Statistical Classification, an Application to Credit Default

A thesis submitted in fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

MATHEMATICAL STATISTICS

in the

DEPARTMENT OF STATISTICS

of

RHODES UNIVERSITY

by

Anele Sikhakhane

September 2024

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Rhodes University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: 9th September 2024

Abstract

Statistical learning has been used in both industry and academia to create credit scoring models. These models are used to predict who might default on their loan repayments, thus minimizing the risk financial institutions face. In this study six traditional and one more recent classifier, namely kNN, LDA, CART, RF, AdaBoost, XGBoost and SynBoost were used to predict who might default on their loans. The data set used in this study was imbalanced thus sampling and performance evaluation techniques were investigated and used to balance the class distribution and assess the classifiers performance. In addition to the standard variables and data set, new variables called synthetic variables and synthetic data sets were produced, investigated and used to predict who might default on their loans. This study found that the synthetic data set had strong predictive power and sampling methods negatively affected the classifiers performance. The best-performing classifier was XGBoost, with an AUC score of 0.7732.

Keywords: **Binary Classification, Credit Card Default, Credit Risk, Machine Learning, Statistical Learning, Synthetic Features.**

Contents

List of Figures

List of Tables

List of Abbreviations

Adaboost: Adaptive Boosting

AUPRC: Area Under the Precision-Recall Curve

AUC: Area Under the Receiver Operating Characteristic Curve

ANN: Artificial Neural Networks

Bagging: Bootstrap Aggregation

CBR: Case Based Reasoning

CART: Classification and Regression Trees

CatBosst: Categorical Boosting

CT: Classification Tree

CV: Cross Validation

DA: Discriminant Analysis

DNN: Deep Neural Networks

EDA: Exploratory Data Analysis

XGBoost: Extreme Gradient Boosting

GBT: Gradient Boosted Trees

GBDT: Gradient Boosting Decision Trees

GSCV: Grid Search Cross Validation

IR: Imbalance Ratio

IG: Information Gain

KC: Keyword Clustering

KCV: k- Fold Cross Validation

KNN: k- Nearest Neighbours

LightGBM: Light Gradient Boosting

LDA: Linear Discriminant Analysis

Logit: Linear Logistic Regression

LOOCV: Leave One Out Cross Validation

LR: Logistic Regression

MCC: Matthews Correlation Coefficient

MARS: Multivariate Adaptive Regression Splines

NB: Naive Bayes

NT dollar: New Taiwan Dollar

mtry: Number of Features for Each Tree

OOB: Out-of-Bag

OSS: One-Sided Sampling

P2P: Peer-to-Peer

PLR: Penalized Logistic Regression

P: Precision

PCA: Principal Components Analysis

QII: Quantitative Input Influence

RF: Random Forest

RO: Random Oversampling

RU: Random Undersampling

RCSM: Reassign Credit Scoring Model

R: Recall

ROC: Receiver Operating Characteristics

SVM: Support Vector Machines

SynBoost: Synthetic Boosting

SMOTE: Synthetic Minority Oversampling

Acknowledgments

I would love to express my extreme gratitude and thanks to my superb supervisor, Mr Jeremy Baxter. His meticulous attention to detail, adept at patiently identifying all my errors, inconsistencies, poor spelling, grammatical mistakes, redundancies and redundancies, are greatly appreciated. Furthermore, his input, suggestions, understanding and care have been immaculate. The completion of this thesis would have been impossible without Jeremy's wisdom, excellent guidance and expertise for which I am extremely grateful, beyond what I can accurately express in a select few words. Jeremy, your mentorship, belief and trust in me have been instrumental in my personal development, I extend my heartfelt thanks to you.

Thank you Jeremy :).

Chapter 1

Introduction

Financial institutions such as banks are exposed to three types of risk, namely credit, market and operational risk (Moscato et al., 2021). It is estimated that approximately 60% of a banks threat is credit risk (Moscato et al., 2021). Credit risk is the probability of financial loss when a borrower fails to repay their loan (Bhatore et al., 2020). There is a need for credit as it allows users to obtain goods or service immediately and pay back the amount loaned to the bank later. Financial institutions need to generate an income and one of the ways this is achieved is by making a profit on the interest charged on the credit given to clients (DeYoung and Rice, 2004).

A problem arises when people are unable to pay back the outstanding amount, this is called a default. A default occurs when a debtor fails to repay a loan (Siaw et al., 2014). Financial loss is experienced when a client defaults. Being able to accurately predict defaults has a large impact on earnings (Anand et al., 2022). Banks have large data sets. It is not feasible that a human can accurately profile all credit accounts thus statistical models and machine learning techniques have been used for credit risk evaluation (Leo et al., 2019).

1.1 Background: Credit Risk

Yeh and Lien (2009) conducted a comparison of the predictive accuracy of six data mining techniques, namely k- Nearest Neighbour (kNN), Naive Bayes (NB), Artificial Neural Networks (ANN's), Discriminant Analysis (DA), Logistic Regression (LR) and Classification Trees (CTs). The data set is a real data set from a Taiwanese bank and contains 30 000 entries of 25 attributes or features or independent variables. Some of the attributes have more than one entry, for example history of payments has five entries detailing the last five monthly payments. The data set is unbalanced, (see section 3.1), with the ratio of defaults to non-defaults being 1:3.52. The details of the loanees have been anonymized to protect the privacy of the loanees. These classifiers were applied to predict the probability of default

on credit card payments.Yeh and Lien (2009) noted that predicting the probability of default was more meaningful than simply classifying the observations into two groups. A novel method called the Sorting Smoothing Method was used to estimate the probability of default. The models were trained on the unbalanced data and the coefficient of determination R^2, was used to assess their predictive accuracy. They concluded that the only model that could accurately predict the probability of default was an ANN, with a coefficient of determination 0.9647.

Chuang and Lin (2009) proposed a reassign credit scoring model (RCSM) which is a two-stage credit scoring model. In the first stage an artificial neural network (ANN) is used to split the data into two groups, namely good credit rating and poor credit rating. In the second stage case based reasoning (CBR) re-distributes good credit ratings that have been found to be poor credit ratings into the poor credit rating group. This two-stage model was compared to four models, namely classification and regression trees (CART), linear discriminant analysis (LDA), ANN's and multivariate adaptive regression splines (MARS). The data set was an unnamed credit approval data set with 1 000 instances and 20 features. The data set was unbalanced with the ratio of bad to good credit ratings being 1:2.33. The models were trained on the unbalanced data and accuracy rate, type one and two errors were used to compare the performance of the models. The two-stage model outperformed the other one-stage models with an accuracy rate 86%. Chuang and Lin (2009) stated that the proposed method displayed favourable attributes as a computer aided credit scoring system.

Kruppa et al. (2013) used machine learning techniques to estimate the probability of default on short term loans. The loans had a maximum life span of 13 months. The data set was taken from an unnamed European manufacturing company. The company manufactures house-hold appliances and lets customers pay off the amount in installments. The data set contained 64 524 instances and 17 features. The data set was imbalanced with the ratio of defaults to non defaults being 1:6. Five models, namely kNN, random forest (RF) LR and bagged nearest neighbours were applied. The models were trained on the imbalanced data. Area under the receiver operating characteristic curve (AUC) and brier score were used to assess the models performance. RF outperformed the other models with an AUC of 0.959 and a brier score of 0.071.

Li et al. (2016) proposed using a hybrid model to predict defaults in small and medium enterprise accounts. The hybrid model consists of three stages. In the first stage LR is trained on the principal components analysis (PCA) output to estimate the probability of default. In the second stage, an ANN is trained on inputs from the data to estimate the probability of default. A new data set is created by adding both these estimates as new features to the original data set, resulting in a total of 24 features. In the final step a ANN is trained on the new data set to predict whether an account will default. The data set was taken from Suomen Asiakastieto Ltd, a Finnish credit rating company. The data set contained 2 081 instances and was balanced with 52.57% of the instances being defaults. The proposed method was benchmarked against LR and ANN's. Accuracy rate was used to assess the models performance.

The proposed method outperformed the other models with an accuracy of 84.59%.

Islam et al. (2018) developed a two-stage model to predict defaults in the Taiwan credit card data set used by Yeh and Lien (2009). In the first step the probability of default is computed using a test called the standard test. The standard test uses extremely randomized trees to identify abnormal transactions. These transactions are then passed to the next test. The second test, called the customer-specific test is used to try identify a specific reason as to why the transaction failed the first test. Using these two tests a total risk probability is computed. If the probability is above a predefined threshold the account is defined as a default (Islam et al., 2017). The proposed method was benchmarked against other popular models, namely RF, naive Bayes, gradient boosted trees (GBT) and kNN. The models were trained on the imbalanced data, the accuracy rate, precision, recall and F1-score were used to assess the performance of the models. Although the data was imbalanced, the accuracy rate which is not a fair metric for imbalanced data (see Section 3.1) was used to assess the performance of the models. The proposed method outperformed the other models on all metrics with an accuracy of 95.84%.

Chang et al. (2018) proposed using extreme gradient boosting (XGBoost) to predict defaults in loans. XGBoost was proposed because the method gained popularity in major big-data competitions as it is highly accurate and fast. Chang et al. (2018) criticized two-stage models such as the model used by Chuang and Lin (2009), for being too complicated to build and apply. The data set on corporate defaults from an unnamed Taiwanese financial institution was used in this study. This data set is different to the one used by Yeh and Lien (2009). The data set had 32 features and was imbalanced with the ratio of defaults to non-defaults being 1:9.4. The class distributions were balanced using cluster-based undersampling. Three data sets each with a different sampling ratio were produced the ratio of defaults to non-defaults were, 1:1, 1:2 and 1:4. The models, namely XGBoost was compared to LR, support vector machines (SVM), and ANN's were then trained on the data sets, accuracy rate and AUC was used to assess the performance of the models. XGBOOST outperformed the other models with an AUC of 0.944 and accuracy rate of 90.00%. Chang et al. (2018) noted that future research can improve on the sampling technique used as random sampling produces a data set that does not reflect the true condition of the overall data.

Hamori et al. (2018) compared the effectiveness of ensemble methods and deep learning in the detection of defaults. Three ensemble methods were considered, namely bootstrap aggregation (bagging), random forest and boosting. Four neural networks with different activation functions and four deep neural networks (DNN) each with different activation functions, were also applied on the data set. In total, eleven models were used to predict defaults in loan payments. The researchers used the same data set that Yeh and Lien (2009) used. The researchers randomly undersampled the majority class to balance the data. Accuracy rate and AUC were used to assess the performance of the models. The ratio of the training to test data sets were 90% to 10% and 75% to 25%. Hamori et al. (2018) concluded that boosting was superior to the other models used with an average accuracy of 70.94% and a AUC

of 0.769.

Zhou et al. (2019) used a heterogeneous ensemble to predict defaults. The ensemble consisted of three tree-based models, namely gradient boosting decision trees (GBDT), XGBoost and light gradient boosting (LightGBM). The base learners were tree-based as they have shown to work well with both high-dimensional and imbalanced data. The data set was a peer-to-peer (P2P) lending data set from an unnamed Chinese lending company. P2P lending is where users can give loans to other users and earn money from the interest of their loans (Zhou et al., 2019). The data set contained 1 138 features and 15 000 instances. A learning model-based feature ranking method was used to reduce the dimensionality of the data set. The data set had a default to non-default ratio of approximately 1:8. The heterogeneous ensemble was benchmarked against nine other models, namely GBDT, XGBoost, LightGBM, ANN, LR, RF, SVM, kNN and adaptive boosting (AdaBoost). The models were trained and tested on the imbalanced data. AUC, sensitivity specificity and F1-score were used to assess the models accuracy. The heterogeneous ensemble outperformed the other models with an AUC of 0.7185 and a F1-score of 0.8615. Zhou et al. (2019) noted that categorical boosting (CatBoost) should be considered for future research as the model uses a special technique to deal with categorical variables.

Bracke et al. (2019) noted that predictions made by various models are hard to interpret due to their complexity. This study aimed to make models more explainable and interpretable. Models that are easier to interpret can be used to explain the decision-making process to other stakeholders in the business. The quantitative input influence (QII) method makes models more explainable. The QII approach was applied on a mortgage loan data set from the UK Financial Conduct Authority. The data set contained 6 million instances of 20 features. Two additional features were created, namely loan to value ratio at origination and current loan to value ratio. The data set was highly imbalanced with only 2.5% of the loans being defaults. Two classifiers were trained on the imbalanced data set, namely linear logistic regression (Logit) and GBT. AUC and Area under the precision-recall curve (AUPRC) were used to assess the models performance. GBT outperformed the Logit classifier with an AUC of 0.81 and a AUPRC of 0.19. Bracke et al. (2019) concluded that making models more explainable is an important addition as this helps with understanding the properties of a data set.

Moscatelli et al. (2020) noted that the 2008 global financial crisis highlighted the shortcomings of default forecasting systems based on credit ratings. The systems were slow to adapt to economic changes and the systems could not model the non-linear interactions between economic, financial and credit variables. They also noted that past research suggests that using models to predict defaults is a suitable alternative to modelling default risk, as models tend to ignore the significance of economic determinants for default risk. However, models can accurately predict defaults from out-of-sample forecasts. The predictive power of statistical and machine learning models were compared on a corporate default data set. The five models considered were LDA, LR, RF, GBT and penalized logistic regression (PLR). The data set was from the Italian Credit Register and contained approximately 250 000 instances of

24 variables about Italian firms from $2011 - 2017$. The data set was imbalanced with approximately 2.5% of the observations being defaults. The data was balanced using random undersampling. The performance of the classifiers was assessed using the accuracy rate. They concluded that the machine learning models outperformed the statistical models. GBT outperformed all other models with an average accuracy of 82.57%. The researchers suggested that the joint use of both types of models may be beneficial in cases where the estimates derived by the models are different.

Xia et al. (2020) proposed a novel credit scoring model that uses narrative data to forecast defaults. Narrative data is any data that the user has given that is subjective and hard to verify, for example the reason a user might give when taking a loan. A keyword clustering (KC) algorithm was used to extract the narrative data from the data sets. Three separate P2P data sets from a former American P2P company, Lending Club, was used to test the approach. The data set was imbalanced; the ratio of defaults to non-defaults was not specified. The Catboost model was trained using the narrative data. The proposed model was benchmarked against six popular models, namely LR, CART, bagged neural network, RF, GBDT and XGBoost. Accuracy rate, type one and two errors were used to assess the performance of the models. The proposed method outperformed the other models on all evaluation metrics.

Liu et al. (2022) compared the predictive power of machine learning and mathematical models. The models were used to estimate the probability of default on a lending data set from a Chinese P2P service named Renrendai. The data set contained 8 650 instances of 29 variables. The data set was unbalanced with 16.28% of the instances being defaults. A mathematical model, three machine learning models, namely kNN, SVM and RF and logistic model were used to predict defaults. The models were trained and assessed using the imbalanced data. AUC and accuracy rate were used to assess the performance of the models. They concluded that machine learning models can be reliably used to forecast defaults with RF outperforming the other models with average AUC of 0.9662 and an average accuracy of 0.9565. Liu et al. (2022) constructed a profit function to further compare the performance of the models, as the cost of miss-classification is not equal. From the profit function, RF were identified as most suitable machine learning model.

Table 1.1 summarizes the techniques used in past research in credit risk modelling. A challenge all researchers face in the credit risk space is the scarcity of real data sets. Often the same data sets are used or the data set used is not made available for academic use. Credit risk data sets are imbalanced due to the nature of the phenomenon, as most people do not default on their loans. Researchers often use random undersampling to balance the data set and then build and assess the models. The explainability and interpretability of models are also important and not just the final accuracy of the model. Many researchers estimate the probability of default instead of simply classifying the observations. Yeh and Lien (2009) noted that estimating probability of default is more useful than simply classifying the observations. The probability of default can be used to estimate the borrowers risk level and the loan

can be accurately priced (Gurnỳ et al., 2013).

1.1.1 Research Aims

Previous research using the Taiwanese data set (Yeh and Lien, 2009) have used the default economic variables to train predictive models. These studies have all demonstrated that supervised statistical learning models do not perform well when trained on the default data. This study aims to train traditional supervised models on the Taiwan data with the addition of new variables. These new transformed variables are produced by linear combinations of the existing economic variables in the data set. The idea is that these new variables may have additional predictive power hence improving the performance of the base models.

1.1.2 Importance of the Study

This study aims to build upon the groundwork laid by Yeh and Lien (2009) by investigating additional factors that may enhance classification accuracy. This study investigated a total of seven classification techniques to find the optimal classifier (see Chapter 2). Additionally, this study investigates how imbalanced data affects classification accuracy and how to assess classifiers under this constraint (see Sections 2.6 and 3.1). This study also addresses the challenge of reducing a data set's dimensionality while maintaining classification accuracy (see Section 3.2). Furthermore, this study aims to investigate the effect synthetic features have on classification accuracy. This led to the development of the ensemble classification technique synthetic boosting (SynBoost) (see Section 2.5.3).

1.1.3 Chapter Outline

This chapter has introduced the application of statistical learning techniques to credit risk and the aims of this study. Chapter 2 introduces and discusses the classification techniques used in this study and how to assess classifiers when the data has an imbalanced class distribution. Chapter 2 also details how synthetic features are produced and how the ensemble classification technique, SynBoost works. Chapter 3 introduces class imbalance and sampling procedures used to balance the class distribution Chapter 3 also discusses the dimensionality of the data set and how to reduce the size using principal components. Chapter 4 introduces the data analysis of the Taiwan data set and discuses the results obtained in this study. Chapter 5 summarizes the results obtained in the study and proposes further research questions that can be investigated.

Table 1.1: Summary of Techniques Used in Past Research.

Paper	data set	Imbalance Ratio	Sampling Technique	Accuracy Measure	Classifiers
Yeh and Lien (2009)	Bank loan	28.40%	NA	R^2	kNN, NB, ANN's, DA, LR and CT
Chuang and Lin (2009)	Credit approval	42.91%	NA	Accuracy rate, type one and two errors	RCMS, CART, LDA, ANN's, and MARS
Kruppa et al. (2013)	Bank loan	16.67%	NA	AUC and brier score	kNN, RF LR and bagged nearest neighbours
Li et al. (2016)	Bank loan	52.57%	NA	Accuracy rate	Hybrid model, LR and ANN's
Islam et al. (2018)	Bank loan	28.40%	NA	Accuracy rate precision, recall and F1-score	Hybrid model, RF, naive Bayes, GBT and kNN
Chang et al. (2018)	Bank loan	10.63%	cluster-based undersampling	AUC	XGBoost, LR, SVM and ANN's
Hamori et al. (2018)	Credit loan	28.40%	random undersampling	Accuracy rate, F1 Score and AUC	Bagging, RF, boosting and ANN's
Zhou et al. (2019)	P2P loan	12.50%	NA	AUC, sensitivity specificity and F1-score	GBDT, XGBoost, LightGBM and AdaBoost
Bracke et al. (2019)	Mortgage loan	2.5%	NA	AUC and AUPRC	LR and GBT
Moscatelli et al. (2020)	Bank loan	2.5%	random undersampling	Accuracy rate	LDA, LR, RF, GBT and PLR
Xia et al. (2020)	P2P loan	NA	NA	Accuracy rate, type one and two errors	Novel model, LR, CART, bagged neural network RF, GBDT and XGBoost
Liu et al. (2022)	P2P loan	16.28%	NA	AUC and accuracy rate	kNN, SVM, RF and LR

Chapter 2

Binary Classification

2.1 k - Nearest Neighbours

k - Nearest Neighbours (kNN) is a non-parametric classification model first introduced by Fix and Hodges (1951). kNN is a non-parametric model making it suitable when little is known about the distribution of the data (Peterson, 2009). kNN is a simple classification model that classifies a test observation based on the classification of its k nearest neighbours. The classification is done by a majority voting system, where the most common class that the k points belong to is what the test observation is classified as (Mucherino et al., 2009). The kNN classifier is described in Algorithm 2.1 below.

Algorithm 2.1

Select or specify the number of neighbours, k;
Repeat the following steps for each observation in the test data;
Calculate the distance (Euclidean distance or other) between the test observation and all points in the training data set;
Determine the k nearest neighbours, by selecting the k observations with the smallest distances;
From these k neighbours, count the number of the training data observations in each category or class;
Assign the test data point, that is the new data point, to the most common category or class among the k neighbours;

The standard method of measuring the distance is Euclidean's distance, which measures the straight line distance between two points. Consider two points $\mathbf{x}_s$ and $\mathbf{x}_t \in \mathscr{R}^p$, the Euclidean distance between the two points is given by,

$$distance\,(\mathbf{x}_s, \mathbf{x}_t) = \sqrt{(\mathbf{x}_s - \mathbf{x}_t)^{'}\,(\mathbf{x}_s - \mathbf{x}_t)} = \sqrt{\sum_{j=1}^{p} (x_{sj} - x_{tj})^2}.$$

In the kNN context the test observation is denoted by, $\mathbf{x}_t$ and a point from the training sample is denoted by $\mathbf{x}_s$.

Consider the simulated data in Figure 2.1 which contains two classes. Class one is denoted by the black diamonds and class two is denoted by the red diamonds. The aim is to classify the test observation, $\mathbf{x}_t$, which is denoted by the blue point.

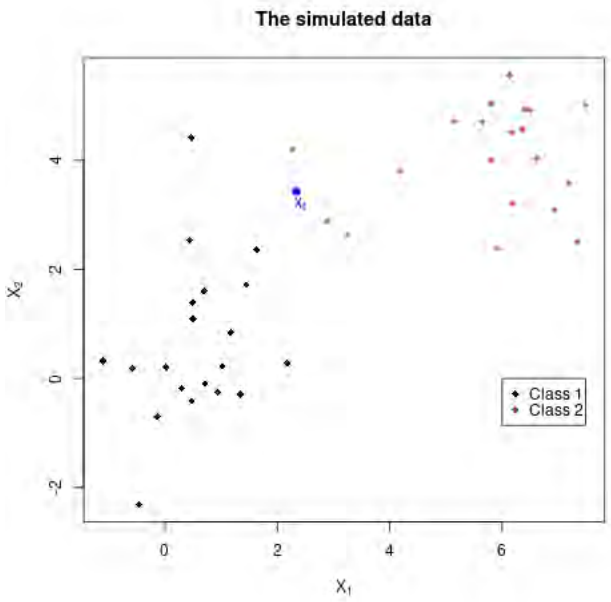


Figure 2.1: Simulated Data.

Figure 2.2 depicts the case where k equals one. The test observation, $\mathbf{x}_t$, has as its nearest neighbour is a red diamond which is observation 39 in the data set and hence labelled 39. The closest neighbour, point 39 belongs to class 2 thus $\mathbf{x}_t$ is classified as class 2.

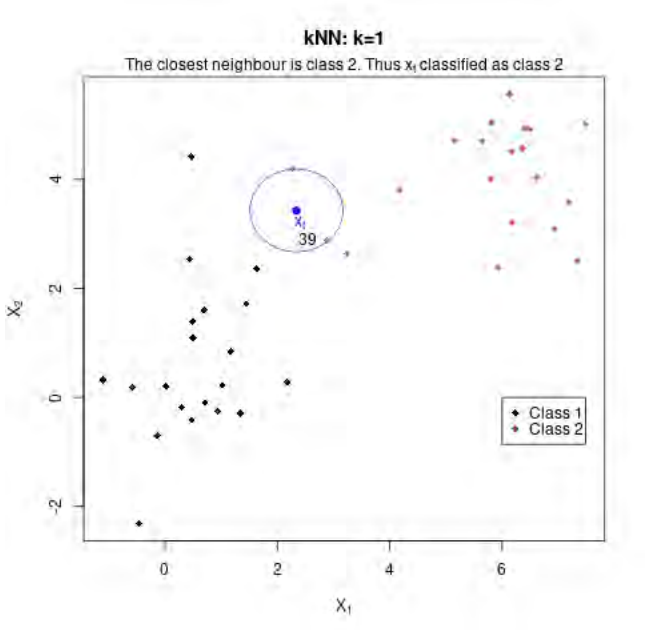


Figure 2.2: The k Nearest Neighbours of $\mathbf{x}_t$ when $k = 1$.

Figure 2.3 depicts the case where k equals three. The test observation, $\mathbf{x}_t$, has three nearest neighbours labelled 21, 39 and 37. All three points are red diamonds, that is they belong to class two. Thus $\mathbf{x}_t$ is classified as belonging to class two.

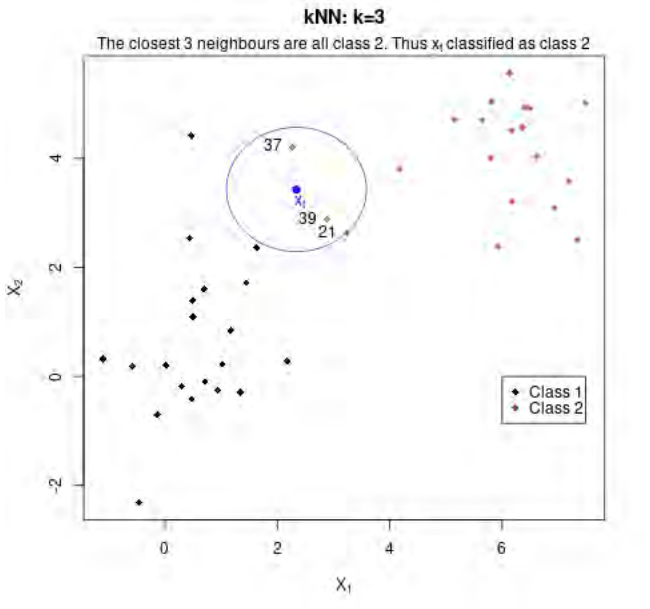


Figure 2.3: The k Nearest Neighbours of $\mathbf{x}_t$ when $k = 3$.

Figure 2.4 depicts the case where k equals 5. The test observation, $\mathbf{x}_t$, has five nearest neighbours labelled 15, 21, 36, 37 and 39. Point 15 is a black diamond which belongs to class one; points 21, 36,

37 and 39 are red diamonds which belong to class 2. Since four of the five neighbours belong to class 2, using majority voting $\mathbf{x}_t$ is classified as belonging to class 2.

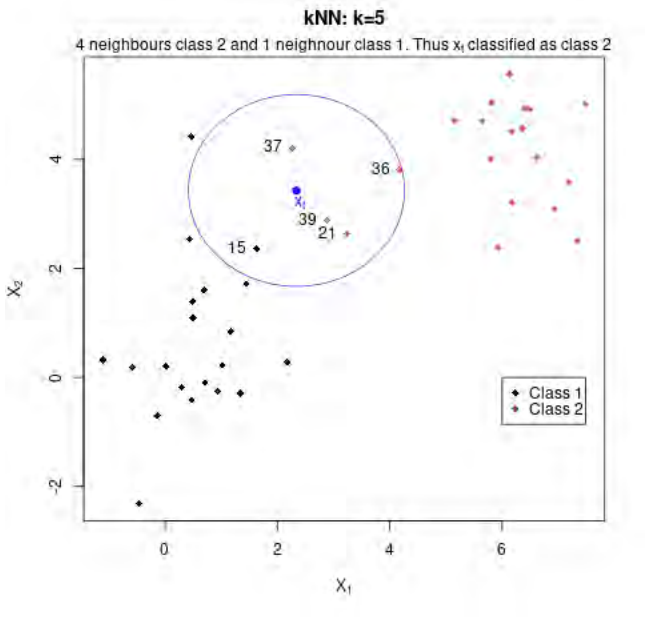


Figure 2.4: The k Nearest Neighbours of $\mathbf{x}_t$ when $k = 5$.

Figure 2.5 depicts the case where k equals seven. The test observation, $\mathbf{x}_t$, has seven nearest neighbours labeled 3, 6, 15, 21, 36, 37, 39. Points 3, 6 and 15 are black diamonds that belong to class one; points 21, 36, 37 and 39 are red diamonds that belong to class 2. Four of the seven points belong to class 2 thus using majority voting the test observation $\mathbf{x}_t$ is classified as belonging to class 2.

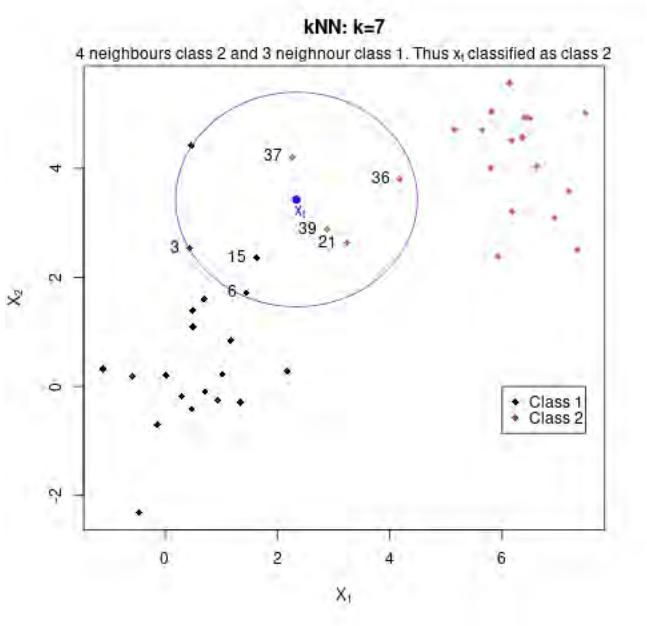


Figure 2.5: The k Nearest Neighbours of $\mathbf{x}_t$ when $k = 7$.

2.1.1 kNN's Limitations and Lazy Learning

kNN is inefficient for high dimensional and large data sets (Kataria and Singh, 2013). This is due to what is termed, the classifier's lazy learning approach. kNN does not generate a model from the training data, instead for each test observation kNN generates a specific model. This lazy learning approach results in high computational costs and great storage requirements (Gutiérrez et al., 2016).

2.1.2 Selecting an Appropriate Value for k

Selecting an appropriate value for k is essential to kNN's performance. When k is too large the largest class will have great influence on the classification, leading to a bias towards the largest class. If k is too small noise in the data will have a great influence on the classification of the test observation (Zavrel, 1997).

The value of k is often selected by running simulations on the training and validation data sets. The value of k which results in the maximum value of classification accuracy is selected. This value is often an odd number to ensure the voting is deterministic. (Maleki et al., 2017)

2.1.3 Application of kNN to the Iris Data Set

Consider the Iris data set (Fisher, 1936). The aim is to classify a plant species based on four features, namely the length and width of sepals and petals. The plant species are *setosa*, *versicolor* and *virginica.*

The data set contains 50 plants for each of the three species.

The Iris data set was split into a 70% training and 30% test set. Each classifier was trained on the training data and subsequently applied on the test data.The kNN classifier was implemented on the test data set, where k is equal to three and five. The R code containing the implementation is can be found in Appendix B 5.2.

Tables 2.1 and 2.2 present the confusion matrices, (see Section 2.6 on page 59) for $k = 3$ and $k = 5$ kNN classifiers.

Table 2.1: Confusion Matrix for $k = 3$.

		Predicted Class		
Actual Class		Setosa	Versicolor	Virginical
	Setosa	17	0	0
	Versicolor	0	10	2
	Virginical	0	1	14

Table 2.2: Confusion Matrix for $k = 5$.

		Predicted Class		
Actual Class		Setosa	Versicolor	Virginical
	Setosa	17	0	0
	Versicolor	0	11	1
	Virginical	0	0	15

From Tables 2.1 and 2.2 the F1 score (see Section 2.6 on page 61) for each value of k, which can be found in Table 2.3

Table 2.3: F1 Score of kNN Classifier.

Value of k	F1-Score
k = 3	0.931
k = 5	0.9887

Figure 2.6 depicts a plot of F1 scores for different values of k.

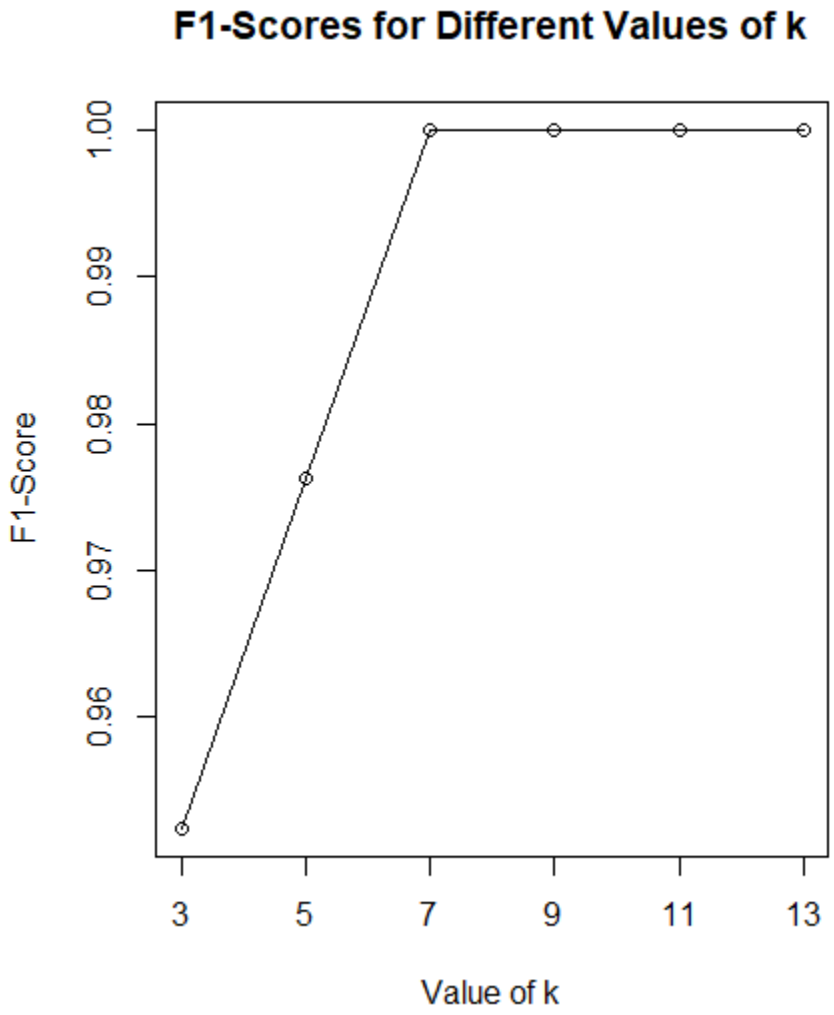


Figure 2.6: kNN F1-Scores for the Iris Data.

From the results in Tables 2.1 and 2.2 both kNN classifiers performed well on the Iris data. However from Table 2.3 the highest F1 score is achieved when the value of k is five. From the two classifiers the best value of k is five. Figure 2.6 depicts the F1 score for different values of k. The F1 scores increase with the values of k reaching a perfect score of one, for values in the range of $7-13$. Thus the optimal value of k is achieved when k is in the range of $7-13$.

2.2 Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) was first introduced by Fisher (1936) for binary classification and later generalized by Rao (1948) to work with multi-class classification. LDA can be used as both a dimensionality reduction technique and as a supervised binary classifier (Wang et al., 2016).

2.2.1 Linear Discriminant Analysis for Classification

Consider a data set with n observations on a p dimensional feature space denoted by $\mathbf{X}_{n\times p}$, with a categorical response, $\mathbf{Y}_{n\times 1}$. The aim is to use Linear Discriminant Analysis (LDA) to classify an observation into one of K classes. The LDA classifier works by deriving a variate which is a linear combination of the independent features, to discriminate between different classes. The variate is called the linear discriminant function which is denoted by $\delta_k(\mathbf{x})$ (Hair et al., 1984). An observation is classified into the class that maximizes $\delta_k(\mathbf{x})$ namely, $Y(\mathbf{x}) = argmax_k\ \delta_k(\mathbf{x})$ (Hastie et al., 2009). In the three dimensional space and for a binary classifier ($K = 2$), this can be visualized as the plane that separates the two classes most optimally (Hastie et al., 2009). The three dimensional interpretation is synonymous with what happens in higher dimensions but those are harder to conceptualize and visualize (Park and Park, 2008).

There are several LDA methods. The method discussed below relies on Bayes theorem. Bayes theorem was introduced by Bayes (1763) and is used to calculate the probability of A conditional on given B, which is expressed as $P(A|B)$,

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}.$$

Let the prior probability that a randomly selected observation belongs to the kth class, $P(Y = k)$, be denoted by π_k, where $\sum_{k=1}^{K} \pi_k = 1$. Let the class-conditional density function of an observation $\mathbf{x}$ belonging to the kth class be denoted by $f_k(\mathbf{x}) = P(\mathbf{X} = \mathbf{x}|Y = k)$. The value of $f_k(\mathbf{x})$ is relatively large if the probability that the observation $\mathbf{x}$ belongs to the kth class is high. If value of $f_k(\mathbf{x})$ is relatively small, the probability that the observation $\mathbf{x}$ belongs to the kth class is small (James et al., 2013). Applying Bayes theorem yields,

$$P(Y = k|\mathbf{X} = \mathbf{x}) = \frac{P(\mathbf{X} = \mathbf{x}|Y = k)P(Y = k)}{P(\mathbf{X} = \mathbf{x})} = \frac{\pi_k f_k(\mathbf{x})}{\sum_{i=1}^{K} \pi_i f_i(\mathbf{x})}. \tag{2.1}$$

In equation 2.1 $P(\mathbf{X} = \mathbf{x}) = \sum_{i=1}^{K} \pi_i f_i(\mathbf{x})$ is a result of the law of total probability. The prior $P(Y = k) = \pi_k$ can be estimated from the training data by calculating the proportion of observations in $Y_{n\times 1}$ that belong to class k. thus, $\hat{\pi}_k = \frac{N_k}{N}$, where N_k denotes the number of observations in the kth class and N denotes the total number of observations in the training set.

Estimating the density function $f_k(\mathbf{x})$ is difficult as the distribution that the data follows is often unknown. To estimate $f_k(\mathbf{x})$, LDA makes the assumption that for each class k, $\mathbf{X}_p \sim N_p(\mu_k, \Sigma)$ (James et al., 2013). The classes have class specific mean, μ_k which called the centroid. The classes share a common covariance matrix Σ (Hastie et al., 2009).

Consider the binary case $(K = 2)$. The two classes, k and i, are compared by substituting the appropriate multivariate normal density density function into equation 2.1 and taking the ratio between the two classes, namely

$$\begin{aligned}\frac{P(Y=k|X=\mathbf{x})}{P(Y=i|X=\mathbf{x})} =& \frac{\pi_k f_k(\mathbf{x})}{\pi_i f_i(\mathbf{x})+\pi_k f_k(\mathbf{x})} \times \frac{\pi_i f_i(\mathbf{x})+\pi_k f_k(\mathbf{x})}{\pi_i f_i(\mathbf{x})} \\ =& \frac{\pi_k f_k(\mathbf{x})}{\pi_i f_i(\mathbf{x})} \\ =& \frac{\pi_k exp\left(-\frac{1}{2}(\mathbf{x}-\mu_k)'\Sigma^{-1}(\mathbf{x}-\mu_k)\right)}{\pi_i exp\left(-\frac{1}{2}(\mathbf{x}-\mu_i)'\Sigma^{-1}(\mathbf{x}-\mu_i)\right)} \\ =& \frac{\pi_k}{\pi_i} exp\left(-\frac{1}{2}(\mathbf{x}-\mu_k)'\Sigma^{-1}(\mathbf{x}-\mu_k)+\frac{1}{2}(\mathbf{x}-\mu_i)'\Sigma^{-1}(\mathbf{x}-\mu_i)\right). \end{aligned} \tag{2.2}$$

Taking the natural log of equation 2.2 and expanding the terms in the parentheses, results in the linear discriminant function 2.3 (James et al., 2013),

$$ln\left(\frac{\pi_k}{\pi_i}\right) - \frac{1}{2}(\mu_k+\mu_i)'\Sigma^{-1}(\mu_k-\mu_i) + \mathbf{x}'\Sigma^{-1}(\mu_k-\mu_i)$$

$$= \mathbf{x}'\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k'\Sigma^{-1}\mu_k + ln(\pi_k) - \left(\mathbf{x}'\Sigma^{-1}\mu_i - \frac{1}{2}\mu_i'\Sigma^{-1}\mu_i + ln(\pi_i)\right) \tag{2.3}$$

The linear discriminant function equations 2.3 and 2.4, are equivalent descriptions of the LDA classification rule. Each observation is assigned to the class for which equation 2.4 is maximised (Hastie et al., 2009):

$$\delta_k(\mathbf{x}) = \mathbf{x}'\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k'\Sigma^{-1}\mu_k + ln(\pi_k). \tag{2.4}$$

Both μ and Σ can be estimated from the training data and are given by

$$\hat{\mu_k} = \frac{\sum_{i:y_i=k} x_i}{N_k}, \tag{2.5}$$

$$\hat{\Sigma} = \frac{\sum_{k=1}^{K}\sum_{i:y_i=k}(x_i-\hat{\mu_k})(x_i-\hat{\mu_k})'}{(N-K)}. \tag{2.6}$$

With binary classification, LDA classifies an observation to class k if,

$$\mathbf{x}'\hat{\Sigma}^{-1}(\hat{\mu_k}-\hat{\mu_i}) > \frac{1}{2}\hat{\mu_k}'\hat{\Sigma}^{-1}\hat{\mu_k} - \frac{1}{2}\hat{\mu_i}'\hat{\Sigma}^{-1}\hat{\mu_i} + ln\left(\frac{\hat{\pi_i}}{\hat{\pi_k}}\right) \tag{2.7}$$

otherwise observations are classified to class i (Hastie et al., 2009).

The normality assumptions under LDA are often violated. However LDA is a robust classifier that works well even when the assumptions are not met. (Hair et al., 1984)

2.2.2 Application of Linear Discriminant Analysis to the Iris Data Set.

Consider a modified version of the Iris data set Fisher (1936), the original data set contains three classes namely setosa, virginica and versicolor and was collected by Edgar (1935). The modified version only contains two classes so that binary classification can take place. The aim is to classify an observation $\mathbf{x}_j$, into one of two classes either as setosa or versicolor. The Iris data set has four features namely, sepal length and width and petal length and width. Let class k denote the setosa class and i denote the versicolor class.

The discriminant function is given by equation 2.4 or equivalently to equation 2.3 . The centroids μ_k, μ_i and covariance Σ, are estimated by equations 2.5 and 2.6 respectfully. The centroids were calculated for each class and are displayed in Table 2.4.

Table 2.4: The Centroids of the Iris Data Set.

Variate	Setosa	Versicolor
Sepal length	$\hat{\mu}_{k1} = 5.006$	$\hat{\mu}_{i1} = 5.9176$
Sepal width	$\hat{\mu}_{k2} = 3.428$	$\hat{\mu}_{i2} = 2.7804$
Petal length	$\hat{\mu}_{k3} = 1.462$	$\hat{\mu}_{i3} = 4.304$
Petal width	$\hat{\mu}_{k4} = 0.246$	$\hat{\mu}_{i4} = 1.304$

The inverse of the covariance matrix $\hat{\Sigma}^{-1}$, was determined by using equation 2.6 and taking the inverse of the resulting matrix. $\hat{\Sigma}^{-1}$ is given by,

$$\hat{\Sigma}^{-1} = \begin{bmatrix} 11.665092 & -7.138548 & -7.453607 & 4.811735 \\ -7.138548 & 11.319771 & 7.006044 & -5.723810 \\ -7.453607 & 7.006044 & 17.250373 & -33.259548 \\ 4.811735 & -5.723810 & -33.259548 & 79.591595 \end{bmatrix}.$$

The Iris data set is a balanced data set since the number of observations in classes i and k are equal, thus the prior probabilities $\hat{\pi}_k$ and $\hat{\pi}_i$ are equal. Consider the observation $\mathbf{x}_1 = \begin{bmatrix} 5.1 & 3.5 & 1.4 & 0.2 \end{bmatrix}'$ and its corresponding response variable, $y_1 = setosa = k$. Substituting the terms $\mathbf{x}_1$, $\hat{\mu}_k$, $\hat{\mu}_i$ and $\hat{\Sigma}^{-1}$ into equation 2.7 results in,

$$\mathbf{x}_1' \hat{\Sigma}^{-1} (\hat{\mu}_k - \hat{\mu}_i) = 1.1382$$

$$\frac{1}{2}\hat{\mu_k}'\hat{\Sigma}^{-1}\hat{\mu_k} - \frac{1}{2}\hat{\mu_i}'\hat{\Sigma}^{-1}\hat{\mu_i} + ln\left(\frac{\hat{\pi_k}}{\hat{\pi_i}}\right) = -0.9898 \tag{2.8}$$

From equation 2.8, since $1.1382 > -0.9898$, the classification rule for LDA indicates that $\mathbf{x}_1$ is classified as belonging to class k which is the setosa class. The LDA classifier was trained on the Iris data and subsequently applied to the trained data set resulting in the confusion matrix shown in Table 2.5.

Table 2.5: Confusion Matrix for LDA.

	Predicted Class		
Actual Class	Class	Setosa	Versicolor
	Setosa	50	0
	Versicolor	0	50

LDA performs perfectly on the Iris data set. Figure 2.7 displays scatterplots of all the features from the Iris data set, observations belonging to the same class share the same colour. It is clear that the Iris data set is well separated, see for example the sepal width and petal length, thus explaining LDA's perfect performance.

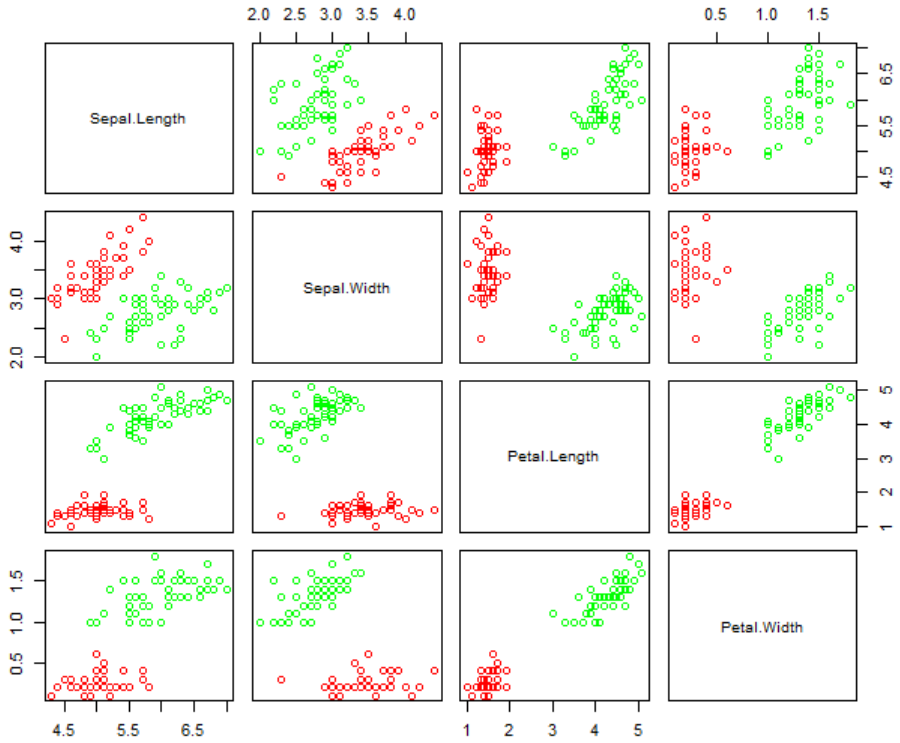


Figure 2.7: Scatterplots of the Iris Data Set.

Table 2.6 summarizes the performance of the LDA classifier on the Iris data set and Figure 2.8 displays the corresponding Receiver Operating Characteristics (ROC) Curve obtained. The LDA classifiers

performance on the data was perfect as seen in Table 2.6 all performance metrics are at their maximum (see Section 2.6). Similarly the ROC curve in Figure 2.8 indicates optimal performance. These results indicate that the data are linearly separable

Table 2.6: Summary of LDA's performance.

Performance Metric	Score
Precision	1
Recall	1
Accuracy	1
F1 score	1
Matthews Correlation Coefficient (MCC)	1
Area Under the Receiver Operating Characteristics Curve (AUC)	1

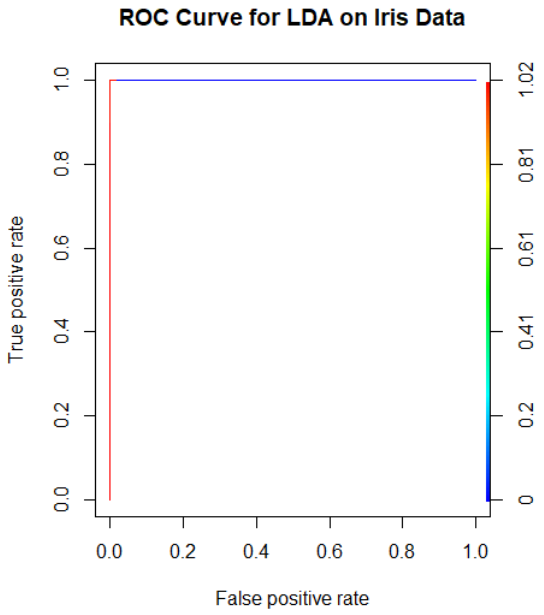


Figure 2.8: Receiver Operating Characteristics Curve for LDA applied to the Iris Data.

2.3 Binary Decision Trees

Decision trees are a versatile method used in both regression and classification problems. Decision trees are easy to interpret because they closely mimic the decision-making process of a human (Kotsiantis, 2013). The interpretability of decision trees may make them more favourable than other black-box methods, such as neural networks, which may have higher accuracy but are not as easy to interpret (Perner, 2011). Hunt et al. (1966) published the first paper using decision trees to model the human

concept of learning. Classification and regression trees (CART) were proposed by Breiman et al. (1984) and is considered the leading work for decision tree methods (Bertsimas and Dunn, 2017).

Figure 2.9 depicts a decision tree illustrating a decision-making process of buying a property. Decision trees have a tree-like structure consisting of a root node, nodes, splits and terminal nodes. The rectangular shapes depict the nodes. Node one is the first node in the tree and is called the root node. A question is asked at each node. Based on the outcome the instance is assigned to one of two child nodes. Consider node one, node two is reached if the property is in a safe neighbourhood. Node three is reached if the property is not in a safe neighbourhood. This process is repeated until a terminal node is reached. Any node produced from a split is called a child node. Nodes two and three are both children of node one. Any subsequent children of node two are descendants of node one. Thus node four is a descendant of node one. A node is terminal if it has no children nodes, for example nodes three, five, six and seven are terminal. Suppose a property is in a safe neighbourhood with good amenities but has less than two bedrooms. If the decision tree in Figure 2.9 is used to determine if a property will be purchased, node seven is reached. Node seven is terminal and hence the property will not be purchased.

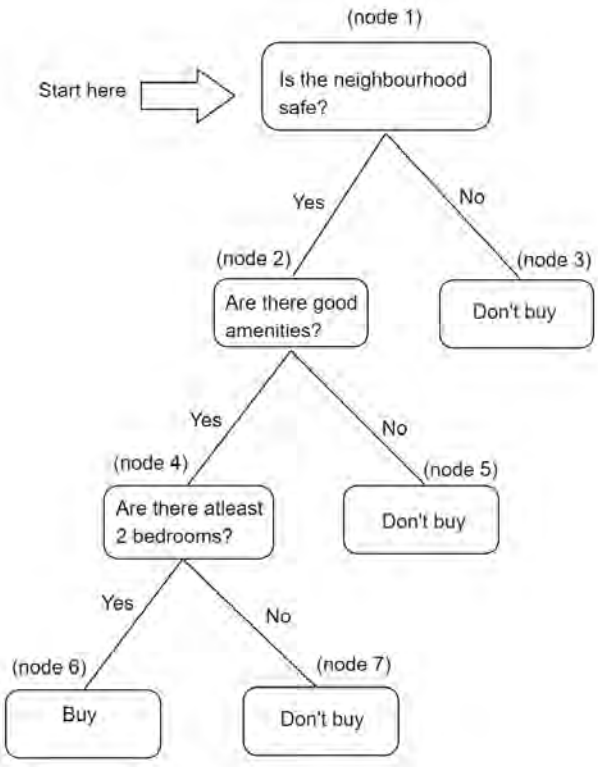


Figure 2.9: A Decision Tree for Buying a Property.

Decision trees output a prediction by asking a series of questions until a terminal node is reached.

Terminal nodes have a class label assigned to them; once a terminal node is reached the observation is given the class label of the terminal node.

2.3.1 Classification Trees

There are two types of decision trees, classification trees and regression trees. Classification trees are used for problems where the response variable is a categorical variable while regression trees are used for cases where the response variable is continuous (James et al., 2013). A classification tree is a rule for predicting the class an observation from its feature variables (Loh and Shih, 1997).

If $\mathbf{C} = \{\omega_1, \omega_2, ..., \omega_c\}$ denotes a set of classes, a classifier is a function or rule, $f(\mathbf{X}) \rightarrow C$, such that for every $\mathbf{x} \in \mathbf{X}$ we find some $\hat{y} = f(\mathbf{x}) = j$ for $j \in \mathbf{C}$.

Binary classification trees are constructed by recursively partitioning the training set $\mathbf{X} \in \mathbb{R}^{n \times p}$ into two mutually exclusive sets. The sets are recursively split until a terminal set or leaf node is reached. Figure 2.10 depicts a classification tree and the corresponding partitioning of the training set at each split. In the classification tree, the nodes are depicted by circles, the terminal nodes are depicted by squares. At each split a node is split into two mutually exclusive sets. In split one the root node is split into $\mathbf{X} = \mathbf{X}_1 \cup \mathbf{X}_2$. In split two node $\mathbf{X}_2$ is split into $\mathbf{X}_2 = \mathbf{X}_3 \cup \mathbf{X}_4$. In split three, $\mathbf{X}_3$ is split into $\mathbf{X}_3 = \mathbf{X}_5 \cup \mathbf{X}_6$. Since the splits are mutually exclusive, the sets $\mathbf{X}_i \cap \mathbf{X}_j = \emptyset$ where $i \neq j$ (Breiman et al., 1984). We can therefore consider a classification as a partition of $\mathbf{X}$ where $\mathbf{X} = \cup_j X_j$ where $\mathbf{X}_j = \{\mathbf{x} | f(\mathbf{x}) = j\}$. Nodes however are generally denoted by t_i, instead of $\mathbf{X}_i$. The right child node is denoted by t_R and the left child node is denoted by t_L, thus $t_p = t_L + t_R$ where t_p denotes the parent node.

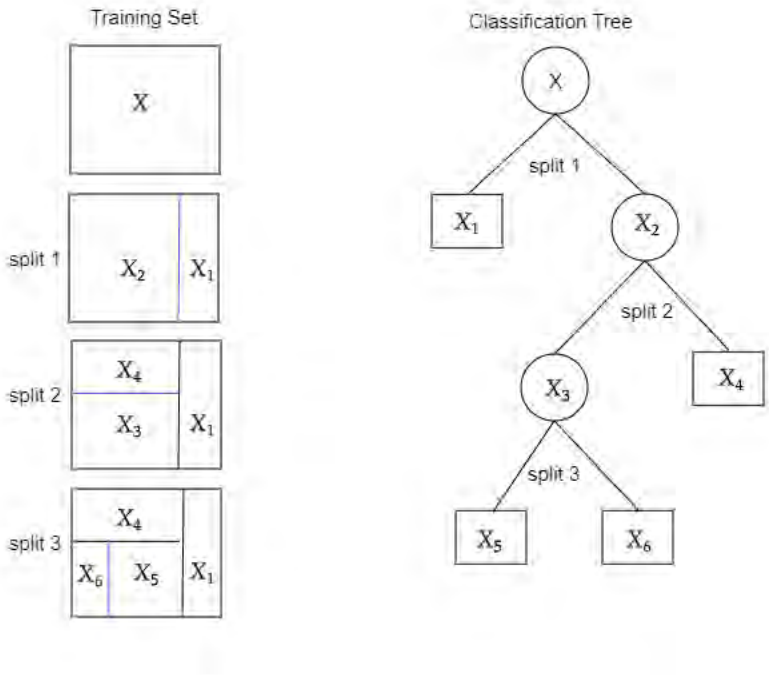


Figure 2.10: Diagram Depicting a Classification Tree and Partitioning of the Training Set.

Consider n, p-variate observations of the independent or feature variables $\mathbf{X}_1, \mathbf{X}_2, ..., \mathbf{X}_n$, with associated observations of the dependent or target variables $y_1, y_2, ..., y_n$. Consider a supervised learning scenario where the data are labeled and hence it is known that $y_j \in \mathbf{C} = \{\omega_1, \omega_2, ..., \omega_c\}$, where ω_j denotes the j^{th} categorical class label. These data are often called the learning sample, $\mathscr{L} = \{(\mathbf{x}_i, y_i) | \mathbf{x}_i \in \mathbf{X}, y_i \in \mathbf{C}\}$.

2.3.2 Growing a tree: Splitting Nodes

There are several important steps that require consideration when constructing a classification tree, for example the selection of splits at each node, deciding when to declare a node terminal and assigning a class label to each terminal node. At each non-terminal node, there is a set of s candidate splits. The splits are generated by conditions, constructed using the training set, and take the form is $\mathbf{X}_i \in \mathbf{A}$, where $\mathbf{A}$ is a subset of the training set $\mathbf{X}$. All observations that meet the condition are sent to child node t_L and all observations that do not are sent to child node t_R. Figure 2.11 depicts a split generated by the condition is $\mathbf{X}_i \in \mathbf{A}$. A univariate split, $\delta = \delta(t)$, is defined as a pair of a variable, $\mathbf{X}_i$ and a threshold value, c, thus $\delta = \delta(t) = (\mathbf{X}_i, c)$.

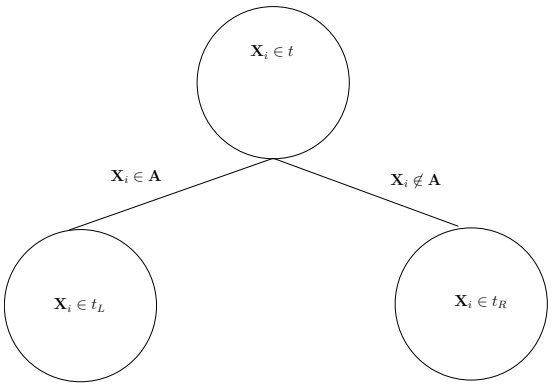


Figure 2.11: Diagram Depicting a Split.

Consider the n observations of the i^{th}quantitative feature variable $\mathbf{X}_i = [x_{i1}, x_{i2}, ..., x_{in}]'$. The observations are ordered such that $x_{ik} < x_{ik+1}$. For numerical variables the splits are generated by $x_{ik} \leq c$, where c is the midpoint between x_{ik} and x_{ik+1} for some $i \leq k \leq n$. For discrete feature variables, the set of all possible values $\mathbf{X}_i$ can take on is in $\mathbf{B} = \{b_1, b_2, ..., b_k\}$. Similarly for an encoded qualitative variable. The feature variable $\mathbf{X}_i$ can take on any of the values in $\mathbf{B}$, for example $x_{ik} = b_a$. The split is generated by is $\mathbf{x} \in \mathbf{D}$, where $\mathbf{D}$ is a non-empty subset of $\mathbf{B}$. (Loh and Shih, 1997)

Nodes are split such that the resulting nodes have a more homogeneous class distribution than the parent node (Laber et al., 2018). This is the idea of reducing impurity or equivalently of improving purity. Figure 2.12 depicts the feature space at the root node and subsequent children nodes of a decision tree. The observations in the feature space have two classes denoted by the blue squares and red circles. After each split the resulting nodes are more homogeneous in class distribution. This is called purity; the resulting children nodes are purer than the parent nodes. A node is pure if all the observations belong to the same class for example $\mathbf{X}_2$ corresponding to node t_2 in Figure 2.12 is pure as all the observations belong to the same class.

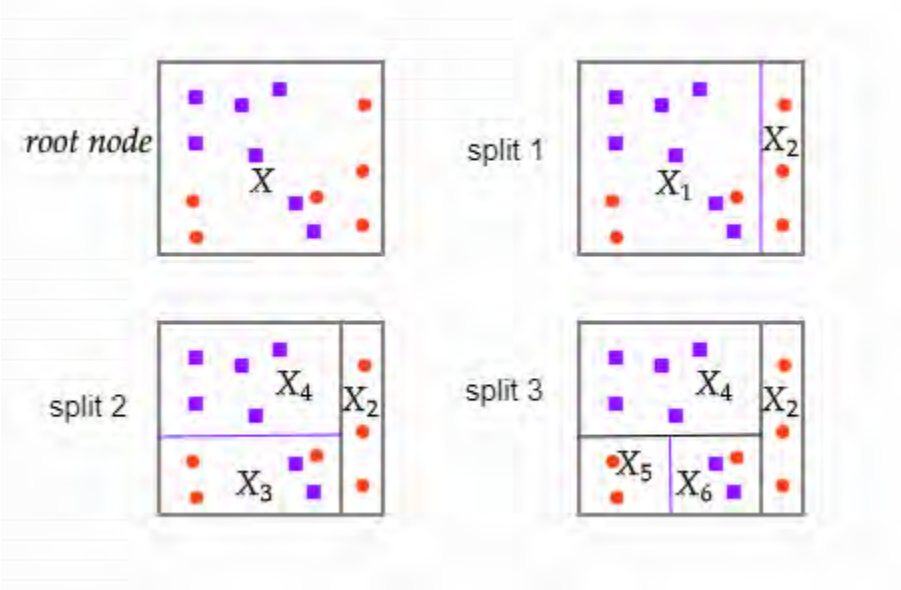


Figure 2.12: Impurity in the Feature Space (adapted from Singh (2023)).

Let n_j denote the number of observations belonging to class ω_j in the training data, and $n_j(t)$ denote the number of class ω_j observations in node t. Let $\pi_j = P(\omega_j)$ denote the prior probability of class ω_j. If these prior probabilities are unknown they can be estimated by the proportion of class samples in the training data, that is $\pi_j = P(\omega_j) \approx \frac{n_j}{n}$, where $n = n_1 + n_2 + ... + n_c$. The probability that an observation or instance is in node t given it is in class ω_j can be estimated as,

$$P(t|\omega_j) \approx \frac{n_j(t)}{n_j}.$$

This can be rewritten as

$$\begin{aligned} P(t|\omega_j) &= P\left(\frac{t \cap \omega_j}{\omega_j}\right) = p\left(\frac{(t_L \cup t_R) \cap \omega_j}{\omega_j}\right) \\ &= P\left(\frac{t_L \cap \omega_j}{\omega_j}\right) + P\left(\frac{t_R \cap \omega_j}{\omega_j}\right) \\ &= P(t_L|\omega_j) + P(t_R|\omega_j). \end{aligned}$$

Where previous studies have been conducted or a researcher has strong prior beliefs, these probabilities can be used and not the empirical frequencies. When the data in the training data are a random sample from a population where no prior information is available then it is reasonable to use the empirical frequency. By the law of total probability and Bayes rule, the probability that a sample is in node t is

estimated by,

$$\begin{aligned} P(t) &= P(\omega_1 \cap t) + P(\omega_2 \cap t) + ... + P(\omega_c \cap t) \\ &= P(t|\omega_1)P(\omega_1) + P(t|\omega_2)P(\omega_2) + ... + P(t|\omega_c)P(\omega_c) = \sum_{j=1}^{c} P(t|\omega_j)\pi_j \\ &\approx \sum_{j=1}^{c} \frac{n_j(t)}{n_j}\pi_j. \end{aligned} \tag{2.9}$$

As a result, an estimate of the probability of an instance sample being of class w_j given it is in node t is the node proportion

$$P(w_j|t) = \frac{P(t|\omega_j)P(\omega_j)}{P(t)} = \frac{\frac{n_j(t)P(\omega_j)}{n_j}}{\sum_{j=1}^{c} \frac{n_j(t)}{n_j} P(\omega_j)}. \tag{2.10}$$

If the prior class probabilities are assumed to be observed in the training data, $\pi_j = P(\omega_j) = \frac{n_j}{n}$, then from equation 2.10 we note that

$$P(\omega_j|t) = \frac{\frac{n_j(t)}{n_j}\left(\frac{n_j}{n}\right)}{\sum_{j=1}^{c} \frac{n_j(t)}{n_j}\left(\frac{n_j}{n}\right)} = \frac{\frac{n_j(t)}{n}}{\sum_{j=1}^{c} \frac{n_j(t)}{n}}$$

$$P(w_j|t) = \frac{n_j(t)}{n}\left(\frac{n}{\sum_{j=1}^{c} n_j(t)}\right) = \frac{n_j(t)}{n(t)}.$$

where $n(t) = \sum_j n_j(t)$ is the number of observations in node t.

Consider a split at node t, $\delta = \{X_i, s\}$. Let $t_L = \{\mathbf{x} \in t | X_i \in A\}$ and $t_R = \{\mathbf{x} \in t | X_i \notin A\}$.The probability of a sample falling in t_L can be estimated by the proportion of observations in the split s which fall in t_L, that is $p_L = \frac{\sum_j n_j(t_L)}{n(t)}$. Similarly $p_R = \frac{\sum_j n_j(t_R)}{n(t)}$, such that $p_L + p_R = 1$.

Let $i(t)$ denote measurement of impurity at node t. A method of guaranteeing purer child nodes is to select the split that results in the maximum decrease in impurity. The decrease in impurity is given by

$$\triangle i(s,t) = i(t) - (p_L i(t_L) + p_R i(t_R)) \tag{2.11}$$

where p_L is the proportion of observations in node t_L and p_R is the proportion of observations in node t_R. The quantity p_L is estimated by $\frac{\sum_j n_j(t_L)}{n}$, similarly p_R is estimated by $\frac{\sum_j n_j(t_R)}{n}$.

Several measures can be used to calculate the impurity, $i(t)$. Breiman et al. (1984), proposed using the Gini diversity index as a measure of impurity. The Gini index is given by,

$$\text{Gini}\,(t) = 1 - \sum_j P(\omega_j|t)^2.$$

Substituting equation 2.11 with the Gini index results in,

$$\triangle\text{Gini}(s,t) = \text{Gini}\,(t) - \left(p_L\text{Gini}(t_L) + p_R\text{Gini}(t_R)\right).$$

The Gini index criterion selects the split s that maximizes $\triangle\text{Gini}\,(s,t)$ at node t (Raileanu and Stoffel, 2004).

Consider the feature space $\mathbf{X} = \{\mathbf{X}_1, \mathbf{X}_2\}$ shown in Figure 2.13 on page 28, the raw data is depicted in Table 2.7.

Table 2.7: A Feature Space.

Observation	$\mathbf{X}_1$	$\mathbf{X}_2$	Y
1	1	3	1
2	1	4	1
3	5	9	1
4	6	4	1
5	1	7	2
6	1	9	2
7	3	6	2
8	3	7	2

The Gini criterion was used to generate the split and associated partition in Figure 2.14. The number of observations in each class are equal so we set $\pi_1 = \frac{4}{8} = \pi_2 = \frac{4}{8} = \frac{1}{2}$. From equation 25, the probability of an observation being in the root node is given by $P(t_1) = \sum_{j=1}^{c} \frac{n_j(t)}{n_j} P(\omega_i)$, where $P(\omega_j) = \frac{n_j}{n} = \pi_j$. For the root node the probability of an observation being in the node is, approximately

$$P(t) = \sum_{j=1}^{c} \frac{n_j(t)}{n_j}\pi_j, \text{hence}$$
$$P(t_1) = \frac{1}{2} \times \frac{4}{4} + \frac{1}{2} \times \frac{4}{4} = 1.$$

The probability of an observation belonging to class w_1 given node t_1 is given by

$$P(\omega_j|t_1) = \frac{n_j(t_1)}{n(t_1)}, \text{hence}$$
$$P(\omega_1|t_1) = \frac{4}{8} = \frac{1}{2} \text{and } P(\omega_2|t_1) = \frac{4}{8} = \frac{1}{2}.$$

The impurity of this node is,

$$\text{Gini}\,(t_1) = 1 - \sum_j P(\omega_j|t)^2 = 1 - (\frac{1}{2})^2 - (\frac{1}{2})^2 = \frac{1}{2}.$$

Recall that for numerical variables the splits are generated by $x_{i,k} \leq c$, where c is the midpoint between $x_{i,k}$ and $x_{i,k+1}$. Feature $\mathbf{X}_1$ without repeated values is given by, $\mathbf{X}_1 = [1,3,5,6]'$, thus $x_{1,1} = 1$ the next largest integer is $x_{1,2} = 3$. The first candidate split is $c = \frac{1+3}{2} = 2$, $\mathbf{X}_1 \leq 2$. For this split, the probability of an observation being in child nodes t_L and t_R is,

$$\begin{aligned} P(t) &= \sum_{j=1}^{c} \frac{n_j(t)}{n_j}\pi_j, \text{ thus} \\ P(t_L) &= \frac{1}{2} \times \frac{2}{4} + \frac{1}{2} \times \frac{2}{4} = \frac{1}{2}, \\ P(t_R) &= \frac{1}{2} \times \frac{2}{4} + \frac{1}{2} \times \frac{2}{4} = \frac{1}{2}. \end{aligned}$$

The class probabilities for node t_L are

$$\begin{aligned} P(\omega_j|t) &= \frac{n_j(t)}{n(t)}, \text{ hence} \\ P(\omega_1|t_L) &= \frac{2}{4} = \frac{1}{2}, \text{ and } P(\omega_2|t_L) = \frac{2}{4} = \frac{1}{2}. \end{aligned}$$

Similarly the class probabilities for node t_R are

$$P(\omega_1|t_R) = \frac{n_j(t_R)}{n(t)} = \frac{2}{4} = \frac{1}{2}, \text{and } P(\omega_2|t_R) = \frac{2}{4} = \frac{1}{2}.$$

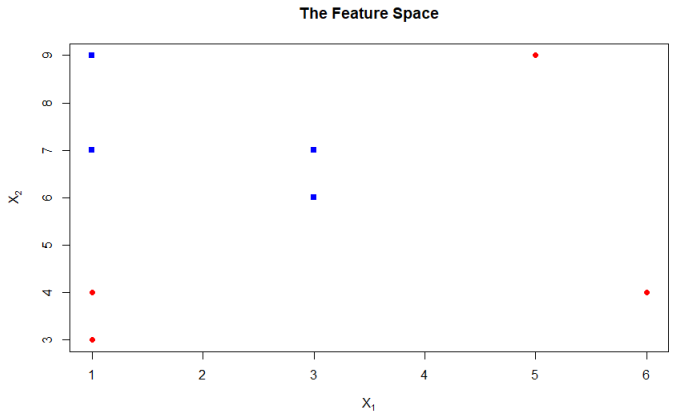
The Gini index of nodes t_L and t_R are,

$$\begin{aligned} \text{Gini}\,(t) &= 1 - \sum_j P(\omega_j|t)^2, \text{ hence} \\ \text{Gini}\,(t_L) &= 1 - (\frac{1}{2})^2 - (\frac{1}{2})^2 = \frac{1}{2}, \text{ and} \\ \text{Gini}\,(t_R) &= 1 - (\frac{1}{2})^2 - (\frac{1}{2})^2 = \frac{1}{2}. \end{aligned}$$

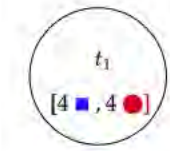
The decrease in impurity for split $\partial = \{X_1, s = 2\}$ is,

$$\begin{aligned} \triangle\text{Gini}(s,t) &= \text{Gini}\,(t) - (p_L\text{Gini}(t_L) + p_R\text{Gini}(t_R))\,, \\ &= \frac{1}{2} - (\frac{1}{2} \times \frac{1}{2} - \frac{1}{2} \times \frac{1}{2}) = 0. \end{aligned}$$

Similarly the decrease in impurity for all other candidate splits were calculated. The results are tabulated in Table 2.8.



(a) The Feature Space at the Root Node.



(b) The Classification Tree at the Root Node.

Figure 2.13: Constructing a Tree for the Data in Table 2.7.

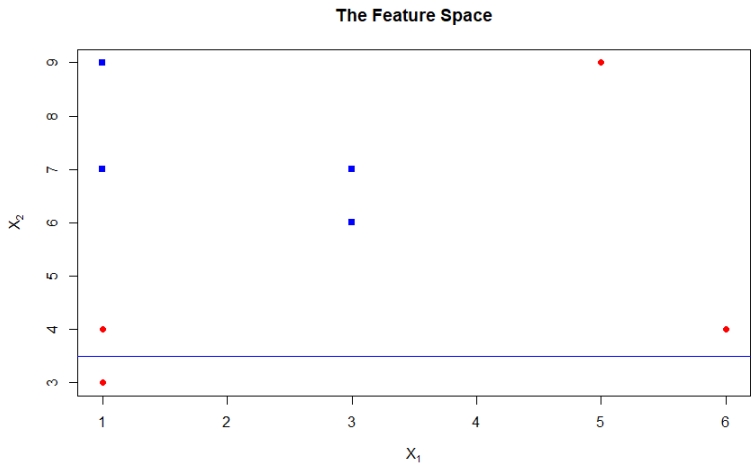
Table 2.8: Table of Gini Indexes at the Root Node.

Candidate Split	Gini(t_L)	Gini(t_R)	$\triangle$Gini(s,t)
$X_1 \leq 2$	$1-(\frac{1}{2})^2-(\frac{1}{2})^2=\frac{1}{2}$	$1-(\frac{1}{2})^2-(\frac{1}{2})^2=\frac{1}{2}$	0
$X_1 \leq 4$	$1-(\frac{2}{3})^2-(\frac{1}{3})^2=\frac{4}{9}$	$1-1=0$	0.1667
$X_1 \leq 5.5$	$1-(\frac{4}{7})^2-(\frac{3}{7})^2=\frac{24}{49}$	$1-1=0$	0.0714
$X_2 \leq 3.5$	$1-1=0$	$1-(\frac{4}{7})^2-(\frac{3}{7})^2=\frac{24}{49}$	0.4286
$X_2 \leq 5$	$1-1=0$	$1-(\frac{4}{5})^2-(\frac{1}{5})^2=\frac{5}{8}$	0.3
$X_2 \leq 6.5$	$1-(\frac{1}{4})^2-(\frac{3}{4})^2=\frac{3}{8}$	$1-(\frac{3}{4})^2-(\frac{1}{4})^2=\frac{3}{8}$	0.125
$X_2 \leq 7.5$	$1-(\frac{1}{2})^2-(\frac{1}{2})^2=\frac{1}{2}$	$1-(\frac{1}{2})^2-(\frac{1}{2})^2=\frac{1}{2}$	0

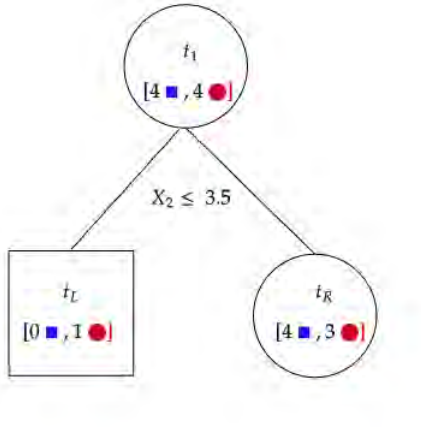
From Table 2.8, split $\mathbf{X}_2 \leq 3.5$ has the largest decrease in impurity and is thus chosen as the split at the root node. Figure 2.14 depicts the feature space and classification tree after the split. The child node t_L is pure and hence no further splitting is required. Child node t_R is not pure and hence the process is repeated again, resulting in Table 2.9.

Table 2.9: Table of Gini Indexes After Split One.

Candidate Split	Gini(t_L)	Gini(t_R)	$\triangle$Gini(s,t)
$X_1 \leq 2$	$1-(\frac{2}{3})^2-(\frac{1}{3})^2=\frac{4}{9}$	$1-(\frac{1}{2})^2-(\frac{1}{2})^2=\frac{1}{2}$	0.014
$X_1 \leq 4$	$1-(\frac{4}{5})^2-(\frac{1}{4})^2=\frac{8}{25}$	$1-1=0$	0.2612
$X_1 \leq 5.5$	$1-(\frac{2}{3})^2-(\frac{1}{3})^2=\frac{4}{9}$	$1-1=0$	0.1088
$X_2 \leq 5$	$1-1=0$	$1-(\frac{4}{5})^2-(\frac{1}{4})^2=\frac{8}{25}$	0.2612
$X_2 \leq 6.5$	$1-(\frac{1}{3})^2-(\frac{2}{3})^2=\frac{1}{2}$	$1-(\frac{3}{4})^2-(\frac{1}{4})^2=\frac{3}{8}$	0.085
$X_2 \leq 7.5$	$1-(\frac{3}{5})^2-(\frac{2}{5})^2=\frac{12}{25}$	$1-(\frac{1}{2})^2-(\frac{1}{2})^2=\frac{1}{2}$	0.001



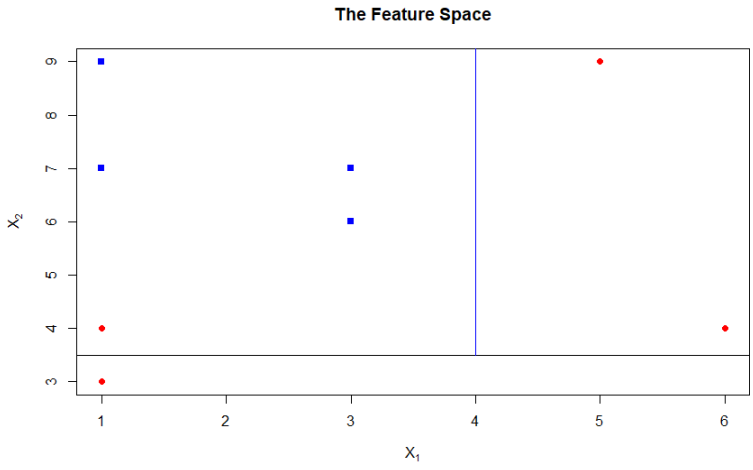
(a) The Feature Space After the First Split.



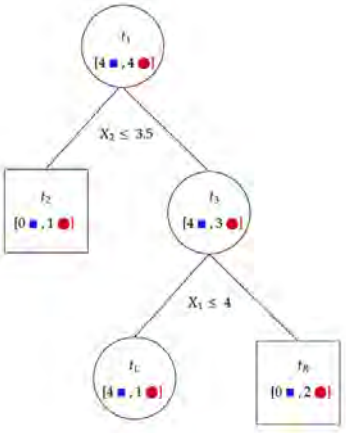
(b) The Classification Tree After the First Split.

Figure 2.14: Constructing a Tree for the Data in Table 2.7.

Candidate splits $X_2 \leq 5$ and $X_1 \leq 4$ both have the maximum decrease in impurity, hence the optimal split is chosen at random (Breiman et al., 1984). Figure 2.15 depicts the feature space and classification tree after the split. The child node t_R is pure and hence no further splitting is required. Child node t_L is not pure and the process is repeated again, resulting in Table 2.10.



(a) The Feature Space After Two Splits.



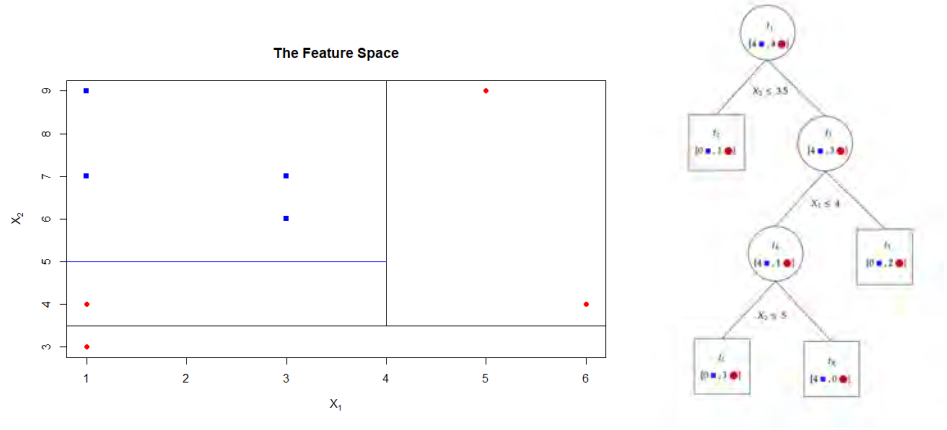
(b) The Classification Tree After Two Splits.

Figure 2.15: Constructing a Tree for the Data in Table 2.7.

Table 2.10: Table of Gini Indexes After Split Two.

Candidate Split	Gini(t_L)	Gini(t_R)	$\triangle$Gini(s,t)
$X_1 \leq 2$	$1-(\frac{2}{3})^2-(\frac{1}{3})^2=\frac{4}{9}$	$1-1=0$	0.053
$X_2 \leq 5$	$1-1=0$	$1-1=0$	0.32
$X_2 \leq 6.5$	$1-(\frac{1}{2})^2-(\frac{1}{2})^2=\frac{1}{2}$	$1-1=0$	0.12
$X_2 \leq 8$	$1-(\frac{3}{4})^2-(\frac{1}{4})^2=\frac{3}{8}$	$1-1=0$	0.02

Candidate split $X_2 \leq 5$ has the maximum decrease in impurity. This split results in Figure 2.16. All the terminal nodes are pure and hence no further splits are required.



(a) The Feature Space After the Third Split.

(b) The Classification Tree After the Third Split.

Figure 2.16: Constructing a Tree for the Data in Table 2.7.

Information gain (IG) is another criterion used to select the best split at a node. IG is based on entropy which measures the impurity of a node (Tangirala, 2020). The entropy of a node t, is given by,

$$\text{entropy}(t) = -\sum_j P(\omega_j|t) ln\left(P(\omega_j|t)\right).$$

IG measures the reduction of entropy, this is synonyms with the reduction of impurity (Raileanu and Stoffel, 2004). IG is given by,

$$IG(s,t) = \text{entropy}(t) - (p_L \text{entropy}(t_L) + p_R \text{entropy}(t_R)).$$

The split s that maximizes $IG(s,t)$ is the split selected at node t. Similar to the example above, IG can be utilized to grow a classification tree from the observations in Figure 2.13. As demonstrated above the root node has class probabilities, $p(t_1)=1$, $p(\omega_1|t_1)=\frac{1}{2}$ and $p(\omega_2|t_1)=\frac{1}{2}$. The impurity of the

root node is,

$$\text{entropy}(t) = -\sum_j P(\omega_j|t) ln\left(P(\omega_j|t)\right)$$
$$\text{entropy}(t_1) = -\frac{1}{2}ln(\frac{1}{2}) - \frac{1}{2}ln(\frac{1}{2}) = 0.6931.$$

For candidate split $X_1 \leq 2$, the class probabilities were found to be $p(\omega_1|t_L) = p(\omega_2|t_L) = p(\omega_1|t_R) = p(\omega_2|t_R) = \frac{1}{2}$. The entropy for the nodes t_R and t_L are;

$$\text{entropy}(t_R) = -\frac{1}{2}ln(\frac{1}{2}) - \frac{1}{2}ln(\frac{1}{2}) = 0.6931.$$
$$\text{entropy}(t_L) = -\frac{1}{2}ln(\frac{1}{2}) - \frac{1}{2}ln(\frac{1}{2}) = 0.6931.$$

The IG for the candidate split is,

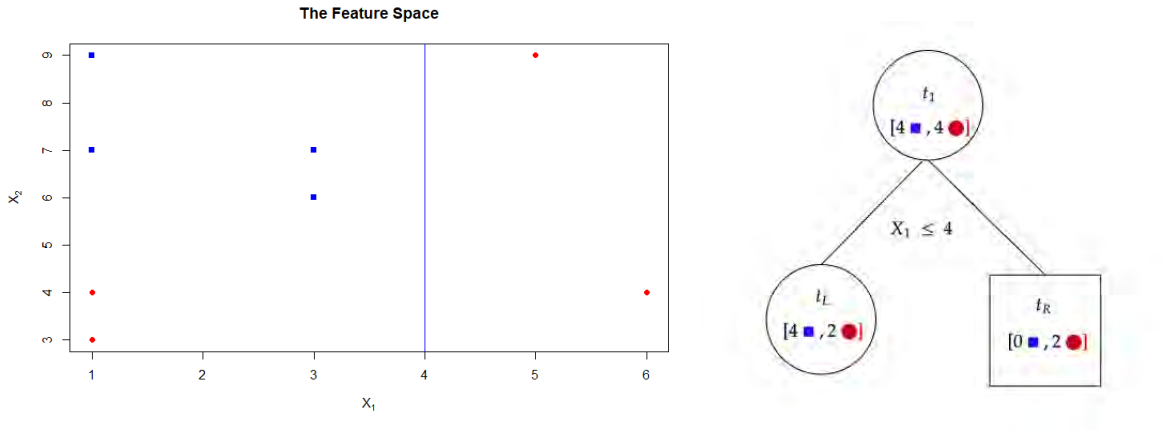
$$IG(s,t) = \text{entropy}(t) - (p_L\text{entropy}(t_L) + p_R\text{entropy}(t_R)),$$
$$= 0.6931 - \frac{1}{2} \times 0.6931 - \frac{1}{2} \times 0.6931 = 0.$$

Similarly for the other candidate splits, the entropy and IG was calculated and are tabulated in Table 2.11.

Table 2.11: The Entropy and IG at the Root Node.

Candidate Split	$entropy(t_L)$	$entropy(t_R)$	$IG(s,t)$
$X_1 \leq 2$	$-\frac{1}{2}ln(\frac{1}{2}) - \frac{1}{2}ln(\frac{1}{2}) = 0.6931$	$-\frac{1}{2}ln(\frac{1}{2}) - \frac{1}{2}ln(\frac{1}{2}) = 0.6931$	0
$X_1 \leq 4$	$-\frac{2}{3}ln(\frac{2}{3}) - \frac{1}{3}ln(\frac{1}{3}) = 0.096$	0	0.6212
$X_1 \leq 5.5$	$-\frac{4}{7}ln(\frac{4}{7}) - \frac{3}{7}ln(\frac{3}{7}) = 0.6829$	0	0.0956
$X_2 \leq 3.5$	0	$-\frac{4}{7}ln(\frac{4}{7}) - \frac{3}{7}ln(\frac{3}{7}) = 0.6829$	0.0956
$X_2 \leq 5$	0	$-\frac{4}{5}ln(\frac{4}{5}) - \frac{1}{5}ln(\frac{1}{5}) = 0.5004$	0.3128
$X_2 \leq 6.5$	$-\frac{1}{4}ln(\frac{1}{4}) - \frac{3}{4}ln(\frac{3}{4}) = 0.5623$	$-\frac{1}{4}ln(\frac{1}{4}) - \frac{3}{4}ln(\frac{3}{4}) = 0.5623$	0.1308
$X_2 \leq 7.5$	$-\frac{1}{2}ln(\frac{1}{2}) - \frac{1}{2}ln(\frac{1}{2}) = 0.6931$	$-\frac{1}{2}ln(\frac{1}{2}) - \frac{1}{2}ln(\frac{1}{2}) = 0.6931$	0

Candidate split $X_1 \leq 4$ has the largest IG and hence is used as the split at the root node. Figure 2.17 depicts the feature space and classification tree after the first split. Child node t_R is pure; no further splitting is required. Child node t_L is not pure; further splitting is required. The process is repeated resulting in Table 2.12.



(a) The Feature Space after One Split. **(b)** The Classification Tree after One Split.

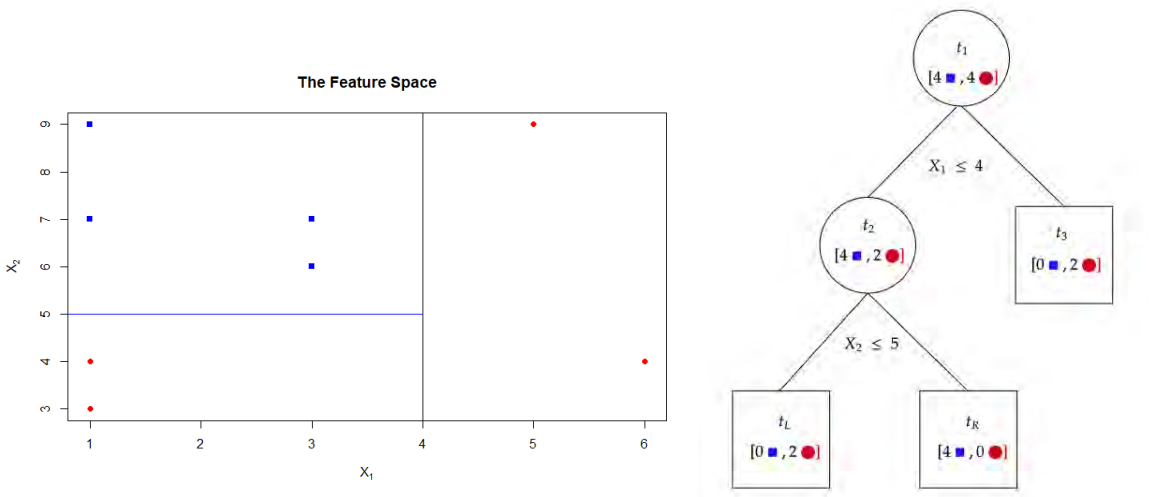
Figure 2.17: Constructing a Tree for the Data in Table 2.7.

Table 2.12: The Entropy and IG after the First Split.

Candidate Split	$entropy(t_L)$	$entropy(t_R)$	$IG(s,t)$
$X_1 \leq 2$	$-\frac{1}{2}ln(\frac{1}{2}) - \frac{1}{2}ln(\frac{1}{2}) = 0.6931$	0	0.1744
$X_2 \leq 3.5$	0	$-\frac{4}{5}ln(\frac{4}{5}) - \frac{1}{5}ln(\frac{1}{5}) = 0.5004$	0.2195
$X_2 \leq 5$	0	0	0.6365
$X_2 \leq 6.5$	$-\frac{2}{3}ln(\frac{2}{3}) - \frac{1}{3}ln(\frac{1}{3}) = 0.6365$	0	0.3183
$X_2 \leq 7.5$	$-\frac{2}{5}ln(\frac{3}{5}) - \frac{2}{5}ln(\frac{2}{5}) = 0.6730$	0	0.074

Candidate split $X_2 \leq 5$ has the greatest $IG(s,t)$ and is the split selected. This results in the feature space and classification tree depicted in Figure 2.18. All terminal nodes are pure; no further splitting is required.

Several splitting criteria can be used to grow decision trees. Mingers (1989) demonstrated that the splitting criterion used does not affect the classifier's accuracy. Stopping rules have been used to grow trees of the correct size. A tree is the correct size when it does not over-fit the training data (Martin, 1995). An example of a stopping rule is using some β as a threshold value. When the maximum decrease in impurity is less than β the node is declared terminal. If β is too large, the splitting may stop prematurely. Splits that may result in a significant reduction in impurity may occur later on and will not be included in the tree. If β is too small, the tree grows too large and over-fits the training data (Breiman et al., 1984). Trees are instead grown to be as large as possible resulting in the tree T^{max}, the tree is then pruned to get the right-sized tree (Martin, 1995).



(a) The Feature Space after the Second Split.

(b) The Classification Tree after the Second Split.

Figure 2.18: Constructing a Tree for the Data in Table 2.7.

Let $\text{cost}(\omega_i|\omega_j)$ denote the cost of missclassifying a class ω_j observation as a class ω_i observation. The cost of missclassification is used for cases where the loss in missclassifying a class ω_j observation as a class ω_i observation, is not the same as missclassifying a class i observation as a class j observation. The term $\text{cost}(\omega_i|\omega_j)$ is a set of positive numbers and is equal to zero if $\omega_i = \omega_j$ (Breiman et al., 1984). If ω_i is the class label of node t, the probability of missclassification at node t is,

$$\sum_{\omega_j \neq \omega_i} P(\omega_j|t).$$

The expected missclassification cost at node t is,

$$\sum_{\omega_j \neq \omega_i} \text{cost}(\omega_i|\omega_j) P(\omega_j|t).$$

Nodes are assigned the class label ω_i to minimize the expected missclassification cost at node t. In the case where $\text{cost}(\omega_i|\omega_j) = 1$, this is given by,

$$\begin{aligned} r(t) &= \min_{\omega_i} \sum_{\omega_j \neq \omega_i} \text{cost}(\omega_i|\omega_j) P(\omega_j|t) \\ &= \min_{\omega_i} (1 - P(\omega_i|t)) \end{aligned}$$

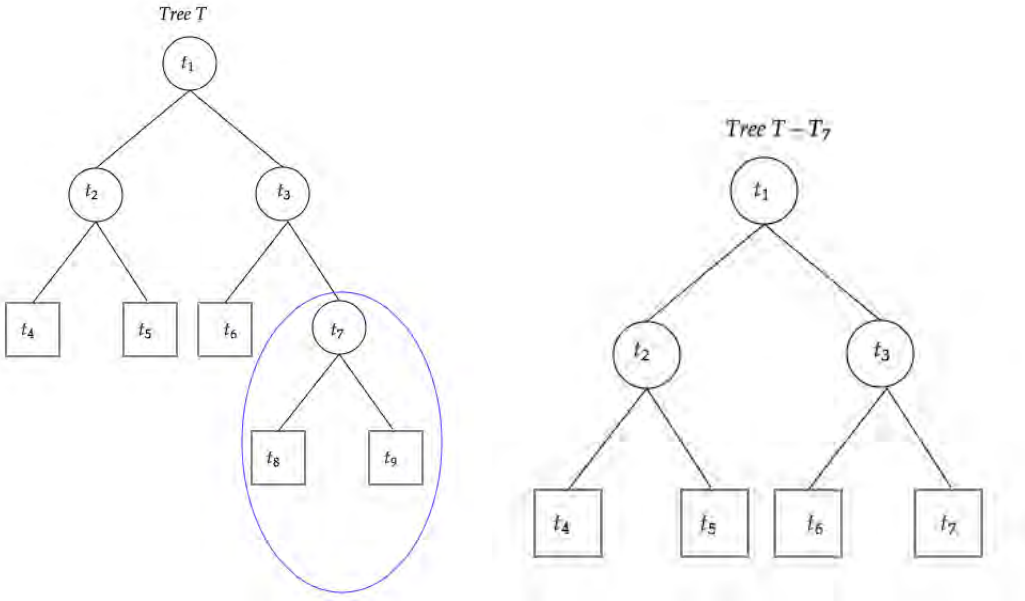
since $\sum_{\omega_j} P(\omega_j|t) = 1$ and $\text{cost}(\omega_i|\omega_j) = 1$. Minimizing the term $\sum_{\omega_j \neq \omega_i} P(\omega_j|t) = \min_{\omega_i} (1 - P(\omega_i|t))$ is the same as maximizing $P(\omega_i|t)$. The term $P(\omega_i|t)$ is maximized when the number of class ω_i observations are at a maximum. Using this result the most common class in node t is selected as the class label for

a terminal node. (Breiman et al., 1984)

2.3.3 Pruning a Tree

Right-sized trees are obtained by pruning the overly large tree T^{max}. Pruning decreases the classification tree's dependency on the training data. The complexity of a tree is determined by its size; the larger the tree, the more complex it is. After a certain amount of splits the accuracy of a classification tree no longer improves significantly whilst the complexity increases. (Breiman et al., 1984)

Pruning removes branches from a tree; the resulting trees are called sub-trees. Consider the classification trees depicted in Figure 2.19. In Figure 2.19a branch T_7 is circled and contains nodes t_7, t_8 and t_9. Pruning branch T_7 from the classification tree results in the tree depicted in Figure 2.19b. In tree $T - T_7$ all the descendants of node t_7 are removed from the tree. This tree is called a pruned sub-tree of T which is denoted by $T > T - T_7$.



(a) Tree T before Pruning. **(b)** Tree T after Pruning.

Figure 2.19: Pruning a Classification Tree (adapted from Breiman et al. (1984)).

Breiman et al. (1984) proposed minimal cost-complexity as a pruning algorithm for decision trees. Minimal cost complexity pruning can be explained in two steps. First a series of sub-trees $T^1 > T^2 > ... > t_1$, are produced by pruning the tree T^{max}. The tree T^{i+1} is a sub-tree of T^i t_1 is the smallest sub-tree of T^{max}, the root node of the tree. After pruning, the best sub-tree T^{i_0} is selected by using an independent data set to estimate the missclassification cost. The sub-tree with the smallest missclassification cost is selected as the right-sized sub-tree. The right-sized sub-tree is used to classify unseen data. Breiman et al. (1984)

Minimal cost-complexity pruning searches for a classification tree that is not overly complex and minimizes the missclassification cost. The number of terminal nodes determines the complexity of a tree. Larger trees have more terminal nodes and are more complex. The complexity of a tree T is denoted by $|\widetilde{T}|$ where $\widetilde{T}$ denotes the terminal nodes (Breiman et al., 1984). In Figures 2.19a and 2.19b the complexity $|\widetilde{T}|$ is five as there are five terminal nodes, similarly the complexity of tree $|T - \widetilde{T_7}|$ is four. $R(T)$ is an estimate for the missclassification cost of a tree T $R(T)$denotes the fraction of cases in the training data that are missclassified by tree T;

$$R(T) = \sum_{t \varepsilon \widetilde{T}} r(t) p(t)$$

where $r(t)$ denotes the expected missclassification cost. The complexity parameter, defined by α, is a positive real number. The cost-complexity measure is given by

$$R_\alpha(T) = R(T) + \alpha|\widetilde{T}|. \tag{2.12}$$

The cost complexity measure penalizes a tree for its complexity and its overall missclassification cost. The smallest minimizing sub-tree $T(\alpha)$, for values of α, is defined as,

$$R_\alpha\left(T(\alpha)\right) = \min_{T^{max} > T} R_\alpha(T).$$

If multiple trees minimize $R_\alpha(T)$ then the smallest sub-tree is selected for $T(\alpha)$.

Tree T^1 is the smallest sub-tree of T^{max} that satisfies $R(T^1) = R(T^{max})$. Let t_L and t_R be any two terminal nodes which are descendants of node t. T^1 is obtained by pruning any terminal nodes of T^{max}, t_L and t_R that satisfy $R(t) = R(t_L) + R(t_R)$.

Minimal cost complexity prunes trees progressively starting with $T^1 > T^2 > ... > t_1$ and ending with the root node t_1. For any node t define,

$$R(t) = \sum_{\omega_j \neq \omega_i} cost\left(\omega_i|\omega_j\right) P(t), \text{and}$$
$$R_\alpha(t) = R(t) + \alpha.$$

In the the case where $cost\left(\omega_i|\omega_j\right) = 1$, then $R(t) = \left(1 - max_{\omega_j} P(\omega_j|t)\right)$. For any branch T_t define,

$$R(T_t) = \sum_{t \in \widetilde{T}_t} R(t), \text{and}$$
$$R_a(T_t) = R(T_t) + \alpha|\widetilde{T_t}|.$$

If $R_\alpha(T_t) < R_\alpha(t)$ the branch T_t has a smaller cost complexity than node t and it is not pruned. When $R_\alpha(T_t) = R_\alpha(t)$ the tree produced from pruning T_t is smaller than the tree with branch T_t, the branch is then pruned from the tree. When $R_\alpha(T_t) > R_\alpha(t)$ the node has a smaller cost complexity and the branch T_t is pruned.

Sub-trees are produced by finding the value of α such that $R_\alpha(T_t) = R_\alpha(t)$, resulting in the branch T_t being pruned. This method is repeated until only the root node is left. This method is called weakest link cutting (Breiman et al., 1984). The weakest link is the first node t, that as the value of α increases will have the same cost-complexity as the branch T_t, resulting in the branch being pruned. Finding the value of α such that $R_\alpha(T_t) = R_\alpha(t)$ is done by solving the inequality,

$$\begin{aligned} R_\alpha(T_t) &< R_\alpha(t), \text{and} \\ R(T_t) + \alpha|\widetilde{T}_t| &< R(t) + \alpha, \text{and} \\ \alpha &< \frac{R(t) - R(T_t)}{|\widetilde{T}_t| - 1}. \end{aligned}$$

Let $w_i(t)$ be a function given by,

$$w_i(t) = \begin{cases} \frac{R(t)-R(T_t)}{|\widetilde{T}_t|-1} & t \notin \widetilde{T^i}, \\ \infty & t \in \widetilde{T^i}. \end{cases}$$

For sub-tree T^i the weakest link $\overline{t}_i$ is given by,

$$w_i(\overline{t}_i) = \min_{t \in T^i} w_i(t).$$

The weakest link $\overline{t}_i$ results in branch $T_{\overline{t}_i}$ being pruned from the tree, $T^{i+1} = T^i - T_{\overline{t}_i}$. The value of the function $w_i(\overline{t}_i)$ is the value of α at which the two cost-complexities become equal; we set $\alpha_{i+1} = w_i(\overline{t}_i)$. If there are multiple weakest links then the sub-tree is produced by pruning all the branches simultaneously. This weakest link pruning process is repeated for all sub-trees until only the root node remains. The weakest link pruning method produces a series sub-trees $T^1 > T^2 > ... > t_1$ and their corresponding increasing values of α: $\alpha_1 < \alpha_2 < ... < \alpha_k$, where, $\alpha_1 = 0$, $\alpha_2 = w_1(\bar{t})$ and $\alpha_{i+1} = w_i(\bar{t})$.

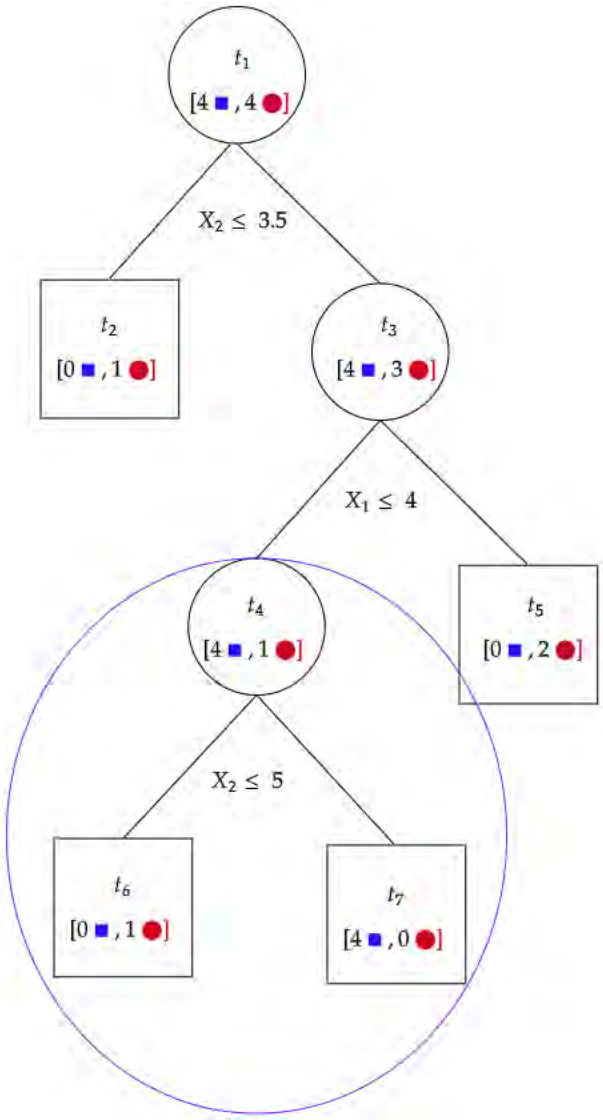


Figure 2.20: Diagram of a Classification Tree.

Consider the classification tree T^{max} depicted in Figure 2.20. The aim is to use the weakest-link cutting algorithm to prune the classification tree generating a series of sub trees, $T^1 > T^2 > ... > t_1$ and corresponding values of α, $\alpha_1 < \alpha_2 < ... < \alpha_k$. To produce the next larget subtree node t_4 has to be pruned. Starting at node t_4 branch T_{t_4} consists of nodes t_6 and t_7, branch T_{t_4} is circled in Figure 2.20. For node t_4 the probability that an observation is in the node is,

$$P(t_4) = \sum_{j=1}^{c} \frac{n_j(t)}{n_j} \pi_j = \frac{1}{2} \times \frac{4}{4} + \frac{1}{2} \times \frac{1}{4} = 0.625.$$

The class probabilities are,

$$P(\omega_j|t) = \frac{n_j(t)}{n(t)}, \text{that is}$$
$$P(\omega_1|t_4) = \frac{4}{5} = 0.8, \text{and} P(\omega_2|t_4) = \frac{1}{5} = 0.2.$$

The estimated missclassification cost of node t_4 is,

$$R(t_4) = [1 - \max_i P(j|t)] \times P(t) = (1 - 0.8) \times 0.625 = 0.125.$$

For node t_6 the probability that an observation is found in the node is,

$$P(t_6) = \sum_{j=1}^{c} \frac{n_j(t)}{n_j} \pi_j = \frac{1}{2} \times \frac{0}{4} + \frac{1}{2} \times \frac{1}{4} = \frac{1}{8}.$$

The class probabilities are,

$$P(\omega_j|t) = \frac{n_j(t)}{n(t)}, \text{hence}$$
$$P(\omega_1|t_6) = \frac{0}{1} = 0, \text{and} P(\omega_2|t_6) = \frac{1}{1} = 1.$$

The estimated missclassification cost of node t_6 is,

$$R(t_6) = [1 - \max_i P(j|t)] \times P(t) = (1 - 1) \times \frac{1}{8} = 0.$$

For node t_7 the probability that an observation reaches the node is,

$$P(t_7) = \sum_{j=1}^{c} \frac{n_j(t)}{n_j} \pi_j = \frac{1}{2} \times \frac{4}{4} + \frac{1}{2} \times \frac{0}{4} = \frac{1}{2}.$$

The class probabilities are,

$$P(\omega_j|t) = \frac{n_j(t)}{n(t)}, \text{hence}$$
$$P(\omega_1|t_7) = \frac{4}{4} = 1, \text{and} P(\omega_2|t_7) = \frac{0}{4} = 0.$$

The estimated missclassification cost of node t_7 is,

$$R(t_7) = [1 - \max_i P(j|t)] \times P(t) = (1 - 1) \times \frac{1}{2} = 0.$$

The estimated missclassification cost of branch T_{t_4} is,

$$R(T_{t_4}) = \sum_{t \in \widetilde{T}_t} R(t) = R(t_6) + R(T_7) = 0.$$

The weakest link algorithm starts pruning from sub-tree T^1 which is the smallest sub-tree of T^{max} that satisfies $R(T_1) = R(T^{max})$. In tree T^{max} any leaf nodes t_L and t_R that satisfy $R(t) = R(t_L) + R(t_R)$ are pruned resulting in T^1. Node t_4 is the only node with two terminal child nodes. It has been shown that $R(t_4) = 0.125$, $R(t_L) = R(T_6) = 0$ and $R(t_R) = R(T_7) = 0$. Thus $R(t_4) \neq R(t_6) + R(t_7)$, no nodes need to be pruned and the sub-tree T^1 is obtained. Using the weakest link algorithm the value of α that node $R(t_4) = R(T_{t_4})$ is,

$$w_1(t_4) = \frac{R(t) - R(T_t)}{|\widetilde{T}_t| - 1} = \frac{0.125 - 0}{2 - 1} = 0.125.$$

Similarly For nodes t_3, t_1 and their corresponding branches T_{t_3} and T_{t_1}. The following values of the function $w_1(t)$ were found and are tabulated in Table 2.13.

Table 2.13: Values of $w_1(t)$ for Sub-Tree T^1.

Node	Branch	$R(t)$	$R(T_t)$	$w_1(t)$
t_1	T_{t_1}	$R(t_1) = 0.5$	$R(T_{t_1}) = 0$	$\frac{0.5-0}{4-1} = 0.1667$
t_3	T_{t_3}	$R(t_3) = 0.375$	$R(T_{t_3}) = 0$	$\frac{0.375-0}{3-1} = 0.1875$
t_4	T_{t_4}	$R(t_4) = 0.125$	$R(T_{t_4}) = 0$	$\frac{0.125-0}{2-1} = 0.125$

The weakest link is the node $\overline{t}_i$ that satisfies $w_i(\overline{t}_i) = \min_{t \in T^i} w_i(t)$. From Table 2.13 node t_4 is the weakest link as it minimizes $w_1(\overline{t}_i)$. Branch T_{t_4} is pruned from T^1 and set $\alpha_2 = w_1(t_4) = 0.125$. Tree $T^2 = T^1 - T_{t_4}$ is depicted in Figure 2.21. The weakest link process is repeated again on this tree, Figure 2.21, resulting in Table 2.14

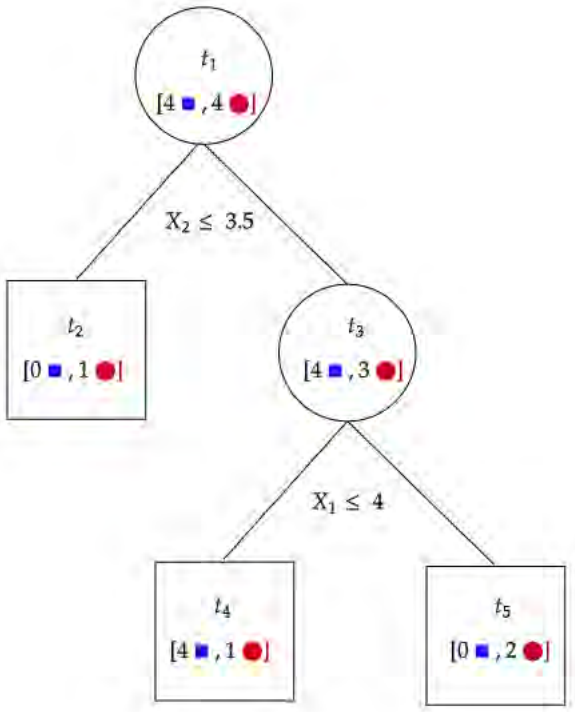


Figure 2.21: Sub-Tree T^2.

Table 2.14: Values of $w_2(t)$ for Sub-Tree T^2.

Node	Branch	$R(t)$	$R(T_t)$	$w_2(t)$
t_1	T_{t_1}	$R(t_1) = 0.5$	$R(T_{t_1}) = 0.125$	$\frac{0.5-0.125}{3-1} = 0.1875$
t_3	T_{t_3}	$R(t_3) = 0.375$	$R(T_{t_3}) = 0.125$	$\frac{0.375-0.125}{2-1} = 0.25$

From Table 2.14, node t_1 is the weakest link as it minimizes $w_2(t)$. Branch T_{t_1} is pruned from T^2 resulting in $t_1 = T^2 - T_{t_1}$ and set $\alpha_3 = w_2(t_1) = 0.1875$. All sub-trees have been obtained as after pruning branch T_{t_1} only the root node remains. All the sub-trees and their corresponding values of α are tabulated in Table 2.15.

Table 2.15: All Sub-Trees and their Corresponding Values of α.

Sub-Tree	α_i
T^1	$\alpha_1 = 0$
T^2	$\alpha_2 = 0.125$
t_1	$\alpha_3 = 0.1875$

Thus all the sub-trees, $T^1 > T^2 > ... > t_1$ and their corresponding values of α have been obtained through the weakest link algorithm. The right-sized tree that will be used for classification needs to

be determined. The sub-tree with the lowest missclassification cost is the sub-tree selected as the best sub-tree. The estimate $R(T)$ however is not a suitable estimate for missclassification cost. As the biggest sub-tree will be favoured as $R(t) \geq R(t_L) + R(t_R)$ (Breiman et al., 1984). This statement is proved below as follows: Recall $R(t) = r(t)p(t)$. where $r(t)$ is at a minimum. This can be written as

$$R(t) = \sum_{\omega_j \neq \omega_j(t)} \text{cost}\left(\omega_j(t)|\omega_j\right) \text{P}(\omega_j, t)$$

where $\omega_j(t)$ is the class label at node t. Recall $t = t_L \cup t_R$ and hence $P(\omega_j, t)$ can be re written as

$$\begin{aligned} P(\omega_j, t) &= P(\omega_j \cap t) = P\left(\omega_j \cap (t_L \cup t_R)\right) \\ &= P(\omega_j \cap t_L) \cup P(\omega_j \cap t_R) = P(\omega_j \cap t_L) + P(\omega_j \cap t_R) \\ &= P(\omega_j, t_L) + P(\omega_j, t_R). \end{aligned}$$

$R(t)$ can be written as $\sum_{\omega_j \neq \omega_j(t)} \text{cost}\left(\omega_j(t)|\omega_j\right)\left(P(\omega_j, t_L) + P(\omega_j, t_R)\right)$, then $R(t) - R(t_L) - R(t_R)$ is

$$\begin{aligned} R(t) - R(t_L) - R(t_R) = &\sum_{\omega_j \neq \omega_j(t)} \text{cost}\left(\omega_j(t)|\omega_j\right) P(\omega_j, t_L) - \sum_{\omega_j \neq \omega_j(t_L)} \text{cost}\left(\omega_j(t_L)|\omega_j\right) P(\omega_j, t_L) \\ &+ \sum_{\omega_j \neq \omega_j(t)} \text{cost}\left(\omega_j(t)|\omega_j\right) P(\omega_j, t_R) - \sum_{\omega_j \neq \omega_j(t_R)} \text{cost}\left(\omega_j(t_r)|\omega_j\right) P(\omega_j, t_R). \end{aligned}$$

Using the class label assignment rule the term $\sum_{\omega_j \neq \omega_j(t_L)} \text{cost}\left(\omega_j(t_L)|\omega_j\right) P(\omega_j, t_L)$ is always at a minimum as the selected class label of a node minimizes the expected missclassification cost. The term $\sum_{\omega_j \neq \omega_j(t)} \text{cost}\left(\omega_j(t)|\omega_j\right) P(\omega_j, t_L)$ however is only at a minimum when $\omega_j(t) = \omega_j(t_L)$ as the class $\omega_j(t)$ is the class label of the parent node t and not necessarily the class label of node t_L. If $\omega_j(t) \neq \omega_j(t_L)$, then

$$\sum_{\omega_j \neq \omega_j(t)} \text{cost}\left(\omega_j(t)|\omega_j\right) P(\omega_j, t_L) > \sum_{\omega_j \neq \omega_j(t_L)} \text{cost}\left(\omega_j(t_L)|\omega_j\right) P(\omega_j, t_L).$$

Similarly if $\omega_j(t) \neq \omega_j(t_R)$, then

$$\sum_{\omega_j \neq \omega_j(t)} \text{cost}\left(\omega_j(t)|\omega_j\right) P(\omega_j, t_R) > \sum_{\omega_j \neq \omega_j(t_R)} \text{cost}\left(\omega_j(t_R)|\omega_j\right) P(\omega_j, t_R).$$

Thus $R(t) - R(t_L) - R(t_R) \geq 0$ and is equal to zero only if $\omega_j(t) = \omega_j(t_L) = \omega_j(t_R)$. (Breiman et al., 1984)

Since $R(t) \geq R(t_L) + R(t_R)$ a better estimate for the missclassification cost is needed. Breiman et al. (1984) proposed using a test sample or k-fold cross validation to estimate the true missclassification

cost. A test sample can be used to estimate the total missclassification cost by randomly partitioning the training data into two sets. One of the sets will be a test set and the other set will be the new training set. The classification tree T^{max} is grown on the training set. The tree T^{max} is pruned using the weakest link algorithm resulting in a series of sub-trees $T^1 > T^2 > ... > t_1$ and their corresponding values of α. For each sub-tree, the observations are classified in the test data set and estimate the missclassification cost. The missclassification cost is estimated by,

$$R_{test}(T) = \frac{\sum_{i,j} \text{cost}(\omega_i|\omega_j) n_{ij}}{n} \tag{2.13}$$

where n_{ij} is the number of observations belonging to class ω_i incorrectly classified as belonging to class ω_j. Using the test sample search for the sub-tree and corresponding value of α such that,

$$R_{test}(T^{i_0}) = \min_i R_{test}(T^i).$$

The tree T^{i_0} is the right-sized tree used to classify unseen data.

Breiman et al. (1984) also proposed using k-fold cross validation to estimate the true missclassification cost of a tree. K-fold cross-validation is done by partitioning the training set L into k sets of equal size. These sets are called folds and are denoted by L_k. The $k'th$ learning sample, L^k contains $\frac{k-1}{k}$ of the total training sample L and is found by,

$$L^k = L - L_k.$$

The learning sample L^k does not contains data from the $k'th$ fold. The k learning samples are used to grow k trees in conjunction to T^{max} which is grown on L. The k trees are denoted by T_k^{max}. Let $T_k(\alpha)$ and $T(\alpha)$ be the minimal cost complexity sub-trees of T_k^{max} and T^{max} respectively. For each fold the series of sub-trees $T_k^1(\alpha) > T_k^2(\alpha) > ... > t_1(\alpha)$ and their corresponding values of α, $\alpha_1^k < \alpha_2^k < \ldots < \alpha_c^k$ can be found by using the weakest-link pruning algorithm.

Since T_k^{max} is grown on L^k the fold L_k can be used as an independent test sample, as L^k does not contain data from L_k. If k is large the performance of the sub-trees $T(\alpha)$ can be estimated by aggregating the performance of the k trees, $T_k(\alpha)$. Let n_{ij}^k be the number of class ω_i observations incorrectly classified by $T_k(\alpha)$ as belonging to class ω_j. The total number of missclassified observations is $n_{ij} = \sum_k n_{ij}^k$, let n_j be the total number of ω_j observations in the test sample. The missclassification cost of the tree $T(a)$ is estimated by,

$$R^{cv}(T(\alpha)) = \frac{\sum_{i,j} \text{cost}(\omega_i|\omega_j) n_{ij} \times \pi_j}{n_j}.$$

Note for any α in the interval $\alpha_i \leq \alpha < \alpha_{i+1}$ the pruned sub-trees $T(\alpha)$ are all the same. For each sub-tree $T(\alpha)$ with $\alpha_i \leq \alpha < \alpha_{i+1}$ Breiman et al. (1984) proposed finding the geometric mean and using that as the value of α. For $T(\alpha)$ with α in the interval $\alpha_i \leq \alpha < \alpha_{i+1}$ define $\alpha_i^{'} = \sqrt{\alpha_i \times \alpha_{i+1}}$

then set $T_i = T(\alpha_i^{'})$. The cross-validation missclassification cost for tree T_i is then,

$$R_{cv}(T_i) = R_{cv}(T(\alpha_i^{'})).$$

The tree T_{i_0} that minimizes $R_{cv}(T_i)$, is then selected as the right-sized tree used to classify unseen data.

2.4 An Ensemble of Classifiers

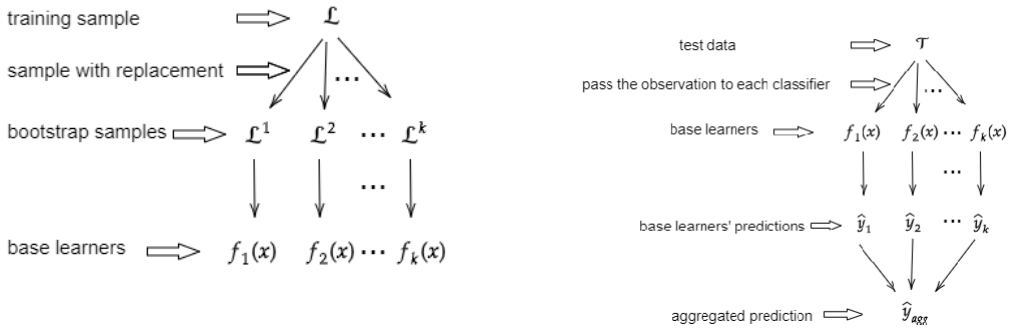
An ensemble is a set of predictors trained on the same problem, where in the context of classification the predictors are classifiers. For an observation each classifier's prediction is combined to produce one prediction. An ensemble is homogeneous if all classifiers in the ensemble are of the same type, for example if an ensemble consists of only neural networks. For homogeneous ensembles, the individual classifiers are called base learners. (Zhou, 2012)

Opitz and Maclin (1999) define an ensemble as ideal if the base learners are accurate and produce different predictions to the other base learners. They indicate that nothing is gained from combining the predictions of identical base learners. Their research found that ensembles often have a better predictive accuracy than any of the base learners in the ensemble, making them favourable statistical learning techniques. Ensembles can consist of weak or strong learners. In the context of binary classification a weak learner is any classifier that performs slightly better than random; the classification error should be less than a half (Freund et al., 1999).

2.4.1 Bagging Classifiers

Breiman (1996b) proposed bootstrap aggregation (Bagging) as a method for generating homogeneous ensembles. Bagging has become a popular ensemble method (Opitz and Maclin, 1999). From one learning sample, Bagging produces a set of learning samples to train the base learners and generate an ensemble.

Figure 2.22 adapted from Mert et al. (2014) illustrates the Bagging process. Consider a labeled training set, $\mathscr{L} \in \mathscr{R}^{n\times p}$ and a test set, $\mathscr{T} \in \mathscr{R}^{m\times p}$. A "new" training sample $\mathscr{L}^i \in \mathscr{R}^{n\times p}$ is produced by randomly sampling with replacement from the original training set for $i = 1,2...,k$. This sampling technique is called bootstrapping (Efron and Tibshirani, 1994). The new sample, $\mathscr{L}^i$, is called a bootstrap sample and is the same size as $\mathscr{L}$. Each bootstrap sample may contain duplicate instances and may omit some instances from the original training data (Zhou, 2012). Multiple bootstrap samples are generated to form a set of training samples, $\mathscr{L}^1, \mathscr{L}^2, ..., \mathscr{L}^k$. For each training set, $\mathscr{L}^i$, a predictor, $\hat{f}_i$, is trained on the data. During testing each predictor outputs a prediction $\hat{y}_i$ for each observation



(a) Training Process. **(b)** Testing Process.

Figure 2.22: The Bagging Procedure. (adpted from Mert et al. (2014))

$i = 1, ..., n$. In the context of classification the base learners predictions, $\hat{y}_i$, are combined by majority voting to form $\hat{y}_{agg}$. This is the class label that is most frequently generated by the base learners and is the final prediction of the ensemble. The ensembles prediction is given by,

$$\hat{y}_{agg} = \hat{F}(x) = \text{argmax}_{\omega_j} \sum_{i=1}^{k} I\left(\hat{f}_i(\mathbf{x}) = \omega_j\right),$$

where ω_j is a class label and $I(\cdot)$ is an indicator function that returns one when the expression in the bracket is true and zero everywhere else. The Bagging process is detailed in Algorithm 2.2.

Algorithm 2.2 Bagging.

Input training sample $\mathscr{L}$ and base learner $\hat{f}_i$.
For $i = 1....k$:
1. Randomly sample with replacement from $\mathscr{L}$ to generate bootstrap sample $\mathscr{L}^i$.
2. Train base learner $\hat{f}_i$ on bootstrap sample $\mathscr{L}^i$.
End For
Output ensemble: $\hat{F}(x) = \text{argmax}_{\omega_j} \sum_{i=1}^{k} I\left(\hat{f}_i(\mathbf{x}) = \omega_j\right)$

Bagging may significantly increase the performance of unstable predictive models in both regression and classification problems (Buhlmann et al., 2002; Skurichina and Duin, 1998). A model is unstable if small changes in the training set result in significant changes in the model's predictions (Breiman, 1996b). An example of an unstable models are decision trees (Dwyer and Holte, 2007). The recursive splitting algorithms used in constructing decision trees make them unstable predictors (Kotsiantis, 2013).

For regression problems, decomposing the prediction error into bias and variance terms it can be shown that Bagging an unstable predictor results in ensembles with a smaller variance than any of the base

learners in the ensemble (Breiman, 1996b). This result does not hold for classification problems as there is no simple additive decomposition of missclassification rate to bias and variance (Rao and Potts, 1997). Breiman (1996b) however, proved that Bagging classifiers may produce an almost optimal classifier.

Let $P(\omega_j|\mathbf{x})$ denote the probability of ω_j being the class label of observation $\mathbf{x}$. For a classifier denoted by $\hat{f}(\mathbf{x}, \mathscr{L}^i)$, the relative frequency that $\hat{f}(\mathbf{x}, \mathscr{L}^i)$ classifies $\mathbf{x}$ into class ω_j is denoted by, $P(\hat{f}(\mathbf{x}) = \omega_j)$. The probability that classifier $\hat{f}(\mathbf{x}, \mathscr{L}^i)$ classifies observation $\mathbf{x}$ correctly is

$$\sum_{\omega_j} P(\hat{f}(\mathbf{x}) = \omega_j) P(\omega_j|\mathbf{x}).$$

The total probability of correct classification is,

$$\int \sum_{\omega_j} P(\hat{f}(\mathbf{x}) = \omega_j) P(\omega_j|\mathbf{x}) P(\mathbf{x}) d\mathbf{x},$$

where $P(\mathbf{x})$ is the distribution of $\mathbf{x}$. Since $\mathscr{L}^i$ is a random vector (Breiman, 2001), $\mathbf{x}$ is also random vector as $\mathbf{x}$ is an element of $\mathscr{L}^i$ thus integration is done over $\mathbf{x}$. The maximum probability of correct classification any classifier can reach, also known as the Bayes error, is given by

$$\int \max_{\omega_j} P(\omega_j|\mathbf{x}) P(\mathbf{x}) d\mathbf{x}.$$

Note that any classifier's performance is bounded by Bayes error (Breiman, 1996b), that is

$$\sum_{\omega_j} P(\hat{f}(\mathbf{x}) = \omega_j) P(\omega_j|\mathbf{x}) \leq \max_{\omega_j} P(\omega_j|\mathbf{x}). \tag{2.14}$$

A classifier can achieve optimal performance only if the two terms in Equation 2.14 are equal, which occurs when

$$P(\hat{f}(\mathbf{x}) = \omega_j) = \begin{cases} 1 & \text{if } P(\omega_j|\mathbf{x}) = \max_{\omega_i} P(\omega_i|\mathbf{x}) \\ 0 & \text{else} \end{cases} \tag{2.15}$$

If an instance $\mathbf{x}$ results in class label ω_j more frequently than any other class and classifier $\hat{f}(\mathbf{x}, \mathscr{L}^i)$ predicts that $\mathbf{x}$ belongs to ω_j more frequently than any other class, the classifier is called order correct at $\mathbf{x}$, that is

$$\text{argmax}_{\omega_j} P(\hat{f}(\mathbf{x}) = \omega_j) = \text{argmax}_{\omega_j} P(\omega_j|\mathbf{x}).$$

Breiman (1996b) demonstrated that an order correct classifier is not necessarily a good classifier. An ensemble prediction denoted by $\hat{F}(\mathbf{x})$ is given by $\hat{F}(\mathbf{x}) = \text{argmax}_{\omega_j} P(\hat{f}(\mathbf{x}) = \omega_j)$. The probability of

correct classification for an ensemble is given by

$$\sum_{\omega_j} I\left(\text{argmax}_{\omega_i} P(\hat{f}(\mathbf{x}) = \omega_i) = \omega_j\right) P(\omega_j|\mathbf{x}). \tag{2.16}$$

The overall probability of correct classification for an ensemble is

$$\int \left[\sum_{\omega_j} I\left(\text{argmax}_{\omega_i} P(\hat{f}(\mathbf{x}) = \omega_i) = \omega_j\right) P(\omega_j|\mathbf{x})\right] P(\mathbf{x}) d\mathbf{x}. \tag{2.17}$$

If a predictor $\hat{f}$ is order correct for all observations of $\mathbf{x}$ that belong to C that is where $\mathbf{x} \in C$, then equation 2.17 becomes

$$\int_{\mathbf{x}\in C} \left[\sum_{\omega_j} I\left(\text{argmax}_{\omega_j} P(\omega_j|\mathbf{x})\right) P(\omega_j|\mathbf{x})\right] P(\mathbf{x}) d\mathbf{x} + \int_{\mathbf{x}\in C'} \left[\sum_{\omega_j} I\left(\hat{F}(\mathbf{x}) = \omega_j\right) P(\omega_j|\mathbf{x})\right] P(\mathbf{x}) d\mathbf{x} \tag{2.18}$$

where $C \cap C' = \emptyset$. The first term in this expression is the same as equation 2.15, which is the Bayes error, as the indicator function returns one when $P(\omega_j|\mathbf{x}) = \text{argmax}_{\omega_j} P(\omega_j|\mathbf{x})$ and zero elsewhere. Equation 2.18 simplifies to,

$$\int_{\mathbf{x}\in \mathbf{C}} \max_{\omega_j} P(\omega_j|\mathbf{x}) P(\mathbf{x}) d\mathbf{x}. + \int_{\mathbf{x}\in C'} \left[\sum_{\omega_j} I\left(\hat{F}(\mathbf{x}) = \omega_j\right) P(\omega_j|\mathbf{x})\right] P(\mathbf{x}) d\mathbf{x}. \tag{2.19}$$

Bagging unstable order correct classifiers may result in an almost optimal classifier (Zhou, 2012; Breiman, 1996b). If an ensemble is order correct for all $\mathbf{x}$ then it is an optimal classifier. Breiman (1996b) noted that Bagging poor predictors may generate ensembles with poor predictive accuracy.

Bootstrapping randomly samples with replacement from the original training set thus some instances may not be appear in a bootstrap sample. Consider a bootstrap sample $\mathscr{L}^i \in \mathscr{R}^{n\times p}$ generated from a training sample $\mathscr{L} \in \mathscr{R}^{n\times p}$. The probability that an instance from $\mathscr{L}$ does not appear in a bootstrap sample is $\left(\frac{n-1}{n}\right)^n$. Taking the limit as $n \to \infty$ results in

$$\lim_{n\to\infty} \left(\frac{n-1}{n}\right)^n = e^{-1}.$$

Thus, the probability that a instance does not appear in a bootstrap sample is $e^{-1} \approx 0.3679$. Approximately 36.79% of the original training set is unused in the training of each base learner (Breiman, 1996c). These unused observations are called out-of-bag (OOB) examples and can be used as an independent test set to estimate the ensembles predictive accuracy (Zhou, 2012) and generalization abilities (Breiman, 1996c). Let $\hat{F}^{00B}$ denote the OOB predictions. The ensembles OOB predictions is

then found by,

$$\hat{F}^{00B}(\mathbf{x}) = \text{argmax}_{\omega_j} \sum_{i=1}^{k} I\left(\hat{f}^k(\mathbf{x}) = \omega_j\right) \cdot I\left(\mathbf{x} \not\in \mathscr{L}^k\right).$$

Then an ensembles OOB generalization error is,

$$\text{OOB err} = \frac{1}{|\mathscr{L}|} \sum_{(\mathbf{x},\omega_j)\in\mathscr{L}} I(\hat{F}^{00B}(\mathbf{x}) \neq \omega_j).$$

2.4.2 Random Forests

Breiman (2001) introduced random forests as an ensemble method. Random forests are constructed using the Bagging procedure where the base learners are decision trees grown using random feature selection and the CART splitting algorithm without any pruning (Biau, 2012). During the training of the decision trees, random forests randomly select $m \leq p$ of the features at each split and subsequently, from the m features the best split is selected (Hastie et al., 2009). This results in the base learners being more unstable which may produce a better ensemble. In classification problems m is usually set to $m \approx \sqrt{p}$, the optimal value of m differs for each problem (Hastie et al., 2009) . Breiman (2001) noted that using random feature selection during the training of the trees produces better ensembles than bagged decision trees.

2.4.3 The Spiral Data

Consider the spiral data set depicted in Figure 2.23: Weinberger (2017) used a similar spiral data set to illustrate the effectiveness of Bagging. This simulated data is a balanced binary classification problem. Observations belonging to the same class have the same colour. Four models were trained on this data set, namely a classification tree, a random forest, a LDA classifier and an ensemble of bagged LDA classifiers. Each model was trained on the data. The performance of the models was visualized and compared (see Tables 2.16).

Figure 2.24 depicts the decision boundary of a classification tree (a) and a random forest (b) trained on the spiral data. The area in blue is the area where observations are classified as belonging to the blue class, similarly for the red class. From Figure 2.24a the classification tree struggles to capture the spiral pattern. However the random forest depicted in Figure 2.24b was able to learn the spiral shape much better than the classification tree.

Table 2.16 details the performance of these classifiers on the training data. The random forest was able to learn the training data with a perfect accuracy and AUC. The classification tree was not able to learn the data well with an accuracy 0.54.

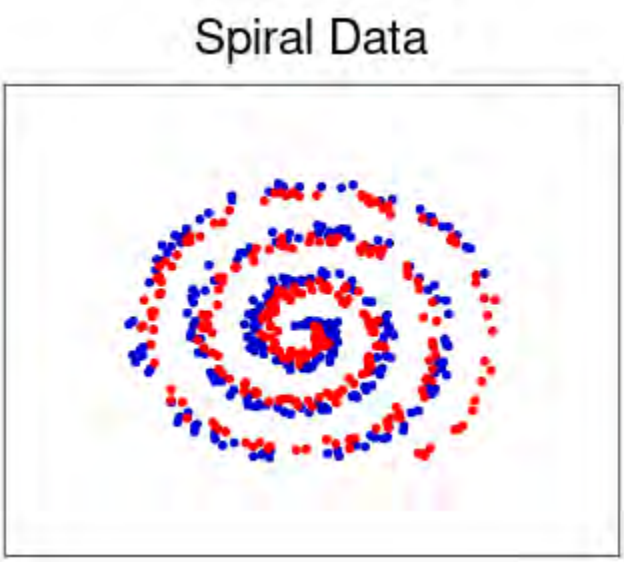


Figure 2.23: The Spiral Data set.

Table 2.16: The Classifier's Performance on the Spiral Data.

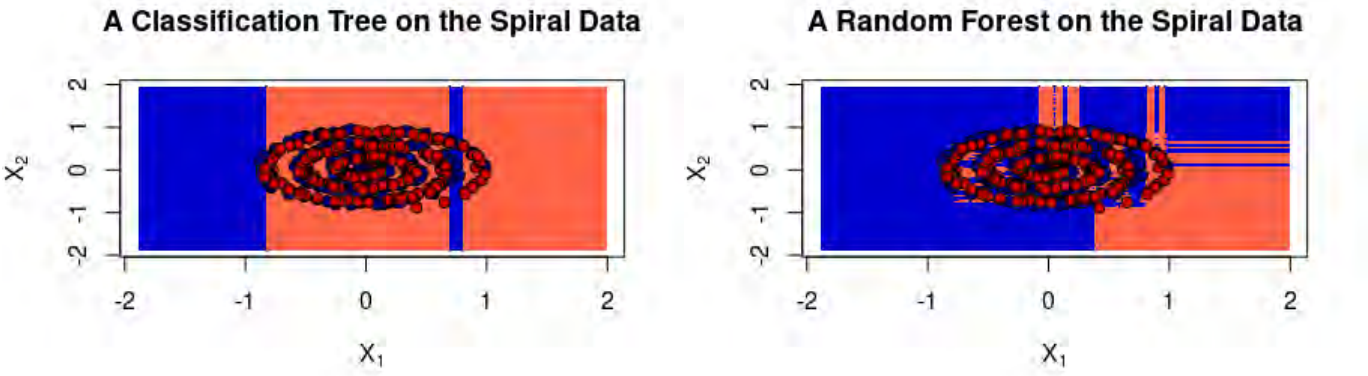
Performance Metric	Classification Tree	Random Forest	LDA	Bagged LDA
Accuracy	0.54	1	0.5	0.528
AUC	0.7188	1	0.5	0.528

Figure 2.25 depicts the decision boundary of a LDA classifier and a set of bagged LDA classifiers. LDA is a stable classifier (Breiman, 1996a) as small changes in the data do not affect the classifier predictions. The two decision boundaries are almost identical and both models were not able to learn the pattern. From Table 2.16 the performance of the two models is also almost identical and poor. This demonstrates that Bagging stable classifiers is not a good idea, as the classifiers predictions do not change by making small alterations to the data set.

2.5 Boosting Classifiers

Boosting is a general homogeneous ensemble method used to convert a group of weak learners into an ensemble with strong predictive power (Maclin and Opitz, 1997). Boosting is the process of sequentially reweighing the training data, so that the instances missclassified by the previous weak learner are given a higher weight. The correctly classified instances are given a lesser weight. The idea is that the greater the weight an instance has the greater focus the next weak learner gives to that instance. (Lemmens and Croux, 2006)

The general process used in boosting algorithms, such as adaptive boosting (Schapire, 2013) is illustrated in Figure 2.26. First each instance in the original learning sample $\mathscr{L}$ is assigned the same weight



(a) Classification Tree.

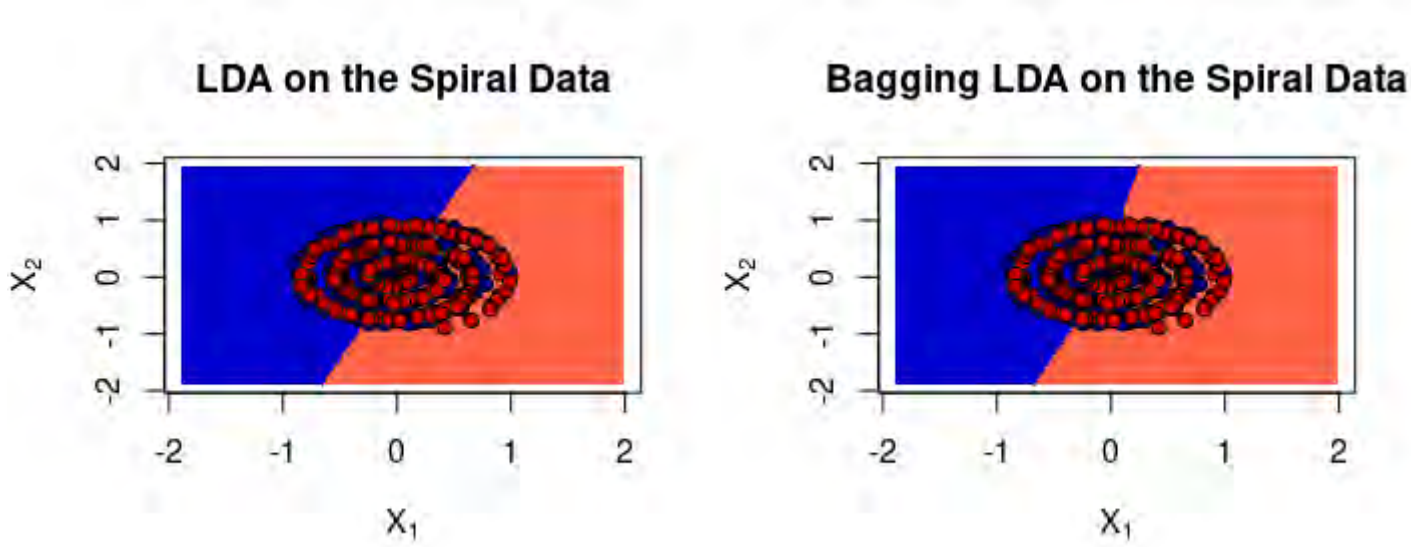
(b) Random Forest.

Figure 2.24: A Classification Tree and a Random Forest Trained on the Spiral Data.

resulting in $\mathscr{L}^1$. The weak learner, denoted by $\hat{f}_1(\mathbf{x})$, is trained on the weighted learning sample $\mathscr{L}^1$. The instances are then reweighed resulting in $\mathscr{L}^2$; instances that the weak learner, $\hat{f}_1(\mathbf{x})$ classifies correctly are assigned a smaller weight and instances the weak learner $\hat{f}_1(\mathbf{x})$ missclassifies are assigned a greater weight. The reweighing process produces a new weighted learning sample denoted by $\mathscr{L}^2$. The weak learner $\hat{f}_1(\mathbf{x})$ then gets assigned a weighted vote denoted by $\hat{y}_1$. The weight is dependent on how well the learner performed, the better the performance the greater the weight. The next weak learner $\hat{f}_2(\mathbf{x})$ is then trained on $\mathscr{L}^2$. These steps are repeated until all k weak learners have been trained. The weighted votes are then aggregated to form an ensemble.

2.5.1 Adaptive Boosting

Adaptive Boosting (Adaboost) is a boosting method proposed by Freund et al. (1996). Consider a binary classification problem with labeled data pairs $(\mathbf{x}_j, y_j)$ and where the classes are given by $y_j \in Y = \{-1, +1\}$. The Adaboost algorithm initializes equal weights for each instance. $D_i(j)$ is used to weight the instances, i denotes the algorithms iteration and j corresponds to an observation in the data. In the first iteration for observation $\mathbf{x}_j$ the weight is $D_1(j) = \frac{1}{N}$. The weighted training sample is found by multiplying the training sample, $\mathscr{L}^i$ by D_i. The first step is to train the weak learners using D_i to weight the training data. The training error of the weak learner is then found and is given by $\varepsilon_i = \sum_{i:f_i(\mathbf{x}_j) \neq y_j} D_i(j)$. Then set $\beta_i = \frac{1}{2} ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$. This term is used to weight the weak learner's vote. As the error term increases the weight β_i decreases thus less accurate learners are given a lesser vote. The



(a) LDA. **(b)** A set of Bagged LDA Classifiers.

Figure 2.25: LDA Classifier and a Set of Bagged LDA Classifiers Trained on the Spiral Data.

weight D_i is then updated by,

$$D_{i+1}(j) = \frac{D_i(j)}{Z_i} \times \begin{cases} e^{-\beta_i} & f_i(\mathbf{x}_j) = y_j \\ e^{\beta_i} & f_i(\mathbf{x}_j) \neq y_j, \end{cases}$$

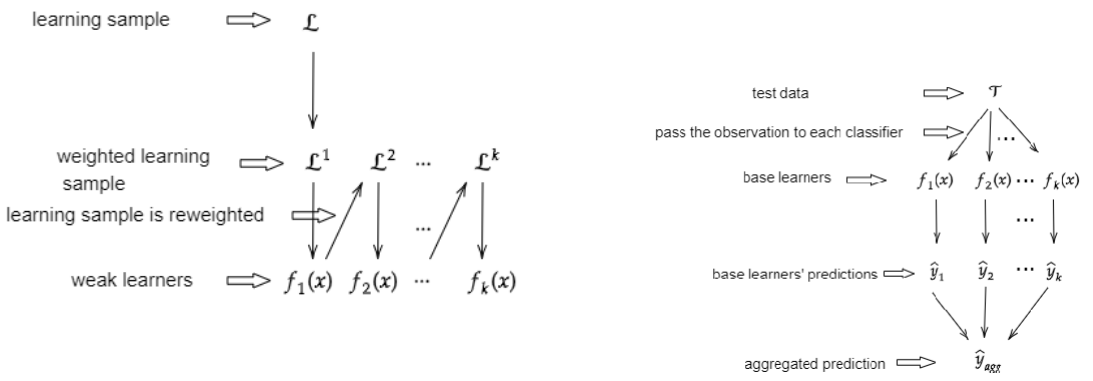
Z_i is a normalizing constant, chosen such that D_{i+1} is a distribution, that is $\sum_{j=1}^{N} D_{i+1} = 1$. Where these steps are repeated until k weak learners have been sequentially trained. The resulting ensemble is given by

$$F(\mathbf{x}) = sign\left(\sum_{i=1}^{k} \beta_i f_i(\mathbf{x})\right).$$

The $sign(\cdot)$ function returns "-1" if the term in the bracket is negative and "$+1$" if the term in the bracket is positive, both of these correspond to one of the two classes. The Adaboost method is detailed in Algorithm 2.3.

2.5.2 Gradient Boosting

Gradient boosting is an additive boosting method that generates an ensemble by adding to an existing ensemble the base learner that minimizes the expected value of some specified loss function (Bentéjac et al., 2021). A loss function measures a models prediction error that some optimization process attempts to minimize (Nie et al., 2018). For example the Gini index used in classification trees is a



(a) Training Process. **(b)** Testing Process.

Figure 2.26: The Boosting Procedure.

Algorithm 2.3 Adaboost Algorithm.

1. Initialize equal weights by setting $D_1(i) = \frac{1}{N}$.

Repeat for each weak learner:

2. Train the weak learner, f_i using D_i to weight the training data.
3. Find the learners training error: $\varepsilon_k = \sum_{i:f_i(\mathbf{x}_j)\neq y_j} D_i(j)$, if $\varepsilon_i > \frac{1}{2}$ then the loop is aborted.
4. Set $\beta_i = \frac{1}{2} ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$.
5. The weight D_i is then updated by, $D_{i+1}(j) = \frac{D_i(j)}{Z_i} \times \begin{cases} e^{-\beta_i} & f_i(\mathbf{x}_j) = y_j \\ e^{\beta_i} & f_i(\mathbf{x}_j) \neq y_j, \end{cases}$.

Once all weak learners have been trained the final ensemble is found by, $F(\mathbf{x}) = sign\left(\sum_{i=1}^{k} \beta_i f_i(\mathbf{x})\right)$.

loss function (Tibrewal, 2018). The goal of the splitting criterion is to grow the tree such that the Gini index is minimized. If $f(\mathbf{x})$ is a base learner and $L[y, f(\mathbf{x})]$ is a loss function then $f(\mathbf{x})$ is added to the ensemble such that

$$f(\mathbf{x}) = \arg\min_{f(\mathbf{x})} E_{\mathbf{x}}\left[E_y\left(L[y, F(\mathbf{x})]\right)|\mathbf{x}\right], \tag{2.20}$$

where $f(\mathbf{x}) = f(\mathbf{x}, \mathbf{a}_m)$ and $\mathbf{a}_m = \{a_1, a_2, ...\}$ are the parameters of the base learners. The parameters $\mathbf{a}_m$ govern the structure of each base learner. If the base learners are classification trees, $\mathbf{a}_m$ is the splitting variables, the terminal node class labels and split locations of each tree (Friedman, 2001).

Consider a classifier $f(\mathbf{x}, \mathbf{a}_m)$. Traditional statistical learning methods find the optimal classifier by optimizing the hyperparameters $\mathbf{a}_m$ (Natekin and Knoll, 2013). With boosting methods the ensemble is given by $F(\mathbf{x}) = sign\left(\sum_{k=1}^{k} \beta_k f_k(\mathbf{x})\right)$. The function $F(\mathbf{x})$ is parameterized in its additive form. Thus the optimization occurs in the function space (Natekin and Knoll, 2013), where the loss function is minimized as in equation 2.20.

Gradient boosting methods build on the steepest descent method. Steepest descent is an optimization method where a differentiable function $f(\mathbf{x})$ is minimized. Let $d_k = -\frac{\partial(x)}{\partial x_k}$, the idea behind steepest descent is to take a step in the direction d_k, until the change in gradient is less than some predefined threshold (Meza, 2010). Using steepest descent the minimum of $f(x)$ can be found by sequentially finding $x_{k+1} = x_k + \alpha_k d_k$, where α_k is the step size given by $\alpha_k = \arg\min_\alpha f(x_k + \alpha d_k)$. The search is done in the opposite of the gradient direction thus it reduces the function $f(x)$ as much as possible with each step. It can be shown that the steepest descent method will only stop searching once a stationary point is found (Yuan, 2006). The gradient descent algorithm is detailed in Algorithm 2.4 (Meza, 2010).

Algorithm 2.4 Steepest Descent

1. For an initial x_0, $d_0 = -\frac{\partial f(x_0)}{\partial x}$ and threshold t.
2. For $k = 0$ to K do:
3. Set $\alpha_k = \arg\min_\alpha f(x_k + \alpha d_k)$.
4. Set $x_{k+1} = x_k + \alpha_k d_k$.
5. Set $d_{k+1} = -\frac{\partial f(x_{k+1})}{\partial x}$.
6. if $||d_{k+1}|| \leq t$ abort loop.
7. end

Gradient boosted trees expect each base learner to minimize (Bentéjac et al., 2021), that is

$$(\beta_k, \mathbf{a}_k) = \arg\min_{\beta,a} \sum_{i=1}^{N} L[y_i, F_{k-1}(\mathbf{x}_i)] + \beta f(\mathbf{x}_i, \mathbf{a}_k). \tag{2.21}$$

If Equation 2.21 is hard to evaluate for a loss function or base learner, then Equation 2.21 can be rewritten as the best greedy step toward Equation 2.20. Let $g_k(\mathbf{x}_i)$ denote the negative gradient of the loss function, that is

$$g_k(\mathbf{x}_i) = \frac{\partial L[y_i, F_{k-1}(\mathbf{x}_i)]}{\partial F_{k-1}(\mathbf{x}_i)}$$

such that $-\mathbf{g}_k = \{g_k(\mathbf{x}_i)\}_1^N$. The idea behind gradient boosted trees is to find a base learner $f(x_i, \mathbf{a}_k)$ that is most highly correlated to $-\mathbf{g}_k$. This base learner can then be used as a step in the direction of the negative gradient of the loss function, such that the loss function is minimized similarly to the gradient descent algorithm. This base learner is found by

$$\mathbf{a}_k = \arg\min_{\beta,a} \sum_{i}^{N} [g_k(\mathbf{x}_i) - \beta f(\mathbf{x}_i, \mathbf{a})].$$

This base learner is then used as the new gradient. The step-size is found by,

$$\beta_k = \arg\min_\beta \sum_{i=1}^{N} L\left(y_i, F_{k-1}(\mathbf{x}_i) + \beta f(x_i, \mathbf{a}_k)\right]).$$

The ensemble then becomes

$$F_k(\mathbf{x}) = F_{k-1}(\mathbf{x}) + \beta_k f(x_i, \mathbf{a}_k)$$

The gradient boosted trees algorithm is detailed in algorithm 2.5.(Friedman, 2001)

Algorithm 2.5 Gradient Boost.

1. Initialize $F_0(\mathbf{x}) = argmin_\beta \sum_{i=1}^{N} L(y_i, \beta)$
2. For $k = 1$ to K do:
3. $\hat{y}_i = \frac{dL[y_i, F_{k-1}(\mathbf{x}_i)]}{dF_{k-1}(\mathbf{x}_i)}$.
4. $\mathbf{a}_k = \arg\min_{\beta,a} \sum_i^N [g_k(\mathbf{x}_i) - \beta f(\mathbf{x}_i, \mathbf{a})]$.
5. $\beta_k = \arg\min_\beta \sum_{i=1}^{N} L\left(y_i, F_{k-1}(\mathbf{x}_i) + \beta f(x_i, \mathbf{a}_k)\right)$.
6. $F_k(\mathbf{x}) = F_{k-1}(\mathbf{x}) + \beta_k f(x_i, \mathbf{a}_k)$.

2.5.3 Extreme Gradient Boosting

Chen and Guestrin (2016) introduced extreme gradient boosting (XGBoost) which is a popular scalable tree boosting system. XGBoost gained popularity due to its dominance in Kaggle competitions, where most of the twenty-nine winning participants used XGBoost (Chen and Guestrin, 2016). Chen and Guestrin (2016) noted that the most important factor in the method's success is its scalability to large data sets. The scalability is due to a novel tree learning algorithm and additional computational enhancements for example, parallel and distributed computing (Chen and Guestrin, 2016).

XGBoost uses decision trees as base learners. A decision tree denoted by f_k can be represented by its leaf scores, denoted by w_j,

$$f_k(\mathbf{x}) = w_{q(\mathbf{x})} \tag{2.22}$$

where $q(\mathbf{x})$ returns the path and structure of a tree that maps $\mathbf{x}$ to a terminal node with weight w_j (Chen and He, 2015). The general learning criterion used in gradient boosting methods is defined as,

$$L^k(\theta) = \sum_{i=1}^{N} L\left(y_i, F_{k-1}(\mathbf{x}_i) + f_k(\mathbf{x}_i)\right) + \Omega(f_k) \tag{2.23}$$

where $L\left(y_i, F_{k-1}(\mathbf{x}_i) + f_k(\mathbf{x}_i)\right)$ denotes a loss function and $\Omega(f_k)$ denotes the regularization parameter. The regularization parameter measures the complexity of a model (Chen and He, 2015). Since the goal is to minimize $L^k(\theta)$, regularization attempts to prevent a base learner from overfitting the training data by penalizing base learners for their complexity (Li et al., 2019). The goal is to add the base learner f_k that will minimize the learning criterion $L^k(\theta)$.

Recall Taylor's theorem (Šikić, 1990), some differentiable function $f(x)$ can be approximated by a

second order Taylor series given by,

$$f(a_1) \approx f(x) + (a_1 - x)f^{'}(x) + \frac{(a_1 - x)^2}{2} f^{''}(x) \tag{2.24}$$

where $f^{'}(x) = \frac{\partial f(x)}{\partial x}$ and $f^{''}(x) = \frac{\partial^2 f(x)}{\partial x^2}$. Substituting $a_1 = x + a$ into the Taylor series in Equation 2.24 yields,

$$f(x+a) \approx f(x) + af^{'}(x) + \frac{(a)^2}{2} f^{''}(x).$$

Applying a second order Taylor series expansion to approximate $\sum_{i=1}^{N} L(y_i, F_{k-1}(\mathbf{x}_i) + f_k(\mathbf{x}_i))$, Equation 2.23 becomes

$$L^k(\theta) \approx \sum_{i=1}^{N} \left[L(y_i, F_{k-1}(\mathbf{x}_i)) + g_i f_k(\mathbf{x}_i) + \frac{1}{2} h_i f_k^2(\mathbf{x}_i) \right] + \Omega(f_k), \tag{2.25}$$

where the first and second order derivatives are denoted by, $g_i = \frac{\partial L(y_i, F_{k-1}(\mathbf{x}_i) + f_k(\mathbf{x}_i))}{\partial L(y_i, F_{k-1}(\mathbf{x}_i))}$ and $h_i = \frac{\partial^2 L(y_i, F_{k-1}(\mathbf{x}_i) + f_k(\mathbf{x}_i))}{\partial L^2(y_i, F_{k-1}(\mathbf{x}_i))}$ (Friedman et al., 2000). The goal is to minimize $L^k(\theta)$ thus constant terms are removed and Equation 2.25 is reduced to

$$\tilde{L}^k(\theta) = \sum_{i=1}^{N} \left[g_i f_k(\mathbf{x}_i) + \frac{1}{2} h_i f_k^2(\mathbf{x}_i) \right] + \Omega(f_k). \tag{2.26}$$

Since the second order Taylor series expansion of $L^k(\theta)$ is an approximation, thus $L^k(\theta) \approx \tilde{L}^k(\theta)$.

$$L^k(\theta) = \sum_{i=1}^{N} \overbrace{L\left(\underbrace{y_i, F_{k-1}(\mathbf{x}_i)}_{\text{Constant}} + \overbrace{f_k(\mathbf{x}_i)}^{f_k \text{ is unknown}} \right)}^{L(y_i, F_{k-1}(\mathbf{x}_i) + f_k(\mathbf{x}_i)) \text{ is unknown}} + \overbrace{\Omega(f_k)}^{\Omega \text{ is unknown}}, \tag{2.27}$$

$$\tilde{L}^k(\theta) = \sum_{i=1}^{N} \left[\underbrace{g_i}_{\text{Constant}} \overbrace{f_k(\mathbf{x}_i)}^{f_k \text{ is unknown}} + \underbrace{\frac{1}{2} h_i}_{\text{Constant}} \overbrace{f_k^2(\mathbf{x}_i)}^{f_k \text{ is unknown}} \right] + \overbrace{\Omega(f_k)}^{\Omega \text{ is unknown}}. \tag{2.28}$$

Comparing the original learning criterion in Equation 2.27 with the approximation in Equation 2.28. Equation 2.27 has three unknown variables that need to be calculated while Equation 2.28 only has two unknown variables. Thus for each iteration i, Equation 2.28 has one less calculation to compute than the original learning criterion. This reduces the time needed to compute the learning criterion making XGBoost more scalable than traditional gradient boosting methods.

Chen and Guestrin (2016) set the regularization parameter in Equation 2.26 as

$$\Omega(f_k) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

where T is the number of nodes, λ is used to weight the leaf nodes and γ is used to weight the number of nodes. For a fixed tree structure the set of instances in node j are defined as $I_j = \{i|q(\mathbf{x}) = j\}$. Substituting Equation 2.22 into Equation 2.26 yields

$$\begin{aligned}
\overset{\sim k}{L}(\theta) &= \sum_{j=1}^{T}\left[(\sum_{i\in I_j} g_i)w_j + \frac{1}{2}\sum_{i\in I_j} h_i w_j^2\right] + \gamma T + \frac{1}{2}\lambda\sum_{j=1}^{T} w_j^2 \\
&= \sum_{j=1}^{T}\left[(\sum_{i\in I_j} g_i)w_j + \frac{1}{2}(\sum_{i\in I_j} h_i + \lambda)w_j^2\right] + \gamma T \\
&= \sum_{j=1}^{T}\left[\frac{1}{2}(\sum_{i\in I_j} h_i + \lambda)w_j^2 + (\sum_{i\in I_j} g_i)w_j\right] + \gamma T. \qquad (2.29)
\end{aligned}$$

Recall for a quadratic function $ax^2 + bx + c$, the minimum of the function is found at $x = -\frac{b}{2a}$ for $a > 0$. $\overset{\sim k}{L}(\theta)$ in Equation 2.29 is a quadratic equation with $a = \frac{1}{2}(\sum_{i\in I_j} h_i + \lambda)$ and $b = \sum_{i\in I_j} g_i$. Thus the optimal weight of node j (minimum value of $\overset{\sim k}{L}(\theta)$) is found when,

$$w_j^* = -\frac{\sum_{i\in I_j} g_i}{(\sum_{i\in I_j} h_i + \lambda)}.$$

Substituting w_j^* back into Equation 2.29 yields,

$$\begin{aligned}
\overset{\sim k}{L}(\theta) &= \sum_{j=1}^{T}\left[\frac{1}{2}(\sum_{i\in I_j} h_i + \lambda)\left(-\frac{\sum_{i\in I_j} g_i}{(\sum_{i\in I_j} h_i + \lambda)}\right)^2 + (\sum_{i\in I_j} g_i)\left(-\frac{\sum_{i\in I_j} g_i}{(\sum_{i\in I_j} h_i + \lambda)}\right)\right] + \gamma T. \\
&= \sum_{j=1}^{T}\left[\frac{1}{2}\frac{\left(\sum_{i\in I_j} g_i\right)^2}{(\sum_{i\in I_j} h_i + \lambda)} - \left(\frac{(\sum_{i\in I_j} g_i)^2}{(\sum_{i\in I_j} h_i + \lambda)}\right)\right] + \gamma T \\
&= -\frac{1}{2}\sum_{j=1}^{T}\frac{\left(\sum_{i\in I_j} g_i\right)^2}{(\sum_{i\in I_j} h_i + \lambda)} + \gamma T. \qquad (2.30)
\end{aligned}$$

The structure of a tree is not given in advance. Searching all infinitely possible tree structures to minimize the learning criterion is impossible (Chen and Guestrin, 2016). However Equation 2.30 can be used as a greedy splitting criterion similar to the impurity score introduced by Breiman et al. (1984). Let $I = I_L \cup I_R$ where I_L and I_R denote the observations in the proposed left and right child

nodes respectively (Chen and He, 2015). The quality of a split is calculated by

$$\text{gain} = \frac{1}{2}\left[\frac{\left(\sum_{i\in I_R} g_i\right)^2}{\left(\sum_{i\in I_R} h_i+\lambda\right)} + \frac{\left(\sum_{i\in I_L} g_i\right)^2}{\left(\sum_{i\in I_L} h_i+\lambda\right)} - \frac{\left(\sum_{i\in I} g_i\right)^2}{\left(\sum_{i\in I} h_i+\lambda\right)}\right] - \gamma$$

where $\frac{\left(\sum_{i\in I_R} g_i\right)^2}{\left(\sum_{i\in I_R} h_i+\lambda\right)} + \frac{\left(\sum_{i\in I_L} g_i\right)^2}{\left(\sum_{i\in I_L} h_i+\lambda\right)}$ measures the quality of a proposed split, $\frac{(\sum_{i\in I} g_i)^2}{(\sum_{i\in I} h_i+\lambda)}$ measures the quality of the current tree and γ penalizes each split for its additional complexity. At each node the candidate split that maximizes gain is selected until the tree is grown to its maximum depth. (Chen and He, 2015; Chen and Guestrin, 2016). All leaf nodes with negative gain are then pruned from the tree resulting in the final base learner (Chen and He, 2015). The tree growing algorithm is detailed in Algorithm 2.6.

Algorithm 2.6 XGBoost Exact Greedy Algorithm (Chen and Guestrin, 2016).

1. $I =$ set of instances in current node.
2. gain = 0.
3. $G = \sum_{i\in I} g_i$, $H = \sum_{i\in I} h_i$.
4. for $k = 1$, to M do:
5. $G_L = 0$, $H_L = 0$.
6. for j in I do:
7. $G_L = G_L + g_j$, $H_L = H_L + h_j$,
8. $G_R = G - G_L$, $H_L = H - H_L$,
9. gain = max(gain, $\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} = \frac{G^2}{H+\lambda}$).

end

end

10. Output: Split with max gain

Ensemble of Boosted Trees with Synthetic Features

Zięba et al. (2016) proposed a novel technique for generating new features from an existing set of features to enhance bankruptcy models. In the bankruptcy context, models are trained only on economic variables thus adding new variables to the data may improve a models predictive power. These new features are called synthetic features which are a combination of the economic variables in the data set.

Zięba et al. (2016) developed synthetic features by using extreme gradient boosting trees (XGBoost) in the following manner. First an XGBoost model, denoted by f_k is trained on the training data denoted by $\mathscr{L}_k$. Two features are sampled from the training data and a arithmetic operation is then randomly selected and performed on the selected features producing a synthetic feature. These synthetic features are then added to the data set generating a new training set.

The features are sampled from a categorical distribution which is given by,

$$\theta_k = [\theta_k^1, \theta_k^2, ..., \theta_k^p]$$

where variable $\mathbf{X}_i$ has probability θ_k^i of being selected. Each features probability of being selected is determined by the features importance, which can be estimated from a trained XGBoost model (Hastie et al., 2009). Zięba et al. (2016) used a feature's relative frequency in the XGBoost ensemble to estimate the probability of it being selected. The relative frequency is the number of times a variable is selected to split the nodes in the ensemble, divided by the total number splits in the ensemble. Let m_i denote the frequency a feature is selected to split nodes in base learner f_k and $\sum_{i=1}^{p} m_i$ be the total frequency of all the features in base learner f_k. The relative frequency of feature $\mathbf{X}_i$ in base learner f_k is given by

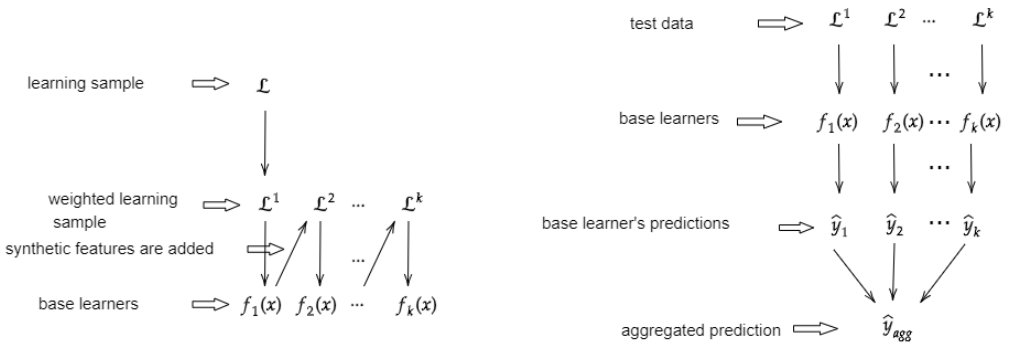
$$\theta_k^i = \frac{m_i}{\sum_{i=1}^{p} m_i}.$$

The relative frequency is found for all p features and is used to construct the categorical distribution θ_k. Any feature $\mathbf{X}_i$ with a frequency m_i below a predefined threshold λ is removed from the data set $\mathscr{L}_k$. Synthetic features are then added to this new data set resulting in a new training set $\mathscr{L}_{k+1}$ which classifier f_{k+1} is trained on. Algorithm 2.7 details the synthetic boosting method (SynBoost) proposed by Zięba et al. (2016).

Algorithm 2.7 Ensemble of Boosted Trees with Synthetic Features.

Input: $\mathscr{L}_k$: training data, λ: feature acceptance threshold, f_k: XGBoost base learner, $\mathscr{D}_{new}$: the number of synthetic features, m_i: feature frequency.
Start
1. for $k = 1, 2...K$ do:
2. Train f_k using $\mathscr{L}_k$.
3. Remove variables from $\mathscr{L}_k$ where $m_i < \lambda$.
4. Estimate θ_k using f_k.
5. for $d = 1, 2...\mathscr{D}_{new}$ do:
6. Generate features $\mathbf{X}_1$ and $\mathbf{X}_2$ by using θ_k to sample from $\mathscr{L}_k$.
7. Randomly sample a single operation $\star$, from the set $\star = \{+, -, \times, \div\}$.
8. Produce a new feature $\mathbf{X}_{new} = \mathbf{X}_1 \star \mathbf{X}_2$.
9. Add the new feature $\mathbf{X}_{new}$ to the data set $\mathscr{L}_k$,resulting in a new data set $\mathscr{L}_{k+1}$.
end For
end For
10. Output: $F = \{f_1, f_2, ..., f_K\}$

Each base learner in the SynBoost ensemble is trained on a different data set. Thus the structure of the final ensemble differs to traditional ensembles. Figure 2.27 depicts the structure of the SynBoost classifier. During the training of base learner $f_i(\mathbf{x})$ the estimated predictive power of the variables is the only information taken from the previous classifier. This is then used to create a new training data set $\mathscr{L}_{i+1}$ by adding synthetic features and removing weak predictors. Since each classifier is trained on different variables, during testing new testing sets matching the variables of the training sets have to be created for each classifier. Each of these testing sets are passed down to the relevant classifier, each producing a prediction which is then aggregated by majority voting.



(a) Training a SynBoost Ensemble. **(b)** Testing a SynBoost Ensemble.

Figure 2.27: SynBoost Structure.

Zięba et al. (2016) compared synthetic features to the general Darwinian evolutionary process. Features determined to be strong predictors have a greater probability of being selected thus a higher chance of reproducing. These features can be viewed as strong parents. Combining these strong parent features may produce a child synthetic feature which may have strong predictive power.

This study used the R programming language to implement all statistical models and techniques. Currently, there is no existing R library or package to implement the SynBoost algorithm as outlined by Zięba et al. (2016). Hence a script was written in R to implement the SynBoost method. This R code can be found in Appendix 5.2. The script consists of two sections, the first section is used to train the models and make alterations to the training data set. The second section of the code is where the Synboost model is tested, the necessary alterations are also made to the testing data such that each base learner can output a prediction. There are several assumptions made in the script such as the case where division by zero occurs, Zięba et al. (2016) did not explicitly outline a procedure to deal with this. In the script if division by zero occurs the script outputs a placeholder value of zero.

2.6 Assessment of Binary Classifiers

The evaluation of a classifier's performance is an important step after implementation as the quality of the classifier's predictive accuracy can only be determined by it's performance metrics. Performance metrics allow the comparison between different classifiers, meaning the best performing classifier can be selected for a problem (Hossin and Sulaiman, 2015). Using several metrics to evaluate the performance of a classifier is necessary as no single metric can show all the strengths and weaknesses of a classifier (Lever et al., 2016).

In this chapter we aim to introduce and discuss several relevant performance metrics which will be

used to measure the performance of the classifiers.

2.6.1 The Binary Confusion Matrix

In binary classification there are two classes being predicted, either the positive or negative class. The positive class contains all the observations that belong to the class of interest. In the context of this study, that would be the account holders who default on their payments. The negative class contains all the other observations that are not part of the positive class (Wang et al., 2012). These predictions are then summarized in a confusion matrix, (see Table 2.17) which is a two dimensional table summarizing the performance of a classifier by detailing the actual and predicted classes done by the classifier (Deng et al., 2016). The metrics in a confusion matrix on their own do not completely describe a classifiers performance but they can be used to define several performance metrics.

Table 2.17: Confusion Matrix: $k = 3$ kNN Classification.

	Actual Class		
Predicted Class	Class	Positive	Negative
	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

In Table 2.17

- True Positive (T_p) counts the number of positive observations classified correctly;
- True Negative (T_n) counts the number of negative observations classified correctly;
- False Negative (F_n) counts the number of positive observations incorrectly classifeid as being negative;
- False positive (F_p) counts the number of negative observations incorrectly classified as being positive.

2.6.2 The Text Classification Data Set

Consider the study conducted by Maria Navin and Pankaja (2016). The aim of the study was to analyze the performance of text classification algorithms in the detection of negative and positive movie reviews. The classification algorithms applied were, kNN, LR, Naïve Bayes (NB) and Support Vector Machines (SVM). For each classifier a confusion matrix was obtained. For the kNN classifier, the confusion matrix in Table 2.18 was obtained.

Table 2.18: Confusion Matrix: $k = 5$ kNN Classification (Maria Navin and Pankaja, 2016)

	Actual Class		
Predicted Class	Class	Positive	Negative
	Positive	79	20
	Negative	22	79

In the context of the study conducted by Maria Navin and Pankaja (2016)

- True Positive ($T_p = 79$) counts the number of positive movie reviews classified as being positive movie reviews. Thus 79 positive reviews were correctly classified by the kNN classifier as positive reviews.
- True Negative ($T_n = 79$) counts the number of negative movie reviews classified as being negative movie reviews. Thus 79 of the negative reviews were correctly classifed by the kNN classifier as negative reviews.
- False Negative ($F_n = 22$) counts the number of positive movie reviews incorrectly classifeid as being negative movie reviews. Thus 22 positive reviews were incorrectly classified by the kNN classifier as negative reviews
- False positive ($F_p = 20$) counts the number of negative movie reviews incorrectly classified as being positive movie reviews. Thus 20 negative reviews were incorrectly classified by the kNN classifier as positive reviews

2.6.3 Accuracy and Class Imbalance

Accuracy is a performance metric that measures the percentage of observations classified correctly and which is given by

$$Accuracy = \frac{T_p + T_n}{T_p + T_n + F_n + F_p} = \frac{T_p + T_n}{n} \tag{2.31}$$

where n is the total number of observations classified (Baldi et al., 2000). Accuracy is bounded between zero and one, the higher the accuracy the better a classifier performs. Accuracy is commonly used to measure the performance of a classifier, however it makes the assumption of equal class distribution (Sun et al., 2009). Accuracy produces misleading results for imbalanced data Thabtah et al. (2020). Consider a data set that contains 95 observations belonging to class A and 5 observations belonging to class B. The classifier would produce an accuracy of 0.95 by classifying all observations as belonging to class A. This classifier would produce a high accuracy but a poor performance, as it is unable to detect any observations belonging to class B. In the context of this research the data set is imbalanced. It is essential that all performance metrics used to evaluate classifiers are insensitive to imbalanced data. Thabtah et al. (2020)

From Table 2.18, the accuracy of the kNN classifier is 79% since,

$$Accuracy = \frac{T_p + T_n}{n} = \frac{79+79}{200} = 0.79.$$

2.6.4 Recall and Precision

Recall and precision are performance metrics that measure the retrieval of data. Both metrics are bounded between zero and one, the higher the value of precision or recall the better a classifier performs (Powers, 2011).

Recall (R) measures the proportion of positive observations that have been correctly predicted as being positive (Vakili et al., 2020):

$$Recall = \frac{T_p}{T_p + F_p}.$$

Precision (P) measures the proportion of positive predictions that have been correctly predicted as being positive (Vakili et al., 2020):

$$Precision = \frac{T_p}{T_p + F_n}.$$

From Table 2.18, the recall and precision of the kNN classifier are,

$$Recall = \frac{T_p}{T_p + F_p} = \frac{79}{79+22} = 0.78.$$

$$Precision = \frac{T_p}{T_p + F_p} = \frac{79}{79+20} = 0.8.$$

Both of these values are high as the metrics are bounded between one and zero, this indicates that the kNN classifier performed well on the data set.

2.6.5 F1-Score

The F1-score is a performance metric that measures a classifiers effectiveness and is used in both binary and multi-label classification (Lipton et al., 2014). The F1-score was first introduced by Van Rijsbergen (1986) and is defined as the harmonic mean of precision and recall (Sasaki, 2002). The F1-score is defined as,

$$F_1 = \frac{2PR}{(P+R)} = \frac{T_p}{T_p + \frac{1}{2}(F_p + F_n)}. \tag{2.32}$$

The F1-score is bounded between one and zero. A value of one means perfect precision and recall indicating the classifier classified all observations correctly. If either recall or precision are equal to zero the classifier will have an F1-score of zero. The higher the F1 score the better the classifier performed. The F1 score however fails to include T_n thus it does not provide a complete picture of the classifiers performance (Powers, 2011).

From Table 2.18, the F1-Score of the kNN classifier is

$$F_1 = \frac{2PR}{(P+R)} = \frac{2 \times 0.8 \times 0.78}{0.8+0.78} = 0.79$$

This is a high F1-score, this indicates that the kNN classifier performed well on the data set.

2.6.6 Matthews Correlation Coefficient

Matthews Correlation Coefficient (MCC) is a performance metric first introduced by Matthews (1975). MCC measures the correlation between actual and predicted values (Chicco and Jurman, 2020) and is given by

$$MCC = \frac{T_p T_n - F_p F_n}{\sqrt{(T_p+F_p)(T_p+F_n)(T_n+F_p)(T_n+F_n)}}. \tag{2.33}$$

MCC produces a high value if for each class, the classifier predicted most of the observations correctly. This makes MCC suitable for imbalanced data. MCC outputs a value in the range [-1,+1], a value of +1 indicates perfect classification, a value of -1 indicates perfect misclassification and a value of 0 means the classifier's performance was as good as tossing a fair coin (Chicco and Jurman, 2020).

From Table 2.18, the MCC of the kNN classifier is approximately 58% since

$$MCC = \frac{79 \times 79 - 20 \times 22}{\sqrt{(79+20)(79+22)(79+20)(79+22)}} = 0.5802.$$

This is a good correlation between predicted and actual values, the kNN classifier performed well on the data set.

2.6.7 The Receiver Operating Characteristic Curve

Receiver Operating Characteristic Curves (ROC's) are commonly used to visualize a classifiers performance (Moore IV et al., 2014). ROC curve is a plot of false positive rates (F_{pr}) plotted on the x-axis, against true positive rates (T_{pr}) plotted on the y-axis. ROC curves can be interpreted as the average true positive rate for different false positive rates (Mandrekar, 2010). A desirable classifier would have

a high true positive rate and simultaneously a low false positive rate. The F_{pr} and T_{pr} rates are plotted at different classification thresholds and are given by

$$F_{pr} = \frac{F_p}{F_p + T_n} \text{ and } T_{pr} = \frac{T_p}{T_p + F_n}.$$

Some binary classifiers output the probability an observation belongs to a class; in this case classification thresholds are used to classify observations. A classification threshold is the cut-off probability that determines which class an unclassified observation belongs to (Freeman and Moisen, 2008). To illustrate this consider two classes, A and B. The goal is to classify observations into one of the two classes. Any observation with a probability score less than the threshold value is classified as belonging to group B and any observation with a probability score greater than the threshold value is classified as belonging to group A.

A discrete classifier is a classifier that only outputs a prediction instead of a score. Since there is no threshold value, discrete classifiers only output one value for T_{pr} and F_{pr}. Thus on the ROC curve discrete classifiers appear as a single point (Fawcett, 2006). The kNN classifier is an example of a discrete classifier.

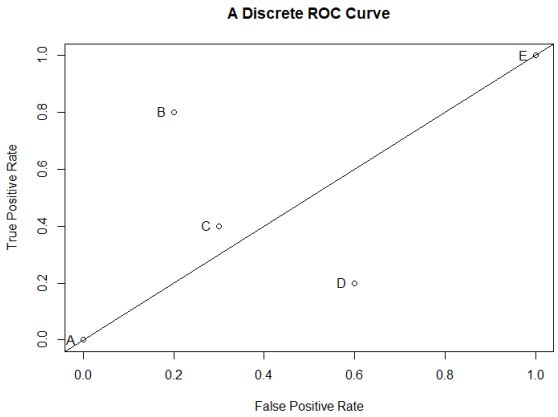


Figure 2.28: A Discrete ROC Curve (adapted from Fawcett (2006)).

Figure 2.28 is derived from simulated data and depicts an ROC curve displaying the performance of five discrete classifiers, named A, B, C, D and E. The diagonal line $T_{pr} = F_{pr}$ represents a classifier's performance that is no better than tossing a fair coin (Fawcett, 2006). Any point that appears below or on the diagonal line represents a classifier that has performed poorly.

In Figure 2.28 the classifiers A, E, and D appear either on or below the diagonal line. These classifiers have performed poorly and are not suitable to be used for the classification problem. Classifiers B and C both appear above the diagonal line and should be considered. However classifier B has both a higher true positive rate and a lower false positive rate and is considered to be the best performing classifier.

Consider the kNN text classifier (Section 2.6.2) From Table 2.18 T_{pr} and F_{pr} are $T_{pr} = \frac{79}{22+79} = 0.782$ and $F_{pr} = \frac{20}{20+79} = 0.202$.

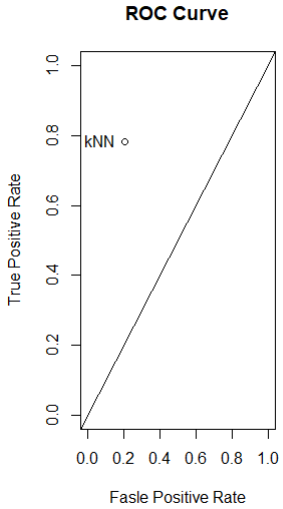


Figure 2.29: ROC Curve for kNN Text Classification from Section 2.6.2

From the true positive and false positive rates ROC in Figure 2.29 is obtained. In the plot the kNN classifier appears above the diagonal line and the true positive rate is higher than the false positive rate indicating the kNN classifier has performed well.

2.6.8 Area Under the Receiver Operating Characteristic Curve

The area under the receiver operating characteristic curve (AUC) provides a single numerical summary of a classifier's performance (Ling et al., 2003). AUC measures how well a classifier can distinguish between two classes. Since AUC is a scalar quantity it makes the comparison of two different classifiers easier. AUC directly measures the ranking of a classifier, meaning classifiers with higher AUC should be selected for the problem (Huang and Ling, 2005). Since AUC is a proportion of area it returns values in the range of $AUC \in [0,1]$. The diagonal line $T_{pr} = F_{pr}$ represents a classifier that classifies observations randomly and has an area of 0.5. Any classifier with an AUC less than or equal to 0.5 is a poorly performing classifier that is not suitable for the problem (Fawcett, 2006).

2.6.9 Cross-Validation

Cross-Validation (CV) is a sampling technique used to estimate a models predictive error (Hastie et al., 2009). A models training error is calculated by applying the model to the training data. However a models training error tends to underestimate a models test error (James et al., 2013).

CV estimates the test error by randomly partitioning the data $\mathscr{L}$ into a training set, denoted by $\mathscr{L}_{train}$, a validation set, denoted by $\mathscr{L}_{val}$. The model is trained on the training set. The trained model is then applied on the validation set to estimate the test error (James et al., 2013). CV assumes the data is identically distributed and that each of the sets are independent of each other (Little et al., 2017). CV can also be used to assesses whether a model is overfitting the training data (Soper, 2021). Overfitting occurs when a model learns the noise in the training data (James et al., 2013). Noise in data are the patterns created by chance that exist in the training data but not in the test set (James et al., 2013). Overfitting is detected when a model has a small training error but a large test error (James et al., 2013).

2.6.9.1 K-Fold Cross-Validation

Often there is not enough data to have a set of data solely for validation. K-Fold cross-validation (KCV) is a technique used to get around this problem (Hastie et al., 2009). KCV is a validation technique where the data, $\mathscr{L}$ is randomly partitioned into K mutually exclusive sets of approximate equal size called folds. The partitioning is done by randomly sampling instances from the data without replacement (Soper, 2021). One fold is used as a validation set and the other $K-1$ folds are used to train the model. The trained model is applied on the validation set and an estimate for the test error is obtained. This training and validation process is repeated K times, where each time a different fold is used as the validation set. This produces a set of K estimates of the test error (Hastie et al., 2009). The average of each folds test error estimate is used as the cross-validated estimate of test error, namely

$$\mathrm{CV}_K = \frac{1}{K}\sum_{i=1}^{K} I(y_i \neq \hat{y}_i),$$

where $I(\cdot)$ is an indicator function, y_i is the actual value and $\hat{y}_i$ is the predicted value. In practice, five or ten folds are used in KCV (James et al., 2013).

2.6.9.2 Leave Out One Cross-Validation

The test error rate estimate is highly dependent on the instances in the training and validation sets which are determined randomly. If K is too small, for example if $K = 2$, the estimate may over or under estimate test error depending on instances the validation and training sets (James et al., 2013). When $K = n$, this is known as leave out one cross-validation (LOOCV) (Hastie et al., 2009). With LOOCV there is no randomness with the partitioning of the training and test sets. Thus for a given trained classifier the LOOCV estimate for test error will always be the same no matter how many times the process is repeated. LOOCV does not scale well if the data set is large as the process is computationally expensive and impractical (James et al., 2013).

2.6.10 Hyperparameter Tuning

Most models have parameters that must be specified before the model runs. These parameters are called hyperparameters (Probst et al., 2019). Hyperparameters govern the structure of a model. For example with the k nearest neighbours algorithm, the value of k is a hyperparameter (Claesen and De Moor, 2015). Hyperparameters can not be learnt during training. Hyperparameters have to be optimized to produce the best performing model (Probst et al., 2019).

2.6.10.1 Grid Search

Grid search cross-validation (GSCV) is an optimization technique used to determine a model's optimal hyperparameters (Adnan et al., 2022). GSCV finds optimal hyperparameters by exhaustively searching over a given set of the hyperparameter space (Liashchynskyi and Liashchynskyi, 2019). GSCV works by training multiple models, each with a unique combination of hyperparameters (Adnan et al., 2022). The hyperparameters that result in the best performing model are then selected as the optimal hyperparameters (Adnan et al., 2022).

Chapter 3

Model Optimization and Improvement

3.1 Imbalanced Data

In binary classification settings, sometimes the data set has an unequal number of observations in each class this is called imbalanced data. The class with the most instances is called the majority class and the minority class is the class with the least instances. (Patel et al., 2020) The ratio of the minority to the majority class may vary from $1:100$ up to and sometimes exceeding $1:10000$. Natural phenomena may cause an imbalance in the data. In the context of credit default hence most debtors are able to pay back their loans, only a minority of debtors default, most credit default data sets are naturally imbalanced. The imbalance may be artificially created. This happens when the data collection is limited this could be due to privacy reasons, limited economic resources etc. (Chawla et al., 2004)

Supervised machine learning models may struggle to learn from imbalanced data and tend to be biased towards the majority class (Thabtah et al., 2020). This is because most models assume that the number of observations in each class are roughly equal (Krawczyk, 2016). The performance metrics used to evaluate a model also have to be reconsidered. Consider the case where the majority class is 99% of the data. If accuracy (see Section 2.6) is used to measure the performance the model may predict that all instances belong to the majority class and achieve an accuracy of 99%. However this model could not detect any of the instances in the minority class. Often times the minority class is of interest and the cost of missclassification is not the same (Krawczyk, 2016). Consider a model used to detect cancerous growths; most cell growths are not cancerous thus this is an imbalanced data set. If the model is unable to detect any cancerous growths it is of no use and could possible do harm to the patients due to the false negative diagnosis (Ganganwar, 2012). Thus suitable techniques must be used to deal with imbalanced data and produce accurate classifier assessment metrics regardless of the class distribution of the data.

3.1.1 Sampling Procedures

Challenges imbalanced data brings can be dealt with in several ways; some of the techniques developed tackle the problem at the data-level. At the data-level, some re-sampling technique is used to generate a balanced data set which is then used to train the model (Patel et al., 2020). Random undersampling (RU) is a sampling technique that balances a data set's class distribution (Ling and Li, 1998). Instances from the majority class are selected at random and then removed from the data set until the classes have an equal number of observations. RU removes information at random. This is a disadvantage as the class distributions may be equal but information that may have been useful in training the model is lost (Ganganwar, 2012).

Kubat et al. (1997) proposed one sided sampling (OSS) as an improvement to RU. Instead of randomly removing majority instances, OSS removes redundant instances. The resulting data set is more balanced and only data that is not useful in training the model is removed. OSS uses Tomek links to remove data (Tomek, 1976). Consider a pair of points $(\mathbf{x}_i, \mathbf{x}_j)$ where $\mathbf{x}_i$ and $\mathbf{x}_j$ belong to different classes. The pair is a Tomek link if there is no point $\mathbf{z}_i$ from a different class label than $\mathbf{x}_i$ or $\mathbf{x}_j$ such that $d(\mathbf{x}_i, \mathbf{x}_j) < d(\mathbf{z}_i, \mathbf{x}_i)$ and $d(\mathbf{x}_i, \mathbf{x}_j) < d(\mathbf{z}_i, \mathbf{x}_j)$, where $d(\cdot)$ is the Euclidean distance between two points. Let $\mathscr{L}$ denote the imbalanced training set and let $\mathscr{C}$ contain all instances from the minority class and one randomly selected instance from the majority class label. Using $\mathscr{C}$ we classify observations from $\mathscr{L}$ using a 1-NN classifier (kNN where $k = 1$). All missclassified observations are sent to $\mathscr{C}$. Any instance from the majority class found to be a Tomek link is then removed from $\mathscr{C}$ resulting in a new balanced data set $\mathscr{L}^1$ (Kubat et al., 1997).

Random oversampling (RO) randomly samples, with replacement, instances in the minority class and duplicates them until the class distributions are equal. This may lead to overfitting as duplicate instances are produced (Ganganwar, 2012).

Chawla et al. (2002) suggested a synthetic minority oversampling technique (SMOTE) as a method to oversample the minority class. Unlike RO that operates at the data-level, SMOTE produces instances by operating in the feature space (Chawla et al., 2002). Instead of producing duplicate instances, SMOTE generates synthetic instances to populate the minority class (Ganganwar, 2012). Synthetic instances are found in the following manner. For each minority instance $\mathbf{x}_i$, find the k-nearest neighbours. The number of nearest neighbours selected is dependent on the number of synthetic instances that need to be generated to balance the data. If the minority class needs to be increased by 100% then one of the k nearest neighbours is selected. Each synthetic instance is generated by finding the difference $\mathbf{d}$ between the feature vectors of the sample and the nearest neighbour, $\mathbf{x}_{neighbour}$. A number $a \in (0,1)$ is then selected at random. The difference $\mathbf{d}$ is multiplied by the random number a. This result is added to the original instance producing a synthetic observation, that is $\mathbf{x}_{smote} = \mathbf{x}_i + \mathbf{d} \times a$. This operation places the synthetic instance at a random distance between the original instance and

the selected nearest neighbour (Chawla et al., 2002). SMOTE generates a larger decision region by populating the minority class with synthetic instances. The idea is that if the region is more general the classifiers can generalize better (Chawla et al., 2002).

3.2 Principal Components Analysis

Principal components analysis (PCA) is an unsupervised technique often used for dimensionality reduction (Wiskott, 2016). Not only does PCA project data onto a lower feature space, PCA preserves as much as possible of the original variation in the data set (Smith, 2002). This means trends and patterns in the original data set are retained in a lower feature space (Smith, 2002). This reduces the computational resources needed to train a model. PCA is used across many fields such as facial recognition, image compression and neuroscience (Paul et al., 2013). The introduction of PCA is attributed to Pearson (1901) but Hotelling (1933) independently developed the technique and named it PCA.

3.2.1 The Derivation of Principal Components

Consider a data matrix $\mathbf{X}_{n\times p} = [\mathbf{X}_1, \mathbf{X}_2, ..., \mathbf{X}_p]$ for each feature $\mathbf{X}_i$, the mean is subtracted such that $\overline{\mathbf{X}} = [0, 0, ..., 0]^{'}$. A consequence of setting the mean of each feature to zero is that the variance of $\mathbf{X}$ is reduced to

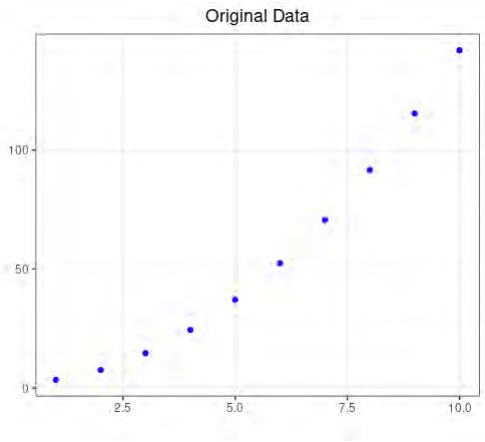
$$S = \text{var}(\mathbf{X}) = \frac{1}{n}\left(\mathbf{X}'\mathbf{X} - \overline{\mathbf{X}}'\overline{\mathbf{X}}\right) = \frac{1}{n}\left(\mathbf{X}'\mathbf{X} - \mathbf{0}'\mathbf{0}\right) = \frac{1}{n}\mathbf{X}'\mathbf{X}$$

PCA aims to find a linear transformation to project the data onto a lower dimensional space (Maćkiewicz and Ratajczak, 1993). PCA is done by finding some matrix $\mathbf{U} \in \mathscr{R}^{p\times k}$ subject to the constraint $\mathbf{U}^{'}\mathbf{U} = \mathbf{I}$. The matrix $\mathbf{U}$ is used to find a linear combination $\mathbf{V} = \mathbf{XU}$, where $\mathbf{V} \in \mathscr{R}^{n\times k}$ and $k \leq p$ (Maćkiewicz and Ratajczak, 1993). Since k is less than p, the linear combination projects the original p-dimensional data onto a lower k-dimensional feature space. Consider the linear transformation given by,

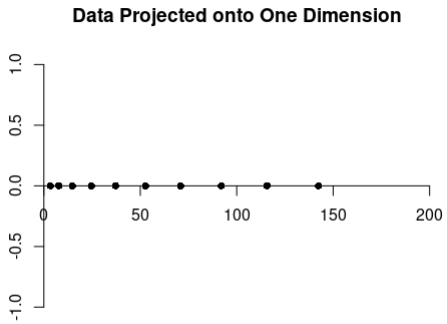
$$\hat{\mathbf{X}} = \mathbf{XU}\mathbf{U}^{'} = \mathbf{V}\mathbf{U}^{'}. \tag{3.1}$$

Since $\mathbf{U} \in \mathscr{R}^{p\times k}$ and $\mathbf{V} \in \mathscr{R}^{n\times k}$, the linear transformation in Equation 3.1 projects the data from $\mathbf{V}$ onto the original feature space, that is $\hat{\mathbf{X}} \in \mathscr{R}^{n\times p}$ (Valle et al., 1999). The matrix denoted by $\hat{\mathbf{X}}$ approximates the original data $\mathbf{X}$ (Samadi et al., 2018). Consider a simulated two dimensional problem where the data is projected onto one dimension and projected back onto two dimensions. Figure 3.1 adapted from Wiskott (2016) illustrate the operation in Equation 3.1. The original two dimensional data is depicted in Figure 3.1a. Some linear combination $\mathbf{V} = \mathbf{XU}$ projects the data onto one dimension depicted in Figure 3.1b. Figure 3.1c depicts the approximated data found using the linear combination in Equation 3.1.

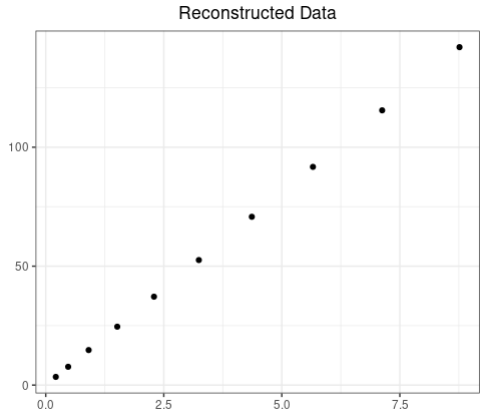
Since the projected data is in a lower dimensional space than the original data, when approximating the original data some information is lost (Wiskott, 2016). Thus the reconstructed and original data differ. Figure 3.1d illustrates this with a combined plot of the two data sets.



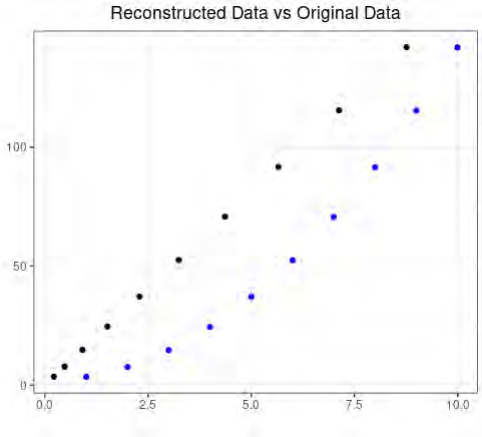
(a) The Original Data.



(b) The Data Projected onto One Dimension.



(c) Reconstruction of the Original Data.



(d) Reconstruction vs Original Data.

Figure 3.1: Deconstruction and Reconstruction of the Original Data (adapted from Wiskott (2016)).

PCA is an optimization problem where a matrix $\mathbf{U}$ is found such that the reconstruction error is minimized (Lin et al., 2011). The reconstruction error is given by,

$$\begin{aligned}
||\mathbf{X}-\hat{\mathbf{X}}||_F^2 =& \text{tr}\left(\left(\mathbf{X}-\mathbf{X}\mathbf{U}\mathbf{U}^{'}\right)^{'}\left(\mathbf{X}-\mathbf{X}\mathbf{U}\mathbf{U}^{'}\right)\right) = \text{tr}\left(\left(\mathbf{X}-\mathbf{X}\mathbf{U}\mathbf{U}^{'}\right)\left(\mathbf{X}-\mathbf{X}\mathbf{U}\mathbf{U}^{'}\right)^{'}\right) \\
=& \text{tr}\left(\mathbf{X}\mathbf{X}^{'}\right) - 2\text{tr}\left(\mathbf{X}\mathbf{U}\mathbf{U}^{'}\mathbf{X}^{'}\right) + \text{tr}\left(\mathbf{X}\mathbf{U}\mathbf{U}^{'}\mathbf{X}^{'}\right) = \text{tr}\left(\mathbf{X}\mathbf{X}^{'}\right) - \text{tr}\left((\mathbf{X}\mathbf{U})(\mathbf{U}^{'}\mathbf{X}^{'})\right) \\
=& \text{tr}\left(\mathbf{X}^{'}\mathbf{X}\right) - \text{tr}\left((\mathbf{U}^{'}\mathbf{X}^{'})(\mathbf{X}\mathbf{U})\right) = \text{tr}\left(\mathbf{X}^{'}\mathbf{X}\right) - \text{tr}\left(\mathbf{V}^{'}\mathbf{V}\right) \\
=& \text{tr}\left(\mathbf{X}^{'}\mathbf{X}\right) - (n-1)\text{tr}\left(\mathbf{S}_v\right),
\end{aligned}$$

where $||\cdot||_F$ is the Frobenius norm, $\text{tr}(A)$ is the trace of the matrix A and $\mathbf{S}_v = \frac{1}{n-1}\mathbf{V}^{'}\mathbf{V}$ (Shalizi, 2016;

Wiskott, 2016). To minimize the reconstruction error, the trace of the sample covariance of the linear projection, $\mathbf{V} = \mathbf{XU}$ needs to be maximized, that is

$$\underset{||\mathbf{U}||=1}{\text{argmax}}\ \text{tr}\left(\mathbf{S}_v\right) = \underset{||\mathbf{U}||=1}{\text{argmax}}\ \text{tr}\left(\frac{1}{n-1}\mathbf{V}'\mathbf{V}\right) = \underset{||\mathbf{U}||=1}{\text{argmax}}\ \text{tr}\left(\frac{1}{n-1}\mathbf{U}'\mathbf{X}'\mathbf{XU}\right) \tag{3.2}$$

$$= \underset{||\mathbf{U}||=1}{\text{argmax}}\ \text{tr}\left(\mathbf{U}'\mathbf{SU}\right) \tag{3.3}$$

where the sample covariance $\mathbf{S}$ is given by $\frac{1}{N-1}\mathbf{X}'\mathbf{X}$ (Ghojogh et al., 2023; Jolliffe and Cadima, 2016) and where $||\mathbf{U}||$ is the norm of $\mathbf{U}$. The matrix $\mathbf{U}'\mathbf{SU}$ can be written as,

$$\mathbf{U}'\mathbf{SU} = \left[\mathbf{u}_1, \mathbf{u}_2 ... \mathbf{u}_k\right]' \mathbf{S} \left[\mathbf{u}_1, \mathbf{u}_2 ... \mathbf{u}_k\right]$$

$$\mathbf{U}'\mathbf{SU} = \begin{bmatrix} \mathbf{u}_1'\mathbf{S}\mathbf{u}_1 & \cdots & \mathbf{u}_1'\mathbf{S}\mathbf{u}_k \\ \vdots & \ddots & \vdots \\ \mathbf{u}_k'\mathbf{S}\mathbf{u}_1 & \cdots & \mathbf{u}_k'\mathbf{S}\mathbf{u}_k \end{bmatrix}. \tag{3.4}$$

Substituting Equation 3.4 into the constraint in Equation 3.3 yields

$$\underset{||\mathbf{u}_1||=1}{\text{argmax}}\ \text{tr}\left(\mathbf{S}_v\right) = \underset{||\mathbf{u}_1||=1}{\text{argmax}}\ \text{tr}\left(\mathbf{U}'\mathbf{SU}\right) = \mathbf{u}_1'\mathbf{S}\mathbf{u}_1 + \mathbf{u}_2'\mathbf{S}\mathbf{u}_2 + ... + \mathbf{u}_k'\mathbf{S}\mathbf{u}_k.$$

Consider the case where only one principal component is derived, that is $k = 1$, $\mathbf{U} = [\mathbf{u}_1]$. The component $\mathbf{Xu_1}$ and can be found by maximizing $\text{var}(\mathbf{Xu}_1)$ by solving for $\mathbf{u}_1$ such that,

$$\underset{||\mathbf{u}_1||=1}{\text{argmax}}\ \text{tr}\left(\mathbf{u}_1'\mathbf{S}\mathbf{u}_1\right) = \underset{||\mathbf{u}_1||=1}{\text{argmax}}\ \text{tr}\left(\frac{\mathbf{u}_1'\mathbf{S}\mathbf{u}_1}{\mathbf{u}_1'\mathbf{u}_1}\right).$$

This is known as the Raleigh quotient of a symmetric matrix and can be solved using a Lagrange multiplier (Ghojogh et al., 2023),

$$\mathscr{L}\left(\mathbf{u}_1, \lambda_1\right) = \mathbf{u}_1'\mathbf{S}\mathbf{u}_1 - \lambda_1\left(\mathbf{u}_1'\mathbf{u}_1 - 1\right),$$

Taking the derivative of $\mathscr{L}\left(\mathbf{u}_1, \lambda_1\right)$ and setting $\mathscr{L}\left(\mathbf{u}_1, \lambda_1\right) = 0$ yields

$$\begin{aligned}\frac{\partial \mathscr{L}(\mathbf{u}_1,\lambda_1)}{\partial \mathbf{u}_1} &= 2\mathbf{S}\mathbf{u}_1 - 2\lambda_1\mathbf{u}_1 = 0\\ \mathbf{S}\mathbf{u}_1 &= \lambda_1\mathbf{u}_1\\ \mathbf{u}_1^{'}\mathbf{S}\mathbf{u}_1 &= \lambda_1(\mathbf{u}_1^{'}\mathbf{u}_1)\\ \mathbf{u}_1^{'}\mathbf{S}\mathbf{u}_1 &= \lambda_1.\end{aligned}$$

Thus $\mathbf{u}_1$ is an eigenvector of $\mathbf{S}$ with eigenvalue λ_1. Since $\text{Var}(\mathbf{X}\mathbf{u}_1) = \mathbf{u}_1^{'}\mathbf{S}\mathbf{u}_1$, the eigenvalue λ_1 is also the variance of $\mathbf{X}\mathbf{u}_1$. Eigenvector $\mathbf{u}_1$ must be selected such that the corresponding eigenvalue is the largest eigenvalue of $\mathbf{S}$. The second principal component is found by solving $\underset{||\mathbf{u}_2||=1}{\text{argmax}}\ \mathbf{u}_2^{'}\mathbf{S}\mathbf{u}_2$ and is also an eigenvector of $\mathbf{S}$, with eigenvalue λ_2 such that $\lambda_1 \geq \lambda_2$. The eigenvectors of symmetric matrices are orthogonal thus $\mathbf{u}_1^{'}\mathbf{S}\mathbf{u}_2 = 0$. To find k principal components, this process would be repeated k times such that $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2 ... \mathbf{u}_k]$ and $\mathbf{U}^{'}\mathbf{S}\mathbf{U}$ reduces to,

$$\mathbf{U}^{'}\mathbf{S}\mathbf{U} = \begin{bmatrix} \mathbf{u}_1^{'}\mathbf{S}\mathbf{u}_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \mathbf{u}_k^{'}\mathbf{S}\mathbf{u}_k \end{bmatrix} = \begin{bmatrix} \lambda_1 & \cdots & 0 \\ 0 & \ddots & \vdots \\ 0 & \cdots & \lambda_k \end{bmatrix}$$

where $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_k$. The matrix denoted by $\mathbf{X} \in \mathscr{R}^{n \times p}$, generally has $\min(n-1, p)$ unique principal components (Hastie et al., 2009). A standard method is to keep any component $\mathbf{X}\mathbf{u}_i$, whose eigenvalue λ_i, is greater than the average eigenvalue (Abdi and Williams, 2010). All components whose eigenvalues satisfy,

$$\lambda_i > \frac{1}{n}\sum_{i=1}^{n}\lambda_i,$$

are retained (Abdi and Williams, 2010).

Chapter 4

Defaults in Payment in the Taiwan Credit Data

4.1 Introduction

Yeh and Lien (2009) used the Taiwan credit card default data set to estimate the probability of default. This is the same data set considered in this study. The data set contains 30 000 observations with 23 features and one target variable. There are a total of 6 636 defaults, that is 22.12% of the account holders defaulted on their payments. Some features are recorded over time, for example, history of payments which tracks the six month payment history of the account holder from April to September 2005. The data set is denoted by $\mathbf{X}$, where $\mathbf{X} = [\mathbf{X}_1, \mathbf{X}_2, ..., \mathbf{X}_{24}]$. The features are denoted by $\mathbf{X}_1 - \mathbf{X}_{23}$ and the target variable is denoted by $\mathbf{X}_{24}$.

X_1 denotes the variable named "Limit_bal" in the datafile which measures the amount of credit given to the account holder. All amounts are measured in the New Taiwan dollar (NT dollar). X_2 denotes "Sex", the account holder's recorded sex, where 1 denotes a male and 2 denotes a female. X_3 denotes "Education" the account holder's highest level of formal education, where 1 denotes graduate school, 2 denotes university, 3 denotes high school and 4 denotes other. X_4 denotes "Marriage" the marital status of the account holder, where 1 denotes married, 2 denotes single and 3 denotes other. X_5 denotes "Age" the account holder's age recorded in years.

Variables $X_6 - X_{11}$ denote "Pay_0 - Pay_6", measures of the monthly payment history as recorded from April to September 2005. X_6 denotes the repayment status in September, X_7 denotes the repayment status in August. Similarly $X_8 - X_{11}$ denotes the repayment status of the months from July to April. The repayment status is encoded as follows: -1 denotes the account holder is not behind in repayment, 1 denotes the account holder is one month behind in repayment, 2 denotes the account holder is two months behind in repayments ..., 8 denotes the account holder is eight months behind in repayments.

Variables $X_{12} - X_{17}$ denote "Bill_amt1 - Bill_amt6", measures of the bill statement in NT dollars from September to April. Similarly to monthly payment history, X_{12}, measures the bill statement in September and $X_{13} - X_{17}$ measures the bill statements from August to April. $X_{18} - X_{23}$, "Pay_amt1-Pay_amt6", measures the previous payment in NT dollars. Similarly to bill statement, X_{18} measures the amount paid in September and $X_{19} - X_{23}$ measures the amounts paid from August to April. X_{24}, named "Default", denotes the default status of the account holder, where 1 denotes a default and 0 denotes an account that has not defaulted.

4.2 Exploratory Data Analysis

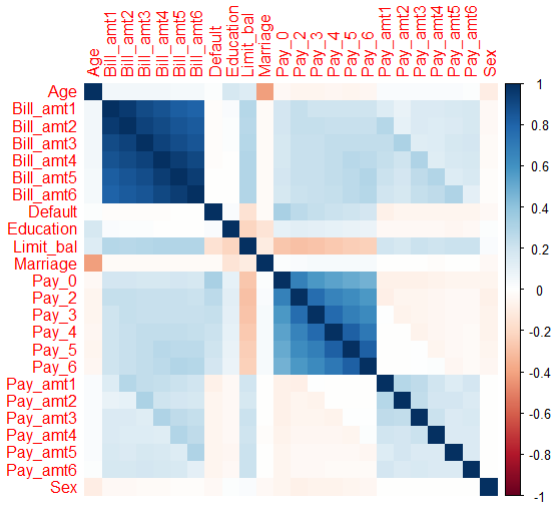


Figure 4.1: Feature Correlation of the Feature Variables.

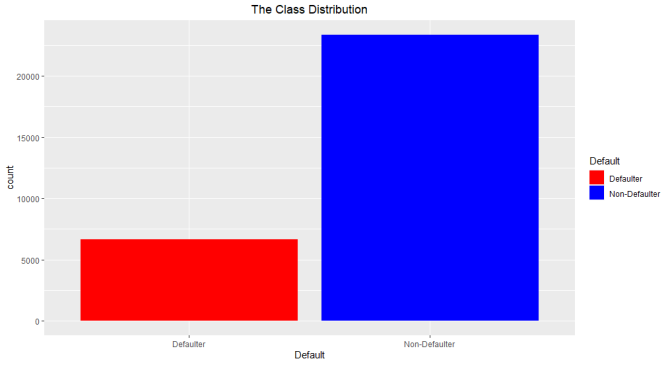


Figure 4.2: Class Distribution of Default in Payment.

Exploratory data analysis (EDA) is a set of methods used to analyze the data where the aim is to examine the data and get a better understanding of the data's properties (Morgenthaler, 2009). Most methods assess the data by visualizing the variables or by using descriptive statistics to better understand the data and its attributes (Komorowski et al., 2016). This understanding is helpful when selecting the classification methods applied on the data, for example discovering if the data is imbalanced and hence applying sampling techniques to balance the class distribution (Deutch et al., 2020).

Correlation measures the strength of the linear relationship between two quantitative variables (Ratner, 2009). Figure 4.1 depicts a heat map of the correlation between the variables in the data set. The diagonal components have a correlation of one as each variable is perfectly correlated to itself. The target variable default is slightly positively correlated to payment history, "Pay_0-Pay_6". The target variables is also slightly negatively correlated to the amount of credit given "Limit_bal". The amount of bill statement features, "Bill_amt1-Bill_amt6" are highly positively correlated to each other. Similarly, the payment history variables, "Pay_0-Pay_6" and previous payment variables "Pay_amt1-Pay_amt6" are also positively correlated to each other.

Figure 4.2 depicts the data's class distribution. Of the 30000 observations, 6636 (22.12%) account holders defaulted on their credit card payments.

Figure 4.3 depicts the level of education grouped by default. This variable had three categories that were not described in the datafile, these categories have been labeled as unknown variables. These data do not provide sufficient evidence of a relationship between the level of education and defaulting or not. Figure 4.4 depicts sex grouped by defaults. There are more females than males in this data set.

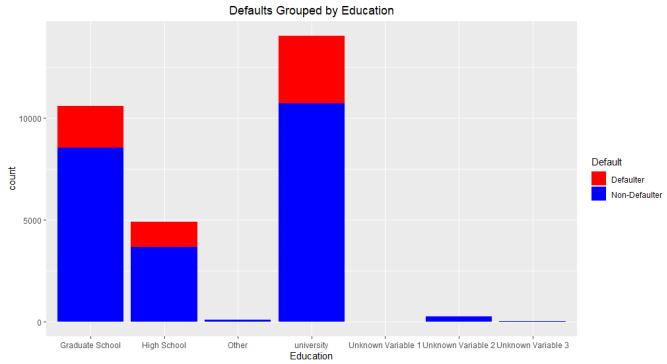


Figure 4.3: Education Level Grouped by Defaults.

For each sex, the proportion of defaults to non-defaults is very similar. There seems to be no apparent relationship between sex and defaulting or not.

Figure 4.5 depicts age grouped by defaults. The age distribution for non-defaulters and defaulters is very similar, with the median age for both defaulters and non-defaulters being thirty-four. Account holders above the age of sixty-four are very unlikely to default on their payments.

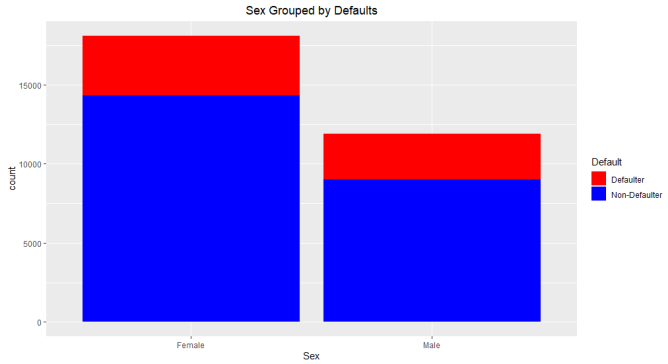
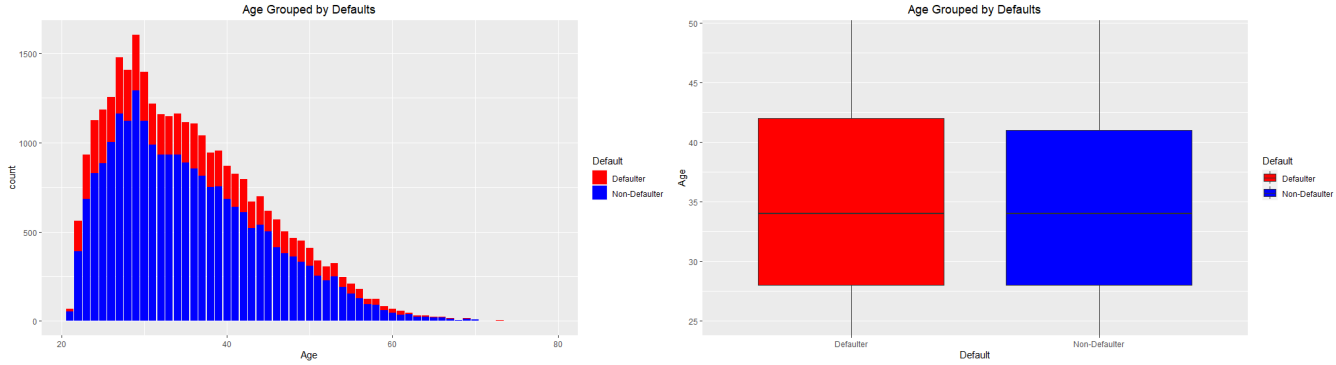


Figure 4.4: Sex Grouped by Defaults.



(a) **(b)**

Figure 4.5: Age Grouped by Defaults.

Figure 4.6 depicts payment history grouped by defaults. The distribution of defaults and non-defaults for payment history is reasonably consistent with time. The exception is "Pay_0", which is the final month of September where the number of people one month behind in payments increased.

Figure 4.7 depicts the bill statements grouped by default. The distribution of the statements is consistent with time. The defaulters have a slightly lower median than the non-defaulters.

Figure 4.8 depicts previous payments grouped by default. The distribution of the payments is consistent with time. Defaulters consistently pay less than non-defaulters.

The data set was randomly split into 70% training and 30% test sets. From Table 4.1 the imbalance ratio between the original, training and test sets are approximately equal, thus the ratio of defaults to non-defaults in the two data sets is roughly equal. Before training the classifiers, the categorical variables were converted to numeric variables and the data was then normalized. Yeh and Lien

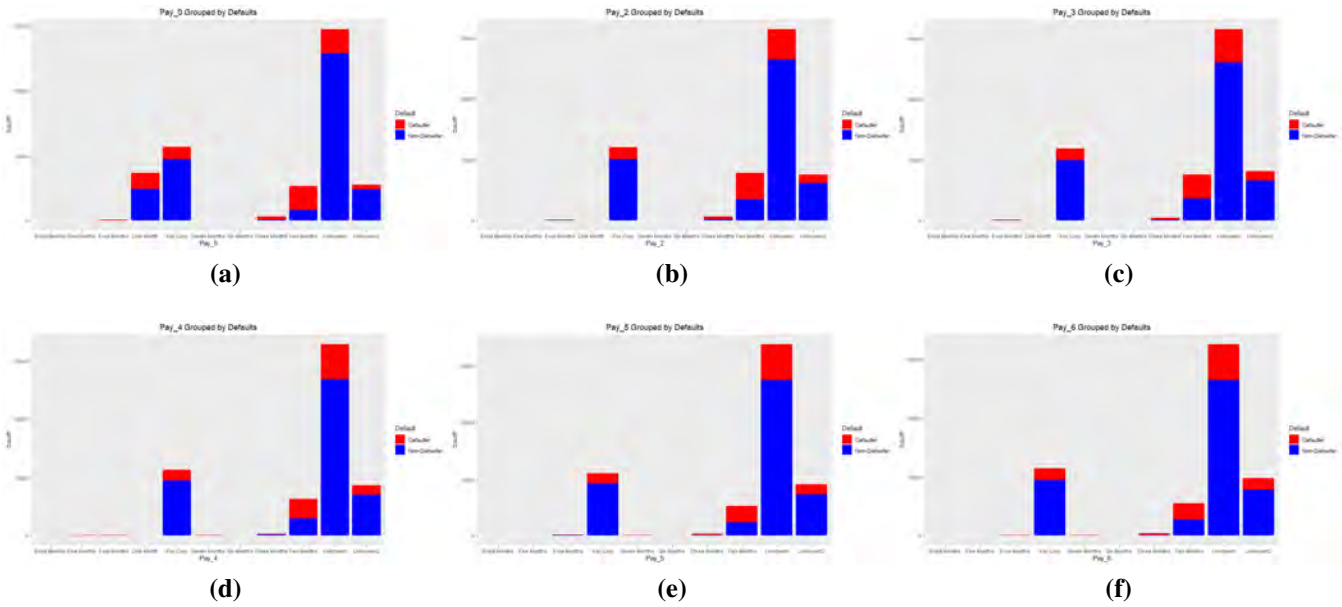


Figure 4.6: Payment History Grouped by Defaults.

Table 4.1: Imbalance of the Taiwan Data Sets.

Data Set	N	Number of Defaults	Imbalance Ratio
Original Data	30 000	6 636	3.5208
Training Data	21 000	4 681	3.4862
Test Data	9 000	1 955	3.6036

(2009) trained five classifiers, namely kNN, LR, Discriminant analysis, Naive Bayes and CTs. As per Yeh and Lien (2009), before seven classifiers, namely k-nearest neighbours (kNN), linear discriminant analysis (LDA), classification and regression trees (CART), random forests (RF), adaptive boosting (AdaBoost), extreme gradient boosting (XGBoost) and gradient boosted trees with synthetic features (SynBoost) were trained and tested on the data. During training, grid search was used to find the optimal hyperparameters. For each classifier, Accuracy was used to select the optimal hyperparameters. The hyperparameters that resulted in the highest Accuracy were determined to be the optimal hyperparameters.

The hyperparameter for the kNN classifier is the value of k. Four values in the hyperparameter space were considered, namely 5, 20, 40 and 170. The hyperparameter for CART is the complexity parameter: all values in the range $0.01 - 0.5$ found by 0.01 increments were searched. The hyperparameters for RF are the number of features for each split (mtry) and the minimum node size. The hyperparameter space searched for minimum node size between 1, 5 and 10. The space searched for mtry was 3, 6 and 9. The hyperparameters for Adaboost are the number of trees and the maximum tree depth. The hyperparameter space searched for the maximum number of trees was 150 and 300. The space searched for the maximum tree depth was 1, 2 and 5. The hyperparameters for XGBoost are

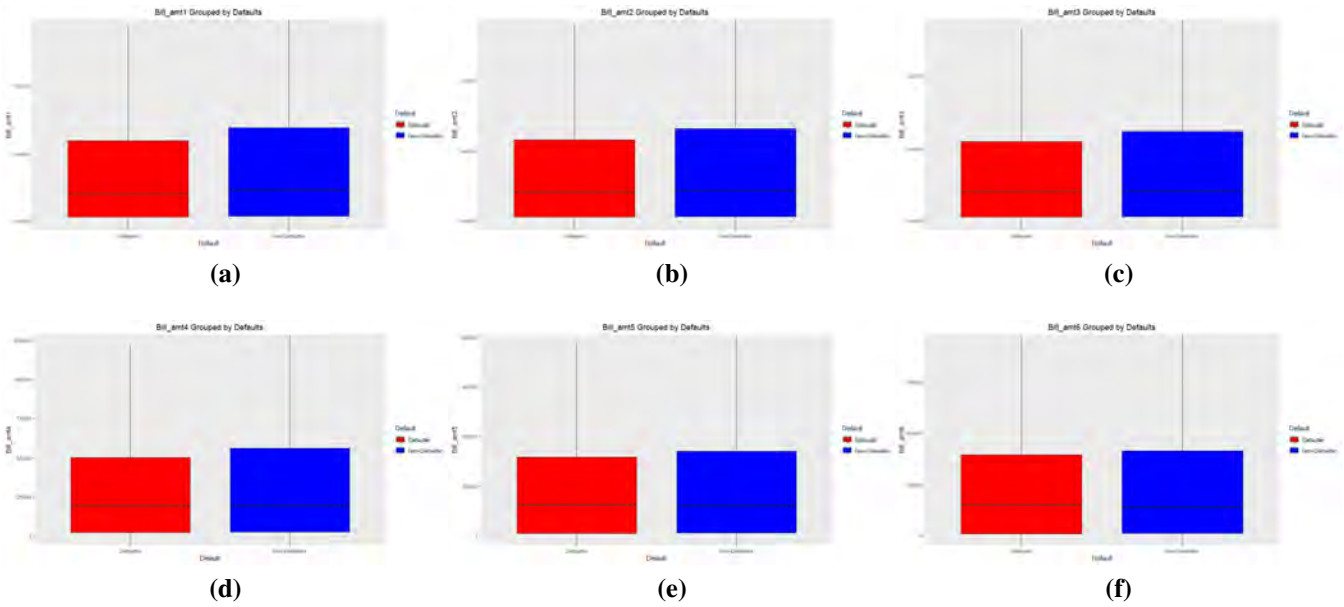


Figure 4.7: Bill Statement Grouped by Defaults.

the number of trees, learning rate, gamma (λ) and maximum tree depth. The space searched for the number of trees were 150 and 300. The space searched for maximum tree depth was 1 and 3. The space searched for gamma was 0.1 and 0.3. The space searched for learning rate was 0.01 and 0.1. The SynBoost classifier is an ensemble of XGBoost trees; the same hyperparameter space as XGBoost was searched for each base learner. Tables A.4-A.8 summarize the optimal hyperparameters selected for each classifier on each training set. The classifiers were subsequently trained and tested using tuned hyperparameters. Appendix 5.2 contains the Rscript used to implement all statistical methods.

4.3 The Results

Tables 4.2 and 4.3 detail the performance of the models on the training and test sets. In this study, AUC is used to select the best performing classifier, the higher the AUC the better the performance as per Section 2.6.8.

From the training data, Table 4.2, the MCC score is low for all classifiers except RF which achieved a MCC of 0.8250. The recall is also low for all classifiers except RF which achieved a score of 0.7473. The precision is high for all classifiers, this is due to the data being imbalanced, thus most instances belong to the majority class and are correctly classified. RF have the highest precision of 0.9831. The AUC scores for all classifiers are high, which shows that the classifiers perform better than random chance (AUC = 0.5). RF have the highest AUC score of 0.9577. The F_1 score is low for all classifiers except RF which scored 0.8491. Most classifiers did not perform well on the training data, however

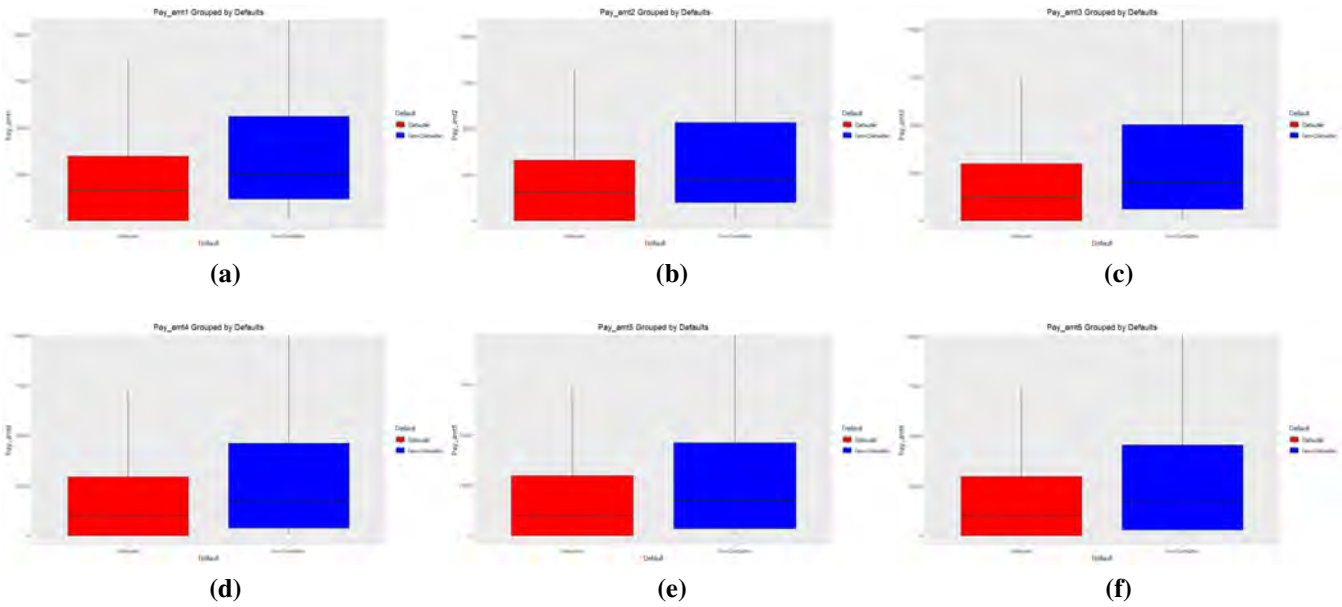


Figure 4.8: Previous Payment Grouped by Defaults.

Table 4.2: The Performance on the Training Data.

Classifier	MCC	Recall	Precision	AUC	F_1 Score
kNN	0.3543	0.2984	0.6652	0.7457	0.4120
LDA	0.3379	0.2508	0.7005	0.7595	0.3693
CART	0.3828	0.3216	0.6913	0.7613	0.4389
RF	0.8250	0.7473	0.9831	0.9577	0.8491
AdaBoost	0.3962	0.3388	0.6953	0.7650	0.4556
XGBoost	0.3828	0.3215	0.6913	0.7613	0.4389
SynBoost	0.4022	0.3536	0.6884	0.7629	0.4672

RF performed best on the training data with for all performance metrics including an AUC score of 0.9577.

From the test data, Table 4.3, the MCC for all classifiers is low; SynBoost has the highest MCC of 0.4215. Similarly the recall is low with; RF having the highest recall score of 0.3872. Precision is high for all classifiers; CART and XGBoost have the joint highest precision of 0.7051. The AUC score for all classifiers is high; CART and XGBoost have the joint highest AUC score of 0.7729. The F_1 score is low for all classifiers, RF have the highest F_1 score of 0.4897. CART and XGBoost are the joint best-performing classifiers on the original data, with the highest AUC on the test data of 0.7729. Overall none of the classifiers performed exceptionally well on the test data. The original data is imbalanced thus, there is a high AUC score for all classifiers. However, considering the AUC score in isolation is misleading as the other metrics are low. RF performed well on the training data (Table 4.2) but the results from the test data (Table 4.3) indicates the classifier may be overfitting as the model performed

Table 4.3: The Performance on the Test Data.

Classifier	MCC	Recall	Precision	AUC	F_1 Score
kNN	0.3640	0.3079	0.6667	0.7498	0.4213
LDA	0.3502	0.2624	0.7037	0.7647	0.3823
CART	0.4072	0.3437	0.7051	0.7729	0.4622
RF	0.4137	0.3872	0.6658	0.7567	0.4897
AdaBoost	0.4131	0.3575	0.6990	0.7710	0.4731
XGBoost	0.4072	0.3437	0.7051	0.7729	0.4622
SynBoost	0.4215	0.3729	0.6963	0.7711	0.4857

poorly on the test data but almost perfectly on the training data.

Previous research has demonstrated that classifiers may struggle to learn from imbalanced data (see Section 3.1). Thus balanced training sets were produced using random oversampling (RO), random undersampling (RU) and synthetic minority oversampling technique (SMOTE). The classifiers were then trained and the hyperparemters tuned on the balanced data sets. Tables 4.4a to 4.6b, describe the performance of the models after balancing the training data.

Table 4.4: The Performance of the classifiers on the RO Training Data.

(a) The Performance on the RO Training Data.

Classifier	MCC	Recall	Precision	AUC	F_1 Score
kNN	0.6586	0.9528	0.7448	0.8421	0.8386
LDA	0.3439	0.6228	0.6888	0.6728	0.6541
CART	0.3870	0.5099	0.7763	0.7061	0.6155
RF	0.9966	0.9999	0.9967	0.9983	0.9983
AdaBoost	0.5991	0.7717	0.8160	0.8000	0.7932
XGBoost	0.4000	0.5150	0.7763	0.7071	0.6192
Synboost	0.3870	0.5099	0.7763	0.7061	0.6155

(b) The Performance on the Test Data after RO.

Classifier	MCC	Recall	Precision	AUC	F_1 Score
kNN	0.2385	0.6742	0.3262	0.5989	0.4396
LDA	0.3184	0.6455	0.3938	0.6371	0.4891
CART	0.3884	0.5366	0.5125	0.6911	0.3884
RF	0.4117	0.4501	0.6032	0.7303	0.5155
AdaBoost	0.3899	0.6348	0.4648	0.6760	0.5366
XGBoost	0.3901	0.5407	0.5121	0.6913	0.5260
Synboost	0.3885	0.5366	0.5125	0.6911	0.5242

From the RO training data, Table 4.4a, the performance of most models has improved compared to the original data set, however, CART has remained the same. The best-performing classifier is RF, with

an AUC of 0.9983. From the test data, Table 4.4b, RF are the best-performing classifier with an AUC of 0.7303. The AUC score test results are slightly worse than those of the classifiers trained on the original data.

Table 4.5: The Performance of the Classifiers on the RU Data.

(a) The Performance on the RU Training Data.

Classifier	MCC	Recall	Precision	AUC	F_1 Score
kNN	0.4208	0.5300	0.7957	0.7236	0.6362
LDA	0.3531	0.6144	0.6964	0.6779	0.6528
CART	0.3953	0.5084	0.7829	0.7114	0.6165
RF	0.9281	0.9780	0.9506	0.9641	0.9641
AdaBoost	0.4352	0.6076	0.7666	0.7227	0.6779
XGBoost	0.4011	0.5210	0.7815	0.7131	0.6252
SynBoost	0.4464	0.6221	0.7687	0.7276	0.6877

(b) The Performance on the Test Data after RU.

Classifier	MCC	Recall	Precision	AUC	F_1 Score
kNN	0.3823	0.5325	0.5073	0.6879	0.5196
LDA	0.31774	0.6348	0.3967	0.6376	0.4883
CART	0.3884	0.5366	0.5125	0.6911	0.5242
RF	0.3864	0.6685	0.4468	0.6701	0.5357
AdaBoost	0.3880	0.6286	0.4655	0.6757	0.5349
XGboost	0.3907	0.5488	0.5083	0.6901	0.5278
SynBoost	0.4034	0.6471	0.4731	0.6820	0.5466

For the RU training data, Table 4.5a, the performance of all the models is similar but slightly improved compared to the models trained on the original data. RF have performed the best an AUC score of 0.9641. For the RU test data, Table 4.5b, the classifiers did not perform well on the test data CART is the best-performing classifier with an AUC of 0.6911. The performance on the test data is similar but lower compared to the models trained on the original data.

For the SMOTE training data, Table 4.6a, SMOTE improved the performance of the kNN, RF and AdaBoost classifiers on the training set. RF are the best performing classifier with an AUC of 0.9989. For the SMOTE test data, 4.6b Synboost is the best performing classifier with an AUC score of 0.7212.

The results of re-sampling the class distributions is a slightly negative effect or decreased the predictive power of the various classifiers. Re-sampling the class distributions has made all the classifier's performance appear slightly worse when compared to the models trained on the original data, for example the AUC scores for all classifiers are highest when trained on the original imbalanced data set. CART and XGboost trained on the original data were the joint best-performing classifiers with an AUC of 0.7729. For the models trained on the RO data, RF were the best-performing classifier with an AUC of 0.7303. For the models trained on the RU data, CART was the best-performing classifier with an AUC

Table 4.6: The Performance of the Classifiers on the SMOTE Data.

(a) The Performance on the SMOTE Training Data.

Classifier	MCC	Recall	Precision	AUC	F_1 Score
kNN	0.6718	0.8442	0.8303	0.8360	0.8372
LDA	0.3417	0.6489	0.6785	0.6710	0.6634
CART	0.3993	0.5234	0.7813	0.7115	0.6269
RF	0.9978	0.9994	0.9984	0.9989	0.9989
AdaBoost	0.7957	0.8620	0.9267	0.8988	0.8969
XGBoost	0.3966	0.5224	0.7793	0.7101	0.6255
SynBoost	0.4153	0.4739	0.8280	0.7297	0.6028

(b) The Performance on the Test Data after SMOTE.

Classifier	MCC	Recall	Precision	AUC	F_1 Score
kNN	0.3070	0.5893	0.4027	0.6359	0.4784
LDA	0.3095	0.6655	0.3802	0.6315	0.4839
CART	0.3885	0.5366	0.5125	0.6911	0.5242
RF	0.3858	0.4910	0.5366	0.6993	0.5128
AdaBoost	0.3738	0.4501	0.5480	0.7013	0.4942
XGBoost	0.3901	0.5407	0.5121	0.6913	0.5260
SynBoost	0.4151	0.4885	0.5784	0.7212	0.5297

Table 4.7: Summary of Best Performing Classifiers after KCV.

Training data	Classifier	AUC
Original Data	XGBoost	0.7732
RO	RF	0.7269
RU	SynBoost	0.7212
SMOTE	SynBoost	0.7217
PCA	LDA	0.7390

of 0.6911. For the models trained on the SMOTE data, SynBoost was the best-performing classifier with an AUC of 0.7212.

The classifier's poor performance may be explained by overfitting. Past research has shown that K-fold cross-validation (KCV) reduces the chances of overfitting (Rao et al., 2008). The classifiers were then trained using 10-fold cross-validation and grid search to find the optimal hyperparameters. The models were then retrained with the optimal hyperparameters and tested on the test data (see Tables A.4-A.8). Tables 4.8a to 4.11b, describe the performance of the models after balancing the training data and applying 10-fold cross-validation.

Table 4.7 summarizes the best performing classifiers for each data set. The performance of the classifiers trained on the cross-validated data were compared to their non cross-validated counter parts. On the original data KCV improved the performance of all classifiers with the exception of CART that

Table 4.8: The Performance of the Classifiers on the Data after KCV.

(a) The Performance on the KCV Training Data.

Classifier	MCC	Recall	Precision	AUC	F_1 Score
kNN	0.4025	0.3619	0.6792	0.7589	0.4722
LDA	0.3379	0.2508	0.7005	0.7595	0.3694
CART	0.3828	0.3215	0.6913	0.7613	0.4389
RF	0.8275	0.7498	0.9846	0.9587	0.8513
AdaBoost	0.3963	0.3435	0.6895	0.7625	0.4586
XGBoost	0.4033	0.3519	0.6926	0.7648	0.4666
SynBoost	0.4027	0.3602	0.6815	0.7599	0.4713

(b) The Performance on the Test Data after KCV.

Classifier	MCC	Recall	Precision	AUC	F_1 Score
kNN	0.3943	0.3591	0.6623	0.7522	0.4657
LDA	0.3503	0.2624	0.7037	0.7587	0.3823
CART	0.4072	0.3437	0.7051	0.7729	0.4622
RF	0.4181	0.3934	0.6670	0.7579	0.4949
AdaBoost	0.4110	0.3596	0.6926	0.7679	0.4734
XGBoost	0.4223	0.3703	0.7009	0.7732	0.4846
SynBoost	0.4197	0.3796	0.6851	0.7660	0.4199

Table 4.9: The Performance of the Classifiers on the Data after RO and KCV.

(a) The Performance on the Cross-Validated RO Training Data.

Classifier	MCC	Recall	Precision	AUC	F_1 Score
kNN	0.6594	0.9530	0.7492	0.8424	0.8389
LDA	0.3439	0.6228	0.6888	0.6728	0.6541
CART	0.3870	0.5099	0.7763	0.7061	0.6155
RF	0.9964	0.9998	0.9966	0.9982	0.9982
AdaBoost	0.6693	0.8313	0.8365	0.8347	0.8339
XGBoost	0.5240	0.7096	0.7897	0.7634	0.7475
SynBoost	0.5134	0.6982	0.7873	0.7584	0.7401

(b) The Performance on the Test Data after RO and KCV.

Classifier	MCC	Recall	Precision	AUC	F_1 Score
kNN	0.2390	0.6747	0.3264	0.5991	0.4400
LDA	0.3184	0.6455	0.3938	0.6371	0.4891
CART	0.3885	0.5366	0.5125	0.6911	0.5242
RF	0.4093	0.4547	0.5958	0.7269	0.5158
AdaBoost	0.3785	0.6159	0.4615	0.6720	0.5276
XGBoost	0.39993	0.6425	0.4709	0.6803	0.5435
SynBoost	0.3849	0.6343	0.4558	0.6725	0.5304

Table 4.10: The Performance of the Classifiers on the Data after RU and KCV.

(a) The Performance on the Cross-Validated RU Training Data.

Classifier	MCC	Recall	Precision	AUC	F_1 Score
kNN	0.4364	0.5608	0.7931	0.7286	0.6570
LDA	0.3531	0.6144	0.6964	0.6779	0.6528
CART	0.3953	0.5084	0.7829	0.7114	0.6165
RF	0.9276	0.9765	0.9515	0.9639	0.9638
AdaBoost	0.4445	0.6355	0.7603	0.7256	0.6923
XGBoost	0.4474	0.6202	0.7706	0.7284	0.6873
SynBoost	0.4146	0.4634	0.8317	0.7318	0.5951

(b) The Performance on the Test Data after RU and KCV.

Classifier	MCC	Recall	Precision	AUC	F_1 Score
kNN	0.3776	0.5555	0.4894	0.6806	0.5204
LDA	0.3177	0.6348	0.3967	0.6376	0.4883
CART	0.3885	0.5366	0.5125	0.6911	0.5242
RF	0.3846	0.6731	0.4432	0.6686	0.5345
AdaBoost	0.3908	0.6552	0.4567	0.6739	0.5382
XGBoost	0.4086	0.6476	0.4783	0.6849	0.5502
SynBoost	0.4152	0.4885	0.5784	0.7212	0.5297

Table 4.11: The Performance of the Classifiers on the Data after SMOTE and KCV.

(a) The Performance on the Cross-Validated SMOTE Training Data.

Classifier	MCC	Recall	Precision	AUC	F_1 Score
kNN	0.6721	0.8445	0.8374	0.8361	0.8374
LDA	0.3382	0.6465	0.6768	0.6693	0.6613
CART	0.3955	0.5193	0.7799	0.7098	0.6235
RF	0.9983	0.9994	0.9988	0.9991	0.9991
AdaBoost	0.7961	0.8670	0.9230	0.8988	0.8941
XGBoost	0.7125	0.7923	0.9030	0.8590	0.8440
SynBoost	0.7278	0.7988	0.9124	0.8668	0.8518

(b) The Performance on the Test Data after Smote and KCV.

Classifier	MCC	Recall	Precision	AUC	F_1 Score
kNN	0.3070	0.5898	0.4024	0.6359	0.4784
LDA	0.3086	0.6660	0.3793	0.6310	0.4833
CART	0.3885	0.5366	0.5125	0.6911	0.5242
RF	0.3887	0.4921	0.5398	0.7011	0.5149
AdaBoost	0.3555	0.4537	0.5205	0.6871	0.4848
XGBoost	0.4056	0.4645	0.5828	0.7211	0.5169
SynBoost	0.3964	0.4382	0.5858	0.7217	0.5013

Table 4.12: PCA Test Data with KCV.

Classifier	MCC	Recall	Precision	AUC	F_1Score
LDA	0.3060	0.2286	0.6593	0.7390	0.3395
kNN	0.3301	0.2808	0.635	0.7309	0.3894
CART	0.3397	0.3591	0.5684	0.7035	0.4401
RF	0.3096	0.3258	0.5491	0.6905	0.4099
AdaBosst	0.3319	0.2941	0.6216	0.7254	0.3993
XGBoost	0.3408	0.3166	0.6123	0.7225	0.4174
SynBoost	0.3461	0.3125	0.6267	0.7296	0.41706

remained the same. The best performing classifier is XGBoost trained on the cross-validated original data with an AUC of 0.7732 (Table 4.8b). On the RO and KCV data, Table 4.9b, all classifiers except CART performed slightly better. The best performing classifier was RF with an AUC of 0.7269. On the RU and KCV data, Table 4.10b, only two classifiers improved in performance namely, SynBoost which had an AUC of 0.7212. On the SMOTE and KCV data, Table 4.11b, the performance of four classifiers improved. Notably CART performance decreased drastically with an AUC of 0.3888. SynBoost had the best performance with an AUC of 0.7217. Overall the classifiers test performance remained roughly the same after tenfold cross-validation.

Principal components analysis (PCA) was applied to reduce the dimensionality of the original training data. Five of the twenty-three components were retained as per Section 3.2. The classifiers were then trained using tenfold cross-validation on the data. Tables A.18 and 4.12 detail the performance of the classifiers on the PCA data.

Although PCA reduced the time required to train the models, this was at the expense of performance. The best performing classier on the PCA data was LDA with an AUC of 0.7390.

4.3.1 Synthetic Features

The SynBoost classifier generates a new training set for each base learner in the ensemble. In this study, the number of base learners in the SynBoost ensemble is equal to seven. For each additional base learner, two synthetic features are produced and added to the training data. Since SynBoost sequentially adds synthetic features and drops weak predictors, the later training sets should have variables deemed to be strong predictors.

Thus for each data set the SynBoost classifier was trained on, the original, the RU, the RO, the PCA and the SMOTE training data sets. The last synthetic data set produced by the seventh base learner in the SynBoost ensemble was extracted. Tables A.9 to A.13 in Appendix 5.2 detail the variables in the synthetic data sets. All six classifiers, excluding Synboost, were trained using the synthetic training data sets. The idea is that SynBoost uses a variety of data sets, including the original data set. The

effects of the synthetic data are not fully observed as the final prediction is aggregated from different data sets, including the original data. The models were trained on the synthetic data using tenfold cross-validation. Tables A.14 to 4.17 detail the classifier's performance on the various synthetic data sets.

Table 4.13: Synthetic Test Data.

Classifier	MCC	Recall	Precision	AUC	F_1-Score
kNN	0.4197	0.3811	0.6835	0.7653	0.4893
LDA	0.3665	0.2772	0.7160	0.7723	0.3997
CART	0.4072	0.3437	0.7051	0.7729	0.4622
RF	0.4057	0.3734	0.6667	0.7559	0.4787
AdaBoost	0.3975	0.3616	0.6651	0.7539	0.4685
XGBoost	0.4072	0.3437	0.7051	0.7729	0.4622

Table 4.14: Synthetic Test Data after RO.

Classifier	MCC	Recall	Precision	AUC	F_1-Score
kNN	0.2383	0.6752	0.3258	0.5988	0.4396
LDA	0.3183	0.6645	0.3876	0.6358	0.4896
CART	0.3477	0.6435	0.4203	0.6521	0.5085
RF	0.3970	0.4675	0.5681	0.7136	0.5129
AdaBoost	0.3744	0.6174	0.4567	0.6695	0.5250
XGBoost	0.3951	0.5361	0.5209	0.6955	0.5284

Table 4.15: Synthetic Test Data after RU.

Classifier	MCC	Recall	Precision	AUC	F_1-Score
kNN	0.3807	0.6235	0.4603	0.6722	0.5297
LDA	0.3369	0.6246	0.4174	0.6483	0.5004
CART	0.3639	0.6404	0.4367	0.6610	0.5193
RF	0.3825	0.6609	0.4463	0.6688	0.5328
AdaBoost	0.3877	0.6619	0.4509	0.6715	0.5364
XGBoost	0.4082	0.5269	0.5430	0.7064	0.5348

Table 4.16: Synthetic Test Data after SMOTE.

Classifier	MCC	Recall	Precision	AUC	F_1-Score
kNN	0.3115	0.5299	0.4295	0.6450	0.4745
LDA	0.2947	0.6598	0.3701	0.6247	0.4742
CART	0.4027	0.5744	0.5084	0.6929	0.5394
RF	0.3714	0.4619	0.5365	0.6964	0.4964
AdaBoost	0.3690	0.4327	0.5537	0.7026	0.4858
XGBoost	0.3993	0.5673	0.5082	0.6921	0.5361

Table 4.17: Synthetic PCA Test Data.

Classifier	MCC	Recall	Precision	AUC	F_1-Score
kNN	0.3406	0.3038	0.6266	0.7288	0.4092
LDA	0.2719	0.1673	0.6943	0.7517	0.2696
CART	0.3323	0.2829	0.6364	0.7320	0.3916
RF	0.3096	0.3233	0.5515	0.6915	0.4076
AdaBoost	0.3337	0.2921	0.6275	0.7282	0.3986
XGBoost	0.3470	0.3141	0.6265	0.7297	0.4184

Tables A.14 to A.19 in Appendix 5.2 details the performance on the training data. Table 4.18 summarizes the classifiers performance when trained on the synthetic data sets. Each classifiers performance is compared to its non-synthetic counterparts for example, the counterpart for the original synthetic test data Table 4.13 is the KCV original test data Table 4.8b. The performance of two classifier improved when trained on the original synthetic data. XGBoost and CART were the best performing classifiers with AUC scores of 0.7729. The performance of five classifiers improved when trained on the RO synthetic data. RF had the highest AUC score of 0.7136. The performance of four classifiers improved when trained on the RU synthetic data. XGBoost had the best performance with an AUC of 0.7064. The performance of three classifiers improved when trained on the SMOTE synthetic data. Adaboost had the highest AUC score of 0.7026. The performance of five classifiers improved when trained on the Synthetic PCA data. LDA was the best performing classifier with an AUC score of 0.7517.

Overall the synthetic features slightly decreased the performance of the best perfroming classifiers with the exception of the RU and SMOTE synthetic data sets. The performance of the classifiers trained

Table 4.18: Summary of Best Performing Classifiers when trained on Synthetic Features.

Data set	Classifier	AUC
Original Synthetic	XGBoost and CART	0.7729
RO Synthetic	RF	0.7136
RU Synthetic	XGBoost	0.7064
SMOTE Synthetic	AdaBoost	0.7026
PCA Synthetic	LDA	0.7517

on the synthetic and non-synthetic data sets are nearly identical, the improvements or hindrance is negligible. The synthetic data sets all had less variables than their non-synthetic counterparts, for example the original data sets have 23 features while the original synthetic data set had 11. This decreased the computational time required to train and apply the models. Since the SynBoost classifier creates synthetic variables by using linear combinations, some of the synthetic variables produced have no real world meaning as some of the variables were originally categorical variables.

4.4 Previous Research using the Taiwan Data

Yeh and Lien (2009) applied six statistical learning techniques, namely Logistic Regression (LR), Naive Bayes (NB), Artificial Neural Networks (ANNS), Discriminant Analysis (DA) and CTs to predict who might default on their loans. Furthermore, Yeh and Lien (2009) introduced a novel technique named sorting smoothing method (SSM) to estimate the probability of defaulting.

Yeh and Lien (2009) used the Taiwanese credit default data set, this is the same data set considered in this study. Yeh and Lien (2009) randomly partitioned the data into a training and test set and noted that since the data is imbalanced, classification accuracy is an inaccurate performance metric. Thus in addition to classification error rate, Yeh and Lien (2009) used the area under a lift chart to assess the classifiers performance. Lift charts are similar to ROC curves (see Section 2.6.7), lift charts measure how well a classifiers predictions compare to random selection (Bekkar et al., 2013). The area under a lift chart or area ratio measures a classifiers performance. A random classifier has an area ratio of 0.5; the higher the area the better the performance (Bekkar et al., 2013). Table 4.19 details the classifiers test performance on the data. The results obtained by Yeh and Lien (2009) are slightly worse than those obtained in this study. The only classifiers performing better than random are NB, ANNs and CTs. Compared to the results in this study where all classifiers performed better than random (see Section 2.6.8 on AUC). From the study conducted by Yeh and Lien (2009) the best performing classifier are ANNs with an area ration of 0.54. However the study only used two performance metrics it is hard to gauge exactly how well the classifiers are performing.

Table 4.19: Test Results Obtained by Yeh and Lien (2009) .

Classifier	Accuracy	Area Ratio
kNN	0.84	0.45
LR	0.82	0.44
DA	0.74	0.43
NB	0.79	0.53
ANNs	0.83	0.54
CT	0.83	0.536

Chapter 5

Conclusion

Credit risk is a problem financial institutions deal with when issuing loans. Statistical learning techniques have been widely used in both academia and industry to predict who might default on their loans. Yeh and Lien (2009) used six statistical learning techniques, namely namely Logistic Regression, Naive Bayes, Artificial Neural Networks, Discriminant Analysis and Classification Trees (CT) to estimate the probability of default. This study added to the work done by Yeh and Lien (2009) by applying a total of seven statistical learning techniques, namely Linear Discriminant Analysis (LDA), k-Nearest Neighbours (kNN), CTs, Random Forests (RF), Adaptive Boosting (AdaBoost), Extreme Gradient Boosting (XGBoost) and Synthetic Boosting (SynBoost) to predict who might default on their loans. Furthermore Yeh and Lien (2009) did not consider the effect imbalanced data has on classifiers. In this study the initial data set was imbalanced thus additional balanced data sets were generated by using random undersampling (RU), random oversampling (RO) and synthetic minority oversampling (SMOTE). This study also investigated which performance metrics are appropriate when the data is imbalanced. The classifiers were trained on the imbalanced data and all the simulated balanced data sets produced. Tenfold cross-validation and hyperparameter tuning were utilised to select the optimal hyperparameters and assess the classifiers performance. PCA was performed in an effort to reduce the dimension of the data and hence complexity of the classifier and hence reduce the computational costs of the various classification methods.

In addition to using the default economic variables available in the data set to train the various classifiers, this study investigated the effects of utilizing synthetic variables produced by the SynBoost classifier which was proposed by Zięba et al. (2016). The SynBoost classifier produces linear combinations of the original variables. This classifier was not available as a downloadable package in R or Rstudio. A script was written in R to implement the SynBoost algorithm. The SynBoost classifier was applied on each of the data sets to produce synthetic variables. From these generated synthetic variables, new synthetic data sets were created. The various classifiers were trained on the new synthetic data sets and their performance compared to the performance of the classifiers trained on the original

data.

Resampling the class distribution with techniques like SMOTE negatively impacted the classifiers performance. This may be due to some bias introduced by sampling procedures. The classifier with the highest AUC when trained on balanced data was the randomly undersampled RF with an area under the receiver operating characteristic curve (AUC) score of 0.7269. Classifiers trained on the synthetic features performed slightly worse than those trained on the original data. However, the decrease in performance was negligible as the highest AUC score in this study was 0.7732 and the highest AUC score when trained on synthetic features was 0.7729. The synthetic data sets have a considerably smaller number of dimensions when compared to the original data; eleven features compared to a total of twenty three. The synthetic features produced by SynBoost may be used as a dimensionality reduction technique that retains the performance of the original data better than PCA. The highest AUC score of classifiers trained on PCA data was 0.7390 which is noticeably smaller than the highest AUC score of 0.7729 when trained on synthetic features.

The SynBoost classifier is also a noticeably strong classifier, consistently featuring as one of the best performing classifiers for the balanced data sets. Overall the best performing classifier is XGBoost trained on the original data with tenfold cross-validation with an AUC score of 0.7732.

5.1 Limitations and Challenges of this Study

Some of the methods used in this study were extremely computationally expensive. Combining this with an unreliable power supply (loadshedding) some methods were run several times before a result could be recorded. Real credit risk data sets are very scarce thus research on this topic is limited.

5.2 Future Research

Future research can further investigate synthetic features. In this study, the SynBoost classifier has seven base learners, thus six synthetic data sets. Future research can investigate what happens if the number of synthetic data sets thus synthetic variables are increased. Future work may also investigate the structure of the data set using techniques like t-distributed stochastic neighbor embedding to visualize the data and degree of class-overlap. Thus also investigating techniques to deal with a data set that has a significant degree of class overlap. Better computational facilities may let one explore a wider array of hyperparameters, which might improve the classifiers performance. Furthermore different classification techniques can be explored such as CatBoost which is able to handle categorical variables unlike XGBoost and perhaps extending the current SynBoost method to handle categorical variables.

References

Abdi, H. and Williams, L. J. (2010). Principal Component Analysis. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(4):433–459.

Adnan, M., Alarood, A. A. S., Uddin, M. I., and Rehman, I. (2022). Utilizing Grid Search Cross-Validation with Adaptive Boosting for Augmenting Performance of Machine Learning Models. *PeerJ Computer Science*, 8:e803.

Anand, M., Velu, A., and Whig, P. (2022). Prediction of Loan Behaviour with Machine Learning Models for Secure Banking. *Journal of Computer Science and Engineering*, 3(1):1–13.

Baldi, P., Brunak, S., Chauvin, Y., Andersen, C. A., and Nielsen, H. (2000). Assessing the Accuracy of Prediction Algorithms for Classification: An Overview. *Bioinformatics*, 16(5):412–424.

Bayes, T. (1763). LII. An Essay Towards Solving A Problem in the Doctrine of Chances. By the Late Rev. Mr. Bayes, FRS Communicated by Mr. Price, in a Letter to John Canton. *Philosophical Transactions of the Royal Society of London*, (53):370–418.

Bekkar, M., Djemaa, H. K., and Alitouche, T. A. (2013). Evaluation Measures for Models Assessment over Imbalanced Data Sets. *Journal of Information Engineering and Applications*, 3(10).

Bentéjac, C., Csörgő, A., and Martínez-Muñoz, G. (2021). A Comparative Analysis of Gradient Boosting Algorithms. *Artificial Intelligence Review*, 54:1937–1967.

Bertsimas, D. and Dunn, J. (2017). Optimal Classification Trees. *Machine Learning*, 106(7):1039–1082.

Bhatore, S., Mohan, L., and Reddy, Y. R. (2020). Machine Learning Techniques for Credit Risk Evaluation: a Systematic Literature Review. *Journal of Banking and Financial Technology*, 4:111–138.

Biau, G. (2012). Analysis of a Random Forests Model. *The Journal of Machine Learning Research*, 13(1):1063–1095.

Bracke, P., Datta, A., Jung, C., and Sen, S. (2019). Machine Learning Explainability in Finance: an Application to Default Risk Analysis. Technical Report, Bank of England. `https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3435104` [Accessed 15/08/2023].

Breiman, L. (1996a). Arcing Classifiers. Technical Report, University of California, Department of Statistics. `https://www.stat.berkeley.edu/users/breiman/arc97.pdf` [Accessed 18/07/2022].

Breiman, L. (1996b). Bagging Predictors. *Machine Learning*, 24(2):123–140.

Breiman, L. (1996c). Out-of-Bag Estimation. Technical Report, University of California, Department of Statistics. `https://www.stat.berkeley.edu/pub/users/breiman/OOBestimation.pdf` [Accessed 04/09/2023].

Breiman, L. (2001). Random Forests. *Machine Learning*, 45:5–32.

Breiman, L., Friedman, J., Olshen, R., and Stone, C. (1984). *Classification and Regression Trees*. Routledge.

Buhlmann, P., Yu, B., et al. (2002). Analyzing Bagging. *Annals of Statistics*, 30(4):927–961.

Chang, Y.-C., Chang, K.-H., and Wu, G.-J. (2018). Application of eXtreme Gradient Boosting Trees in the Construction of Credit Risk Assessment Models for Financial Institutions. *Applied Soft Computing*, 73:914–920.

Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-Sampling Technique. *Journal of Artificial Intelligence Research*, 16:321–357.

Chawla, N. V., Japkowicz, N., and Kotcz, A. (2004). Special Issue on Learning from Imbalanced Data Sets. *Association for Computing Machinery's Special Interest Group on Knowledge Discovery and Data Mining Explorations Newsletter*, 6(1):1–6.

Chen, T. and Guestrin, C. (2016). Xgboost: A Scalable Tree Boosting System. In *Proceedings of the 22nd Assoication for Computing Machinery Special Interest Group on Knowledge Discovery in Data International Conference on Knowledge Discovery and Data Mining*, pages 785–794. Association for Computing Machinery.

Chen, T. and He, T. (2015). Higgs Boson Discovery with Boosted Trees. In Cowan, G., Germain, C., Guyon, I., Kegl, B., and Rousseau, D., editors, *Neural Information Processing Systems 2014 Workshop on High-Energy Physics and Machine Learning*, volume 42, pages 69–80. Proceedings of Machine Learning Research.

Chicco, D. and Jurman, G. (2020). The Advantages of the Matthews Correlation Coefficient (MCC) over F1 Score and Accuracy in Binary Classification Evaluation. *BioMed Central Genomics*, 21(1):1–13.

Chuang, C.-L. and Lin, R.-H. (2009). Constructing a Reassigning Credit Scoring Model. *Expert Systems with Applications*, 36(2):1685–1694.

Claesen, M. and De Moor, B. (2015). Hyperparameter Search in Machine Learning. In *Proceedings of the 11th Metaheuristics International Conference*, pages 1–5.

Deng, X., Liu, Q., Deng, Y., and Mahadevan, S. (2016). An Improved Method to Construct Basic Probability Assignment Based on the Confusion Matrix for Classification Problem. *Information Sciences*, 340:250–261.

Deutch, D., Gilad, A., Milo, T., and Somech, A. (2020). ExplainED: Explanations for EDA Notebooks. *Proceedings of the Very Large Data Base Endowmen*, 13(12):2917–2920.

DeYoung, R. and Rice, T. (2004). How Do Banks Make Money? The Fallacies of Fee Income. *Economic Perspectives, Federal Reserve Bank of Chicago*, 28(4):34–51.

Dwyer, K. and Holte, R. (2007). Decision Tree Instability and Active Learning. In *Machine Learning: ECML 2007: 18th European Conference on Machine Learning*, pages 128–139. Springer.

Edgar, A. (1935). The Irises of the Gaspe Peninsula. *Bulletin of the American Iris Society*, 59:2–5.

Efron, B. and Tibshirani, R. J. (1994). *An Introduction to The Bootstrap*. Chapman and Hall.

Fawcett, T. (2006). An Introduction to ROC Analysis. *Pattern Recognition Letters*, 27(8):861–874.

Fisher, R. A. (1936). The Use of Multiple Measurements in Taxonomic Problems. *Annals of Eugenics*, 7(2):179–188.

Fix, E. and Hodges, L. (1951). Discriminatory Analysis. Nonparametric Discrimination: Consistency Properties. *International Statistical Review/Revue Internationale de Statistique*, 57(3):238–247.

Freeman, E. A. and Moisen, G. G. (2008). A Comparison of the Performance of Threshold Criteria for Binary Classification in Terms of Predicted Prevalence and Kappa. *Ecological Modelling*, 217(1-2):48–58.

Freund, Y., Schapire, R., and Abe, N. (1999). A Short Introduction to Boosting. *Japanese Society for Artificial Intelligence*, 14(771-780):1612.

Freund, Y., Schapire, R. E., et al. (1996). Experiments with A New Boosting Algorithm. *International Conference on Machine Learning*, 96:148–156.

Friedman, J., Hastie, T., and Tibshirani, R. (2000). Additive Logistic Regression: A Statistical View of Boosting (With Discussion and a Rejoinder by the Authors). *The Annals of Statistics*, 28(2):337–407.

Friedman, J. H. (2001). Greedy Function Approximation: A Gradient Boosting Machine. *Annals of Statistics*, 29:1189–1232.

Ganganwar, V. (2012). An Overview of Classification Algorithms for Imbalanced Datasets. *International Journal of Emerging Technology and Advanced Engineering*, 2(4):42–47.

Ghojogh, B., Karray, F., and Crowley, M. (2023). Eigenvalue and Generalized Eigenvalue Problems: Tutorial. *arXiv preprint arXiv:1903.11240*.

Gurnỳ, P., Gurnỳ, M., et al. (2013). Comparison of Credit Scoring Models on Probability of Default Estimation for US Banks. *Prague Economic Papers*, 22(2):163–181.

Gutiérrez, P. D., Lastra, M., Bacardit, J., Benítez, J. M., and Herrera, F. (2016). GPU-SME-kNN: Scalable and Memory Efficient kNN and Lazy Learning Using GPUs. *Information Sciences*, 373:165–182.

Hair, J. F., Anderson, R. E., Tatham, R. L., and Black, W. C. (1984). Multivariate Data Analysis With Readings. *Tulsa, Oklahoma: Petroleum Publishing*.

Hamori, S., Kawai, M., Kume, T., Murakami, Y., and Watanabe, C. (2018). Ensemble Learning or Deep Learning? Application to Default Risk Analysis. *Journal of Risk and Financial Management*, 11(1):12.

Hastie, T., Tibshirani, R., Friedman, J. H., and Friedman, J. H. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, volume 2. Springer.

Hossin, M. and Sulaiman, M. N. (2015). A Review on Evaluation Metrics for Data Classification Evaluations. *International Journal of Data Mining & Knowledge Management Process*, 5(2):1–11.

Hotelling, H. (1933). Analysis of a Complex of Statistical Variables into Principal Components. *Journal of Educational Psychology*, 24(6):417.

Huang, J. and Ling, C. X. (2005). Using AUC and Accuracy in Evaluating Learning Algorithms. *Institute of Electrical and Electronics Engineers Transactions on Knowledge and Data Engineering*, 17(3):299–310.

Hunt, E. B., Marin, J., and Stone, P. J. (1966). *Experiments in Induction*. Academic Press.

Islam, S. R., Eberle, W., and Ghafoor, S. K. (2017). Mining Bad Credit Card Accounts from OLAP and OLTP. In *Proceedings of the International Conference on Compute and Data Analysis*, pages 129–137.

Islam, S. R., Eberle, W., and Ghafoor, S. K. (2018). Credit Default Mining Using Combined Machine Learning and Heuristic Approach. *arXiv preprint arXiv:1807.01176*.

James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An Introduction to Statistical Learning*, volume 112. Springer.

Jolliffe, I. T. and Cadima, J. (2016). Principal Component Analysis: A Review and Recent Developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150202.

Kataria, A. and Singh, M. (2013). A Review of Data Classification Using k-Nearest Neighbour Algorithm. *International Journal of Emerging Technology and Advanced Engineering*, 3(6):354–360.

Komorowski, M., Marshall, D. C., Salciccioli, J. D., and Crutain, Y. (2016). *Secondary Analysis of Electronic Health Records*. Springer.

Kotsiantis, S. B. (2013). Decision Trees: A Recent Overview. *Artificial Intelligence Review*, 39(4):261–283.

Krawczyk, B. (2016). Learning from Imbalanced Data: Open Challenges and Future Directions. *Progress in Artificial Intelligence*, 5(4):221–232.

Kruppa, J., Schwarz, A., Arminger, G., and Ziegler, A. (2013). Consumer Credit Risk: Individual Probability Estimates using Machine Learning. *Expert Systems with Applications*, 40(13):5125–5131.

Kubat, M., Matwin, S., et al. (1997). Addressing the Curse of Imbalanced Training Sets: One-Sided Selection. *International Conference on Machine Learning*, 97(1):179.

Laber, E., Molinaro, M., and Pereira, F. M. (2018). Binary Partitions with Approximate Minimum Impurity. In *International Conference on Machine Learning*, pages 2854–2862. Proceedings of Machine Learning Research.

Lemmens, A. and Croux, C. (2006). Bagging and Boosting Classification Trees to Predict Churn. *Journal of Marketing Research*, 43(2):276–286.

Leo, M., Sharma, S., and Maddulety, K. (2019). Machine Learning in Banking Risk Management: A Literature Review. *Risks*, 7(1):29.

Lever, J., Krzywinski, M., and Altman, N. (2016). Classification Evaluation. *Nature Methods*, 13(8):603–604.

Li, C., Chen, Z., Liu, J., Li, D., Gao, X., Di, F., Li, L., and Ji, X. (2019). Power Load Forecasting Based on the Combined Model of LSTM and XGBoost. In *Proceedings of the 2019 the International Conference on Pattern Recognition and Artificial Intelligence*, pages 46–51.

Li, K., Niskanen, J., Kolehmainen, M., and Niskanen, M. (2016). Financial Innovation: Credit Default Hybrid Model for SME Lending. *Expert Systems with Applications*, 61:343–355.

Liashchynskyi, P. and Liashchynskyi, P. (2019). Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS. *arXiv preprint arXiv:1912.06059*.

Lin, Z., Liu, R., and Su, Z. (2011). Linearized Alternating Direction Method with Adaptive Penalty for Low-Rank Representation. In Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., and Weinberger, K., editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc.

Ling, C. X., Huang, J., Zhang, H., et al. (2003). AUC: A Statistically Consistent and more Discriminating Measure than Accuracy. In *Proceedings of the 18th International Conference on Artificial Intelligence*, volume 3, pages 519–524.

Ling, C. X. and Li, C. (1998). Data Mining for Direct Marketing: Problems and Solutions. In *Knowledge Dsscovery and Data Mining*, volume 98, pages 73–79.

Lipton, Z. C., Elkan, C., and Naryanaswamy, B. (2014). Optimal Thresholding of Classifiers to Maximize F1 Measure. In *Machine Learning and Knowledge Discovery in Databases: European Conference, European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases 2014, Nancy, France, September 15-19, 2014. Proceedings, Part II 14*, pages 225–239. Springer.

Little, M. A., Varoquaux, G., Saeb, S., Lonini, L., Jayaraman, A., Mohr, D. C., and Kording, K. P. (2017). Using and Understanding Cross-Validation Strategies. Perspectives on Saeb. *GigaScience*, 6(5):1–6.

Liu, Y., Yang, M., Wang, Y., Li, Y., Xiong, T., and Li, A. (2022). Applying Machine Learning Algorithms to Predict Default Probability in the Online Credit Market: Evidence from China. *International Review of Financial Analysis*, 79:101971.

Loh, W.-Y. and Shih, Y.-S. (1997). Split Selection Methods for Classification Trees. *Statistica Sinica*, 7(4):815–840.

Maćkiewicz, A. and Ratajczak, W. (1993). Principal Components Analysis (PCA). *Computers & Geosciences*, 19(3):303–342.

Maclin, R. and Opitz, D. (1997). Empirical evaluation of bagging and boosting. In *Proceedings of the 1997 14th National Conference on Artificial Intelligence, AAAI 97*, pages 546–551. Association for the Advancement of Artificial Intelligence.

Maleki, M., Manshouri, N., and Kayikçioglu, T. (2017). A Novel Simple Method to Select Optimal k in k-Nearest Neighbor Classifier. *International Journal of Computer Science and Information Security*, 15(2):464.

Mandrekar, J. N. (2010). Receiver Operating Characteristic Curve in Diagnostic Test Assessment. *Journal of Thoracic Oncology*, 5(9):1315–1316.

Maria Navin, J. and Pankaja, R. (2016). Performance Analysis of Text Classification Algorithms using Confusion Matrix. *International Journal of Engineering and Technical Research*, 6(4):75–8.

Martin, J. K. (1995). An Exact Probability Metric for Decision Tree Splitting and Stopping. In *Pre-Proceedings of the Fifth International Workshop on Artificial Intelligence and Statistics*, pages 379–385. Proceedings of Machine Learning Research.

Matthews, B. W. (1975). Comparison of the Predicted and Observed Secondary Structure of T4 Phage Lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure*, 405(2):442–451.

Mert, A., Kılıç, N., and Akan, A. (2014). Evaluation of Bagging Ensemble Method with Time-Domain Feature Extraction for Diagnosing of Arrhythmia Beats. *Neural Computing and Applications*, 24:317–326.

Meza, J. C. (2010). Steepest Descent. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(6):719–722.

Mingers, J. (1989). An Empirical Comparison of Selection Measures for Decision-Tree Induction. *Machine Learning*, 3(4):319–342.

Moore IV, H. E., Andlauer, O., Simon, N., and Mignot, E. (2014). Exploring Medical Diagnostic Performance using Interactive, Multi-Parameter Sourced Receiver Operating Characteristic Scatter Plots. *Computers in Biology and Medicine*, 47:120–129.

Morgenthaler, S. (2009). Exploratory Data Analysis. *Wiley Interdisciplinary Reviews: Computational Statistics*, 1(1):33–44.

Moscatelli, M., Parlapiano, F., Narizzano, S., and Viggiano, G. (2020). Corporate Default Forecasting with Machine Learning. *Expert Systems with Applications*, 161:113567.

Moscato, V., Picariello, A., and Sperlí, G. (2021). A Benchmark of Machine Learning Approaches for Credit Score Prediction. *Expert Systems with Applications*, 165:113986.

Mucherino, A., Papajorgji, P., and Pardalos, P. M. (2009). *Data Mining in Agriculture*, volume 34. Springer Science & Business Media.

Natekin, A. and Knoll, A. (2013). Gradient Boosting Machines, A Tutorial. *Frontiers in Neurorobotics*, 7:21.

Nie, F., Hu, Z., and Li, X. (2018). An Investigation for Loss Functions Widely used in Machine Learning. *Communications in Information and Systems*, 18(1):37–52.

Opitz, D. and Maclin, R. (1999). Popular Ensemble Methods: An Empirical Study. *Journal of Artificial Intelligence Research*, 11:169–198.

Park, C. H. and Park, H. (2008). A Comparison of Generalized Linear Discriminant Analysis Algorithms. *Pattern Recognition*, 41(3):1083–1097.

Patel, H., Singh Rajput, D., Thippa Reddy, G., Iwendi, C., Kashif Bashir, A., and Jo, O. (2020). A Review on Classification of Imbalanced Data for Wireless Sensor Networks. *International Journal of Distributed Sensor Networks*, 16(4):1550147720916404.

Paul, L. C., Suman, A. A., and Sultan, N. (2013). Methodological Analysis of Principal Component Analysis (PCA) Method. *International Journal of Computational Engineering & Management*, 16(2):32–38.

Pearson, K. (1901). LIII. On Lines and Planes of Closest Fit to Systems of Points in Space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572.

Perner, P. (2011). How to Interpret Decision Trees? In Perner, P., editor, *Industrial Conference on Data Mining*, pages 40–55. Springer.

Peterson, L. E. (2009). k-Nearest Neighbor. *Scholarpedia*, 4(2):1883.

Powers, D. M. (2011). Evaluation: from Precision, Recall and F-measure to ROC, Informedness, Markedness and Correlation. *International Journal of Machine Learning Technologies*, 2:37–63.

Probst, P., Boulesteix, A.-L., and Bischl, B. (2019). Tunability: Importance of Hyperparameters of Machine Learning Algorithms. *The Journal of Machine Learning Research*, 20(1):1934–1965.

Raileanu, L. E. and Stoffel, K. (2004). Theoretical Comparison Between the Gini Index and Information Gain Criteria. *Annals of Mathematics and Artificial Intelligence*, 41(1):77–93.

Rao, C. R. (1948). The Utilization of Multiple Measurements in Problems of Biological Classification. *Journal of the Royal Statistical Society. Series B (Methodological)*, 10(2):159–203.

Rao, J. S. and Potts, W. J. (1997). Visualizing Bagged Decision Trees. In *Knowledge Discovery and Data Mining*, pages 243–246.

Rao, R. B., Fung, G., and Rosales, R. (2008). On the Dangers of Cross-Validation. An Experimental Evaluation. In *Proceedings of the 2008 Society for Industrial and Applied Mathematics International Conference on Data Mining*, pages 588–596. Society for Industrial and Applied Mathematics.

Ratner, B. (2009). The Correlation Coefficient: Its Values Range Between+ 1/- 1, or Do They? *Journal of Targeting, Measurement and Analysis For Marketing*, 17(2):139–142.

Samadi, S., Tantipongpipat, U., Morgenstern, J. H., Singh, M., and Vempala, S. (2018). The Price of Fair PCA: One Extra Dimension. *Advances in Neural Information Processing Systems*, 31:10999–11010.

Sasaki, Y. (2002). The Truth of the F-Measure, Manchester: MIB-School of Computer Science. `https://www.cs.odu.edu/~mukka/cs795sum11dm/Lecturenotes/Day3/F-measure-YS-26Oct07.pdf` [Accessed 01/06/2022].

Schapire, R. E. (2013). Explaining Adaboost. In "Schölkopf, B., editor, *Empirical Inference: Festschrift in Honor of Vladimir N. Vapnik*, pages 37–52. Springer.

Shalizi, C. (2016). Principal Component Analysis. `https://www.stat.cmu.edu/~cshalizi/uADA/12/lectures/ch18.pdf` [Accessed 05/21/2023].

Siaw, A., Ntiamoah, E., Oteng, E., and Opoku, B. (2014). An Empirical Analysis of the Loan Default Rate of Microfinance Institutions. *European Journal of Business and Management*, 6(22):12–17.

Šikić, Z. (1990). Taylor's Theorem. *International Journal of Mathematical Education in Science and Technology*, 21(1):111–115.

Singh, H. (2023). GSplitting Decision Trees with Gini Impurity. `https://www.analyticsvidhya.com/blog/2021/03/how-to-select-best-split-in-decision-trees-gini-impurity/` [Accessed 12/12/2023].

Skurichina, M. and Duin, R. P. (1998). Bagging for Linear Classifiers. *Pattern Recognition*, 31(7):909–930.

Smith, L. I. (2002). A Tutorial on Principal Components Analysis. `https://ourarchive.otago.ac.nz/bitstream/handle/10523/7534/OUCS-2002-12.pdf` [Accessed 20/05/2023].

Soper, D. S. (2021). Greed is Good: Rapid Hyperparameter Optimization and Model Selection using Greedy k-Fold Cross Validation. *Electronics*, 10(16):1973.

Sun, Y., Wong, A. K., and Kamel, M. S. (2009). Classification of Imbalanced Data: A Review. *International Journal of Pattern Recognition and Artificial Intelligence*, 23(04):687–719.

Tangirala, S. (2020). Evaluating the Impact of GINI Index and Information Gain on Classification using Decision Tree Classifier Algorithm. *International Journal of Advanced Computer Science and Applications*, 11(2):612–619.

Thabtah, F., Hammoud, S., Kamalov, F., and Gonsalves, A. (2020). Data Imbalance in Classification: Experimental Evaluation. *Information Sciences*, 513:429–441.

Tibrewal, K. (2018). Learning from Loss Functions: A Modified Algorithm for Classification Trees. Pomona College. `https://pages.pomona.edu/~jsh04747/Student%20Theses/KashviTibrewal18.pdf` [Accessed 10/11/2023].

Tomek, I. (1976). Two Modifications of CNN. *Institute of Electrical and Electronics Engineers Transactions on Systems, Man and Cybernetics*, 6:769–772.

Vakili, M., Ghamsari, M., and Rezaei, M. (2020). Performance Analysis and Comparison of Machine and Deep Learning Algorithms for IOT Data Classification. *arXiv preprint arXiv:2001.09636*.

Valle, S., Li, W., and Qin, S. J. (1999). Selection of the Number of Principal Components: The Variance of the Reconstruction Error Criterion with a Comparison to Other Methods. *Industrial & Engineering Chemistry Research*, 38(11):4389–4401.

Van Rijsbergen, C. J. (1986). A Non-Classical Logic for Information Retrieval. *The Computer Journal*, 29(6):481–485.

Wang, J., You, J., Li, Q., and Xu, Y. (2012). Extract Minimum Positive and Maximum Negative Features for Imbalanced Binary Classification. *Pattern Recognition*, 45(3):1136–1145.

Wang, S., Lu, J., Gu, X., Du, H., and Yang, J. (2016). Semi-Supervised Linear Discriminant Analysis for Dimension Reduction and Classification. *Pattern Recognition*, 57:179–189.

Weinberger, K. (2017). Machine Learning Lecture 30 Bagging -Cornell CS4780 SP17. `https://www.youtube.com/watch?v=0LB1cy2sCXc` [Accessed 04/09/2023].

Wiskott, L. (2016). Principal Component Analysis. `https://www.ini.rub.de/PEOPLE/wiskott/Teaching/Material/PrincipalComponentAnalysis-LectureNotesPublic.pdf` [Accessed 21/05/2023].

Xia, Y., He, L., Li, Y., Liu, N., and Ding, Y. (2020). Predicting Loan Default in Peer-to-Peer Lending using Narrative Data. *Journal of Forecasting*, 39(2):260–280.

Yeh, I.-C. and Lien, C.-h. (2009). The Comparisons of Data Mining Techniques for the Predictive Accuracy of Probability of Default of Credit Card Clients. *Expert Systems with Applications*, 36(2):2473–2480.

Yuan, Y.-x. (2006). A New Stepsize for the Steepest Descent Method. *Journal of Computational Mathematics*, 24:149–156.

Zavrel, J. (1997). An Empirical Re-Examination of Weighted Voting for kNN. In *Proceedings of the 7th Belgian-Dutch Conference on Machine Learning*, pages 139–148.

Zhou, J., Li, W., Wang, J., Ding, S., and Xia, C. (2019). Default Prediction In P2P Lending from High-Dimensional Data Based on Machine Learning. *Physica A: Statistical Mechanics and its Applications*, 534:122370.

Zhou, Z.-H. (2012). *Ensemble Methods: Foundations and Algorithms*. Chapman and Hall.

Zięba, M., Tomczak, S. K., and Tomczak, J. M. (2016). Ensemble Boosted Trees with Synthetic Features Generation in Application to Bankruptcy Prediction. *Expert Systems with Applications*, 58:93–101.

Appendix A: Tables

For each variable μ_1 denotes the population mean for the non-defaulters and μ_2 denotes the population mean for the defaulters. Similarly for each variable σ_1^2 denotes the population variance for the non-defaulters and σ_2^2 denotes the population variance for the defaulters. All tests were done at the five percent significance level that is $\alpha = 0.05$.

Table A.1: Hypothesis Tests for Categorical Data.

Variable	Test	Null Hypothesis	Test Statistic	P-value
Sex	Pearson's X^2 Test	Default is Independent of Sex	47.7090	0
Marriage	Pearson's X^2 Test	Default is Independent of Marriage	35.6620	0
Education	Fisher's Exact Test	Default is Independent of Education	NA	0.0004
Age	Fisher's Exact Test	Default is Independent of Age	NA	0.0004
Pay_0	Fisher's Exact Test	Default is Independent of Pay_0	NA	0.0004
Pay_2	Fisher's Exact Test	Default is Independent of Pay_2	NA	0.0004
Pay_3	Fisher's Exact Test	Default is Independent of Pay_3	NA	0.0004
Pay_4	Fisher's Exact Test	Default is Independent of Pay_4	NA	0.0004
Pay_5	Fisher's Exact Test	Default is Independent of Pay_5	NA	0.0004
Pay_6	Fisher's Exact Test	Default is Independent of Pay_6	NA	0.0004

Table A.2: Hypothesis Tests for Numerical Variables.

Variable	Test	Null Hypothesis	Test Statistic	P-value
Bill_amt1	Independent Samples T Test ($\sigma_1^2 = \sigma_2^2$)	$\mu_1 > \mu_2$	3.4030	0.9997
Bill_amt2	Independent Samples T Test ($\sigma_1^2 = \sigma_2^2$)	$\mu_1 > \mu_2$	2.4585	0.9930
Bill_amt3	Independent Samples T Test ($\sigma_1^2 = \sigma_2^2$)	$\mu_1 > \mu_2$	2.4381	0.9926
Bill_amt4	Independent Samples T Test ($\sigma_1^2 = \sigma_2^2$)	$\mu_1 > \mu_2$	1.7592	0.9607
Bill_amt5	Independent Samples T Test ($\sigma_1^2 = \sigma_2^2$)	$\mu_1 > \mu_2$	1.1709	0.8792
Bill_amt6	Independent Samples T Test ($\sigma_1^2 = \sigma_2^2$)	$\mu_1 > \mu_2$	0.9305	0.8239
Pay_amt1	Independent Samples T Test ($\sigma_1^2 \neq \sigma_2^2$)	$\mu_1 > \mu_2$	17.5130	1
Pay_amt2	Independent Samples T Test ($\sigma_1^2 \neq \sigma_2^2$)	$\mu_1 > \mu_2$	14.8170	1
Pay_amt3	Independent Samples T Test ($\sigma_1^2 \neq \sigma_2^2$)	$\mu_1 > \mu_2$	11.8930	1
Pay_amt4	Independent Samples T Test ($\sigma_1^2 \neq \sigma_2^2$)	$\mu_1 > \mu_2$	12.2220	1
Pay_amt5	Independent Samples T Test ($\sigma_1^2 \neq \sigma_2^2$)	$\mu_1 > \mu_2$	11.2460	1
Pay_amt6	Independent Samples T Test ($\sigma_1^2 \neq \sigma_2^2$)	$\mu_1 > \mu_2$	11.0580	1
Limit_bal	Independent Samples T Test ($\sigma_1^2 \neq \sigma_2^2$)	$\mu_1 > \mu_2$	28.9520	1

Table A.3: F-Test for Equal Variance.

Variable	Test	Null Hypothesis	Test Statistic	P-Value
Bill_amt1	F-test	$\sigma_1^2 = \sigma_2^2$	1.006	0.7750
Bill_amt2	F-test	$\sigma_1^2 = \sigma_2^2$	1.0176	0.3734
Bill_amt3	F-test	$\sigma_1^2 = \sigma_2^2$	0.9698	0.1206
Bill_amt4	F-test	$\sigma_1^2 = \sigma_2^2$	1.0008	0.9640
Bill_amt5	F-test	$\sigma_1^2 = \sigma_2^2$	1.0268	0.1763
Bill_amt6	F-test	$\sigma_1^2 = \sigma_2^2$	1.0011	0.9526
Pay_amt1	F-test	$\sigma_1^2 = \sigma_2^2$	0.2807	0
Pay_amt2	F-test	$\sigma_1^2 = \sigma_2^2$	0.2152	0
Pay_amt3	F-test	$\sigma_1^2 = \sigma_2^2$	0.4811	0
Pay_amt4	F-test	$\sigma_1^2 = \sigma_2^2$	0.4497	0
Pay_amt5	F-test	$\sigma_1^2 = \sigma_2^2$	0.5524	0
Pay_amt6	F-test	$\sigma_1^2 = \sigma_2^2$	0.5133	0
Limit_bal	F-test	$\sigma_1^2 = \sigma_2^2$	0.7683	0

Table A.4: KNN Optimal Hyperparameters.

Data Set	Optimal k	Accuracy
Original	20	0.8093
Randomly Oversampled	5	0.7405
Randomly Undersampled	40	0.6942
Smote	5	0.7474

Table A.5: CART Optimal Hyperparameters.

Data Set	Optimal Complexity Parameter	Accuracy
Original	0.17	0.8168
Randomly Oversampled	0.36	0.6819
Randomly Undersampled	0.001	0.6862
Smote	0.001	0.6875

Table A.6: RF Optimal Hyperparameters.

Data Set	Optimal mtry	Optimal Minimum Node Size	Accuracy
Original	3	10	0.8143
Randomly Oversampled	9	1	0.9412
Randomly Undersampled	3	10	0.7089
Smote	9	1	0.8637

Table A.7: AdaBoost Optimal Hyperparameters.

Data Set	Optimal mtry	Optimal Maximum Depth	Accuracy
Original	300	2	0.8186
Randomly Oversampled	300	5	0.7832
Randomly Undersampled	300	2	0.7107
Smote	300	5	0.8563

Table A.8: XGBoost Optimal Hyperparameters.

Data Set	Optimal Number of Trees	Optimal Depth	Learning Rate	Gamma	Accuracy
Original	150	1	0.01	0.1	0.8168
Randomly Oversampled	150	1	0.01	0.1	0.6838
Randomly Undersampled	150	1	0.01	0.1	0.6900
Smote	150	1	0.01	0.1	0.6872

Table A.9: Synthetic Training Data.

Variables
Limit_Bal
Pay_0
Pay_2
Pay_3
Pay_4
Pay_5
Pay_6
(Pay_0 + Pay_0)
(Pay_0 + Pay_0)
(Pay_0 + (Pay_0 + Pay_0))
(Pay_0 * (Pay_0 + Pay_0) 1)

Table A.10: Synthetic RU Training Data.

Variables
Limit_Bal
Pay_0
Pay_2
Pay_3
Pay_6
Bill_Amt1
Bill_Amt2
Pay_Amt1
Pay_Amt2
Pay_Amt3
Pay_Amt4
Pay_Amt6
(Pay_0 + Pay_0)
(Pay_0 + Pay_0)
(Pay_4 + Pay_0)
(Pay_4 + Pay_0)
(Limit_Bal * Pay_0)
(Limit_Bal - Pay_0)

Table A.11: Synthetic RO Training Data.

Variable
Pay_0
Pay_3
Pay_4
Limit_Bal
Bill_Amt1
Bill_Amt2
Bill_Amt3
Bill_Amt5
Pay_Amt1
Pay_Amt2
Pay_Amt3
Pay_Amt4
Pay_Amt5
Pay_Amt6
((Pay_0 + Pay_0) + Pay_2)
(Pay_0 + Pay_Amt3)
((Pay_0 ÷ Pay_Amt3) × Pay_Amt2)
((Pay_0 ÷ Pay_Amt3) - Pay_Amt2)
(((Pay_0 ÷ Pay_Amt3) - Pay_Amt2) + Pay_Amt3)
(Limit_Bal + Pay_Amt3)
(Limit_Bal × Pay_Amt3)

Table A.12: Synthetic SMOTED Training Data.

Variable
Age
Pay_0
Pay_5
Pay_6
Limit_Bal
Bill_Amt1
Pay_Amt1
Pay_Amt3
Pay_Amt5
Pay_Amt6
(Education - Pay_0)
(Pay_6 - Limit_Bal)
(Pay_3 - Pay_0)
(Pay_3 + Pay_0)
(Age + Pay_Amt3
(Age + Pay_Amt3

Table A.13: Synthetic PCA Training Data.

Variable
PC1
PC5
PC1 + PC1
(PC1 + (PC2 + PC2))
(PC1 + (PC2 + PC2))
(PC1 + (PC2 + PC2)) ÷ PC1
((PC1 + (PC2 + PC2)) + PC1)
(PC1 + (PC2 + PC2)) ÷((PC1 + (PC2 + PC2)) + PC1)
(PC1 + (PC2 + PC2)) ÷((PC1 + (PC2 + PC2)) + PC1)

Table A.14: Synthetic Training Data.

Classifier	MCC	Recall	Precision	AUC	F_1 Score
kNN	0.3975	0.3604	0.6718	0.7550	0.4691
LDA	0.3576	0.2688	0.7152	0.7686	0.3907
CART	0.3828	0.3215	0.6913	0.7613	0.4389
RF	0.6445	0.5381	0.9124	0.8969	0.6770
AdaBoost	0.3902	0.3504	0.6699	0.7530	0.4601
XGBoost	0.3828	0.3215	0.6913	0.7613	0.4389

Table A.15: RO Synthetic Training Data.

Classifier	MCC	Recall	Precision	AUC	F_1 Score
kNN	0.6498	0.9532	0.7423	0.8386	0.8347
LDA	0.3392	0.6360	0.6806	0.6700	0.6576
CART	0.3917	0.6285	0.7229	0.6976	0.6724
RF	0.9806	0.9987	0.9820	0.9903	0.9903
AdaBoost	0.6491	0.8206	0.8267	0.8246	0.8236
XGBoost	0.3996	0.4966	0.7964	0.7157	0.6118

Table A.16: RU Synthetic Training Data.

Classifier	MCC	Recall	Precision	AUC	F_1-Score
kNN	0.4297	0.6112	0.7599	0.7193	0.6775
LDA	0.3613	0.5915	0.7120	0.6835	0.6462
CART	0.4238	0.6242	0.7485	0.7151	0.6807
RF	0.9182	0.9551	0.9621	0.9591	0.9586
AdaBoost	0.4408	0.6486	0.7507	0.7227	0.6959
XGBoost	0.4119	0.5185	0.7937	0.7200	0.6272

Table A.17: SMOTED Synthetic Training Data.

Classifier	MCC	Recall	Precision	AUC	F_1-Score
kNN	0.6889	0.8174	0.8633	0.8450	0.8397
LDA	0.3281	0.6434	0.6709	0.6642	0.6568
CART	0.4199	0.5619	0.7797	0.7186	0.6531
RF	0.9980	0.9989	0.9991	0.9990	0.9990
AdaBoost	0.7868	0.8520	0.9264	0.8947	0.8876
XGBoost	0.4081	0.5476	0.7765	0.7135	0.6423

Table A.18: PCA Training Data with KCV.

Classifier	MCC	Recall	Precision	AUC	F_1Score
LDA	0.2976	0.2229	0.6586	0.7354	0.3308
kNN	0.3299	0.2820	0.6383	0.7304	0.3912
CART	0.3389	0.3551	0.5763	0.7048	0.4394
RF	0.8739	0.8191	0.9859	0.9682	0.8947
AdaBosst	0.3336	0.2943	0.6301	0.7273	0.4013
XGBoost	0.3371	0.3123	0.6158	0.7215	0.4145
SynBoost	0.3412	0.3072	0.6293	0.7280	0.4129

Table A.19: Synthetic PCA Training Data.A.14

Classifier	MCC	Recall	Precision	AUC	F_1-Score
kNN	0.3362	0.2987	0.6300	0.7276	0.4052
LDA	0.2607	0.1581	0.6909	0.7466	0.2573
CART	0.3277	0.2796	0.6370	0.7295	0.3887
RF	0.8665	0.8146	0.9784	0.9638	0.8890
AdaBoost	0.3300	0.2865	0.6329	0.7280	0.3944
XGBoost	0.3414	0.3083	0.6284	0.7277	0.4136

Appendix B: kNN

iris <- read.csv(“data.csv”)

ind <- sample(2, nrow(iris), replace=TRUE, prob=c(0.7, 0.3)) trainData <- iris[ind==1,] testData <- iris[ind==2,]

trainData1 <- trainData[-5] testData1 <- testData[-5]

dim(trainData)

dim(trainData1) dim(testData) dim(testData1)

iris_train_labels <- trainData$Species dim(iris_train_labels) class(iris_train_labels) iris_test_labels <- testData$Species dim(iris_test_labels)

library(class)

iris_test_pred1 <- knn(train = trainData1, test = testData1, cl= iris_train_labels,k = 3,prob=TRUE) iris_test_pred2 <- knn(train = trainData1, test = testData1, cl= iris_train_labels,k = 5,prob=TRUE)

library(gmodels)

CrossTable(x = iris_test_labels, y = iris_test_pred1,prop.chisq=FALSE)

CrossTable(x = iris_test_labels, y = iris_test_pred2,prop.chisq=FALSE)

Appendix C: SynBoost R Script

```
#Boosted model using CART
install.packages("MASS")
install.packages("ROCR")
install.packages("caret")
install.packages("pROC")
install.packages("randomForest")
install.packages("ada")
install.packages("adabag")
install.packages("ranger")
install.packages("ROSE")
install.packages("themis")
install.packages("mltools")
install.packages("corrr")
install.packages("ggcorrplot")
install.packages("factoextra")
install.packages("corrplot")
install.packages("ggplot2")
install.packages("plot3D")
install.packages("tidyverse")
install.packages("Rtsne")
install.packages("xgboost")
install.packages("DescTools")
install.packages("ECoL")
rm(list=ls())
library(MASS)
```

```
library(ROCR)
library(caret)
library(pROC)
library(randomForest)
library(ada)
library(adabag)
library(ranger)
library(ROSE)
library(themis)
library(mltools)
library(doParallel)
library(corrr)
library(ggcorrplot)
library(factoextra)
library(corrplot)
library(ggplot2)
library(plot3D)
library(tidyverse)
library(Rtsne)
library(ECoL)
library(xgboost)
library(DescTools)
cl <- parallel::detectCores() |>
parallel::makePSOCKcluster()
doParallel::registerDoParallel(cl, cores = length(cl))
Mode2 <- function(x, na.rm = FALSE) {
# it takes two areguments x, and na.rm
if(na.rm){ #if na.rm is false it means no need to remove NA values
x = x[!is.na(x)]
}
valx <- unique(x)
return(valx[which.max(tabulate(match(x, valx)))])
```

```
}
set.seed(376)
xgb_grid <- expand.grid(
nrounds = c(150,300),
eta=c(0.01,0.1),
gamma=c(0.1,0.3),
colsample_bytree = 0.75,
min_child_weight = 1,
subsample = 0.8,
max_depth = c(1,3)
)
#Ensemble method starts here
#################################################################################################
setwd("L:/Masters/DATA SET")
thedata1 <- read.csv("credit_proper.csv")
str(thedata1)
thedata1 <- thedata1[,-1]
colnames(thedata1) <- str_to_title(colnames(thedata1))
thedata1$Default <- as.factor(thedata1$Default)
sum(thedata1[,24] == 1)
sum(thedata1[,24] == 0)
#min(thedata1)
#max(thedata1$AGE)
# data is a tibble
# avoid the dplyr library
thedata1 <- data.frame(thedata1)
thedata_notarget <- thedata1[,-24]
#thedata_labels <- thedata1[,25]
#thedata_notarget_scale <- as.data.frame(scale(thedata_notarget,center = T, scale = T))
str(thedata1)
#thedata <- cbind(thedata_notarget_scale, thedata1[,24])
#colnames(thedata)[24] <- "Default"
```

```
thedata <- thedata1
thedata$Default <- factor(thedata$Default) #response into factor
str(thedata)
########################
#split data into 70% train & 30% test
########################
set.seed(123)
#thedata_num1 <- thedata[]
#scale the data
num_cat <- c("LIMIT_BAL", "BILL_AMT1", "BILL_AMT2", "BILL_AMT3","BILL_AMT4", "BILL_AMT5",
"BILL_AMT6", "PAY_AMT1", "PAY_AMT2","PAY_AMT3","PAY_AMT4","PAY_AMT5", "PAY_AMT6")
num_cat <- str_to_title(num_cat)
thedata_num <- thedata[,num_cat]
cat_cat <- c("SEX", "EDUCATION", "MARRIAGE", "AGE", "PAY_0", "PAY_2", "PAY_3", "PAY_4",
"PAY_5", "PAY_6")
cat_cat <- str_to_title(cat_cat)
thedata_cat <- thedata[,cat_cat]
process <- preProcess(thedata_num, method = c("range"))
thedata_num_scale <- predict(process,thedata_num)
Default <- thedata[,c("Default")]
thedata <- cbind(thedata_cat, thedata_num_scale, Default)
#colnames(thedata) <- "Default"
str(thedata)
#thedata$Default <- as.factor(thedata$Default - 1)
# Create a vector that has 70% of the row names, randomly sampled
set.seed(376)
thedt = sort(sample(nrow(thedata), nrow(thedata)*.7))
# create test and training data
thetrain <- thedata[thedt,]
thetest <- thedata[-thedt,]
thetrain_SMT <- thedata1[-thedt,]
thetrain_SMT$Default <- as.factor(thetrain_SMT$Default)
str(thetrain) # 21000 obs
```

```
str(thetest) # 9000 obs
sum(thetrain[,24] == 1)
sum(thetrain[,24] == 0)
sum(thetest[,24] == 1)
sum(thetest[,24] == 0)
thetrain$Default <- as.factor(thetrain$Default)
thetest$Default <- as.factor(thetest$Default)
thetrain
# Remove response from test & train
names(thedata)
thetraindata_notarget <- thetrain[,-24]
thetestdata_notarget <- thetest[,-24]
thetrain_data_notarget2 <- dplyr::mutate(thetraindata_notarget, ID = row_number())
#cl arguments (class labes)
train_labels <- thetrain[,24]
test_labels <- thetest[,24]
#OVERSAMPLING THE TRAINING data
#thetrain_under <- as.data.frame(ovun.sample(Default~., thetrain, method = "over", seed = 376)$data)
#thetraindata_notarget_under <- thetrain_under[,-24]
train_labels <- thetrain[,24]
thetrain <- thetrain[,-24]
thetest <- thetest[,-24]
#total num of base learners
num_base <- 7
#store base learners here
base_learners <- list()
#index of removed features
removed_features <- list()
train_vars <- list()
#keep track of added features
#Specify size of Dnew
num_newft <- 2
```

```
anx <- 0
added_features <- as.data.frame(matrix(NA, ncol = 3,nrow = 1))
total_preds2 <- as.data.frame(matrix(NA, ncol = num_base,nrow = length(train_labels)))
ctrl <- trainControl(method="cv", number = 10)
for(hk in 1:num_base){
print("Base learner: ")
print(hk)
# thetrain <- xgb.DMatrix(data = as.matrix(thetrain), label= as.numeric(train_labels)-1 )
#features frequency used to sample
distr_df <- data.frame()
#For reporducibility
#xgb_model <- xtrain(data=thetrain, nrounds = 500,
# params = parmx)
str(thetrain)
#train_vars <- append(train_vars,as.character(colnames(thetrain)))
set.seed(376)
xgb_tr <- cbind(thetrain,train_labels)
colnames(xgb_tr)[dim(xgb_tr)[2]] <- "Default"
xgb_model <- train(Default~., data=xgb_tr, method="xgbTree", tuneLength=1, tuneGrid = xgb_grid,
trControl = ctrl)
print("Done training")
total_preds2[,hk] <- predict(xgb_model, thetrain)
#store base learner for testing
base_learners[[hk]] <- xgb_model
##############################################################################################
#Feature importance
imp_ft <- varImp(xgb_model, scale = T)
#number of features
num_ft <- nrow(thetrain)
#All feature names
ft_names <- names(thetrain)
# print(dim(ft_names))
# print(str(ft_names))
```

```
# ft_names <- lapply(ft_names, as.character)
#names of features
#names of importnant features
#imp_names <- as.data.frame(imp_ft[,1])
#convert to dataframe & char
#imp_names <- sapply(imp_names, as.character)
imp_df <- varImp(xgb_model, scale = F)[["importance"]]
imp_df <- t(as.data.frame(imp_df))
names_df <- colnames(imp_df)
imp_df <- as.data.frame(unlist(unlist(imp_df)))
names_df <- gsub("‘", "", names_df)
colnames(imp_df) <- names_df
new_order <- colnames(thetrain)
#new_order <- new_order[-c("Default")]
imp_df <- imp_df[,new_order]
#For each feature find the frequency in the ensemble
# for (i in 1:num_ft)
# {
# poss_name <- ft_names[i]
# print(poss_name)
#if name is in list assign it its value
distr_df <- imp_df
#}
#find the index of features below certain index to be removed
index_removed_vars <- which(distr_df[1,] < 0.019)
############################################################
#set frequency to 0 Such that these are not sampled from later
distr_df[distr_df < 0.019] <- 0
#find the index of the feature that is to be sampled
f1_index <- sample(x=seq(1:dim(distr_df)[2] ),1, prob = c(distr_df[1,]), replace = T)
f2_index <- sample(x=seq(1:dim(distr_df)[2] ),1, prob = c(distr_df[1,]), replace = T)
#f1_index
```

```
#f2_index
#Select feature using the matrix
f1 <- thetrain[,f1_index]
f2 <- thetrain[,f2_index]
#find the names of sampled features
f1_name <- colnames(thetrain)[f1_index]
f2_name <- colnames(thetrain)[f2_index]
#################
#Remove vars which fall short of frequency threshold
if(length(index_removed_vars) != 0){
thetrain <- subset(thetrain, select = -c(index_removed_vars))
}
index_removed_vars <- t(as.data.frame(index_removed_vars))
#store each removed var
for(ind in 1:ncol(index_removed_vars)){
removed_features <- append(removed_features,index_removed_vars[1,ind])
}
#an indicator to mark when new data set starts
removed_features <- append(removed_features, -1)
# anx <- 0
for(j in 1:num_newft){
#Randomly select an operation
set.seed(NULL)
oper_df <- c("mult", "add", "sub", "div")
oper <- oper_df[round(runif(1,1,4))]
if( oper == "mult")
{
fnew <- f1*f2
f_oper <- paste("(", f1_name, " * ", f2_name, ")", as.character(j))
} else if (oper == "add")
{
fnew <- f1+f2
```

```
f_oper <- paste("(", f1_name, " + ", f2_name, ")", as.character(j))
} else if (oper == "sub")
{
fnew <- f1-f2
f_oper <- paste("(", f1_name, " - ", f2_name, ")", as.character(j))
} else{
f3 <- f2
f3[f3 == 0] <- 1
fnew <- f1/f3
f_oper <- paste("(", f1_name, " / ", f2_name, ")", as.character(j))
rm(f3)
}
#add new coloumn to the training data
thetrain <- as.data.frame(cbind(thetrain, as.data.frame(fnew)))
ft_num <- ncol(thetrain)
#name the new variable with operation that took place
colnames(thetrain)[ft_num] <- f_oper
#keep track of operation that took place
#For this example only one feature is added for each ensemble
#added_features[hk+j-1,1] <- f1_index
#added_features[hk+j-1,2] <- oper
#added_features[hk+j-1,3] <- f2_index
added_features <- rbind(added_features, t(as.data.frame(c(f1_index,oper,f2_index))))
}
}
#added_features <- na.omit(added_features)
#added_features <- added_features[]
#TEST IS DONE HERE
added_features <- added_features[2:dim(added_features)[1],]
added_features376 <- added_features
#added_fe
#test_data <- thedata[,-24] #thetrain1
```

```
total_preds <- as.data.frame(matrix(NA, ncol = num_base,nrow = length(test_labels)))
#removed_features <- t(as.data.frame(unlist(removed_features)))
# Finding test preds is done here
for(hk in 1:num_base){
print("Base learner: ")
print(hk)
colnames(thetest)
model_xgb <- base_learners[[hk]]
#thetrain <- xgb.DMatrix(data=as.matrix(thetrain), label= as.numeric(train_labels)-1 )
#thetest <- xgb.DMatrix(data = as.matrix(test_data), label= as.numeric(train_labels)-1 )
total_preds[,hk] <- predict(model_xgb,thetest)
#total_preds2[,hk] <- (as.data.frame(round(predict(model_xgb, xgb.DMatrix(as.matrix(test_data))))))
print("DONE")
#This appends new rows to dataframe
for(j in 1:num_newft){
#thetrainnf <- thetrain
f1 <- thetest[,as.integer(added_features[j,1])]
f2 <- thetest[,as.integer(added_features[j,3])]
oper <- added_features[j,2]
f1_name <- colnames(thetest)[as.integer(added_features[j,1])]
f2_name <- colnames(thetest)[as.integer(added_features[j,3])]
if( oper == "mult")
{
fnew <- f1*f2
f_oper <- paste("(", f1_name, " * ", f2_name, ")", as.character(j))
} else if (oper == "add")
{
fnew <- f1+f2
f_oper <- paste("(", f1_name, " + ", f2_name, ")", as.character(j))
} else if (oper == "sub")
{
fnew <- f1-f2
```

```
f_oper <- paste("(", f1_name, " - ", f2_name, ")", as.character(j))
} else{
f3 <- f2
f3[f3 == 0] <- 1
fnew <- f1/f3
f_oper <- paste("(", f1_name, " / ", f2_name, ")", as.character(j))
rm(f3)
}
thetest <- as.data.frame(cbind(thetest, as.data.frame(fnew)))
ft_num <- ncol(thetest)
#name the new variable with operation that took place
colnames(thetest)[ft_num] <- f_oper
}
added_features <- added_features[-seq(1:num_newft),]
cnt <- 1
cols_dltd <- data.frame()
rem_conc <- c()
rem_conc2 <- c()
rem_index <- 0
#This removes excess rows
while(rem_index != -1){
#rem_index <- rem_index + 1
rem_index <- as.numeric(unlist(removed_features[cnt]))
if(rem_index != -1){
if(rem_index != 0)
{
rem_conc <- c(rem_conc, rem_index)
}
}
rem_conc2 <- c(rem_conc2, cnt)
# col <- col + 1
#rem_conc <- c(rem_conc, cnt)
```

```
cnt <- cnt+1
}
#rem_conc2 <- c(rem_conc2, cnt)
removed_features[rem_conc2] <- NULL
thetest <- thetest[, -rem_conc]
}
#MAJORITY VOTING IS DONE HERE
#FINAL PRED is done here
############################################################
final_pred <- as.data.frame(matrix(NA, ncol = 1,nrow = dim(total_preds)[1]))
final_pred2 <- as.data.frame(matrix(NA, ncol = 1,nrow = dim(total_preds2)[1]))
for(i in 1:dim(total_preds)[1]){
final_pred[i,1] <- Mode2(as.numeric(total_preds[i,])-1)
}
for(i in 1:dim(total_preds2)[1]){
final_pred2[i,1] <- Mode2(as.numeric(total_preds2[i,])-1)
}
#train
confusionMatrix(as.factor(final_pred2$V1), train_labels,positive = "1", mode="everything")
auc(as.factor(final_pred2$V1), as.numeric(train_labels))
mcc(as.factor(final_pred2$V1), train_labels)
#test
confusionMatrix(as.factor(final_pred$V1), test_labels, positive = "1", mode="everything")
auc(as.factor(final_pred$V1), as.numeric(test_labels))
mcc(as.factor(final_pred$V1), test_labels)
```

Appendix D: Implementation of Traditional Classifiers

```
########################
## LDA Classifier: Credit data set
## Author: Anele
## June 2022
## This script implements the LDA classifier on the credit Default data set
########################
rm(list=ls())
install.packages("MASS")
install.packages("ROCR")
install.packages("caret")
install.packages("pROC")
install.packages("randomForest")
install.packages("ada")
install.packages("adabag")
install.packages("ranger")
install.packages("ROSE")
install.packages("themis")
install.packages("mltools")
install.packages("corrr")
install.packages("ggcorrplot")
install.packages("factoextra")
install.packages("corrplot")
install.packages("ggplot2")
```

```
install.packages("plot3D")
install.packages("tidyverse")
install.packages("Rtsne")
install.packages("xgboost")
install.packages("DescTools")
install.packages("ECoL")
########################
## Read in the data
########################
#Implement LDA
library(MASS)
library(ROCR)
library(caret)
library(pROC)
library(randomForest)
library(ada)
library(adabag)
library(ranger)
library(ROSE)
library(themis)
library(mltools)
library(doParallel)
library(corrr)
library(ggcorrplot)
library(factoextra)
library(corrplot)
library(ggplot2)
library(plot3D)
library(tidyverse)
library(Rtsne)
library(ECoL)
library(xgboost)
```

```
library(DescTools)
cl <- parallel::detectCores() |>
parallel::makePSOCKcluster()
doParallel::registerDoParallel(cl, cores = length(cl))
setwd("L:/Masters/DATA SET")
thedata1 <- read.csv("credit_proper.csv")
str(thedata1)
thedata1 <- thedata1[,-1]
colnames(thedata1) <- str_to_title(colnames(thedata1))
#r_val <- overlapping(thedata1,thedata1$Default, measures = "all")
thedata1$Default <- as.factor(thedata1$Default)
#apply(thedata1,2,min)
#apply(thedata1,2,max)
#apply(thedata1,2,boxplot, outline=FALSE)
#hist(as.factor(thedata$Default))
#fviz_eig(thedata1, addlabels = TRUE)
#fviz_pca_var(thedata1, col.var = "black")
sum(thedata1[,24] == 1)
sum(thedata1[,24] == 0)
#min(thedata1)
#max(thedata1$AGE)
# data is a tibble
# avoid the dplyr library
thedata1 <- data.frame(thedata1)
thedata_notarget <- thedata1[,-24]
#thedata_labels <- thedata1[,25]
#thedata_notarget_scale <- as.data.frame(scale(thedata_notarget,center = T, scale = T))
str(thedata1)
#thedata <- cbind(thedata_notarget_scale, thedata1[,24])
#colnames(thedata)[24] <- "Default"
thedata <- thedata1
thedata$Default <- factor(thedata$Default) #response into factor
```

```
#thedata$SEX <- as.factor(thedata1$SEX)
#thedata$EDUCATION <- as.factor(thedata1$EDUCATION)
#thedata$MARRIAGE <- as.factor(thedata1$MARRIAGE)
#thedata$PAY_0 <- as.factor(thedata$PAY_0)
#thedata$PAY_2 <- as.factor(thedata$PAY_2)
#thedata$PAY_3 <- as.factor(thedata$PAY_3)
#thedata$PAY_4 <- as.factor(thedata$PAY_4)
#thedata$PAY_5 <- as.factor(thedata$PAY_5)
#thedata$PAY_6 <- as.factor(thedata$PAY_6)
str(thedata)
########################
#split data into 70% train & 30% test
########################
set.seed(123)
#thedata_num1 <- thedata[]
#scale the data
num_cat <- c("LIMIT_BAL", "BILL_AMT1", "BILL_AMT2", "BILL_AMT3","BILL_AMT4", "BILL_AMT5",
"BILL_AMT6", "PAY_AMT1", "PAY_AMT2","PAY_AMT3","PAY_AMT4","PAY_AMT5", "PAY_AMT6")
num_cat <- str_to_title(num_cat)
thedata_num <- thedata[,num_cat]
cat_cat <- c("SEX", "EDUCATION", "MARRIAGE", "AGE", "PAY_0", "PAY_2", "PAY_3", "PAY_4",
"PAY_5", "PAY_6")
cat_cat <- str_to_title(cat_cat)
thedata_cat <- thedata[,cat_cat]
process <- preProcess(thedata_num, method = c("range"))
thedata_num_scale <- predict(process,thedata_num)
Default <- thedata[,c("Default")]
thedata <- cbind(thedata_cat, thedata_num_scale, Default)
#colnames(thedata) <- "Default"
str(thedata)
#thedata$Default <- as.factor(thedata$Default - 1)
# Create a vector that has 70% of the row names, randomly sampled
set.seed(376)
```

```
thedt = sort(sample(nrow(thedata), nrow(thedata)*.7))
# create test and training data
thetrain <- thedata[thedt,]
thetest <- thedata[-thedt,]
thetrain_SMT <- thedata1[-thedt,]
thetrain_SMT$Default <- as.factor(thetrain_SMT$Default)
str(thetrain) # 21000 obs
str(thetest) # 9000 obs
sum(thetrain[,24] == 1)
sum(thetrain[,24] == 0)
sum(thetest[,24] == 1)
sum(thetest[,24] == 0)
thetrain$Default <- as.factor(thetrain$Default)
thetest$Default <- as.factor(thetest$Default)
thetrain
# Remove response from test & train
names(thedata)
thetraindata_notarget <- thetrain[,-24]
thetestdata_notarget <- thetest[,-24]
thetrain_data_notarget2 <- dplyr::mutate(thetraindata_notarget, ID = row_number())
#cl arguments (class labes)
train_labels <- thetrain[,24]
test_labels <- thetest[,24]
#OVERSAMPLING THE TRAINING data
thetrain_over <- as.data.frame(ovun.sample(Default~., thetrain, method = "over", seed = 376)$data)
thetrain_under <- as.data.frame(ovun.sample(Default~., thetrain, method = "under", seed = 376)$data)
thetrain_smote <- smotenc(thetrain, "Default", k=5)
thetraindata_notarget_over <- thetrain_over[,-24]
train_labels_over <- thetrain_over[,24]
thetraindata_notarget_under <- thetrain_under[,-24]
train_labels_under <- thetrain_under[,24]
thetraindata_notarget_smote <- thetrain_smote[,-24]
```

```
train_labels_smote <- thetrain_smote[,24]
################################### PCA ########################
cov_mat <- cor(thetrain[,c(1:23)])
eigen_inf <- eigen(cov_mat)
eigen_vals <- eigen_inf$values
mean_val <- mean(eigen_vals)
eigen_vecs <- eigen_inf$vectors
eigen_ret <- as.data.frame(eigen_vecs[,1:5])
train_pca_notarg <- as.data.frame(lapply(thetrain[,c(1:23)], as.numeric))
train_pca <- as.matrix(train_pca_notarg) %*% as.matrix(eigen_ret)
train_pca1 <- as.data.frame(train_pca)
train_pca_notarg <- train_pca1
test_pca_notarg <- as.data.frame(lapply(thetest[,c(1:23)], as.numeric))
test_pca1 <- as.matrix(test_pca_notarg) %*% as.matrix(eigen_ret)
test_pca1 <- as.data.frame(test_pca1)
test_pca_notarg <- test_pca1
train_pca <- cbind(train_pca1,thetrain$Default)
test_pca <- cbind(test_pca1,thetest$Default)
colnames(train_pca)[dim(train_pca)[2]] <- "Default"
colnames(test_pca)[dim(test_pca)[2]] <- "Default"
#ggcorrplot(corr_mat)
#data.pca <- princomp(corr_mat)
#summary(data.pca)
#data.pca$loadings[,1:3]
#fviz_eig(data.pca, addlabels = TRUE)
#fviz_pca_var(data.pca, col.var = "black")
#pca_kept <- get_pca_var(data.pca)
#pca_anal <- prcomp(thetrain[,c(1:23)], scale = TRUE)
#pca_data <- pca_anal$x
#pca_test <- prcomp(thetest[,c(1:23)], scale = TRUE)$x
#pca_test_kept <- as.data.frame(pca_test[,1:3])
#fviz_eig(pca_anal, addlabels = TRUE)
```

```
#fviz_pca_var(pca_anal, col.var = "black")
#pca_kept <- as.data.frame(pca_data[,1:3])
#pca_train <- cbind(pca_kept,thetrain$Default)
#colnames(pca_train)[4] <- "Default"
#pca_train_illus <- as.data.frame(ovun.sample(Default~., pca_train, method = "under", seed = 376)$data)
#ggplot(pca_train_illus, aes(x=PC1, y = PC2, color = Default)) + geom_point()
#ggplot(thedata1, aes(x = AGE, y = PAY_6, color = Default)) + geom_point() + ggtitle("Payment History vs Age") + theme(plot.title = element_text(hjust = 0.5)) + scale_color_manual(values=c("blue", "red"))
#LDA
################################################################################
ctrl = trainControl(method = "cv", number = 10)
set.seed(376)
lda.fit <- train(Default ~., data=thetrain, method = "lda", tuneLength = 1, trControl = ctrl)
lda.fit
lda.fit$results
#ctrl = trainControl(method = "cv", number = 10)
#set.seed(376)
#lda.fit2 <- train(Default ~., data=thetrain_smote2, method = "lda", tuneLength = 1, trControl = ctrl)
#lda.fit2
set.seed(376)
lda.fit_over <- train(Default ~., data=thetrain_over, method = "lda", tuneLength = 1, trControl = ctrl)
lda.fit_over
lda.fit_over$results
set.seed(376)
lda.fit_under <- train(Default ~., data=thetrain_under, method = "lda", tuneLength = 1, trControl = ctrl)
lda.fit_under
lda.fit_under$results
set.seed(376)
lda.fit_smote <- train(Default ~., data=thetrain_smote, method = "lda", tuneLength = 1, trControl = ctrl)
lda.fit_smote
```

```
lda.fit_smote$results
set.seed(376)
lda.fit_pca <- train(Default ~., data=train_pca, method = "lda", tuneLength = 1, trControl = ctrl)
lda.fit_pca
lda.fit_pca$results
lda_pred_train <- predict(lda.fit,thetraindata_notarget, trControl = trainControl(method = "cv"))
confusionMatrix(lda_pred_train, train_labels, positive = "1", mode="everything")
mcc(lda_pred_train, train_labels)
auc(lda_pred_train, as.numeric(train_labels))
########TRAIN
#over
lda_pred_train_over <- predict(lda.fit_over,thetraindata_notarget_over)
confusionMatrix(lda_pred_train_over, train_labels_over, positive = "1", mode="everything")
mcc(lda_pred_train_over, train_labels_over)
auc(lda_pred_train_over, as.numeric(train_labels_over))
#under
lda_pred_train_under <- predict(lda.fit_under,thetraindata_notarget_under)
confusionMatrix(lda_pred_train_under, train_labels_under, positive = "1", mode="everything")
mcc(lda_pred_train_under, train_labels_under)
auc(lda_pred_train_under, as.numeric(train_labels_under))
#smote
lda_pred_train_smote <- predict(lda.fit_smote,thetraindata_notarget_smote)
confusionMatrix(lda_pred_train_smote, train_labels_smote, positive = "1", mode="everything")
mcc(lda_pred_train_smote, train_labels_smote)
auc(lda_pred_train_smote, as.numeric(train_labels_smote))
#pca
lda_pred_train_pca <- predict(lda.fit_pca,train_pca_notarg)
confusionMatrix(lda_pred_train_pca, train_labels, positive = "1", mode="everything")
mcc(lda_pred_train_pca, train_labels)
auc(lda_pred_train_pca, as.numeric(train_labels))
########TEST
lda_pred_test <- predict(lda.fit,thetestdata_notarget)
```

```
confusionMatrix(lda_pred_test, test_labels, positive = "1", mode="everything")
lda_roc <- roc(lda_pred_test, as.numeric(test_labels))
plot(lda_roc)
auc(lda_pred_test, as.numeric(test_labels))
mcc(lda_pred_test, test_labels)
#lda_pred_test2 <- predict(lda.fit2,thetest2)
#confusionMatrix(lda_pred_test2, test_labels, positive = "1", mode="everything")
#lda_roc2 <- roc(lda_pred_test2, as.numeric(test_labels))
#plot(lda_roc2)
#auc(lda_pred_test2, as.numeric(test_labels))
#mcc(lda_pred_test2, test_labels)
#over
lda_pred_test_over <- predict(lda.fit_over,thetestdata_notarget)
confusionMatrix(lda_pred_test_over, test_labels, positive = "1", mode="everything")
lda_roc_over <- roc(lda_pred_test_over, as.numeric(test_labels))
plot(lda_roc_over)
auc(lda_pred_test_over, as.numeric(test_labels))
mcc(lda_pred_test_over, test_labels)
#under
lda_pred_test_under <- predict(lda.fit_under,thetestdata_notarget)
confusionMatrix(lda_pred_test_under, test_labels, positive = "1", mode="everything")
lda_roc_under <- roc(lda_pred_test_under, as.numeric(test_labels))
plot(lda_roc_under)
auc(lda_pred_test_under, as.numeric(test_labels))
mcc(lda_pred_test_under, test_labels)
#smote
lda_pred_test_smote <- predict(lda.fit_smote,thetestdata_notarget)
confusionMatrix(lda_pred_test_smote, test_labels, positive = "1", mode="everything")
lda_roc_smote <- roc(lda_pred_test_smote, as.numeric(test_labels))
plot(lda_roc_smote)
auc(lda_pred_test_smote, as.numeric(test_labels))
mcc(lda_pred_test_smote, test_labels)
```

```
lda_pred_test_pca <- predict(lda.fit_pca,test_pca_notarg)
confusionMatrix(lda_pred_test_pca, test_labels, positive = "1", mode="everything")
lda_roc_pca <- roc(lda_pred_test_pca, as.numeric(test_labels))
plot(lda_roc_pca)
auc(lda_pred_test_pca, as.numeric(test_labels))
mcc(lda_pred_test_pca, test_labels)
#############################################################################################
#KNN
#############################################################################################
#train
ctrl <- trainControl(method="cv",number = 10)
set.seed(376)
ptm <- proc.time()
knn.fit <- train(Default ~., data=thetrain, method = "knn",
tuneGrid = data.frame(k = c(5,20,40,60,170)),
tuneLength = 1, trControl = ctrl)
proc.time() - ptm
knn.fit
set.seed(376)
knn.fit_over <- train(Default ~., data=thetrain_over, method = "knn",
tuneGrid = data.frame(k = c(5,20,40,60,170)),
tuneLength = 1, trControl = ctrl)
knn.fit_over
set.seed(376)
knn.fit_under <- train(Default ~., data=thetrain_under, method = "knn",
tuneGrid = data.frame(k = c(5,20,40,60,170)),
tuneLength = 1, trControl = ctrl)
knn.fit_under
set.seed(376)
knn.fit_smote <- train(Default ~., data=thetrain_smote, method = "knn",
tuneGrid = data.frame(k = c(5,20,40,60,170)),
tuneLength = 1, trControl = ctrl)
```

```
knn.fit_smote
####
set.seed(376)
knn.fit_pca <- train(Default ~., data=train_pca, method = "knn",
tuneGrid = data.frame(k = c(5,20,40,60,170)),
tuneLength = 1, trControl = ctrl)
knn.fit_pca
####
#test
knn_pred_train <- predict(knn.fit,thetraindata_notarget)
confusionMatrix(knn_pred_train, train_labels, positive = "1", mode="everything")
mcc(knn_pred_train, train_labels)
auc(knn_pred_train, as.numeric(train_labels))
#over
knn_pred_train_over <- predict(knn.fit_over,thetraindata_notarget_over)
confusionMatrix(knn_pred_train_over, train_labels_over, positive = "1", mode="everything")
mcc(knn_pred_train_over, train_labels_over)
auc(knn_pred_train_over, as.numeric(train_labels_over))
#under
knn_pred_train_under <- predict(knn.fit_under,thetraindata_notarget_under)
confusionMatrix(knn_pred_train_under, train_labels_under, positive = "1", mode="everything")
mcc(knn_pred_train_under, train_labels_under)
auc(knn_pred_train_under, as.numeric(train_labels_under))
#smote
knn_pred_train_smote <- predict(knn.fit_smote,thetraindata_notarget_smote)
confusionMatrix(knn_pred_train_smote, train_labels_smote, positive = "1", mode="everything")
mcc(knn_pred_train_smote, train_labels_smote)
auc(knn_pred_train_smote, as.numeric(train_labels_smote))
#pca
knn_pred_train_pca <- predict(knn.fit_pca,train_pca_notarg)
confusionMatrix(knn_pred_train_pca, train_labels, positive = "1", mode="everything")
mcc(knn_pred_train_pca, train_labels)
```

```
auc(knn_pred_train_pca, as.numeric(train_labels))
#######test
knn_pred_test <- predict(knn.fit,thetestdata_notarget)
confusionMatrix(knn_pred_test, test_labels, positive = "1", mode="everything")
knn_roc <- roc(knn_pred_test, as.numeric(test_labels))
plot(knn_roc)
auc(knn_pred_test, as.numeric(test_labels))
mcc(knn_pred_test, test_labels)
#over
knn_pred_test_over <- predict(knn.fit_over,thetestdata_notarget)
confusionMatrix(knn_pred_test_over, test_labels, positive = "1", mode="everything")
knn_roc_over <- roc(knn_pred_test_over, as.numeric(test_labels))
plot(knn_roc_over)
auc(knn_pred_test_over, as.numeric(test_labels))
mcc(knn_pred_test_over, test_labels)
#under
knn_pred_test_under <- predict(knn.fit_under,thetestdata_notarget)
confusionMatrix(knn_pred_test_under, test_labels, positive = "1", mode="everything")
knn_roc_under <- roc(knn_pred_test_under, as.numeric(test_labels))
plot(knn_roc_under)
auc(knn_pred_test_under, as.numeric(test_labels))
mcc(knn_pred_test_under, test_labels)
#smote
knn_pred_test_smote <- predict(knn.fit_smote,thetestdata_notarget)
confusionMatrix(knn_pred_test_smote, test_labels, positive = "1", mode="everything")
knn_roc_smote <- roc(knn_pred_test_smote, as.numeric(test_labels))
plot(knn_roc_smote)
auc(knn_pred_test_smote, as.numeric(test_labels))
mcc(knn_pred_test_smote, test_labels)
#pca
knn_pred_test_pca <- predict(knn.fit_pca,test_pca_notarg)
confusionMatrix(knn_pred_test_pca, test_labels, positive = "1", mode="everything")
```

```
knn_roc_pca <- roc(knn_pred_test_pca, as.numeric(test_labels))
plot(knn_roc_pca)
auc(knn_pred_test_pca, as.numeric(test_labels))
mcc(knn_pred_test_pca, test_labels)
##############################################################################################
##############################################################################################
#CART
##############################################################################################
#ctrl <- trainControl(method="repeatedcv",repeats = 3, )
modelLookup("rpart")
ctrl <- trainControl(method = "cv", number = 10)
set.seed(376)
cart.fit <- train(Default ~., data=thetrain, method = "rpart",
tuneGrid = expand.grid(cp = seq(0.01, 0.5, 0.01))
,trControl = ctrl)
cart.fit
#set.seed(376)
#cart.fit2 <- train(Default ~., data=thetrain2, method = "rpart",
# trControl = ctrl, tuneGrid = expand.grid(cp = seq(0.1, 0.5, 0.001)))
#cart.fit <- train(Default ~., data=thetrain, method = "rpart", tuneGrid = expand.grid(cp = seq(0.01,
0.5, 0.01)))
set.seed(376)
cart.fit_over <- train(Default ~., data=thetrain_over, method = "rpart",
tuneGrid = expand.grid(cp = seq(0.01, 0.5, 0.01))
,trControl = ctrl)
cart.fit_over
set.seed(376)
cart.fit_under <- train(Default ~., data=thetrain_under, method = "rpart",
tuneGrid = expand.grid(cp = seq(0.01, 0.5, 0.01))
,trControl = ctrl)
cart.fit_under
set.seed(376)
cart.fit_smote <- train(Default ~., data=thetrain_smote, method = "rpart",
```

```
tuneGrid = expand.grid(cp = seq(0.01, 0.5, 0.01))
,trControl = ctrl)
cart.fit_smote
set.seed(376)
cart.fit_pca <- train(Default ~., data=train_pca, method = "rpart",
tuneGrid = expand.grid(cp = seq(0.01, 0.5, 0.01))
,trControl = ctrl)
cart.fit_pca
##########train
cart_pred_train <- predict(cart.fit,thetraindata_notarget)
confusionMatrix(cart_pred_train, train_labels, positive = "1", mode="everything")
mcc(cart_pred_train, train_labels)
auc(cart_pred_train, as.numeric(train_labels))
cart_pred_train_over <- predict(cart.fit_over,thetraindata_notarget_over)
confusionMatrix(cart_pred_train_over, train_labels_over, positive = "1", mode="everything")
mcc(cart_pred_train_over, train_labels_over)
auc(cart_pred_train_over, as.numeric(train_labels_over))
cart_pred_train_under <- predict(cart.fit_under,thetraindata_notarget_under)
confusionMatrix(cart_pred_train_under, train_labels_under, positive = "1", mode="everything")
mcc(cart_pred_train_under, train_labels_under)
auc(cart_pred_train_under, as.numeric(train_labels_under))
cart_pred_train_smote <- predict(cart.fit_smote,thetraindata_notarget_smote)
confusionMatrix(cart_pred_train_smote, train_labels_smote, positive = "1", mode="everything")
mcc(cart_pred_train_smote, train_labels_smote)
auc(cart_pred_train_smote, as.numeric(train_labels_smote))
cart_pred_train_pca <- predict(cart.fit_pca,train_pca_notarg)
confusionMatrix(cart_pred_train_pca, train_labels, positive = "1", mode="everything")
mcc(cart_pred_train_pca, train_labels)
auc(cart_pred_train_pca, as.numeric(train_labels))
########test
cart_pred_test <- predict(cart.fit,thetestdata_notarget)
confusionMatrix(cart_pred_test, test_labels, positive = "1", mode="everything")
```

```
cart_roc <- roc(cart_pred_test, as.numeric(test_labels))
plot(cart_roc)
auc(cart_pred_test, as.numeric(test_labels))
mcc(cart_pred_test, test_labels)
#cart_pred_test2 <- predict(cart.fit2,thetest2)
#confusionMatrix(cart_pred_test2, test_labels, positive = "1", mode="everything")
#cart_roc2 <- roc(cart_pred_test2, as.numeric(test_labels))
#plot(cart_roc2)
#auc(cart_pred_test2, as.numeric(test_labels))
#mcc(cart_pred_test2, test_labels)
cart_pred_test_over <- predict(cart.fit_over,thetestdata_notarget)
confusionMatrix(cart_pred_test_over, test_labels, positive = "1", mode="everything")
cart_roc_over <- roc(cart_pred_test_over, as.numeric(test_labels))
plot(cart_roc_over)
auc(cart_pred_test_over, as.numeric(test_labels))
mcc(cart_pred_test_over, test_labels)
cart_pred_test_under <- predict(cart.fit_under,thetestdata_notarget)
confusionMatrix(cart_pred_test_under, test_labels, positive = "1", mode="everything")
cart_roc_under <- roc(cart_pred_test_under, as.numeric(test_labels))
plot(cart_roc_under)
auc(cart_pred_test_under, as.numeric(test_labels))
mcc(cart_pred_test_under, test_labels)
cart_pred_test_smote <- predict(cart.fit_smote,thetestdata_notarget)
confusionMatrix(cart_pred_test_smote, test_labels, positive = "1", mode="everything")
cart_roc_smote <- roc(cart_pred_test_smote, as.numeric(test_labels))
plot(cart_roc_smote)
auc(cart_pred_test_smote, as.numeric(test_labels))
mcc(cart_pred_test_smote, test_labels)
cart_pred_test_pca <- predict(cart.fit_pca,test_pca_notarg)
confusionMatrix(cart_pred_test_pca, test_labels, positive = "1", mode="everything")
cart_roc_pca <- roc(cart_pred_test_pca, as.numeric(test_labels))
plot(cart_roc_pca)
```

```
auc(cart_pred_test_pca, as.numeric(test_labels))
mcc(cart_pred_test_pca, test_labels)
#############################################################################################
#RF
#############################################################################################
ctrl <- trainControl(method="cv", number = 10)
#rf.fit <- train(Default ~., data=thetrain, method = "rf", trControl = ctrl_cart)
rfGrid <- expand.grid(mtry = c(3,5,9), splitrule = "gini", min.node.size=c(1,5,10))
set.seed(376)
#modelLookup("rf")
#rf.fit <- randomForest(Default~., thetrain)
set.seed(376)
rf.fit <- train(Default ~., data=thetrain, method = "ranger", tuneLength = 1,
num.trees=150, tuneGrid = rfGrid, trControl = ctrl)
rf.fit
set.seed(376)
rf.fit_over <- train(Default ~., data=thetrain_over, method = "ranger", tuneLength = 1,
num.trees=150, tuneGrid = rfGrid, trControl = ctrl)
rf.fit_over
set.seed(376)
rf.fit_under <- train(Default ~., data=thetrain_under, method = "ranger",
num.trees=150, tuneGrid = rfGrid, trControl = ctrl)
rf.fit_under
ptm <- proc.time()
set.seed(376)
rf.fit_smote <- train(Default ~., data=thetrain_smote, method = "ranger",
num.trees=150, tuneGrid = rfGrid, trControl = ctrl)
rf.fit_smote
proc.time() - ptm
set.seed(376)
rf.fit_pca <- train(Default ~., data=train_pca, method = "ranger",
num.trees=150, tuneGrid = rfGrid, trControl = ctrl)
```

```
rf.fit_pca
###############
rf_pred_train <- predict(rf.fit,thetraindata_notarget)
confusionMatrix(rf_pred_train, train_labels, positive = "1", mode="everything")
mcc(rf_pred_train, train_labels)
auc(rf_pred_train, as.numeric(train_labels))
rf_pred_train_over <- predict(rf.fit_over,thetraindata_notarget_over)
confusionMatrix(rf_pred_train_over, train_labels_over, positive = "1", mode="everything")
mcc(rf_pred_train_over, train_labels_over)
auc(rf_pred_train_over, as.numeric(train_labels_over))
rf_pred_train_under <- predict(rf.fit_under,thetraindata_notarget_under)
confusionMatrix(rf_pred_train_under, train_labels_under, positive = "1", mode="everything")
mcc(rf_pred_train_under, train_labels_under)
auc(rf_pred_train_under, as.numeric(train_labels_under))
rf_pred_train_smote <- predict(rf.fit_smote,thetraindata_notarget_smote)
confusionMatrix(rf_pred_train_smote, train_labels_smote, positive = "1", mode="everything")
mcc(rf_pred_train_smote, train_labels_smote)
auc(rf_pred_train_smote, as.numeric(train_labels_smote))
rf_pred_train_pca <- predict(rf.fit_pca,train_pca_notarg)
confusionMatrix(rf_pred_train_pca, train_labels, positive = "1", mode="everything")
mcc(rf_pred_train_pca, train_labels)
auc(rf_pred_train_pca, as.numeric(train_labels))
#########################test
rf_pred_test <- predict(rf.fit,thetestdata_notarget)
confusionMatrix(rf_pred_test, test_labels, positive = "1", mode="everything")
rf_roc <- roc(rf_pred_test, as.numeric(test_labels))
plot(rf_roc)
auc(rf_pred_test, as.numeric(test_labels))
mcc(rf_pred_test, test_labels)
rf_pred_test_over <- predict(rf.fit_over,thetestdata_notarget)
confusionMatrix(rf_pred_test_over, test_labels, positive = "1", mode="everything")
rf_roc_over <- roc(rf_pred_test_over, as.numeric(test_labels))
```

```
plot(rf_roc_over)
auc(rf_pred_test_over, as.numeric(test_labels))
mcc(rf_pred_test_over, test_labels)
rf_pred_test_under <- predict(rf.fit_under,thetestdata_notarget)
confusionMatrix(rf_pred_test_under, test_labels, positive = "1", mode="everything")
rf_roc_under <- roc(rf_pred_test_under, as.numeric(test_labels))
plot(rf_roc_under)
auc(rf_pred_test_under, as.numeric(test_labels))
mcc(rf_pred_test_under, test_labels)
rf_pred_test_smote <- predict(rf.fit_smote,thetestdata_notarget)
confusionMatrix(rf_pred_test_smote, test_labels, positive = "1", mode="everything")
rf_roc_smote <- roc(rf_pred_test_smote, as.numeric(test_labels))
plot(rf_roc_smote)
auc(rf_pred_test_smote, as.numeric(test_labels))
mcc(rf_pred_test_smote, test_labels)
rf_pred_test_pca <- predict(rf.fit_pca,test_pca_notarg)
confusionMatrix(rf_pred_test_pca, test_labels, positive = "1", mode="everything")
rf_roc_pca <- roc(rf_pred_test_pca, as.numeric(test_labels))
plot(rf_roc_pca)
auc(rf_pred_test_pca, as.numeric(test_labels))
mcc(rf_pred_test_pca, test_labels)
##############################################################################################
#RF
#ADABOOST
##############################################################################
ctrl <- trainControl(method="cv", number = 10)
ada_grid <- expand.grid(mfinal = c(100,300), coeflearn = "Freund", maxdepth = c(1,2,5))
#rf.fit <- train(Default ~., data=thetrain, method = "rf", trControl = ctrl_cart)
#adb.fit <- ada(Default~., thetrain )
#adb.fit <- boosting(Default~., data=thetrain)
set.seed(376)
adb.fit <- train(Default~., data=thetrain, method = "AdaBoost.M1", tuneLength = 1,
```

```
tuneGrid = ada_grid, par = TRUE, trControl = ctrl)
set.seed(376)
adb.fit_over <- train(Default~., data=thetrain_over, method = "AdaBoost.M1",
tuneLength = 1, tuneGrid = ada_grid, par = TRUE, trControl = ctrl)
set.seed(376)
adb.fit_under <- train(Default~., data=thetrain_under, method = "AdaBoost.M1", tuneLength = 1,
tuneGrid = ada_grid, par = TRUE, trControl = ctrl)
set.seed(376)
adb.fit_smote <- train(Default~., data=thetrain_smote, method = "AdaBoost.M1", tuneLength = 1,
tuneGrid = ada_grid, par = TRUE, trControl = ctrl)
set.seed(376)
adb.fit_pca <- train(Default~., data=train_pca, method = "AdaBoost.M1", tuneLength = 1,
tuneGrid = ada_grid, par = TRUE, trControl = ctrl)
adb_pred_train <- predict(adb.fit,thetraindata_notarget)
confusionMatrix(adb_pred_train, train_labels, positive = "1", mode="everything")
auc(adb_pred_train, as.numeric(train_labels))
mcc(adb_pred_train, train_labels)
adb_pred_train_over <- predict(adb.fit_over,thetraindata_notarget_over)
confusionMatrix(adb_pred_train_over, train_labels_over, positive = "1", mode="everything")
auc(adb_pred_train_over, as.numeric(train_labels_over))
mcc(adb_pred_train_over, train_labels_over)
adb_pred_train_under <- predict(adb.fit_under,thetraindata_notarget_under)
confusionMatrix(adb_pred_train_under, train_labels_under, positive = "1", mode="everything")
auc(adb_pred_train_under, as.numeric(train_labels_under))
mcc(adb_pred_train_under, train_labels_under)
adb_pred_train_smote <- predict(adb.fit_smote,thetraindata_notarget_smote)
confusionMatrix(adb_pred_train_smote, train_labels_smote, positive = "1", mode="everything")
auc(adb_pred_train_smote, as.numeric(train_labels_smote))
mcc(adb_pred_train_smote, train_labels_smote)
adb_pred_train_pca <- predict(adb.fit_pca,train_pca_notarg)
confusionMatrix(adb_pred_train_pca, train_labels, positive = "1", mode="everything")
auc(adb_pred_train_pca, as.numeric(train_labels))
```

```
mcc(adb_pred_train_pca, train_labels)
############################test
adb_pred_test <- predict(adb.fit,thetestdata_notarget)
confusionMatrix(adb_pred_test, test_labels, positive = "1", mode="everything")
adb_roc <- roc(adb_pred_test, as.numeric(test_labels))
#plot(adb_roc)
auc(adb_pred_test, as.numeric(test_labels))
mcc(adb_pred_test, test_labels)
adb_pred_test_over <- predict(adb.fit_over,thetestdata_notarget)
confusionMatrix(adb_pred_test_over, test_labels, positive = "1", mode="everything")
adb_roc_over <- roc(adb_pred_test_over, as.numeric(test_labels))
plot(adb_roc_over)
auc(adb_pred_test_over, as.numeric(test_labels))
mcc(adb_pred_test_over, test_labels)
adb_pred_test_under <- predict(adb.fit_under,thetestdata_notarget)
confusionMatrix(adb_pred_test_under, test_labels, positive = "1", mode="everything")
adb_roc_under <- roc(adb_pred_test_under, as.numeric(test_labels))
plot(adb_roc_under)
auc(adb_pred_test_under, as.numeric(test_labels))
mcc(adb_pred_test_under, test_labels)
adb_pred_test_smote <- predict(adb.fit_smote,thetestdata_notarget)
confusionMatrix(adb_pred_test_smote, test_labels, positive = "1", mode="everything")
adb_roc_smote <- roc(adb_pred_test_smote, as.numeric(test_labels))
plot(adb_roc_smote)
auc(adb_pred_test_smote, as.numeric(test_labels))
mcc(adb_pred_test_smote, test_labels)
adb_pred_test_pca <- predict(adb.fit_pca,test_pca)
confusionMatrix(adb_pred_test_pca, test_labels, positive = "1", mode="everything")
adb_roc_pca <- roc(adb_pred_test_pca, as.numeric(test_labels))
plot(adb_roc_pca)
auc(adb_pred_test_pca, as.numeric(test_labels))
mcc(adb_pred_test_pca, test_labels)
```

```
#XGBOOST
#############################################################################################
ctrl <- trainControl(method="cv", number = 1)
#thetrain_xgb <- thetrain
#thetrain_xgb$Default <- as.numeric(thetrain_xgb$Default)-1
set.seed(376)
xgb_grid <- expand.grid(
nrounds = c(150,300),
eta=c(0.01,0.1),
gamma=c(0.1,0.3),
colsample_bytree = 0.75,
min_child_weight = 1,
subsample = 0.8,
max_depth = c(1,3)
)
#xgb.fit <- train(Default~., data=thetrain, method = "xgbTree", tuneLength = 1,
# tuneGrid = xgb_grid, par = TRUE, trControl = ctrl)
set.seed(376)
xgb.fit <- train(Default~., data=thetrain, method="xgbTree", trControl = ctrl, tuneLength=1, tuneGrid
= xgb_grid)
set.seed(376)
xgb.fit_under <- train(Default~., data=thetrain_under, method="xgbTree", trControl = ctrl, tuneLength=1,
tuneGrid = xgb_grid)
set.seed(376)
xgb.fit_over <- train(Default~., data=thetrain_over, method="xgbTree", trControl = ctrl, tuneLength=1,
tuneGrid = xgb_grid)
set.seed(376)
xgb.fit_smote <- train(Default~., data=thetrain_smote, method="xgbTree", trControl = ctrl, tuneLength=1,
tuneGrid = xgb_grid)
set.seed(376)
xgb.fit_pca <- train(Default~., data=train_pca, method="xgbTree", trControl = ctrl, tuneLength=1,
tuneGrid = xgb_grid)
xgb_pred_train <- predict(xgb.fit,thetraindata_notarget)
```

```
confusionMatrix(xgb_pred_train, train_labels, positive = "1", mode="everything")
auc(xgb_pred_train, as.numeric(train_labels))
mcc(xgb_pred_train, train_labels)
xgb_pred_train_under <- predict(xgb.fit_under,thetraindata_notarget_under)
confusionMatrix(xgb_pred_train_under, train_labels_under, positive = "1", mode="everything")
auc(xgb_pred_train_under, as.numeric(train_labels_under))
mcc(xgb_pred_train_under, train_labels_under)
xgb_pred_train_over <- predict(xgb.fit_over,thetraindata_notarget_over)
confusionMatrix(xgb_pred_train_over, train_labels_over, positive = "1", mode="everything")
auc(xgb_pred_train_over, as.numeric(train_labels_over))
mcc(xgb_pred_train_over, train_labels_over)
xgb_pred_train_smote <- predict(xgb.fit_smote,thetraindata_notarget_smote)
confusionMatrix(xgb_pred_train_smote, train_labels_smote, positive = "1", mode="everything")
auc(xgb_pred_train_smote, as.numeric(train_labels_smote))
mcc(xgb_pred_train_smote, train_labels_smote)
xgb_pred_train_pca <- predict(xgb.fit_pca,train_pca_notarg)
confusionMatrix(xgb_pred_train_pca, train_labels, positive = "1", mode="everything")
auc(xgb_pred_train_pca, as.numeric(train_labels))
mcc(xgb_pred_train_pca, train_labels)
##############################################################################################
xgb_pred_test <- predict(xgb.fit,thetestdata_notarget)
confusionMatrix(xgb_pred_test, test_labels, positive = "1", mode="everything")
xgb_roc <- roc(xgb_pred_test, as.numeric(test_labels))
#plot(adb_roc)
auc(xgb_pred_test, as.numeric(test_labels))
mcc(xgb_pred_test, test_labels)
xgb_pred_test_under <- predict(xgb.fit_under,thetestdata_notarget)
confusionMatrix(xgb_pred_test_under, test_labels, positive = "1", mode="everything")
xgb_roc <- roc(xgb_pred_test_under, as.numeric(test_labels))
#plot(adb_roc)
auc(xgb_pred_test_under, as.numeric(test_labels))
mcc(xgb_pred_test_under, test_labels)
```

```
xgb_pred_test_over <- predict(xgb.fit_over,thetestdata_notarget)
confusionMatrix(xgb_pred_test_over, test_labels, positive = "1", mode="everything")
xgb_roc <- roc(xgb_pred_test_over, as.numeric(test_labels))
#plot(adb_roc)
auc(xgb_pred_test_over, as.numeric(test_labels))
mcc(xgb_pred_test_over, test_labels)
xgb_pred_test_smote <- predict(xgb.fit_smote,thetestdata_notarget)
confusionMatrix(xgb_pred_test_smote, test_labels, positive = "1", mode="everything")
xgb_roc <- roc(xgb_pred_test_smote, as.numeric(test_labels))
#plot(adb_roc)
auc(xgb_pred_test_smote, as.numeric(test_labels))
mcc(xgb_pred_test_smote, test_labels)
xgb_pred_test_pca <- predict(xgb.fit_pca,test_pca)
confusionMatrix(xgb_pred_test_pca, test_labels, positive = "1", mode="everything")
xgb_roc <- roc(xgb_pred_test_pca, as.numeric(test_labels))
#plot(adb_roc)
auc(xgb_pred_test_pca, as.numeric(test_labels))
mcc(xgb_pred_test_pca, test_labels)
#ada_grid <- expand.grid(mfinal = c(100,300), coeflearn = "Freund", maxdepth = c(1,2,5))
#xgb_thetrain <- xgb.DMatrix(data = as.matrix(thetrain[,-24]), label= as.numeric(thetrain$Default)-1
)
#xgb_model <- xgb.train(data=xgb_thetrain, nrounds = 500,
# params = parmx)
#xgb_model_cv <- xgb.cv(data=xgb_thetrain, nrounds = 300, params=parmx, nfold=5)
#xgb_preds_train <- as.data.frame(round(predict(xgb_model, xgb.DMatrix(as.matrix(as.matrix(thetrain[,-
24])))))))
#xgb_pred_train <- as.factor(as.matrix(xgb_preds_train))
#confusionMatrix(xgb_pred_train, train_labels, positive = "1", mode="everything")
#xgb_roc <- roc(xgb_pred_train, as.numeric(train_labels))
#plot(adb_roc)
#auc(xgb_pred_train, as.numeric(train_labels))
#mcc(xgb_pred_train, train_labels)
xgb_preds <- as.data.frame(round(predict(xgb_model, xgb.DMatrix(as.matrix(as.matrix(thetest[,-24])))))))
```

```
xgb_pred_test <- as.factor(as.matrix(xgb_preds))
confusionMatrix(xgb_pred_test, test_labels, positive = "1", mode="everything")
adb_roc <- roc(xgb_pred_test, as.numeric(test_labels))
#plot(adb_roc)
auc(xgb_pred_test, as.numeric(test_labels))
mcc(xgb_pred_test, test_labels)
set.seed(376)
adb.fit <- train(Default~., data=thetrain, method = "AdaBoost.M1", tuneLength = 1,
tuneGrid = ada_grid, par = TRUE, trControl = ctrl)
#################################################################################################
lda.fit$resample
lda.fit$results
lda.fit_over$resample
lda.fit_over$results
lda.fit_under$resample
lda.fit_under$results
lda.fit_smote$resample
lda.fit_smote$results
knn.fit$resample
knn.fit$results
knn.fit_over$resample
knn.fit_over$results
knn.fit_under$resample
knn.fit_under$results
knn.fit_smote$resample
knn.fit_smote$results
cart.fit$resample
cart.fit$results
cart.fit_over$resample
cart.fit_over$results
cart.fit_under$resample
cart.fit_under$results
```

```
cart.fit_smote$resample
cart.fit_smote$results
rf.fit$resample
rf.fit$results
rf.fit_over$resample
rf.fit_over$results
rf.fit_under$resample
rf.fit_under$results
rf.fit_smote$resample
rf.fit_smote$results
adb.fit$resample
adb.fit$results
adb.fit_over$resample
adb.fit_over$results
adb.fit_under$resample
adb.fit_under$results
adb.fit_smote$resample
adb.fit_smote$results
#rm(list = ls())
```