             6819 non-null   float64
 10   Continuous interest rate (after tax)                    6819 non-null   float64
 11   Operating Expense Rate                                  6819 non-null   float64
 12   Research and development expense rate                   6819 non-null   float64
 13   Cash flow rate                                          6819 non-null   float64
 14   Interest-bearing debt interest rate                     6819 non-null   float64
 15   Tax rate (A)                                            6819 non-null   float64
 16   Net Value Per Share (B)                                 6819 non-null   float64
 17   Net Value Per Share (A)                                 6819 non-null   float64
 18   Net Value Per Share (C)                                 6819 non-null   float64
 19   Persistent EPS in the Last Four Seasons                 6819 non-null   float64
 20   Cash Flow Per Share                                     6819 non-null   float64
 21   Revenue Per Share (Yuan ¥)                              6819 non-null   float64
 22   Operating Profit Per Share (Yuan ¥)                     6819 non-null   float64
 23   Per Share Net profit before tax (Yuan ¥)                6819 non-null   float64
 24   Realized Sales Gross Profit Growth Rate                 6819 non-null   float64
 25   Operating Profit Growth Rate                            6819 non-null   float64
 26   After-tax Net Profit Growth Rate                        6819 non-null   float64
 27   Regular Net Profit Growth Rate                          6819 non-null   float64
 28   Continuous Net Profit Growth Rate                       6819 non-null   float64
 29   Total Asset Growth Rate                                 6819 non-null   float64
 30   Net Value Growth Rate                                   6819 non-null   float64
 31   Total Asset Return Growth Rate Ratio                    6819 non-null   float64
 32   Cash Reinvestment %                                     6819 non-null   float64
 33   Current Ratio                                           6819 non-null   float64
 34   Quick Ratio                                             6819 non-null   float64
 35   Interest Expense Ratio                                  6819 non-null   float64
```

```
print(df.info()) #numeric data -> in float, no categorical data
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6819 entries, 0 to 6818
Data columns (total 96 columns):
 #   Column                                                  Non-Null Count  Dtype
---  ------                                                  --------------  -----
 0   Bankrupt?                                               6819 non-null   int64
 1    ROA(C) before interest and depreciation before interest 6819 non-null   float64
 2    ROA(A) before interest and % after tax                  6819 non-null   float64
 3    ROA(B) before interest and depreciation after tax       6819 non-null   float64
 4    Operating Gross Margin                                  6819 non-null   float64
 5    Realized Sales Gross Margin                             6819 non-null   float64
 6    Operating Profit Rate                                   6819 non-null   float64
 7    Pre-tax net Interest Rate                               6819 non-null   float64
 8    After-tax net Interest Rate                             6819 non-null   float64
 9    Non-industry income and expenditure/revenue             6819 non-null   float64
 10   Continuous interest rate (after tax)                    6819 non-null   float64
 11   Operating Expense Rate                                  6819 non-null   float64
 12   Research and development expense rate                   6819 non-null   float64
 13   Cash flow rate                                          6819 non-null   float64
 14   Interest-bearing debt interest rate                     6819 non-null   float64
 15   Tax rate (A)                                            6819 non-null   float64
 16   Net Value Per Share (B)                                 6819 non-null   float64
 17   Net Value Per Share (A)                                 6819 non-null   float64
 18   Net Value Per Share (C)                                 6819 non-null   float64
 19   Persistent EPS in the Last Four Seasons                 6819 non-null   float64
 20   Cash Flow Per Share                                     6819 non-null   float64
 21   Revenue Per Share (Yuan ¥)                              6819 non-null   float64
 22   Operating Profit Per Share (Yuan ¥)                     6819 non-null   float64
 23   Per Share Net profit before tax (Yuan ¥)                6819 non-null   float64
 24   Realized Sales Gross Profit Growth Rate                 6819 non-null   float64
 25   Operating Profit Growth Rate                            6819 non-null   float64
 26   After-tax Net Profit Growth Rate                        6819 non-null   float64
 27   Regular Net Profit Growth Rate                          6819 non-null   float64
 28   Continuous Net Profit Growth Rate                       6819 non-null   float64
 29   Total Asset Growth Rate                                 6819 non-null   float64
 30   Net Value Growth Rate                                   6819 non-null   float64
 31   Total Asset Return Growth Rate Ratio                    6819 non-null   float64
 32   Cash Reinvestment %                                     6819 non-null   float64
 33   Current Ratio                                           6819 non-null   float64
 34   Quick Ratio                                             6819 non-null   float64
 35   Interest Expense Ratio                                  6819 non-null   float64
 72   Quick Asset Turnover Rate                               6819 non-null   float64
 73   Working capitcal Turnover Rate                          6819 non-null   float64
 74   Cash Turnover Rate                                      6819 non-null   float64
 75   Cash Flow to Sales                                      6819 non-null   float64
 76   Fixed Assets to Assets                                  6819 non-null   float64
 77   Current Liability to Liability                          6819 non-null   float64
 78   Current Liability to Equity                             6819 non-null   float64
 79   Equity to Long-term Liability                           6819 non-null   float64
 80   Cash Flow to Total Assets                               6819 non-null   float64
 81   Cash Flow to Liability                                  6819 non-null   float64
 82   CFO to Assets                                           6819 non-null   float64
 83   Cash Flow to Equity                                     6819 non-null   float64
 84   Current Liability to Current Assets                     6819 non-null   float64
 85   Liability-Assets Flag                                   6819 non-null   int64
 86   Net Income to Total Assets                              6819 non-null   float64
 87   Total assets to GNP price                               6819 non-null   float64
 88   No-credit Interval                                      6819 non-null   float64
 89   Gross Profit to Sales                                   6819 non-null   float64
 90   Net Income to Stockholder's Equity                      6819 non-null   float64
 91   Liability to Equity                                     6819 non-null   float64
 92   Degree of Financial Leverage (DFL)                      6819 non-null   float64
 93   Interest Coverage Ratio (Interest expense to EBIT)      6819 non-null   float64
 94   Net Income Flag                                         6819 non-null   int64
 95   Equity to Liability                                     6819 non-null   float64
dtypes: float64(93), int64(3)
memory usage: 5.0 MB
None
```

*Figure 5.4.1 Checking for dtype for the variables*

```
print(df[' Net Income Flag'].value_counts())
```

```
 Net Income Flag
1    6819
Name: count, dtype: int64
```

```
print(df[' Liability-Assets Flag'].value_counts())
```

```
 Liability-Assets Flag
0    6811
1       8
Name: count, dtype: int64
```

*Figure 5.4.2 Checking for data distribution on X94 and X85*

In figure 5.4.1, it shows that there are two integer data types (X94 and X85) and X0 are the target variables, further investigation on the data distribution has been done as shown in figure 5.4.2. In Figure 5.4.2, feature X94, which represents the Net Income Flag, is observed to have a constant value of 1 across all records. This lack of variability indicates that it does not contribute any influence on the prediction and can be considered a redundant variable. For X85 which denoted as Liability-Assets Flag, there are two classes, 0 and 1, which may consider as variables that have significant to the target variable.

Data Preparation

Remove redundant features

```
[ ] df[' Net Income Flag'].value_counts() #remove redundant value
```

| Net Income Flag | count |
|---|---|
| 1 | 6819 |

dtype: int64

```
[ ] X = df.drop(columns=['Bankrupt?',' Net Income Flag'])
    y = df['Bankrupt?']
```

*Figure 5.4.3 Remove redundant features*

In figure 5.4.3, it has shown that there is a constant feature, “Net Income Flag”. Since it is not significant to the target variable, hence it is dropped.

Balance the datasets

```
print(df['Bankrupt?'].value_counts()) #imbalance data
```

```
Bankrupt?
0    6599
1     220
Name: count, dtype: int64
```

*Figure 5.4.4 Imbalance datasets*

```
from imblearn.combine import SMOTEENN
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import EditedNearestNeighbours
smote_enn = SMOTEENN(sampling_strategy='auto',
                     smote=SMOTE(random_state=42),  # Fix random_state for SMOTE
                     enn=EditedNearestNeighbours(),  # Default settings for ENN
                     random_state=42)  # Fix random_state for SMOTEENN

X_resampled, y_resampled = smote_enn.fit_resample(X, y)
y_resampled.value_counts()
```

| Bankrupt? | count |
|---|---|
| 0 | 6599 |
| 1 | 6258 |

dtype: int64

*Figure 5.4.5 Balance the datasets*

In figure 5.4.4, it shows that there is an imbalance distribution between 0 and 1, hence SMOTE-ENN to balance the dataset. In figure 5.4.5, it shown the process of balancing the datasets through SMOTE-ENN, and the latest dataset contains 6599 companies with normal status and 6258 companies with distress status.

Univariate Feature Selection

```python
from scipy.stats import ttest_ind

# Initialize a list to store p-values
p_values = []

# Loop through each feature to apply T-test
for feature in X_resampled.columns:
    # Split data into two groups based on the target
    group_0 = X_resampled[y_resampled == 0][feature]
    group_1 = X_resampled[y_resampled == 1][feature]

    # Perform T-test between the two groups
    _, p_value = ttest_ind(group_0, group_1)

    # Append the p-value for each feature
    p_values.append(p_value)

# Convert p-values into a DataFrame for better viewing
p_values_df = pd.DataFrame({
    'Feature': X_resampled.columns,
    'P-Value': p_values
}).sort_values(by='P-Value', ascending=True)

print(p_values_df)

# Select features with p-value less than the significance level (e.g., 0.05)
selected_features = p_values_df[p_values_df['P-Value'] < 0.05]
print("Selected Features based on T-test (p-value < 0.05):")
print(selected_features)
```

*Figure 5.4.6 Feature selection*

Table 5.4.1: Features that are selected by t-test

| | Feature | P-Value |
|---|---|---|
| X0 | ROAI before interest and depreciation before interest | 0 |
| X41 | Operating profit/Paid-in capital | 0 |
| X22 | Per Share Net profit before tax (Yuan Â¥) | 0 |
| X21 | Operating Profit Per Share (Yuan Â¥) | 0 |
| X42 | Net profit before tax/Paid-in capital | 0 |
| X18 | Persistent EPS in the Last Four Seasons | 0 |
| X17 | Net Value Per Share I | 0 |
| X16 | Net Value Per Share (A) | 0 |
| X15 | Net Value Per Share (B) | 0 |
| X14 | Tax rate (A) | 0 |
| X53 | Working Capital to Total Assets | 0 |
| X37 | Net worth/Assets | 0 |
| X56 | Cash/Total Assets | 0 |

| | | |
|---|---|---|
| X67 | Retained Earnings to Total Assets | 0 |
| X81 | CFO to Assets | 0 |
| X83 | Current Liability to Current Assets | 0 |
| X85 | Net Income to Total Assets | 0 |
| X88 | Gross Profit to Sales | 0 |
| X4 | Realized Sales Gross Margin | 0 |
| X3 | Operating Gross Margin | 0 |
| X2 | ROA(B) before interest and depreciation after tax | 0 |
| X1 | ROA(A) before interest and % after tax | 0 |
| X59 | Current Liability to Assets | 0 |
| X36 | Debt ratio % | 0 |
| X51 | Operating profit per person | 2.39E-285 |
| X69 | Total expense/Assets | 6.49E-217 |
| X93 | Equity to Liability | 1.69E-209 |
| X60 | Operating Funds to Liability | 1.09E-188 |
| X54 | Quick Assets/Total Assets | 1.28E-187 |
| X12 | Cash flow rate | 1.66E-186 |
| X19 | Cash Flow Per Share | 2.46E-186 |
| X79 | Cash Flow to Total Assets | 1.30E-134 |
| X44 | Total Asset Turnover | 1.71E-122 |
| X39 | Borrowing dependency | 4.03E-121 |
| X90 | Liability to Equity | 2.35E-97 |
| X89 | Net Income to Stockholder's Equity | 4.42E-87 |
| X65 | Current Liabilities/Equity | 1.41E-82 |
| X77 | Current Liability to Equity | 1.41E-82 |
| X78 | Equity to Long-term Liability | 1.89E-77 |
| X48 | Fixed Assets Turnover Frequency | 1.81E-73 |
| X64 | Working Capital/Equity | 1.68E-69 |
| X43 | Inventory and accounts receivable/Net value | 7.23E-57 |
| X28 | Total Asset Growth Rate | 1.58E-55 |
| X80 | Cash Flow to Liability | 8.75E-47 |
| X58 | Cash/Current Liability | 3.96E-41 |

| | | |
|---|---|---|
| X11 | Research and development expense rate | 9.00E-38 |
| X25 | After-tax Net Profit Growth Rate | 2.88E-37 |
| X26 | Regular Net Profit Growth Rate | 4.66E-36 |
| X55 | Current Assets/Total Assets | 2.38E-35 |
| X82 | Cash Flow to Equity | 2.61E-35 |
| X73 | Cash Turnover Rate | 1.09E-23 |
| X13 | Interest-bearing debt interest rate | 1.29E-22 |
| X31 | Cash Reinvestment % | 1.44E-21 |
| X71 | Quick Asset Turnover Rate | 1.06E-16 |
| X30 | Total Asset Return Growth Rate Ratio | 2.80E-13 |
| X49 | Net Worth Turnover Rate (times) | 5.17E-12 |
| X40 | Contingent liabilities/Net worth | 1.05E-11 |
| X24 | Operating Profit Growth Rate | 3.07E-09 |
| X8 | Non-industry income and expenditure/revenue | 1.77E-08 |
| X76 | Current Liability to Liability | 1.35E-06 |
| X63 | Current Liabilities/Liability | 1.35E-06 |
| X75 | Fixed Assets to Assets | 3.85E-06 |
| X86 | Total assets to GNP price | 6.20E-06 |
| X50 | Revenue per person | 4.24E-05 |
| X47 | Inventory Turnover Rate (times) | 4.32E-05 |
| X70 | Current Asset Turnover Rate | 8.03E-05 |
| X29 | Net Value Growth Rate | 0.00013445 |
| X10 | Operating Expense Rate | 0.000151196 |
| X27 | Continuous Net Profit Growth Rate | 0.0003026 |
| X68 | Total income/Total expense | 0.001690591 |
| X6 | Pre-tax net Interest Rate | 0.002332457 |
| X9 | Continuous interest rate (after tax) | 0.002671253 |
| X46 | Average Collection Days | 0.002845449 |
| X7 | After-tax net Interest Rate | 0.003861377 |
| X66 | Long-term Liability to Current Assets | 0.004469414 |
| X45 | Accounts Receivable Turnover | 0.004822885 |
| X33 | Quick Ratio | 0.005165048 |

| | | |
|---|---|---|
| X91 | Degree of Financial Leverage (DFL) | 0.012887153 |
| X35 | Total debt/Total net worth | 0.015892463 |
| X20 | Revenue Per Share (Yuan Â¥) | 0.038817475 |
| X87 | No-credit Interval | 0.043919515 |
| X34 | Interest Expense Ratio | 0.048736981 |

In figure 5.4.6, it has shown the process of deriving the significant features by performing t – test. In table 5.4.1 has displayed 82 features that have been selected, with a threshold of p values must be less than 0.05.

## Modeling

## Bagging

```
#Bagging
from sklearn.model_selection import StratifiedKFold

accuracies, precisions, recalls, f1_scores, aucs = [], [], [], [], []
type_I_errors, type_II_errors = [],[]

skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

start_time = time.time() # start time

decision_tree = DecisionTreeClassifier(criterion = "entropy",max_depth = 7,min_samples_split = 2,min_samples_leaf=4,random_state = 42)
model = BaggingClassifier(estimator=decision_tree, n_estimators=10, n_jobs=-1,random_state=42)

for i in range(1, 301):
    fold_accuracies, fold_precisions, fold_recalls, fold_f1s, fold_aucs = [], [], [], [], []
    fold_type_I_errors, fold_type_II_errors = [], []

    for train_index, val_index in skf.split(X_train, y_train):
        X_train_fold, X_val_fold = X_train.iloc[train_index], X_train.iloc[val_index]
        y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.iloc[val_index]

        scaler = StandardScaler()
        X_train_scaled = scaler.fit_transform(X_train_fold)
        X_val_scaled = scaler.transform(X_val_fold)

        model.fit(X_train_scaled, y_train_fold)
        y_val_pred = model.predict(X_val_scaled)
        y_val_prob = model.predict_proba(X_val_scaled)[:, 1]

        accuracy = accuracy_score(y_val_fold, y_val_pred)
        precision = precision_score(y_val_fold, y_val_pred, pos_label=1)
        recall = recall_score(y_val_fold, y_val_pred, pos_label=1)
        f1 = f1_score(y_val_fold, y_val_pred, pos_label=1)
        auc = metrics.roc_auc_score(y_val_fold, y_val_prob)

        cm = confusion_matrix(y_val_fold, y_val_pred)
        FP, TN, FN, TP = cm[0][1], cm[0][0], cm[1][0], cm[1][1]
        type_I_error = FP / (FP + TN) if (FP + TN) != 0 else 0
        type_II_error = FN / (FN + TP) if (FN + TP) != 0 else 0

        # Append fold metrics
        fold_accuracies.append(accuracy)
        fold_precisions.append(precision)
        fold_recalls.append(recall)
        fold_f1s.append(f1)
        fold_aucs.append(auc)
        fold_type_I_errors.append(type_I_error)
        fold_type_II_errors.append(type_II_error)

    avg_acc = sum(fold_accuracies) / len(fold_accuracies)
    avg_precision = sum(fold_precisions) / len(fold_precisions)
    avg_recall = sum(fold_recalls) / len(fold_recalls)
    avg_f1 = sum(fold_f1s) / len(fold_f1s)
    avg_aucs = sum(fold_aucs) / len(fold_aucs)
    avg_typeI = sum(fold_type_I_errors) / len(fold_type_I_errors)
    avg_typeII = sum(fold_type_II_errors) / len(fold_type_II_errors)
```

```
        print(">>Fold",i)
        print("AUC:",avg_acc)
        print("Recall:",avg_recall)
        print("Type II:",avg_typeII)

        # Average over 5 folds
        accuracies.append(round(avg_acc, 4))
        precisions.append(round(avg_precision, 4))
        recalls.append(round(avg_recall, 4))
        f1_scores.append(round(avg_f1, 4))
        aucs.append(round(avg_aucs, 4))
        type_I_errors.append(round(avg_typeI, 4))
        type_II_errors.append(round(avg_typeII, 4))


        # Early stopping logic
        if len(accuracies) >= 5 and len(set(accuracies[-5:])) == 1:
            print(f"Early stopping at iteration {i} because accuracy hasn't changed for 5 iterations.")
            break
        elif len(recalls) >= 5 and len(set(recalls[-5:])) == 1:
            print(f"Early stopping at iteration {i} because recall hasn't changed for 5 iterations.")
            break

end_time = time.time()
duration = round(end_time - start_time, 2)
```

```
>>Fold 1
AUC: 0.9453277129269347
Recall: 0.973689710466101
Type II: 0.026310289533899068
>>Fold 2
AUC: 0.9453277129269347
Recall: 0.973689710466101
Type II: 0.026310289533899068
>>Fold 3
AUC: 0.9453277129269347
Recall: 0.973689710466101
Type II: 0.026310289533899068
>>Fold 4
AUC: 0.9453277129269347
Recall: 0.973689710466101
Type II: 0.026310289533899068
>>Fold 5
AUC: 0.9453277129269347
Recall: 0.973689710466101
Type II: 0.026310289533899068
Early stopping at iteration 5 because accuracy hasn't changed for 5 iterations.
```

*Figure 5.4.7.1(a) Modeling in bagging ensemble learning environment (decision tree)*

```
print(f"Final Accuracy: {accuracies[-1]}")
print(f"Final Precision: {precisions[-1]}")
print(f"Final Recall: {recalls[-1]}")
print(f"Final F1 Score: {f1_scores[-1]}")
print(f"Final AUC: {aucs[-1]}")
print(f"Final Type I Error: {type_I_errors[-1]}")
print(f"Final Type II Error: {type_II_errors[-1]}")
print(f"Total training time: {duration} seconds.")
```

```
Final Accuracy: 0.9453
Final Precision: 0.9195
Final Recall: 0.9737
Final F1 Score: 0.9458
Final AUC: 0.986
Final Type I Error: 0.0819
Final Type II Error: 0.0263
Total training time: 89.46 seconds.
```

*Figure 5.4.7.1(b) Output from modeling in bagging ensemble learning environment (decision tree)*

```
# Make sure test data is scaled with the same scaler used on training data
X_test_scaled = scaler.transform(X_test)

# Predict once
y_test_pred = model.predict(X_test_scaled)
y_test_prob = model.predict_proba(X_test_scaled)[:, 1]

# Evaluate
accuracy = accuracy_score(y_test, y_test_pred)
precision = precision_score(y_test, y_test_pred, pos_label=1)
recall = recall_score(y_test, y_test_pred, pos_label=1)
f1 = f1_score(y_test, y_test_pred, pos_label=1)
auc = metrics.roc_auc_score(y_test, y_test_prob)

cm = confusion_matrix(y_test, y_test_pred)
FP, TN, FN, TP = cm[0][1], cm[0][0], cm[1][0], cm[1][1]
type_I_error = FP / (FP + TN) if (FP + TN) != 0 else 0
type_II_error = FN / (FN + TP) if (FN + TP) != 0 else 0

# Print final test performance
print(f"Final Accuracy: {accuracy:.4f}")
print(f"Final Precision: {precision:.4f}")
print(f"Final Recall: {recall:.4f}")
print(f"Final F1 Score: {f1:.4f}")
print(f"Final AUC: {auc:.4f}")
print(f"Final Type I Error: {type_I_error:.4f}")
print(f"Final Type II Error: {type_II_error:.4f}")
```

```
Final Accuracy: 0.9463
Final Precision: 0.9193
Final Recall: 0.9735
Final F1 Score: 0.9456
Final AUC: 0.9876
Final Type I Error: 0.0786
Final Type II Error: 0.0265
```

*Figure 5.4.7.1(c) Output from testing in bagging ensemble learning environment (decision tree)*

In figure 5.4.7.1(a) has shown the implementation of base learner, Decision Tree within the bagging ensemble learning framework. Initially, 5-fold cross-validation is applied, where the dataset is scaled and fitted to the model for each fold. The performance result from each fold is stored in a list, and the average performance across all folds is then computed to evaluate overall effectiveness. During the training process, a loop is employed to determine early convergence — this is defined either by the model producing consistent results for five consecutive iterations or reaching a maximum of 300 iterations. Additionally, the computational time required for model training is recorded for further evaluation. Figure 5.4.7.1(b) showed the outcome obtained from the model training. In the outcome, it displayed the validation result including the accuracy, precision, recall, Fl score, auc score, type I error and type II error and the training time. Lastly the model trained is used for predicting on the test data and result as shown in Figure 5.4.7.1(c).

```
from sklearn.inspection import permutation_importance
import scipy.stats as stats

# Compute permutation importance
perm_importance = permutation_importance(model, X_test_scaled, y_test, n_repeats=10, random_state=42)

# Compute p-values using a permutation test
p_values = np.array([
    stats.norm.sf(abs(mean) / std) * 2 if std > 1e-10 else 1.0  # Normalized by standard deviation
    for mean, std in zip(perm_importance.importances_mean, perm_importance.importances_std)
])

importance_df = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': perm_importance.importances_mean,
    'P-Value': p_values
})

# Filter significant features (p-value < 0.05)
significant_features = importance_df[importance_df['P-Value'] < 0.05]
print(len(significant_features))
```

```
26
```

```
significant_features.to_excel("DT_Bagging_significant_features.xlsx", index=False)

print(f"Significant features saved to 'DT_Bagging_significant_features.xlsx'.")
```

```
Significant features saved to 'DT_Bagging_significant_features.xlsx'.
```

*Figure 5.4.7.1(d) Output from extracting significant feature via permutation importance and z-score*

Figure 5.4.7.1(d) displays the output from extracting significant features using permutation importance and z-score analysis. Features with a p-value below the threshold of 0.05 are considered statistically significant and are selected for further analysis. The selected significant features are then recorded and saved into an Excel (.xlsx) file for subsequent model training.

## AdaBoosting

```python
#AdaBoosting

from sklearn.model_selection import StratifiedKFold

accuracies, precisions, recalls, f1_scores, aucs = [], [], [], [], []
type_I_errors, type_II_errors = [],[]

skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

start_time = time.time() # start time

decision_tree = DecisionTreeClassifier(criterion = "entropy", max_depth = 7,min_samples_split = 2,min_samples_leaf=4,random_state = 42)
model = AdaBoostClassifier(estimator=decision_tree,n_estimators=10,learning_rate=0.5)

for i in range(1, 301):

    fold_accuracies, fold_precisions, fold_recalls, fold_f1s, fold_aucs = [], [], [], [], []
    fold_type_I_errors, fold_type_II_errors = [], []

    for train_index, val_index in skf.split(X_train, y_train):
        X_train_fold, X_val_fold = X_train.iloc[train_index], X_train.iloc[val_index]
        y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.iloc[val_index]

        scaler = StandardScaler()
        X_train_scaled = scaler.fit_transform(X_train_fold)
        X_val_scaled = scaler.transform(X_val_fold)

        model.fit(X_train_scaled, y_train_fold)
        y_val_pred = model.predict(X_val_scaled)
        y_val_prob = model.predict_proba(X_val_scaled)[:, 1]

        accuracy = accuracy_score(y_val_fold, y_val_pred)
        precision = precision_score(y_val_fold, y_val_pred, pos_label=1)
        recall = recall_score(y_val_fold, y_val_pred, pos_label=1)
        f1 = f1_score(y_val_fold, y_val_pred, pos_label=1)
        auc = metrics.roc_auc_score(y_val_fold, y_val_prob)

        cm = confusion_matrix(y_val_fold, y_val_pred)
        FP, TN, FN, TP = cm[0][1], cm[0][0], cm[1][0], cm[1][1]
        type_I_error = FP / (FP + TN) if (FP + TN) != 0 else 0
        type_II_error = FN / (FN + TP) if (FN + TP) != 0 else 0

        # Append fold metrics
        fold_accuracies.append(accuracy)
        fold_precisions.append(precision)
        fold_recalls.append(recall)
        fold_f1s.append(f1)
        fold_aucs.append(auc)
        fold_type_I_errors.append(type_I_error)
        fold_type_II_errors.append(type_II_error)

    avg_acc = sum(fold_accuracies) / len(fold_accuracies)
    avg_precision = sum(fold_precisions) / len(fold_precisions)
    avg_recall = sum(fold_recalls) / len(fold_recalls)
    avg_f1 = sum(fold_f1s) / len(fold_f1s)
    avg_aucs = sum(fold_aucs) / len(fold_aucs)
    avg_typeI = sum(fold_type_I_errors) / len(fold_type_I_errors)
    avg_typeII = sum(fold_type_II_errors) / len(fold_type_II_errors)
```

```python
    print(">>Fold",i)
    print("AUC:",avg_acc)
    print("Recall:",avg_recall)
    print("Type II:",avg_typeII)

    # Average over 5 folds
    accuracies.append(round(avg_acc, 4))
    precisions.append(round(avg_precision, 4))
    recalls.append(round(avg_recall, 4))
    f1_scores.append(round(avg_f1, 4))
    aucs.append(round(avg_aucs, 4))
    type_I_errors.append(round(avg_typeI, 4))
    type_II_errors.append(round(avg_typeII, 4))


    # Early stopping logic
    if len(accuracies) >= 5 and len(set(accuracies[-5:])) == 1:
        print(f"Early stopping at iteration {i} because accuracy hasn't changed for 5 iterations.")
        break
    elif len(recalls) >= 5 and len(set(recalls[-5:])) == 1:
        print(f"Early stopping at iteration {i} because recall hasn't changed for 5 iterations.")
        break

end_time = time.time()
duration = round(end_time - start_time, 2)
```

```
Recall: 0.9773191153116562
Type II: 0.022680884688343745
>>Fold 287
AUC: 0.9626628373787908
Recall: 0.9807212480149078
Type II: 0.01927875198509218
>>Fold 288
AUC: 0.9646635785312828
Recall: 0.9811750201404814
Type II: 0.018824979859518535
>>Fold 289
AUC: 0.9625511704033105
Recall: 0.9793586447064637
Type II: 0.020641355293536258
>>Fold 290
AUC: 0.9617740102526096
Recall: 0.9770926153283863
Type II: 0.02290738467161363
>>Fold 291
AUC: 0.9627741955407325
Recall: 0.9786799169156881
Type II: 0.021320083084312044
>>Fold 292
AUC: 0.9631072818232351
Recall: 0.983897138121234
Type II: 0.016102861878765883
>>Fold 293
AUC: 0.9601062936199124
Recall: 0.9727801071241966
Type II: 0.027219892875803363
>>Fold 294
AUC: 0.9634405533938608
Recall: 0.9791329168822278
Type II: 0.02086708311777227
>>Fold 295
AUC: 0.9625525291828794
Recall: 0.9807225349466309
Type II: 0.01927746505336906
>>Fold 296
AUC: 0.9614408004446915
Recall: 0.9798152480818283
Type II: 0.020184751918171734
>>Fold 297
AUC: 0.9609962324748317
Recall: 0.9750510268428219
Type II: 0.02494897315717812
>>Fold 298
AUC: 0.9627740720153172
Recall: 0.9823085496022094
Type II: 0.017691450397790597
>>Fold 299
AUC: 0.9597736396763634
Recall: 0.9755060859001187
Type II: 0.02449391409988135
>>Fold 300
AUC: 0.9617734543882402
Recall: 0.9800394315879967
Type II: 0.019960568412003468
```

*Figure 5.4.7.2(a) Modeling in adaboosting ensemble learning environment (decision tree)*

```
print(f"Final Accuracy: {accuracies[-1]}")
print(f"Final Precision: {precisions[-1]}")
print(f"Final Recall: {recalls[-1]}")
print(f"Final F1 Score: {f1_scores[-1]}")
print(f"Final AUC: {aucs[-1]}")
print(f"Final Type I Error: {type_I_errors[-1]}")
print(f"Final Type II Error: {type_II_errors[-1]}")
print(f"Total training time: {duration} seconds.")
```

```
Final Accuracy: 0.9618
Final Precision: 0.9441
Final Recall: 0.98
Final F1 Score: 0.9617
Final AUC: 0.9921
Final Type I Error: 0.0558
Final Type II Error: 0.02
Total training time: 9713.48 seconds.
```

*Figure 5.4.7.2(b) Output from modeling in adaboosting ensemble learning environment (decision tree)*

```python
# Make sure test data is scaled with the same scaler used on training data
X_test_scaled = scaler.transform(X_test)

# Predict once
y_test_pred = model.predict(X_test_scaled)
y_test_prob = model.predict_proba(X_test_scaled)[:, 1]

# Evaluate
accuracy = accuracy_score(y_test, y_test_pred)
precision = precision_score(y_test, y_test_pred, pos_label=1)
recall = recall_score(y_test, y_test_pred, pos_label=1)
f1 = f1_score(y_test, y_test_pred, pos_label=1)
auc = metrics.roc_auc_score(y_test, y_test_prob)

cm = confusion_matrix(y_test, y_test_pred)
FP, TN, FN, TP = cm[0][1], cm[0][0], cm[1][0], cm[1][1]
type_I_error = FP / (FP + TN) if (FP + TN) != 0 else 0
type_II_error = FN / (FN + TP) if (FN + TP) != 0 else 0

# Print final test performance
print(f"Final Accuracy: {accuracy:.4f}")
print(f"Final Precision: {precision:.4f}")
print(f"Final Recall: {recall:.4f}")
print(f"Final F1 Score: {f1:.4f}")
print(f"Final AUC: {auc:.4f}")
print(f"Final Type I Error: {type_I_error:.4f}")
print(f"Final Type II Error: {type_II_error:.4f}")
```

```
Test Accuracy: 0.9640
Test Precision: 0.9435
Test Recall: 0.9838
Test F1 Score: 0.9632
Test AUC: 0.9918
Test Type I Error: 0.0543
Test Type II Error: 0.0162
```

*Figure 5.4.7.2(c) Output from testing in adaboosting ensemble learning environment (decision tree)*

In figure 5.4.7.1(a) has shown the implementation of base learner, Decision Tree within the adaboosting ensemble learning framework. Initially, 5-fold cross-validation is applied, where the dataset is scaled and fitted to the model for each fold. The performance result from each fold is stored in a list, and the average performance across all folds is then computed to evaluate overall effectiveness. During the training process, a loop is employed to determine early convergence — this is defined either by the model producing consistent results for five consecutive iterations or reaching a maximum of 300 iterations. Additionally, the computational time required for model training is recorded for further evaluation. Figure 5.4.7.2(b) showed the outcome obtained from the model training. In the outcome, it displayed the validation result including the accuracy, precision, recall, Fl score, auc score, type I error

and type II error and the training time. Lastly the model trained is used for predicting on the test data and result as shown in Figure 5.4.7.2(c).

```
from sklearn.inspection import permutation_importance
import scipy.stats as stats

# Compute permutation importance
perm_importance = permutation_importance(model, X_val_scaled, y_val_fold, n_repeats=10, random_state=42)

# Compute p-values using a permutation test
p_values = np.array([
    stats.norm.sf(abs(mean) / std) * 2 if std > 1e-10 else 1.0  # Normalized by standard deviation
    for mean, std in zip(perm_importance.importances_mean, perm_importance.importances_std)
])

importance_df = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': perm_importance.importances_mean,
    'P-Value': p_values
})

# Filter significant features (p-value < 0.05)
significant_features = importance_df[importance_df['P-Value'] < 0.05]
print(len(significant_features))
```

```
24
```

```
[ ] significant_features.to_excel("DT_Adaboosting_significant_features.xlsx", index=False)

    print(f"Significant features saved to 'DT_Adaboosting_significant_features.xlsx'.")
```

```
Significant features saved to 'DT_Adaboosting_significant_features.xlsx'.
```

*Figure 5.4.7.2(d)Output from extracting significant feature via permutation importance and z-score*

Figure 5.4.7.2(d) displays the output from extracting significant features using permutation importance and z-score analysis. Features with a p-value below the threshold of 0.05 are considered statistically significant and are selected for further analysis. The selected significant features are then recorded and saved into an Excel (xlsx) file for subsequent model training.

## Stacking

```
# Stacking
from sklearn.model_selection import StratifiedKFold

accuracies, precisions, recalls, f1_scores, aucs = [], [], [], [], []
type_I_errors, type_II_errors = [],[]

skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

start_time = time.time() # start time
base_estimators =[ (f'DecisionTree{i}',DecisionTreeClassifier(criterion = "entropy", max_depth = 7,min_samples_split = 2,min_samples_leaf=4,random_state = 42)) for i in range (10)]
model = StackingClassifier(estimators=base_estimators,final_estimator=LogisticRegression())


for i in range(1, 301):
    fold_accuracies, fold_precisions, fold_recalls, fold_f1s, fold_aucs = [], [], [], [], []
    fold_type_I_errors, fold_type_II_errors = [], []

    for train_index, val_index in skf.split(X_train, y_train):
        X_train_fold, X_val_fold = X_train.iloc[train_index], X_train.iloc[val_index]
        y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.iloc[val_index]

        scaler = StandardScaler()
        X_train_scaled = scaler.fit_transform(X_train_fold)
        X_val_scaled = scaler.transform(X_val_fold)

        model.fit(X_train_scaled, y_train_fold)
        y_val_pred = model.predict(X_val_scaled)
        y_val_prob = model.predict_proba(X_val_scaled)[:, 1]

        accuracy = accuracy_score(y_val_fold, y_val_pred)
        precision = precision_score(y_val_fold, y_val_pred, pos_label=1)
        recall = recall_score(y_val_fold, y_val_pred, pos_label=1)
        f1 = f1_score(y_val_fold, y_val_pred, pos_label=1)
        auc = metrics.roc_auc_score(y_val_fold, y_val_prob)

        cm = confusion_matrix(y_val_fold, y_val_pred)
        FP, TN, FN, TP = cm[0][1], cm[0][0], cm[1][0], cm[1][1]
        type_I_error = FP / (FP + TN) if (FP + TN) != 0 else 0
        type_II_error = FN / (FN + TP) if (FN + TP) != 0 else 0

        # Append fold metrics
        fold_accuracies.append(accuracy)
        fold_precisions.append(precision)
        fold_recalls.append(recall)
        fold_f1s.append(f1)
        fold_aucs.append(auc)
        fold_type_I_errors.append(type_I_error)
        fold_type_II_errors.append(type_II_error)

    avg_acc = sum(fold_accuracies) / len(fold_accuracies)
    avg_precision = sum(fold_precisions) / len(fold_precisions)
    avg_recall = sum(fold_recalls) / len(fold_recalls)
    avg_f1 = sum(fold_f1s) / len(fold_f1s)
    avg_aucs = sum(fold_aucs) / len(fold_aucs)
    avg_typeI = sum(fold_type_I_errors) / len(fold_type_I_errors)
    avg_typeII = sum(fold_type_II_errors) / len(fold_type_II_errors)
```

```
        print(">>Fold",i)
        print("AUC:",avg_acc)
        print("Recall:",avg_recall)
        print("Type II:",avg_typeII)

        # Average over 5 folds
        accuracies.append(round(avg_acc, 4))
        precisions.append(round(avg_precision, 4))
        recalls.append(round(avg_recall, 4))
        f1_scores.append(round(avg_f1, 4))
        aucs.append(round(avg_aucs, 4))
        type_I_errors.append(round(avg_typeI, 4))
        type_II_errors.append(round(avg_typeII, 4))


        # Early stopping logic
        if len(accuracies) >= 5 and len(set(accuracies[-5:])) == 1:
            print(f"Early stopping at iteration {i} because accuracy hasn't changed for 5 iterations.")
            break
        elif len(recalls) >= 5 and len(set(recalls[-5:])) == 1:
            print(f"Early stopping at iteration {i} because recall hasn't changed for 5 iterations.")
            break

    end_time = time.time()
    duration = round(end_time - start_time, 2)
```

```
>>Fold 1
AUC: 0.9167682663207956
Recall: 0.9319547720715224
Type II: 0.06804522792847748
>>Fold 2
AUC: 0.9167682663207956
Recall: 0.9319547720715224
Type II: 0.06804522792847748
>>Fold 3
AUC: 0.9167682663207956
Recall: 0.9319547720715224
Type II: 0.06804522792847748
>>Fold 4
AUC: 0.9167682663207956
Recall: 0.9319547720715224
Type II: 0.06804522792847748
>>Fold 5
AUC: 0.9167682663207956
Recall: 0.9319547720715224
Type II: 0.06804522792847748
Early stopping at iteration 5 because accuracy hasn't changed for 5 iterations.
```

*Figure 5.4.7.3(a) Modeling in stacking ensemble learning environment (decision tree)*

```
print(f"Final Accuracy: {accuracies[-1]}")
print(f"Final Precision: {precisions[-1]}")
print(f"Final Recall: {recalls[-1]}")
print(f"Final F1 Score: {f1_scores[-1]}")
print(f"Final AUC: {aucs[-1]}")
print(f"Final Type I Error: {type_I_errors[-1]}")
print(f"Final Type II Error: {type_II_errors[-1]}")
print(f"Total training time: {duration} seconds.")
```

```
Final Accuracy: 0.9168
Final Precision: 0.9016
Final Recall: 0.932
Final F1 Score: 0.9164
Final AUC: 0.9578
Final Type I Error: 0.0978
Final Type II Error: 0.068
Total training time: 783.41 seconds.
```

*Figure 5.4.7.3(b) Output from modeling in adaboosting ensemble learning environment (decision tree)*

```
# Make sure test data is scaled with the same scaler used on training data
X_test_scaled = scaler.transform(X_test)

# Predict once
y_test_pred = model.predict(X_test_scaled)
y_test_prob = model.predict_proba(X_test_scaled)[:, 1]

# Evaluate
accuracy = accuracy_score(y_test, y_test_pred)
precision = precision_score(y_test, y_test_pred, pos_label=1)
recall = recall_score(y_test, y_test_pred, pos_label=1)
f1 = f1_score(y_test, y_test_pred, pos_label=1)
auc = metrics.roc_auc_score(y_test, y_test_prob)

cm = confusion_matrix(y_test, y_test_pred)
FP, TN, FN, TP = cm[0][1], cm[0][0], cm[1][0], cm[1][1]
type_I_error = FP / (FP + TN) if (FP + TN) != 0 else 0
type_II_error = FN / (FN + TP) if (FN + TP) != 0 else 0

# Print final test performance
print(f"Test Accuracy: {accuracy:.4f}")
print(f"Test Precision: {precision:.4f}")
print(f"Test Recall: {recall:.4f}")
print(f"Test F1 Score: {f1:.4f}")
print(f"Test AUC: {auc:.4f}")
print(f"Test Type I Error: {type_I_error:.4f}")
print(f"Test Type II Error: {type_II_error:.4f}")
```

```
Test Accuracy: 0.9147
Test Precision: 0.8946
Test Recall: 0.9319
Test F1 Score: 0.9128
Test AUC: 0.9671
Test Type I Error: 0.1010
Test Type II Error: 0.0681
```

*Figure 5.4.7.3I Output from testing in adaboosting ensemble learning environment (decision tree)*

In figure 5.4.7.3(a) has shown the implementation of base learner, Decision Tree within the adaboosting ensemble learning framework. Initially, 5-fold cross-validation is applied, where the dataset is scaled and fitted to the model for each fold. The performance result from each fold is stored in a list, and the average performance across all folds is then computed to evaluate overall effectiveness. During the training process, a loop is employed to determine early convergence — this is defined either by the model producing consistent results for five consecutive iterations or reaching a maximum of 300 iterations. Additionally, the computational time required for model training is recorded for further evaluation. Figure 5.4.9.2 showed the outcome obtained from the model training. In the outcome, it displayed the validation result including the accuracy, precision, recall, Fl score, auc score, type I error and type II error and the training time. Lastly the model trained is used for predicting on the test data and result as shown in Figure 5.4.7.3(c).

```
from sklearn.inspection import permutation_importance
import scipy.stats as stats

# Compute permutation importance
perm_importance = permutation_importance(model,X_test_scaled, y_test, n_repeats=10, random_state=42)

# Compute p-values using a permutation test
p_values = np.array([
    stats.norm.sf(abs(mean) / std) * 2 if std > 1e-10 else 1.0  # Normalized by standard deviation
    for mean, std in zip(perm_importance.importances_mean, perm_importance.importances_std)
])

importance_df = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': perm_importance.importances_mean,
    'P-Value': p_values
})

# Filter significant features (p-value < 0.05)
significant_features = importance_df[importance_df['P-Value'] < 0.05]
print(len(significant_features))
```

```
29
```

```
significant_features.to_excel("DT_Stacking_significant_features.xlsx", index=False)

print(f"Significant features saved to 'DT_Stacking_significant_features.xlsx'.")
```

```
Significant features saved to 'DT_Stacking_significant_features.xlsx'.
```

*Figure 5.4.7.3(d) Output from extracting significant feature via permutation importance and z-score*

Figure 5.4.7.3(d) displays the output from extracting significant features using permutation importance and z-score analysis. Features with a p-value below the threshold of 0.05 are considered statistically significant and are selected for further analysis. The selected significant features are then recorded and saved into an Excel (xlsx) file for subsequent model training.

```
Ada_Boosting_df = pd.read_excel("/content/DT_Adaboosting_significant_features.xlsx")
Stacking_df = pd.read_excel("/content/DT_Stacking_significant_features.xlsx")
Bagging_df = pd.read_excel("/content/DT_Bagging_significant_features.xlsx")


top_features_adaBoosting = Ada_Boosting_df['Feature'].tolist()
top_features_stacking = Stacking_df['Feature'].tolist()
top_features_bagging = Bagging_df['Feature'].tolist()


exact_matches = set(top_features_adaBoosting) & set(top_features_stacking) & set(top_features_bagging)
num_exact_matches = len(exact_matches)

# Display the matched features
print(f"Number of exact matches: {num_exact_matches}")
print("Matched features:")
print(list(exact_matches))
```

```
Number of exact matches: 10
Matched features:
[' Contingent liabilities/Net worth', ' Net Income to Total Assets', ' Retained Earnings to Total Assets', ' Borrow
```

```
# write the exact matches into a txt file
exact_matches_list = list(exact_matches)
with open("duplicates_decision_tree.txt", "w") as file:
    for item in exact_matches_list:
        file.write(item + "\n")
```

*Figure 5.4.8.1(a) Extracting overlapping significant feature*

```
dt_combined_features = list(set(top_features_adaBoosting + top_features_stacking + top_features_bagging))
print(len(dt_combined_features))
```

```
46
```

```
with open('combine_decision_tree.txt', 'w') as f:
    text=f.write(str(dt_combined_features))
```

*Figure 5.4.8.1(b) Combining significant features within a particular base learner*

Figure 5.4.8.1(a) displays the output of the overlapping significant feature which is then recorded and saved into a text file for subsequent model training Following that, Figure 5.4.8.1(b) illustrates the combination of significant features within a specific base learner to further enhance the feature set used in training.

Model training with significant features selected from ensemble learning techniques

SVM (Support Vector Machine)

```python
from sklearn import svm

datasets = {
    "base": (X_resampled_train, y_resampled_train, X_resampled_test, y_resampled_test),
    "selected": (X_resample_selected_train, y_resample_selected_train, X_resample_selected_test, y_resample_selected_test),
    "dt_similar": (X_resampled_selected_dt_similar_train, y_resampled_selected_dt_similar_train, X_resampled_selected_dt_similar_test, y_resampled_selected_dt_similar_test),
    "lr_similar": (X_resampled_selected_lr_similar_train, y_resampled_selected_lr_similar_train, X_resampled_selected_lr_similar_test, y_resampled_selected_lr_similar_test),
    "dt_combined": (X_combined_features_dt_train, y_combined_features_dt_train, X_combined_features_dt_test, y_combined_features_dt_test),
    "lr_combined": (X_combined_features_lr_train, y_combined_features_lr_train, X_combined_features_lr_test, y_combined_features_lr_test),
    "dt_stacking": (X_resampled_selected_dt_stacking_train, y_resampled_selected_dt_stacking_train, X_resampled_selected_dt_stacking_test, y_resampled_selected_dt_stacking_test),
    "dt_bagging": (X_resampled_selected_dt_bagging_train, y_resampled_selected_dt_bagging_train, X_resampled_selected_dt_bagging_test, y_resampled_selected_dt_bagging_test),
    "dt_adaboosting": (X_resampled_selected_dt_adaboosting_train, y_resampled_selected_dt_adaboosting_train, X_resampled_selected_dt_adaboosting_test, y_resampled_selected_dt_adaboosting_test),
    "lr_stacking": (X_resampled_selected_lr_stacking_train, y_resampled_selected_lr_stacking_train, X_resampled_selected_lr_stacking_test, y_resampled_selected_lr_stacking_test),
    "lr_bagging": (X_resampled_selected_lr_bagging_train, y_resampled_selected_lr_bagging_train, X_resampled_selected_lr_bagging_test, y_resampled_selected_lr_bagging_test),
    "lr_adaboosting": (X_resampled_selected_lr_adaboosting_train, y_resampled_selected_lr_adaboosting_train, X_resampled_selected_lr_adaboosting_test, y_resampled_selected_lr_adaboosting_test),
    "combine_bagging":(X_resampled_combine_bagging_train, y_resampled_combine_bagging_train, X_resampled_combine_bagging_test, y_resampled_combine_bagging_test)
}

# Store trained models and predictions
svm_models = {}
svm_predictions = {}

# Train and predict for each dataset
for name, (X_train, y_train, X_test,y_test) in datasets.items():
    model = svm.SVC(probability=True)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    y_prob = model.predict_proba(X_test)[:, 1]

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, pos_label=1)
    recall = recall_score(y_test, y_pred, pos_label=1)
    f1 = f1_score(y_test, y_pred, pos_label=1)
    auc = metrics.roc_auc_score(y_test, y_prob)

    # Confusion Matrix Calculation
    cm = confusion_matrix(y_test, y_pred)
    FP, TN, FN, TP = cm[0][1], cm[0][0], cm[1][0], cm[1][1]
    type_I_error = FP / (FP + TN) if (FP + TN) != 0 else 0
    type_II_error = FN / (FN + TP) if (FN + TP) != 0 else 0

    svm_models[name] = model
    svm_predictions[name] = {
        "y_pred": y_pred,
        "accuracy": accuracy,
        "precision": precision,
        "recall": recall,
        "f1_score": f1,
        "auc": auc,
        "type_I_error": type_I_error,
        "type_II_error": type_II_error
    }

    print(f"{name} SVM model trained and predicted.")
```

*Figure 5.4.8.2 Modeling SVM classifier in training and testing with a variety of combinations of significant features*

Table 5.4.2: Description of combined significant features

| **Combination of significant features** | **Description** |
|---|---|
| base | All features (94) |
| selected | Features selected under t-test |
| dt_similar | Overlapping features within decision tree ensemble learning environment |

| | |
|---|---|
| lr_similar | Overlapping features within logistic regression ensemble learning environment |
| dt_combine | Combination of features within decision tree ensemble learning environment |
| lr_combine | Combination of features within logistic regression ensemble learning environment |
| dt_stacking | Significant features from decision tree in stacking ensemble learning environment |
| dt_bagging | Significant features from decision tree in bagging ensemble learning environment |
| dt_adaboosting | Significant features from decision tree in adaboosting ensemble learning environment |
| lr_stacking | Significant features from logistic regression in stacking ensemble learning environment |
| lr_bagging | Significant features from logistic regression in bagging ensemble learning environment |
| lr_adaboosting | Significant features from logistic regression in |

| | |
|---|---|
| | adaboosting ensemble learning environment |
| combine_bagging | Significant features in bagging ensemble learning environment |

```
for name, metrics in svm_predictions.items():
    print(f"Model: {name}")
    print(f"  Accuracy: {metrics['accuracy']:.4f}")
    print(f"  Recall: {metrics['recall']:.4f}")
    print(f"  AUC: {metrics['auc']:.4f}")
    print(f" False Negative: {metrics['type_II_error']:.4f}")
    print("------")
```

```
  Recall: 0.0103
  AUC: 0.8752
 False Negative: 0.9897
------
Model: dt_combined
  Accuracy: 0.7867
  Recall: 0.8291
  AUC: 0.8693
 False Negative: 0.1709
------
Model: lr_combined
  Accuracy: 0.7149
  Recall: 0.7182
  AUC: 0.7905
 False Negative: 0.2818
------
Model: dt_stacking
  Accuracy: 0.6923
  Recall: 0.6912
  AUC: 0.7748
 False Negative: 0.3088
------
Model: dt_bagging
  Accuracy: 0.7110
  Recall: 0.7112
  AUC: 0.7920
 False Negative: 0.2888
------
Model: dt_adaboosting
  Accuracy: 0.7618
  Recall: 0.7637
  AUC: 0.8483
 False Negative: 0.2363
------
Model: lr_stacking
  Accuracy: 0.7058
  Recall: 0.6814
  AUC: 0.7755
 False Negative: 0.3186
------
Model: lr_bagging
  Accuracy: 0.5866
  Recall: 0.3856
  AUC: 0.6442
 False Negative: 0.6144
------
Model: lr_adaboosting
  Accuracy: 0.5915
  Recall: 0.3126
  AUC: 0.6952
 False Negative: 0.6874
------
Model: combine_bagging
  Accuracy: 0.7258
  Recall: 0.7312
```

*Figure 5.4.8.3 Output of modeling SVM classifier in testing with a variety of combinations of significant feature*

```python
import matplotlib.pyplot as plt
import numpy as np

datasets= ["base(94)","selected(82)","dt_overlap(10)","lr_overlap(7)","dt_combined(46)","lr_combined(40)","dt_stacking(29)","dt_bagging(26)","dt_adaboosting(26)",
           "lr_stacking(32)","lr_bagging(29)","lr_adaboosting(14)","combine_bagging(7)"]
fnr_values = [0.1401,0.1439,0.3061,0.9897,0.1709,0.2818,0.3088,0.2888,0.2363,0.3186,0.6144,0.6874,0.2688]


x = np.arange(len(datasets))

plt.plot(x, fnr_values, marker='o', linestyle='-', color='steelblue', linewidth=2, markersize=6)

# Add data point labels
for i, val in enumerate(fnr_values):
    plt.text(x[i], val, f'{val:.3f}%', ha='center', fontsize=8)

plt.xticks(x, datasets, rotation=45, ha='right')
plt.title('False Negative Rate Trend Across Feature Combinations with SVM classifier')
plt.ylabel('False Negative Rate (%)')
plt.ylim(0, 1.0)
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```

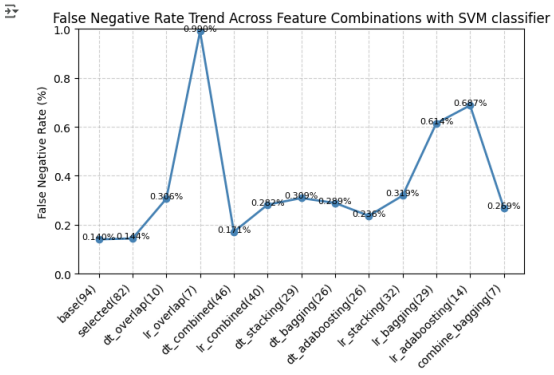



*Figure 5.4.8.4 Visualization of output of modeling SVM classifier in testing*

Figure 5.4.8.2 illustrates the implementation of code used to load the SVM classifier and train it with various combinations of significant features. The descriptions of these combined significant features are detailed in Table 5.4.2. Model evaluation focuses on three primary performance metrics including accuracy, recall and auc score. Test results are presented in figure 5.4.8.3 with a graph visualization shown in figure 5.4.8.4.

## Random Forest

```python
from sklearn.ensemble import RandomForestClassifier

datasets = {
    "base": (X_resampled_train, y_resampled_train, X_resampled_test, y_resampled_test),
    "selected": (X_resample_selected_train, y_resample_selected_train, X_resample_selected_test, y_resample_selected_test),
    "dt_similar": (X_resampled_selected_dt_similar_train, y_resampled_selected_dt_similar_train, X_resampled_selected_dt_similar_test, y_resampled_selected_dt_similar_test),
    "lr_similar": (X_resampled_selected_lr_similar_train, y_resampled_selected_lr_similar_train, X_resampled_selected_lr_similar_test, y_resampled_selected_lr_similar_test),
    "dt_combined": (X_combined_features_dt_train, y_combined_features_dt_train, X_combined_features_dt_test, y_combined_features_dt_test),
    "lr_combined": (X_combined_features_lr_train, y_combined_features_lr_train, X_combined_features_lr_test, y_combined_features_lr_test),
    "dt_stacking": (X_resampled_selected_dt_stacking_train, y_resampled_selected_dt_stacking_train, X_resampled_selected_dt_stacking_test, y_resampled_selected_dt_stacking_test),
    "dt_bagging": (X_resampled_selected_dt_bagging_train, y_resampled_selected_dt_bagging_train, X_resampled_selected_dt_bagging_test, y_resampled_selected_dt_bagging_test),
    "dt_adaboosting": (X_resampled_selected_dt_adaboosting_train, y_resampled_selected_dt_adaboosting_train, X_resampled_selected_dt_adaboosting_test, y_resampled_selected_dt_adaboosting_test),
    "lr_stacking": (X_resampled_selected_lr_stacking_train, y_resampled_selected_lr_stacking_train, X_resampled_selected_lr_stacking_test, y_resampled_selected_lr_stacking_test),
    "lr_bagging": (X_resampled_selected_lr_bagging_train, y_resampled_selected_lr_bagging_train, X_resampled_selected_lr_bagging_test, y_resampled_selected_lr_bagging_test),
    "lr_adaboosting": (X_resampled_selected_lr_adaboosting_train, y_resampled_selected_lr_adaboosting_train, X_resampled_selected_lr_adaboosting_test, y_resampled_selected_lr_adaboosting_test),
    "combine_bagging":(X_resampled_combine_bagging_train, y_resampled_combine_bagging_train, X_resampled_combine_bagging_test, y_resampled_combine_bagging_test)
}

# Store trained models and predictions
RF_models = {}
RF_predictions = {}

# Train and predict for each dataset
for name, (X_train, y_train, X_test,y_test) in datasets.items():
    model = RandomForestClassifier(criterion = "gini",max_depth = 5,min_samples_split = 2,min_samples_leaf=1,random_state = 42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    y_prob = model.predict_proba(X_test)[:, 1]

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, pos_label=1)
    recall = recall_score(y_test, y_pred, pos_label=1)
    f1 = f1_score(y_test, y_pred, pos_label=1)
    auc = roc_auc_score(y_test, y_prob)

    # Confusion Matrix Calculation
    cm = confusion_matrix(y_test, y_pred)
    FP, TN, FN, TP = cm[0][1], cm[0][0], cm[1][0], cm[1][1]
    type_I_error = FP / (FP + TN) if (FP + TN) != 0 else 0
    type_II_error = FN / (FN + TP) if (FN + TP) != 0 else 0

    RF_models[name] = model
    RF_predictions[name] = {
        "y_pred": y_pred,
        "accuracy": accuracy,
        "precision": precision,
        "recall": recall,
        "f1_score": f1,
        "auc": auc,
        "type_I_error": type_I_error,
        "type_II_error": type_II_error
    }

    print(f"{name} RF model trained and predicted.")
```

*Figure 5.4.9.1 Modeling Random Forest classifier in training and testing with a variety of combinations of significant features*

```
for name, metrics in RF_predictions.items():
    print(f"Model: {name}")
    print(f"  Accuracy: {metrics['accuracy']:.4f}")
    print(f"  Recall: {metrics['recall']:.4f}")
    print(f"  AUC: {metrics['auc']:.4f}")
    print(f" False Negative: {metrics['type_II_error']:.4f}")
    print("------")
```

```
  Recall: 0.9183
  AUC: 0.9631
 False Negative: 0.0817
------
Model: dt_combined
  Accuracy: 0.9266
  Recall: 0.9405
  AUC: 0.9785
 False Negative: 0.0595
------
Model: lr_combined
  Accuracy: 0.9209
  Recall: 0.9308
  AUC: 0.9752
 False Negative: 0.0692
------
Model: dt_stacking
  Accuracy: 0.9251
  Recall: 0.9329
  AUC: 0.9759
 False Negative: 0.0671
------
Model: dt_bagging
  Accuracy: 0.9295
  Recall: 0.9448
  AUC: 0.9786
 False Negative: 0.0552
------
Model: dt_adaboosting
  Accuracy: 0.9292
  Recall: 0.9529
  AUC: 0.9775
 False Negative: 0.0471
------
Model: lr_stacking
  Accuracy: 0.9186
  Recall: 0.9329
  AUC: 0.9723
 False Negative: 0.0671
------
Model: lr_bagging
  Accuracy: 0.9173
  Recall: 0.9297
  AUC: 0.9706
 False Negative: 0.0703
------
Model: lr_adaboosting
  Accuracy: 0.9217
  Recall: 0.9346
  AUC: 0.9721
 False Negative: 0.0654
------
Model: combine_bagging
  Accuracy: 0.9269
  Recall: 0.9416
```

*Figure 5.4.9.2 Output of modeling Random Forest classifier in testing with a variety combination of significant features*

```
fnr_values_rf = [0.0438,0.0487,0.0498,0.0817,0.0595,0.0692,0.0671,0.0552,0.0471,0.0671,0.0703,0.0654,0.0584]

x = np.arange(len(datasets))

plt.plot(x, fnr_values_rf, marker='o', linestyle='-', color='red', linewidth=2, markersize=6)

# Add data point labels
for i, val in enumerate(fnr_values_rf):
    plt.text(x[i], val, f'{val:.3f}%', ha='center', fontsize=8)

plt.xticks(x, datasets, rotation=45, ha='right')
plt.title('False Negative Rate Trend Across Feature Combinations with RF classifier')
plt.ylabel('False Negative Rate (%)')
plt.grid(True, linestyle='--', alpha=0.6)
plt.ylim(0.04, 0.085)
plt.tight_layout()
plt.show()
```

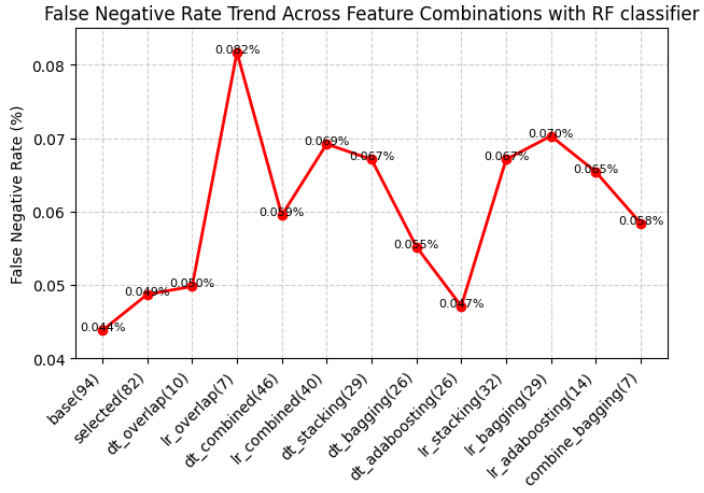



*Figure 5.4.9.3 Visualization of output of modeling Random Forest classifier in testing*

Figure 5.4.9.1 illustrates the implementation of code used to load the Random Forest classifier and train it with various combinations of significant features. Model evaluation focuses on three primary performance metrics including accuracy, recall and auc score. Test results are displayed in figure 5.4.9.2, with a graph visualization shown in figure 5.4.9.3

## 5.5 Implementation Issues and Challenges

One of the key challenges in this project is the features provided in the dataset not fully aligned with the variables in the Altman-Z score for financial distress prediction. Due to the absence of certain variables, the model cannot directly apply the z score equation. As a result, feature selection was conducted using t-test, only selecting those features with p value less than 0.5 as the statistical significance and contribute to the financial distress detection. Besides that, another challenge encountered was the high computational resources requirement when implementing the ensemble learning technique on base learners like logistic regression and decision tree. Among the techniques, adaboosting required the most computational time likely due to its sequential architecture where the weak learners are trained after another with iteratively weight updates. This limits parallelization and increases overall processing time with large datasets.

## 5.6 Concluding Remark

In this chapter, experimental setup and simulation procedures were presented, including hardware and software configurations, system operation, and implementation details. The proposed model was evaluated within different ensemble learning environments to assess the performance of various ensemble techniques. Further analysis was conducted by introducing two new classifiers to train on the significant features selected by each base learner within each ensemble framework. Based on the results gained, the degrees of effectiveness of ensemble learning techniques vary based on the base learner used. The insights gained from the experimental process serve as a valuable foundation for informed decision-making in suggesting the best ensemble learning techniques in the financial distress detection.

# Chapter 6

# SYSTEM EVALUATION AND DISCUSSION

## 6.1 System Testing and Performance Metrics

Comparison of Logistic Regression and Decision Tree on three ensemble learning techniques

Logistic Regression

Table 6.1.1: Result of applying logistic regression as base learner in three ensemble learning environment

| Method | Bagging | Stacking | Adaboosting |
|---|---|---|---|
| Test Accuracy | 0.9090 | 0.9069 | 0.8867 |
| False positive | 0.1005 | 0.0961 | 0.0971 |
| **False Negative** | 0.0806 | 0.0898 | 0.1309 |
| AUC | 0.9674 | 0.9661 | 0.9488 |
| Precision | 0.8938 | 0.8971 | 0.8918 |
| Recall | 0.9194 | 0.9102 | 0.8691 |
| F1 score | 0.9064 | 0.9036 | 0.8803 |
| **Time Processing** | 59.99 | 200.09 | 11.54 |

Decision Tree

Table 6.1.2: Result of applying decision tree as base learner in three ensemble learning environment

| Method | Bagging | Stacking | Adaboosting |
|---|---|---|---|
| Test Accuracy | 0.9463 | 0.9147 | 0.9601 |
| False positive | 0.0786 | 0.1010 | 0.0572 |
| **False Negative** | 0.0265 | 0.0681 | 0.0211 |
| AUC | 0.9876 | 0.9671 | 0.9923 |
| Precision | 0.9193 | 0.8946 | 0.9425 |
| Recall | 0.9735 | 0.9319 | 0.9805 |
| F1 score | 0.9456 | 0.9128 | 0.9611 |
| **Time Processing** | 89.46 | 783.41 | 9489.64 |

In table 6.1.1 and table 6.1.2 have presented the result of applying logistic regression and decision tree as base learners in three ensemble learning environment respectively. Based on table 6.1.1, bagging demonstrated the best performance in logistic regression, achieving the lowest false negative rate of 8.06%. This indicates only 8.06% of distressed companies were misclassified as non-distressed. Besides, it recorded the highest AUC score of 96.74%, reflecting that the model's strong ability to distinguish between normal and distressed companies. The model required approximately 1 minute to converge during training, highlighting its efficiency. In contrast, Table 6.1.2 shows that Adaboosting with decision tree achieved the best performance of lowest false negative rate of 2.11% and highest AUC score of 99.23%. On the other hand, it is computationally expensive since it required 9489.64 seconds and yet to fully converging. In contrast, bagging produced a comparable false negative rate of 2.65% but with significantly lower computational cost, making it the recommended ensemble technique for financial distress detection due to its balanced performance and efficiency.

**SVM**

Table 6.1.3: Result of applying SVM in training on variety combination of significant features

| Model Type | No of Features | Accuracy | Recall | AUC | False Negative Rate |
|---|---|---|---|---|---|
| Base Feature | 94 | 0.8180 | 0.8599 | 0.8959 | 0.1401 |
| Selected Feature(p value) | 82 | 0.8173 | 0.8561 | 0.8933 | 0.1439 |
| Decision Tree Overlap | 10 | 0.7022 | 0.6939 | 0.7764 | 0.3061 |
| Logistic Regression Overlap | 7 | 0.5254 | 0.0103 | 0.8752 | 0.9897 |
| Decision Tree Combine | 46 | 0.7867 | 0.8291 | 0.8693 | 0.1709 |
| Logistic Regression Combine | 40 | 0.7149 | 0.7182 | 0.7905 | 0.2818 |
| Bagging Combine | 47 | 0.7258 | 0.7312 | 0.8097 | 0.2688 |
| Decision Tree Stacking | 29 | 0.6923 | 0.6912 | 0.7748 | 0.3088 |
| Decision Tree Bagging | 26 | 0.7110 | 0.7112 | 0.7920 | 0.2888 |
| Decision Tree Adaboosting | 26 | 0.7618 | 0.7637 | 0.8483 | 0.2363 |
| Logistic Regression Stacking | 32 | 0.7058 | 0.6814 | 0.7755 | 0.3186 |

| | | | | | |
|---|---|---|---|---|---|
| Logistic Regression Bagging | 29 | 0.5866 | 0.3856 | 0.6442 | 0.6144 |
| Logistic Regression Adaboosting | 14 | 0.5915 | 0.3126 | 0.6952 | 0.6874 |

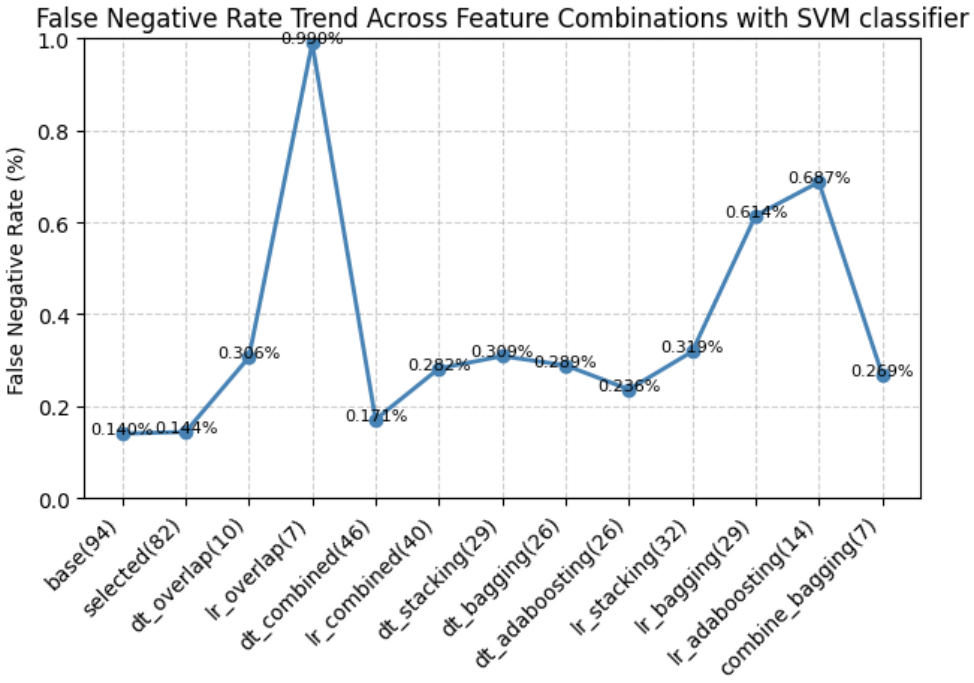



*Figure 6.1.1 Visualization of result of applying SVM in training on variety combination of significant features*

Summary

Based on table 6.1.3, it is clear that feature set derived from the Decision Tree-based selection has better performance compared to those selected from logistic regression. Specifically, the Decision Tree combination achieved a lower false negative rate of 17.09% than the rate of 28.18% from logistic regression combination. Besides that, it also suggested decision tree model under bagging ensemble learning environment has a balance performance which shows an average performance with a false negative rate of 28.88% and shorter time processing time compared to Adaboosting environment. This suggested that using Decision Tree as a base

learner in bagging environment could effectively achieve a better performance and optimal time processing in detecting the financial distress. To further evaluate this, a combination of features from both weak learners was tested within the Bagging framework. The Bagging Combine model outperformed the Logistic Regression Combine across all metrics—achieving higher accuracy, recall, and AUC, as well as a lower false negative rate. However, the Decision Tree Combine model still recorded better recall and a lower false negative rate than Bagging Combine. This suggests that while Bagging improves overall model robustness, particularly when blending features from diverse weak learners, the Decision Tree alone exhibits strong predictive power. A graphical visualization of these results is presented in Figure 6.1.1, which provides a clear overview of the model performance across various ensemble learning configurations.

**Random Forest**

Table 6.1.4: Result of applying Random Forest in training on variety combination of significant features

| Model Type | No of Features | Accuracy | Recall | AUC | False Negative Rate |
|---|---|---|---|---|---|
| Base Feature | 94 | 0.9339 | 0.9562 | 0.9805 | 0.0438 |
| Selected Feature (p value) | 82 | 0.9321 | 0.9513 | 0.9797 | 0.0487 |
| Decision Tree Similar | 10 | 0.9264 | 0.9502 | 0.9754 | 0.0498 |
| Logistic Regression Similar | 7 | 0.9080 | 0.9183 | 0.9631 | 0.0817 |
| Decision Tree Combine | 46 | 0.9266 | 0.9405 | 0.9785 | 0.0595 |
| Logistic Regression Combine | 40 | 0.9209 | 0.9308 | 0.9752 | 0.0692 |
| Bagging Combine | 47 | 0.9269 | 0.9416 | 0.9776 | 0.0584 |
| Decision Tree Stacking | 29 | 0.9251 | 0.9329 | 0.9759 | 0.0671 |
| Decision Tree Bagging | 26 | 0.9295 | 0.9448 | 0.9786 | 0.0552 |
| Decision Tree Adaboosting | 26 | 0.9292 | 0.9529 | 0.9775 | 0.0471 |
| Logistic Regression Stacking | 32 | 0.9186 | 0.9329 | 0.9723 | 0.0671 |

| | | | | | |
|---|---|---|---|---|---|
| Logistic Regression Bagging | 29 | 0.9173 | 0.9217 | 0.9706 | 0.0703 |
| Logistic Regression Adaboosting | 94 | 0.9217 | 0.9346 | 0.9721 | 0.0654 |

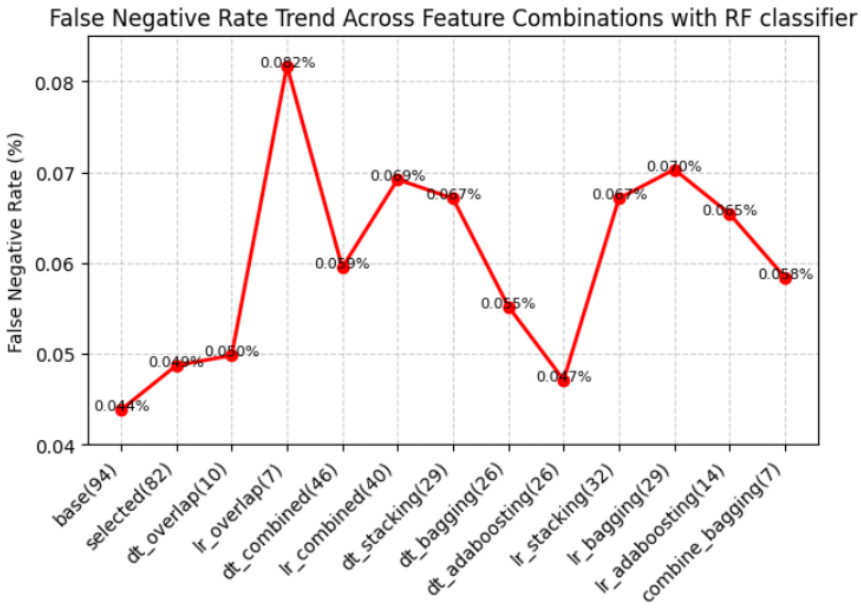



*Figure 6.1.2 Visualization of result of applying Random Forest in training on variety combination of significant features*

Summary

Based on table 6.1.4, it is clear that the features selected from three ensemble learning environments, it is observed that feature set derived from the Decision Tree-based selection has better performance compared to those selected from logistic regression. Specifically, the Decision Tree combination achieved a lower false negative rate of 5.95% than the rate of 6.92% from logistic regression combination. Besides that, it also suggested decision tree model under bagging ensemble learning environment has a balance performance which it shown an average of performance with a false negative rate of 5.52% and shorter time processing time compared to adaboosting environment. This suggested that using Decision Tree as a choice as the base

learner in a bagging environment is a recommended choice to achieve a better performance in detecting the financial distress. To further validate this, a combination of features from both weak learners was tested within the Bagging framework. A slightly different trend compared to when using SVM classifier, the Bagging Combine model achieved higher recall and a lower false negative rate than the Decision Tree Combine model. This indicates that the choice of classifier can influence results. As an overview bagging ensemble framework is a recommended choice in financial distress detection.

## 6.2 Testing Setup and Result

Financial Implication

Table 6.2.1: Financial Indicators from Literature Review

| Category | Description | General Indicator |
|---|---|---|
| Liquidity Ratios | It measures a company's financial health and the ease of convert assets into cash to pay off liabilities [22]. | Current Ratio, Cash Ratio, Quick Ratio, Operating Cash Flow Ratio |
| Solvency (Debt) Ratios | It measures amount of company's assets financed by debt [22]. | Debt to Equity Ratio, Long Term Debt Ratio, Equity Ratio, Short Term Debt Ratio |
| Profitability Ratios | It measures company’s ability to generate profit relative to its sales, assets, and equity. [22]. | Margin Ratio:<br>Gross Profit Margin, Net Profit Margin, Operating Profit Margin, Net Profit Margin<br><br>Return Ratio: |

| | | |
|---|---|---|
| | | Return on Assets, Return on Equity |
| Operational Efficiency Ratios | It measures company's ability to effectively employ its resources to produce income [22]. | Inventory Turnover Ratio, Accounts Payables Turnover, Account Receivables Turnover, Assets Turnover Ratio |

Table 6.2.2: Categorization of features into respective categories

| Category | Description | Feature from the dataset [33] |
|---|---|---|
| Solvency | It measures amount of company's assets financed by debt [22]. | X1-X28 |
| Capital structure ratios | It assess company's long-term financial stability and the proportion of debt and equity in its financing [29] | X29-X37 |
| Others | - | X38-X50 |
| Profitability | It measures the company's ability to generate profit relative to its sales, assets, and equity [22]. | X51-X69 |
| Turnover ratios | It measures the amount of assets or liabilities that a company replaces in relation to its sales for determining efficiency in utilizing its assets.[30] | X70-X82 |

| | | |
|---|---|---|
| Cash flow ratios | It assess company's ability to pay dividends to investors by comparing cash flows to other elements of an entity's financial statements [31] | X83-X87 |
| Growth | It assess company's performance and predicting future performance by expressing the annual change in a variable as a percentage. [32] | X88-X95 |

Table 6.2.3 Overlap indicators selected using t-test and the financial indicator categories identified

| Feature | Category |
|---|---|
| X0: ROA(C) before interest and depreciation before interest | Liquidity |
| X1: ROA(A) before interest and % after tax | |
| X2: ROA(B) before interest and depreciation after tax | |
| X3: Operating Gross Margin | |
| X4: Realized Sales Gross Margin | |
| X6: Pre-tax net Interest Rate | |
| X7: After-tax net Interest Rate | |
| X8: Non-industry income and expenditure/revenue | |
| X9: Continuous interest rate (after tax) | |
| X10: Operating Expense Rate | |
| X11: Research and development expense rate | |
| X12: Cash flow rate | |
| X13: Interest-bearing debt interest rate | |
| X14: Tax rate (A) | |
| X15: Net Value Per Share (B) | |
| X16: Net Value Per Share (A) | |

| | |
|---|---|
| X17: Net Value Per Share I | |
| X18: Persistent EPS in the Last Four Seasons | |
| X19: Cash Flow Per Share | |
| X20: Revenue Per Share (Yuan Â¥) | |
| X21: Operating Profit Per Share (Yuan Â¥) | |
| X22: Per Share Net profit before tax (Yuan Â¥) | |
| X24: Operating Profit Growth Rate | |
| X25: After-tax Net Profit Growth Rate | |
| X26: Regular Net Profit Growth Rate | |
| X27: Continuous Net Profit Growth Rate | |
| X28: Total Asset Growth Rate | |
| X29: Net Value Growth Rate | Capital structure ratios |
| X30: Total Asset Return Growth Rate Ratio | |
| X31: Cash Reinvestment Percentage | |
| X33: Quick Ratio | |
| X34: Interest Expense Ratio | |
| X35: Total debt/Total net worth | |
| X36: Debt ratio % | |
| X37: Net worth/Assets | |
| X39: Borrowing dependency | Others |
| X40: Contingent liabilities/Net worth | |
| X41: Operating profit/Paid-in capital | |
| X42: Net profit before tax/Paid-in capital | |
| X43: Inventory and accounts receivable/Net value | |
| X44: Total Asset Turnover | |
| X45: Accounts Receivable Turnover | |
| X46: Average Collection Days | |
| X47: Inventory Turnover Rate (times) | |
| X48: Fixed Assets Turnover Frequency | |
| X49: Net Worth Turnover Rate (times) | |
| X50: Revenue per person | |
| X51: Operating profit per person | Profitability |

| | |
|---|---|
| X53: Working Capital to Total Assets | |
| X54: Quick Assets/ Total Assets | |
| X55: Current Assets/Total Assets | |
| X56: Cash/Total Assets | |
| X58: Cash/Current Liability | |
| X59: Current Liability to Assets | |
| X60: Operating Funds to Liability | |
| X63: Current Liabilities/Liability | |
| X64: Working Capital/Equity | |
| X65: Current Liabilities/Equity | |
| X66: Long-term Liability to Current Assets | |
| X67: Retained Earnings to Total Assets | |
| X68: Total income/Total expense | |
| X69: Total expense/Assets | |
| X70: Current Asset Turnover Rate | Turnover Ratios |
| X71: Quick Asset Turnover Rate | |
| X73: Cash Turnover Rate | |
| X75: Fixed Assets to Assets | |
| X76: Current Liability to Liability | |
| X77: Current Liability to Equity | |
| X78: Equity to Long-term Liability | |
| X79: Cash Flow to Total Assets | |
| X80: Cash Flow to Liability | |
| X81: CFO to Assets | |
| X82: Cash Flow to Equity | |
| X83: Current Liability to Current Assets | Cash Flow Ratios |
| X85: Net Income to Total Assets | |
| X86: Total assets to GNP price | |
| X87: No-credit Interval | |
| X88: Gross Profit to Sales | Growth |
| X89: Net Income to Stockholder's Equity | |
| X90: Liability to Equity | |

| | |
|---|---|
| X91: Degree of Financial Leverage (DFL) | |
| X93: Net Income Flag | |

Table 6.2.4 Overlap indicators selected using t-test and the general financial indicator

| No | Features |
|---|---|
| 1 | X33: Quick Ratio |
| 2 | X36: Debt Ratio |
| 3 | X85: Net Income to Total Assets |
| 4 | X89: Net Income to Stockholder's Equity |
| 5 | X44: Total Asset Turnover |
| 6 | X47: Inventory Turnover Rate |
| 7 | X45: Accounts Receivable Turnover |

In general, to assess the financial position of a company whether it is distress or non-distress, there are a few key financial indicator categories including liquidity analysis, operational efficiency (efficacy) analysis, debt (solvency) analysis and profitability analysis [23] as iutlined in Table 6.2.1. Based on table 6.2.2, it has displayed the features that have been mapped to these categories like solvency, capital structure ratios, profitability, turnover ratios, cash flow ratios, growth and others. In table 6.2.3, it has highlighted an overlap between statistically selected features (via t-test) and financial indicator categories identified. Specifically, the selected features include 27 from solvency ratios, 8 from capital structure ratios, 15 from profitability ratios, 11 from turnover ratios, 4 from cash flow ratios, 5 from growth indicators, and 12 from other categories. It suggests that the solvency category features display a significant importance in financial distress as evidenced by their frequent selection through the t-test. In table 6.2.4, there are a total of 7 features overlapping with the general indicators, reinforcing the model’s practical relevance and its effectiveness in identifying distress through features widely supported in financial literature.

Table 6.2.5 Significant features from bagging environment

| Feature Name |
|---|
| X0: ROAI before interest and depreciation before interest |
| X1: ROA(A) before interest and % after tax |
| X6: Pre-tax net Interest Rate |
| X8: Non-industry income and expenditure/revenue |
| X9: Continuous interest rate (after tax) |
| X10: Operating Expense Rate |
| X11: Research and development expense rate |
| X12: Cash flow rate |
| X14: Tax rate (A) |
| X15: Net Value Per Share (B) |
| X16: Net Value Per Share (A) |
| X17: Net Value Per Share I |
| X18: Persistent EPS in the Last Four Seasons |
| X22: Per Share Net profit before tax (Yuan ¥) |
| X25: After-tax Net Profit Growth Rate |
| X26: Regular Net Profit Growth Rate |
| X27: Continuous Net Profit Growth Rate |
| X34: Interest Expense Ratio |
| X35: Total debt/Total net worth |
| X36: Debt ratio % |
| X37: Net worth/Assets |
| X39: Borrowing dependency |
| X40: Contingent liabilities/Net worth |
| X41: Operating profit/Paid-in capital |
| X43: Inventory and accounts receivable/Net value |
| X44: Total Asset Turnover |
| X45: Accounts Receivable Turnover |
| X46: Average Collection Days |
| X48: Fixed Assets Turnover Frequency |
| X49: Net Worth Turnover Rate (times) |

| |
|---|
| X51: Operating profit per person |
| X54: Quick Assets/Total Assets |
| X56: Cash/Total Assets |
| X59: Current Liability to Assets |
| X60: Operating Funds to Liability |
| X67: Retained Earnings to Total Assets |
| X71: Quick Asset Turnover Rate |
| X73: Cash Turnover Rate |
| X76: Current Liability to Liability |
| X78: Equity to Long-term Liability |
| X80: Cash Flow to Liability |
| X82: Cash Flow to Equity |
| X85: Net Income to Total Assets |
| X89: Net Income to Stockholder's Equity |
| X90: Liability to Equity |
| X91: Degree of Financial Leverage (DFL) |
| X93: Equity to Liability |

Table 6.2.6 Overlap indicators selected from bagging ensemble learning framework and the financial indicator categories identified.

| Feature | Category |
|---|---|
| X0:ROAI before interest and depreciation before interest | Solvency |
| X1:ROA(A) before interest and % after tax | |
| X6:Pre-tax net Interest Rate | |
| X8:Non-industry income and expenditure/revenue | |
| X9: Continuous interest rate (after tax) | |
| X10: Operating Expense Rate | |
| X11: Research and development expense rate | |
| X12: Cash flow rate | |
| X14: Tax rate (A) | |

| | |
|---|---|
| X15: Net Value Per Share (B) | |
| X16: Net Value Per Share (A) | |
| X17: Net Value Per Share I | |
| X18: Persistent EPS in the Last Four Seasons | |
| X22: Per Share Net profit before tax (Yuan ¥) | |
| X25: After-tax Net Profit Growth Rate | |
| X26: Regular Net Profit Growth Rate | |
| X27: Continuous Net Profit Growth Rate | |
| X34: Interest Expense Ratio | Capital Structure Ratios |
| X35: Total debt/Total net worth | |
| X36: Debt ratio % | |
| X37: Net worth/Assets | |
| X39: Borrowing dependency | Others |
| X40: Contingent liabilities/Net worth | |
| X41: Operating profit/Paid-in capital | |
| X43: Inventory and accounts receivable/Net value | |
| X44: Total Asset Turnover | |
| X45: Accounts Receivable Turnover | |
| X46: Average Collection Days | |
| X48: Fixed Assets Turnover Frequency | |
| X49: Net Worth Turnover Rate (times) | |
| X51: Operating profit per person | Profitability |
| X54: Quick Assets/Total Assets | |
| X56: Cash/Total Assets | |
| X59: Current Liability to Assets | |
| X60: Operating Funds to Liability | |
| X67: Retained Earnings to Total Assets | |
| X71: Quick Asset Turnover Rate | Turnover Ratios |
| X73: Cash Turnover Rate | |
| X76: Current Liability to Liability | |
| X78: Equity to Long-term Liability | |

| | |
|---|---|
| X80: Cash Flow to Liability | |
| X82: Cash Flow to Equity | |
| X85: Net Income to Total Assets | Cash Flow Ratios |
| X89: Net Income to Stockholder's Equity | Growth |
| X90: Liability to Equity | |
| X91: Degree of Financial Leverage (DFL) | |
| X93: Equity to Liability | |

Table 6.2.7 Overlap indicators selected using bagging ensemble framework and the general financial indicator

| No | Features |
|---|---|
| 1 | X45: Accounts Receivable Turnover |
| 2 | X36: Debt Ratio |
| 3 | X85: Net Income to Total Assets |
| 4 | X89: Net Income to Stockholder's Equity |

In table 6.2.5, it has shown a total of 47 significant features selected by bagging ensemble environment from both logistic regression and decision tree. As shown in Table 6.2.6, the selected features include 17 from solvency ratios, 4 from capital structure ratios, 6 from profitability ratios, 1 from cash flow ratios, 6 from turnover ratios, 4 from growth indicators, and 9 from other categories. Solvency category features dominated both ensemble learning techniques and statistical method highlighting their critical importance in detecting financial distress. In table 6.2.7 shows that a total of 4 features overlapping from bagging ensemble framework with the general indicators, suggesting that bagging ensemble framework's capability relevant and theoretically supported indicators. By narrowing the scope to key features, the ensemble approach enhances both the efficiency and practical relevance of the predictive model.

## 6.3 Project Challenges

One of the key challenges encountered in this project was the limited availability of computational resources, particularly when training resource-intensive models such as AdaBoosting. AdaBoosting involves iterative training where weights are updated subsequently in each iteration to focus on correcting the errors made by previous models. This architecture increases computational demand. Since the project was implemented using Google Colab, which has restricted memory and disk space, it posed a limitation on executing large-scale model training. As a result, the ability to conduct extensive hyperparameter tuning and explore deeper model configurations was constrained, potentially impacting model optimization. Additionally, when evaluating model performance, it is crucial to select performance metrics that reflect real-world concerns. For example, greater emphasis was placed on the false negative rate because misclassifying a financially distressed company as normal could have severe consequences. This focus ensured that the evaluation process prioritized the detection of financial distress accurately.

## 6.4 Objectives Evaluation

Objective 1: Familiarize the architecture ensemble learning techniques

Evaluation: The objective has been achieved. Two weak learners such as logistic regression and decision tree, have been implemented into three ensemble learning environments which are bagging, stacking and adaboosting. Each of ensemble learning techniques has its own uniqueness, for example, bagging focuses on training multiple models on different subsets of data and aggregating predictions through major voting. For adaboosting, the techniques focus on optimizing overall performance by allocating higher weight on misclassified classes to let the model make more focuses on wrongly classified ones. For stacking, it involves training multiple base models parallelly and according to the combination of outputs as input to the meta model, which learns from the intermediate predictions the same target.

Objective 2: Compare and contrast three ensemble learning techniques (stacking, bagging and boosting) using classifiers like logistic regression and decision tree in classifying financial status of companies

Evaluation: The objective has been achieved. According to the result obtained using Logistic Regression as the weak learner in three ensemble learning environment, bagging demonstrated the best performance with the lowest negative rate of 8.06%, compared to 8.98% from stacking

and 13.09% from adaboosting. In contrast, when applying decision tree as weak learner, adaboosting outperformed with the lowest false negative rate, indicating its strength in detecting the financial distress. However, it required significantly more computational time, making it less efficient. Bagging achieved slightly lower performance than adaboosting, offered a balance by maintaining competitive performance with lower computational cost, proving to be more efficient choice in detecting the financial distress.

Furthermore, the features selected from decision tree and logistic regression in three ensemble learning environments have been applied on new classification models like SVM and Random Forest to further evaluate the performance of each ensemble learning techniques. Results have further proven bagging ensemble learning

Objective 3: Relate the findings to interpret the business implications of financial distress

Evaluation: The objective has been achieved. The features selected from t-test with p value less than 0.05 have been compared against financial indicators identified in literature. It confirms that statistically significant features are aligned with financial indicators used in real world financial distress analysis. This also further reinforces its practical relevance in the financial distress detection in business contexts. Additionally, the significant features selected from bagging were also compared with literature-based indicators. The findings provide further evidence that applying machine learning techniques in financial distress detection not only captures features aligned with established financial guidelines but also has the potential to uncover underling significant features.

## 6.5 Concluding Remark

This chapter has presented the system testing procedures and performance metrics used to evaluate which proposed ensemble learning technique works the best with the base learner. Further evaluation is recorded under the testing up and result but more emphasis on the business respective like the overlapping features within the literature-based indicators and the indicators selected from ensemble learning framework. The challenges encountered during implementation has been discussed and should be considered for improvement in future work.Lastly, the project objectives were evaluated to check whether it is aligned with throughout the project process.

# Chapter 7 Conclusion and Recommendation

## 7.1 Conclusion

In these findings has demonstrated that Bagging is the recommended ensemble learning techniques for financial distress detection. It delivers outstanding performance with the lowest false negative rate and maintains a reasonable computational time, making it both accurate and efficient. In contrast, Adaboosting requires significantly more computational resources to achieve a higher precision while applying decision tree as the basic learner. However, when using logistic regression, it converged the fastest but resulted in the lowest validation loss. Similarly, Stacking shows inconsistent performance, sometimes required longer time to converge and occasionally yielding results that are either the lowest or average among the three methods. These observations highlight Bagging's robustness in financial distress detection. Moreover, across the ensemble environments, logistic regression generally showed lower predictive performance compared to decision tree, suggesting that decision tree might be a more suitable weak learner in this scenario.

## 7.2 Recommendation

The dataset used in this study for financial distress detection primarily consists of financial ratios and corporate governance indicators. For practical deployment, it is recommended to evaluate ensemble learning techniques on datasets that incorporate more diverse features, such as macroeconomic indicators or industry-specific factors, to better assess the model's generalizability. Additionally, this project tested only two classifiers (Logistic Regression and Decision Tree) within the ensemble frameworks. It is recommended to explore and compare a broader range of base classifiers, such as Support Vector Machines (SVM), Random Forests, or Gradient Boosting Machines, within the Bagging framework. This will help further validate the robustness and adaptability of Bagging as an ensemble method across different types of base learners.

# REFERENCES

**Article:**

[1] A. Hayes, "Financial distress: Definition, signs, and remedies," Investopedia, Apr. 18, 2021. https://www.investopedia.com/terms/f/financial_distress.asp

[2] V. Agarwal and R. Taffler, "Comparing the performance of market-based and accounting-based bankruptcy prediction models," Journal of Banking & Finance, vol. 32, no. 8, pp. 1541–1551, https://doi.org/10.1016/j.jbankfin.2007.07.014.

[3] R. S. H, "Ensemble Models in Machine Learning – Intuitive Tutorials," Intuitive Tutorials, May 12, 2023. https://intuitivetutorial.com/2023/05/12/ensemble-models-in machine-learning/ (accessed Sep. 05, 2024).

[4] W. Kenton, "Altman Z-Score," *Investopedia*, Jun. 21, 2022. https://www.investopedia.com/terms/a/altman.asp

[5] TradingView, "Springate score," *TradingView*, 2024. https://www.tradingview.com/support/solutions/43000597848-springate-score/

[6] TradingView, "Zmijewski score," *TradingView*. https://www.tradingview.com/support/solutions/43000597850-zmijewski-score/

[7] M. Arora and C. Jiyani, "GAP iNTERDISCIPLINARITIES A Global Journal of Interdisciplinary Studies AN ANALSIS OF EFFICACY OF FINANCIAL DISTRESS PREDICTION SPRINGATE AND GROVER MODEL," 2022. Accessed: Nov. 29, 2024. [Online]. Available: https://www.gapinterdisciplinarities.org/res/articles/(74-77)%20AN%20ANALSIS%20OF%20EFFICACY%20OF%20FINANCIAL%20DISTRESS%20PREDICTION%20SPRINGATE%20AND%20GROVER%20MODEL.pdf

[8] Muhtar Sapiri, "A Qualitative Analysis on the Role of Auditors in Preventing Financial Crises," *Golden Ratio Of Auditing Research*, vol. 4, no. 2, pp. 89–106, Mar. 2024, doi: https://doi.org/10.52970/grar.v4i2.393.

[9] Cristina Maria Voinea, V. State, Dan Marius Coman, and Ana-Maria Dascălu, "The Role and Importance of the Financial Audit Report in the Decision-Making Process in Audited Companies," *Valahian Journal of Economic Studies*, vol. 15, no. 1, pp. 87–94, Jun. 2024, doi: https://doi.org/10.2478/vjes-2024-0007.

[10] K. L. Tran, H. A. Le, T. H. Nguyen, and D. T. Nguyen, "Explainable Machine Learning for Financial Distress Prediction: Evidence from Vietnam," *Data*, vol. 7, no. 11, p. 160, Nov. 2022, doi: https://doi.org/10.3390/data7110160.

[11] M. J. Rahman and H. Zhu, "Predicting financial distress using machine learning approaches: Evidence China," *Journal of Contemporary Accounting & Economics*, vol. 20, no. 1, p. 100403, Feb. 2024, doi: https://doi.org/10.1016/j.jcae.2024.100403.

[12] "What is ensemble learning? | IBM," *www.ibm.com*, Feb. 09, 2024. https://www.ibm.com/topics/ensemble-learning

[13] S. Wang and G. Chi, "Cost-sensitive stacking ensemble learning for company financial distress prediction," Expert Systems With Applications, vol. 255, p. 124525, Dec. 2024, doi: 10.1016/j.eswa.2024.124525.

[14] F. T. Kristanti and V. Dhaniswara, "The accuracy of artificial neural networks and logit models in predicting the companies' financial distress," Journal of technology management & innovation, vol. 18, no. 3, pp. 42–50, 2023, doi: https://doi.org/10.4067/s0718-27242023000300042.

[15] "The Journal of Informatics Development," E-ISSN: 2963-0568, P-ISSN: 2963-055X. Available online: https://ejournal.itbwigalumajang.ac.id/index.php/jid. Accessed: [Insert Date of Access].

[16] "UCI Machine Learning Repository," archive.ics.uci.edu. https://archive.ics.uci.edu/dataset/572/taiwanese+bankruptcy+prediction

[17] R. A. A. Viadinugroho, "Imbalanced Classification in Python: SMOTE-ENN Method," Medium, Sep. 30, 2021. https://towardsdatascience.com/imbalanced-classification-in-python-smote-enn-method-db5db06b8d50

[18] "T-tests vs Chi-Square Tests: Statistical Testing in Practice," Dataheadhunters.com, Jan. 05, 2024. https://dataheadhunters.com/academy/t-tests-vs-chi-square-tests-statistical-testing-in-practice/

[19] Abid Ali Awan, "What is Bagging in Machine Learning? A Guide With Examples," *Datacamp.com*, Nov. 20, 2023. https://www.datacamp.com/tutorial/what-bagging-in-machine-learning-a-guide-with-examples

[20] "What is Boosting in Machine Learning?," *Enterprise AI*. https://www.techtarget.com/searchenterpriseai/feature/What-is-boosting-in-machine-learning

[21] "Stacking in Machine Learning – Javatpoint," *www.javatpoint.com*. https://www.javatpoint.com/stacking-in-machine-learning

[22] T. Tamplin, "Liquidity Ratio | Definition, Types, Applications, and Limitations," *Finance Strategist*, Jun. 27, 2023. https://www.financestrategists.com/wealth-management/accounting-ratios/liquidity-ratio/

[23] I. Emerling, "KEY FINANCIAL RATIOS TO ASSESS THE RISK OF BANKRUPTCY BASED ON SELECTED PUBLICLY TRADED COMPANIES," *SGEM International Multidisciplinary Scientific Conferences on Social Sciences and Arts*, Sep. 2014, doi: https://doi.org/10.5593/sgemsocial2014/b22/s6.049.

[24] Y. AKER and A. KARAVARDAR, "Using Machine Learning Methods in Financial Distress Prediction: Sample of Small and Medium Sized Enterprises Operating in Turkey," *Ege Akademik Bakis (Ege Academic Review)*, Jan. 2023, doi: https://doi.org/10.21121/eab.1027084.

[25] geeksforgeeks, "Understanding Logistic Regression," *GeeksforGeeks*, May 09, 2024. https://www.geeksforgeeks.org/understanding-logistic-regression/

[26] A. R. Rout, "Advantages and Disadvantages of Logistic Regression," *GeeksforGeeks*, Jan. 10, 2023. https://www.geeksforgeeks.org/advantages-and-disadvantages-of-logistic-regression/

[27] GeeksForGeeks, "Decision tree – geeksforgeeks," *GeeksforGeeks*, May 17, 2024. https://www.geeksforgeeks.org/decision-tree/

[28] CFI Team, "Financial ratios," *Corporate Finance Institute*, 2023. https://corporatefinanceinstitute.com/resources/accounting/financial-ratios/

[29] https://www.facebook.com/rajmaurya51, "What are capital structure ratios in accounting?," Fundamentals of Accounting, Nov. 23, 2020. https://fundamentalsofaccounting.org/what-are-capital-structure-ratios/#google_vignette (accessed May 07, 2025).

[30] S. Bragg, "AccountingTools," *AccountingTools*, Mar. 27, 2018. https://www.accountingtools.com/articles/what-is-a-turnover-ratio.html

[31] S. Bragg, "AccountingTools," *AccountingTools*, Apr. 02, 2018. https://www.accountingtools.com/articles/cash-flow-ratios.html

[32] J. Chen, "Growth Rates," *Investopedia*, Mar. 31, 2023. https://www.investopedia.com/terms/g/growthrates.asp

[33] D. Liang, C.-C. Lu, C.-F. Tsai, and G.-A. Shih, "Financial ratios and corporate governance indicators in bankruptcy prediction: A comprehensive study," *European Journal of Operational Research*, vol. 252, no. 2, pp. 561–572, Jul. 2016.

# Appendix

Edited_Preliminary_FYP.ipynb ☆ Saving...

File Edit View Insert Runtime Tools Help

Commands + Code + Text

```
[ ] !pip install -q kaggle

[ ] from google.colab import files
    files.upload()
```

```
Choose Files No file chosen  Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving kaggle.json to kaggle.json
{'kaggle.json': b'{"username":"phoebe03","key":"f7c294f72f45b7d1b3d0ec25a4f4ba9e"}'}
```

```
[ ] !mkdir ~/.kaggle
    !cp kaggle.json ~/.kaggle

[ ] !chmod 600 ~/.kaggle/kaggle.json

[ ] !kaggle datasets download -d fedesoriano/company-bankruptcy-prediction
    #Taiwan Economic Journal for the years 1999 to 2009
```

```
Dataset URL: https://www.kaggle.com/datasets/fedesoriano/company-bankruptcy-prediction
License(s): copyright-authors
```

```
[ ] !unzip /content/company-bankruptcy-prediction.zip -d /content/company_bankruptcy_pred.dataset
```

```
Archive:  /content/company-bankruptcy-prediction.zip
  inflating: /content/company_bankruptcy_pred.dataset/data.csv
```

```
[ ] import os
    import cv2 as cv
    import pandas as pd
    import matplotlib.pyplot as plt
    import numpy as np

[ ] df = pd.read_csv("/content/company_bankruptcy_pred.dataset/data.csv")
    print(df.shape[0])
```

```
6819
```

```
[ ] print(df.columns)
```

Show hidden output

```
[ ] print(df.info()) #numeric data -> in float, no categorical data
```

Show hidden output

```
[ ] print(df[' Net Income Flag'].value_counts())
```

Show hidden output

```
[ ] print(df[' Liability-Assets Flag'].value_counts())
```

Show hidden output

```
[ ] print(df['Bankrupt?'].value_counts()) #imbalance data

[ ] print(df.isnull().sum()) #no missing value
```

Show hidden output

```
[ ] df[' Net Income Flag'].value_counts() #remove redundant value
```

Show hidden output

```
[ ] X = df.drop(columns=['Bankrupt?',' Net Income Flag'])
    y = df['Bankrupt?']

[1] print(X.columns.value_counts())
```

Show hidden output

```
[ ] from imblearn.combine import SMOTEENN
    from imblearn.over_sampling import SMOTE
    from imblearn.under_sampling import EditedNearestNeighbours
    smote_enn = SMOTEENN(sampling_strategy='auto',
                         smote=SMOTE(random_state=42),  # Fix random_state for SMOTE
                         enn=EditedNearestNeighbours(),  # Default settings for ENN
                         random_state=42)  # Fix random_state for SMOTEENN

    X_resampled, y_resampled = smote_enn.fit_resample(X, y)
    y_resampled.value_counts()
```

Show hidden output

```
[ ] missing_data = X_resampled.isnull().sum()
    print(missing_data)
```

```python
from scipy.stats import ttest_ind

# Initialize a list to store p-values
p_values = []

# Loop through each feature to apply T-test
for feature in X_resampled.columns:
    # Split data into two groups based on the target
    group_0 = X_resampled[y_resampled == 0][feature]
    group_1 = X_resampled[y_resampled == 1][feature]

    # Perform T-test between the two groups
    _, p_value = ttest_ind(group_0, group_1)

    # Append the p-value for each feature
    p_values.append(p_value)

# Convert p-values into a DataFrame for better viewing
p_values_df = pd.DataFrame({
    'Feature': X_resampled.columns,
    'P-Value': p_values
}).sort_values(by='P-Value', ascending=True)

print(p_values_df)
```

Show hidden output

```python
# Select features with p-value less than the significance level (e.g., 0.05)
selected_features = p_values_df[p_values_df['P-Value'] < 0.05]
print("Selected Features based on T-test (p-value < 0.05):")
print(selected_features)
```

Show hidden output

```python
sorted_features = sorted(selected_features)
for index, row in selected_features.iterrows():
    print(f"Feature: {row['Feature']}, P-Value: {row['P-Value']}")
```

Show hidden output

```python
selected_columns= selected_features['Feature'].tolist()

columns_to_drop = [col for col in X_resampled.columns if col not in selected_columns]
X_resampled_selected = X_resampled.drop(columns=columns_to_drop)

print(X_resampled_selected.shape)
```

Show hidden output

```python
from sklearn.model_selection import train_test_split
#70% train 30% test
X_resampled_selected_train , X_resampled_selected_test , y_resampled_train ,  y_resampled_test = train_test_split(X_resampled_selected,y_resampled,test_size=0.3,random_state=42)
X_resampled_selected_train = X_resampled_selected_train.reset_index(drop=True)
y_resampled_train = y_resampled_train.reset_index(drop=True)
```

```python
from google.colab import drive
drive.mount('/content/drive')
```

Show hidden output

```python
train_data = pd.DataFrame(X_resampled_selected_train)
train_data['label'] = y_resampled_train

test_data = pd.DataFrame(X_resampled_selected_test)
test_data['label'] = y_resampled_test

train_csv_path =  '/content/drive/My Drive/train_data.csv'
test_csv_path =  '/content/drive/My Drive/test_data.csv'

train_data.to_csv(train_csv_path, index=False)
test_data.to_csv(test_csv_path, index=False)
print("Data saved successfully to Google Drive.")
```

Latest_Decision Tree - AdaBoosting.ipynb

File Edit View Insert Runtime Tools Help

mmands + Code + Text

```
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn import metrics
import time
import pandas as pd
```

```
from google.colab import drive
drive.mount('/content/drive')
```

Show hidden output

```
from pathlib import Path

# Specify the file path
train_csv_path =  '/content/drive/My Drive/train_data.csv'
test_csv_path =  '/content/drive/My Drive/test_data.csv'

# Check if the file exists using pathlib
train_file = Path(train_csv_path)
test_file = Path(test_csv_path)

if train_file.exists():
    print("train_data.csv exists.")
else:
    print("train_data.csv does not exist.")

if test_file.exists():
    print("test_data.csv exists.")
else:
    print("test_data.csv does not exist.")
```

```
train_data.csv exists.
test_data.csv exists.
```

```
import pandas as pd
test_data = pd.read_csv(test_csv_path)
train_data = pd.read_csv(train_csv_path)

print("test:",test_data['label'].value_counts())
print("train:",train_data['label'].value_counts())
```

Show hidden output

```
print(train_data.shape)
```

Show hidden output

```
X_train = train_data.drop(columns=['label'])
y_train = train_data['label']
```

```
[ ] X_train = train_data.drop(columns=['label'])
    y_train = train_data['label']

    X_test = test_data.drop(columns=['label'])
    y_test = test_data['label']
```

```
print(X_train.shape)
print(X_test.shape)
```

Show hidden output

```
[ ] #AdaBoosting

    from sklearn.model_selection import StratifiedKFold

    accuracies, precisions, recalls, f1_scores, aucs = [], [], [], [], []
    type_I_errors, type_II_errors = [],[]

    skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

    start_time = time.time() # start time

    decision_tree = DecisionTreeClassifier(criterion = "entropy", max_depth = 7,min_samples_split = 2,min_samples_leaf=4,random_state = 42)
    model = AdaBoostClassifier(estimator=decision_tree,n_estimators=10,learning_rate=0.5)

    for i in range(1, 301):

        fold_accuracies, fold_precisions, fold_recalls, fold_f1s, fold_aucs = [], [], [], [], []
        fold_type_I_errors, fold_type_II_errors = [], []

        for train_index, val_index in skf.split(X_train, y_train):
            X_train_fold, X_val_fold = X_train.iloc[train_index], X_train.iloc[val_index]
            y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.iloc[val_index]

            scaler = StandardScaler()
            X_train_scaled = scaler.fit_transform(X_train_fold)
            X_val_scaled = scaler.transform(X_val_fold)

            model.fit(X_train_scaled, y_train_fold)
            y_val_pred = model.predict(X_val_scaled)
            y_val_prob = model.predict_proba(X_val_scaled)[:, 1]

            accuracy = accuracy_score(y_val_fold, y_val_pred)
            precision = precision_score(y_val_fold, y_val_pred, pos_label=1)
            recall = recall_score(y_val_fold, y_val_pred, pos_label=1)
            f1 = f1_score(y_val_fold, y_val_pred, pos_label=1)
            auc = metrics.roc_auc_score(y_val_fold, y_val_prob)

            cm = confusion_matrix(y_val_fold, y_val_pred)
            FP, TN, FN, TP = cm[0][1], cm[0][0], cm[1][0], cm[1][1]
            type_I_error = FP / (FP + TN) if (FP + TN) != 0 else 0
            type_II_error = FN / (FN + TP) if (FN + TP) != 0 else 0

            # Append fold metrics
            fold_accuracies.append(accuracy)
            fold_precisions.append(precision)
            fold_recalls.append(recall)
            fold_f1s.append(f1)
            fold_aucs.append(auc)
            fold_type_I_errors.append(type_I_error)
            fold_type_II_errors.append(type_II_error)

        avg_acc = sum(fold_accuracies) / len(fold_accuracies)
        avg_precision = sum(fold_precisions) / len(fold_precisions)
        avg_recall = sum(fold_recalls) / len(fold_recalls)
        avg_f1 = sum(fold_f1s) / len(fold_f1s)
        avg_aucs = sum(fold_aucs) / len(fold_aucs)
        avg_typeI = sum(fold_type_I_errors) / len(fold_type_I_errors)
        avg_typeII = sum(fold_type_II_errors) / len(fold_type_II_errors)

        print(">>Fold",i)
        print("AUC:",avg_acc)
        print("Recall:",avg_recall)
        print("Type II:",avg_typeII)

        # Average over 5 folds
        accuracies.append(round(avg_acc, 4))
        precisions.append(round(avg_precision, 4))
        recalls.append(round(avg_recall, 4))
        f1_scores.append(round(avg_f1, 4))
        aucs.append(round(avg_aucs, 4))
        type_I_errors.append(round(avg_typeI, 4))
        type_II_errors.append(round(avg_typeII, 4))


        # Early stopping logic
        if len(accuracies) >= 5 and len(set(accuracies[-5:])) == 1:
            print(f"Early stopping at iteration {i} because accuracy hasn't changed for 5 iterations.")
            break
        elif len(recalls) >= 5 and len(set(recalls[-5:])) == 1:
            print(f"Early stopping at iteration {i} because recall hasn't changed for 5 iterations.")
            break

    end_time = time.time()
    duration = round(end_time - start_time, 2)
```

Show hidden output

```
print(f"Final Accuracy: {accuracies[-1]}")
print(f"Final Precision: {precisions[-1]}")
print(f"Final Recall: {recalls[-1]}")
print(f"Final F1 Score: {f1_scores[-1]}")
print(f"Final AUC: {aucs[-1]}")
print(f"Final Type I Error: {type_I_errors[-1]}")
print(f"Final Type II Error: {type_II_errors[-1]}")
print(f"Total training time: {duration} seconds.")
```

Show hidden output

```
from sklearn.inspection import permutation_importance
import scipy.stats as stats

# Compute permutation importance
perm_importance = permutation_importance(model, X_val_scaled, y_val_fold, n_repeats=10, random_state=42)

# Compute p-values using a permutation test
p_values = np.array([
    stats.norm.sf(abs(mean) / std) * 2 if std > 1e-10 else 1.0  # Normalized by standard deviation
    for mean, std in zip(perm_importance.importances_mean, perm_importance.importances_std)
])

importance_df = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': perm_importance.importances_mean,
    'P-Value': p_values
})

# Filter significant features (p-value < 0.05)
significant_features = importance_df[importance_df['P-Value'] < 0.05]
print(len(significant_features))
```

Show hidden output

```
significant_features.to_excel("DT_Adaboosting_significant_features.xlsx", index=False)

print(f"Significant features saved to 'DT_Adaboosting_significant_features.xlsx'.")
```

Show hidden output

```
Ada_Boosting_df = pd.read_excel("/content/DT_Adaboosting_significant_features.xlsx")
Stacking_df = pd.read_excel("/content/DT_Stacking_significant_features.xlsx")
Bagging_df = pd.read_excel("/content/DT_Bagging_significant_features.xlsx")


top_features_adaBoosting = Ada_Boosting_df['Feature'].tolist()
top_features_stacking = Stacking_df['Feature'].tolist()
top_features_bagging = Bagging_df['Feature'].tolist()


exact_matches = set(top_features_adaBoosting) & set(top_features_stacking) & set(top_features_bagging)
num_exact_matches = len(exact_matches)

# Display the matched features
print(f"Number of exact matches: {num_exact_matches}")
print("Matched features:")
print(list(exact_matches))
```

Show hidden output

```
[ ] # write the exact matches into a txt file
    exact_matches_list = list(exact_matches)
    with open("duplicates_decision_tree.txt", "w") as file:
        for item in exact_matches_list:
            file.write(item + "\n")
```

```
[ ] dt_combined_features = list(set(top_features_adaBoosting + top_features_stacking + top_features_bagging))
    print(len(dt_combined_features))
```

```
46
```

```
[ ] with open('combine_decision_tree.txt', 'w') as f:
        text=f.write(str(dt_combined_features))
```

Latest_Decision Tree - Bagging.ipynb

File Edit View Insert Runtime Tools Help

ommands + Code + Text

```
[ ] import numpy as np
    from sklearn.model_selection import KFold
    from sklearn.preprocessing import StandardScaler
    from sklearn.metrics import accuracy_score
    from sklearn.ensemble import BaggingClassifier
    from sklearn.tree import DecisionTreeClassifier
    from sklearn.metrics import confusion_matrix
    from sklearn.metrics import precision_score, recall_score, f1_score
    from sklearn import metrics
    import time
```

```
from google.colab import drive
drive.mount('/content/drive')
```

Show hidden output

```
[ ] from pathlib import Path

    # Specify the file path
    train_csv_path =  '/content/drive/My Drive/train_data.csv'
    test_csv_path =  '/content/drive/My Drive/test_data.csv'

    # Check if the file exists using pathlib
    train_file = Path(train_csv_path)
    test_file = Path(test_csv_path)

    if train_file.exists():
        print("train_data.csv exists.")
    else:
        print("train_data.csv does not exist.")

    if test_file.exists():
        print("test_data.csv exists.")
    else:
        print("test_data.csv does not exist.")
```

Show hidden output

```
[ ] import pandas as pd
    test_data = pd.read_csv(test_csv_path)
    train_data = pd.read_csv(train_csv_path)

    print("test:",test_data['label'].value_counts())
    print("train:",train_data['label'].value_counts())
```

Show hidden output

```
X_train = train_data.drop(columns=['label'])
y_train = train_data['label']

X_test = test_data.drop(columns=['label'])
y_test = test_data['label']
```

```
print(X_train.shape)
print(X_test.shape)
```

Show hidden output

```
from sklearn.model_selection import GridSearchCV
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [3, 5, 7],
    'min_samples_split': [2, 4, 6],
    'min_samples_leaf': [1, 2, 4]
}

grid_search = GridSearchCV(
    estimator=DecisionTreeClassifier(random_state=42),
    param_grid=param_grid,
    scoring='roc_auc',
    cv=5,
    n_jobs=-1
)
grid_search.fit(X_train, y_train)

print("Best params:", grid_search.best_params_)
```

Show hidden output

```
#Bagging
from sklearn.model_selection import StratifiedKFold

accuracies, precisions, recalls, f1_scores, aucs = [], [], [], [], []
type_I_errors, type_II_errors = [],[]

skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

start_time = time.time() # start time

decision_tree = DecisionTreeClassifier(criterion = "entropy",max_depth = 7,min_samples_split = 2,min_samples_leaf=4,random_state = 42)
model = BaggingClassifier(estimator=decision_tree, n_estimators=10, n_jobs=-1,random_state=42)

for i in range(1, 301):
    fold_accuracies, fold_precisions, fold_recalls, fold_f1s, fold_aucs = [], [], [], [], []
    fold_type_I_errors, fold_type_II_errors = [], []

    for train_index, val_index in skf.split(X_train, y_train):
        X_train_fold, X_val_fold = X_train.iloc[train_index], X_train.iloc[val_index]
        y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.iloc[val_index]
```

```
        scaler = StandardScaler()
        X_train_scaled = scaler.fit_transform(X_train_fold)
        X_val_scaled = scaler.transform(X_val_fold)

        model.fit(X_train_scaled, y_train_fold)
        y_val_pred = model.predict(X_val_scaled)
        y_val_prob = model.predict_proba(X_val_scaled)[:, 1]

        accuracy = accuracy_score(y_val_fold, y_val_pred)
        precision = precision_score(y_val_fold, y_val_pred, pos_label=1)
        recall = recall_score(y_val_fold, y_val_pred, pos_label=1)
        f1 = f1_score(y_val_fold, y_val_pred, pos_label=1)
        auc = metrics.roc_auc_score(y_val_fold, y_val_prob)

        cm = confusion_matrix(y_val_fold, y_val_pred)
        FP, TN, FN, TP = cm[0][1], cm[0][0], cm[1][0], cm[1][1]
        type_I_error = FP / (FP + TN) if (FP + TN) != 0 else 0
        type_II_error = FN / (FN + TP) if (FN + TP) != 0 else 0

        # Append fold metrics
        fold_accuracies.append(accuracy)
        fold_precisions.append(precision)
        fold_recalls.append(recall)
        fold_f1s.append(f1)
        fold_aucs.append(auc)
        fold_type_I_errors.append(type_I_error)
        fold_type_II_errors.append(type_II_error)

    avg_acc = sum(fold_accuracies) / len(fold_accuracies)
    avg_precision = sum(fold_precisions) / len(fold_precisions)
    avg_recall = sum(fold_recalls) / len(fold_recalls)
    avg_f1 = sum(fold_f1s) / len(fold_f1s)
    avg_aucs = sum(fold_aucs) / len(fold_aucs)
    avg_typeI = sum(fold_type_I_errors) / len(fold_type_I_errors)
    avg_typeII = sum(fold_type_II_errors) / len(fold_type_II_errors)

    print(">>Fold",i)
    print("AUC:",avg_acc)
    print("Recall:",avg_recall)
    print("Type II:",avg_typeII)

    # Average over 5 folds
    accuracies.append(round(avg_acc, 4))
    precisions.append(round(avg_precision, 4))
    recalls.append(round(avg_recall, 4))
    f1_scores.append(round(avg_f1, 4))
    aucs.append(round(avg_aucs, 4))
    type_I_errors.append(round(avg_typeI, 4))
    type_II_errors.append(round(avg_typeII, 4))


    # Early stopping logic
    if len(accuracies) >= 5 and len(set(accuracies[-5:])) == 1:
        print(f"Early stopping at iteration {i} because accuracy hasn't changed for 5 iterations.")
        break
    elif len(recalls) >= 5 and len(set(recalls[-5:])) == 1:
        print(f"Early stopping at iteration {i} because recall hasn't changed for 5 iterations.")
        break
```

```
end_time = time.time()
duration = round(end_time - start_time, 2)
```

Show hidden output

```
print(f"Final Accuracy: {accuracies[-1]}")
print(f"Final Precision: {precisions[-1]}")
print(f"Final Recall: {recalls[-1]}")
print(f"Final F1 Score: {f1_scores[-1]}")
print(f"Final AUC: {aucs[-1]}")
print(f"Final Type I Error: {type_I_errors[-1]}")
print(f"Final Type II Error: {type_II_errors[-1]}")
print(f"Total training time: {duration} seconds.")
```

Show hidden output

```
import matplotlib.pyplot as plt
iterations = len(recalls)

plt.plot(range(1, iterations + 1), type_II_errors, marker='o', color='b')
plt.title('False Negative vs Iteration')
plt.xlabel('Iteration (Epoch)')
plt.ylabel('False Negative')
plt.grid(True)
plt.show()
```

Show hidden output

```
# Make sure test data is scaled with the same scaler used on training data
X_test_scaled = scaler.transform(X_test)

# Predict once
y_test_pred = model.predict(X_test_scaled)
y_test_prob = model.predict_proba(X_test_scaled)[:, 1]

# Evaluate
accuracy = accuracy_score(y_test, y_test_pred)
precision = precision_score(y_test, y_test_pred, pos_label=1)
recall = recall_score(y_test, y_test_pred, pos_label=1)
f1 = f1_score(y_test, y_test_pred, pos_label=1)
auc = metrics.roc_auc_score(y_test, y_test_prob)

cm = confusion_matrix(y_test, y_test_pred)
FP, TN, FN, TP = cm[0][1], cm[0][0], cm[1][0], cm[1][1]
type_I_error = FP / (FP + TN) if (FP + TN) != 0 else 0
type_II_error = FN / (FN + TP) if (FN + TP) != 0 else 0

# Print final test performance
print(f"Final Accuracy: {accuracy:.4f}")
print(f"Final Precision: {precision:.4f}")
print(f"Final Recall: {recall:.4f}")
print(f"Final F1 Score: {f1:.4f}")
print(f"Final AUC: {auc:.4f}")
print(f"Final Type I Error: {type_I_error:.4f}")
print(f"Final Type II Error: {type_II_error:.4f}")
```

Show hidden output

```
from sklearn.inspection import permutation_importance
import scipy.stats as stats

# Compute permutation importance
perm_importance = permutation_importance(model, X_test_scaled, y_test, n_repeats=10, random_state=42)

# Compute p-values using a permutation test
p_values = np.array([
    stats.norm.sf(abs(mean) / std) * 2 if std > 1e-10 else 1.0  # Normalized by standard deviation
    for mean, std in zip(perm_importance.importances_mean, perm_importance.importances_std)
])

importance_df = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': perm_importance.importances_mean,
    'P-Value': p_values
})

# Filter significant features (p-value < 0.05)
significant_features = importance_df[importance_df['P-Value'] < 0.05]
print(len(significant_features))
```

```
26
```

```
[ ] significant_features.to_excel("DT_Bagging_significant_features.xlsx", index=False)

    print(f"Significant features saved to 'DT_Bagging_significant_features.xlsx'.")
```

```
Significant features saved to 'DT_Bagging_significant_features.xlsx'.
```

Latest_Decision Tree- Stacking.ipynb

File Edit View Insert Runtime Tools Help

nmands + Code + Text

```
[ ] #import libraries

    import numpy as np
    from sklearn.model_selection import KFold
    from sklearn.preprocessing import StandardScaler
    from sklearn.metrics import accuracy_score
    from sklearn.tree import DecisionTreeClassifier
    from sklearn.linear_model import LogisticRegression
    from sklearn.metrics import confusion_matrix
    from sklearn.metrics import precision_score, recall_score, f1_score
    from sklearn import metrics
    from sklearn.ensemble import StackingClassifier
    import time

    from sklearn.model_selection import GridSearchCV
```

```
[ ] from google.colab import drive
    drive.mount('/content/drive')
```

Show hidden output

```
[ ] from pathlib import Path

    # Specify the file path
    train_csv_path =  '/content/drive/My Drive/train_data.csv'
    test_csv_path =  '/content/drive/My Drive/test_data.csv'

    # Check if the file exists using pathlib
    train_file = Path(train_csv_path)
    test_file = Path(test_csv_path)

    if train_file.exists():
        print("train_data.csv exists.")
    else:
        print("train_data.csv does not exist.")

    if test_file.exists():
        print("test_data.csv exists.")
    else:
        print("test_data.csv does not exist.")
```

Show hidden output

```
[ ] import pandas as pd
    test_data = pd.read_csv(test_csv_path)
    train_data = pd.read_csv(train_csv_path)

    print("test:",test_data['label'].value_counts())
    print("train:",train_data['label'].value_counts())
```

Show hidden output

```
print(train_data.shape)
```

Show hidden output

```
[ ] X_train = train_data.drop(columns=['label'])
    y_train = train_data['label']

    X_test = test_data.drop(columns=['label'])
    y_test = test_data['label']
```

```
[ ] print(X_train.shape)
    print(X_test.shape)
```

Show hidden output

```
# Stacking
from sklearn.model_selection import StratifiedKFold

accuracies, precisions, recalls, f1_scores, aucs = [], [], [], [], []
type_I_errors, type_II_errors = [],[]

skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

start_time = time.time() # start time
base_estimators =[ (f'DecisionTree{i}',DecisionTreeClassifier(criterion = "entropy", max_depth = 7,min_samples_split = 2,min_samples_leaf=4,random_state = 42)) for i in range (10)]
model = StackingClassifier(estimators=base_estimators,final_estimator=LogisticRegression())


for i in range(1, 301):
    fold_accuracies, fold_precisions, fold_recalls, fold_f1s, fold_aucs = [], [], [], [], []
    fold_type_I_errors, fold_type_II_errors = [], []

    for train_index, val_index in skf.split(X_train, y_train):
        X_train_fold, X_val_fold = X_train.iloc[train_index], X_train.iloc[val_index]
        y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.iloc[val_index]

        scaler = StandardScaler()
        X_train_scaled = scaler.fit_transform(X_train_fold)
        X_val_scaled = scaler.transform(X_val_fold)

        model.fit(X_train_scaled, y_train_fold)
        y_val_pred = model.predict(X_val_scaled)
        y_val_prob = model.predict_proba(X_val_scaled)[:, 1]

        accuracy = accuracy_score(y_val_fold, y_val_pred)
        precision = precision_score(y_val_fold, y_val_pred, pos_label=1)
        recall = recall_score(y_val_fold, y_val_pred, pos_label=1)
        f1 = f1_score(y_val_fold, y_val_pred, pos_label=1)
        auc = metrics.roc_auc_score(y_val_fold, y_val_prob)

        cm = confusion_matrix(y_val_fold, y_val_pred)
        FP, TN, FN, TP = cm[0][1], cm[0][0], cm[1][0], cm[1][1]
        type_I_error = FP / (FP + TN) if (FP + TN) != 0 else 0
        type_II_error = FN / (FN + TP) if (FN + TP) != 0 else 0

        # Append fold metrics
        fold_accuracies.append(accuracy)
        fold_precisions.append(precision)
        fold_recalls.append(recall)
        fold_f1s.append(f1)
        fold_aucs.append(auc)
        fold_type_I_errors.append(type_I_error)
        fold_type_II_errors.append(type_II_error)

    avg_acc = sum(fold_accuracies) / len(fold_accuracies)
    avg_precision = sum(fold_precisions) / len(fold_precisions)
    avg_recall = sum(fold_recalls) / len(fold_recalls)
    avg_f1 = sum(fold_f1s) / len(fold_f1s)
    avg_aucs = sum(fold_aucs) / len(fold_aucs)
    avg_typeI = sum(fold_type_I_errors) / len(fold_type_I_errors)
    avg_typeII = sum(fold_type_II_errors) / len(fold_type_II_errors)

    print(">>Fold",i)
    print("AUC:",avg_acc)
    print("Recall:",avg_recall)
    print("Type II:",avg_typeII)

    # Average over 5 folds
    accuracies.append(round(avg_acc, 4))
    precisions.append(round(avg_precision, 4))
    recalls.append(round(avg_recall, 4))
    f1_scores.append(round(avg_f1, 4))
    aucs.append(round(avg_aucs, 4))
    type_I_errors.append(round(avg_typeI, 4))
    type_II_errors.append(round(avg_typeII, 4))
```

```
        # Early stopping logic
        if len(accuracies) >= 5 and len(set(accuracies[-5:])) == 1:
            print(f"Early stopping at iteration {i} because accuracy hasn't changed for 5 iterations.")
            break
        elif len(recalls) >= 5 and len(set(recalls[-5:])) == 1:
            print(f"Early stopping at iteration {i} because recall hasn't changed for 5 iterations.")
            break

    end_time = time.time()
    duration = round(end_time - start_time, 2)
```

Show hidden output

```
print(f"Final Accuracy: {accuracies[-1]}")
print(f"Final Precision: {precisions[-1]}")
print(f"Final Recall: {recalls[-1]}")
print(f"Final F1 Score: {f1_scores[-1]}")
print(f"Final AUC: {aucs[-1]}")
print(f"Final Type I Error: {type_I_errors[-1]}")
print(f"Final Type II Error: {type_II_errors[-1]}")
print(f"Total training time: {duration} seconds.")
```

Show hidden output

```
import matplotlib.pyplot as plt
iterations = len(recalls)

plt.plot(range(1, iterations + 1), type_II_errors, marker='o', color='b')
plt.title('False Negative vs Iteration')
plt.xlabel('Iteration (Epoch)')
plt.ylabel('False Negative')
plt.grid(True)
plt.show()
```

```
# Make sure test data is scaled with the same scaler used on training data
X_test_scaled = scaler.transform(X_test)

# Predict once
y_test_pred = model.predict(X_test_scaled)
y_test_prob = model.predict_proba(X_test_scaled)[:, 1]

# Evaluate
accuracy = accuracy_score(y_test, y_test_pred)
precision = precision_score(y_test, y_test_pred, pos_label=1)
recall = recall_score(y_test, y_test_pred, pos_label=1)
f1 = f1_score(y_test, y_test_pred, pos_label=1)
auc = metrics.roc_auc_score(y_test, y_test_prob)

cm = confusion_matrix(y_test, y_test_pred)
FP, TN, FN, TP = cm[0][1], cm[0][0], cm[1][0], cm[1][1]
type_I_error = FP / (FP + TN) if (FP + TN) != 0 else 0
type_II_error = FN / (FN + TP) if (FN + TP) != 0 else 0

# Print final test performance
print(f"Test Accuracy: {accuracy:.4f}")
print(f"Test Precision: {precision:.4f}")
print(f"Test Recall: {recall:.4f}")
print(f"Test F1 Score: {f1:.4f}")
print(f"Test AUC: {auc:.4f}")
print(f"Test Type I Error: {type_I_error:.4f}")
print(f"Test Type II Error: {type_II_error:.4f}")
```

Show hidden output

```
from sklearn.inspection import permutation_importance
import scipy.stats as stats

# Compute permutation importance
perm_importance = permutation_importance(model,X_test_scaled, y_test, n_repeats=10, random_state=42)

# Compute p-values using a permutation test
p_values = np.array([
    stats.norm.sf(abs(mean) / std) * 2 if std > 1e-10 else 1.0  # Normalized by standard deviation
    for mean, std in zip(perm_importance.importances_mean, perm_importance.importances_std)
])

importance_df = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': perm_importance.importances_mean,
    'P-Value': p_values
})
```

```
# Filter significant features (p-value < 0.05)
significant_features = importance_df[importance_df['P-Value'] < 0.05]
print(len(significant_features))
```

Show hidden output

```
significant_features.to_excel("DT_Stacking_significant_features.xlsx", index=False)

print(f"Significant features saved to 'DT_Stacking_significant_features.xlsx'.")
```

Show hidden output

Latest_LR- Stacking.ipynb

File Edit View Insert Runtime Tools Help

nmands + Code + Text

```
[ ]  #import libraries

     import numpy as np
     from sklearn.preprocessing import StandardScaler
     from sklearn.metrics import accuracy_score
     from sklearn.linear_model import LogisticRegression
     from sklearn.metrics import confusion_matrix
     from sklearn.metrics import precision_score, recall_score, f1_score
     from sklearn import metrics
     from sklearn.ensemble import StackingClassifier
     import time
     from sklearn.model_selection import GridSearchCV
```

```
[ ]  from google.colab import drive
     drive.mount('/content/drive')
```

Show hidden output

```
[ ]  from pathlib import Path

     # Specify the file path
     train_csv_path =  '/content/drive/My Drive/train_data.csv'
     test_csv_path =  '/content/drive/My Drive/test_data.csv'

     # Check if the file exists using pathlib
     train_file = Path(train_csv_path)
     test_file = Path(test_csv_path)

     if train_file.exists():
         print("train_data.csv exists.")
     else:
         print("train_data.csv does not exist.")

     if test_file.exists():
         print("test_data.csv exists.")
     else:
         print("test_data.csv does not exist.")
```

Show hidden output

```
[ ]  import pandas as pd
     test_data = pd.read_csv(test_csv_path)
     train_data = pd.read_csv(train_csv_path)

     print("test:",test_data['label'].value_counts())
     print("train:",train_data['label'].value_counts())
```

Show hidden output

```
print(train_data.shape)
```

Show hidden output

```
X_train = train_data.drop(columns=['label'])
y_train = train_data['label']

X_test = test_data.drop(columns=['label'])
y_test = test_data['label']
```

```
print(X_train.shape)
print(X_test.shape)
```

Show hidden output

```python
#Stacking
from sklearn.model_selection import StratifiedKFold

accuracies, precisions, recalls, f1_scores, aucs = [], [], [], [], []
type_I_errors, type_II_errors = [],[]

skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
base_estimators = [(f'logreg{i}', LogisticRegression(solver='lbfgs',max_iter=200,class_weight="balanced", random_state=42)) for i in range (10)]
model = StackingClassifier(estimators=base_estimators,final_estimator=LogisticRegression())

start_time = time.time() # start time



for i in range(1, 301):
    fold_accuracies, fold_precisions, fold_recalls, fold_f1s, fold_aucs = [], [], [], [], []
    fold_type_I_errors, fold_type_II_errors = [], []

    for train_index, val_index in skf.split(X_train, y_train):
        X_train_fold, X_val_fold = X_train.iloc[train_index], X_train.iloc[val_index]
        y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.iloc[val_index]

        scaler = StandardScaler()
        X_train_scaled = scaler.fit_transform(X_train_fold)
        X_val_scaled = scaler.transform(X_val_fold)

        model.fit(X_train_scaled, y_train_fold)
        y_val_pred = model.predict(X_val_scaled)
        y_val_prob = model.predict_proba(X_val_scaled)[:, 1]

        accuracy = accuracy_score(y_val_fold, y_val_pred)
        precision = precision_score(y_val_fold, y_val_pred, pos_label=1)
        recall = recall_score(y_val_fold, y_val_pred, pos_label=1)
        f1 = f1_score(y_val_fold, y_val_pred, pos_label=1)
        auc = metrics.roc_auc_score(y_val_fold, y_val_prob)

        cm = confusion_matrix(y_val_fold, y_val_pred)
        FP, TN, FN, TP = cm[0][1], cm[0][0], cm[1][0], cm[1][1]
        type_I_error = FP / (FP + TN) if (FP + TN) != 0 else 0
        type_II_error = FN / (FN + TP) if (FN + TP) != 0 else 0

        # Append fold metrics
        fold_accuracies.append(accuracy)
        fold_precisions.append(precision)
        fold_recalls.append(recall)
        fold_f1s.append(f1)
        fold_aucs.append(auc)
        fold_type_I_errors.append(type_I_error)
        fold_type_II_errors.append(type_II_error)

    avg_acc = sum(fold_accuracies) / len(fold_accuracies)
    avg_precision = sum(fold_precisions) / len(fold_precisions)
    avg_recall = sum(fold_recalls) / len(fold_recalls)
    avg_f1 = sum(fold_f1s) / len(fold_f1s)
    avg_aucs = sum(fold_aucs) / len(fold_aucs)
    avg_typeI = sum(fold_type_I_errors) / len(fold_type_I_errors)
    avg_typeII = sum(fold_type_II_errors) / len(fold_type_II_errors)

    print(">>Fold",i)
    print("AUC:",avg_acc)
    print("Recall:",avg_recall)
    print("Type II:",avg_typeII)

    # Average over 5 folds
    accuracies.append(round(avg_acc, 4))
    precisions.append(round(avg_precision, 4))
    recalls.append(round(avg_recall, 4))
    f1_scores.append(round(avg_f1, 4))
    aucs.append(round(avg_aucs, 4))
    type_I_errors.append(round(avg_typeI, 4))
    type_II_errors.append(round(avg_typeII, 4))


    # Early stopping logic
    if len(accuracies) >= 5 and len(set(accuracies[-5:])) == 1:
        print(f"Early stopping at iteration {i} because accuracy hasn't changed for 5 iterations.")
        break
    elif len(recalls) >= 5 and len(set(recalls[-5:])) == 1:
        print(f"Early stopping at iteration {i} because recall hasn't changed for 5 iterations.")
        break
```

```
end_time = time.time()
duration = round(end_time - start_time, 2)
```

Show hidden output

```
import matplotlib.pyplot as plt
iterations = len(accuracies)

plt.plot(range(1, iterations + 1), accuracies, marker='o', color='b')
plt.title('Accuracies vs Iteration')
plt.xlabel('Iteration (Epoch)')
plt.ylabel('Acccuracies')
plt.grid(True)
plt.show()
```

Show hidden output

```
print(f"Final Accuracy: {accuracies[-1]}")
print(f"Final Precision: {precisions[-1]}")
print(f"Final Recall: {recalls[-1]}")
print(f"Final F1 Score: {f1_scores[-1]}")
print(f"Final AUC: {aucs[-1]}")
print(f"Final Type I Error: {type_I_errors[-1]}")
print(f"Final Type II Error: {type_II_errors[-1]}")
print(f"Total training time: {duration} seconds.")
```

Show hidden output

```
# Make sure test data is scaled with the same scaler used on training data
X_test_scaled = scaler.transform(X_test)

# Predict once
y_test_pred = model.predict(X_test_scaled)
y_test_prob = model.predict_proba(X_test_scaled)[:, 1]

# Evaluate
accuracy = accuracy_score(y_test, y_test_pred)
precision = precision_score(y_test, y_test_pred, pos_label=1)
recall = recall_score(y_test, y_test_pred, pos_label=1)
f1 = f1_score(y_test, y_test_pred, pos_label=1)
auc = metrics.roc_auc_score(y_test, y_test_prob)

cm = confusion_matrix(y_test, y_test_pred)
FP, TN, FN, TP = cm[0][1], cm[0][0], cm[1][0], cm[1][1]
type_I_error = FP / (FP + TN) if (FP + TN) != 0 else 0
type_II_error = FN / (FN + TP) if (FN + TP) != 0 else 0

# Print final test performance
print(f"Test Accuracy: {accuracy:.4f}")
print(f"Test Precision: {precision:.4f}")
print(f"Test Recall: {recall:.4f}")
print(f"Test F1 Score: {f1:.4f}")
print(f"Test AUC: {auc:.4f}")
print(f"Test Type I Error: {type_I_error:.4f}")
print(f"Test Type II Error: {type_II_error:.4f}")
```

Show hidden output

```
from sklearn.inspection import permutation_importance
import scipy.stats as stats

# Compute permutation importance
perm_importance = permutation_importance(model, X_test_scaled, y_test, n_repeats=10, random_state=42)

# Compute p-values using a permutation test
p_values = np.array([
    stats.norm.sf(abs(mean) / std) * 2 if std > 1e-10 else 1.0  # Normalized by standard deviation
    for mean, std in zip(perm_importance.importances_mean, perm_importance.importances_std)
])

importance_df = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': perm_importance.importances_mean,
    'P-Value': p_values
})

# Filter significant features (p-value < 0.05)
significant_features = importance_df[importance_df['P-Value'] < 0.05]
print(len(significant_features))
```

Show hidden output

```
significant_features.to_excel("LR_Stacking_significant_features.xlsx", index=False)

print(f"Significant features saved to 'LR- Stacking_significant_features.xlsx'.")
```

Show hidden output

Latest_LR- Bagging.ipynb

le Edit View Insert Runtime Tools Help

nds + Code + Text

```
#import libraries

import numpy as np
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn import metrics
from sklearn.ensemble import BaggingClassifier
import time
from sklearn.model_selection import GridSearchCV
```

```
from google.colab import drive
drive.mount('/content/drive')
```

Show hidden output

```
from pathlib import Path

# Specify the file path
train_csv_path =  '/content/drive/My Drive/train_data.csv'
test_csv_path =  '/content/drive/My Drive/test_data.csv'

# Check if the file exists using pathlib
train_file = Path(train_csv_path)
test_file = Path(test_csv_path)

if train_file.exists():
    print("train_data.csv exists.")
else:
    print("train_data.csv does not exist.")

if test_file.exists():
    print("test_data.csv exists.")
else:
    print("test_data.csv does not exist.")
```

Show hidden output

```
import pandas as pd
test_data = pd.read_csv(test_csv_path)
train_data = pd.read_csv(train_csv_path)

print("test:",test_data['label'].value_counts())
print("train:",train_data['label'].value_counts())
```

Show hidden output

```
print(train_data.shape)
```

Show hidden output

```
X_train = train_data.drop(columns=['label'])
y_train = train_data['label']

X_test = test_data.drop(columns=['label'])
y_test = test_data['label']
```

```
print(X_train.shape)
print(X_test.shape)
```

Show hidden output

```
#Bagging
from sklearn.model_selection import StratifiedKFold

accuracies, precisions, recalls, f1_scores, aucs = [], [], [], [], []
type_I_errors, type_II_errors = [],[]

skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

start_time = time.time() # start time
```

```
logreg = LogisticRegression(solver='lbfgs',max_iter=200,class_weight="balanced",random_state=42)
model = BaggingClassifier(estimator=logreg, n_estimators=10,random_state=42)


for i in range(1, 301):
    fold_accuracies, fold_precisions, fold_recalls, fold_f1s, fold_aucs = [], [], [], [], []
    fold_type_I_errors, fold_type_II_errors = [], []

    for train_index, val_index in skf.split(X_train, y_train):
        X_train_fold, X_val_fold = X_train.iloc[train_index], X_train.iloc[val_index]
        y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.iloc[val_index]

        scaler = StandardScaler()
        X_train_scaled = scaler.fit_transform(X_train_fold)
        X_val_scaled = scaler.transform(X_val_fold)

        model.fit(X_train_scaled, y_train_fold)
        y_val_pred = model.predict(X_val_scaled)
        y_val_prob = model.predict_proba(X_val_scaled)[:, 1]

        accuracy = accuracy_score(y_val_fold, y_val_pred)
        precision = precision_score(y_val_fold, y_val_pred, pos_label=1)
        recall = recall_score(y_val_fold, y_val_pred, pos_label=1)
        f1 = f1_score(y_val_fold, y_val_pred, pos_label=1)
        auc = metrics.roc_auc_score(y_val_fold, y_val_prob)

        cm = confusion_matrix(y_val_fold, y_val_pred)
        FP, TN, FN, TP = cm[0][1], cm[0][0], cm[1][0], cm[1][1]
        type_I_error = FP / (FP + TN) if (FP + TN) != 0 else 0
        type_II_error = FN / (FN + TP) if (FN + TP) != 0 else 0

        # Append fold metrics
        fold_accuracies.append(accuracy)
        fold_precisions.append(precision)
        fold_recalls.append(recall)
        fold_f1s.append(f1)
        fold_aucs.append(auc)
        fold_type_I_errors.append(type_I_error)
        fold_type_II_errors.append(type_II_error)

    avg_acc = sum(fold_accuracies) / len(fold_accuracies)
    avg_precision = sum(fold_precisions) / len(fold_precisions)
    avg_recall = sum(fold_recalls) / len(fold_recalls)
    avg_f1 = sum(fold_f1s) / len(fold_f1s)
    avg_aucs = sum(fold_aucs) / len(fold_aucs)
    avg_typeI = sum(fold_type_I_errors) / len(fold_type_I_errors)
    avg_typeII = sum(fold_type_II_errors) / len(fold_type_II_errors)

    print(">>Fold",i)
    print("AUC:",avg_acc)
    print("Recall:",avg_recall)
    print("Type II:",avg_typeII)

    # Average over 5 folds
    accuracies.append(round(avg_acc, 4))
    precisions.append(round(avg_precision, 4))
    recalls.append(round(avg_recall, 4))
    f1_scores.append(round(avg_f1, 4))
    aucs.append(round(avg_aucs, 4))
    type_I_errors.append(round(avg_typeI, 4))
    type_II_errors.append(round(avg_typeII, 4))


    # Early stopping logic
    if len(accuracies) >= 5 and len(set(accuracies[-5:])) == 1:
        print(f"Early stopping at iteration {i} because accuracy hasn't changed for 5 iterations.")
        break
    elif len(recalls) >= 5 and len(set(recalls[-5:])) == 1:
        print(f"Early stopping at iteration {i} because recall hasn't changed for 5 iterations.")
        break

end_time = time.time()
duration = round(end_time - start_time, 2)
```

Show hidden output

```
print(f"Final Accuracy: {accuracies[-1]}")
print(f"Final Precision: {precisions[-1]}")
print(f"Final Recall: {recalls[-1]}")
print(f"Final F1 Score: {f1_scores[-1]}")
print(f"Final AUC: {aucs[-1]}")
print(f"Final Type I Error: {type_I_errors[-1]}")
print(f"Final Type II Error: {type_II_errors[-1]}")
print(f"Total training time: {duration} seconds.")
```

Show hidden output

```
# Make sure test data is scaled with the same scaler used on training data
X_test_scaled = scaler.transform(X_test)

# Predict once
y_test_pred = model.predict(X_test_scaled)
y_test_prob = model.predict_proba(X_test_scaled)[:, 1]

# Evaluate
accuracy = accuracy_score(y_test, y_test_pred)
precision = precision_score(y_test, y_test_pred, pos_label=1)
recall = recall_score(y_test, y_test_pred, pos_label=1)
f1 = f1_score(y_test, y_test_pred, pos_label=1)
auc = metrics.roc_auc_score(y_test, y_test_prob)

cm = confusion_matrix(y_test, y_test_pred)
FP, TN, FN, TP = cm[0][1], cm[0][0], cm[1][0], cm[1][1]
type_I_error = FP / (FP + TN) if (FP + TN) != 0 else 0
type_II_error = FN / (FN + TP) if (FN + TP) != 0 else 0

# Print final test performance
print(f"Final Accuracy: {accuracy:.4f}")
print(f"Final Precision: {precision:.4f}")
print(f"Final Recall: {recall:.4f}")
print(f"Final F1 Score: {f1:.4f}")
print(f"Final AUC: {auc:.4f}")
print(f"Final Type I Error: {type_I_error:.4f}")
print(f"Final Type II Error: {type_II_error:.4f}")
```

```
Final Accuracy: 0.9090
Final Precision: 0.8938
Final Recall: 0.9194
Final F1 Score: 0.9064
Final AUC: 0.9674
Final Type I Error: 0.1005
Final Type II Error: 0.0806
```

```
from sklearn.inspection import permutation_importance
import scipy.stats as stats

# Compute permutation importance
perm_importance = permutation_importance(model, X_test_scaled, y_test, n_repeats=10, random_state=42)

# Compute p-values using a permutation test
p_values = np.array([
    stats.norm.sf(abs(mean) / std) * 2 if std > 1e-10 else 1.0  # Normalized by standard deviation
    for mean, std in zip(perm_importance.importances_mean, perm_importance.importances_std)
])

importance_df = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': perm_importance.importances_mean,
    'P-Value': p_values
})

# Filter significant features (p-value < 0.05)
significant_features = importance_df[importance_df['P-Value'] < 0.05]
print(len(significant_features))
```

Show hidden output

```
significant_features.to_excel("LR_Bagging_significant_features.xlsx", index=False)

print(f"Significant features saved to 'LR- Bagging_significant_features.xlsx'.")
```

Show hidden output

Final training_classfier.ipynb

File Edit View Insert Runtime Tools Help

nmands + Code + Text

```
#import libraries
from sklearn import svm
import pandas as pd
import numpy as np
from imblearn.combine import SMOTEENN
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import EditedNearestNeighbours

from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_score
```

```
[ ] !pip install -q kaggle
```

```
[ ] from google.colab import files
    files.upload()
```

```
Choose Files No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving kaggle.json to kaggle.json
{'kaggle.json': b'{"username":"phoebe03","key":"f7c294f72f45b7d1b3d0ec25a4f4ba9e"}'}
```

```
[ ] !mkdir ~/.kaggle
    !cp kaggle.json ~/.kaggle
```

```
[ ] !kaggle datasets download -d fedesoriano/company-bankruptcy-prediction
    #Taiwan Economic Journal for the years 1999 to 2009
```

```
Warning: Your Kaggle API key is readable by other users on this system! To fix this, you can run 'chmod 600 /root/.kaggle/kaggle.json'
Dataset URL: https://www.kaggle.com/datasets/fedesoriano/company-bankruptcy-prediction
License(s): copyright-authors
```

```
[ ] !unzip /content/company-bankruptcy-prediction.zip -d /content/company_bankruptcy_pred.dataset
```

```
Archive:  /content/company-bankruptcy-prediction.zip
  inflating: /content/company_bankruptcy_pred.dataset/data.csv
```

```
[ ] bankruptcy_dataset_df = pd.read_csv("/content/company_bankruptcy_pred.dataset/data.csv")
    print(bankruptcy_dataset_df.shape[0])
```

```
6819
```

```
[ ] X = bankruptcy_dataset_df.drop(columns=['Bankrupt?',' Net Income Flag'])
    y = bankruptcy_dataset_df['Bankrupt?']
```

```
[ ] smote_enn = SMOTEENN(sampling_strategy='auto',
                         smote=SMOTE(random_state=42),  # Fix random_state for SMOTE
                         enn=EditedNearestNeighbours(),  # Default settings for ENN
                         random_state=42)  # Fix random_state for SMOTEENN

    X_resampled, y_resampled = smote_enn.fit_resample(X, y)
    y_resampled.value_counts()
```

| Bankrupt? | count |
|---|---|
| 0 | 6599 |
| 1 | 6258 |

dtype: int64

```
[ ] Lr_adaboosting_df = pd.read_excel('/content/LR_AdaBoosting_significant_features.xlsx')
    selected_Lr_adaboosting=Lr_adaboosting_df['Feature'].tolist()

    columns_to_drop = [col for col in X_resampled.columns if col not in selected_Lr_adaboosting]
    X_resampled_selected_lr_adaboosting = X_resampled.drop(columns=columns_to_drop)
    print(X_resampled_selected_lr_adaboosting.shape)
```

```
(12857, 14)
```

```
Lr_stacking_df = pd.read_excel('/content/LR_Stacking_significant_features.xlsx')
selected_Lr_stacking=Lr_stacking_df['Feature'].tolist()

columns_to_drop = [col for col in X_resampled.columns if col not in selected_Lr_stacking]
X_resampled_selected_lr_stacking = X_resampled.drop(columns=columns_to_drop)
print(X_resampled_selected_lr_stacking.shape)
```

```
(12857, 32)
```

```
Lr_bagging_df = pd.read_excel('/content/LR_Bagging_significant_features.xlsx')
selected_Lr_bagging=Lr_bagging_df['Feature'].tolist()

columns_to_drop = [col for col in X_resampled.columns if col not in selected_Lr_bagging]
X_resampled_selected_lr_bagging = X_resampled.drop(columns=columns_to_drop)
print(X_resampled_selected_lr_bagging.shape)
```

```
(12857, 29)
```

```
with open('duplicates_logistic_reg.txt', 'r') as f:
   text=f.read()
```

```
columns_to_drop = [col for col in X_resampled.columns if col not in text]
X_resampled_selected_lr_similar = X_resampled.drop(columns=columns_to_drop)
print(X_resampled_selected_lr_similar.shape)
```

```
(12857, 7)
```

```
with open('duplicates_decision_tree.txt', 'r') as f:
   text2=f.read()
```

```
columns_to_drop = [col for col in X_resampled.columns if col not in text2]
X_resampled_selected_dt_similar = X_resampled.drop(columns=columns_to_drop)
print(X_resampled_selected_dt_similar.shape)
```

```
(12857, 10)
```

```
Dt_adaboosting_df = pd.read_excel('/content/DT_Adaboosting_significant_features.xlsx')
selected_Dt_adaboosting=Dt_adaboosting_df['Feature'].tolist()

columns_to_drop = [col for col in X_resampled.columns if col not in selected_Dt_adaboosting]
X_resampled_selected_dt_adaboosting = X_resampled.drop(columns=columns_to_drop)
print(X_resampled_selected_dt_adaboosting.shape)
```

```
(12857, 26)
```

```
Dt_bagging_df = pd.read_excel('/content/DT_Bagging_significant_features.xlsx')
selected_Dt_bagging=Dt_bagging_df['Feature'].tolist()

columns_to_drop = [col for col in X_resampled.columns if col not in selected_Dt_bagging]
X_resampled_selected_dt_bagging= X_resampled.drop(columns=columns_to_drop)
print(X_resampled_selected_dt_bagging.shape)
```

```
(12857, 26)
```

```
Dt_stacking_df = pd.read_excel('/content/DT_Stacking_significant_features.xlsx')
selected_Dt_stacking=Dt_stacking_df['Feature'].tolist()

columns_to_drop = [col for col in X_resampled.columns if col not in selected_Dt_stacking]
X_resampled_selected_dt_stacking = X_resampled.drop(columns=columns_to_drop)
print(X_resampled_selected_dt_stacking.shape)
```

```
(12857, 29)
```

```
with open('combine_logistic_reg.txt', 'r') as f:
   text3=f.read()
```

```
columns_to_drop = [col for col in X_resampled.columns if col not in text3]
X_resampled_selected_lr_combine = X_resampled.drop(columns=columns_to_drop)
print(X_resampled_selected_lr_combine.shape)
```

```
with open('combine_decision_tree.txt', 'r') as f:
   text4=f.read()
```

```
columns_to_drop = [col for col in X_resampled.columns if col not in text4]
X_resampled_selected_dt_combine = X_resampled.drop(columns=columns_to_drop)
print(X_resampled_selected_dt_combine.shape)
```

```
(12857, 46)
```

```
selected_combine_bagging = list(set(X_resampled_selected_dt_bagging + X_resampled_selected_lr_bagging))

columns_to_drop = [col for col in X_resampled.columns if col not in selected_combine_bagging]
X_resampled_combine_bagging= X_resampled.drop(columns=columns_to_drop)
print(X_resampled_combine_bagging.shape)
```

```
(12857, 47)
```

```
with open("Combine_bagging.txt", "w") as f:
    for item in selected_combine_bagging:
        f.write(item + "\n")
```

```
from sklearn.model_selection import train_test_split
#Full dataset 70% train 30% test
X_resampled_train , X_resampled_test , y_resampled_train ,  y_resampled_test = train_test_split(X_resampled,y_resampled,test_size=0.3,random_state=42)
X_resampled_selected_train = X_resampled_train.reset_index(drop=True)
y_resampled_train = y_resampled_train.reset_index(drop=True)

#Similar dataset (DT)
X_resampled_selected_dt_similar_train, X_resampled_selected_dt_similar_test , y_resampled_selected_dt_similar_train , y_resampled_selected_dt_similar_test = train_test_split(X_resampled_selected_dt_similar,y_resampled,test_size=0.3,random_state=42)
X_resampled_selected_dt_similar_train = X_resampled_selected_dt_similar_train.reset_index(drop=True)
y_resampled_selected_dt_similar_train = y_resampled_selected_dt_similar_train.reset_index(drop=True)

#Similar dataset (LR)
X_resampled_selected_lr_similar_train, X_resampled_selected_lr_similar_test , y_resampled_selected_lr_similar_train , y_resampled_selected_lr_similar_test = train_test_split(X_resampled_selected_lr_similar,y_resampled,test_size=0.3,random_state=42)
X_resampled_selected_lr_similar_train = X_resampled_selected_lr_similar_train.reset_index(drop=True)
y_resampled_selected_lr_similar_train = y_resampled_selected_lr_similar_train.reset_index(drop=True)

#Combine feature(dt)
X_combined_features_dt_train , X_combined_features_dt_test, y_combined_features_dt_train , y_combined_features_dt_test = train_test_split(X_resampled_selected_dt_combine,y_resampled,test_size=0.3,random_state=42)
X_combined_features_dt_train = X_combined_features_dt_train.reset_index(drop=True)
y_combined_features_dt_train = y_combined_features_dt_train.reset_index(drop=True)

#Combine feature(lr)
X_combined_features_lr_train , X_combined_features_lr_test, y_combined_features_lr_train , y_combined_features_lr_test = train_test_split(X_resampled_selected_lr_combine,y_resampled,test_size=0.3,random_state=42)
X_combined_features_lr_train = X_combined_features_lr_train.reset_index(drop=True)
y_combined_features_lr_train = y_combined_features_lr_train.reset_index(drop=True)

# DECISION TREE
#70% train 30% test
X_resampled_selected_dt_stacking_train , X_resampled_selected_dt_stacking_test , y_resampled_selected_dt_stacking_train ,  y_resampled_selected_dt_stacking_test = train_test_split(X_resampled_selected_dt_stacking,y_resampled,test_size=0.3,random_state=42)
X_resampled_selected_dt_stacking_train = X_resampled_selected_dt_stacking_train.reset_index(drop=True)
y_resampled_selected_dt_stacking_train = y_resampled_selected_dt_stacking_train.reset_index(drop=True)

X_resampled_selected_dt_bagging_train , X_resampled_selected_dt_bagging_test , y_resampled_selected_dt_bagging_train ,  y_resampled_selected_dt_bagging_test = train_test_split(X_resampled_selected_dt_bagging,y_resampled,test_size=0.3,random_state=42)
X_resampled_selected_dt_bagging_train= X_resampled_selected_dt_bagging_train.reset_index(drop=True)
y_resampled_selected_dt_bagging_train = y_resampled_selected_dt_bagging_train.reset_index(drop=True)

X_resampled_selected_dt_adaboosting_train , X_resampled_selected_dt_adaboosting_test , y_resampled_selected_dt_adaboosting_train ,  y_resampled_selected_dt_adaboosting_test = train_test_split(X_resampled_selected_dt_adaboosting,y_resampled,test_size=0.3,random_state=42)
X_resampled_selected_dt_adaboosting_train= X_resampled_selected_dt_adaboosting_train.reset_index(drop=True)
y_resampled_selected_dt_adaboosting_train = y_resampled_selected_dt_adaboosting_train.reset_index(drop=True)

# Logistic Regression
#70% train 30% test
X_resampled_selected_lr_stacking_train , X_resampled_selected_lr_stacking_test , y_resampled_selected_lr_stacking_train ,  y_resampled_selected_lr_stacking_test = train_test_split(X_resampled_selected_lr_stacking,y_resampled,test_size=0.3,random_state=42)
X_resampled_selected_lr_stacking_train= X_resampled_selected_lr_stacking_train.reset_index(drop=True)
y_resampled_selected_lr_stacking_train = y_resampled_selected_lr_stacking_train.reset_index(drop=True)

X_resampled_selected_lr_bagging_train , X_resampled_selected_lr_bagging_test , y_resampled_selected_lr_bagging_train ,  y_resampled_selected_lr_bagging_test = train_test_split(X_resampled_selected_lr_bagging,y_resampled,test_size=0.3,random_state=42)
X_resampled_selected_lr_bagging_train= X_resampled_selected_lr_bagging_train.reset_index(drop=True)
y_resampled_selected_lr_bagging_train = y_resampled_selected_lr_bagging_train.reset_index(drop=True)

X_resampled_selected_lr_adaboosting_train , X_resampled_selected_lr_adaboosting_test , y_resampled_selected_lr_adaboosting_train ,  y_resampled_selected_lr_adaboosting_test = train_test_split(X_resampled_selected_lr_adaboosting,y_resampled,test_size=0.3,random_state=42)
X_resampled_selected_lr_adaboosting_train= X_resampled_selected_lr_adaboosting_train.reset_index(drop=True)
y_resampled_selected_lr_adaboosting_train = y_resampled_selected_lr_adaboosting_train.reset_index(drop=True)

#Bagging (Combine)
X_resampled_combine_bagging_train , X_resampled_combine_bagging_test , y_resampled_combine_bagging_train , y_resampled_combine_bagging_test = train_test_split(X_resampled_combine_bagging,y_resampled,test_size=0.3,random_state=42)
X_resampled_combine_bagging_train = X_resampled_combine_bagging_train.reset_index(drop=True)
y_resampled_combine_bagging_train  = y_resampled_combine_bagging_train.reset_index(drop=True)
```

```
#selected from p value
from pathlib import Path
from google.colab import drive
drive.mount('/content/drive')

# Specify the file path
train_csv_path =  '/content/drive/My Drive/train_data.csv'
test_csv_path =  '/content/drive/My Drive/test_data.csv'

train_file = Path(train_csv_path)
test_file = Path(test_csv_path)

test_data = pd.read_csv(test_csv_path)
train_data = pd.read_csv(train_csv_path)

X_resample_selected_train = train_data.drop(columns=['label'])
y_resample_selected_train = train_data['label']

X_resample_selected_test = test_data.drop(columns=['label'])
y_resample_selected_test = test_data['label']
```

Show hidden output

```
print(X_resample_selected_train.shape)
```

Show hidden output

```
from sklearn import svm

datasets = {
    "base": (X_resampled_train, y_resampled_train, X_resampled_test, y_resampled_test),
    "selected": (X_resample_selected_train, y_resample_selected_train, X_resample_selected_test, y_resample_selected_test),
    "dt_similar": (X_resampled_selected_dt_similar_train, y_resampled_selected_dt_similar_train, X_resampled_selected_dt_similar_test, y_resampled_selected_dt_similar_test),
    "lr_similar": (X_resampled_selected_lr_similar_train, y_resampled_selected_lr_similar_train, X_resampled_selected_lr_similar_test, y_resampled_selected_lr_similar_test),
    "dt_combined": (X_combined_features_dt_train, y_combined_features_dt_train, X_combined_features_dt_test, y_combined_features_dt_test),
    "lr_combined": (X_combined_features_lr_train, y_combined_features_lr_train, X_combined_features_lr_test, y_combined_features_lr_test),
    "dt_stacking": (X_resampled_selected_dt_stacking_train, y_resampled_selected_dt_stacking_train, X_resampled_selected_dt_stacking_test, y_resampled_selected_dt_stacking_test),
    "dt_bagging": (X_resampled_selected_dt_bagging_train, y_resampled_selected_dt_bagging_train, X_resampled_selected_dt_bagging_test, y_resampled_selected_dt_bagging_test),
    "dt_adaboosting": (X_resampled_selected_dt_adaboosting_train, y_resampled_selected_dt_adaboosting_train, X_resampled_selected_dt_adaboosting_test, y_resampled_selected_dt_adaboosting_test),
    "lr_stacking": (X_resampled_selected_lr_stacking_train, y_resampled_selected_lr_stacking_train, X_resampled_selected_lr_stacking_test, y_resampled_selected_lr_stacking_test),
    "lr_bagging": (X_resampled_selected_lr_bagging_train, y_resampled_selected_lr_bagging_train, X_resampled_selected_lr_bagging_test, y_resampled_selected_lr_bagging_test),
    "lr_adaboosting": (X_resampled_selected_lr_adaboosting_train, y_resampled_selected_lr_adaboosting_train, X_resampled_selected_lr_adaboosting_test, y_resampled_selected_lr_adaboosting_test),
    "combine_bagging":(X_resampled_combine_bagging_train, y_resampled_combine_bagging_train, X_resampled_combine_bagging_test, y_resampled_combine_bagging_test)
}

# Store trained models and predictions
svm_models = {}
svm_predictions = {}

# Train and predict for each dataset
for name, (X_train, y_train, X_test,y_test) in datasets.items():
    model = svm.SVC(probability=True)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    y_prob = model.predict_proba(X_test)[:, 1]

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, pos_label=1)
    recall = recall_score(y_test, y_pred, pos_label=1)
    f1 = f1_score(y_test, y_pred, pos_label=1)
    auc = metrics.roc_auc_score(y_test, y_prob)

    # Confusion Matrix Calculation
    cm = confusion_matrix(y_test, y_pred)
    FP, TN, FN, TP = cm[0][1], cm[0][0], cm[1][0], cm[1][1]
    type_I_error = FP / (FP + TN) if (FP + TN) != 0 else 0
    type_II_error = FN / (FN + TP) if (FN + TP) != 0 else 0

    svm_models[name] = model
    svm_predictions[name] = {
        "y_pred": y_pred,
        "accuracy": accuracy,
        "precision": precision,
        "recall": recall,
        "f1_score": f1,
        "auc": auc,
        "type_I_error": type_I_error,
        "type_II_error": type_II_error
    }

    print(f"{name} SVM model trained and predicted.")
```

Show hidden output

```
for name, metrics in svm_predictions.items():
    print(f"Model: {name}")
    print(f"  Accuracy: {metrics['accuracy']:.4f}")
    print(f"  Recall: {metrics['recall']:.4f}")
    print(f"  AUC: {metrics['auc']:.4f}")
    print(f" False Negative: {metrics['type_II_error']:.4f}")
    print("------")
```

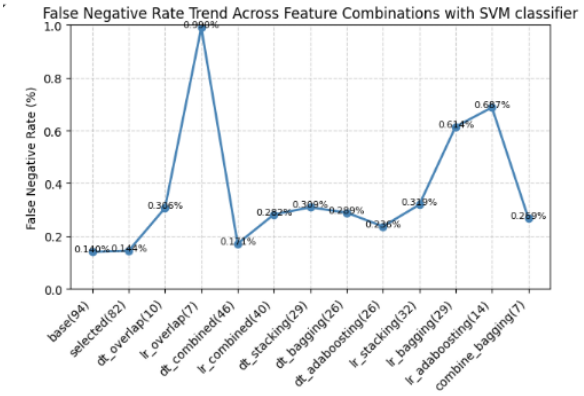
Show hidden output

```
from sklearn.ensemble import RandomForestClassifier

datasets = {
    "base": (X_resampled_train, y_resampled_train, X_resampled_test, y_resampled_test),
    "selected": (X_resample_selected_train, y_resample_selected_train, X_resample_selected_test, y_resample_selected_test),
    "dt_similar": (X_resampled_selected_dt_similar_train, y_resampled_selected_dt_similar_train, X_resampled_selected_dt_similar_test, y_resampled_selected_dt_similar_test),
    "lr_similar": (X_resampled_selected_lr_similar_train, y_resampled_selected_lr_similar_train, X_resampled_selected_lr_similar_test, y_resampled_selected_lr_similar_test),
    "dt_combined": (X_combined_features_dt_train, y_combined_features_dt_train, X_combined_features_dt_test, y_combined_features_dt_test),
    "lr_combined": (X_combined_features_lr_train, y_combined_features_lr_train, X_combined_features_lr_test, y_combined_features_lr_test),
    "dt_stacking": (X_resampled_selected_dt_stacking_train, y_resampled_selected_dt_stacking_train, X_resampled_selected_dt_stacking_test, y_resampled_selected_dt_stacking_test),
    "dt_bagging": (X_resampled_selected_dt_bagging_train, y_resampled_selected_dt_bagging_train, X_resampled_selected_dt_bagging_test, y_resampled_selected_dt_bagging_test),
    "dt_adaboosting": (X_resampled_selected_dt_adaboosting_train, y_resampled_selected_dt_adaboosting_train, X_resampled_selected_dt_adaboosting_test, y_resampled_selected_dt_adaboosting_test),
    "lr_stacking": (X_resampled_selected_lr_stacking_train, y_resampled_selected_lr_stacking_train, X_resampled_selected_lr_stacking_test, y_resampled_selected_lr_stacking_test),
    "lr_bagging": (X_resampled_selected_lr_bagging_train, y_resampled_selected_lr_bagging_train, X_resampled_selected_lr_bagging_test, y_resampled_selected_lr_bagging_test),
    "lr_adaboosting": (X_resampled_selected_lr_adaboosting_train, y_resampled_selected_lr_adaboosting_train, X_resampled_selected_lr_adaboosting_test, y_resampled_selected_lr_adaboosting_test),
    "combine_bagging":(X_resampled_combine_bagging_train, y_resampled_combine_bagging_train, X_resampled_combine_bagging_test, y_resampled_combine_bagging_test)
}

# Store trained models and predictions
RF_models = {}
RF_predictions = {}

# Train and predict for each dataset
for name, (X_train, y_train, X_test,y_test) in datasets.items():
    model = RandomForestClassifier(criterion = "gini",max_depth = 5,min_samples_split = 2,min_samples_leaf=1,random_state = 42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    y_prob = model.predict_proba(X_test)[:, 1]

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, pos_label=1)
    recall = recall_score(y_test, y_pred, pos_label=1)
    f1 = f1_score(y_test, y_pred, pos_label=1)
    auc = roc_auc_score(y_test, y_prob)

    # Confusion Matrix Calculation
    cm = confusion_matrix(y_test, y_pred)
    FP, TN, FN, TP = cm[0][1], cm[0][0], cm[1][0], cm[1][1]
    type_I_error = FP / (FP + TN) if (FP + TN) != 0 else 0
    type_II_error = FN / (FN + TP) if (FN + TP) != 0 else 0

    RF_models[name] = model
    RF_predictions[name] = {
        "y_pred": y_pred,
        "accuracy": accuracy,
        "precision": precision,
        "recall": recall,
        "f1_score": f1,
        "auc": auc,
        "type_I_error": type_I_error,
        "type_II_error": type_II_error
    }

    print(f"{name} RF model trained and predicted.")
```

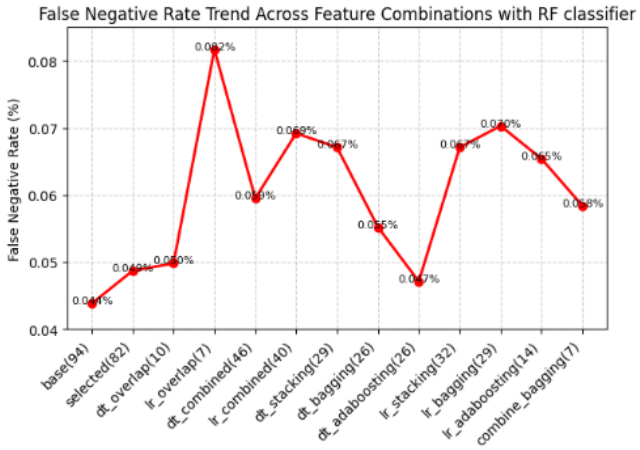
```
for name, metrics in RF_predictions.items():
    print(f"Model: {name}")
    print(f"  Accuracy: {metrics['accuracy']:.4f}")
    print(f"  Recall: {metrics['recall']:.4f}")
    print(f"  AUC: {metrics['auc']:.4f}")
    print(f" False Negative: {metrics['type_II_error']:.4f}")
    print("------")
```

Show hidden output

```
import matplotlib.pyplot as plt
import numpy as np

datasets= ["base(94)","selected(82)","dt_overlap(10)","lr_overlap(7)","dt_combined(46)","lr_combined(40)","dt_stacking(29)","dt_bagging(26)","dt_adaboosting(26)",
          "lr_stacking(32)","lr_bagging(29)","lr_adaboosting(14)","combine_bagging(7)"]
fnr_values = [0.1401,0.1439,0.3061,0.9897,0.1709,0.2818,0.3088,0.2888,0.2363,0.3186,0.6144,0.6874,0.2688]


x = np.arange(len(datasets))

plt.plot(x, fnr_values, marker='o', linestyle='-', color='steelblue', linewidth=2, markersize=6)

# Add data point labels
for i, val in enumerate(fnr_values):
    plt.text(x[i], val, f'{val:.3f}%', ha='center', fontsize=8)

plt.xticks(x, datasets, rotation=45, ha='right')
plt.title('False Negative Rate Trend Across Feature Combinations with SVM classifier')
plt.ylabel('False Negative Rate (%)')
plt.ylim(0, 1.0)
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```




```
fnr_values_rf = [0.0438,0.0487,0.0498,0.0817,0.0595,0.0692,0.0671,0.0552,0.0471,0.0671,0.0703,0.0654,0.0584]

x = np.arange(len(datasets))

plt.plot(x, fnr_values_rf, marker='o', linestyle='-', color='red', linewidth=2, markersize=6)

# Add data point labels
for i, val in enumerate(fnr_values_rf):
    plt.text(x[i], val, f'{val:.3f}%', ha='center', fontsize=8)

plt.xticks(x, datasets, rotation=45, ha='right')
plt.title('False Negative Rate Trend Across Feature Combinations with RF classifier')
plt.ylabel('False Negative Rate (%)')
plt.grid(True, linestyle='--', alpha=0.6)
plt.ylim(0.04, 0.085)
plt.tight_layout()
plt.show()
```



**POSTER**

UNIVERSITI TUNKU ABDUL RAHMAN
FACULTY OF INFORMATION COMMUNICATION AND TECHNOLOGY

# Financial Distress Detection using Ensemble Learning


## 01 INTRODUCTION

- Financial distress is a scenario in which an individual or a company fails to generate the revenues to cover their financial responsibilities.
- Ensemble learning is a combination of multiple individual models
- It has consistence predicting result, better accuracy and time efficiency than a single machine learning method and statistical prediction models

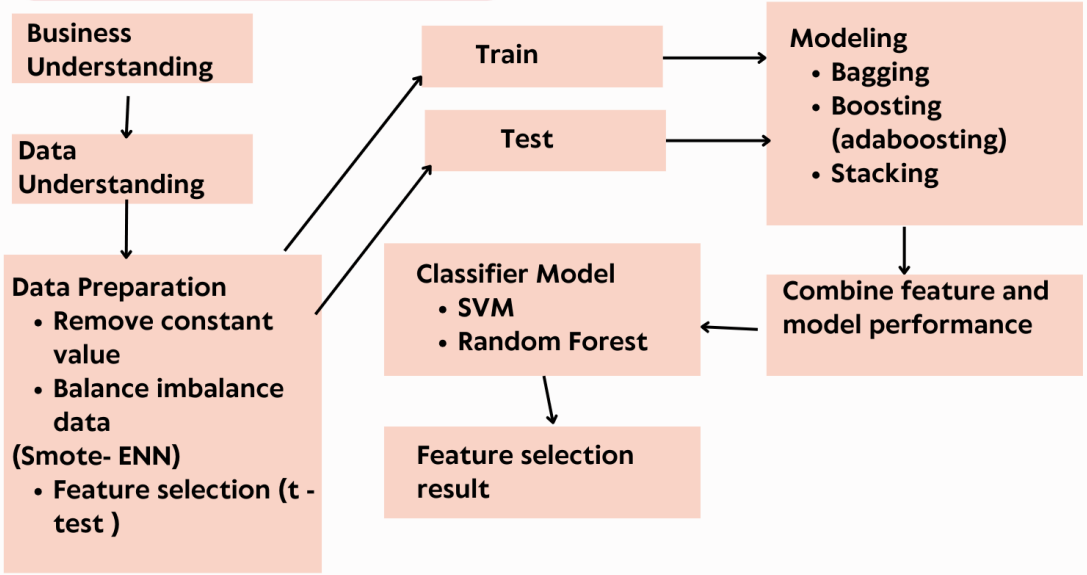
## 02 AIM AND OBJECTIVES

Aim : Identify the optimal ensemble learning technique in detecting financial distress risk.

Objectives :

- Familiarize the architecture ensemble learning techniques
- Compare and contrast three ensemble learning techniques using classifiers like logistic regression and decision tree in classifying financial status
- Relate the findings to interpret business implications of financial distress

## 03 PROPOSED METHOD




Student: Phoebe Wong Hui Lei  Supervisor : Dr Tong Dong Ling