Statistical and Mathematical Learning: an Application to Fraud Detection and Prevention

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

MASTER OF SCIENCE

in the

DEPARTMENT OF STATISTICS
RHODES UNIVERSITY

by

Sisipho Hamlomo

December 2021

Abstract

Credit card fraud is an ever-growing problem. There has been a rapid increase in the rate of fraudulent activities in recent years resulting in a considerable loss to several organizations, companies, and government agencies. Many researchers have focused on detecting fraudulent behaviours early using advanced machine learning techniques. However, credit card fraud detection is not a straightforward task since fraudulent behaviours usually differ for each attempt and the dataset is highly imbalanced, that is, the frequency of non-fraudulent cases outnumbers the frequency of fraudulent cases. In the case of the European credit card dataset, we have a ratio of approximately one fraudulent case to five hundred and seventy-eight non-fraudulent cases. Different methods were implemented to overcome this problem, namely random undersampling, one-sided sampling, SMOTE combined with Tomek links and parameter tuning. Predictive classifiers, namely logistic regression, decision trees, k-nearest neighbour, support vector machine and multilayer perceptrons, are applied to predict if a transaction is fraudulent or non-fraudulent. The model's performance is evaluated based on recall, precision, F_1-score, the area under receiver operating characteristics curve, geometric mean and Matthew correlation coefficient. The results showed that the logistic regression classifier performed better than other classifiers except when the dataset was oversampled.

Keywords: Financial fraud, Imbalanced dataset, Bootstrap, Cross validation, Support vector machine, k-nearest neighbour, Decision trees, Logistic regression, Multilayer perceptron.

Contents

List of Tables

List of Figures

List of Abbreviations

ABC: Artificial bee colony

AE: Auto-encoder.

ANN: Artificial neural network.

AUC: Area under ROC curve.

BBN: Bayesian belief network.

CCW: Class confidence weighted.

CE: Cross entropy.

CNN: Convolutional neural network.

CNP: Card not present.

CP: Card present.

CV: Cross validation.

DBN: Deep belief neural network.

DT: Decision tree.

ENRFE: Enhanced recursive feature elimination.

ERM: Empirical risk minimization.

FFD Financial fraud detection.

FNR: False negative rate.

FPR: False positive rate.

G-mean: Geometry mean.

GS: Grid search.

HDDT: Hellinger distance decision tree.

iid: Independent and identically distributed.

kNN: k-nearest neighbour.

LOF: Local outlier factor.

LR: Logistic regression.

LOOCV: Leave one out cross validation.

MCC: Matthews correlation coefficient.

MLE: Maximum likelihood estimate.

MLP: Multilayer perceptron

PCA: Principle component analysis.

PR: Precision and recall curve.

QDA: Quadratic discriminant analysis.

RBM: Restricted Boltzmann machine.

RF: Random forest.

RFE: Recursive feature elimination.

RO: Random oversampling.

ROC: Receiver operating curve.

RS: Random search.

RU: Random undersampling.

SMOTE: Synthetic minority oversampling technique.

TPR: True positive rate.

t-SNE: t-distributed stochastic neighbour embedding.

Acknowledgements

I would like to acknowledge and give my warmest thanks to my supervisor Mr J Baxter and co-supervisor, Dr M Atemkeng, who made this work possible. Their support, patience, and advice throughout this challenging project have been invaluable. I would also like to thank Stones Dalitso Chindipha for taking his time to proofread this work.

Special thanks to Aliziwe Mzolwa and my family as a whole for their continuous support and love.

Lastly, I would like to acknowledge the financial assistance of the National Research Foundation (NRF) towards this research. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to NRF.

Chapter 1

Introduction

1.1 Financial Fraud Detection

Financial fraud detection (FFD) is vital to preventing the devastating consequences of financial fraud. FFD involves distinguishing fraudulent financial transactions from non-fraudulent transactions, revealing fraudulent behaviour or activities and allowing decision-makers to develop appropriate strategies to decrease the impact of fraud. Data mining plays a vital role in FFD as it is widely used to extract and uncover hidden patterns in large amounts of data (Ngai et al., 2011). Technological advancement and the internet boom has caused an increase in fraudulent schemes in the business world. Some of these commonly observed schemes include credit card fraud, financial statement fraud, e-commerce transaction fraud, insurance fraud, money laundering and telecommunications fraud (Hu et al., 2011). The use of statistical and machine learning based technologies has been shown to be an effective method in detecting fraud (Zhou & Kapoor, 2011). However, fraudsters are adaptive and are usually able to find ways of bypassing the models which are build to detect fraud (Zhou & Kapoor, 2011). Existing fraud detection techniques usually share very similar data mining principles but they can differ in many aspects with specialized domain knowledge (Bolton & Hand, 2002). Billions of dollars are lost annually due to credit card fraud (Chan et al. (1999), Chen et al. (2006)). Ngai et al. (2011) found 49 journal articles detailing mathematical fraud detection models, mostly relating to insurance fraud. The latter journal author noted that the learning techniques used to detect fraud include logistic regression (LR), artificial neural networks (ANN), Bayesian belief networks (BBN) and decision trees (DT). They note that these articles provide many different solutions to the problems inherent in the detection and classification of fraudulent transactions. West & Bhattacharya (2016) discuss more recent developments in the area of fraud classification, including the use of automated real time processes that can be used to detect fraud.

There are two types of frauds that can be identified in a set of transactions; card not present (CNP) and card-present fraud (CP) (Thennakoon et al., 2019). CNP fraud occurs when a

payment card is used without the cardholder physically presenting the card at the transaction time. Merchants unintentionally process fraudulent transactions since the perpetrator has gained access to the card's magnetic stripe information and know the payment card number, the card's three-digit security code, and the cardholder's name and address. There is no way for the merchant to verify the cardholder's signature or request additional identification because the payment card is never physically handled by the merchant. The victim, who keeps the compromised card, is generally unaware of the situation. However, in CP fraud the perpetrator has the actual stolen card or a fraudulent duplicated card made using the card number and magnetic stripe information (Talbot et al., 2003). Unlike transactions where a card is present, the responsibility for the loss of fraudulent CNP transactions lies with the merchant, which means the payment processor will charge the merchant for the full value of the fraudulent purchase.

1.2 Credit Card Fraud in South Africa

Fraud affects almost every business area, from data breaches that jeopardize end-customer privacy and payment security to ransom attacks that cost businesses vast amounts of money (Amanze & Onukwugha, 2018). Over the past decades, financial fraud has brought shocking losses to the global economy, threatening the efficiency and stability of capital markets (Zhu et al., 2021). According to SABRIC (2021) the gross losses due to fraud committed on South African issued debit cards amounted to more than R520 million in 2020, an increase of 26.5% compared to 2019; and more than R469 million for issued credit cards which is a decrease of 28.4% compared to 2019. Due to financial uncertainty, people used debit cards as opposed to buying on credit, as they were more comfortable spending money they already had. During the COVID-19 lockdown restrictions, consumer buyer behaviour shifted to online platforms. In addition, debit cards were enabled for online purchases, creating new opportunities for scammers to steal card information from bank customers. Social vulnerabilities resulting from fear and confusion caused by the pandemic and adjusting to lockdowns were also exploited by criminals. The Covid-19 lockdown forced many small and large companies to move their businesses to the internet to provide worldwide services. Any type of fraud can severely impact the business, whether it's perpetrated by opportunistic individuals or serious and organized crime groups. However, serious and organized crime can often increase the scale and impacts of fraud, and professional fraudsters make their fraudulent activities more difficult to detect.

1.3 Objectives

This thesis aims to perform a predictive analysis of the European credit card transaction dataset using machine learning techniques to detect fraudulent transaction. The approach is to use predictive models to identify whether a transaction is fraudulent or non-fraudulent. Several machine learning algorithms , namely logistic regression, decision trees, support vector machines, k-nearest neighbours, multilayer perceptron and artificial neural networks are implemented and the results are discussed. To address the class imbalance, a data-level and algorithmic level approach is implemented. In data-level approach, sampling techniques such as random undersampling, SMOTE with Tomek link and one-sided sampling were used whereas in the algorithmic level approach parameter tuning techniques such as random search were used to find parameters that improve classifier performance.

1.4 Thesis Outline

Chapter 2 discusses previous work done on credit card fraud detection and challenges in fraud detection. This brief review of the literature suggests that decision trees, binary logistic regression classifiers, support vector machines and neural networks would be appropriate classifiers to apply to the data in question.

Chapter 3 introduces supervised, unsupervised and semi-supervised classification, how classification models are estimated and related challenges as well as the problem of overfitting.

Chapter 4 introduces and discusses the k-nearest neighbour classifier, decision trees, binary logistic regression classifiers, support vector machines, neural networks and the different ways in which these classifiers can be directly modified to mitigate the bias towards the majority class.

Chapter 5 discusses different techniques of measuring binary classification performance for balanced and imbalanced data.

Chapter 6 discusses accuracy assessments for imbalanced data at the algorithmic and data-level.

Chapter 7 provides a brief background information on the European credit card dataset, analysis of the results and compare them to similar studies.

Chapter 8 summarizes the results and provides several insights for future work.

Chapter 2

Background and Challenges in Fraud Detection

This chapter introduces credit card fraud and different techniques that have been successfully applied in the context of credit card fraud.

2.1 Related Work

Ghosh & Reilly (1994) conducted a feasibility study to determine an ANN's effectiveness for fraud detection on a Mellon bank credit card portfolio. They trained an ANN-based system on a sample of non-fraudulent and fraudulent transactions. The data was sampled such that there are thirty (30) non-fraudulent cases for each fraudulent case. In the training phase, four hundred and fifty thousand (450 000) transactions were used out of 650 000 available accounts. The ANN was trained on observations of fraud due to lost cards, stolen cards, counterfeit, mail order, and non-received issue fraud. The trained ANN was tested on approximately two million unsampled transactions that were authorized in a two month period. The training dataset was taken from the transactions before the test set. The ANN detected more fraudulent accounts with significantly fewer false positives, reduced by a factor of twenty (20), over rule-based fraud detection procedures. The ANN-based system provided a substantial improvement in accuracy and the timeous detection of fraudulent transactions. The ANN has the ability to detect fraudulent patterns on credit accounts and it has achieved a reduction in total fraud losses of 40% from 20% previously achieved using the rule-based method. This reduced the case load for human review since they reviewed approximately seven hundred and fifty (750) accounts per day resulting on average to only one detected fraudulent account per week.

Shen et al. (2007) investigated the effectiveness of applying classification models to credit card fraud detection problems. DTs, ANNs, and LR models were built and tested. Models were built and subsequently evaluated on transactions from 2005 for training; 2006 for

testing and model validation. The credit card dataset had 0.07% fraudulent transactions. In training phase, all fraudulent cases were used and a sample of non-fraudulent cases were removed. A lift table and lift chart were used to describe the models' usefulness and create the scored dataset. The ANN and LR approach outperformed the DT model in classifying credit card fraud. Credit issuers can utilize fraud models to compare the transaction information with historical trading patterns to predict a current transaction's probability of being fraudulent and provide a scientific basis for intelligent authorized anti-fraud strategies or refuse to authorize a transaction and launch investigations of suspicious transactions.

Sánchez et al. (2009) proposed the use of association rules to extract knowledge so that standard behaviour patterns may be obtained of fraudulent transactions from credit card databases to detect and prevent fraud. Their methodology optimizes the execution times, reduces excessive generation rules and makes the results more intuitive. They noted globalization causes a sharp increase in commercial management and hence there is a need for intelligent tools which integrally solve the problems of extracting knowledge from operational databases to support decision making. They concluded that it was possible to provide a more comprehensive, proactive online solution to provide knowledge for commercial decision making by extracting knowledge using fuzzy logic techniques which are applied to operational databases and strategic decisions for organizations.

Sahin & Duman (2011) developed and applied ANN and LR to a credit fraud detection problem. The credit dataset distribution was highly imbalanced, with nine hundred and seventy eight (978) fraudulent transactions and twenty two million non-fraudulent transactions. The dataset was preprocessed before the classification models were developed. Stratified sampling was used to under-sample the non-fraudulent transactions so that these models could learn both classes. The stratified samples of non fraudulent transactions were combined with fraudulent transactions to form three samples with different fraudulent to non-fraudulent ratios. The first sample had a ratio of one fraudulent to one non-fraudulent transaction; the second had a ratio of one fraudulent to four non-fraudulent transaction; and the third had a ratio of one fraudulent to nine non-fraudulent transaction. Sahin & Duman (2011) observed that the ANN classifier outperformed the LR in classifying transactions into fraudulent or non-fraudulent transactions. Prediction accuracy and the true positive rate (TPR) were used as model performance measures. They however noted that as the training dataset distribution became more biased, the performance of all classifiers decreases in identifying fraudulent transactions.

Ogwueleka (2011) designed an ANN architecture for credit card fraud detection using unsupervised learning. The ANN was applied to the transactional data to generate four clusters, namely low, high, risky and high-risky. The self-organizing map neural network technique provided the optimal classification of each transaction into its associated group. Receiver operating curves (ROC) for credit card fraud detection watch detected over 95% of fraud cases without false positives. This was followed by the ROC curve for LR which detected over 75%

without any false positives. Quadratic discriminant analysis (QDA) performed worst with over a detection of 60%. The credit card ANN fraud detection system detected most of the fraudulent transactions with the probability of a false-positive being below 3%. Ogwueleka (2011) concluded that the ANN system was effective in detecting fraudulent transactions.

Jha et al. (2012) employed a transaction aggregation strategy to detect credit card fraud. Transactions were aggregated to capture consumer buying behaviour prior to each transaction. These aggregated transactions were used for model estimation to detect fraudulent transactions. They estimated the logit model using primary and derived attributes, where derived attributes were created by aggregating values of transactions over different time periods. They concluded that transaction aggregation is a good strategy for fraud detection as the model with derived attributes performed in classifying transactions.

Fu et al. (2016) proposed using a convolutional neural network (CNN) based framework for fraud detection. A CNN was used to capture the hidden patterns of fraud behaviours learned from labelled data. The transaction data was represented by a feature matrix on which CNN was applied to identify a set of latent patterns for each sample. They criticized the fraud detection models proposed by Ghosh & Reilly (1994) and Maes et al. (2002) for being overly complex and noted that there was a high probability of overfitting. They applied CNN to effectively reduce feature redundancy, avoid model overfitting and to reveal latent patterns of fraudulent transactions.

Nami & Shajari (2018) developed a method involving two stages for fraud detection, namely the extraction of suitable transnational features and a dynamic random forest algorithm. In the feature extraction stage, additional features are derived from primary transnational data in an effect to better understand cardholders' spending behaviour. The cardholders' behaviour varies over time so his/her new behaviour deviated from recent transactions. A similarity measure was established based on transaction time. This measure assigns greater weight to recent transactions. In the second stage, the dynamic random forest algorithm was applied to the first time initial detection and a minimum risk model was applied in cost-sensitive detection. They found that the recent behaviour of cardholders exerted a considerable effect on decision-making regarding the evaluation of transactions as either fraudulent or non-fraudulent. The use of both primary and derived transnational features increases the F-measure. An average increase of 23% in the prevention of damage was attained through the cost-sensitive approach.

Pumsirirat & Yan (2018) developed a model based on a deep autoencoder (AE) and restricted Boltzmann machine (RBM). This model reconstructs the non-fraudulent transactions to find anomalies from normal patterns. This model detects fraudulent cases that cannot be detected based on the previous history or supervised learning approaches. The deep learning AE is an unsupervised learning algorithm that applies back-propagation by setting the inputs equal to the outputs. Tensor flow was used to implement AE, RBM and H20, a multilayer feedforward ANN that is trained with stochastic gradient descent using back-propagation,

using deep learning. The root squared error and ROC curve were computed. Benchmark experiments with other tools were used to confirm that AE and RBM in deep learning can accurately detected credit card fraud in a large dataset. Pumsirirat & Yan (2018) used three credit card datasets, namely German, Australian and European to measure the area under the ROC curve (AUC) for AE and RBM. The AE and RBM obtained an AUC score of over 95% for European credit card dataset.

Kiran et al. (2018) conducted research on credit card fraud detection using the Naive Bayes (NB) and k-nearest neighbour (kNN) classifiers. They observed different results for these classifiers applied to the same dataset. The purpose of using different classifiers was to enhance the accuracy and flexibility of the algorithm. The dataset comprises the records of European credit card holders who made transaction using their credit card in the month of September 2013. This dataset contains records of transactions that were made in a two day period. The number of transactions that were made within this specified period were 284 807. 492 of these transactions were identified as being fraudulent. The dataset is highly imbalanced with more observation oriented as a positive class, namely non-fraudulent. A principal component analysis was conducted. Unfortunately the dataset does not provide any background information on the original features due to confidentiality agreements. This makes it difficult to interpret the principal components. The kNN classifier applied to the principal component dataset was 95% accurate while the NB classifier was 90% accurate. Kiran et al. (2018) concluded that credit card fraud detection can't be efficient in practice if it is done using only one classifier.

Dornadula & Geetha (2019) developed a novel credit card fraud detection system for streaming transactional data to analyse the customers' past transaction details and extract behavioural patterns. Cardholders were clustered into different groups based on their transaction amounts (low, medium and high) using range partitioning. A sliding window algorithm was implemented to aggregate the cardholders' transactions from different groups. This helped to extract features that described the cardholders' behavioural patterns. Different classifiers, namely local outlier factor (LOF), isolation forest (iForest), SVM, LR, DT and random forest (RF), were trained over the groups separately. The classifier with the best rating score was chosen as the best method for predicting fraud. The Matthews Correlation Coefficient (MCC) was used to measure accuracy due to the imbalanced data. The MCC results were improved after SMOTE was applied. Dornadula & Geetha (2019) observed that the DT classifier outperformed all other classifiers before SMOTE was applied to the dataset. RF classifier outperformed all other classifiers one the dataset was balanced. LR, DT and RF are the only classifiers which showed significant improvement after SMOTE was applied to the European credit card dataset.

Darwish (2020) designed an improved two-level credit card fraud tracking model based on an imbalanced dataset using semantic fusion of k-means and the artificial bee colony (ABC) algorithm. This improves identification precision and accelerates the convergence of detection.

The ABC algorithm filters the dataset's characteristics using an integrated rule engine to evaluate whether the transaction is fraudulent or non-fraudulent. They concluded that the fusion of multiple pieces of evidence and learning are appropriate approaches for addressing these real-world problems where behavioural patterns are intricate and there is little or no knowledge about the semantics of the application domain.

2.2 Challenges in Fraud Detection

2.2.1 Concept Drift

The change in fraudulent activity and customer behaviour leads to an evolution of online transaction distribution, this is known as concept drift (Kubat & Widmer, 1995). Changes in the hidden patterns can cause more or less drastic changes in the target variable. An effective model should be able to track such changes and adapt to them quickly. A difficult problem in handling concept drift is distinguishing between true concept drift and noise. Some algorithms may overreact to noise, erroneously interpreting it as concept drift, while others may be highly robust to noise, adjusting to the changes too slowly (Stolfo et al., 1997).

2.2.2 Imbalanced Class Distribution

The skewed distribution of the classes is considered as one of the most critical problems in FFD (Maes et al., 2002). In general, the imbalanced class problem happens when there are far fewer samples of fraudulent cases than non-fraudulent cases. In a supervised learning approach, the class imbalance problem arises when the minority class is very small, leading to numerous issues, such as preventing the model from identifying patterns in the minority class data (Stolfo et al., 1997). In addition, class imbalance has a severe impact on the performance of classifiers, which tend to be overwhelmed by the majority class and ignore the minority class since most data mining algorithms are not designed to cope with such class imbalance (Liu et al., 2008). The imbalance class problem can be addressed at an algorithmic level but typically is addressed on the data-level (Krawczyk, 2016). At the algorithmic level, existing learning algorithms are directly modified to mitigate the bias towards the majority class and adapt them to mining data with skewed distributions, for example regularization parameter, class weight and stopping criterion. At the data level, a pre-processing step is performed to rebalance the dataset. Several pre-processing techniques have been proposed to overcome the class imbalance problem on the data level, including random oversampling, random undersampling, synthetic minority oversampling technique (SMOTE) and cost-sensitive learning (Mekterović et al., 2021).

2.2.3 Large Amounts of Data

The actual transaction label is only available several days after the transaction occurred since investigators can not timeously check all the transactions. Analysts can not keep up with the exponential rate at which transactions are made in the e-commerce space. The fastest analyst can go through approximately a thousand (1 000) transactions a day, which may lead to mislabelled cases in the dataset to be used for building the model due to human error and bias (Bhattacharyya et al., 2011). Credit card datasets are usually large with high dimensionality. The presence of many features makes the process of data mining and detection difficult and complicated (Hilas & Sahalos, 2007). In addition, high dimensionality slows down the model building process. Several techniques have been proposed to overcome this challenge, such as principal component analysis (PCA), feature selection and t-distributed stochastic neighbour embedding (t-SNE). These techniques help reduce the size of the model and thus reduce the computation time (Abdallah et al., 2016).

2.2.4 Real Time Detection

Fraud detection systems are divided into two different categories: offline detection and online detection based on different types of fraud. When detecting online fraud, an online credit card payment application requires immediate detection and response. However some applications require offline detection. Online fraud detection must be able to manage limited resources to ensure the detection process works effectively. Therefore, the effectiveness of any proposed online fraud detection solution benefits from reduced amounts of data and the reduced associated computational complexity of the methods used for detection (Abdallah et al., 2016). The online banking system is fixed; the customer has access to the same banking system, leading to good credentials to characterize common behavioural sequences and identify suspicions of fraudulent online banking. However customer behaviour patterns are diverse; fraudsters tend to simulate the real behaviour of customers. They often change their behaviour to compete with advances in fraud detection (Minastireanu & Mesnita, 2019).

2.3 Chapter 2 Summary

This literature review suggest that kNN, DT, SVM, LR and MLP are appropriate classifiers to apply to the analysis of the data in this thesis.

Chapter 3

Classification

This chapter introduces classification. Section 3.4.1 discusses the direct estimation of the expected risk or test error from the empirical risk or train error, an optimistic approximation of the expected risk. Several challenges arise when attempting to estimate the expected risk. Sections 3.4.2 and 3.4.4 discuss overfitting and model complexity in the context of model estimation.

3.1 Introduction to Supervised, Semi-supervised and Unsupervised Learning

Classification is the process of predicting the class label given the input variables or features (James et al., 2013a). The objective of classification is to estimate the correct class label using the input variables, $\mathbf{x}_i$, to predict the discrete output or target variables y_i for each $i \in \{1, 2, ..., N\}$. This mapping can be denoted by the function $y_i = f(\mathbf{x}_i)$. Data mining algorithms can categorized into three learning methods: supervised, unsupervised, or semi-supervised (Neelamegam & Ramaraj, 2013). In supervised learning, the algorithm works with a set of observations with known class labels. Suppose $f(\mathbf{x}_i) \in \{1, 2, ..., M\}$ denotes M discrete classes. Let $D_N = \{\mathbf{x}_i, y_i\}_{i=1}^N$ denote a set of N observations where $\mathbf{x}_i \in \mathbb{R}^d$ is the input vector and $f(\mathbf{x}_i) = y_i \in \{1, 2, ..., M\}$ is the corresponding known class label. Each class label y_i is an integer between 1 and M indicating the class label of the i^{th} training observation. Furthermore, let $D_{train} = \{\mathbf{x}_i, y_i\}_{i=1}^t \subset D_N$ denote the training set and $D_{test} = \{\mathbf{x}_i, y_i\}_{i=t+1}^N \subset D_N$ denote the test set such that $D_{train} \cap D_{test} = \emptyset$ and $D_{train} \cup D_{test} = D_N$ where $t < N$. The supervised classification model is trained on D_{train}. In the training phase the classification algorithm has access to the feature variables and their corresponding class label for all observations in D_{train}. In the testing phase, the classification algorithm has access to feature variables only in D_{test}. The classification algorithm uses all feature variables in D_{test} to predict the corresponding class labels and these new class labels are compared to the actual class labels to measure the model's performance. In unsupervised learning, there are

no known class labels. These algorithm aims to group observations in $D_N \cap \{\mathbf{x}_i\}_{i=1}^N = \{\mathbf{x}_i\}_{i=1}^N$ based on the similarity of these attribute values, typically for a clustering task (Lebret et al., 2015). The result provided by clustering is usually a partition $D_z = \{\mathbf{z}_i\}_{i=1}^N$ of $D_N \cap \{\mathbf{x}_i\}_{i=1}^N$ into K groups such that

$$z_{ik} = \begin{cases} 1, & \text{if } D_N \cap \{\mathbf{x}_i\}_{i=1}^N \text{ belongs to the } k^{th} \text{ group, or} \\ 0 & \text{otherwise.} \end{cases}$$

Semi-supervised learning is used when a small subset of labelled class observations is available, but large number of the output variables are unlabelled, that is all the class information is not available. Let $\{\mathbf{x}_i, y_i\}_{i=1}^t$ denote the set of t observation with known class labels and $\{\mathbf{x}_i\}_{i=t+1}^{t+n}$ denote the set of n unlabelled observations where class labels are unknown. It follows that $D_N = \{\mathbf{x}_i, y_i\}_{i=1}^t \cup \{\mathbf{x}_i\}_{i=t+1}^{t+n}$ is the set of N observations where $N = t + n$ and $n \gg t$. Semi-supervised learning combines this information to exceed the classification performance that can be obtained either by dropping the unlabelled data and applying supervised learning to the subset of the data with known class labels or by dropping the class labels of the known subset and applying unsupervised learning to all the data, that is now all unlabelled (Lebret et al., 2015).

3.2 Multiclass Classification

Multiclass classification is a classification task that consists of more than two class labels, that is $|\{1, 2, ..., M\}| > 2$. For example, suppose an academic researcher wants to host the 2021 mathematics Olympiad for all high schools in Makhanda. The academic researcher wants to predict which high school will win bronze, silver and gold, that is $y_i \in \{\text{bronze}, \text{silver}, \text{gold}\}$ denotes the class labels. The classification model will be trained on D_{train} where $\mathbf{x}_i$ could contain feature variables such as the schools overall performance in mathematics, if the school is a private or public school, how many times has the school won the mathematics Olympiad in the past etc. Let 1 denote bronze, 2 denote silver and 3 denote gold, that is $y_i \in \{1, 2, 3\}$. This is a multiclass classification examples since there are more than two different class labels where each high school is assigned to only one class label.

3.3 Binary Classification

Binary classification is a classification task where the target or output variable can take one of only two class labels; that is $y_i \in \{c_1, c_2\}$ where c_i denotes the i^{th} class label. For example, consider the classification of transactions into fraudulent or non-fraudulent transactions. The class label is coded as 1 for a fraudulent transaction and 0 for non-fraudulent transaction. Since there are only two class labels, that is $y_i \in \{0, 1\}$, this is a binary classification task.

Binary classification can be viewed as binary regression since we can predict the conditional probability that a binary response is 1 given the feature variables $\mathbf{x}_i$ (James et al., 2013b).

3.4 Model Estimation

3.4.1 Introduction

Suppose $\mathcal{X}$ denotes the input space, that is $\mathcal{X} = \mathbb{R}^d$, where d denotes the dimensions, and $\mathcal{Y}$ denotes the output space. The joint probability distribution on $\mathcal{X} \times \mathcal{Y}$ is denoted as $P_{X,Y}$. Let (X, Y) denote a pair of random variables distributed according to $P_{X,Y}$ and D_N denote independent, identically distributed (iid) random sample from $P_{X,Y}$. The objective of decisional modeling is to find a function $f \in \mathcal{F}$, $f : \mathcal{X} \longrightarrow \mathcal{Y}$ that predicts $y \in \mathcal{Y}$ from $\mathbf{x} \in \mathcal{X}$. The function f is chosen from a family of parametric functions or a hypothesis space, $\mathcal{F}$. The performance of a predictor f is measured by the expected loss, or risk or generalization error, which is defined as

$$R(f) = E_{P_{X,Y}}[\mathcal{L}(\mathbf{x}, y, f(\mathbf{x}))] \tag{3.4.1}$$

where the expectation is taken with respect to the distribution $P_{X,Y}$ and $\mathcal{L}(\cdot)$ is the loss, or error, function. The loss function quantifies the extent to which $f(\mathbf{x})$ corresponds to the expected value of y. The most commonly used loss is the 0-1 loss function (Nguyen & Sanner, 2013) which is defined as

$$\mathcal{L}(\mathbf{x}, y, f(\mathbf{x})) = \begin{cases} 0, & \text{if} \quad f(\mathbf{x}) = y \\ 1, & \text{otherwise.} \end{cases} \tag{3.4.2}$$

In Eq. 3.4.2, the loss is zero if the prediction is correct and 1 if the prediction is incorrect. This means any misclassification brings the same penalty regardless of the type of error. For example, if we are classifying transactions as being fraudulent or non-fraudulent the misclassification of a fraudulent transaction in the class of non-fraudulent transaction brings the same penalty as the misclassification of a non-fraudulent transaction to the class of fraudulent transactions. The 0-1 loss function is not adequate for highly imbalanced datasets (García-Gómez & Tortajada, 2015). The ultimate goal of supervised learning is to find $f^*_{D_N} \in \mathcal{F}$ for which the risk $R_{D_N}(f)$ is minimal by searching for a model that minimizes the training error, thus

$$f^*_{D_N} = \underset{f \in \mathcal{F}}{\operatorname{argmin}} \; R_{D_N}(f).$$

Unfortunately in most cases Eq. 3.4.1 cannot be evaluated since the distribution $P_{X,Y}$ is unknown to the learning algorithm (Vapnik, 1992). However an approximation of Eq. 3.4.1

can be computed by taking the average loss function on the training dataset

$$R_{D_N} = \frac{1}{N} \sum_{i=1}^{N} L(\mathbf{x}_i, y_i, f). \tag{3.4.3}$$

Learning the function f by minimizing Eq .3.4.3 is known as the Empirical Risk Minimization principal (ERM) (Vapnik, 1992). The risk is an important measure of goodness of the predictor $f(\cdot)$ since it tells how it performs on average in terms of the loss function. The minimum risk is defined as

$$R(f_{min}) = \inf_{f \in \mathcal{F}} R(f),$$

where the *infimum* is often taken with respect to all measurable functions. Suppose that the learning algorithm chooses the predictor f^* from $\mathcal{F}$ such that

$$f^* = \operatorname*{argmin}_{f \in \mathcal{F}} R(h)$$

and let $f^*_{D_N} \in \mathcal{F}$ denote a function that minimizes the empirical risk. The excess risk of the output $f^*_{D_N}$ of the learning algorithm is defined and decomposed as follows:

$$R(f^*_{D_N}) - R(f_{min}) = \underbrace{R(f^*) - R(f_{min})}_{\text{approximation error}} + \underbrace{R(f^*_{D_N}) - R(f^*)}_{\text{estimation error}}.$$

This decomposition reflects a trade-off that is similar to the bias-variance trade-off (Zhao, 2017). The approximation error, that is the bias of an estimator, term behaves like a bias square term, and the estimation error, that is the variance, behaves like the variance term in standard statistical estimation problems. Similar to the bias-variance trade-off, there is also a trade-off between the approximation error and the estimation error. The approximation error is zero for the family that contains the best model and strictly positive for families that do not contain this model (Zhao, 2017). Let $\mathcal{F}_1$ and $\mathcal{F}_2$ denote two families. The best model of $\mathcal{F}_1$ has a better approximation error than the best model of $\mathcal{F}_2$ if $\mathcal{F}_1$ is more complex than $\mathcal{F}_2$. The estimation error is zero if the function in $\mathcal{F}$ that minimizes the empirical risk also minimizes the the expected risk. The model that minimizes the empirical risk is likely to move away from a model that minimizes the expected risk when the capacity of family $\mathcal{F}$ is high, hence the estimation error increases (Tian, 2021).

3.4.2 Model Overfitting

Overfitting is a fundamental issue in supervised learning that prevents a models from generalizing well on test data (Schaffer, 1993). Generally overfitting can be categorized into noise learning and model complexity. Under noise learning, the training set has few observations,

that is D_{train} is small, or has less representative data. Refer to section 3.4.4 for a discussion on model complexity.

3.4.3 Overfitting in the Context of Regression

Consider Figs. 3.1, 3.2 and 3.3. In these figures the blue stars denote the training set, D_{train}, the orange stars denote the test set, D_{test}, e_i denotes the error of the i^{th} observation and the black line denotes the fitted regression model. The total training error can be calculated as

$$e^{train} = \sum_{i=1}^{t} (y_i^{train} - \hat{y}_i^{train})$$

and the total test error by

$$e^{test} = \sum_{i=1+t}^{N} (y_i^{test} - \hat{y}_i^{test}).$$

The linear model, a polynomial of degree 1, in Fig. 3.1, is underfitting the training data since it is unable to capture the non-linear relationship between input observations, $\mathbf{X}_{train}$, and the target variables, $\mathbf{y}_{train}$. These models usually have high bias and low variance. This happens when there is less data to build an accurate model or when non-linear data is used to build a linear model.

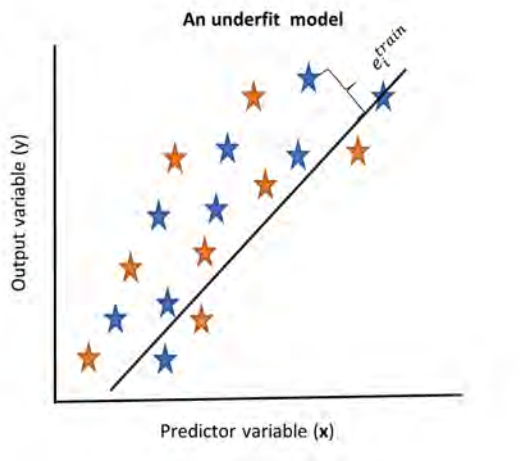


Figure 3.1: Illustration of an underfitting model (adapted from Weston (2014)).

Fig. 3.2 shows a polynomial of degree two fitted on the same dataset. The model appears to be a good fit on the training data and also generalizes well on the test data (see Fig. 3.4). This model has low bias in that it is a better model for the training data that captures the relationship between predictor variable and response variable, and low variance as there isn't much difference between the training and test error. The polynomial of degree 2 has low training error, low test and is less complex than the model fitted in Fig. 3.3, a polynomial of degree 3.

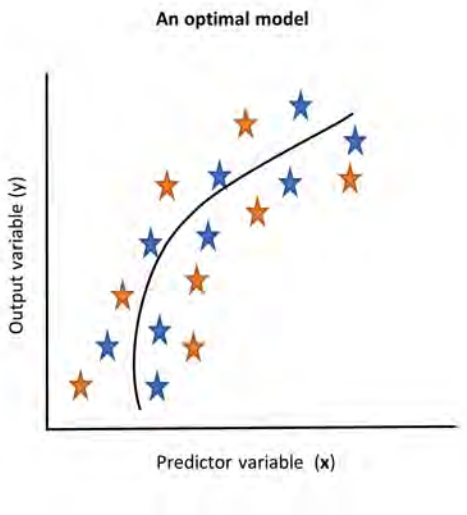


Figure 3.2: Illustration of an optimal model (adapted from Weston (2014)).

Fig. 3.3 shows a polynomial of degree 3 fitted to the same dataset. The model is too well trained, it takes noise in the training set as set of observations used to train the model. This reduces the model's ability to predict the response variable on the test data, an unseen data during training, since it has 'memorized' the training data. The training error is very low but the test error is high, therefore the model is overfitted. When a model is overfitting it has high variance and low bias and this reduces the model's ability to generalize well.

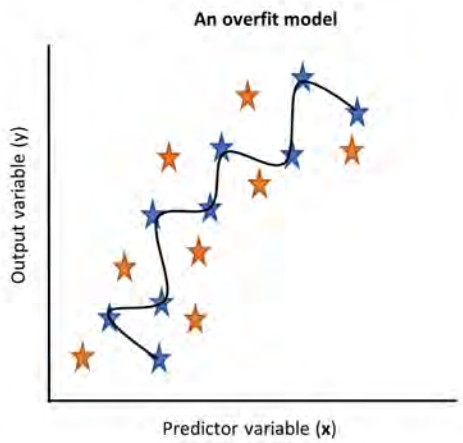


Figure 3.3: Illustration of an overfitting model (adapted from Weston (2014)).

3.4.4 Model Complexity

Complex models in general perform better on training data than simpler models since they have more parameters that can be adjusted during training to get a good fit (Myung, 2000). Hence complex models have small training error when compared to non-complex models. However a model that is too complex may end up overfitting the training data. As a result, there is low training error and high test error (Cheung & Rensvold, 2002). A complex model showing high variance may improve in performance if trained on more observations. In most cases, the error rate on the training set starts off low when there are few observations and increases as more observations are used (Sejdinovic, 2021). On the other hand, the error rate on the test set is typically high for a model trained on a few observations and decreases with more training observations. If the test error rate remains higher than the training error rate even when the training set is large, then the model is overfitting (Rashidi et al., 2019). Vapnik (1992) proposed regularization as an implicit approach to control model complexity and avoid overfitting. The optimal model, $R(f^*_{D_N})$, in the considered family is minimized by taking the sum between $R_{D_N}(f)$ and a regularization term, $G(f)$, that is

$$f^*_{D_N} = \underset{f \in \mathcal{F}}{\operatorname{argmin}} \left[R_{D_N}(f) + \alpha G(f) \right]$$

where $\alpha > 0$ balances the trade-off between goodness of fit and model complexity. The higher the value of α, the higher the model complexity penalty. The value of α is selected, based on the training set, to achieve a good balance between goodness of fit and model complexity. In decision modelling, the goal is to find a model with the best generalization error, not a model with the lowest training error. Training error is not a suitable estimator of the generalization error (Myung, 2000). To minimize the generalization error one must look at the trade-off between minimizing the model complexity and the training error.

Fig. 3.4 illustrates the trade-off between bias and variance. A high error rate on both the training set and test set indicates that the model is too simple in that the model underfits the data and fails to capture any relationship presented in the dataset. In this case, the model will have low variance and high bias. If the error rate is low on the training data but high on the test data then the model maybe be too complex, the model is overfitting. In this case the model will have high variance and low bias. The shaded area in this figure indicates the ideal zone. In this region, both the test and training error are low and the model is not over or under fitting the dataset. Let $e^{\min}$ denote the difference between the training and test error that generalizes the model. If $e^{\text{train}} - e^{\text{test}} > e^{\min}$, the model is complex and therefore overfitting, see for example Fig. 3.3. If $e^{\text{train}} - e^{\text{test}} < e^{\min}$ then the model is underfitting, see for example Fig. 3.1. Finally, if $e^{\text{train}} - e^{\text{test}} = e^{\min}$ then the model is optimal, see for example Fig. 3.2.

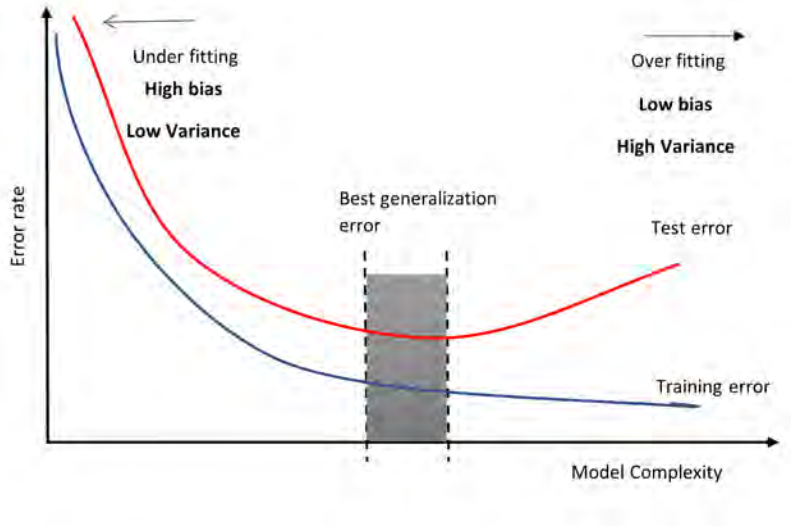


Figure 3.4: Illustration of model complexity (adapted from Rashidi et al. (2019)).

3.5 Chapter 3 Summary

This chapter discussed a brief introduction to supervised, unsupervised and semi-supervised classification. Section 3.4.1 discussed model estimation and demonstrated that direct estimation of the generalization error by the training error is excessively optimistic. Challenges that emerge when building a classifier, namely overfitting and model complexity, are discussed.

Chapter 4

An Introduction to Supervised Classification

This chapter introduces supervised models and the theory of the classification models used for fraud detection is discussed. The k-nearest neighbour, decision tree, logistic regression, support vector machine and multilayer perceptron supervised learners are discussed in sections 4.1, 4.2, 4.3, 4.4 and 4.5 respectively.

4.1 k-Nearest Neighbour

The k-nearest neighbour (kNN) classifier can be based on Euclidean distance. Euclidean distance represents the straight line distance between the test observation, $\mathbf{x}_t$, and the training observation, $\mathbf{x}_i$. Thus,

$$d^2(\mathbf{x}_i, \mathbf{x}_t) = \sum_{p=1}^{d} (\mathbf{x}_{ip} - \mathbf{x}_{tp})^2 = ||\mathbf{x}_i - \mathbf{x}_t||^2$$

where $t = 1, 2, ..., N$. The kNN classifier first identifies the k points in the training data that are closest to $\mathbf{x}_t$ based on a distance measure, for example Euclidean distance. The kNN classifier estimates the test observation's label according to the majority of the class's in this neighbourhood (James et al., 2013b). When there is no weighting, this majority voting method can be expressed as

$$\hat{f}(\mathbf{x}_t) = \underset{v \in \{-1,+1\}}{\operatorname{argmax}} \sum_{(\mathbf{x}_i, y_i) \in N_k(\mathbf{x}_t)} \mathbb{I}(y_i = v) \tag{4.1.1}$$

where $\hat{f}(\mathbf{x}_t)$ is the predicted class label and $N_k(\mathbf{x}_t)$ denotes the set of k training observations closest to $\mathbf{x}_t$ with $k \in \mathbb{Z}^+$ (Liu & Chawla, 2011). The majority voting method in Eq. 4.1.1

can be rewritten as follows:

$$\begin{aligned}\hat{f}(\mathbf{x}_t) &= \underset{v\in\{-1,+1\}}{\operatorname{argmax}} \sum_{(\mathbf{x}_i,y_i)\in N_k(\mathbf{x}_t)} \mathbb{I}(y_i = v) \\ &= \max\Big\{ \sum_{(\mathbf{x}_i,y_i)\in N_k(\mathbf{x}_t)} \mathbb{I}(y_i = -1), \sum_{(\mathbf{x}_i,y_i)\in N_k(\mathbf{x}_t)} \mathbb{I}(y_i = +1)\Big\} \\ &= \max\Big\{ \frac{1}{k} \sum_{(\mathbf{x}_i,y_i)\in N_k(\mathbf{x}_t)} \mathbb{I}(y_i = -1), \frac{1}{k} \sum_{(\mathbf{x}_i,y_i)\in N_k(\mathbf{x}_t)} \mathbb{I}(y_i = +1)\Big\} \\ &= \max\Big\{ P(y_i = -1), P(y_i = +1)\Big\} \end{aligned} \tag{4.1.2}$$

where $P(y_i = -1)$ and $P(y_i = +1)$ represent the probability of class -1 and class $+1$ appearing in $N_k(\mathbf{x}_t)$ respectively.

Fig. 4.1 illustrates the kNN algorithm with 3 nearest neighbours. In this figure, a small training set consists of seven red triangles, that belong to class A, and seven blue stars, that belong to class B. The goal is to predict the class of the new observation labelled by the green cross. The kNN classifier identifies three observations closest to the green cross by calculating the Euclidean distance between the green cross and all other observations in the training set. The neighbourhood is shown in Fig. 4.1 by the circle. The neighbourhood contains two red triangles and one blue star. From this it can be deduced that $P(\text{Red} \quad \text{Observation}) = \frac{2}{3}$ and $P(\text{Blue} \quad \text{Observation}) = \frac{1}{3}$. Based on the majority voting, the kNN classifier will predict that the cross belongs to class A, the red class.

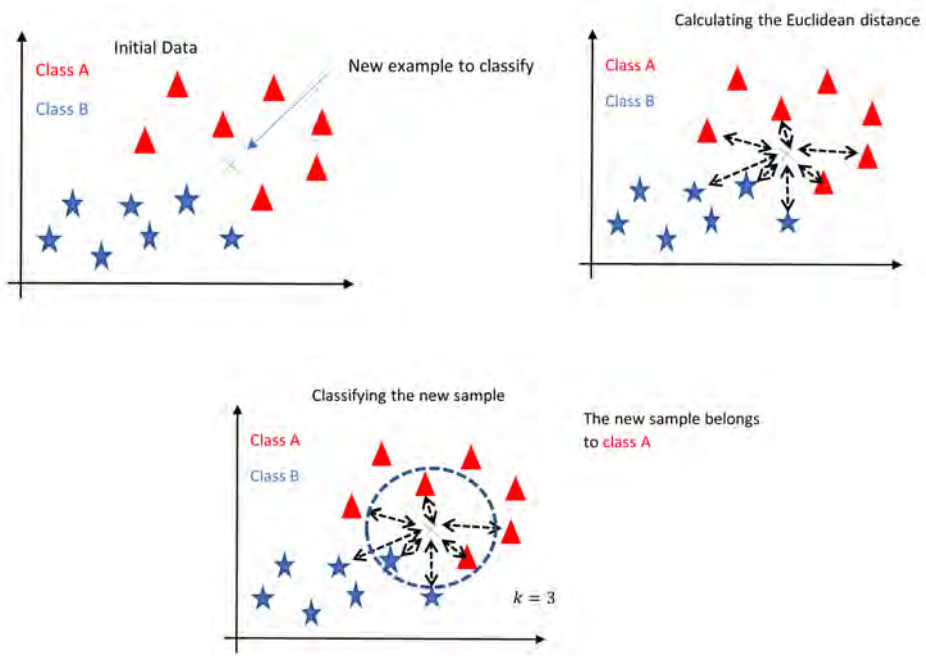


Figure 4.1: An example of a kNN classifier (adapted from Navlani (2018)).

4.1.1 The Weighted k-Nearest Neighbours

The classification performance of the kNN classifier is affected by how much the k nearest neighbours vary in their distances (Fan et al., 2019). Dudani (1976) proposed that the samples nearby should be weighted more heavily than those farther away when making the decision using the distance-weighted kNN (DWkNN) algorithm. Let $\{\mathbf{x}_1, ..., \mathbf{x}_k\}$ denote the set of k nearest neighbours of the test observation $\mathbf{x}_t$, arranged in an increasing order according to the distance between $\mathbf{x}_t$ and $\mathbf{x}_i$. The distance weighted function, with weight w_i, for the i^{th} nearest neighbour of the test observation is defined as follows:

$$w_i = \begin{cases} \dfrac{d(\mathbf{x}_t, \mathbf{x}_k) - d(\mathbf{x}_t, \mathbf{x}_i)}{d(\mathbf{x}_t, \mathbf{x}_k) - d(\mathbf{x}_t, \mathbf{x}_1)}, & \text{if} \quad d(\mathbf{x}_t, \mathbf{x}_k) \neq d(\mathbf{x}_t, \mathbf{x}_1) \\ 1 & \text{if} \quad d(\mathbf{x}_t, \mathbf{x}_k) = d(\mathbf{x}_t, \mathbf{x}_1). \end{cases} \tag{4.1.3}$$

The DWkNN algorithm adjusts the kNN algorithm in Eq. 4.1.1 as

$$\hat{f}(\mathbf{x}_t) = \underset{v \in \{-1,+1\}}{\operatorname{argmax}} \sum_{(\mathbf{x}_i, y_i) \in N_k(\mathbf{x}_t)} w_i \times \mathbb{I}(y_i = v).$$

According to Eq. 4.1.3 the nearest neighbour gets weight of 1, while the furthest neighbour a weight of 0 and the other neighbours' weights are scaled linearly to the interval in between 0 and 1 (Gou et al., 2012). Even though the weighted majority voting method solves the problem of large distance variances among the k nearest neighbours, the effect of this method becomes insignificant if the neighbourhood of the test observation is too dense and one, or both, of the classes is overly presented (Liu & Chawla, 2011).

4.1.2 Selection of the Value of k

kNN DWkNN are classifiers that classify the test observation based on the majority vote of its neighbours. The value of k will determine the performance of the classifier; different k values can have a large impact on the predictive accuracy and picking a good value of k is generally unintuitive (Ling et al., 2021). The kNN classifier tries to estimate the conditional class probability from observations in a local region of the data space which contains the k nearest neighbours of the test observation (Zavrel, 1997). The estimate is affected by the selection of k since the radius of the local region is determined by the distance of the k^{th} furthest neighbour. When the value of k is very small, the local estimate tends to be very poor if the nearest neighbours are not nearby due to data sparseness or the nearest neighbours are not reliable due to noise. Increasing the value of k takes into account a larger region around the test observation in the data space and makes it possible to overcome this effect by smoothing the estimate. Unfortunately, a large value of k easily causes over-smoothing and the classification performance degrades (Gou et al., 2012). Zavrel (1997) suggested that the value of k must be determined empirically through cross validation (see section 5.7.2).

4.1.3 Class Imbalance and k-Nearest Neighbours

When the dataset is highly imbalanced, the traditional kNN algorithm tends to perform poorly as the class which is overly presented tends to dominate the neighbourhood of the test observation. Eq. 4.1.2 demonstrates that the traditional kNN classification algorithm is based on finding the class label that has a higher prior value, that is $\max\Big\{P(y_i=-1), P(y_i=+1)\Big\}$. This suggests that it only uses the prior information to predict the class label and this has suboptimal performance on the minority class when the dataset is highly imbalanced (Liu & Chawla, 2011). Liu & Chawla (2011) proposed the use of a class confidence weighted (CCW) kNN algorithm to transform the traditional kNN rule of using prior probability to using the posterior probability by incorporating Bayes' theorem in the traditional kNN algorithm. To incorporate Bayes' theorem in Eq. 4.1.2, let $N_k(\mathbf{x}_t)$ denote the sample space, $P(y_i=+1)$ and $P(y_i=-1)$ denote the priors of the two classes in the sample space. Bayes' rule (Wackerly et al., 2014) states that if $\{B_1, B_2, ..., B_n\}$ is a partition of S, the sample space, such that $P(B_i) > 0$, for $i = 1, 2, ..., n$, then

$$P(B_j|A) = \frac{P(A|B_j)P(B_j)}{P(A)} = \frac{P(A|B_j)P(B_j)}{\sum_{i=1}^{n} P(A|B_i)P(B_i)} \tag{4.1.4}$$

The collection of sets $\{B_1, B_2, ..., B_k\}$ is said to be a partition of S if for some $k \in \mathbb{Z}^+$, $S = B_1 \cup B_2 \cup B_3 ... \cup B_n$ and $B_i \cap B_j = \emptyset$ for $i \neq j$. Let class -1 be the majority class, then it is expected that the inequality, $P(y_i = -1) \gg P(y_i = +1)$ holds in most feature space regions. The traditional kNN tends to be bias towards the majority class especially in the overlapping regions since the majority class is likely to be overly presented there and thus the weighted kNN becomes ineffective in correcting the bias (Liu & Chawla, 2011). The class confidence weighted (CCW) kNN algorithm captures the probability of attribute values given a class label. The CCW kNN on a training observation is defined as

$$w_i^{\text{CCW}} = P(\mathbf{x}_i|y_i).$$

The majority voting rule for kNN, Eq. 4.1.1 can the be rewritten in the context of CCW as follows:

$$\begin{aligned}
\hat{f}_{\text{CCW}}(\mathbf{x}_t) &= \underset{v\in\{-1,+1\}}{\text{argmax}} \sum_{(\mathbf{x}_i,y_i)\in N_k(\mathbf{x}_t)} \mathbb{I}(y_i = v) \times w_i^{\text{CCW}} \\
&= \underset{v\in\{-1,+1\}}{\text{argmax}} \sum_{(\mathbf{x}_i,y_i)\in N_k(\mathbf{x}_t)} \mathbb{I}(y_i = v) \times P(\mathbf{x}_i|y_i) \\
&= \max\Big\{\frac{1}{k} \sum_{(\mathbf{x}_i,y_i)\in N_k(\mathbf{x}_t)} \mathbb{I}(y_i = -1) \times P(\mathbf{x}_i|y_i), \frac{1}{k} \sum_{(\mathbf{x}_i,y_i)\in N_k(\mathbf{x}_t)} \mathbb{I}(y_i = +1) \times P(\mathbf{x}_i|y_i)\Big\} \\
&= \max\Big\{P(y_i = -1)P(\mathbf{x}_i|y_i = -1), P(y_i = +1)P(\mathbf{x}_i|y_i = +1)\Big\}.
\end{aligned}$$

Using Eq. 4.1.4, $P(y_i = -1)$ can be rewritten as

$$\begin{aligned}
P(y_i = -1|\mathbf{x}_i) &= \frac{P(\mathbf{x}_i|y_i = -1)P(y_i = -1)}{P(\mathbf{x}_i)} \\
&\propto P(\mathbf{x}_i|y_i = -1)P(y_i = -1).
\end{aligned}$$

Therefore it follows that

$$\hat{f}_{\text{CCW}}(\mathbf{x}_t) = \max\Big\{P(y_i = -1|\mathbf{x}_i), P(y_i = +1|\mathbf{x}_i)\Big\}$$

where $P(y_i = -1|\mathbf{x}_i)$ and $P(y_i = +1|\mathbf{x}_i)$ denotes the probability of $\mathbf{x}_t$ belonging to class -1 and class $+1$ respectively, given $\mathbf{x}_t$ in $N_k(\mathbf{x}_t)$.

4.2 Decision Trees

A decision tree (DT) is constructed by recursively partitioning the feature space of the training set. The objective is to find a set of decision rules that naturally partition the feature space to provide an informative hierarchical classification model (Myles et al., 2004). To classify a new instance, we start with the root of the constructed tree and follow the path

corresponding to the observed value of the attribute in the node of the tree. This process is continued until a leaf is reached where the associated label is used to obtain the predicted class value (Jenhani et al., 2008). When building a decision tree, the complexity lies in determining the best split for each attribute. A splitting index is used to evaluate the goodness of the alternative splits for an attribute (Loh & Shih, 1997). Quinlan (1986) proposed the use of information gain. Consider a binary classification problem, the Gini, denoted by $Gini(D_N)$ and the entropy, denoted by $Ent(D_N)$ are defined as (Du & Zhan, 2002):

$$Gini(D_N) = 1 - \sum_{j=1}^{2} P_j^2$$

and

$$Ent(D_N) = -\sum_{j=1}^{2} P_j \log P_j$$

where P_j is the relative frequency of class j in dataset D_N. The information gain of attribute A in the dataset D_N can be calculated based on entropy or Gini index as follows

$$Gain_{Gini}(D_N, A) = Gini(D_N) - \sum_{v \in A} \frac{|D_N^v|}{|D_N|} \times Gini(D_N^v) \tag{4.2.1}$$

and

$$Gain_{Ent}(D_N, A) = Ent(D_N) - \sum_{v \in A} \frac{|D_N^v|}{|D_N|} \times Ent(D_N^v) \tag{4.2.2}$$

where v denotes any possible values of attribute A, $D_N^v \subset D_N$ for which attribute A has value v, $|D_N^v|$ is the number of elements in D_N^v and $|D_N|$ is the number of elements in D_N. The test node that makes the classification progress the most corresponds to selecting an attribute with the maximum gain. Maximum gain is attained when the choice of attribute makes it possible to correctly classify all the data. When gain is zero, the data points will all be misclassified after the split. The first term of Eq. 4.2.1 and Eq. 4.2.2 don't depend on the attribute A. Thus maximizing the gain means minimizing $\sum_{v \in A} \frac{|D_N^v|}{|D_N|} \times Gini(D_N^v)$ or $\sum_{v \in A} \frac{|D_N^v|}{|D_N|} \times Ent(D_N^v)$ (Du & Zhan, 2002). The information gain measure in Eq. 4.2.1 or Eq. 4.2.2 is biased towards attributes with many outcomes. Thus, DTs prefer selecting attributes that have large number of values (Quinlan, 1986). Quinlan (1986) introduced the gain ratio measure to improve the gain measure and compensate for this bias. The gain ratio measure is defined as

$$GainRatio = \frac{Gain_{Ent}(D_N, A)}{Ent(D_N)}. \tag{4.2.3}$$

The gain ratio measure has the following limitations: it may not always be defined, for example if $Ent(D_N) = 0$, and it may choose attributes with very low $Ent(D_N)$ rather than those with high gain. As a result, Quinlan (1986) proposed that Eq. 4.2.3 be applied to select from those attributes whose initial gain is at least as high as the average gain of all the attributes. De Mántaras (1991) proposed an attribute selection criterion based on the distance between partitions. This solves the problem of bias in favor of multi-valued attributes without having limitations that Quinlan's gain ratio have.

4.2.1 Decision Tree Building Example

To demonstrate how DT works we use the tennis play dataset[2] given in table 4.1. In this example, the objective of the DT classifier is to predict if tennis will be played or not, based on outlook, temperature, humidity and wind features.

Table 4.1: The tennis dataset, D_N.

Day	Outlook	Temperature	Humidity	Wind	Play
D1	sunny	hot	high	weak	no
D2	sunny	hot	high	strong	no
D3	overcast	hot	high	weak	yes
D4	rain	mild	high	weak	yes
D5	rain	cool	normal	weak	yes
D6	rain	cool	normal	strong	no
D7	overcast	cool	normal	strong	yes
D8	sunny	mild	high	weak	no
D9	sunny	cool	normal	weak	yes
D10	rain	mild	normal	weak	yes
D11	sunny	mild	normal	strong	yes
D12	overcast	mild	high	strong	yes
D13	overcast	hot	normal	weak	yes
D14	rain	mild	high	strong	no

The first step to growing a tree is to decide on the root node. To determine the root node, we have to calculate the information gain of all attributes or features and select the feature with maximum gain:

$$
\begin{aligned}
Gain_{Gini}(D_N, Outlook) &= Gini(D_N) - \sum_{v \in A} \frac{|D_N^v|}{|D_N|} \times Gini(D_N^v) \\
&= Gini(D_N) - \left(Gini(D_N^1) \times \frac{|D_N^1|}{|D_N|} + Gini(D_N^2) \times \frac{|D_N^2|}{|D_N|} + Gini(D_N^3) \times \frac{|D_N^3|}{|D_N|} \right) \\
&= Gini(D_N) - \left(\frac{5}{14} \times \frac{3}{5} \times \frac{2}{5} + \frac{4}{14} \times \frac{0}{4} \times \frac{4}{4} + \frac{5}{14} \times \frac{2}{5} \times \frac{3}{5} \right) \\
&= Gini(D_N) - 0.1714.
\end{aligned}
$$

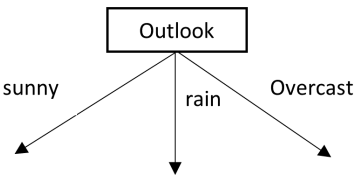
[2] https://www.kaggle.com/fredericobreno/play-tennis

$$Gain_{Gini}(D_N, Temperature) = Gini(D_N) - \left(\frac{4}{14} \times \frac{2}{4} \times \frac{2}{4} + \frac{6}{14} \times \frac{2}{6} \times \frac{4}{6} + \frac{4}{14} \times \frac{1}{4} \times \frac{3}{4}\right)$$
$$= Gini(D_N) - 0.2202.$$

$$Gain_{Gini}(D_N, Humidity) = Gini(D_N) - \left(\frac{7}{14} \times \frac{4}{7} \times \frac{3}{7} + \frac{7}{14} \times \frac{1}{7} \times \frac{6}{7}\right)$$
$$= Gini(D_N) - 0.1837.$$

$$Gain_{Gini}(D_N, Wind) = Gini(D_N) - \left(\frac{8}{14} \times \frac{2}{8} \times \frac{6}{8} + \frac{6}{14} \times \frac{3}{6} \times \frac{3}{6}\right)$$
$$= Gini(D_N) - 0.2143.$$

From the above calculations, the feature variable outlook has maximum gain and is thus selected as the root node. The root node will have three branches, namely sunny, rain and overcast, see Fig. 4.2. The next step is to determine the child nodes of outlook. For this, we explore each of the feature branches, namely sunny, rain and overcast. Starting with the feature branch sunny, determine the subset D_N^1, of D_N, under the feature branch sunny as shown in Fig. 4.2.



Day	Outlook	Temperature	Humidity	Wind	Play
D1	sunny	hot	high	weak	no
D2	sunny	hot	high	strong	no
D8	sunny	mild	high	weak	no
D9	sunny	cold	normal	weak	yes
D11	sunny	mild	normal	strong	yes

Figure 4.2: Root node and subset, D_N^1 of D_N.

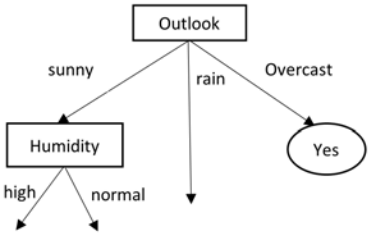
To generate a node at the feature branch sunny, we calculate information gain of all feature variables in D_N^1 as follows:

$$Gain_{Gini}(D_N^1, Temperature) = Gini(D_N^1) - \left(\frac{2}{5} \times \frac{2}{2} \times \frac{0}{2} + \frac{2}{5} \times \frac{1}{2} \times \frac{1}{2} + \frac{1}{5} \times \frac{1}{1} \times \frac{0}{1}\right)$$
$$= Gini(D_N^1) - 0.1.$$

$$Gain_{Gini}(D_N^1, Humidity) = Gini(D_N^1) - \left(\frac{3}{5} \times \frac{3}{3} \times \frac{0}{3} + \frac{2}{5} \times \frac{2}{2} \times \frac{0}{2}\right)$$
$$= Gini(D_N^1) - 0.$$

$$Gain_{Gini}(D_N^1, Wind) = Gini(D_N^1) - \left(\frac{3}{5} \times \frac{1}{3} \times \frac{2}{3} + \frac{2}{5} \times \frac{1}{2} \times \frac{1}{2}\right)$$
$$= Gin(D_N^1) - 0.2333.$$

Humidity has the maximum gain, thus the branch sunny has humidity as the child node, see Fig. 4.3. The feature value overcast represents a pure class, that is overcast only presents one class which is *yes*. Since overcast is a pure class we can make a decision and thus the leaf node under overcast is *yes*, see Fig. 4.3.



Day	Outlook	Temperature	Wind	Play
D4	rain	mild	weak	yes
D5	rain	cool	weak	yes
D6	rain	cool	strong	no
D10	rain	mild	weak	yes
D14	rain	mild	strong	no

Figure 4.3: Growing the tree and a subset D_N^2, of D_N.

The feature value rain has three corresponding *yes*'s and two *no*'s that is we can decide to have yes as a leaf node since $P(\text{yes}|\text{Outlook is rain}) = \frac{3}{5}$ which is greater than $P(\text{yes}|\text{Outlook is rain}) = \frac{2}{5}$. Since the features, temperature and wind have not been used we expend and calculate their information gain using the subset D_N^2 as shown in Fig. 4.3. The information gain for features temperature and wind are calculated as follows

$$Gain_{Gini}(D_N^2, Temperature) = Gini(D_N^2) - \left(\frac{3}{5} \times \frac{2}{3} \times \frac{1}{3} + \frac{2}{5} \times \frac{1}{2} \times \frac{1}{2}\right)$$
$$= Gini(D_N^2) - 0.2333.$$

$$Gain_{Gini}(D_N^2, Wind) = Gini(D_N^2) - \left(\frac{3}{5} \times \frac{2}{3} \times \frac{1}{3} + \frac{2}{5} \times \frac{2}{2} \times \frac{0}{2}\right)$$
$$= Gini(D_N^2) - 0.1333.$$

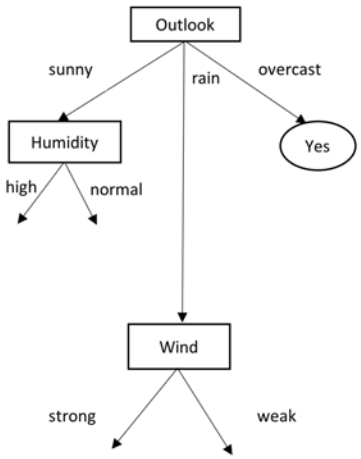
The feature variable wind has maximum gain and thus the branch rain will have wind as a child. The resulting tree is shown in Fig. 4.4. The only feature variable left is temperature. Under the internal node wind, the branch weak, will have a leaf node *yes* since

$$P(\text{yes}|\text{wind is weak}) = \frac{P(\text{yes} \cap \text{wind is weak})}{P(\text{wind is weak})} = \frac{\frac{6}{14}}{\frac{8}{14}} = 0.75$$

and

$$P(\text{no}|\text{wind is weak}) = \frac{\frac{2}{14}}{\frac{8}{14}} = 0.25.$$

Therefore, the branch strong of the internal node wind will have a leaf node *no*. Similarly, the leaf node of the branch normal has the leaf node *yes* since $P(\text{yes}|\text{humidity is normal}) = \frac{6}{7} = 0.857$ and $P(\text{no}|\text{humidity is normal}) = \frac{1}{7} = 0.1429$. It then follows that the branch high of the internal node humidity has the leaf node *no*.



Day	Wind	Temperature	Play
D1	weak	hot	no
D3	weak	hot	yes
D4	weak	mild	yes
D5	weak	cold	yes
D8	weak	mild	no
D9	weak	cold	yes
D10	weak	mild	yes
D13	weak	hot	yes

Figure 4.4: Growing tree and a subset, D_N^3, of D_N.

The fully grown tree shown in Fig. 4.5, can be used to predict the class label for a given input. For example if $\mathbf{x} = (\text{rain}, \text{high}, \text{weak})$, the DT classifier will predict the class label as *yes*.

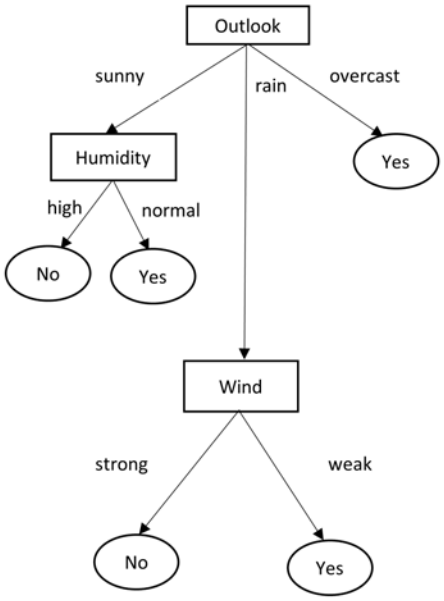


Figure 4.5: A fully grown tree.

4.2.2 Pruning

A DT performance on the training data may be a misleading indication of the actual predictive accuracy. A complex DT that achieves high accuracy on a training set will often perform poorly on the test set while a simple DT that performed far less accurately in training may perform better on the test (Schaffer, 1993). DTs are, at least initially, grown to overfit the training set. This means that a decision tree classification model has probably accounted for variation in the training set that is not representative of the entire population (Myles et al., 2004). To reduce overfitting in the training set and improve generalization error, a test set which is independent of the training set is used to evaluate the decision rules. Decision rules or leaves that reduce predictive accuracy of the decision tree classification model based on the test set are removed, thus reducing the complexity of the model. When a decision rule is removed, the associated branch node is replaced with a leaf node (Myles et al., 2004). Unpruned trees can potentially lead to a problem of small disjuncts as the tree is grown to their full complete size on imbalanced training set (Chawla, 2003). Liu et al. (2010) proposed the use of Fishers exact test as a method for pruning the decision tree which improves accuracy and the added benefit of it is that all the rules found are statistically significant.

4.2.3 Class Imbalance and Decision Trees

C4.5 and CART are two popular algorithms for decision tree induction (Cieslak & Chawla, 2008). However their corresponding splitting criterion, namely information gain and Gini

measure, are considered to be sensitive to class imbalance (Flach, 2003). Traditional pruning algorithms are based on error estimations in that a node is pruned if the predicted error rate is decreased. This pruning technique will not always perform well on imbalanced datasets (Liu et al., 2010). Pruning can have a detrimental effect on learning from imbalanced datasets, since lower error rates can be obtained by removing branches that lead to minority class leaves (Chawla, 2003). Drummond & Holte (2000a) proposed two methods of dealing with imbalanced dataset. The first method utilizes a cost insensitive splitting criterion with a cost insensitive pruning method which produces a decision tree algorithm that is least affected by cost or prior class distribution. The second method is to grow a cost independent tree which is then pruned in a cost sensitive way. Kearns (1990) suggested an improved splitting criterion for top down decision tree induction known as DKM. Different authors have implemented DKM as a decision tree splitting criterion in imbalanced dataset and it has shown improved performance (Drummond & Holte (2000a), Flach (2003), Zadrozny & Elkan (2001)). However DKM has also been shown to be weakly skew insensitively (Drummond & Holte (2000a), Flach (2003)). Cieslak & Chawla (2008) proposed the use of Hellinger distance as a decision tree splitting criterion which they showed to be skew-insensitive. In the decision tree constuction algorithm, a feature is selected as a splitting attribute when it produces the largest Hellinger distance between two classes. Suppose (Θ, λ) is a measurable space, Θ is a non empty set and λ is a σ-algebra on Θ, such that P and Q are two continuous distributions with respect to the parameter λ. Let p and q be the densities of the measures P and Q with respect to λ. The definition of Hellinger distance can be given as:

$$d_H(P,Q) = \sqrt{\int_\Omega \left(\sqrt{P} - \sqrt{Q}\right)^2 d\lambda}.$$

They extended their work in Cieslak et al. (2012) by demonstrating that the Hellinger distance decision trees (HDDT) are robust in the presence of class imbalance and when combined with bagging they mitigate the need for sampling. They further showed that HDDTs are not significantly worse than C4.5 for balanced datasets. Therefore, it is sensible to use Hellinger distance over gain ratio even on balanced datasets. Drummond & Holte (2000a) showed that a combination of a cost insensitive splitting criterion such as DKM and cost sensitive pruning generally performs well.

4.3 The Binary Logistic Regression Model

Suppose y_i denotes a binary (Bernoulli) response variable. The logistic function is defined as (Maalouf & Trafalis, 2011)

$$E(y_i|\mathbf{x}_i, \boldsymbol{\beta}) = \frac{e^{\mathbf{x}_i'\boldsymbol{\beta}}}{1 + e^{\mathbf{x}_i'\boldsymbol{\beta}}} = p_i \tag{4.3.1}$$

where $\boldsymbol{\beta}$ is the vector of parameters with the assumption that $x_{i0} = 1$ so that $\boldsymbol{\beta}_0$ is a constant term and is included in the vector $\boldsymbol{\beta}$. Let $\{Y_1, Y_2, ..., Y_N\}$ denote a set of independent random variables each from an exponential family of distributions. The distribution of each Y_i has a canonical form and depends on a single parameter θ_i, that is

$$f(y_i; \theta_i) = \exp[y_i b_i(\theta_i) + c_i(\theta_i) + d_i(y_i)]$$

where b_i, c_i, d_i are function of θ_i. The distributions of all the Y_i's are of the same form. Therefore the joint density of $Y_1, Y_2, ..., Y_N$ is (as per Dobson & Barnett (2018))

$$f(y_1, y_2, ..., y_N; \theta_1, \theta_2, ..., \theta_N) = \prod_{i=1}^{N} \exp[y_i b(\theta_i) + c(\theta_i) + d(y_i)]$$

$$f(\mathbf{y}; \boldsymbol{\theta}) = \exp\Big[\sum_{i=1}^{N} y_i b(\theta_i) + \sum_{i=1}^{N} c(\theta_i) + \sum_{i=1}^{N} d(y_i)\Big]. \tag{4.3.2}$$

Suppose that $E(Y_i) = \mu_i$, where μ_i is some function of θ_i. For a generalized linear model, there is a transformation of μ_i such that

$$g(\mu_i) = \mathbf{x}_i^{'}\boldsymbol{\beta}.$$

The importance of this transformation is that $g(\mu_i)$ has many of the desirable properties of a linear regression model (Hosmer Jr et al., 2013). Consider a binary classification problem. Eq. 4.3.1 provides the conditional probability that $y_i = 1$ given $\mathbf{x}_i$. This is denoted by $p_i(\mathbf{x}_i)$. It follows that the quantity $1 - p_i(\mathbf{x}_i)$ gives the conditional probability that $y_i = 0$ given $\mathbf{x}_i$ (Hosmer Jr et al., 2013). All Y_i's have a Bernoulli distribution with the same parameters. Therefore, the joint probability distribution of $\boldsymbol{y}$ is given by

$$f(\boldsymbol{y}|\mathbf{x}_i, \boldsymbol{\beta}) = \prod_{i=1}^{N} p_i^{y_i}(1 - p_i)^{1-y_i}. \tag{4.3.3}$$

This distribution belongs to an exponential family since,

$$\begin{aligned}
\ln\Big(f(y_i|\mathbf{x}_i,\boldsymbol{\beta})\Big) &= \ln\Big(\prod_{i=1}^{N} p_i^{y_i}(1-p_i)^{1-y_i}\Big) \\
&= \sum_{i=1}^{N}\Big[\ln\Big(p_i^{y_i}(1-p_i)^{1-y_i}\Big)\Big] \\
&= \sum_{i=1}^{N}\Big[\ln(p_i^{y_i}) + \ln\big((1-p_i)^{1-y_i}\big)\Big] \\
&= \sum_{i=1}^{N}\Big[y_i\ln(p_i) + (1-y_i)\ln(1-p_i)\Big] \qquad (4.3.4)
\end{aligned}$$

$$f(y_i|\mathbf{x}_i,\boldsymbol{\beta}) = \exp\Big(\sum_{i=1}^{N} y_i \ln\Big(\frac{p_i}{1-p_i}\Big) + \sum_{i=1}^{N}\ln(1-p_i)\Big). \qquad (4.3.5)$$

Eq. 4.3.5 has the same form as Eq. 4.3.2 where

$$\eta_i = b(\theta_i) = \ln\Big(\frac{p_i}{1-p_i}\Big) = \mathbf{x}_i'\boldsymbol{\beta}$$

where η_i denotes the link function. Let $\boldsymbol{\beta}_{\max}$ denote the parameter vector for a saturated model, that is a model that fits the data perfectly, and $\boldsymbol{b}_{\max}$ denote the maximum likelihood estimator of $\boldsymbol{\beta}_{\max}$. Let $l(\boldsymbol{b};\boldsymbol{y})$ denote the maximum value of the likelihood function for the model of interest. The deviance is defined as (Dobson & Barnett, 2018)

$$D = 2[l(\boldsymbol{b}_{\max};\boldsymbol{y}) - l(\boldsymbol{b};\boldsymbol{y})]. \qquad (4.3.6)$$

For a saturated model, the maximum log-likelihood estimates are $\hat{p}_i = y_i$. From Eq. 4.3.4, the maximum value of the log-likelihood function is

$$l(\boldsymbol{b}_{\max};\boldsymbol{y}) = \sum_{i=1}^{N}\Big[y_i\ln(y_i) + (1-y_i)\ln(1-y_i)\Big]. \qquad (4.3.7)$$

For any other model with $p < N$ parameters, let $\hat{p}_i$ denote the maximum likelihood estimates for the probabilities and let $\hat{y}_i = \hat{p}_i$ denote the fitted model values. The log-likelihood function evaluated at these values is

$$l(\boldsymbol{b};\boldsymbol{y}) = \sum_{i=1}^{N}\Big[y_i\ln(\hat{y}_i) + (1-y_i)\ln(1-\hat{y}_i)\Big]. \qquad (4.3.8)$$

Substituting Eq. 4.3.7 and Eq. 4.3.8 into Eq. 4.3.6 gives

$$D = 2\sum_{i=1}^{N}\Big[y_i\ln\Big(\frac{y_i}{\hat{y}_i}\Big) + (1-y_i)\ln\Big(\frac{1-y_i}{1-\hat{y}_i}\Big)\Big]. \qquad (4.3.9)$$

For binary classification, the likelihood of a saturated model is equal to 1, $l(\boldsymbol{b}_{\max}; \boldsymbol{y}) = 0$ (Hosmer Jr et al., 2013). It then follows from Eq. 4.3.6 that the deviance is given by

$$D = -2l(\boldsymbol{b}; \boldsymbol{y}).$$

The log-likelihood in Eq. 4.3.4 can be written as (Minka, 2003)

$$l(\boldsymbol{\beta}) = \sum_{i=1}^{N} y_i \mathbf{x}_i' \boldsymbol{\beta} - \sum_{i=1}^{N} \ln\left(1 + e^{\mathbf{x}_i' \boldsymbol{\beta}}\right) - \frac{\lambda}{2}||\boldsymbol{\beta}||^2 \tag{4.3.10}$$

where $\frac{\lambda}{2}||\boldsymbol{\beta}||^2$ is a regularization or penalty term. For $\lambda > 0$, Eq. 4.3.10 gives a regularized estimate of $\boldsymbol{\beta}$ which often provides a good generalized performance, especially when the dimensionality is high (Nigam et al., 1999). The function in Eq. 4.3.10 is non-linear, maximizing it requires the maximum likelihood estimate (MLE) of $\boldsymbol{\beta}$. Minka (2003) showed that conjugate gradient method provides a better estimate of $\boldsymbol{\beta}$, since it guarantees convergence in at most d steps.

Fig 4.6 illustrates an example of binary logistic regression. The blue dotted line indicates the decision boundary. If the classification model returns a prediction probability of more than 0.5 for given $\mathbf{x}_i$, then the transaction would be classified as class 1, for example a fraudulent transaction. Alternatively, if the model returns a prediction probability of less than 0.5 for given $\mathbf{x}_i$, then the transaction will be classified as class 0, for example a non-fraudulent transaction. An increase above 0.5 in the decision boundary will cause an increase in the false negative rate (FNR), that is the model will predict the transactions as non-fraudulent whereas they are actual fraudulent. Alternatively, a decrease below 0.5 in the decision boundary will cause an increase in the FPR.

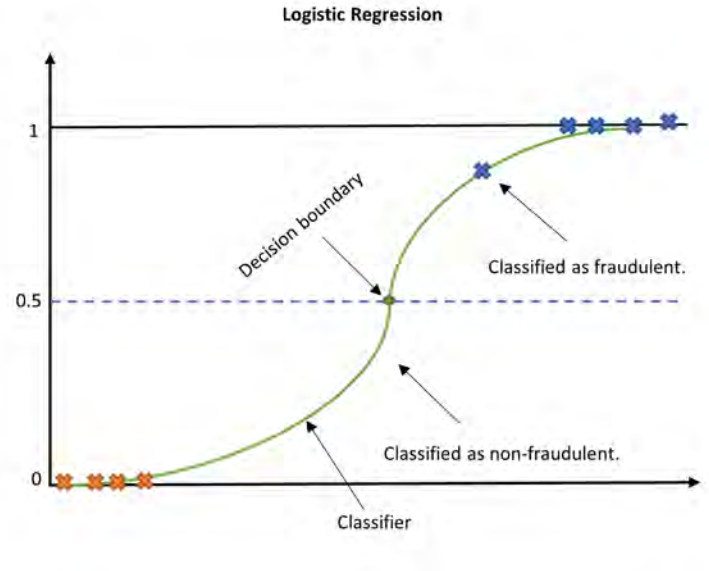


Figure 4.6: Demonstration of logistic regression (adapted from DeMaris (1995)).

4.3.1 Class Imbalance and Logistic Regression

The method used for computing the probability of Y_0, given $\mathbf{x}_0$, is a function of the maximum-likelihood estimate, $\hat{\boldsymbol{\beta}}$, namely

$$P(Y_0 = 1|\hat{\boldsymbol{\beta}}) = \frac{1}{1 + e^{-\mathbf{x}_0'\hat{\boldsymbol{\beta}}}} \tag{4.3.11}$$

and is statistically consistent (Firth, 1993). King & Zeng (2001) demonstrated that $\hat{\boldsymbol{\beta}}$ is a biased estimate of $\boldsymbol{\beta}$ in class imbalanced datasets and that it generates predicted probabilities that underestimate the actual probability of a rare event, for example fraudulent transactions. To correct the bias done to class imbalance, Manski & Lerman (1977) proposed the following weighted log likelihood

$$\begin{aligned} l(\boldsymbol{\beta}) &= \sum_{i=1}^{N} \frac{Q_i}{H_i}(y_i \ln p_i + (1 - y_i)\ln(1 - p_i)) \\ &= \sum_{i=1}^{N} w_i(y_i \ln p_i + (1 - y_i)\ln(1 - p_i)) \end{aligned} \tag{4.3.12}$$

where $w_i = \frac{Q_i}{H_i} = \left(\frac{\tau}{\bar{y}}\right)y_i + \left(\frac{1-\tau}{1-\bar{y}}\right)(1 - y_i)$, with $\bar{y}$ denoting the proportion of the events in the sample and τ the proportion of events in the population. King & Zeng (2001) proposed a small-sample correction to Eq. 4.3.12 which can make a difference when the probability of the event of interest is low.

4.4 Support Vector Machines

4.4.1 The Linearly Separable Case

Let $\mathbf{x}_i \in \mathbb{R}^d$ be a feature vector and $y_i \in \{-1, +1\}$ denote the corresponding class labels, where d is the dimension of the feature vector. For binary class classification, for example fraudulent and no-fraudulent transactions, we construct a function

$$\begin{aligned} f: \quad & \mathbb{R}^d \longrightarrow \mathbb{R} \\ & \mathbf{x}_i \to f(\mathbf{x}_i) = \begin{cases} +1, & \text{if transaction is fraudulent, or} \\ -1, & \text{otherwise} \end{cases} \end{aligned}$$

that predicts whether the new observation $\mathbf{x}_i$ belongs to the fraudulent class or to the non-fraudulent class. The optimal hyperplane is defined as

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b = 0$$

where $\mathbf{w}$ is a weight vector, $\mathbf{x}_i$ is the input feature vector and b is the bias (Huang et al., 2018). The support vector machine (SVM) model finds $\mathbf{w}$ and b such that the hyperplane separates the data and maximizes the margin between the classes, where the margin is given by

$$\text{Margin} = \frac{2}{||\mathbf{w}||}.$$

The weight vector, $\mathbf{w}$, and the bias, b, satisfy the following inequalities for all elements in the training set (Chi et al., 2008)

$$\begin{cases} \mathbf{w}^T\mathbf{x}_i + b \geq +1, & \text{if} \quad y_i = +1 \quad \text{and} \quad y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b) \geq +1 \\ \mathbf{w}^T\mathbf{x}_i + b \leq -1, & \text{if} \quad y_i = -1 \quad \text{and} \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq +1 \end{cases} \tag{4.4.1}$$

The objective of the learning phase in SVM is to maximize the margins between classes in the feature space. This is equivalent to (Chi et al., 2008)

$$\begin{cases} \min\limits_{\mathbf{w},b} & \frac{1}{2}||\mathbf{w}||^2 \qquad \text{subject to} \\ y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq +1, & \forall i \in \{1, 2, 3, ..., N\} \end{cases}. \tag{4.4.2}$$

Since Eq. 4.4.2 is an optimization problem over two constraints, $\mathbf{w}$ and b. The Lagrangian is used to remove the two constraints. The objective is to introduce the slack variable, α, and optimize the following Lagrangian function with respect to $\mathbf{w}$ and b.

$$\begin{aligned} L(\mathbf{w}, b, \boldsymbol{\alpha}) &= \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{N}\alpha_i\Big(y_i(\mathbf{w}\cdot\mathbf{x}_i + b) - 1\Big) \\ &= \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{N}\alpha_i y_i(\mathbf{w}\cdot\mathbf{x}_i + b) + \sum_{i=1}^{N}\alpha_i. \end{aligned} \tag{4.4.3}$$

Minimizing $L(\mathbf{w}, b, \boldsymbol{\alpha})$ with respect to $\mathbf{w}$ and b and setting to zero yields, the local minimum, that is

$$\begin{aligned}
\frac{\partial L(\mathbf{w}, b, \boldsymbol{\alpha})}{\partial \mathbf{w}} &= \frac{\partial}{\partial \mathbf{w}}\Big(\frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{N}\alpha_i y_i(\mathbf{w}\cdot\mathbf{x}_i + b) + \sum_{i=1}^{N}\alpha_i\Big) \\
&= \frac{\partial}{\partial \mathbf{w}}\Big(\frac{1}{2}||\mathbf{w}||^2\Big) - \frac{\partial}{\partial \mathbf{w}}\Big(\sum_{i=1}^{N}\alpha_i y_i(\mathbf{w}\cdot\mathbf{x}_i + b)\Big) + \frac{\partial}{\partial \mathbf{w}}\Big(\sum_{i=1}^{N}\alpha_i\Big) \\
\mathbf{0} &= \mathbf{w} - \sum_{i=1}^{N}\alpha_i y_i \mathbf{x}_i \\
\therefore \mathbf{w} &= \sum_{i=1}^{N}\alpha_i y_i \mathbf{x}_i
\end{aligned} \tag{4.4.4}$$

and

$$\begin{aligned}
\frac{\partial L(\mathbf{w}, b, \boldsymbol{\alpha})}{\partial b} &= \frac{\partial}{\partial b}\Big(\frac{1}{2}||\mathbf{w}||^2\Big) - \frac{\partial}{\partial b}\Big(\sum_{i=1}^{N}\alpha_i y_i(\mathbf{w}\cdot\mathbf{x}_i + b)\Big) + \frac{\partial}{\partial b}\Big(\sum_{i=1}^{N}\alpha_i\Big) \\
0 &= 0 - \sum_{i=1}^{N}\alpha_i y_i \\
\therefore \sum_{i=1}^{N}\alpha_i y_i &= 0.
\end{aligned} \tag{4.4.5}$$

Eqs. 4.4.4 and 4.4.5 suggest that the local minimum falls when $\mathbf{w} = \sum_{i=1}^{N}\alpha_i y_i \mathbf{x}_i$ and $\sum_{i=1}^{N}\alpha_i y_i = 0$. Substituting 4.4.4 and 4.4.5 into 4.4.3 yields

$$\begin{aligned}
L_{\max}(\mathbf{w}, b, \boldsymbol{\alpha}) &= \frac{1}{2}\Big(\sum_{i=1}^{N}\alpha_i y_i \mathbf{x}_i\Big)^2 - \sum_{i=1}^{N}\alpha_i y_i\Big(\sum_{j=1}^{N}\alpha_j y_j \mathbf{x}_j \cdot \mathbf{x}_i + b\Big) + \sum_{i=1}^{N}\alpha_i \\
&= \frac{1}{2}\Big(\sum_{i=1}^{N}\alpha_i y_i \mathbf{x}_i\Big)\Big(\sum_{j=1}^{N}\alpha_j y_j \mathbf{x}_j\Big) - \sum_{i=1}^{N}\alpha_i y_i \sum_{j=1}^{N}\alpha_j y_j \mathbf{x}_j \cdot \mathbf{x}_i - b\sum_{i=1}^{N}\alpha_i y_i + \sum_{i=1}^{N}\alpha_i \\
&= \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j(\mathbf{x}_i\cdot\mathbf{x}_j) - \sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j(\mathbf{x}_i\cdot\mathbf{x}_j) - b\sum_{i=1}^{N}\alpha_i y_i + \sum_{i=1}^{N}\alpha_i \\
&= \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j(\mathbf{x}_i\cdot\mathbf{x}_j) - \sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j(\mathbf{x}_i\cdot\mathbf{x}_j) + \sum_{i=1}^{N}\alpha_i \\
&= \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{N}\alpha_i\alpha_j y_i y_j(\mathbf{x}_i\cdot\mathbf{x}_j)
\end{aligned} \tag{4.4.6}$$

In this context, the dual objective function is, $\max_{\alpha}\Big(\sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{N}\alpha_i\alpha_j y_i y_j\langle \mathbf{x}_i, \mathbf{x}_j\rangle\Big)$, subject to $\sum_{i=1}^{N}\alpha_i y_i = 0,\ \alpha_i \geq 0\ \ \forall i \in \{1,2,3,...,N\}$ where $\langle\cdot,\cdot\rangle$ is the inner product. The decision function for classifying a new observation $\mathbf{x}$ is

$$f(\mathbf{x}) = \text{sign}(\mathbf{w}^T\mathbf{x} + b) = \text{sign}\Big(\sum_{i=1}^{N}\alpha_i y_i\langle \mathbf{x}_i, \mathbf{x}\rangle + b\Big).$$

4.4.2 The Linearly Non-Separable Case

In most real world problems, the datasets are not completely linearly separable even thought they are mapped into a higher dimensional space (Ma & He, 2013). Therefore, the constraint in Eq. 4.4.1 is violated. To overcome this problem, Eq 4.4.1 is relaxed by introducing a set of slack variables $\xi_i \geq 0$. The soft margin optimization problem is then formulated as follows

$$\begin{cases} \min\limits_{\mathbf{w},b}\Big(\frac{1}{2}||\mathbf{w}||^2 + C\sum\limits_{i=1}^{N}\xi_i\Big), & \text{subject to} \\ y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1 - \xi_i & \\ \text{and} \quad \xi_i \geq 0, & \text{for} \quad i \in \{1,2,3,...,N\} \end{cases} \tag{4.4.7}$$

where $\sum_{i=1}^{N}\xi_i$ is a measure of total misclassifications and C denotes a misclassification cost (Ma & He, 2013). For $0 \leq \xi_i < 1$, the data points fall inside the region of separation but on the right side of the decision surface. For $\xi_i > 1$, they fall on the wrong side of the decision surface (Liu & Huang, 2002). For very large C, that is $\lim\limits_{C\to\infty} L(\mathbf{w}, b, \boldsymbol{\alpha}, \xi, \delta)$, the penalty for misclassifying points is very high, so the decision boundary will perfectly separate the data if possible. For very small C, that is $\lim\limits_{C\to 0} L(\mathbf{w}, b, \boldsymbol{\alpha}, \xi, \delta)$, the classifier can maximize the margin between most of the points while misclassifying few points, since there is small penalty (Weston, 2014). From Eq. 4.4.7, the Lagrangian equation is

$$L(\mathbf{w}, b, \boldsymbol{\alpha}, \xi, \delta) = \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i\Big[y_i(\mathbf{w}\cdot\mathbf{x}_i + b) - 1 + \xi_i\Big] - \sum_{i=1}^{N}\delta_i\xi_i \tag{4.4.8}$$

The derivative of Eq. 4.4.8 with respect to $\mathbf{w}$ and b will result in Eq. 4.4.4 and Eq. 4.4.5 respectively. Taking the derivative of $L(\mathbf{w}, b, \boldsymbol{\alpha}, \xi, \delta)$ with respect to δ_i and ξ_i respectively, gives

$$\frac{\partial L(\mathbf{w}, b, \boldsymbol{\alpha}, \xi, \delta)}{\partial \delta_i} = 0 \implies \xi_i = 0 \tag{4.4.9}$$

and

$$\frac{\partial L(\mathbf{w}, b, \boldsymbol{\alpha}, \xi, \delta)}{\partial \xi_i} = C - \alpha_i - \delta_i$$
$$0 = C - \alpha_i - \delta_i \implies C = \alpha_i + \delta_i. \tag{4.4.10}$$

Substituting Eq. 4.4.9 and Eq. 4.4.10 into Eq. 4.4.8 results in Eq. 4.4.6. Eq. 4.4.7 is called the primal form of optimization. The dual form, which gives the same optimal solution (Ma & He, 2013), is $\max_{\alpha} \Big(\sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \Big)$, subject to $\sum_{i=1}^{N} \alpha_i y_i = 0$, $\alpha_i \geq 0 \quad \forall i \in \{1, 2, 3, ..., N\}$. The decision function for classifying a new observation $\mathbf{x}$ is still

$$f(\mathbf{x}) = \text{sign}\Big(\sum_{i=1}^{N} \alpha_i y_i \langle \mathbf{x}_i, \mathbf{x} \rangle + b \Big) \tag{4.4.11}$$

Fig. 4.7 shows the linear SVM binary classifier. For linearly separable data, the classifier perfectly classifies the dataset without any misclassifications. For non-linearly separable data, the classifier fails to perfectly separate the classes and it violates the constraint in Eq. 4.4.1.

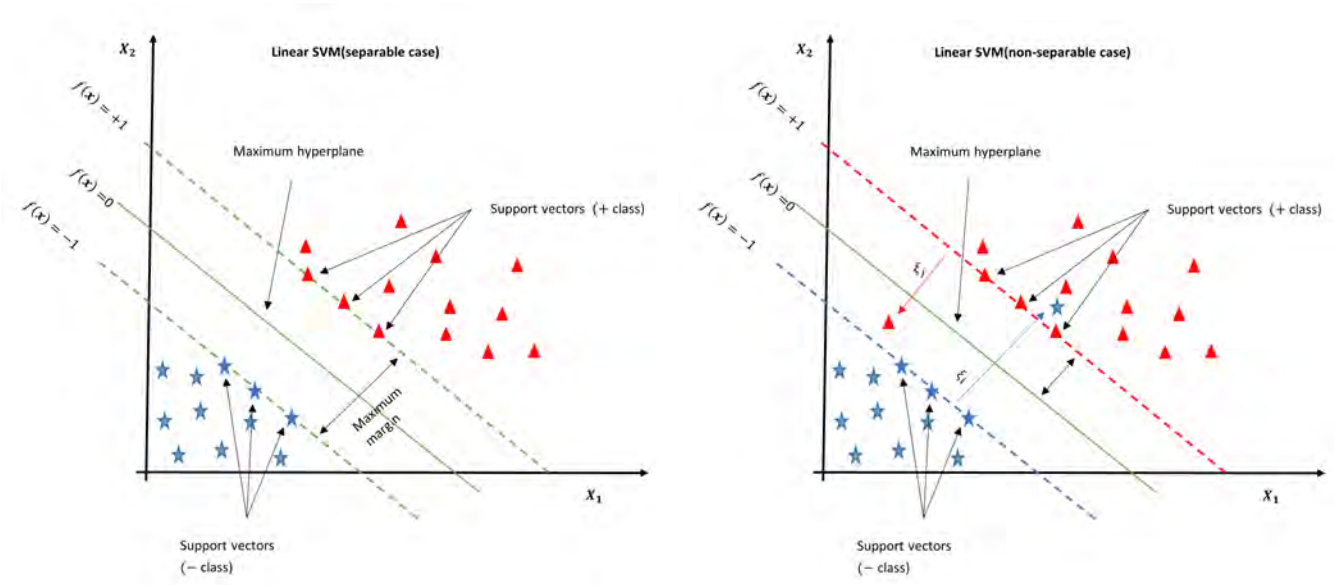


Figure 4.7: Linear SVM, binary classification (adapted from Huang et al. (2018)).

4.4.3 The Soft Margin Error Cost

Consider the objective function in Eq. 4.4.7. The first term of the objective function focuses on maximizing the margin, while the second term attempts to minimize the penalty term associated with the misclassifications. Eq. 4.4.7 assumes the same misclassification cost for both classes, for example fraudulent and non-fraudulent transactions (Palade,

2013). In imbalanced data, the low presence of positive observations makes them appear further from the optimal class boundary than the negative observations. This can be solved by shifting the hyperplane towards the minority class, but this shift can cause more false negative predictions and lower the models performance on the minority class (Ma & He, 2013). Veropoulos et al. (1999) proposed cost sensitive learning. In this method, the objective function in Eq. 4.4.7 is modified to assign C^+ to the misclassification cost for positive class and C^- to the misclassification cost of a negative class. Therefore, Eq. 4.4.7 is rewritten as $\min_{\mathbf{w},b}\Big(\frac{1}{2}||\mathbf{w}||^2 + C^+\sum_{i=1}^{N}\xi_i + C^-\sum_{i=1}^{N}\xi_i\Big)$, subject to $y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1-\xi_i$ and $\xi_i \geq 0$, for $i \in \{1,2,3,...,N\}$. The effect of class imbalance is reduced by assigning a higher misclassification cost to the minority class than the majority class (Veropoulos et al., 1999). It then follows that the Lagrangian form of the modified objective function is $\max_{\mathbf{w},b}\Big(\sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{N}\alpha_i\alpha_j y_i y_j\langle\mathbf{x}_i,\mathbf{x}_j\rangle\Big)$, subject to $\sum_{i=1}^{N}\alpha_i y_i = 0$, $0 \leq \alpha_i^+ \leq C^+$ and $0 \leq \alpha_i^- \leq C^-$ where α_i^+ and α_i^- represent the Lagrangian multipliers of positive and negative class respectively.

4.4.4 The Kernel Transformation

Linear classifiers cannot separate the class well when the data is intrinsically non-linear (Palade, 2013). A general approach is to map the data points onto a higher dimensional space where the linearly non-separable data in the original feature space becomes linearly separable (Zhou, 2019). However, the learning process may be very slow since the inner product will be difficult to calculate in the higher dimensional space (Schölkopf, 2001). Boser et al. (1992) proposed the use of kernel functions as a solution to this problem. According to Mercer's theorem every positive semi-definite symmetric function is a kernel (Cristianini & Shawe-Taylor, 2000). Let $\phi(\cdot)$ be a function that maps the data from a input space into a feature space, that is

$$\phi : \mathbf{x}_i \in \mathbb{R}^d \longrightarrow \phi(\mathbf{x}_i) \in \mathbb{F} \subseteq \mathbb{R}^d$$

where d is the dimension of the feature space. The goal is to choose the mapping ϕ that converts non-linear relations between the response variable and the feature variables into linear relations (Maalouf & Trafalis, 2011). The kernel is related to the transformation $\phi(\mathbf{x}_i)$ by the equation (Zhang & Wu, 2012)

$$K(\mathbf{x}_i,\mathbf{x}_j) = \Big\langle\phi(\mathbf{x}_i),\phi(\mathbf{x}_j)\Big\rangle$$

and $\mathbf{w}$ is also transformed into feature space where

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \phi(\mathbf{x}_i).$$

Eq. 4.4.11 is rewritten as

$$f(\mathbf{x}) = \text{sign}\Big(\sum_i \alpha_i y_i \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}) \rangle + b\Big) = \text{sign}\Big(\sum_i \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b\Big).$$

The use of the kernel function makes it possible to compute the separating hyperplane without explicitly carrying out the mapping into feature space (Howley & Madden, 2005). The following are the most commonly used kernel functions (Mitchell, 2011):

i. Linear kernel: $K(\mathbf{x}_i, \mathbf{x}) = \Big\langle \mathbf{x}_i, \mathbf{x} \Big\rangle$;

ii. Polynomial kernel: $K(\mathbf{x}_i, \mathbf{x}) = \Big(\Big\langle \mathbf{x}_i, \mathbf{x} \Big\rangle\Big)^d$;

iii. Radial basis function kernel: $K(\mathbf{x}_i, \mathbf{x}) = \exp\Big(- \gamma ||\mathbf{x}_i - \mathbf{x}||^2 + C\Big)$;

iv. Sigmoid kernel: $K(\mathbf{x}_i, \mathbf{x}) = \tanh\Big(\gamma \Big\langle \mathbf{x}_i, \mathbf{x} \Big\rangle + r\Big)$;

where d denotes the degree of a polynomial, C the cost and r the coefficient. The linear kernel is equivalent to a first-degree polynomial kernel and corresponds to the original input space. Each kernel corresponds to a feature space, and since no explicit mapping takes place with that feature space, the optimal linear separators can be found efficiently in the feature spaces with millions of dimensions (Howley & Madden, 2005). An alternative to using one of these default kernels is to derive a custom kernel suitable for a particular problem (Mitchell, 2011).

4.5 Multilayer Perceptron

The multilayer perceptron is the most widely used model in neural network applications using back-propagation learning algorithm (Ruck et al., 1990). MLP is a predictive neural network model that maps a set of input data to an appropriate set of outputs. It connects multiple layers in a directed graph, that is the node's signal path only goes in one direction. Weights and output signals interconnect nodes; a function of the sum of node inputs is modified by a simple nonlinear transfer function or activation. The node's output is scaled by the connecting weight and fed forward as an input to the nodes in the next layer of the network (McGovern, 2016). The definition of architecture in a MLP nueral network is crucial since

lack of connectivity can make the network incapable of solving the problem of insufficient adjustable parameters. On the other hand, an excess of connections may cause the model to overfit (Ramchoun et al., 2016). MLP uses different loss functions depending on the problem type. The most commonly used loss function for classification is cross-entropy (CE), also known as the log loss function (Pedregosa et al., 2011) which is defined as

$$\mathrm{CE}(y, \hat{y}) = -\sum_{j=1}^{M} y_{ij} \log(\hat{y}_{ij}) \tag{4.5.1}$$

where M denotes the number of class labels, y_{ij} a binary indicator of whether or not the j^{th} label is the correct classification of the i^{th} observation and $\hat{y}_{ij}$ is the model probability of assigning j^{th} class label to the i^{th} observation. CE quantifies the accuracy of a classifier by penalizing misclassification. For perfect classification, CE is zero. For binary classification, Eq. 4.5.1 can be rewritten as follows

$$CE(y, \hat{y}, \mathbf{W}) = -y \ln(\hat{y}) - (1 - y) \ln(1 - \hat{y}) + \alpha ||\mathbf{W}||_2^2$$

where $\alpha ||\mathbf{W}||_2^2$ is an L2-regularization term, that is a penalty term, that penalizes complex models and $\alpha > 0$ is the hyper-parameter that controls the magnitude of the penalty. MLP randomly initializes the weights and repeatedly updates them to minimize the loss function. The input vector is propagated through the network to obtain an output and an error is calculated by comparing the actual and predicted output. This error is propagated back through the network and weights are adjusted until the overall error is satisfactorily small. This process is called back-propagation. In gradient descend, the gradient $\nabla CE_{\mathbf{W}}$ of the loss function with respect to the weights is computed as follows

$$\mathbf{W}^{i+1} = \mathbf{W}^{i} - \epsilon \nabla CE^{i}_{\mathbf{W}}$$

where ϵ denotes the learning rate and i the iteration step (Pedregosa et al., 2011). This weight is updated until the algorithm reaches the maximum number of iterations or an improvement in the loss function is sufficiently small.

Fig. 4.8 shows a two hidden layer MLP. The input layer consist of a set of neurons $\{x_i|x_1, x_2, ..., x_d\}$ representing the input features. Each neuron in the hidden layer transforms the values from the previous layer with a weighted linear summation $w_1x_1 + w_2x_2 + ... + w_dx_d$, followed by a non-linear activation function. The output layer receives the values from the last hidden layer and transforms them into output values.

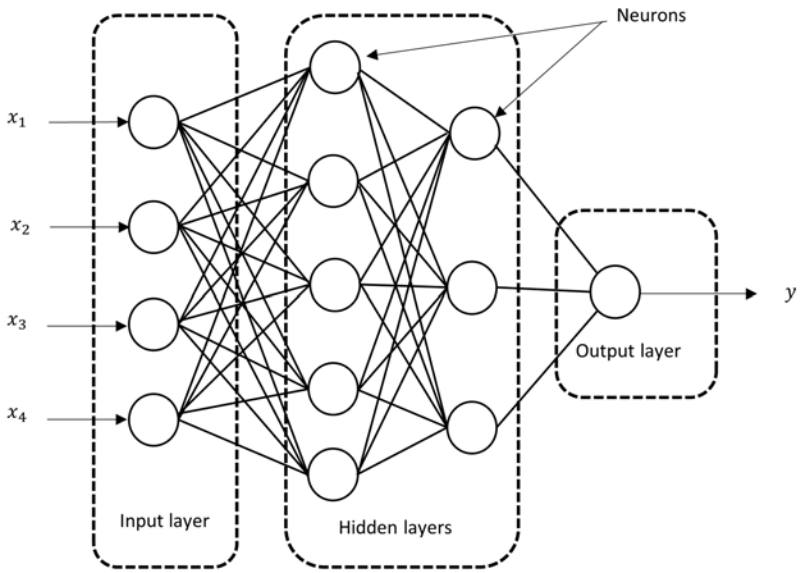


Figure 4.8: Illustration of multilayer perceptron with 2 hidden layers (adapted from McGovern (2016)).

4.6 Chapter 4 Summary

Many data mining classifiers are built to handle balanced dataset and thus the cost of misclassifying each observation is the same. Since the misclassification cost is the same, this suggest that the data mining classifiers should be adjusted to handle imbalance dataset since the misclassification cost is not the same. For example, in the brain cancer dataset, the cost of misclassifying a patient as healthy is heavy than the cost of misclassifying the patient as ill. In this chapter, the theory of each classifier is discussed and several suggestions on how the classifies can mitigate the bias due to the majority class are presented.

Chapter 5

Binary Classifier Performance Evaluation

Section 5.1 introduces binary classification performance metrics. Sections 5.2 to 5.6 of this chapter introduces different metrics that can be used to measure the performance of a binary classifier when dealing with imbalanced dataset. Section 5.7 discusses model validation which is the task of verifying that the classifiers are performing as expected.

5.1 Introduction

Suppose the positive class has N^+ observations and the negative class has N^- observations such that $N = N^+ + N^-$ and assume that the positive class is the minority class, thus, $N^+ < N^-$. Let f denote the binary classifier which is defined as

$$f : \mathbb{R}^d \longrightarrow \mathbb{R}$$

$$\mathbf{x} \mapsto f(\mathbf{x}) = \begin{cases} +1, & \text{for example if transaction is fraudulent, and} \\ -1, & \text{otherwise} \end{cases}$$

and $\mathbb{I}(\cdot, \cdot)$ denotes an indicator function, defined as

$$\mathbb{I}(y_i, f(\mathbf{x}_i)) = \begin{cases} +1, & \text{if} \quad y_i = f(\mathbf{x}_i) \\ -1, & \text{if} \quad y_i \neq f(\mathbf{x}_i). \end{cases}$$

The confusion matrix is given by

	Actual positive	Actual negative
Predicted as positive	TP	FP
Predicted as negative	FN	TN

where TP denotes a true positive, TN denotes a true negative, FP denotes a false positive, FN denotes a false negative and are computed as per table 5.1

Table 5.1: Confusion matrix for a binary classifier.

	Actual positive	Actual negative
Predicted as positive	$\text{TP}=\sum_{i=1}^{N}\mathbb{I}(y_i=+1, f(\mathbf{x}_i)=+1)$	$\text{FP}=\sum_{i=1}^{N}\mathbb{I}(y_i=-1, f(\mathbf{x}_i)=+1)$
Predicted as negative	$\text{FN}=\sum_{i=1}^{N}\mathbb{I}(y_i=+1, f(\mathbf{x}_i)=-1)$	$\text{TN}=\sum_{i=1}^{N}\mathbb{I}(y_i=-1, f(\mathbf{x}_i)=-1)$

In this context

$$\begin{aligned} N^+ &= \sum_{i=1}^{N}\mathbb{I}(y_i=+1, f(\mathbf{x}_i)=+1) + \sum_{i=1}^{N}\mathbb{I}(y_i=+1, f(\mathbf{x}_i)=-1) \\ &= \text{TP} + \text{FN} \end{aligned}$$

and

$$\begin{aligned} N^- &= \sum_{i=1}^{N}\mathbb{I}(y_i=-1, f(\mathbf{x}_i)=+1) + \sum_{i=1}^{N}\mathbb{I}(y_i=-1, f(\mathbf{x}_i)=-1) \\ &= \text{FP} + \text{TN}. \end{aligned}$$

The percentage accuracy of the classifier is given by

$$\begin{aligned} \text{Accuracy} &= P(f(\mathbf{x})=y) \\ &= \frac{\sum_{i=1}^{N}\mathbb{I}(y_i=+1, f(\mathbf{x}_i)=+1) + \sum_{i=1}^{N}\mathbb{I}(y_i=-1, f(\mathbf{x}_i)=-1)}{N} \\ &= \frac{\text{TP}+\text{TN}}{N}. \end{aligned}$$

The error rate of the classifier is given by

$$\begin{aligned} \text{Error rate} &= P(f(\mathbf{x})\neq y) \\ &= \frac{\sum_{i=1}^{N}\mathbb{I}(y_i=+1, f(\mathbf{x}_i)=-1) + \sum_{i=1}^{N}\mathbb{I}(y_i=-1, f(\mathbf{x}_i)=+1)}{N} \\ &= \frac{\text{FN}+\text{FP}}{N}. \end{aligned}$$

Alternatively, the error rate is given by error rate $= 1 -$ accuracy. The accuracy and error rate are not adequate classifier performance metrics for imbalance classifier performance. In these instances, more attention is paid to the minority class (Zhou, 2019).

5.2 ROC Curve and AUC

A receiver operating curve (ROC) is a graphical plot that illustrates the performance of a binary classifier system as its discrimination threshold is varied. By considering all possible values of the threshod, t, a ROC curve can be constructed as a plot of sensitivity (Eq. 5.2.1) versus 1−specificity (Eq. 5.2.2) (Calì & Longobardi, 2015). ROC curves are robust against changes to class distributions (Tharwat, 2020). If the ratio of positive to negative samples changes in a test set, the ROC curve will not change. In other words, ROC curves are insensitive to imbalanced data. This is because ROC curves depend on the true positive rate (TPR) and false-positive rate (FPR) (Tharwat, 2020), which are defined as

$$\text{TPR} = \frac{TP}{TP + FN} = \frac{TP}{N^+} \tag{5.2.1}$$

and

$$\text{FPR} = \frac{FP}{FP + TN} = \frac{FP}{N^-}. \tag{5.2.2}$$

Let

$$\begin{aligned} g :&\mathbb{R}^d \longrightarrow \mathbb{R} \\ &\mathbf{x} \longrightarrow [0, 1] \end{aligned}$$

denote a probabilistic classifier, thus g assigns a probability score for each input variable. In most cases, the ROC curve provides a quantitative result, $g(\mathbf{x})$. Therefore, it is crucial to define a threshold to group transactions into classes, for example fraudulent or non-fraudulent transactions. We classify a transaction as fraudulent if $g(\mathbf{x}) \geq t$ or non-fraudulent if $g(\mathbf{x}) < t$ where t denotes the threshold. For example, consider $t = 0.5$. If a classifier returns a predictive value of $g(\mathbf{x}) = 0.67$ for a given input variable $\mathbf{x}$ then that transaction will be classified as fraudulent since $g(\mathbf{x}) > t$. The area under ROC curve (AUC) is the measure of quality of a probabilistic classifier and is defined as (Vuk & Curk, 2006)

$$\begin{aligned} \text{Area}_{\text{ROC}} &= \int_0^1 \frac{TP}{N^+} \, d\frac{FP}{N^-} \\ &= \frac{1}{N^+ N^-} \int_0^{N^-} TP \, dFP. \end{aligned}$$

A random classifier has a AUC of 0.5 while a perfect classifier has a AUC of 1. Consider Fig. 5.1(a). The point $(0, 0)$ represents a classifier with no positive classifications, while all negative observations are correctly classified and hence TPR=0 and FPR=0. The point $(1, 1)$ represents a classifier where all positive observations are correctly classified while the negative observations are misclassified. The point $(1, 0)$ represents a classifier where all positive and negative samples are misclassified. The point$(0, 1)$ represents a classifier where all positive

and negative samples are correctly classified; thus, this point represents perfect classification. Fig. 5.1(b) shows the relationship between sensitivity and specificity. The threshold, indicated by the black dotted line in the centre of the graph, is where the sensitivity and specificity are the same. If the threshold is moved to the left, the sensitivity increases and the specificity decreases. At line A, sensitivity reaches its maximum value of 100% since we have no false negatives, that is no fraudulent transactions were classified as non-fraudulent. If the threshold is moved to the right, specificity increases and sensitivity decreases. At line B, specificity is 100% since we have no false positives, that is no non-fraudulent transactions were classified as fraudulent.

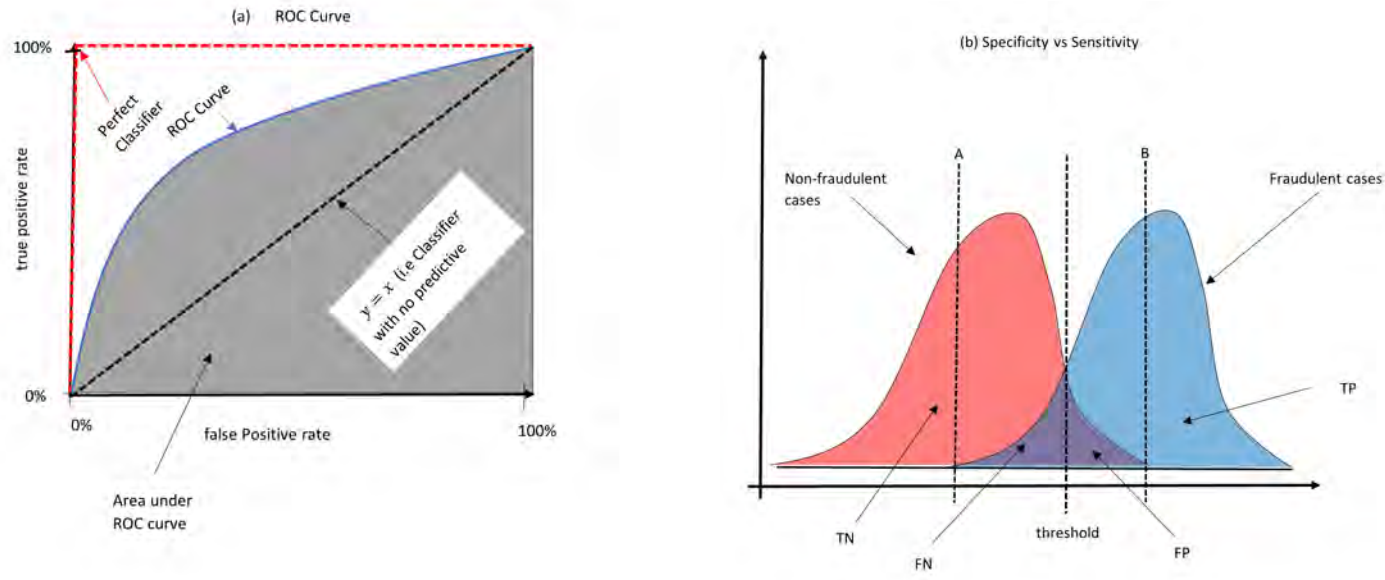


Figure 5.1: Demonstration of ROC curve, AUC and threshold (adapted from Chawla et al. (2002)).

5.3 The Geometric Mean

Geometric mean, or G-mean, is a metric that can be used to measure the balance between a classifiers performance on the majority and minority classes. A poor performance in predicting the positive observations will lead to a low G-mean value, even if the negative observations are correctly classified (Hido et al., 2009). The G-mean metric is defined as

$$\text{G-mean} = \sqrt{\frac{TP}{N^+} \times \frac{TN}{N^-}}.$$

The G-mean minimizes the negative influence of the skewed distribution of classes. It does not show the contribution of each class to the overall performance, nor which is the prevalent class. This means that different combinations of TPR and TNR may produce the same result for this metric (Bekkar et al., 2013).

5.4 Precision and Recall

Precision and recall curves (PR) are a common way to measure performance in highly imbalanced dataset (Goadrich et al., 2006). These two evaluation metrics focus on the correct classification of the positive observations. Precision measures how many observations that are classified as positive are actually positive, while recall measures how many positive observations are correctly classified as positive. Recall is defined as

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

and precision as

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FN}}.$$

By definition, precision does not contain any information about FN, and recall does not contain any information about FP. Therefore, neither provides a complete evaluation of classifier performance, while they are complementary to each other. Though a high precision and a high recall are desired, there are often conflicts to achieve the two goals together since FP usually becomes larger when TP increases (Zhou, 2019).

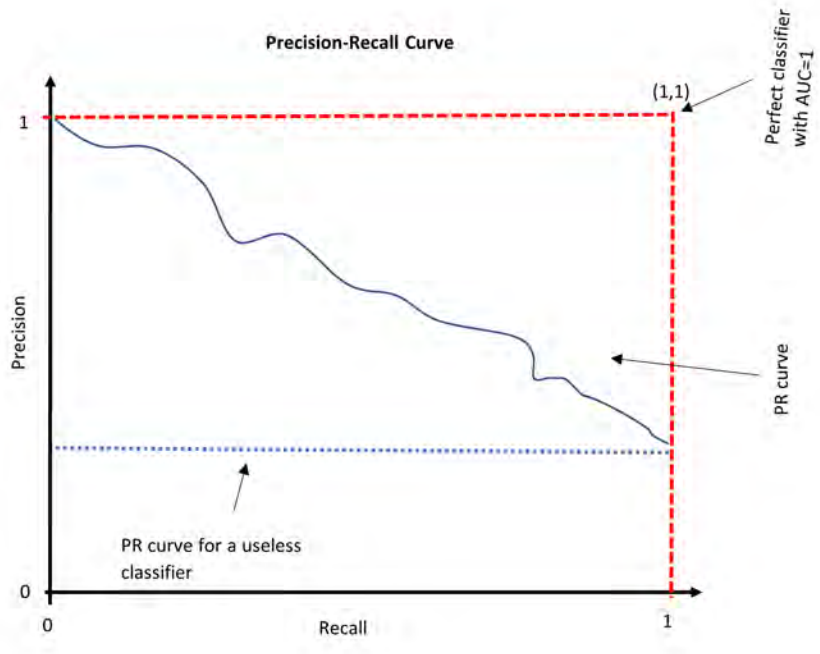


Figure 5.2: Illustration of the Precision-Recall curve (adapted from Zhou (2019)).

Fig. 5.2 illustrates the tradeoff between precision and recall. The point $(1, 1)$ represents a perfect classifier, where high precision relates to a low false-positive rate, and high recall relates to a low false-negative rate. The high value of the area under the curve represents

a high precision and recall. High precision and recall show that the classifier is returning accurate results.

5.5 F-measure and β Varied F-measure

The F-measure is the harmonic mean of precision and recall which is defined as (Sokolova et al., 2006)

$$\mathrm{F} = 2 \times \frac{\text{recall} \times \text{precison}}{\text{recall} + \text{precison}}. \tag{5.5.1}$$

It is worth noting that the F-measure will be zero if recall is zero or precision is zero and F$\in [0, 1]$. F increases proportionally with the increase of precision and recall; a high value of F indicates that the model performs better on the positive class. If the F =1, this means that the classification model is perfect. The F-measure can be written in a more general manner, called the β varied F-measure, which is defined as

$$F_\beta = \frac{(1 + \beta^2) \times \text{recall} \times \text{precision}}{\beta^2 \times \text{recall} + \text{precision}}$$

where $\beta \in \mathbb{R}^+$. β is adjusted according to the relative importance of recall vs precision (Sokolova et al., 2006). In Eq. 5.5.1, β is assumed to be 1. Decreasing β leads to a reduction of precision importance. The rationale behind the β varied F-measure is that misclassification within the minority class is often more expensive than misclassification of majority examples; consequently improving the recall will affect the F-measure more than the precision (Bekkar et al., 2013).

5.6 Matthews Correlation Coefficient

The Matthews correlation coefficient (MCC) measures the correlation between actual values y and predicted values $\hat{y}$. It is defined as (Chicco & Jurman, 2020)

$$\begin{aligned}\mathrm{MCC} &= \frac{Cov(y, \hat{y})}{\sigma_y \times \sigma_{\hat{y}}} \\ &= \frac{\mathrm{TP} \times \mathrm{TN} - \mathrm{FP} \times \mathrm{FN}}{\sqrt{(\mathrm{TP} + \mathrm{FP}) \times (\mathrm{TP} + \mathrm{FN}) \times (\mathrm{TN} + \mathrm{FP}) \times (\mathrm{TN} + \mathrm{FN})}}.\end{aligned}$$

MCC is a binary classification measure that assigns a high score if the model can correctly classify most positive and negative observations (Jurman et al., 2012). The MCC score ranges from -1 to +1, MCC score $\in [-1, 1]$, where a score of -1 indicates perfect misclassification and a score of $+1$ indicates a perfect classification. MCC score of 0 means that the classifier is no better than a random flip of a fair coin. However, suppose the MCC score is exactly

equal $-1, 0$ or $+1$. In these cases, it is not a reliable indicator of how similar a predictor is to a random guessing since MCC is dependent on the dataset (Chicco et al., 2021). MCC is a single performance measure less influenced by imbalanced dataset, that is it is insensitive to class imbalance.

5.7 Model Validation

5.7.1 The Validation Set Approach

As mentioned in section 3.4.4, direct estimation of the generalization error by the training error is excessively optimistic, more special when the model is complex. When model complexity is low and the training set is very large, a generalized lower limit $B(N, \mathcal{F})$ (sufficiently small) is defined, and the expected risk can be obtained as

$$R(f^*_{D_N}) \leq R_{D_N}(f^*_{D_N}) + B(N, \mathcal{F}).$$

To estimate the test error associated with fitting a statistical model, the dataset D_N is randomly split into D_{train} and D_{test}. The model is fit on D_{train} and the fitted model is used to predict the response for the observations in D_{test}. The resulting test set error rate provides an estimate of the actual test error rate. Training only on the part of the available dataset, that is D_{train}, can have serious disadvantages when N is small or D_N is an imbalanced dataset (James et al., 2013a). Firstly, since one of the classes is relatively rare in D_N, further decreasing the number of observations belonging to the rare class has a negative impact on the quality of the resulting model. Secondly, the test estimate of the actual test error can be highly variable depending precisely on which observations are included in D_{train} and which observations are included in D_{test}. The classification model tends to perform worse when trained on few observations, suggesting that the test set error rate may overestimate the actual test error rate for the model fit on D_N.

5.7.2 Cross Validation

5.7.2.1 k-fold Cross Validation

In k-fold cross validation (k-fold CV), D_N is partitioned into k disjoint subsets of approximately equal sizes. This partitioning is done by randomly sampling D_N without replacement. The model is evaluated on one of the partitions where the other $k-1$ folds are used for training. This procedure is repeated until each of the k subsets has served as a test set, each fold serves as a test set only once. The average of the k performance measurements on the k test sets is the cross validated performance. In a more general setup, let $\hat{f}_{-k}(\mathbf{x}_i)$ denote the model that wasn't trained on the k^{th} subset of D_N. The value of $\hat{f}_{-k}(\mathbf{x}_i)$ is the predicted

value for the observation y_i which is an element of the k^{th} subset. The k-fold CV estimate of the prediction error is (James et al., 2013a)

$$\text{Err}_{(CV)} = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(y_i, \hat{f}_{-k}(\mathbf{x}_i)).$$

CV often involves stratified random sampling. This sampling technique is an unbiased estimate of the population proportion which means that when D_N is randomly partitioned the class proportions in the individual subsets reflects the proportions in D_N. To avoid a biased evaluation, subsets that are used for evaluating the model should reflect this class ratio (Kohavi, 1995).

Fig. 5.3 depicts how $k = 10$ fold CV works. In this figure, D_N is randomly split in 10 disjoint subsets. Each subset contains approximately 10% of D_N. The classification model is trained on the training set, consisting of 9 combined folds of data, and subsequently tested on the test set, the fold not included in the training set. On each fold, the classification model's performance is measured, for example using recall, precision, geometric mean, etc., on the test set. This process is repeated until all 10 folds have served as the test set. The average of these 10 folds performance provides the classifiers final performance score, for example $\text{Average}_{recall} = \frac{1}{10}\sum_{j=1}^{10}\text{recall}_j$.

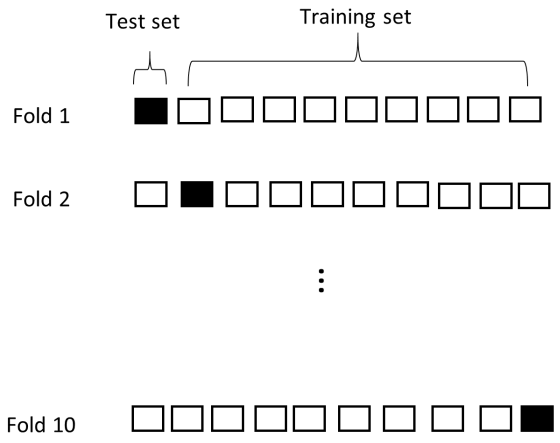


Figure 5.3: Illustration of k-fold cross validation (adapted from Terrible (2017)).

5.7.2.2 Leave One Out Cross Validation

Leave-one-out cross validation (LOOCV) is a special case of k-fold CV where $k = N$. In LOOCV, $N-1$ observations are used to train the model and only one observation is used for

testing. As there are N possible partitions the classifier is trained N times. The test error in LOOCV is approximately an unbiased estimate of the true prediction error, but it has high variance since the N training sets are the same, in that two different training sets only differ by one case (Friedman et al., 2001). The bias for LOOCV is lower compared to k-fold CV. The advantage of using k-fold CV is that it is less computational expensive than LOOCV, especially when N is large, and has lower variability (Berrar, 2018).

5.7.3 The Bootstrap

The bootstrapping technique is the most widely used statistical tool to quantify uncertainty associated with a given estimator or statistical learning method (James et al., 2013a). The idea is to randomly draw samples, with replacement, from the training set, where each randomly selected sample has the same size as the original training set. This procedure is repeated B times to produce B bootstrap datasets. For each bootstrap, the model is fit and the behavior of the fits over B replications is examined. The bootstrap technique has other important advantages besides providing a more accurrate point estimate for prediction error, it provides direct assessment of variability for estimated parameters in the prediction rule (Efron & Tibshirani, 1997).

5.7.3.1 Bias and Standard Error

Let $\hat{\theta} = s(\mathbf{X})$ denote an estimator for θ where $\mathbf{X}$ denotes the training dataset and $\theta = t(P)$ denotes some parameter of the distribtuion. A bootstrap estimate of the standard error can be obtained as follows: draw B independent bootstrap samples $\mathbf{X}^{*(1)}, \mathbf{X}^{*(2)}, ..., \mathbf{X}^{*(B)}$ from $\hat{P}$ where

$$\mathbf{X}_1^{*(b)}, \mathbf{X}_2^{*(b)}, ..., \mathbf{X}_N^{*(b)} \overset{\text{iid}}{\sim} \hat{P} \quad \forall b \in \{1, 2, ..., B\}.$$

By evaluating the bootstrap replications we obtain

$$\hat{\theta}^{*(b)} = s(\mathbf{X}^{*(b)}) \quad \forall b \in \{1, 2, ..., B\}.$$

The estimate of the standard error is

$$\widehat{\text{se}}_{\text{boot}}(\hat{\theta}) = \left[\frac{1}{B-1} \sum_{b=1}^{B} \left(\hat{\theta}^{*(b)} - \hat{\theta}^{*(\cdot)} \right)^2 \right]^{\frac{1}{2}} \tag{5.7.1}$$

where $\hat{\theta}^{*(\cdot)} = \dfrac{1}{B} \sum_{b=1}^{B} \hat{\theta}^{*(b)}$ and $\hat{\theta}^{*(b)}$ is the b^{th} bootstrap estimate. Eq. 5.7.1 estimates the standard deviation of $\hat{\theta}$. The accuracy of a point estimate is determined by the deviation of the expected value of an estimator from the true value, that is the bias. A bootstrap estimate

for bias is defined as

$$\widehat{\text{bias}}(\hat{\theta}) = \hat{\theta}^{*(\cdot)} - \hat{\theta}.$$

In the bootstrap estimate of bias, the unknown population parameter θ is replaced by the estimate $\hat{\theta}$ from the sample. If the bias is too large to be ignored, the corrected bias is calculated as (Ruppert & Matteson, 2011):

$$\hat{\theta}_{\text{bc}} = \hat{\theta} - \widehat{\text{bias}}(\hat{\theta}) = 2\hat{\theta} - \hat{\theta}^{*(\cdot)}.$$

Bias correction is not necessarily a good thing since the variability in a bias-corrected estimate may be high (Davison & Hinkley, 1997).

5.7.3.2 Bootstrap Estimates of Prediction Error

To estimate the prediction error using bootstrap the model is fit on each set of bootstrap samples and a record of how well it predicts the original training set is kept. Thus, the estimate prediction error from bootstrap samples is

$$\begin{aligned}\widehat{Err}_{boot} &= \frac{1}{B}\sum_{b=1}^{B}\frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(\mathbf{x}_i, y_i, \hat{f}_b(\mathbf{x}_i)) \\ &= \frac{1}{B}\frac{1}{N}\sum_{b=1}^{B}\sum_{i=1}^{N}\mathcal{L}(\mathbf{x}_i, y_i, \hat{f}_b(\mathbf{x}_i)) \end{aligned} \tag{5.7.2}$$

where $\hat{f}_b(\mathbf{x}_i)$ is the predicted value at $\mathbf{x}_i$ from the model fit to the b^{th} bootstrap dataset. Unfortunately, this is generally not a good estimator since the bootstrap samples used to produce $\hat{f}_b(\mathbf{x}_i)$ may contain $\mathbf{x}_i$, that is the bootstrap samples may overlap (Friedman et al., 2001). The leave-one-out bootstrap estimator provides an improvement by imitating cross-validation and is defined as

$$\widehat{Err}_{boot(1)} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{|C^{-i}|}\sum_{b\in C^{-i}}\mathcal{L}(\mathbf{x}_i, y_i, \hat{f}_b(\mathbf{x}_i)) \tag{5.7.3}$$

where C^{-i} denotes the set of indices for bootstrap samples that do not contain observation i and $|C^{-i}|$ denotes the number of such samples. Eq. 5.7.3 solves the overfitting suffered by Eq. 5.7.2, but this is still biased. The bias is due to non-distinct observations that result from sampling with replacement. Efron & Gong (1983) proposed the .632 estimator which is defined as

$$\widehat{Err}_{.632} = 0.368\overline{err} + 0.632\widehat{Err}_{boot(1)}$$

where $\overline{err} = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(\mathbf{x}_i, y_i, f(\mathbf{x}_i))$ is the naive estimate of prediction error, that is the training error. The .632 estimator is designed to correct the upward bias in $\widehat{Err}_{boot(1)}$ by averaging it with the downwardly biased estimate $\widehat{Err}_{.632}$. Unfortunately, if the model is highly overfitting, that is training error is zero, the .632 estimator will be downward biased. Efron & Tibshirani (1997) proposed the .632+ estimator designed to be the less biased between $\overline{err}$ and $\widehat{Err}_{boot(1)}$. The .632+ rule puts greater weight on $\widehat{Err}_{boot(1)}$ when the amount of overfitting, measured by $\widehat{Err}_{boot(1)} - \overline{err}$, is large. The .632+ estimator is defined as

$$\widehat{Err}_{.632+} = (1-\hat{w})\overline{err} + \hat{w}\widehat{Err}_{boot(1)}$$

where $\hat{w} = \frac{0.632}{1-0.368\hat{R}}$ and $\hat{R} = \frac{Err_{boot(1)} - \overline{err}}{\hat{\gamma}}$. γ denotes a no-information error rate, $\hat{\gamma}$ is an estimate of γ obtained by evaluating the prediction model on all possible combinations of target variables, y_i, and predictors, $\mathbf{x}_j$, which is defined as

$$\hat{\gamma} = \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}\mathcal{L}(\mathbf{x}_i, y_i, \hat{f}(\mathbf{x}_j)).$$

The weight $\hat{w}$ ranges from .632 if $\hat{R} = 0$ to 1 if $\hat{R} = 1$ and thus, $\widehat{Err}_{.632+} \in \left[\widehat{Err}_{.632},\ \widehat{Err}_{boot(1)}\right]$.

5.7.3.3 Bootstrap-t Confidence Interval

Suppose that $\hat{\theta}$ is approximately normally distributed with θ and variance $\text{se}(\hat{\theta})^2$. Let $\widehat{\text{se}}_{\mathbf{X}}(\hat{\theta})$ be an estimator of $\text{se}(\hat{\theta})$ based on the sample $\mathbf{X}$. The bootstrap-t confidence interval is constructed by generating the bootstrap replications $\hat{\theta}^{*(1)}, \hat{\theta}^{*(2)}, ..., \hat{\theta}^{*(B)}$ which provide an estimate of the sampling distribution of $\hat{\theta}$. For the bootstrap sample $\mathbf{X}^{*(b)}$, we can compute

$$T^{*(b)} = \frac{\hat{\theta}^{*(b)} - \hat{\theta}}{\widehat{\text{se}}_{\mathbf{X}^*}(\hat{\theta})}.$$

Using the values of $T^{*(b)}$, critical values $t_{1-\frac{\alpha}{2}}$ and $t_{\frac{\alpha}{2}}$ can be estimated by $\hat{t}_{1-\frac{\alpha}{2}}$ and $\hat{t}_{\frac{\alpha}{2}}$ respectively, such that (Efron, 1979)

$$\frac{1}{B}\sum_{b=1}^{B}\mathbb{I}\left\{T^{*(b)} \leq \hat{t}_{1-\frac{\alpha}{2}}\right\} \approx \frac{\alpha}{2}$$

and

$$\frac{1}{B}\sum_{b=1}^{B}\mathbb{I}\left\{T^{*(b)} \leq \hat{t}_{\frac{\alpha}{2}}\right\} \approx \frac{\alpha}{2}.$$

Then a 100(1-α)% confidence interval for θ can be derived as follows

$$\begin{aligned}
\hat{P}^*\Big(\hat{t}_{\frac{\alpha}{2}} \leq T^{*(b)} \leq \hat{t}_{1-\frac{\alpha}{2}}\Big) &\approx 1-\alpha \\
\hat{P}^*\Big(\hat{t}_{\frac{\alpha}{2}} \leq \frac{\hat{\theta}^{*(b)} - \hat{\theta}}{\hat{\text{se}}_{\mathbf{z}^*}(\hat{\theta})} \leq \hat{t}_{1-\frac{\alpha}{2}}\Big) &\approx 1-\alpha \\
\hat{P}^*\Big(\hat{t}_{\frac{\alpha}{2}} \leq \frac{\hat{\theta} - \theta}{\text{se}(\hat{\theta})} \leq \hat{t}_{1-\frac{\alpha}{2}}\Big) &\approx 1-\alpha \\
\hat{P}^*\Big(\hat{t}_{\frac{\alpha}{2}}\text{se}(\hat{\theta}) \leq \hat{\theta} - \theta \leq \hat{t}_{1-\frac{\alpha}{2}}\text{se}(\hat{\theta})\Big) &= 1-\alpha \\
\hat{P}^*\Big(\hat{t}_{1-\frac{\alpha}{2}}\text{se}(\hat{\theta}) - \hat{\theta} \leq \theta \leq \hat{t}_{\frac{\alpha}{2}}\text{se}(\hat{\theta}) - \hat{\theta}\Big) &= 1-\alpha \\
\therefore \hat{P}^*\Big(\hat{\theta} - \hat{t}_{1-\frac{\alpha}{2}}\text{se}(\hat{\theta}) \leq \theta \leq \hat{\theta} - \hat{t}_{\frac{\alpha}{2}}\text{se}(\hat{\theta})\Big) &= 1-\alpha \\
\hat{P}^*(\hat{\theta}_L \leq \theta \leq \hat{\theta}_U) &= 1-\alpha
\end{aligned}$$

where $\hat{\theta}_L = \hat{\theta} - \hat{t}_{1-\frac{\alpha}{2}}\text{se}(\hat{\theta})$ and $\hat{\theta}_U = \hat{\theta} - \hat{t}_{\frac{\alpha}{2}}\text{se}(\hat{\theta})$. The 100(1-$\alpha$)% confidence interval for θ, namely

$$\Big[\hat{\theta}_L, \hat{\theta}_U\Big] = \Big[\hat{\theta} - \hat{t}_{1-\frac{\alpha}{2}}\text{se}(\hat{\theta}), \hat{\theta} - \hat{t}_{\frac{\alpha}{2}}\text{se}(\hat{\theta})\Big] \tag{5.7.4}$$

is given by the empirical quantile of the bootstrap replications, that is

$$\hat{P}^*(\hat{\theta}^* \leq \hat{\theta}_L) = \frac{1}{B}\sum_{b=1}^{B} \mathbb{I}\Big\{\hat{\theta}^{*(b)} \leq \hat{\theta}_L\Big\} \approx \frac{\alpha}{2}$$

and

$$\hat{P}^*(\hat{\theta}^* \geq \hat{\theta}_U) = \frac{1}{B}\sum_{b=1}^{B} \mathbb{I}\Big\{\hat{\theta}^{*(b)} \geq \hat{\theta}_U\Big\} \approx \frac{\alpha}{2}$$

The confidence interval in Eq. 5.7.4 is similar to the common student-t interval (refer to Rice (2006)) except that the t-value is replaced by the bootstrap estimates $\hat{t}_{1-\frac{\alpha}{2}}$ and $\hat{t}_{\frac{\alpha}{2}}$ (Johnson, 2001).

5.8 Chapter 5 Summary

For highly imbalanced datasets, the accuracy measure of any classifier will always be high; thus, accuracy is not a reliable performance metric. This chapter discussed several binary classifier performances, namely AUC, G-mean, precision and recall curve, F-measure and MCC. Section 5.7 discussed two types of model validation, namely CV and bootstrap.

Chapter 6

Improving Model Performance

This chapter introduces different ways of improving classifier performance. The classification performance of any classifier built on imbalanced dataset can be improved at data-level, that is manipulating the given dataset or at algorithmic level, that is manipulating classifier parameters. Sections 6.1, 6.2 and 6.3 discuss the techniques used to deal with imbalanced dataset at data-level while sections 6.4 focuses on algorithmic level.

6.1 Sampling Techniques

6.1.1 Random Undersampling and Oversampling

The random oversampling (RO) technique increases the number of minority class members in the training set. The advantage of oversampling is that no information from the original training set is lost since we keep all minority and majority classes members. However, the disadvantage is that it significantly increases the size of the training set. This increases training time and the amount of memory required to store the training set. When dealing with very high dimensional datasets, care should be taken on how one might ensure that time complexity and memory complexity are kept under reasonable constraint (Liu, 2004). RO tends to overfit since it duplicates minority class observations (Liu et al., 2008). Chawla et al. (2002) proposed SMOTE as a solution to this problem, see section 6.1.2.

The random undersampling (RU) technique uses a random subset of the majority class to train the classifier. It ignores many majority class observations to ensure that the classes are more balanced, and the training process becomes faster. The main disadvantage of RU is that potentially useful information can be contained in the ignored cases (Ganganwar, 2012).

To improve classifier performance under random sampling, Kubat et al. (1997) proposed the one-sided sampling method that tries to find a consistent subset, D', of the original data D_N in the sense that the 1-NN rule learned from D' can correctly classify all examples in D_N.

Initially, D' contains all the minority class examples and one randomly selected majority class example. A 1-NN, that is $k = 1$, classifier is constructed on D' to classify the examples in D. All misclassified majority examples are added into D'. The Tomek link as proposed by Tomek (1976), is then used to remove the borderline or noisy cases in the majority class in D'. Suppose $C(\cdot)$ is the class label. Let $d(\mathbf{x}_i, \mathbf{x}_j)$ denote the Euclidean distance between $\mathbf{x}_i$ and $\mathbf{x}_j$. The pair $(\mathbf{x}_i, \mathbf{x}_j)$ is a Tomek link if $C(\mathbf{x}_i) \neq C(\mathbf{x_j}) \quad \forall i, j$ and $\forall k \in \mathbb{R}^+, \nexists \mathbf{x}_k$ such that $d(\mathbf{x}_i, \mathbf{x}_k) < d(\mathbf{x}_i, \mathbf{x}_j)$ or $d(\mathbf{x}_j, \mathbf{x}_k) < d(\mathbf{x}_j, \mathbf{x}_i)$. Figs. 6.1(a) and (b) illustrate how random sampling and random oversampling can both be applied to balance the credit card dataset. Consider any of the two circled points in Fig. 6.1(c). The blue star point has the orange triangle as its nearest neighbour and vice versa. These two circled points have different class labels, for example the blue star point belongs to a non-fraudulent class and the orange point belongs to a fraudulent class. Since these points have two different class labels and are each other's nearest neighbours, they are Tomek links.

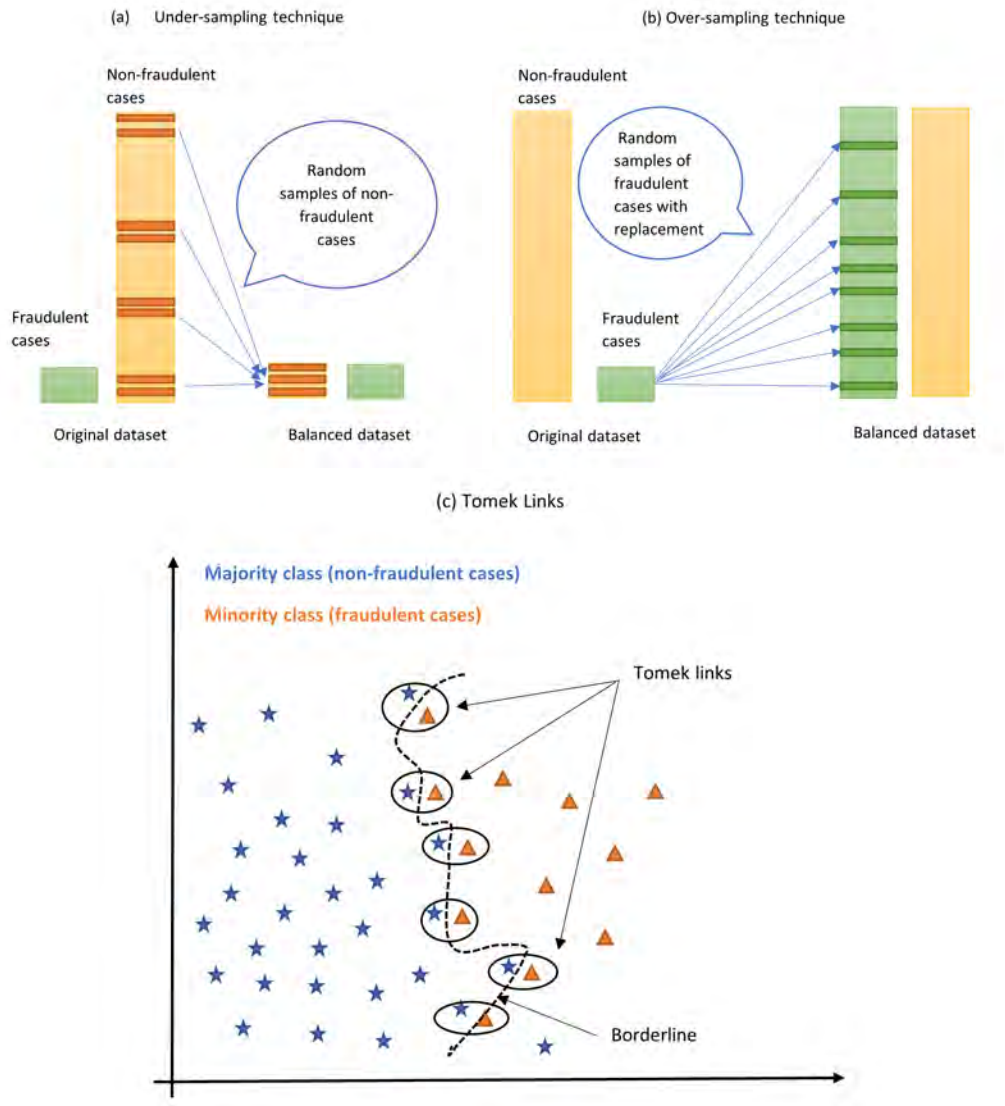


Figure 6.1: Demonstration of random undersampling, oversampling and the Tomek link (adapted from Agarwal (2018)).

6.1.2 The Synthetic Minority Oversampling Technique

The synthetic minority oversampling technique (SMOTE) is an oversampling technique that forms new minority class observations by interpolating several minority class examples that lie close together. This process potentially leads to overfitting on the multiple copies of minority class observations. The minority class is over-sampled by taking each minority class sample and introducing synthetic instances along the line segments joining all of the k minority class nearest neighbours, as shown in Fig. 6.2. Depending upon the amount of over-sampling required, neighbours from the k nearest neighbours are randomly chosen (Chawala, 2009). To generate synthetic instances, let M denote the sampling rate set according to the imbalanced proportion and s denote a random integer such that $s \in (0, 1)$, $\forall s \in \mathbb{Z}$. For each observation $\mathbf{x}$ in N^+, the nearest neighbour of $\mathbf{x}$ is obtained by calculating the Euclidean distance between $\mathbf{x}$ and every other sample in the set N^+. For each observation in N^+, $\mathbf{x}_i$ where $i \in (1, 2, ..., N^+)$ are randomly selected from its $k-$nearest neighbours and used to construct a new set N_1^+. Now, for each $\mathbf{x}_j \in N^+$ where $j \in (1, 2, ..., M)$ a synthetic sample, $\mathbf{x}_{synthetic}$, is generated by

$$\mathbf{x}_{synthetic} = \mathbf{x} + s \times d(\mathbf{x}, \mathbf{x}_j)$$

where $d(\mathbf{x}, \mathbf{x}_j)$ denotes the Euclidean distance.

Fig 6.2 depicts how SMOTE creates synthetic instances. In this example, the star symbols denote non-fraudulent cases (the majority class) and the triangle symbols denote fraudulent cases (the minority class). In this example, the value of k is chosen to be seven, that is we calculate the Euclidean distance from the selected minority sample to seven nearest neighbours. Each calculated Euclidean distance is multiplied by a random number, s that is between 0 and 1 and the selected minority sample is added to each multiplied Euclidean distance produce seven synthesized samples.

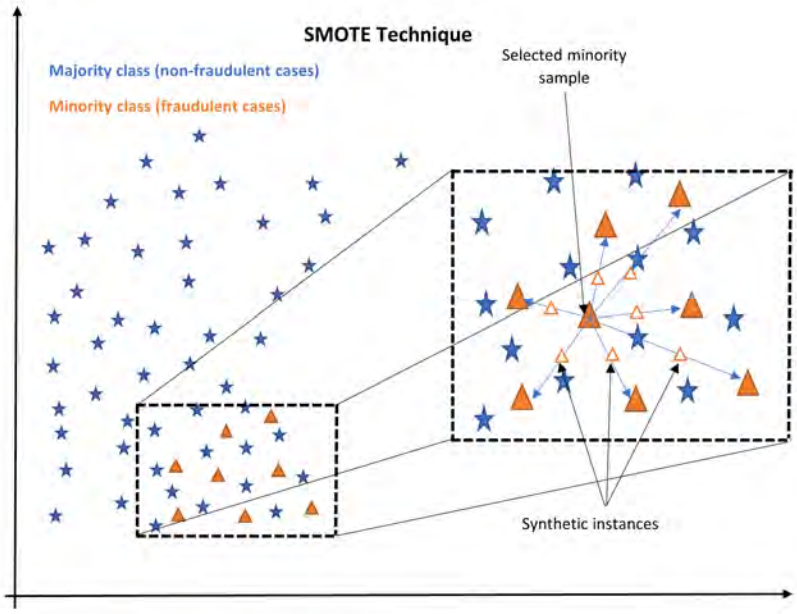


Figure 6.2: Example of the SMOTE algorithm using $k = 7$ neighbours (adapted from Walimbe (2017)).

6.2 Dimensionality Reduction

6.2.1 Principal Component Analysis

Principal component analysis (PCA) is a dimension reduction technique often used to reduce the size of large datasets by transforming a large set of feature variables into a smaller set of variables that still contain important information from the larger dataset. Reducing the number of feature variables in a dataset sacrifices accuracy, that is, dimensionality reduction trades accuracy for simplicity. Small datasets are easier to explore, visualize and analyze with machine learning algorithms without dealing with irrelevant variables (Wold et al., 1987). Suppose $\mathbf{X}$ is a data matrix with N rows and m columns. Let $\mathbf{x}_i$ denote a column vector for each $i \in \{1, 2, ..., m\}$, thus $\mathbf{x}_i$ denotes the m observations of feature variable i. A linear combination of those $\mathbf{x}$ variables can be expressed as $\mathbf{t} = w_1\mathbf{x}_1 + w_2\mathbf{x}_2 + ... + w_m\mathbf{x}_m$ where $\mathbf{t}$ is the new vector in the same space as the $\mathbf{x}$ variables. The matrix $\mathbf{X}$ contains variation that is relevant to the problem, namely the classification of fraudulent and non-fraudulent transactions. Therefore, $\mathbf{t} = \mathbf{Xw}$ must have as much variation as possible. If this amount of variation in $\mathbf{t}$ is significant, then it may be a good summary of the $\mathbf{X}$ variables. For example, thirty variables of $\mathbf{X}$ could be replaced by two $\mathbf{t}$ variables which retain most of the relevant information. The objective of principal component is to find a linear combination of feature

variables that maximizes the variance, that is

$$\max \ \frac{\text{var}(\mathbf{t})}{\mathbf{w}^T\mathbf{w}}. \tag{6.2.1}$$

Since $\mathbf{w}^T\mathbf{w} = ||\mathbf{w}||^2 = ||\mathbf{w}||||\mathbf{w}||$, Eq. 6.2.1 is equivalent to the constraint problem

$$\underset{||\mathbf{w}||=1}{\text{argmax}} \ \text{var}(\mathbf{t}) \tag{6.2.2}$$

where $||\cdot||^2$ denotes the squared norm (Bro & Smilde, 2014). Let

$$\mathbf{S} = \frac{1}{N-1}\sum_{i=1}^{N}\mathbf{x}_i\mathbf{x}_i^{\mathrm{T}} = \frac{\mathbf{X}^{\mathrm{T}}\mathbf{X}}{N-1}$$

denote the covariance matrix of the centered variables, i.e. after substrating the mean. Eq. 6.2.2 can be re-expressed as

$$\underset{||\mathbf{w}||=1}{\text{argmax}} \ \mathbf{w}^{\mathrm{T}}\mathbf{S}\mathbf{w}.$$

$\mathbf{S}$ is a symmetric, positive semidefinite matrix by construction and has eigenvalue decomposition of the form $\mathbf{S} = \mathbf{Q}\Lambda\mathbf{Q}^{\mathrm{T}}$, where $\mathbf{Q}$ is an orthogonal matrix, $\mathbf{Q}\mathbf{Q}^{\mathrm{T}} = \mathbf{I}$ where the columns of $\mathbf{Q}$ are the orthonormal eigenvalues of $\mathbf{S}$, and Λ is a diagonal matrix of the corresponding non-negative eigenvalues of $\mathbf{S}$, λ_i such that $\lambda_1 \geq \lambda_2, ... \geq \lambda_m \geq 0$ (Abdi & Williams, 2010). Hence,

$$\begin{aligned}
\mathbf{w}^{\mathrm{T}}\mathbf{S}\mathbf{w} &= \mathbf{w}^{\mathrm{T}}\mathbf{Q}\Lambda\mathbf{Q}^{\mathrm{T}}\mathbf{w} = \mathbf{p}^{\mathrm{T}}\Lambda\mathbf{p} \\
&= \begin{bmatrix} p_1 & p_2 & \cdots & p_m \end{bmatrix} \begin{bmatrix} \lambda_1 & & & \\ & \ddots & & \\ & & & \lambda_m \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_m \end{bmatrix} \\
&= \lambda_1 p_1^2 + \lambda_2 p_2^2 + \cdots + \lambda_m p_m^2 = \sum_{i=1}^{m} \lambda_i p_i^2
\end{aligned}$$

where $\mathbf{p} = \mathbf{Q}^{\mathrm{T}}\mathbf{w}$. The norm of $\mathbf{p}$ is

$$\begin{aligned}
||\mathbf{p}||^2 = ||\mathbf{Q}^{\mathrm{T}}\mathbf{w}||^2 &= (\mathbf{Q}^{\mathrm{T}}\mathbf{w})^{\mathrm{T}}(\mathbf{Q}^{\mathrm{T}}\mathbf{w}) \\
&= \mathbf{w}^{\mathrm{T}}\underbrace{\mathbf{Q}\mathbf{Q}^{\mathrm{T}}}_{\mathbf{I}}\mathbf{w} = \mathbf{w}^{\mathrm{T}}\mathbf{w} = ||\mathbf{w}||^2 \\
&= 1 \quad (\text{since } \ ||\mathbf{w}|| = ||\mathbf{w}||^2 = 1).
\end{aligned}$$

Since $\mathbf{p}$ is a unit vector, $\sum_{i}^{m} p_i^2 = 1$. Thus, $\mathbf{p}^{\mathrm{T}}\Lambda\mathbf{p}$ is always an average of the $\lambda_i' s$ with averaging weights given by p_i^2. To make this average as big as possible, set $p_1 = 1$ and $p_i = 0$ for $i > 1$ since λ_1 is the biggest eigenvalue. That is $\mathbf{p} = \mathbf{e}_1$ maximizes $\mathbf{p}^{\mathrm{T}}\Lambda\mathbf{p}$ where

$$\mathbf{e}_1 = \begin{bmatrix} 1 & 0 & \cdots & 0 \end{bmatrix}$$

denotes the first standard basis vector. It then follows that $\mathbf{p}^{\mathrm{T}}\Lambda\mathbf{p} \leq \lambda_1$ for every unit vector $\mathbf{p}$ which implies that $\mathbf{q}_1 = \mathbf{Q}\mathbf{e}_1$ maximizes $\mathbf{p}^{\mathrm{T}}\Lambda\mathbf{p}$ where $\mathbf{q}_1$ denotes the first column of $\mathbf{Q}$ (Roughgarden & Valiant, 2021). The second principal component is computed the same way with the condition that it has the second highest variance and is uncorrelated to the first component. This continues until all principal components have been computed, that is, the remaining principal components are computed by solving the optimization problem $\max \mathbf{w}_i\mathbf{S}\mathbf{w}_i$ subject to $\mathbf{w}_i^{\mathrm{T}}\mathbf{w}_i = 1$ and $\mathbf{w}_i^{\mathrm{T}}\mathbf{w}_j = 0 \ \forall 1 < j < i$ for the i^{th} principal component.

6.3 Feature Selection

Guyon et al. (2002) proposed the recursive feature elimination (RFE) technique to reduce dimensionality of the feature space. Suppose we have a prediction model that assigns weight w_i to feature x_i according to their importance. The feature with the smallest value of w_i is considered to be the least important feature. The objective of RFE is to recursively remove the least important features until a specified number of features is reached. At first, the model is trained on D_{train}. In the credit card dataset, D_{train} contains 30 features. The importance of each feature is obtained by assigning a weight to each feature. RFE recursively removes the least important features at each step and re-ranks the remaining features by retaining the model based on the remaining features. However, the least important feature may still be an important feature when combined with other features (Guyon & Elisseeff, 2003). Hence, simply removing a redundant or least important feature may degrade model performance. Chen & Jeong (2007) proposed enhanced recursive feature elimination (EnRFE) to overcome this issue. The EnRFE algorithm assesses the importance of the potentially least important feature and evaluates the model's performance after this feature is removed, based on the corresponding value of w_i. If the model's performance degrades after this feature is removed, the feature will not be removed even though it has the smallest value of w_i. The feature with the second smallest value of w_i is then considered. The process is repeated until a feature that does not degrade the model's performance when it is removed is found.

6.4 Parameter Tuning

Hyper-parameters are parameters whose values are used to control the learning process (Wistuba et al., 2015). Effectively searching the hyper-parameters' space using optimazation techniques can help identify the optimal parameters for models. The parameter tuning optimization process consist of four components: an estimator with its objective function, a search space, the optimization method used to find the hyper-parameters and an evaluation function to compare the performance of the different hyper-parameters. Sometimes hyper-parameters cannot be learned from the training data as they increase model complexity and cause the model to overfit (Claesen & De Moor, 2015). In general the hyper-parameter optimization problem can be setup up as follows (Wistuba et al., 2015): let $\mathcal{A}_{\boldsymbol{\lambda}} : D_N \rightarrow \mathcal{M}$ be a learning algorithm where $\mathcal{M}$ denotes the space of all models and $\boldsymbol{\lambda} \in \boldsymbol{\Theta}$ is a chosen search space where $\boldsymbol{\Theta} = \boldsymbol{\Theta}_1 \times \boldsymbol{\Theta}_2 \times ... \times \boldsymbol{\Theta}_d$ is a d-dimensional hyper-parameter space. The objective of hyperparameter optimization is to find the hyper-parameter configuration $\boldsymbol{\lambda}^*$. That is we seek

$$\begin{aligned} \boldsymbol{\lambda}^* &= \underset{\boldsymbol{\lambda} \in \Lambda}{\operatorname{argmin}} \; L\Big(\mathcal{A}_{\boldsymbol{\lambda}}\Big(D_{train}\Big), D_{val}\Big) \\ &\approx \underset{\lambda \in \{\lambda_1, \lambda_2, ..., \lambda_S\}}{\operatorname{argmin}} \; \Psi(\lambda) \end{aligned}$$

where Ψ denotes a hyper-parameter response function. In general, very little information is known about the response function Ψ or the search space $\boldsymbol{\Theta}$. The critical step in hyper-parameter optimization is choosing the set of trials $\{\lambda_1, \lambda_2, ..., \lambda_S\}$ (Bergstra & Bengio, 2012).

6.4.1 Grid Search

The Grid search (GS) method is based on defining a hyper-parameter search space and detecting the optimal hyper-parameter combination in the search space (Bergstra et al., 2011). Let Λ be indexed by k configuration variables, for example in a SVM k would be the regularization parameter, kernel, gamma and so on. GS requires that we choose a set of values for each variable, L_i where $i \in \{1, 2, ..., k\}$. The set of trials is formed by combining every possible combination of values; the number of trials is $\prod_{i=1}^{k} |L_i|$ elements. This product over k makes GS inefficient for high dimensionality hyper-parameter configuration space since the number of joint values increases exponentially as the number of hyper-parameters grows (Bergstra & Bengio, 2012). GS cannot exploit the well-performing regions further by itself, and therefore combining it with manual search is a good strategy in finding hyper-parameters (Yang & Shami, 2020). GS is computationally expensive; it is only an effective hyper-parameter optimization method when the hyper-parameter configuration space is small (Yang & Shami, 2020).

6.4.2 Random Search

Bergstra & Bengio (2012) proposed random search (RS) as a solution to certain GS limitations. RS is a technique where random combinations of hyper-parameters are used to find the best solution for the model. Unlike GS, which test every possible combination of values in the search space, RS tries a random combination of values in the search space. GS is computationally more expensive than RS. If the search space is large enough, then the global optimum or at least an approximation can be found (Yang & Shami, 2020). RS is more practical than GS since it can be applied even when using a cluster of computers that can fail. Adding new trials to the set or ignoring failed trials are feasible since the trials are iid, which is not the case for GS (Bergstra & Bengio, 2012). RS samples a fixed number of parameter combinations from a specified distribution, which improves system efficiency by reducing the probability of wasting time on a poor performing region. Bergstra & Bengio (2012) showed that RS is more efficient than GS in high dimensional spaces since functions Ψ of interest have low effective dimensionality.

6.5 Chapter 6 Summary

To address the imbalanced dataset and improve the classification performance of all the classifiers at data-level, this chapter discussed several sampling techniques, namely RO, RU, one-sided sampling and SMOTE. To reduce overfitting when sampling techniques are applied to the imbalanced dataset, Tomek links are introduced. Furthermore, different parameter tuning methods to improve classification performance at the algorithmic level, namely GS and RS are discussed.

Chapter 7

Results and Discussion

In the chapter we present the experimental results on the European credit card dataset. Section 7.1 gives a brief background of the European credit card dataset. Section 7.2 discusses the analysis of the results obtained from each classifier and finally, section 7.3 compare these results to similar studies.

7.1 The European Credit Card Dataset

This dataset contains information about fraudulent and non-fraudulent transactions in the credit card transaction domain. It has records of European credit cardholders who made transactions over two days using their credit cards in September 2013. The dataset[3] is in CSV format. Background information about the feature variables is not provided due to confidentiality issues (Dal Pozzolo et al., 2015). Features labelled $V1$ to $V28$, are the principal components obtained from a PCA of the original data by Dal Pozzolo et al. (2015), 'Time' and 'Amount' are the only features that have not been transformed. The time feature denotes the number of seconds elapsed between the transaction and the first transaction in the dataset. The amount feature denotes the transaction amount. The response variable is defined as

$$y = \begin{cases} 1, & \text{if transaction is fraudulent, and} \\ 0, & \text{otherwise.} \end{cases}$$

This dataset contains 284 807 observations of the 30 feature variables. Of these only 492 observations were identified as being fraudulent. The dataset is highly imbalanced, with fraud cases accounting for 0.173% of all transactions (see Fig. 7.1). The dataset has no missing values. 1 825 transactions have an amount of €0.00 of which 27 are classified as fraudulent transactions and 1 798 are classified as non-fraudulent transactions. There are 1

[3] `https://query.data.world/s/3dwtejin6vc6r44dgd3vu66flypzon`

064 duplicated transactions, 17 are from the fraudulent class and 1 047 are from the non-fraudulent class.

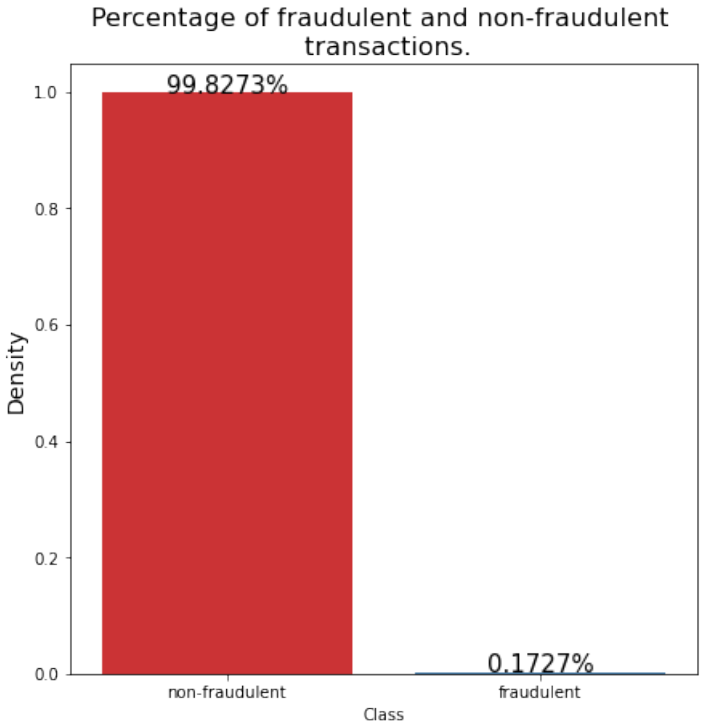


Figure 7.1: A percentage barplot of the fraudulent and non-fraudulent transactions for the European Credit Card Dataset.

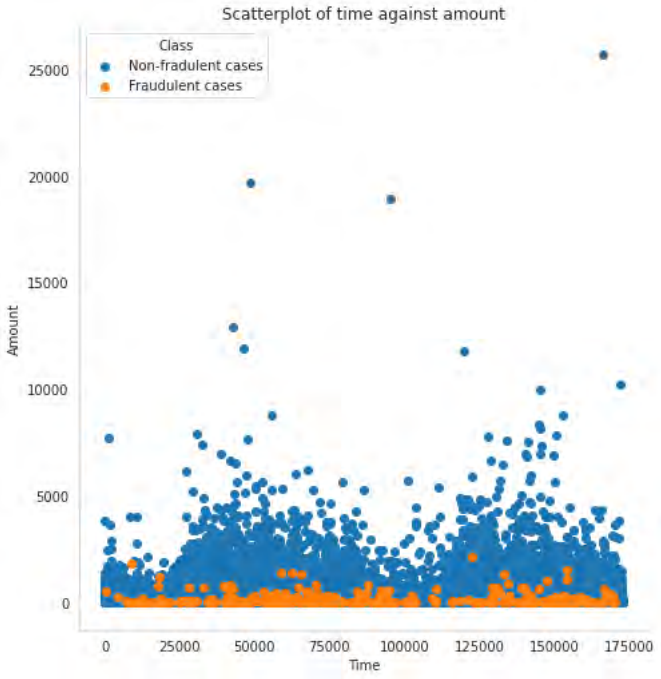


Figure 7.2: Scatterplot of the time and amount of fraudulent and non-fraudulent transactions.

Figs. 7.2 and 7.4 suggest that there are outliers with regards the amount feature, where most of the amount outliers are in the non-fraudulent class. There are no fraudulent transactions with an amount exceeding approximately €2 500. 281 469 (99.841%) of the transactions have a transaction amount of less than €2 500. Fig. 7.3 suggests that the fraudulent and non-fraudulent transaction are evenly distributed over time, there is no clear distinction between them. The amount distribution shows the €1.00 amount as the most frequently deducted amount for fraudulent and non-fraudulent transactions. The distribution is skewed to the right; the *mean* of the fraudulent and non-fraudulent transactions is to the right of the *median* transaction value.

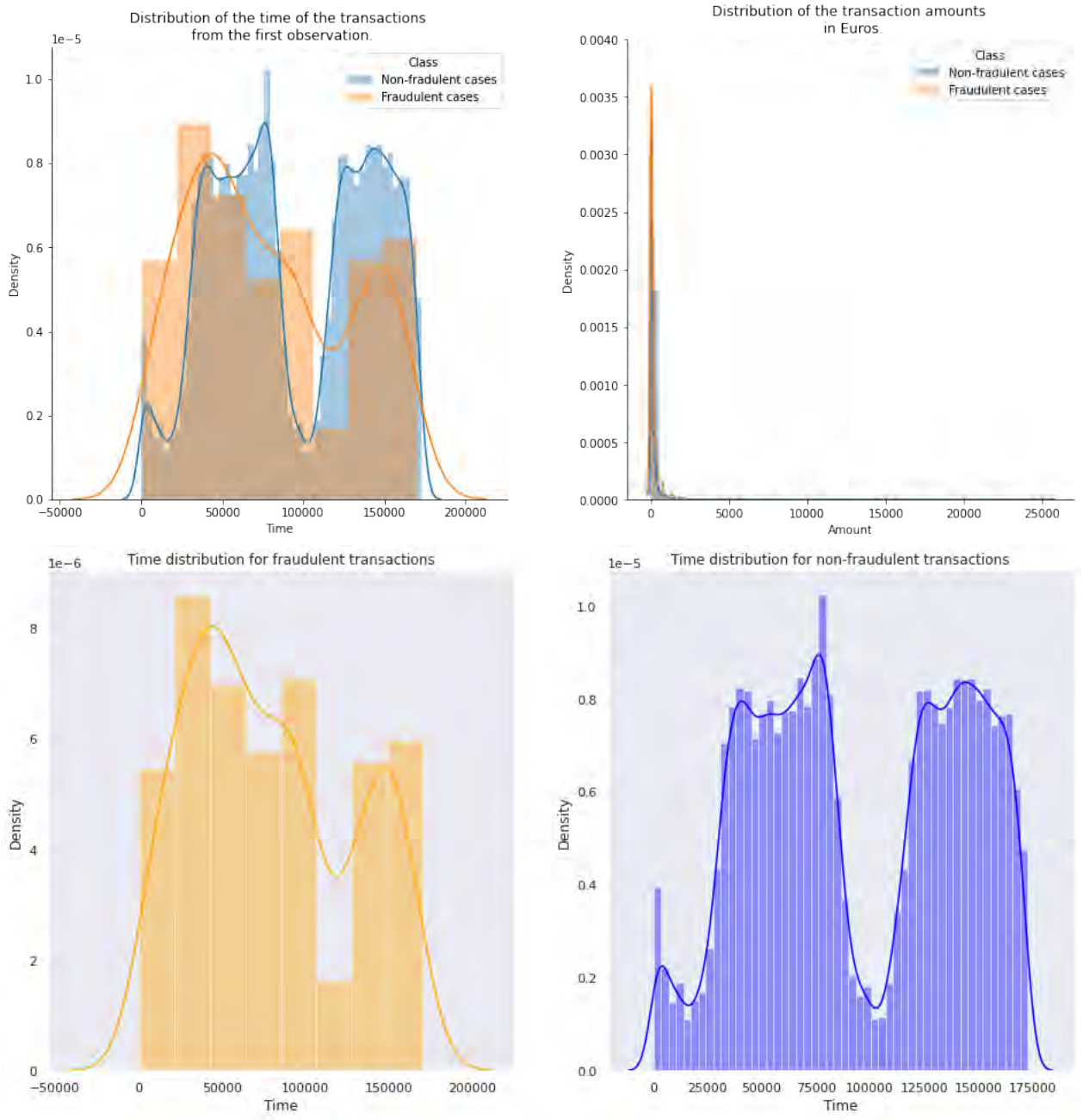


Figure 7.3: Distribution of both fraudulent and non-fraudulent transactions over time and amount.

The box plot of time does not show any obvious *mean* difference and the fraudulent and non-

fraudulent cases have similar variability. In the amount box-plot, there is too much variation in the amount column, and hence we have a condensed box-plot (Fig. 7.4).

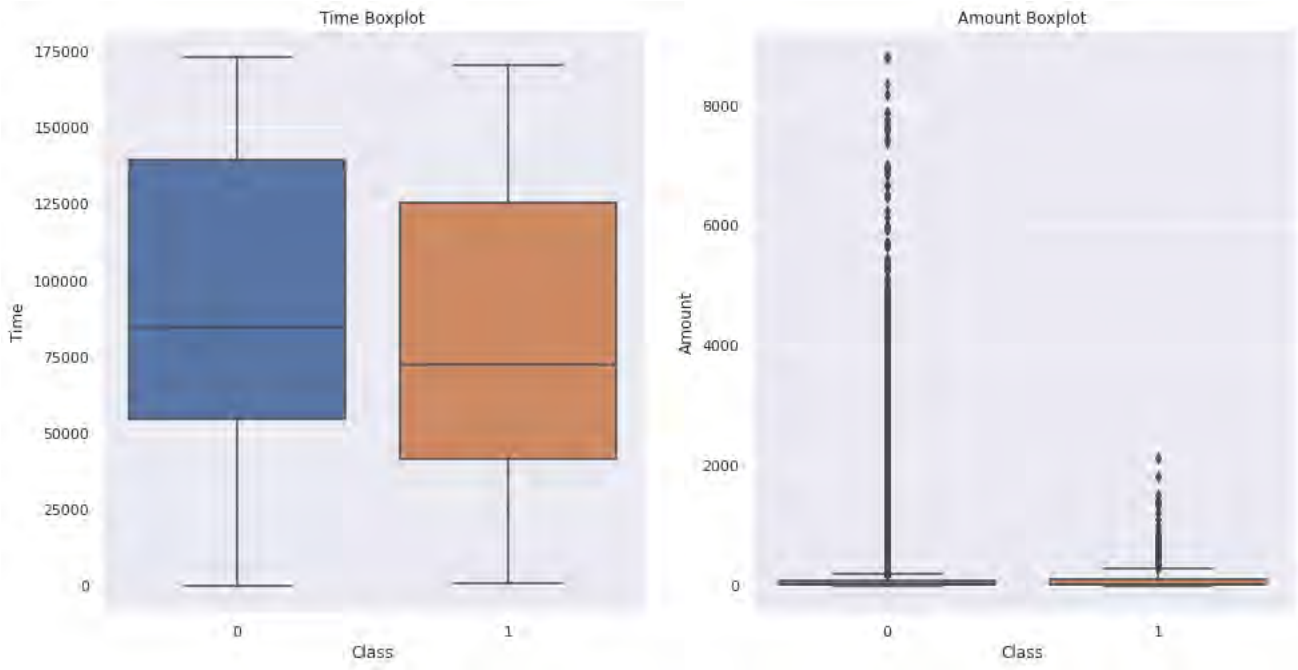


Figure 7.4: Boxplots of the time and amount feature variables

7.2 Analysis of the European Credit Card Dataset

This study has developed five classification models using the python programming language, namely, SVM, kNN, LR, DT and MLP. Several parameters for measuring models' performance have been computed: precision, recall, F_1 measure, G-mean, AUC and MCC. To evaluate these models, 70% of the data was used for training and 30 % for testing. In the data cleaning stage, 1 825 transactions with €0.00 amount and 1 064 duplicated transactions were removed from the dataset. This reduced the number of fraudulent transactions and non-fraudulent transactions. The dataset used to build the classification models has 281 981 observations, of which 448 (0.1589%) are fraudulent and 281 470 (99.8411%) are non-fraudulent transactions. The feature variables 'Time' and 'Amount' were normalized as these were the only feature variables not transformed by PCA and there was high variation. For example, the transaction with the minimum amount was €0.01 and the transaction with maximum amount was €25 691.16.

In the first phase of this study, classification models were built on the normalized data and performance measures were obtained. In the second phase, the one-sided sampling method was applied on the normalized data which removed 12 728 observations from the non-fraudulent transactions. In the third phase of this study PCA was performed on the normalized one-sided sampled dataset and 17 components were retained out of 30 features. Lastly, random undersampling was used to balance the classes. Due to limitations mentioned

in section 6.4.2, hyper-parameters for each classification model were obtained using RS with 5 fold cross validation.

Consider tables 7.1 and 7.2, the classifier performance on the test and training data for SVM, kNN, LR, DT and MLP. The training results show LR as the worst performing classifier amongst all these classifier. SVM, LR and DT are likely to be overfitting as for example in table 7.1 DT has no FPs, that is precision is equal to one and has no FNs, that is recall is equal to one. The overfitted DT classifier perfectly classifies fraudulent and non-fraudulent transactions on the train dataset. On the test data, DT has low precision which implies high a FPR and low recall which implies a high FNR. There is a large difference between training and test error. Table 7.3 and 7.4 shows the hyper-parametrized classification model results. The error difference between FNR on the train and test set decreased from 21% to 9% for SVM, 7% to 6% for kNN, 4% to 2% for LR, 26% to 10% for DT, 20% to 4% for MLP. This indicates a reduction in overfitting (Wong et al., 2016). The amount of improvement in performance varies across the classifiers. For example, LR classifier with default parameters has a FNR of 37% on the test set. The hyper-parameterized LR classifier on the test set has a FNR of 19%, a decrease of 18%. The decrease in FNR is at a cost of FPR, since a decrease in precision results to an increase in FPR.

Table 7.1: Training results on the normalized data with default model parameters.

Classification models	Precision	Recall	F_1
SVM	0.98	0.83	0.90
kNN	0.96	0.78	0.86
LR	0.90	0.67	0.77
DT	1.00	1.00	1.00
MLP	0.98	0.91	0.94

Table 7.2: Test results on the normalized data with default model parameters.

Classification models	Precision	Recall	F_1
SVM	0.91	0.62	0.74
kNN	0.93	0.71	0.81
LR	0.86	0.63	0.72
DT	0.77	0.74	0.76
MLP	0.98	0.71	0.80

Table 7.3: Training results on the normalized data with tuned model parameters.

Classification models	Precision	Recall	F_1
SVM	0.97	0.84	0.90
kNN	0.93	0.78	0.85
LR	0.80	0.79	0.79
DT	0.93	0.82	0.87
MLP	0.86	0.80	0.83

Table 7.4: Test results on the normalized data with tuned model parameters.

Classification models	Precision	Recall	F_1
SVM	0.91	0.75	0.82
kNN	0.93	0.72	0.81
LR	0.81	0.81	0.81
DT	0.84	0.72	0.78
MLP	0.89	0.76	0.82

The bootstrap sample mean is an estimate of the population mean. Since the mean is based on sample data and not the entire population, the sample mean is unlikely to equal the population mean. Confidence intervals are based on the sampling distribution of a statistic. If a statistic is an unbiased estimator of a population parameter, its sampling distribution is centred on the true value of the parameter. The bootstrap distribution approximates the sampling distribution of the statistic (Johnson, 2001). Therefore, an average of 95% of the values in the bootstrap distribution provides a 95% bootstrapped confidence interval (C.I) for the parameter. C.Is help to evaluate the actual significance of the estimate of the population parameter (James et al., 2013a). Tables 7.5 to 7.9 show the bootstrap point estimates, 95% C.I and the standard error for each classifier built on the normalized dataset. From these tables, it can be noted that LR outperforms all other classifiers with an AUC score of 0.9034 and G-mean of 0.8983.

Table 7.5: SVM bootstrap estimate on the normalized test set.

	Model estimate	Bootstrap point estimate	95% bootstrap C.I	Standard error
Precision	0.9091	0.9310	[0.8749, 1.0000]	0.0408
Recall	0.7463	0.7552	[0.6893, 0.8090]	0.0368
F_1	0.8197	0.8314	[0.8031, 0.8585]	0.0193
G-mean	0.8638	0.8676	[0.8275, 0.8992]	0.0221
AUC	0.8731	0.8776	[0.8447, 0.9044]	0.0184
MCC	0.8234	0.8370	[0.8139, 0.8620]	0.0178

Table 7.6: kNN bootstrap estimate on the normalized test set.

	Model estimate	Bootstrap point estimate	95% bootstrap C.I	Standard error
Precision	0.9320	0.9060	[0.8487, 0.9754]	0.0450
Recall	0.7164	0.7404	[0.6890, 0.8046]	0.0344
F_1	0.8101	0.8129	[0.7784, 0.8602]	0.0258
G-mean	0.8592	0.8717	[0.8268, 0.8968]	0.0207
AUC	0.8582	0.8701	[0.8445, 0.9023]	0.0172
MCC	0.8169	0.8178	[0.7824, 0.8634]	0.0259

Table 7.7: LR bootstrap estimate on the normalized test set.

	Model estimate	Bootstrap point estimate	95% bootstrap C.I	Standard error
Precision	0.8129	0.8200	[0.7600, 0.8600]	0.0300
Recall	0.8071	0.8123	[0.7620, 0.8503]	0.0290
F_1	0.8100	0.8148	[0.7664, 0.8313]	0.0213
G-mean	0.8983	0.9009	[0.8727, 0.9219]	0.0162
AUC	0.9034	0.9060	[0.8809, 0.9250]	0.0145
MCC	0.8097	0.8158	[0.7670, 0.8314]	0.0214

Table 7.8: DT bootstrap estimate on the normalized test set.

	Model estimate	Bootstrap point estimate	95% bootstrap C.I	Standard error
Precision	0.8435	0.8647	[0.7665, 0.9527]	0.0607
Recall	0.7239	0.7808	[0.6822, 0.8562]	0.0556
F_1	0.7791	0.8168	[0.7497, 0.8649]	0.0418
G-mean	0.8507	0.8853	[0.8322, 0.9261]	0.0335
AUC	0.8618	0.8937	[0.8410, 0.9294]	0.0293
MCC	0.7811	0.8207	[0.7590, 0.8681]	0.0381

Table 7.9: MLP bootstrap estimate on the normalized test set.

	Model estimate	Bootstrap point estimate	95% bootstrap C.I	Standard error
Precision	0.8947	0.8598	[0.7931, 0.9005]	0.0349
Recall	0.7612	0.8011	[0.7322, 0.8402]	0.0345
F_1	0.8226	0.8249	[0.7972, 0.8539]	0.0220
G-mean	0.8724	0.8952	[0.8605, 0.9255]	0.0184
AUC	0.8805	0.8922	[0.8736, 0.9159]	0.0163
MCC	0.8250	0.8385	[0.7978, 0.8651]	0.0215

Consider tables 7.10 and 7.11 the classifier performance on the test and training data for SVM, kNN, LR, DT and MLP. SVM, MLP and DT are likely to be overfitting since there is a large difference between the FNR in the train and test predictions. For example, the SVM classifier has a FNR of 20% in training and 37% in the test. The DT classifier still classifiers the fraudulent and non-fraudulent transactions without any misclassification on the train dataset. For classification models built on the one-sided sampled dataset with default parameters it can be noted that there is a slight increase in recall for all classifiers except LR which remained the same. This increase in recall implies that the classifiers were able to correctly classify more fraudulent transactions. The downside to this is the increase in FPR since precision decreased. Consider table 7.12 and 7.13 the classifier performance on the test and training data for hyper-parameterized SVM, kNN, LR, DT and MLP. The gap between the test and train predictions has decreased for all other classifiers except kNN which remained the same. This gap decreased from 17% to 5% for SVM, 3% to 0% for LR, 22% to 3% for DT and 12% to 1% for MLP, this indicates a reduction in overfitting.

Table 7.10: Training results on the one-sided sampled data with default model parameters.

Classification models	Precision	Recall	F_1
SVM	0.98	0.80	0.88
kNN	0.96	0.78	0.86
LR	0.88	0.60	0.71
DT	1.00	1.00	1.00
MLP	0.95	0.93	0.94

Table 7.11: Test results on the one-sided sampled data with default model parameters.

Classification models	Precision	Recall	F_1
SVM	0.92	0.63	0.75
kNN	0.86	0.73	0.79
LR	0.86	0.63	0.72
DT	0.69	0.78	0.73
MLP	0.83	0.81	0.82

Table 7.12: Training results on the one-sided sampled data with tuned model parameters.

Classification models	Precision	Recall	F_1
SVM	0.97	0.83	0.89
kNN	0.96	0.78	0.86
LR	0.81	0.79	0.80
DT	0.96	0.78	0.86
MLP	0.956	0.79	0.86

Table 7.13: Test results on the one-sided sampled data with tuned model parameters.

Classification models	Precision	Recall	F_1
SVM	0.88	0.78	0.83
kNN	0.86	0.73	0.79
LR	0.77	0.79	0.78
DT	0.86	0.75	0.80
MLP	0.86	0.78	0.82

Consider tables 7.14 to 7.18 the classifier performance based on one-sided test data. The model estimates fall with the bootstrap confidence interval and they are close to the bootstrap point estimate for all the classifiers. The LR classifier outperforms all other classifiers with an AUC score of 0.8953 and G-mean of 0.8892. kNN classifier is the least performing classifier with an AUC score of 0.8656 and G-mean of 0.8551.

Table 7.14: SVM bootstrap estimate on the one-sided sampled test set.

	Model estimate	Bootstrap point estimate	95% bootstrap C.I	Standard error
Precision	0.8750	0.9035	[0.8498, 0.9479]	0.0310
Recall	0.7836	0.7782	[0.7251, 0.8162]	0.0300
F_1	0.8268	0.8351	[0.7921, 0.8676]	0.0235
G-mean	0.8851	0.8804	[0.8488, 0.9024]	0.0179
AUC	0.8917	0.8890	[0.8625, 0.9080]	0.0150
MCC	0.8278	0.8370	[0.7943, 0.8697]	0.0231

Table 7.15: kNN bootstrap estimate on the one-sided sampled test set.

	Model estimate	Bootstrap point estimate	95% bootstrap C.I	Standard error
Precision	0.8596	0.8754	[0.7817, 0.9250]	0.0477
Recall	0.7313	0.7610	[0.7216, 0.7972]	0.0225
F_1	0.7903	0.8126	[0.7632, 0.8509]	0.0268
G-mean	0.8551	0.8713	[0.8473, 0.8924]	0.0135
AUC	0.8656	0.8804	[0.8607, 0.8985]	0.0112
MCC	0.7926	0.8151	[0.7642, 0.8532]	0.0274

Table 7.16: LR bootstrap estimate on the one-sided sampled test set.

	Model estimate	Bootstrap point estimate	95% bootstrap C.I	Standard error
Precision	0.7737	0.7800	[0.7100, 0.8200]	0.0300
Recall	0.7910	0.8002	[0.7664, 0.8327]	0.0217
F_1	0.7823	0.7883	[0.7522, 0.8137]	0.0201
G-mean	0.8892	0.8941	[0.8746, 0.9123]	0.0124
AUC	0.8953	0.8999	[0.8830, 0.9162]	0.0108
MCC	0.7820	0.7885	[0.7546, 0.8137]	0.0195

Table 7.17: DT bootstrap estimate on the one-sided sampled test set.

	Model estimate	Bootstrap point estimate	95% bootstrap C.I	Standard error
Precision	0.8632	0.8675	[0.7487, 0.9527]	0.0649
Recall	0.7537	0.7886	[0.6889, 0.8591]	0.0573
F_1	0.8048	0.8164	[0.7500, 0.8649]	0.0418
G-mean	0.8681	0.8811	[0.8322, 0.9193]	0.0301
AUC	0.8768	0.8952	[0.8432, 0.9294]	0.0284
MCC	0.8063	0.8223	[0.7567, 0.8681]	0.0401

Table 7.18: MLP bootstrap estimate on the one-sided sampled test set.

	Model estimate	Bootstrap point estimate	95% bootstrap C.I	Standard error
Precision	0.8595	0.9060	[0.8451, 0.9417]	0.0341
Recall	0.7761	0.7938	[0.6884, 0.8319]	0.0517
F_1	0.8157	0.8468	[0.8247, 0.8645]	0.0142
G-mean	0.8809	0.8922	[0.8617, 0.9161]	0.0174
AUC	0.8880	0.8893	[0.7947, 0.9217]	0.0425
MCC	0.8165	0.8374	[0.7651, 0.8829]	0.0370

Consider table 7.19 and 7.20 the classifier performance on the test and training data for SVM, kNN, LR, DT and MLP. The classification models SVM and DT are likely to be overfitting, since for example the gap between train and test predictions for FNR is large. SVM has a FNR of 21% on train prediction and a FNR of 34% on test prediction. DT is still the only classifier that classifiers the fraudulent and non-fraudulent transactions without any misclassification in training. This is not a surprised, as mentioned in section 4.2.2, DTs

are initially grown to overfit the training set. On the test set, the kNN and SVM classifiers improved on their ability to classify fraudulent transactions without increasing the FPR compared to table 7.11. The classification performance of DT and LR degraded and the FPR increased for both classifiers compared to table 7.11. The MLP classifier improved its ability to classify fraudulent transaction but increased the FPR since precision decreased compared to table 7.11.

Consider table 7.21 and 7.22 the classifier performance on the test and training data for hyper-parameterized SVM, kNN, LR, DT and MLP. The gap between the train and test prediction for FNR has decreased; it decreased from 13% to 4% for SVM, 3% to 2% for kNN, 26% to 1% for DT and 3% to 0% for MLP. The hyper-parameterized LR classifier has the ability to classifier more fraudulent transactions with the cost of increasing the FPR. The gap between train and test prediction has increased compared to the LR trained with default parameters, refer to table 7.21 and 7.22. On the test data, all classifiers have improved in their ability to classify fraudulent transaction with SVM showing a large improvement of 23% compared to table 7.20.

Table 7.19: Training results after applying PCA with default model parameters.

Classification models	Precision	Recall	F_1
SVM	0.98	0.79	0.88
kNN	0.96	0.78	0.86
LR	0.88	0.58	0.70
DT	1.00	1.00	1.00
MLP	0.94	0.78	0.85

Table 7.20: Test results after applying PCA with default model parameters.

Classification models	Precision	Recall	F_1
SVM	0.92	0.66	0.77
kNN	0.86	0.75	0.80
LR	0.85	0.58	0.69
DT	0.63	0.74	0.68
MLP	0.89	0.75	0.82

Table 7.21: Training results after applying PCA with tuned model parameters.

Classification models	Precision	Recall	F_1
SVM	0.97	0.84	0.90
kNN	0.96	0.76	0.85
LR	0.80	0.78	0.79
DT	0.94	0.80	0.86
MLP	0.91	0.81	0.85

Table 7.22: Test results after applying PCA with tuned model parameters.

Classification models	Precision	Recall	F_1
SVM	0.89	0.80	0.84
kNN	0.88	0.74	0.80
LR	0.79	0.81	0.80
DT	0.83	0.79	0.81
MLP	0.85	0.80	0.82

Consider tables 7.23 to 7.27 the classifier performance based on the test data after PCA has been applied. The model estimates still fall with the bootstrap confidence interval and

they are close to the bootstrap point estimate for all the classifiers. The LR classifier has again outperforms all other classifiers with an AUC score of 0.9028 and G-mean of 0.8976. kNN classifier is the worst performing classifier with an AUC score of 0.8693 and G-mean of 0.8595. The AUC and G-mean scores of SVM, LR, DT and MLP are very close.

Table 7.23: SVM bootstrap estimate on the test set after applying PCA.

	Model estimate	Bootstrap point estimate	95% bootstrap C.I	Standard error
Precision	0.8917	0.9230	[0.8766, 0.9602]	0.0257
Recall	0.7985	0.7914	[0.7020, 0.8533]	0.0455
F_1	0.8425	0.8501	[0.8024, 0.8856]	0.0278
G-mean	0.8935	0.8880	[0.8344, 0.9236]	0.0270
AUC	0.8992	0.8957	[0.8510, 0.9266]	0.0228
MCC	0.8436	0.8534	[0.8134, 0.8862]	0.0252

Table 7.24: kNN bootstrap estimate on the test set after applying PCA.

	Model estimate	Bootstrap point estimate	95% bootstrap C.I	Standard error
Precision	0.8761	0.8626	[0.7805, 0.9270]	0.0440
Recall	0.7388	0.7694	[0.7349, 0.8030]	0.0240
F_1	0.8016	0.8123	[0.7737, 0.8544]	0.0278
G-mean	0.8595	0.8066	[0.8562, 0.8953]	0.0141
AUC	0.8693	0.8846	[0.8673, 0.9014]	0.0120
MCC	0.8042	0.8139	[0.7742, 0.8568]	0.0282

Table 7.25: LR bootstrap estimate on the test set after applying PCA.

	Model estimate	Bootstrap point estimate	95% bootstrap C.I	Standard error
Precision	0.7883	0.7900	[0.7300, 0.8400]	0.0300
Recall	0.8060	0.8027	[0.7689, 0.8327]	0.0196
F_1	0.7970	0.7936	[0.7634, 0.8173]	0.0172
G-mean	0.8976	0.8955	[0.8760, 0.9123]	0.0111
AUC	0.9028	0.9012	[0.8843, 0.9162]	0.0098
MCC	0.7968	0.7937	[0.7649, 0.8176]	0.0168

Table 7.26: DT bootstrap estimate on the test set after applying PCA.

	Model estimate	Bootstrap point estimate	95% bootstrap C.I	Standard error
Precision	0.8281	00.8473	[0.7831, 0.9223]	0.0469
Recall	0.7910	0.7435	[0.6384, 0.7957]	0.0530
F_1	0.8092	0.7896	[0.7113, 0.8455]	0.0404
G-mean	0.8893	0.8596	[0.7964, 0.8917]	0.0324
AUC	0.8954	0.8716	[0.8191, 0.8977]	0.0265
MCC	0.8091	0.7910	[0.7146, 0.8484]	0.0405

Table 7.27: MLP bootstrap estimate on the test set after applying PCA.

	Model estimate	Bootstrap point estimate	95% bootstrap C.I	Standard error
Precision	0.8492	0.8707	[0.8040, 0.9453]	0.0517
Recall	0.7985	0.7948	[0.7000, 0.8319]	0.0425
F_1	0.8231	0.8277	[0.7918, 0.8640]	0.0255
G-mean	0.8935	0.8748	[0.8049, 0.9307]	0.0392
AUC	0.8991	0.8976	[0.8748, 0.9244]	0.0153
MCC	0.8232	0.8130	[0.7384, 0.8550]	0.0392

Table 7.28 summarizes the classifier's performance on the random under sampled dataset, that is where the non-fraudulent transactions were reduced to 448 observations to balance the class labels. LR classifier has outperformed all the other classifiers.

Table 7.28: Estimates for the random undersampled dataset on test set.

	Precision	Recall	F_1	G-meam	AUC	MCC
SVM	0.9750	0.8706	0.9196	0.9224	0.9241	0.8534
kNN	0.9574	0.8906	0.9221	0.9242	0.9252	0.8534
LR	0.9600	0.9041	0.9309	0.9324	0.9330	0.8679
DT	0.8960	0.9064	0.8998	0.8975	0.8984	0.7993
MLP	0.9514	0.9063	0.9278	0.9291	0.9296	0.8611

Table 7.29 summarizes the classifier's performance on over-sampled dataset using SMOTE in combination with Tomek links, that is the fraudulent transactions were increased to 281 470 observations to balance the class labels. MLP has outperformed all the other classifiers with zero FNR. The MLP classifier has the ability to classify all fraudulent without any misclassifications and with low FPR. The LR classifier is the worst performing classifier. All MLP classifier performance measures have significantly improved compared to table 7.28. The DT classifier has slightly decreased compared to table 7.28.

Table 7.29: Classifiers built after SMOTE was applied on the test data.

	Precision	Recall	F_1	G-meam	AUC	MCC
SVM	0.9826	0.9743	0.9784	0.9785	0.9785	0.9571
kNN	0.9977	1.0000	0.9988	0.9988	0.9988	0.9977
LR	0.9716	0.9177	0.9438	0.9450	0.9454	0.8922
DT	0.9968	0.9986	0.9977	0.9977	0.9977	0.9954
MLP	0.9993	1.0000	0.9996	0.9996	0.9996	0.9993

7.3 Comparative Studies

Raghavan & El Gayar (2019) accessed the effectiveness of fraud detection using various classification techniques, namely SVM, RF, kNN, RBM, CNN and deep belief neural networks (DBN). The purpose of the study was to present an empirical comparison between different traditional classifiers and deep learning models. They investigated the performance of various classifier using three different datasets, namely European, Australian and German credit card dataset. To improve classifier performance, methods such as data cleaning and hyper-parameter searching were considered. These authors used CV on the training set to determine the value of k that optimizes the kNN classifier. A GS algorithm was used to find hyper-parameters for other classifiers and these hyper-parameters were used to build the models. All implementation was conducted in python. The hyper-parameterized SVM classifier applied to the European credit card dataset had an AUC score of 0.8887 and a MCC score of 0.8145. The AUC score is 0.0156 higher compared to the AUC score in table 7.5. The MCC score is 0.0089 lower compared to the MCC score in table 7.5. The AUC and MCC score in this article falls within the 95% bootstrap C.I provided in table 7.5. In this study, a RS algorithm was used to search for SVM hyper-parameters. The AUC and MCC scores in this study are close to those reported by Raghavan & El Gayar (2019) although different hyper-parameter search algorithms were considered. This is not a surprise as RS has all the practical advantages of GS (Bergstra & Bengio, 2012). Raghavan & El Gayar (2019)'s hyper-perimeterized kNN classifier had an AUC score of 0.9004 and a MCC score of 0.8354. The AUC score is 0.0422 higher compared to the AUC score in table 7.5. The MCC score is 0.0185 higher compared to the MCC score in table 7.5. The AUC and MCC estimates for the kNN classifier reported in Raghavan & El Gayar (2019) fall with the 95% bootstrap C.I reported in table 7.6.

Puh & Brkić (2019) compared classification performance of three different classifiers, namely RF, SVM and LR to detect fraudulent transactions. They used the SMOTE algorithm to address class imbalance. In the SMOTE algorithm, the sampling strategy parameter which denotes the desired ratio of the number of observations in the majority class over the number of samples in the minority class was set to be 40%. This ratio was chosen empirically by conducting a variety of preliminary tests on the training data. In this study we used SMOTE with Tomek links and the ratio was set to 1:1, that is one fraudulent transaction for each non-fraudulent transaction. Puh & Brkić (2019) used GS with CV to determine hyper-parameters for the classifiers. The hyper-parameterized SVM classifier had an AUC score of 0.8877 and an average precision score of 0.7978 obtained by CV. The hyper-parameterized LR had an AUC score of 0.9114 and an average precision of 0.7337. It can be noted that the AUC score reported in this article is close to the AUC score obtained in this study for hyper-parameterized LR trained normalized dataset, one-sided data and after PCA was applied, see tables 7.7, 7.16 and 7.25. The same can be observed for the AUC score of the SVM classifier. The AUC scores reported in Puh & Brkić (2019) falls within our 95% bootstrap C.I. The average precision score reported in this article is lower compared to tables 7.7, 7.16,

and 7.25. These average precision scores falls outside the 95% bootstrap C.I.

Rajora et al. (2018) conducted a comparative study on ten different classifiers. The classifiers were built on the European credit card dataset with and without the feature variable 'time' to determine its effectiveness when detecting fraudulent transactions. The feature variables, 'time' and 'amount' were normalized before any classifier was built. Rajora et al. (2018) used RU to balance the class distribution and the dataset was split into 70:30, that is 70% of the dataset was used for training and 30% was used for testing. To compare the results of Rajora et al. (2018) with this study we consider the performance of the classifiers when the feature variable 'time' is considered. In Rajora et al. (2018) the SVM classifier had a precision score of 0.94, recall of 0.94 and AUC of 0.94. The kNN classifier has a precision score of 0.94, recall of 0.94 and AUC of 0.94. The LR classifier had a precision score of 0.95, recall of 0.94 and AUC of 0.94 and finally the DT classifier had a precision score of 0.91, recall of 0.91 and AUC of 0.91. The performance scores obtained in this article are close compared to those in table 7.28. The difference in the precision and AUC scores is no more than 4% for all the classifiers. Rajora et al. (2018) had better recall scores compared to these reported in table 7.28, this is possibly due to the duplicated transactions that were not removed and the zero transaction amounts that were labelled as fraudulent since the number of fraudulent transactions remained 492 in their study.

Chapter 8

Conclusion

Credit card fraud costs financial companies millions of rands each year. Fraudsters are continually trying to find new ways to bypass the existing fraud detection systems. As a result, continuously updated fraud detection systems have become essential for banks and financial institutions to minimize their losses. For fraud detection, this study investigated five supervised classifiers, namely SVM, kNN, LR, DT and MLP. The classifiers were built on different datasets, namely the original dataset and the resampled datasets (one-sided, RU and SMOTE with Tomek link) as the European credit card dataset is highly imbalanced. PCA was applied on the one-sided sampled dataset and 17 features were extracted. All hyper-parameters were obtained using RS with ten-fold CV to avoid overfitting. The hyper-parameterized classifiers were validated using the bootstrap point estimate and a 95% C.I was attained. Performance measures for each classifier were reasonably close to the bootstrap point estimate and fall with the 95% bootstrap C.I.

The LR classifier outperformed all other classifiers except when the dataset was oversampled in which case it was the worst performing classifier with a G-mean of 0.9450% and a AUC of 0.9454%. The best LR results were obtained when the dataset was RO followed by when the dataset was just normalized. There is a slight difference in the classification performance of an LR classifier for when the dataset was normalized and when RO was applied. This suggests that normalizing the dataset and searching for hyper-parameters is effective and could potentially be preferred over just randomly oversampling the majority class, which could potentially ignore useful hidden patterns that lead to a better fraud detection system. PCA did not improve the classification performance for any classifier. This is not a surprise as mentioned in section 6.2.1, since PCA trades accuracy for simplicity and there are not too many features in these datasets.

8.1 Limitations of this Study

Credit card fraud detection is extremely challenging. Credit card companies maintain the datasets of their transactions. However, they do not release these data to the public domain due to privacy and security concerns. The publicly released datasets don't disclose the features of the data, for example, merchant categories, geographic locations, dates and times of the transactions. This leads to classifiers being built on the dataset with limited information.

8.2 Future Work

A cost-sensitive learning approach can be implemented by considering the misclassification costs. The cost for misclassifying a fraudulent transaction as non-fraudulent is higher than the cost for misclassifying a fraudulent transaction as fraudulent. For DTs, this would include using a cost-sensitive splitting criterion with a cost insensitive pruning method, see section 4.2.3 or by using Fisher's exact test. For LR, it would include using the weighted log-likelihood, see section 4.3.1. The classification performance of the kNN classifier could be improved by implementing the kNN CCW classifier since this uses the posterior probability and does not only depend on prior probability to predict the class label, see section 4.1.3. The classification performance of the SVM classifier may be improved by deriving a customized kernel suitable to this particular problem. The classification performance of an MLP classifier may improve by adding more neurons, layers and increasing the search space to find hyper-parameters that capture more fraudulent transactions with increasing FPR.

Credit card fraud is related to the non-stationary nature of transaction distributions since fraudsters are adaptive and are usually able to find ways of bypassing the classifiers built to detect fraudulent transactions. Therefore, it is essential to consider the changing behaviours while developing an effective fraud detecting classifier. A thorough study of the non-stationary nature of credit card fraud detection cold be performed, however, this requires vast amounts of data (Kulkarni & Ade, 2016).

References

Abdallah, A., Maarof, M. A., & Zainal, A. (2016). Fraud detection system: A survey. *Journal of Network and Computer Applications*, 100(68), 90–113.

Abdi, H. & Williams, L. J. (2010). Principal component analysis. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(4), 433–459.

Agarwal, R. (2018). The 5 most useful techniques to handle imbalanced datasets. `https://www.kdnuggets.com/2020/01/5-most-useful-techniques-handle-imbalanced-datasets.html`, accessed April 2021.

Amanze, B. & Onukwugha, C. (2018). Data mining application in credit card fraud detection system. *International Journal of Trend in Research and Development*, 5(4), 23–26.

Bekkar, M., Djemaa, H. K., & Alitouche, T. A. (2013). Evaluation measures for models assessment over imbalanced data sets. *Journal of Information Engineering and Applications*, 3(10), 27–38.

Bergstra, J., Bardenet, R., Bengio, Y., & Kégl, B. (2011). Algorithms for hyper-parameter optimization. *Advances in Neural Information Processing Systems*, 24, 2546–2554.

Bergstra, J. & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(2), 281–305.

Berrar, D. (2018). Understanding cross validation's purpose. `https://www.researchgate.net/profile/Daniel-Berrar/publication/324701535_Cross-Validation/links/5cb4209c92851c8d22ec4349/Cross-Validation.pdf`, accessed October 2021.

Bhattacharyya, S., Jha, S., Tharakunnel, K., & Westland, J. C. (2011). Data mining for credit card fraud: A comparative study. *Decision Support Systems*, 50(3), 602–613.

Bolton, R. J. & Hand, D. J. (2002). Statistical fraud detection: A review. *Statistical Science*, 17(3), 235–255.

Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the 5th Annual Workshop on Computational Learning Theory* (pp. 144–152).: Citeseer.

Bro, R. & Smilde, A. K. (2014). Principal component analysis. *Analytical Methods*, 6(9), 2812–2831.

Calì, C. & Longobardi, M. (2015). Some mathematical properties of the ROC curve and their applications. *Ricerche di Matematica*, 64(2), 391–402.

Chan, P. K., Fan, W., Prodromidis, A. L., & Stolfo, S. J. (1999). Distributed data mining in credit card fraud detection. *Intelligent Systems and Their Applications*, 14(6), 67–74.

Chawala, N. V. (2009). Data Mining for Imbalanced Datasets: An Overview. In R. L. Maimon O. (Ed.), *Data Mining and Knowledge Discovery Handbook.* chapter 40, (pp. 875–886). Boston: Springer.

Chawla, N. V. (2003). C4.5 and imbalanced data sets: Investigating the effect of sampling method, probabilistic estimate, and decision tree structure. In *Proceedings of the International Conference on Machine Learning*: Citeseerx.

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16, 321–357.

Chen, R. C., Chen, T. S., & Lin, C. C. (2006). A new binary support vector system for increasing detection rate of credit card fraud. *International Journal of Pattern Recognition and Artificial Intelligence*, 20(2), 227–239.

Chen, X. W. & Jeong, J. C. (2007). Enhanced recursive feature elimination. In *Sixth International Conference on Machine Learning and Applications* (pp. 429–435).: IEEE.

Cheung, G. W. & Rensvold, R. B. (2002). Evaluating goodness-of-fit indexes for testing measurement invariance. *Structural Equation Modeling*, 9(2), 233–255.

Chi, M., Feng, R., & Bruzzone, L. (2008). Classification of hyperspectral remote-sensing data with primal SVM for small-sized training dataset problem. *Advances in Space Research*, 41(11), 1793–1799.

Chicco, D. & Jurman, G. (2020). The advantages of the Matthews correlation coefficient (MCC) over f1 score and accuracy in binary classification evaluation. *BMC Genomics*, 21(1), 1–13.

Chicco, D., Tötsch, N., & Jurman, G. (2021). The Matthews correlation coefficient (MCC) is more reliable than balanced accuracy, bookmaker informedness, and markedness in two-class confusion matrix evaluation. *BioData Mining*, 14(1), 1–22.

Cieslak, D. A. & Chawla, N. V. (2008). Learning decision trees for unbalanced data. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases* (pp. 241–256).: Springer.

Cieslak, D. A., Hoens, T. R., Chawla, N. V., & Kegelmeyer, W. P. (2012). Hellinger distance decision trees are robust and skew-insensitive. *Data Mining and Knowledge Discovery*, 24(1), 136–158.

Claesen, M. & De Moor, B. (2015). Hyperparameter search in machine learning. In *Proceedings of the 11th Metaheuristics International Conference* (pp. 1–5).

Cristianini, N. & Shawe-Taylor, J. (2000). *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press.

Dal Pozzolo, A., Caelen, O., Johnson, R. A., & Bontempi, G. (2015). Calibrating probability with undersampling for unbalanced classification. In *Symposium Series on Computational Intelligence* (pp. 159–166).: IEEE.

Darwish, S. M. (2020). A bio-inspired credit card fraud detection model based on user behavior analysis suitable for business management in electronic banking. *Journal of Ambient Intelligence and Humanized Computing*, 11(2), 4873–4887.

Davison, A. C. & Hinkley, D. V. (1997). *Bootstrap Methods and Their Application*. Cambridge University Press.

De Mántaras, R. L. (1991). A distance-based attribute selection measure for decision tree induction. *Machine Learning*, 6(1), 81–92.

DeMaris, A. (1995). A tutorial in logistic regression. *Journal of Marriage and the Family*, 57(4), 956–968.

Dobson, A. J. & Barnett, A. G. (2018). *An Introduction to Generalized Linear Models*. CRC Press.

Dornadula, V. N. & Geetha, S. (2019). Credit card fraud detection using machine learning algorithms. *Procedia Computer Science*, 165, 631–641.

Drummond, C. & Holte, R. (2000a). Exploiting the cost (in)sensitivity of decision tree splitting criteria. *International Conference on Machine Learning*, 1(1), 239–246.

Drummond, C. & Holte, R. C. (2000b). Exploiting the cost (in)sensitivity of decision tree splitting criteria. In *ICML*.

Du, W. & Zhan, Z. (2002). Building decision tree classifier on private data. *Electrical Engineering and Computer Science*, 14(8), 1–8.

Dudani, S. A. (1976). The distance-weighted k-nearest-neighbor rule. *IEEE Transactions on Systems, Management, and Cybernetics*, 6(4), 325–327.

Efron, B. (1979). Bootstrap methods: Another look at the jackknife. *Annals of Statistics*, 7(1), 1–26.

Efron, B. & Gong, G. (1983). A leisurely look at the bootstrap, the jackknife, and cross-validation. *The American Statistician*, 37(1), 36–48.

Efron, B. & Tibshirani, R. (1997). Improvements on cross-validation: The 632+ bootstrap method. *Journal of the American Statistical Association*, 92(438), 548–560.

Fan, G. F., Guo, Y. H., Zheng, J.-M., & Hong, W. C. (2019). Application of the weighted k-nearest neighbor algorithm for short-term load forecasting. *Energies*, 12(5), 1–19.

Firth, D. (1993). Bias reduction of maximum likelihood estimates. *Biometrika*, 80(1), 27–38.

Flach, P. A. (2003). The geometry of ROC space: Understanding machine learning metrics through ROC isometrics. In *Proceedings of the 20th International Conference on Machine Learning* (pp. 194–201).: Citeseerx.

Friedman, J., Hastie, T., & Tibshirani, R. (2001). *The Elements of Statistical Learning*. Springer Series in Statistics, New York.

Fu, K., Cheng, D., Tu, Y., & Zhang, L. (2016). Credit card fraud detection using convolutional neural networks. In *International Conference on Neural Information Processing* (pp. 483–490).: Springer.

Ganganwar, V. (2012). An overview of classification algorithms for imbalanced datasets. *International Journal of Emerging Technology and Advanced Engineering*, 2(4), 42–47.

García-Gómez, J. M. & Tortajada, S. (2015). Definition of loss functions for learning from imbalanced data to minimize evaluation metrics. In *Data Mining in Clinical Medicine* chapter 2, (pp. 19–37). Springer.

Ghosh, S. & Reilly, D. L. (1994). Credit card fraud detection with a neural-network. In *1994 Proceedings of the Twenty-Seventh Hawaii International Conference on System Sciences* (pp. 621–630).: IEEE.

Goadrich, M., Oliphant, L., & Shavlik, J. (2006). Gleaner: Creating ensembles of first-order clauses to improve recall-precision curves. *Machine Learning*, 64(1-3), 231–261.

Gou, J., Du, L., Zhang, Y., Xiong, T., et al. (2012). A new distance-weighted k-nearest neighbor classifier. *Journal of Information and Computational Science*, 9(6), 1429–1436.

Guyon, I. & Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3(3), 1157–1182.

Guyon, I., Weston, J., Barnhill, S., & Vapnik, V. (2002). Gene selection for cancer classification using support vector machines. *Machine Learning*, 46(1), 389–422.

Hido, S., Kashima, H., & Takahashi, Y. (2009). Roughly balanced bagging for imbalanced data. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 2(5-6), 412–426.

Hilas, C. S. & Sahalos, J. N. (2007). An application of decision trees for rule extraction towards telecommunications fraud detection. In *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems* (pp. 1112–1121).: Springer.

Hosmer Jr, D. W., Lemeshow, S., & Sturdivant, R. X. (2013). *Applied Logistic Regression*. John Wiley & Sons.

Howley, T. & Madden, M. G. (2005). The genetic kernel support vector machine: Description and evaluation. *Artificial Intelligence Review*, 24(3-4), 379–395.

Hu, N., Liu, L., & Sambamurthy, V. (2011). Fraud detection in online consumer reviews. *Decision Support Systems*, 50(3), 614–626.

Huang, S., Cai, N., Pacheco, P. P., Narrandes, S., Wang, Y., & Xu, W. (2018). Applications of support vector machine learning in cancer genomics. *Cancer Genomics Proteomics*, 15(1), 41–51.

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

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013b). *An Introduction to Statistical Learning with Applications in R*. Springer, New York.

Jenhani, I., Amor, N. B., & Elouedi, Z. (2008). Decision trees as possibilistic classifiers. *International Journal of Approximate Reasoning*, 48(3), 784–807.

Jha, S., Guillen, M., & Westland, J. C. (2012). Employing transaction aggregation strategy to detect credit card fraud. *Expert Systems with Applications*, 39(16), 12650–12657.

Johnson, R. W. (2001). An introduction to the bootstrap. *Teaching Statistics*, 23(2), 49–54.

Jurman, G., Riccadonna, S., & Furlanello, C. (2012). A comparison of MCC and CEN error measures in multi-class prediction. *PLoS ONE*, 7(8), 1–8.

Kearns, M. J. (1990). *The Computational Complexity of Machine Learning*. MIT Press.

King, G. & Zeng, L. (2001). Logistic regression in rare events data. *Political Analysis*, 9(2), 137–163.

Kiran, S., Kumar, N., Guru, J., Katariya, D., Kumar, R., & Sharma, M. (2018). Credit card fraud detection using Naïve Bayes model based and kNN classifier. *International Journal of Advance Research, Ideas and Innovations in Technoloy*, 4(3), 44–47.

Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *International Joint Conference on Artificial Intelligence* (pp. 1137–1145).: Montreal.

Krawczyk, B. (2016). Learning from imbalanced data: Open challenges and future directions. *Progress in Artificial Intelligence*, 5(4), 221–232.

Kubat, M., Matwin, S., et al. (1997). Addressing the curse of imbalanced training sets: One-sided selection. In *International Conference on Machine Learning* (pp. 179–186).: Citeseerx.

Kubat, M. & Widmer, G. (1995). Adapting to drift in continuous domains. In *European Conference on Machine Learning* (pp. 307–310).: Springer.

Kulkarni, P. & Ade, R. (2016). Logistic regression learning model for handling concept drift with unbalanced data in credit card fraud detection system. In *Proceedings of the Second International Conference on Computer and Communication Technologies* (pp. 681–689).: Springer.

Lebret, R., Iovleff, S., Langrognet, F., Biernacki, C., Celeux, G., & Govaert, G. (2015). Rmixmod: The R package of the model-based unsupervised, supervised and semi-supervised classification mixmod library. *Journal of Statistical Software*, 67(6), 241–270.

Ling, Y., Zhang, X., & Zhang, Y. (2021). Improved kNN algorithm based on probability and adaptive k value. In *2021 7th International Conference on Computing and Data Engineering* (pp. 34–40).: ACM.

Liu, A. Y. C. (2004). *The effect of oversampling and undersampling on classifying imbalanced text datasets*. PhD thesis, Citeseerx.

Liu, W. & Chawla, S. (2011). Class confidence weighted kNN algorithms for imbalanced data sets. In *Pacific Asia Conference on Knowledge Discovery and Data Mining* (pp. 345–356).: Springer.

Liu, W., Chawla, S., Cieslak, D. A., & Chawla, N. V. (2010). A robust decision tree algorithm for imbalanced data sets. In *Proceedings of the 2010 SIAM International Conference on Data Mining* (pp. 766–777).: SIAM.

Liu, X. Y., Wu, J., & Zhou, Z. H. (2008). Exploratory undersampling for class-imbalance learning. *IEEE Transactions on Systems, Management, and Cybernetics, Part B (Cybernetics)*, 39(2), 539–550.

Liu, Y. & Huang, H. (2002). Fuzzy support vector machines for pattern recognition and data mining. *International Journal of Fuzzy Systems*, 4(3), 826–835.

Loh, W. Y. & Shih, Y.-S. (1997). Split selection methods for classification trees. *Statistica Sinica*, 7(4), 815–840.

Ma, Y. & He, H. (2013). *Imbalanced Learning: Foundations, Algorithms, and Applications*. Wiley.

Maalouf, M. & Trafalis, T. B. (2011). Robust weighted kernel logistic regression in imbalanced and rare events data. *Computational Statistics & Data Analysis*, 55(1), 168–183.

Maes, S., Tuyls, K., Vanschoenwinkel, B., & Manderick, B. (2002). Credit card fraud detection using bayesian and neural networks. In *Proceedings of the 1st International Naiso Congress on Neuro Fuzzy Technologies* (pp. 261–270).: Citeseerx.

Manski, C. F. & Lerman, S. R. (1977). The estimation of choice probabilities from choice based samples. *Econometrica: Journal of the Econometric Society*, 45(8), 1977–1988.

McGovern, S. (2016). Deep learning frameworks slides. `https://www.slideshare.net/SheamusMcGovern/deep-learning-frameworks-slides`, accessed November 2021.

Mekterović, I., Karan, M., Pintar, D., & Brkić, L. (2021). Credit card fraud detection in card-not-present transactions: Where to invest? *Applied Sciences*, 11(15).

Minastireanu, E. A. & Mesnita, G. (2019). An analysis of the most used machine learning algorithms for online fraud detection. *Informatica Economica*, 23(1), 5–16.

Minka, T. P. (2003). A comparison of numerical optimizers for logistic regression. *CMU Technical Report*, (pp. 1–18).

Mitchell, T. M. (2011). Machine learning. `https://www.cs.cmu.edu/~tom/10701_sp11/slides/Kernels_SVM_04_7_2011-ann.pdf`, accessed November 2021.

Morgan, N. & Bourlard, H. (1989). Generalization and parameter estimation in feedforward nets: Some experiments. *Advances in neural information processing systems*, 2, 630–637.

Myles, A. J., Feudale, R. N., Liu, Y., Woody, N. A., & Brown, S. D. (2004). An introduction to decision tree modeling. *Journal of Chemometrics Society*, 18(6), 275–285.

Myung, I. J. (2000). The importance of complexity in model selection. *Journal of Mathematical Psychology*, 44(1), 190–204.

Nami, S. & Shajari, M. (2018). Cost-sensitive payment card fraud detection based on dynamic random forest and k-nearest neighbors. *Expert Systems with Applications*, 110, 381–392.

Navlani, A. (2018). kNN classification using scikit-learn. `https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn`, accessed April 2021.

Neelamegam, S. & Ramaraj, E. (2013). Classification algorithm in data mining: An overview. *International Journal of P2P Network Trends and Technology*, 4(8), 369–374.

Ngai, E. W., Hu, Y., Wong, Y. H., Chen, Y., & Sun, X. (2011). The application of data mining techniques in financial fraud detection: A classification framework and an academic review of literature. *Decision Support Systems*, 50(3), 559–569.

Nguyen, T. & Sanner, S. (2013). Algorithms for direct 0–1 loss optimization in binary classification. In *International Conference on Machine Learning* (pp. 1085–1093).: PMLR.

Nigam, K., Lafferty, J., & McCallum, A. (1999). Using maximum entropy for text classification. In *Workshop on Machine Learning for Information Filtering* (pp. 61–67).: Stockholom.

Ogwueleka, F. N. (2011). Data mining application in credit card fraud detection system. *Journal of Engineering Science and Technology*, 6(3), 311–322.

Palade, V. (2013). Class imbalance learning methods for support vector machines. In *Imbalanced Learning: Foundations, Algorithms, and Applications* chapter 6, (pp. 83–99). Wiley Online Library.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in python. `https://scikit-learn.org/stable/modules/neural_networks_supervised.html#multi-layer-perceptron`, accessed November 2021.

Puh, M. & Brkić, L. (2019). Detecting credit card fraud using selected machine learning algorithms. In *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics* (pp. 1250–1255).: IEEE.

Pumsirirat, A. & Yan, L. (2018). Credit card fraud detection using deep learning based on auto-encoder and restricted boltzmann machine. *International Journal of Advanced Computer Science and Applications*, 9(1), 18–25.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1(1), 81–106.

Raghavan, P. & El Gayar, N. (2019). Fraud detection using machine learning and deep learning. In *2019 International Conference on Computational Intelligence and Knowledge Economy* (pp. 334–339).: IEEE.

Rajora, S., Li, D.-L., Jha, C., Bharill, N., Patel, O. P., Joshi, S., Puthal, D., & Prasad, M. (2018). A comparative study of machine learning techniques for credit card fraud detection based on time variance. In *Symposium Series on Computational Intelligence* (pp. 1958–1963).: IEEE.

Ramchoun, H., Idrissi, M. A. J., Ghanou, Y., & Ettaouil, M. (2016). Multilayer perceptron: Architecture optimization and training. *International Journal of Interactive Multimedia and Artificial Intelligence*, 4(1), 26–30.

Rashidi, H. H., Tran, N. K., Betts, E. V., Howell, L. P., & Green, R. (2019). Artificial intelligence and machine learning in pathology: The present landscape of supervised methods. *Academic pathology*, 6.

Rice, J. A. (2006). *Mathematical Statistics and Data Analysis*. Cengage Learning.

Roughgarden, T. & Valiant, G. (2021). CS168: The modern algorithmic toolbox . `https://web.stanford.edu/class/cs168/l/l8.pdf`, accessed October 2021.

Ruck, D. W., Rogers, S. K., Kabrisky, M., Oxley, M. E., & Suter, B. W. (1990). The multilayer perceptron as an approximation to a bayes optimal discriminant function. *IEEE Transactions on Neural Networks*, 1(4), 296–298.

Ruppert, D. & Matteson, D. S. (2011). *Statistics and Data Analysis for Financial Engineering*. Springer.

SABRIC (2021). SABRIC Annual Report 2020. `https://www.sabric.co.za/media/lejmweri/sabric_annual-report_2020.pdf`.

Sahin, Y. & Duman, E. (2011). Detecting credit card fraud by ANN and logistic regression. In *2011 International Symposium on Innovations in Intelligent Systems and Applications* (pp. 315–319).: IEEE.

Sánchez, D., Vila, M., Cerda, L., & Serrano, J. M. (2009). Association rules applied to credit card fraud detection. *Expert Systems with Applications*, 36(2), 3630–3640.

Sasaki, Y. (2007). The truth of the f-measure. *Teach Tutor Mater*.

Schaffer, C. (1993). Overfitting avoidance as bias. *Machine Learning*, 10(2), 153–178.

Schölkopf, B. (2001). The kernel trick for distances. In *Advances in Neural Information Processing Systems* (pp. 301–307).: MIT Press.

Sejdinovic, D. (2021). Model complexity and generalization. `http://www.stats.ox.ac.uk/~sejdinov/teaching/sdmml15/materials/HT15_lecture12-nup.pdf`, accessed October 2021.

Shen, A., Tong, R., & Deng, Y. (2007). Application of classification models on credit card fraud detection. In *2007 International Conference on Service Systems and Service Management* (pp. 1–4).: IEEE.

Sokolova, M., Japkowicz, N., & Szpakowicz, S. (2006). Beyond accuracy, F1-score and ROC: A family of discriminant measures for performance evaluation. In *Australasian Joint Conference on Artificial Intelligence* (pp. 1015–1021).: Springer.

Stolfo, S., Fan, D. W., Lee, W., Prodromidis, A., & Chan, P. (1997). Credit card fraud detection using meta-learning: Issues and initial results. In *AAAI-97 Workshop on Fraud Detection and Risk Management* (pp. 83–90).

Talbot, K. I., Duley, P. R., & Hyatt, M. H. (2003). Specific emitter identification and verification. *Technology Review*, 113, 113–130.

Terrible, M. (2017). Understanding cross validation's purpose. `https://medium.com/@mtterribile/understanding-cross-validations-purpose-53490faf6a86`, accessed September 2021.

Tharwat, A. (2020). Classification assessment methods. *New England Journal of Entrepreneurship*, 17(1), 168–192.

Thennakoon, A., Bhagyani, C., Premadasa, S., Mihiranga, S., & Kuruwitaarachchi, N. (2019). Real-time credit card fraud detection using machine learning. In *2019 9th International Conference on Cloud Computing, Data Science & Engineering* (pp. 488–493).: IEEE.

Tian, L. (2021). Tilted empirical risk minimization . `https://blog.ml.cmu.edu/2021/04/02/term/`, accessed June 2021.

Tomek, I. (1976). Two modifications of CNN. *Transactions on Systems, Management, and Cybernetics*, SMC-6(11), 769–772.

Vapnik, V. (1992). Principles of risk minimization for learning theory. In *Advances in Neural Information Processing Systems* (pp. 831–838).

Varmedja, D., Karanovic, M., Sladojevic, S., Arsenovic, M., & Anderla, A. (2019). Credit card fraud detection-machine learning methods. In *18th International Symposium INFOTEH-JAHORINA* (pp. 1–5).: IEEE.

Veropoulos, K., Campbell, C., Cristianini, N., et al. (1999). Controlling the sensitivity of support vector machines. In *Proceedings of the International Joint Conference on Artificial Inteligence* (pp. 55–60).: Stockholm.

Vuk, M. & Curk, T. (2006). ROC curve, lift chart and calibration plot. *Metodoloski Zvezki*, 3(1), 89–108.

Wackerly, D., Mendenhall, W., & Scheaffer, R. (2014). *Mathematical Statistics with Applications*. Cengage Learning.

Walimbe, R. (2017). Data science central. https://www.datasciencecentral.com/profiles/blogs/handling-imbalanced-data-sets-in-supervised-learning-using-family, accessed April 2021.

West, J. & Bhattacharya, M. (2016). Intelligent financial fraud detection: A comprehensive review. *Computers & Security*, 100(57), 47–66.

Weston, J. (2014). Support vector machine tutorial. http://www.cs.columbia.edu/~kathy/cs4701/documents/jason_svm_tutorial.pdf, accessed July 2021.

Widmer, G. & Kubat, M. (1996). Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1), 69–101.

Wistuba, M., Schilling, N., & Schmidt-Thieme, L. (2015). Learning hyperparameter optimization initializations. In *International Conference on Data Science and Advanced Analytics* (pp. 1–10).: IEEE.

Wold, S., Esbensen, K., & Geladi, P. (1987). Principal component analysis. *Chemometrics and Intelligent Laboratory Systems*, 2(1-3), 37–52.

Wong, S. C., Gatt, A., Stamatescu, V., & McDonnell, M. D. (2016). Understanding data augmentation for classification: When to warp? In *International Conference on Digital Image Computing: Techniques and Applications* (pp. 1–6).: IEEE.

Yang, L. & Shami, A. (2020). On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415, 295–316.

Yang, T., Cao, L., & Zhang, C. (2010). A novel prototype reduction method for the k-nearest neighbor algorithm with k $\geq$ 1. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining* (pp. 89–100).: Springer.

Zadrozny, B. & Elkan, C. (2001). Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers. In *International Conference on Machine Learning* (pp. 609–616).: Citeseer.

Zavrel, J. (1997). An empirical re-examination of weighted voting for k-NN. In *Proceedings of the 7th Belgian-Dutch Conference on Machine Learning* (pp. 139–148).: Citeseerx.

Zhang, Y. D. & Wu, L. (2012). An MR brain images classifier via principal component analysis and kernel support vector machine. *Progress In Electromagnetics Research*, 130, 369–388.

Zhao, T. (2017). Lecture 1: Introduction to supervised learning. https://www2.isye.gatech.edu/~tzhao80/Lectures/8803.pdf, accessed September 2021.

Zhou, W. & Kapoor, G. (2011). Detecting evolutionary financial statement fraud. *Decision Support Systems*, 50(3), 570–575.

Zhou, Z. H. (2019). *Ensemble Methods: Foundations and Algorithms*. Chapman and Hall/CRC.

Zhu, X., Ao, X., Qin, Z., Chang, Y., Liu, Y., He, Q., & Li, J. (2021). Intelligent financial fraud detection practices in post-pandemic era: A survey. *The Innovation*, 2(4).

Appendix: Python Code

This appendix contains all the python code used in this analysis.

```
# -*- coding: utf-8 -*-
"""Credit Card dataset analysis.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1F2V6qfVpkH24-VIfxc24WCs-rRaTmLN-
"""

# Commented out IPython magic to ensure Python compatibility.
import warnings
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn import metrics
import matplotlib.pyplot as plt
warnings.filterwarnings("ignore")
# %matplotlib inline
import statistics
import pickle
import scipy.stats as stats
import statsmodels.api as sm
from scipy.stats import ttest_ind
from statsmodels.formula.api import ols
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import matplotlib.patches as  mpatches
from sklearn.metrics import average_precision_score
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import plot_precision_recall_curve
from sklearn.metrics import roc_curve, roc_auc_score
from statistics import mean
from sklearn.feature_selection import RFECV
from imblearn.combine import SMOTETomek
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import confusion_matrix, classification_report
```

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import matthews_corrcoef
from sklearn.model_selection import RandomizedSearchCV
from imblearn.metrics import geometric_mean_score
from sklearn.metrics import accuracy_score, f1_score, precision_score,
    recall_score
from google.colab import files
import io
#importing creditcard
D= pd.read_csv("https://query.data.world/s/3dwtejin6vc6r44dgd3vu66flypzon")
D.drop(D.columns[D.columns.str.contains('unnamed',case = False)],axis = 1,
    inplace = True)

"""
**Data cleaning and processing**"""

#
    ##############################################################################################

#Look for missing values
print('There are {} missing values in the dataset.'.format(np.sum(D.isnull().
    values.any()))) ##Check if there are any missing values
#Dataset with no zero amount transactions
creditdata=D.loc[D['Amount']!=0.00]
fraud_cases=D.loc[(D['Class']==1) & (D['Amount']!=0)]
non_fraud_cases=D.loc[(D['Class']==0) & (D['Amount']!=0)]
#
    ##############################################################################################

#Look for zero amounts in the non-fraudulent dataset
zero_amount_fraudulent_case=fraud_cases.loc[fraud_cases['Amount']==0.00]
#Look for zero amounts in the non-fraudulent dataset
zero_amount_nonfraudulent_case=non_fraud_cases.loc[non_fraud_cases['Amount'
    ]==0.00]
#
    ##############################################################################################

#Look for zero amounts
class1_zeroamounts=D.loc[(D['Class']==1) & (D['Amount']==0)]
class0_zeroamounts=D.loc[(D['Class']==0) & (D['Amount']==0)]
#
    ##############################################################################################

print('There are {} transcations in the fraudulent data and {} transcations in
     the non-fraudulent data that have zero amounts, the whole dataset has {}
    transcations with zero amounts'.format(np.sum(class1_zeroamounts['Amount'
    ]==0.00), np.sum(class0_zeroamounts['Amount']==0.00),np.sum(D['Amount'
```

```
    ]==0.00)))
#Divide dataset into fraudulent and non-fraudulent cases with zero amounts
    removed
fraud_cases=creditdata.loc[creditdata['Class']==1]
non_fraud_cases=creditdata.loc[creditdata['Class']==0]
non_fraud_cases.shape

print("There are {} duplicates in this dataset".format(creditdata.duplicated(
    keep="first").sum()))

############Check for duplicates
print('There are {} duplicated fraudulent transactions and {} duplicated non-
    fraudulent transactions.'.format((creditdata.loc[creditdata['Class']==1]).
    duplicated(keep='first').sum(),(creditdata.loc[creditdata['Class']==0]).
    duplicated(keep='first').sum()))
#### Remove all duplicates
creditdata.drop_duplicates(keep="first", inplace=True)

print(creditdata.shape)
print("There are {} transactions with zero amount".format(np.sum(creditdata['
    Amount']==0.00)))
#########Check if duplicates have been removed
print("There are {} duplicates".format(np.sum(creditdata.duplicated(keep="
    first"))))
print("There are {} fraudulent and {} non-fraudulent transactions".format(np.
    sum(creditdata['Class']==1),np.sum(creditdata['Class']==0)))

#Most frequent amount
print(statistics.mode(fraud_cases['Amount']))
print(statistics.mode(non_fraud_cases['Amount']))
print(np.sum(creditdata['Amount']==1.00))
print(max(fraud_cases['Amount']))

all_bank_notes=creditdata.loc[creditdata['Amount']%10==0]
fraudulent_transaction_banknote=fraud_cases.loc[fraud_cases['Amount']%10==0] #
    fraud transaction bank notes
nonfraudulent_transaction_banknotes=non_fraud_cases[non_fraud_cases['Amount'
    ]%10==0] #Normal transcation bank notes
print("There are",np.sum(fraud_cases['Amount']%10==0),"bank notes in
    fraudulent transcations.")
print("There are",np.sum(non_fraud_cases['Amount']%10==0),"bank notes in
    normal transcations.")
print("There are",np.sum(creditdata['Amount']%10==0),"bank notes in the whole
    dataset")

#
    ##########################################################################################
```

```
#Remove bank notes from the fraud dataset i.e R10,R20,R30,...
cp_fraud=fraud_cases.loc[fraud_cases['Amount']%10==0] #Card not Present fraud
cnp_fraud=fraud_cases #Card present fraud
print(cp_fraud.shape)
print(cnp_fraud.shape)

less2500=np.sum(creditdata['Amount']<2500)
more2500=np.sum(creditdata['Amount']>=2500)
print("{}% of the transactions have transaction amount less than 2500 and {}%
    have transaction amount more than 2500".format(round(less2500/len(
    creditdata)*100,3),round(more2500/len(creditdata)*100,3)))

print(less2500)

"""**Graphical representation of the dataset**"""

#Plot histogram for Amount variable
g=sns.FacetGrid(creditdata,hue="Class", size=10,legend_out=False)
g.map(sns.distplot, "Amount")
g.add_legend()
plt.gcf().set_size_inches(6,6)
plt.ylim(0,0.004)
plt.title("Distribution of the transaction amounts \n in Euros.")
new_labels = ['Non-fradulent cases', 'Fraudulent cases']
for t, l in zip(g._legend.texts, new_labels): t.set_text(l)
plt.ylabel("Frequency (%)")
plt.show()
#Summary
#D['Amount'].describe()

sns.set(rc={"figure.figsize": (16, 8)})
fig, axes = plt.subplots(1,2)

sns.distplot(fraud_cases['Amount'],kde="true",ax=axes[0],color="#FC6A03")
sns.distplot(non_fraud_cases["Amount"],kde="True",ax=axes[1])
#Truncate the x-axis ?
axes[0].set_xlim(0,1000)
axes[1].set_xlim(0,2000)
###############################################################################
axes[0].set_title("Amount distribution for fraudulent transactions\n with
    truncated axis")
axes[1].set_title("Amount distribution for non-fraudulent transactions\n with
    truncated axis")
axes[0].grid(b=None)
axes[1].grid(b=None)
plt.show()

#Plot histogram for Time variable
```

```
g=sns.FacetGrid(creditdata, hue="Class", size=10,legend_out=False)
g.map(sns.distplot, "Time")
g.add_legend()
plt.ylabel("Density")
plt.title("Distribution of the time of the transactions \n from the first
    observation.")
plt.gcf().set_size_inches(6, 6)
new_labels = ['Non-fradulent cases', 'Fraudulent cases']
for t, l in zip(g._legend.texts, new_labels): t.set_text(l)
plt.show()
#Summary
D['Time'].describe()

sns.set(rc={"figure.figsize": (16, 8)})
fig, axes = plt.subplots(1,2)
sns.kdeplot(fraud_cases['Time'],color="Orange",shade=True,ax=axes[0],cmap="
    Oranges")
sns.kdeplot(non_fraud_cases["Time"],color="blue",shade=True,ax=axes[0],cmap="
    Blues")
sns.kdeplot(fraud_cases['Amount'],color="Orange",shade=True,ax=axes[1],cmap="
    Oranges")
sns.kdeplot(non_fraud_cases["Amount"],color="blue",shade=True,ax=axes[1],cmap=
    "Blues")
axes[0].set_title("Density plot for Time attribute")
axes[1].set_title("Density plot for Amount attribute")
handles = [mpatches.Patch(facecolor=plt.cm.Oranges(100), label="Fraudulent
    cases"),
           mpatches.Patch(facecolor=plt.cm.Blues(100), label="Non-fraudulent
               cases")]
axes[0].legend(handles=handles)
axes[1].legend(handles=handles)
axes[0].grid(b=None)
plt.grid(b=None)
plt.show()

sns.set(rc={"figure.figsize": (16, 8)})
fig, axes = plt.subplots(1,2)
sns.distplot(fraud_cases['Time'],color="orange",kde=True,ax=axes[0])
sns.distplot(non_fraud_cases["Time"],color="blue",kde=True,ax=axes[1])
axes[0].set_title("Time of the fraudulent transactions relative \n to the
    first transaction.")
axes[1].set_title("Time of the non-fraudulent transactions relative \n to the
    first transaction.")
axes[0].grid(b=None)
axes[1].grid(b=None)
plt.show()

#Ploting barplot for target
```

```
plt.figure(figsize=(7,7))
g = sns.barplot(D['Class'], D['Class'], palette='Set1', estimator=lambda x:
    len(x) / len(D) )
#Anotating the graph
for p in g.patches:
        width, height = p.get_width(), p.get_height()
        x, y = p.get_xy()
        g.text(x+width/2,
               y+height,
               '{:.4%}'.format(height),
               horizontalalignment='center',fontsize=15)

#Setting the labels
#plt.xlabel('Class distribution', fontsize=14)
plt.ylabel('Density', fontsize=14)
plt.title('Percentage of fraudulent and non-fraudulent \n transactions.',
    fontsize=16)
labels=["non-fraudulent", "fraudulent"]
plt.xticks(range(2),labels)
plt.show()

sns.set_style("whitegrid")
g=sns.FacetGrid(creditdata, hue="Class", size = 7,legend_out=False)
g.map(plt.scatter, "Time", "Amount")
plt.title("Scatterplot of time against amount")
g.add_legend()
new_labels = ['Non-fradulent cases', 'Fraudulent cases']
for t, l in zip(g._legend.texts, new_labels): t.set_text(l)
plt.gcf().set_size_inches(7,7)
plt.grid(b=None)
plt.show()

sns.set_style("whitegrid")
g=sns.FacetGrid(creditdata, hue="Class", size = 6)
g.map(plt.scatter, "Amount", "Time")
plt.title("Amount vs Time scatter plot")
g.add_legend()
plt.gcf().set_size_inches(7,7)
plt.show()

sns.set(rc={"figure.figsize": (16, 8)})
fig, ax = plt.subplots(1,2)
g=sns.boxplot(x = "Class", y = "Time", data = D,ax=ax[0])
sns.boxplot(x = "Class", y = "Amount", data = D,ax=ax[1])#,order=['Non-
    fradulent \n cases','Fraudulent \n cases']
new_labels = ['Non-fradulent cases', 'Fraudulent cases']
ax[0].set_title("Time Boxplot")
ax[1].set_title("Amount Boxplot")
```

```
ax[0].grid(b=None)
ax[1].grid(b=None)
plt.show()

def data_array(creditdata: pd.DataFrame) -> (np.ndarray, np.ndarray):
    X = creditdata.iloc[:, 0:30].values
    y = creditdata.Class.values
    return X, y

def plot(X: np.ndarray, y: np.ndarray):
    plt.figure(figsize=(7,7))
    plt.scatter(X[y == 0, 0], X[y == 0, 1], label="Non-fraudulent class",
        alpha=0.5, linewidth=0.15)
    plt.scatter(X[y == 1, 0], X[y == 1, 1], label="Fraudulent class", alpha
        =0.5, linewidth=0.15, c='r')
    plt.title("Credit card dataset")
    plt.legend()
    return plt.show()

X,y=data_array(creditdata_normalized)
plot(X,y)

#Are the means significantly different?
print(ttest_ind(fraud_cases['Time'],non_fraud_cases['Time'],equal_var=False))
##Since p-value is less than alpha, we reject H_o and conclude that there is a
    significant difference between the means.
stats.levene(fraud_cases['Time'],non_fraud_cases['Time'],center="mean")
#Is there significant difference in the population?
##Since the p-value is greater than alpha, we fail to reject H_o and conclude
   that there is insufficient evidence to say that there is a difference
   between the variances.

#Are the means significantly different?
print(ttest_ind(fraud_cases['Amount'],non_fraud_cases['Amount'],equal_var=
   False))
##Since p-value is less than alpha, we reject H_o and conclude that there is a
    significant difference between the means.
stats.levene(fraud_cases['Amount'],non_fraud_cases['Amount'],center="mean")
#Is there significantl difference in the population?
##Since the p-value is less than alpha, we reject H_o clude that there is a
   significant difference between the population variance.

#Perform one-way Anova
#
   ##############################################################################

model=ols('Class~Time+Amount+Time*Amount',data=creditdata).fit()
ano_table=sm.stats.anova_lm(model,typ=2)
```

```
print(ano_table)
#
    ##########################################################################################

print("The interaction effect is not significant")

#Bank notes, R10,R20,R30,.....
sns.set_style("whitegrid")
sns.lmplot(x="Time",y="Amount",hue="Class",data=all_bank_notes,order = 2)
plt.xlabel("Time")
plt.ylabel("Amount")
plt.gcf().set_size_inches(7,7)
plt.title("Fraudulent transcations")
plt.show()

sns.set(rc={"figure.figsize": (16, 8)})
fig, axes = plt.subplots(1,2)

sns.set_style("whitegrid")
sns.regplot(data=cp_fraud,x="Time",y="Amount",ax=axes[0],order = 2)
sns.regplot(data=cnp_fraud,x="Time",y="Amount",order = 5)
axes[0].set_xlim(0,175000)
axes[1].set_xlim(0,175000)
axes[0].set_title("Card present fraud (CP)")
axes[1].set_title("Card not present fraud (CNP)")
plt.show()

#The money stolen between 0 and 50000 seconds
sum=0
for i in range(len(creditdata)):
  if((creditdata.iloc[i]['Time']>=0 and creditdata.iloc[i]["Time"]<=50000) and
      creditdata.iloc[i]['Class']==1):
    sum=sum + creditdata.iloc[i]['Amount']
print("The money stolen between 0 and 50000 seconds is R",round(sum,2))
#
    ##########################################################################################

#The money stolen between 50000 and 100000 seconds
sum=0
for i in range(len(creditdata)):
  if((creditdata.iloc[i]['Time']>50000 and creditdata.iloc[i]["Time"]<=100000)
      and creditdata.iloc[i]['Class']==1):
    sum=sum + creditdata.iloc[i]['Amount']
print("The money stolen between 50000 and 100000 seconds is ",round(sum,2))
#
    ##########################################################################################

#The money stolen between 50000 and 100000 seconds
```

```
sum=0
for i in range(len(creditdata)):
  if((creditdata.iloc[i]['Time']>100000 and creditdata.iloc[i]["Time"
     ]<=150000) and creditdata.iloc[i]['Class']==1):
    sum=sum + creditdata.iloc[i]['Amount']
print("The money stolen between 100000 and 150000 seconds is ",round(sum,2))
#
   ##############################################################################

#The money stolen above 150000 seconds
sum=0
for i in range(len(creditdata)):
  if(creditdata.iloc[i]['Time']>150000 and creditdata.iloc[i]['Class']==1):
    sum=sum + creditdata.iloc[i]['Amount']
print("The money stolen above 150000 seconds is R",round(sum,2))
#
   ##############################################################################

total_amount=np.sum(D['Amount'])
stolen_amount_percentage=(np.sum(fraud_cases['Amount'])/total_amount)*100
print("The stolen amount percentage is",round(stolen_amount_percentage,2),'%')
print("Total stolen amount",np.sum(fraud_cases['Amount']))

data = {'0-50000': 15537.42, '50000 - 100000': 22671.17, '100000 - 150000':
    15442.25}
time_interval = list(data.keys())
amount = list(data.values())

fig, axs = plt.subplots(figsize=(7,7))
plt.title("Categorizing stolen amount within time intervals")
plt.xlabel("Time in seconds")
plt.ylabel("Amount stolen")
plt.grid(b=None)
plt.bar(time_interval, amount,align='center')

"""**Normalizing Dataset**"""

from sklearn.preprocessing import StandardScaler
creditdata_1=creditdata
creditdata_1['Amount(Normalized)'] = StandardScaler().fit_transform(
    creditdata_1['Amount'].values.reshape(-1,1))
creditdata_1['Time(Normalized)'] = StandardScaler().fit_transform(creditdata_1
    ['Time'].values.reshape(-1,1))
creditdata_normalized = creditdata_1.drop(columns = ['Amount', 'Time'], axis
    =1)
creditdata=creditdata.drop(['Amount(Normalized)','Time(Normalized)'],axis=1)
creditdata.head()
```

```
X=creditdata_normalized.drop('Class',axis=1)
y=creditdata_normalized['Class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
    random_state = 500, stratify=y)

"""**Support Vector Machine**"""

#Import svm model
from sklearn import svm
#Create a svm Classifier
svm_clf = svm.SVC()
#Train the model using the training sets
svm_clf .fit(X_train, y_train)

y_pred=svm_clf.predict(X_test)
print(classification_report(y_test,y_pred))

y_pred_tr=svm_clf.predict(X_train)
print(classification_report(y_train,y_pred_tr))

from sklearn.model_selection import RandomizedSearchCV
# defining parameter range
param_grid = {'C':[ 0.8,0.85,0.9,0.95],'kernel':['linear','rbf','poly','
    sigmoid']}
rs_svm = RandomizedSearchCV(svm.SVC(max_iter=200, tol=2,probability=True),
    param_grid, scoring='roc_auc', verbose = True, n_jobs=-1, cv=5)
rs_svm.fit(X_train, y_train)

print(rs_svm.best_params_)

from sklearn import svm
svm_hyp=svm.SVC(C=0.8,kernel='rbf',max_iter=200, tol=2,probability=True).fit(
    X_train,y_train)
#Predict the response for test dataset
y_pred = svm_hyp.predict_proba(X_test)
y_hat=svm_hyp.predict(X_test)
print(classification_report(y_test,y_hat))

#Predict the response for train dataset
y_pred_tr = svm_hyp.predict(X_train)
print(classification_report(y_train,y_pred_tr))

average_precision = average_precision_score(y_test, y_pred[:,1])
svm_pr_curve = plot_precision_recall_curve(svm_hyp, X_test, y_test)
svm_pr_curve.ax_.set_title('Precision-Recall curve: '
                   'Avearge precision={0:0.2f}'.format(average_precision))

from sklearn.metrics import roc_curve, roc_auc_score
```

```
prob=svm_hyp.predict_proba(X_test)[:,1]
false_positive_rate, true_positive_rate, threshold = roc_curve(y_test, prob)
fig, ax = plt.subplots(figsize=(8,5))
plt.title('ROC Curve for SVM')
plt.plot(false_positive_rate, true_positive_rate)
plt.plot([0, 1], ls="--")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()

y_hat=svm_hyp.predict(X_test)
auc=roc_auc_score(y_test,y_hat)
prec = precision_score(y_test, y_hat)
rec = recall_score(y_test, y_hat)
f1 = f1_score(y_test,y_hat)
gmean=geometric_mean_score(y_test,y_hat)
mcc=matthews_corrcoef(y_test,y_hat)
print("\tprecision:%0.4f"%prec,"\trecall:%0.4f"%rec,"\tF1-score:%0.4f"%f1,"\
    tgeometric mean:%0.4f"%gmean,"\troc auc:%0.4f"%auc,"\tMCC:%0.4f"%mcc)

"""**SVM Bootstrapp estimate**"""

from mlxtend.evaluate import bootstrap_point632_score


def bootstrap_estimate_and_ci(estimator, X, y, scoring_func=None, random_seed
    =260,
                                  method='.632', alpha=0.05, n_splits=10):
     scores = bootstrap_point632_score(estimator, X, y, scoring_func=
         scoring_func,
                                       n_splits=n_splits, random_seed=
                                           random_seed,
                                       method=method)
     estimate = np.mean(scores)
     lower_bound = np.percentile(scores, 100*(alpha/2))
     upper_bound = np.percentile(scores, 100*(1-alpha/2))
     stderr = np.std(scores)

     return estimate, lower_bound, upper_bound, stderr

def data_array(creditdata: pd.DataFrame) -> (np.ndarray, np.ndarray):
     X=creditdata.drop('Class',axis=1)
     y = creditdata['Class']
     X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,stratify=
         y)
     return X_train.values,X_test.values,y_train.values,y_test.values

xtrain,xtest,ytrain,ytest=data_array(creditdata_normalized)
```

```
# Calculate a bootstrap estimate for recall and a 95% confidence interval
estimator=svm.SVC(C=0.9,kernel='rbf',max_iter=200, tol=2,probability=True)
est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=precision_score)
print(f"precision estimate on test dataset: {est:.4f}, confidence interval: [{
    low:.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=recall_score)
print(f"recall estimate on test dataset: {est:.4f}, confidence interval: [{low
    :.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=f1_score)
print(f"f1 estimate on test dataset: {est:.4f}, confidence interval: [{low:.4f
    }, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=geometric_mean_score)
print(f"geometric mean estimate on test dataset: {est:.4f}, confidence
    interval: [{low:.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=roc_auc_score)
print(f"AUC estimate on test dataset: {est:.4f}, confidence interval: [{low:.4
    f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=matthews_corrcoef)
print(f"MCC estimate on test dataset: {est:.4f}, confidence interval: [{low:.4
    f}, {up:.4f}], " f"standard error: {stderr:.4f}")

"""**One-Sided Sampling for SVM**"""

from imblearn.under_sampling import OneSidedSelection
oss = OneSidedSelection(n_neighbors=1, random_state=200,sampling_strategy="
    majority")
X_oss, y_oss= oss.fit_resample(X, y)

y_oss.value_counts()

y.value_counts()

X_train, X_test, y_train, y_test=train_test_split(X_oss,y_oss,test_size=0.3,
    stratify=y_oss, random_state=42)
xtest=X_test.values
ytest=y_test.values
```

```
from sklearn import svm
svm_und=svm.SVC(probability=True).fit(X_train,y_train)

yhat=svm_und.predict(X_train)
print(classification_report(y_train,yhat))

yhat=svm_und.predict(X_test)
print(classification_report(y_test,yhat))

"""**Random search for One-Sided Sampling**"""

from sklearn.model_selection import RandomizedSearchCV
# defining parameter range
max_iter=[25,50,75,100]
param_grid = {'C':[0.1,0.4,0.7,0.95],'kernel':['linear','rbf','poly','sigmoid'
    ],'max_iter':max_iter}
oss_svm = RandomizedSearchCV(svm.SVC(tol=2,probability=True),param_grid,
    scoring='roc_auc', verbose = True, n_jobs=-1, cv=5)
oss_svm.fit(X_train, y_train)

print(oss_svm.best_params_)

from sklearn import svm
svm_hyp=svm.SVC(C=0.95,kernel='rbf',max_iter=100, tol=2,probability=True).fit(
    X_train,y_train)
#Predict the response for test dataset
y_pred = svm_hyp.predict_proba(X_test)
y_hat=svm_hyp.predict(X_test)
print(classification_report(y_test,y_hat))

yhat=svm_hyp.predict(X_train)
print(classification_report(y_train,yhat))

average_precision = average_precision_score(y_test, y_pred[:,1])
svm_pr_curve = plot_precision_recall_curve(svm_hyp, X_test, y_test)
svm_pr_curve.ax_.set_title('Precision-Recall curve: '
                   'Avearge precision={0:0.2f}'.format(average_precision))

from sklearn.metrics import roc_curve, roc_auc_score
prob=svm_hyp.predict_proba(X_test)[:,1]
false_positive_rate, true_positive_rate, threshold = roc_curve(y_test, prob)
fig, ax = plt.subplots(figsize=(8,5))
plt.title('ROC Curve for SVM on balanced dataset.')
plt.plot(false_positive_rate, true_positive_rate)
plt.plot([0, 1], ls="--")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
```

```
plt.show()

y_hat=svm_hyp.predict(X_test)
auc=roc_auc_score(y_test,y_hat)
prec = precision_score(y_test, y_hat)
rec = recall_score(y_test, y_hat)
f1 = f1_score(y_test,y_hat)
gmean=geometric_mean_score(y_test,y_hat)
mcc=matthews_corrcoef(y_test,y_hat)
print("\tprecision:%0.4f"%prec,"\trecall:%0.4f"%rec,"\tF1-score:%0.4f"%f1,"\
    tgeometric mean:%0.4f"%gmean,"\troc auc:%0.4f"%auc,"\tMCC:%0.4f"%mcc)

estimator=svm.SVC(C=0.95,kernel='rbf',max_iter=100, tol=2,probability=True).
    fit(X_train,y_train)
est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=precision_score)
print(f"precision estimate on test dataset: {est:.4f}, confidence interval: [{
    low:.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=recall_score)
print(f"recall estimate on test dataset: {est:.4f}, confidence interval: [{low
    :.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=f1_score)
print(f"f1 estimate on test dataset: {est:.4f}, confidence interval: [{low:.4f
    }, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=geometric_mean_score)
print(f"geometric mean estimate on test dataset: {est:.4f}, confidence
    interval: [{low:.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=roc_auc_score)
print(f"AUC estimate on test dataset: {est:.4f}, confidence interval: [{low:.4
    f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=matthews_corrcoef)
print(f"MCC estimate on test dataset: {est:.4f}, confidence interval: [{low:.4
    f}, {up:.4f}], " f"standard error: {stderr:.4f}")

"""**Random undersampling**

##########################################################################################
```

```
"""

from imblearn.under_sampling import RandomUnderSampler
rus = RandomUnderSampler(random_state=42)
X_rus, y_rus = rus.fit_resample(X, y)

y_rus.value_counts()

svm_rus=svm.SVC().fit(X_train,y_train)
y_pred=svm_rus.predict(X_train)
print(classification_report(y_train,y_pred))

y_pred=svm_rus.predict(X_test)
print(classification_report(y_test,y_pred))

from sklearn import svm
#K-Fold cross validation
kf = StratifiedKFold(n_splits=10,shuffle=True,random_state=42)
auc_score =[]
recall_scor=[]
precision_scor=[]
geo_score=[]
f1=[]
mcc=[]
i=1
for train_index,test_index in kf.split(X_rus,y_rus):
    #print('{} of KFold {}'.format(i,kf.n_splits))
    xtrain,xtest = X_rus.iloc[train_index],X_rus.iloc[test_index]
    ytrain,ytest = y_rus.iloc[train_index],y_rus.iloc[test_index]

    #model
    svm_rus=svm.SVC().fit(xtrain,ytrain)
    score = roc_auc_score(ytest,svm_rus.predict(xtest))
    score1 = recall_score(ytest,svm_rus.predict(xtest))
    score2 = precision_score(ytest,svm_rus.predict(xtest))
    score3=geometric_mean_score(ytest,svm_rus.predict(xtest))
    score4=f1_score(ytest,svm_rus.predict(xtest))
    score5=matthews_corrcoef(ytest,svm_rus.predict(xtest))
    #
        ##########################################################################################

    auc_score.append(score)
    recall_scor.append(score1)
    precision_scor.append(score2)
    geo_score.append(score3)
    f1.append(score4)
    mcc.append(score5)
```

```
    i+=1

print("\tprecision:%0.4f"%mean(precision_scor),"\trecall:%0.4f"%mean(
    recall_scor),"\tF1-score:%0.4f"%mean(f1),"\tgeometric mean:%0.4f"%mean(
    geo_score),"\troc auc:%0.4f"%mean(auc_score),"\tMCC:%0.4f"%mean(mcc))

y_hat=svm_rus.predict(X_test)
auc=roc_auc_score(y_test,y_hat)
prec = precision_score(y_test, y_hat)
rec = recall_score(y_test, y_hat)
f1 = f1_score(y_test,y_hat)
gmean=geometric_mean_score(y_test,y_hat)
mcc=matthews_corrcoef(y_test,y_hat)
print("\tprecision:%0.4f"%prec,"\trecall:%0.4f"%rec,"\tF1-score:%0.4f"%f1,"\
    tgeometric mean:%0.4f"%gmean,"\troc auc:%0.4f"%auc,"\tMCC:%0.4f"%mcc)

"""**SMOTE with TomekLink**"""

from imblearn.combine import SMOTETomek
from imblearn.under_sampling import TomekLinks
smt = SMOTETomek(tomek=TomekLinks(sampling_strategy='majority'),random_state
    =42)
X_res, y_res = smt.fit_resample(X, y)

y_res.value_counts()

X_train, X_test,y_train,y_test=train_test_split(X_res,y_res,test_size=0.3,
    stratify=y_res)

from sklearn import svm
svm_smote=svm.SVC().fit(X_train,y_train)
y_pred=svm_smote.predict(X_train)
print(classification_report(y_train,y_pred))

y_pred=svm_smote.predict(X_test)
print(classification_report(y_test,y_pred))

y_hat=svm_smote.predict(X_test)
auc=roc_auc_score(y_test,y_hat)
prec = precision_score(y_test, y_hat)
rec = recall_score(y_test, y_hat)
f1 = f1_score(y_test,y_hat)
gmean=geometric_mean_score(y_test,y_hat)
mcc=matthews_corrcoef(y_test,y_hat)
print("\tprecision:%0.4f"%prec,"\trecall:%0.4f"%rec,"\tF1-score:%0.4f"%f1,"\
    tgeometric mean:%0.4f"%gmean,"\troc auc:%0.4f"%auc,"\tMCC:%0.4f"%mcc)

"""**k-Nearest Neighbour**"""
```

```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier().fit(X_train,y_train)
y_pred=knn.predict(X_test)

print(classification_report(y_test,y_pred))

y_tr=knn.predict(X_train)
print(classification_report(y_train,y_tr))

#Knn rs search
from sklearn.model_selection import RandomizedSearchCV
n_neighbors=[i for i in range(2,7)]
p=[1,2]
param_grid={'n_neighbors':n_neighbors,'p':p}
rs_knn=RandomizedSearchCV(KNeighborsClassifier(),param_grid,cv=5,n_jobs=-1,
    verbose=True, scoring='roc_auc')
best_knn=rs_knn.fit(X_train,y_train)

print(best_knn.best_params_)

from sklearn.neighbors import KNeighborsClassifier
knn_hyp = KNeighborsClassifier(n_neighbors=5,p=1).fit(X_train,y_train)
predict=knn_hyp.predict(X_test)
print(classification_report(y_test,predict))

predict_train=knn_hyp.predict(X_train)
print(classification_report(y_train,predict_train))

Knn_hat=knn_hyp.predict_proba(X_test)[:,1]
average_precision = average_precision_score(y_test, Knn_hat)
knn_pr_curve = plot_precision_recall_curve(knn_hyp, X_test, y_test)
knn_pr_curve.ax_.set_title('Precision-Recall curve: '
                   'Avearge precision={0:0.2f}'.format(average_precision))

from sklearn.metrics import roc_curve, roc_auc_score
false_positive_rate, true_positive_rate, threshold = roc_curve(y_test, Knn_hat
    )
fig, ax = plt.subplots(figsize=(8,5))
plt.title('ROC Curve for KNN')
plt.plot(false_positive_rate, true_positive_rate)
plt.plot([0, 1], ls="--")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()

#Test dataset
auc=roc_auc_score(y_test,predict)
```

```
prec = precision_score(y_test, predict)
rec = recall_score(y_test, predict)
f1 = f1_score(y_test, predict)
gmean=geometric_mean_score(y_test,predict)
mcc=matthews_corrcoef(y_test,predict)
print("\tprecision:%0.4f"%prec,"\trecall:%0.4f"%rec,"\tF1-score:%0.4f"%f1,"\
    tgeometric mean:%0.4f"%gmean,"\troc auc:%0.4f"%auc,"\tMCC:%0.4f"%mcc)

"""**kNN bootstrap point Estimate**"""

# Calculate a bootstrap estimate for recall and a 95% confidence interval
estimator=KNeighborsClassifier(n_neighbors = 5,p=1)
est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=precision_score)
print(f"precision estimate on test dataset: {est:.4f}, confidence interval: [{
    low:.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=recall_score)
print(f"recall estimate on test dataset: {est:.4f}, confidence interval: [{low
    :.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=f1_score)
print(f"f1 estimate on test dataset: {est:.4f}, confidence interval: [{low:.4f
    }, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=geometric_mean_score)
print(f"geometric mean estimate on test dataset: {est:.4f}, confidence
    interval: [{low:.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=roc_auc_score)
print(f"roc_auc estimate on test dataset: {est:.4f}, confidence interval: [{
    low:.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=matthews_corrcoef)
print(f"MCC estimate on test dataset: {est:.4f}, confidence interval: [{low:.4
    f}, {up:.4f}], " f"standard error: {stderr:.4f}")

"""**kNN One Sided Sampling**"""

X_train, X_test, y_train, y_test=train_test_split(X_oss,y_oss,test_size=0.3,
    stratify=y_oss, random_state=42)

from sklearn.neighbors import KNeighborsClassifier
```

```
knn_und=KNeighborsClassifier().fit(X_train,y_train)
y_pred=knn_und.predict(X_train)
print(classification_report(y_train,y_pred))

y_pred=knn_und.predict(X_test)
print(classification_report(y_test,y_pred))

#Knn rs search
from sklearn.model_selection import RandomizedSearchCV
n_neighbors=[i for i in range(2,7)]
p=[1,2]
param_grid={'n_neighbors':n_neighbors,'p':p}
oss_knn=RandomizedSearchCV(KNeighborsClassifier(),param_grid,cv=5,n_jobs=-1,
    verbose=True, scoring='roc_auc')
best_knn=oss_knn.fit(X_train,y_train)

print(best_knn.best_params_)

from sklearn.neighbors import KNeighborsClassifier
knn_hyp = KNeighborsClassifier(n_neighbors=5,p=2).fit(X_train,y_train)
predict=knn_hyp.predict(X_test)
print(classification_report(y_test,predict))

predict_train=knn_hyp.predict(X_train)
print(classification_report(y_train,predict_train))

Knn_hat=knn_hyp.predict_proba(X_test)[:,1]
average_precision = average_precision_score(y_test, Knn_hat)
knn_pr_curve = plot_precision_recall_curve(knn_hyp, X_test, y_test)
knn_pr_curve.ax_.set_title('Precision-Recall curve: '
                   'Avearge precision={0:0.2f}'.format(average_precision))

from sklearn.metrics import roc_curve, roc_auc_score
false_positive_rate, true_positive_rate, threshold = roc_curve(y_test, Knn_hat
    )
fig, ax = plt.subplots(figsize=(8,5))
plt.title('ROC Curve for KNN')
plt.plot(false_positive_rate, true_positive_rate)
plt.plot([0, 1], ls="--")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()

#Test dataset
auc=roc_auc_score(y_test,predict)
prec = precision_score(y_test, predict)
rec = recall_score(y_test, predict)
f1 = f1_score(y_test, predict)
```

```
gmean=geometric_mean_score(y_test,predict)
mcc=matthews_corrcoef(y_test,predict)
print("\tprecision:%0.4f"%prec,"\trecall:%0.4f"%rec,"\tF1-score:%0.4f"%f1,"\
    tgeometric mean:%0.4f"%gmean,"\troc auc:%0.4f"%auc,"\tMCC:%0.4f"%mcc)

"""**KNN Bootstrap Point Estimate For One-Sided Sampled Dataset**"""

estimator=KNeighborsClassifier(n_neighbors = 5,p=1)
est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=precision_score)
print(f"precision estimate on test dataset: {est:.4f}, confidence interval: [{
    low:.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=recall_score)
print(f"recall estimate on test dataset: {est:.4f}, confidence interval: [{low
    :.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=f1_score)
print(f"f1 estimate on test dataset: {est:.4f}, confidence interval: [{low:.4f
    }, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=geometric_mean_score)
print(f"geometric mean estimate on test dataset: {est:.4f}, confidence
    interval: [{low:.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=roc_auc_score)
print(f"roc_auc estimate on test dataset: {est:.4f}, confidence interval: [{
    low:.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=matthews_corrcoef)
print(f"MCC estimate on test dataset: {est:.4f}, confidence interval: [{low:.4
    f}, {up:.4f}], " f"standard error: {stderr:.4f}")

"""
    ##########################################################################################


**Random Undersampling**
"""

knn_rus=KNeighborsClassifier().fit(X_train,y_train)
y_pred=knn_rus.predict(X_train)
print(classification_report(y_train,y_pred))
```

```
y_pred=knn_rus.predict(X_test)
print(classification_report(y_test,y_pred))

from sklearn.neighbors import KNeighborsClassifier
#K-Fold cross validation
kf = StratifiedKFold(n_splits=10,shuffle=True,random_state=42)
auc_score =[]
recall_scor=[]
precision_scor=[]
geo_score=[]
f1=[]
mcc=[]
i=1
for train_index,test_index in kf.split(X_rus,y_rus):
    #print('{} of KFold {}'.format(i,kf.n_splits))
    xtrain,xtest = X_rus.iloc[train_index],X_rus.iloc[test_index]
    ytrain,ytest = y_rus.iloc[train_index],y_rus.iloc[test_index]

    #model
    knn_rus=KNeighborsClassifier().fit(xtrain,ytrain)
    score = roc_auc_score(ytest,knn_rus.predict(xtest))
    score1 = recall_score(ytest,knn_rus.predict(xtest))
    score2 = precision_score(ytest,knn_rus.predict(xtest))
    score3=geometric_mean_score(ytest,knn_rus.predict(xtest))
    score4=f1_score(ytest,knn_rus.predict(xtest))
    score5=matthews_corrcoef(ytest,knn_rus.predict(xtest))
    #
        ##########################################################################

    auc_score.append(score)
    recall_scor.append(score1)
    precision_scor.append(score2)
    geo_score.append(score3)
    f1.append(score4)
    mcc.append(score5)

    i+=1

print("\tprecision:%0.4f"%mean(precision_scor),"\trecall:%0.4f"%mean(
    recall_scor),"\tF1-score:%0.4f"%mean(f1),"\tgeometric mean:%0.4f"%mean(
    geo_score),"\troc auc:%0.4f"%mean(auc_score),"\tMCC:%0.4f"%mean(mcc))

"""**SMOTE with TomekLink**"""

from sklearn.neighbors import KNeighborsClassifier
knn_smote=KNeighborsClassifier().fit(X_train,y_train)
y_pred=knn_smote.predict(X_train)
```

```
print(classification_report(y_train,y_pred))

y_pred=knn_smote.predict(X_test)
print(classification_report(y_test,y_pred))

auc=roc_auc_score(y_test,y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test,y_pred)
gmean=geometric_mean_score(y_test,y_pred)
mcc=matthews_corrcoef(y_test,y_pred)
print("\tprecision:%0.4f"%prec,"\trecall:%0.4f"%rec,"\tF1-score:%0.4f"%f1,"\
    tgeometric mean:%0.4f"%gmean,"\troc auc:%0.4f"%auc,"\tMCC:%0.4f"%mcc)

"""**Logistic Regression**"""

from sklearn.linear_model import LogisticRegression
#Basic Logistic regression on raw dataset
lr = LogisticRegression().fit(X_train, y_train)
y_pred=lr.predict(X_test)
yhat=lr.predict_proba(X_test)
print(classification_report(y_test,y_pred))

y_pred_train=lr.predict(X_train)
print(classification_report(y_train,y_pred_train))

#Logistic regression random search
from sklearn.model_selection import RandomizedSearchCV
param_grid={'C':[0.1,0.3,0.7,0.9],'solver':['lbfgs','newton-cg','sag','saga'],
     'penalty':['l2','l1','elasticnet','none'],'class_weight'
    :[{0:1,1:5},{0:1,1:10},{0:1,1:30}]}
rs_lr=RandomizedSearchCV(LogisticRegression(max_iter=1000),param_grid,cv=5,
    n_jobs=-1,verbose=True, scoring='roc_auc')
rs_lr.fit(X_train,y_train)

print(rs_lr.best_params_)

lr_hyp = LogisticRegression(C=0.1,solver='saga',penalty='l2',class_weight
    ={0:1,1:5},max_iter=1000).fit(X_train, y_train)
y_pred_tr=lr_hyp.predict(X_train)
print(classification_report(y_train,y_pred_tr))

y_pred=lr_hyp.predict(X_test)
print(classification_report(y_test,y_pred))

yhat=lr_hyp.predict_proba(X_test)
average_precision = average_precision_score(y_test, yhat[:,1])
lr_pr_curve = plot_precision_recall_curve(lr_hyp, X_test, y_test)
```

```
lr_pr_curve.ax_.set_title('Precision-Recall curve: '
                  'Avearge precision={0:0.2f}.'.format(average_precision))

from sklearn.metrics import roc_curve, roc_auc_score
yhat=lr_hyp.predict_proba(X_test)
false_positive_rate, true_positive_rate, threshold = roc_curve(y_test, yhat
    [:,1])
fig, ax = plt.subplots(figsize=(8,5))
plt.title('ROC Curve for LR.')
plt.plot(false_positive_rate, true_positive_rate)
plt.plot([0, 1], ls="--")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()

y_pred=lr_hyp.predict(X_test)
auc=roc_auc_score(y_test,y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
gmean=geometric_mean_score(y_test,y_pred)
mcc=matthews_corrcoef(y_test,y_pred)
print("\tprecision:%0.4f"%prec,"\trecall:%0.4f"%rec,"\tF1-score:%0.4f"%f1,"\
    tgeometric mean:%0.4f"%gmean,"\troc auc:%0.4f"%auc,"\tMCC:%0.4f"%mcc)

"""**Logistic Regression Bootstrap Estimate**"""

estimator=LogisticRegression(C=0.1,solver='saga',penalty='l2',class_weight
    ={0:1,1:5},max_iter=1000)
est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=precision_score)
print(f"precision estimate on test dataset: {est:.2f}, confidence interval: [{
    low:.2f}, {up:.2f}], " f"standard error: {stderr:.2f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=recall_score)
print(f"recall estimate on test dataset: {est:.4f}, confidence interval: [{low
    :.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=f1_score)
print(f"f1 estimate on test dataset: {est:.4f}, confidence interval: [{low:.4f
    }, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=geometric_mean_score)
print(f"Geomtric mean estimate on test dataset: {est:.4f}, confidence interval
    : [{low:.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")
```

```
est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=roc_auc_score)
print(f"AUC estimate on test dataset: {est:.4f}, confidence interval: [{low:.4
    f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=matthews_corrcoef)
print(f"MCC estimate on test dataset: {est:.4f}, confidence interval: [{low:.4
    f}, {up:.4f}], " f"standard error: {stderr:.4f}")

"""**Logistic Regression One-Sided Sampling**"""

from sklearn.linear_model import LogisticRegression
lr_und = LogisticRegression().fit(X_train,y_train)
y_pred=lr_und.predict(X_train)
print(classification_report(y_train,y_pred))

y_pred=lr_und.predict(X_test)
print(classification_report(y_test,y_pred))

#Logistic regression random search
from sklearn.model_selection import RandomizedSearchCV
param_grid={'C':[0.4,0.8,1.2,1.6,2],'solver':['lbfgs','newton-cg','sag','saga'
    ], 'penalty':['l2','l1','elasticnet','none'],'class_weight'
    :[{0:1,1:3},{0:1,1:5},{0:1,1:10},{0:1,1:15}]}
oss_lr=RandomizedSearchCV(LogisticRegression(max_iter=1000),param_grid,cv=5,
    n_jobs=-1,verbose=True, scoring='roc_auc')
oss_lr.fit(X_train,y_train)

print(oss_lr.best_params_)

from sklearn.linear_model import LogisticRegression
lr_hyp = LogisticRegression(C=0.4,solver='saga',penalty='l2',max_iter=1000,
    class_weight={0:1,1:5}).fit(X_train, y_train)
y_pred=lr_hyp.predict(X_test)
print(classification_report(y_test,y_pred))

y_pred_tr=lr_hyp.predict(X_train)
print(classification_report(y_train,y_pred_tr))

yhat=lr_hyp.predict_proba(X_test)
average_precision = average_precision_score(y_test, yhat[:,1])
lr_pr_curve = plot_precision_recall_curve(lr_hyp, X_test, y_test)
lr_pr_curve.ax_.set_title('Precision-Recall curve: '
                  'Avearge precision={0:0.2f}'.format(average_precision))

from sklearn.metrics import roc_curve, roc_auc_score
```

```
yhat=lr_hyp.predict_proba(X_test)
false_positive_rate, true_positive_rate, threshold = roc_curve(y_test, yhat
    [:,1])
fig, ax = plt.subplots(figsize=(8,5))
plt.title('ROC Curve for LR')
plt.plot(false_positive_rate, true_positive_rate)
plt.plot([0, 1], ls="--")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()

y_pred=lr_hyp.predict(X_test)
auc=roc_auc_score(y_test,y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
gmean=geometric_mean_score(y_test,y_pred)
mcc=matthews_corrcoef(y_test,y_pred)
print("\tprecision:%0.4f"%prec,"\trecall:%0.4f"%rec,"\tF1-score:%0.4f"%f1,"\
    tgeometric mean:%0.4f"%gmean,"\troc auc:%0.4f"%auc,"\tMCC:%0.4f"%mcc)

"""**Logistic Regression Bootstrap Estimate for One-Sided Sampled Dataset**"""

estimator=LogisticRegression(C=0.4,solver='saga',penalty='l2',max_iter=1000,
    class_weight={0:1,1:5})
est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=precision_score)
print(f"precision estimate on test dataset: {est:.2f}, confidence interval: [{
    low:.2f}, {up:.2f}], " f"standard error: {stderr:.2f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=recall_score)
print(f"recall estimate on test dataset: {est:.4f}, confidence interval: [{low
    :.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=f1_score)
print(f"f1 estimate on test dataset: {est:.4f}, confidence interval: [{low:.4f
    }, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=geometric_mean_score)
print(f"Geomtric mean estimate on test dataset: {est:.4f}, confidence interval
    : [{low:.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=roc_auc_score)
```

```
print(f"AUC estimate on test dataset: {est:.4f}, confidence interval: [{low:.4
    f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=matthews_corrcoef)
print(f"MCC estimate on test dataset: {est:.4f}, confidence interval: [{low:.4
    f}, {up:.4f}], " f"standard error: {stderr:.4f}")

"""**Random undersampling**"""

lr_rus=LogisticRegression().fit(X_train,y_train)
y_pred=lr_rus.predict(X_train)
print(classification_report(y_train,y_pred))

y_pred=lr_rus.predict(X_test)
print(classification_report(y_test,y_pred))

from sklearn.linear_model import LogisticRegression
#K-Fold cross validation
kf = StratifiedKFold(n_splits=10,shuffle=True,random_state=42)
auc_score =[]
recall_scor=[]
precision_scor=[]
geo_score=[]
f1=[]
mcc=[]
i=1
for train_index,test_index in kf.split(X_rus,y_rus):
    #print('{} of KFold {}'.format(i,kf.n_splits))
    xtrain,xtest = X_rus.iloc[train_index],X_rus.iloc[test_index]
    ytrain,ytest = y_rus.iloc[train_index],y_rus.iloc[test_index]

    #model
    lr_rus=LogisticRegression().fit(xtrain,ytrain)
    score = roc_auc_score(ytest,lr_rus.predict(xtest))
    score1 = recall_score(ytest,lr_rus.predict(xtest))
    score2 = precision_score(ytest,lr_rus.predict(xtest))
    score3=geometric_mean_score(ytest,lr_rus.predict(xtest))
    score4=f1_score(ytest,lr_rus.predict(xtest))
    score5=matthews_corrcoef(ytest,lr_rus.predict(xtest))
    #
        ##########################################################################################

    auc_score.append(score)
    recall_scor.append(score1)
    precision_scor.append(score2)
    geo_score.append(score3)
    f1.append(score4)
```

```
        mcc.append(score5)

        i+=1

print("\tprecision:%0.4f"%mean(precision_scor),"\trecall:%0.4f"%mean(
    recall_scor),"\tF1-score:%0.4f"%mean(f1),"\tgeometric mean:%0.4f"%mean(
    geo_score),"\troc auc:%0.4f"%mean(auc_score),"\tMCC:%0.4f"%mean(mcc))

"""**SMOTE and TomekLink**"""

from sklearn.linear_model import LogisticRegression
lr_smote=LogisticRegression().fit(X_train,y_train)
y_pred=lr_smote.predict(X_train)
print(classification_report(y_train,y_pred))

y_pred=lr_smote.predict(X_test)
print(classification_report(y_test,y_pred))

y_pred=lr_smote.predict(X_test)
auc=roc_auc_score(y_test,y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
gmean=geometric_mean_score(y_test,y_pred)
mcc=matthews_corrcoef(y_test,y_pred)
print("\tprecision:%0.4f"%prec,"\trecall:%0.4f"%rec,"\tF1-score:%0.4f"%f1,"\
    tgeometric mean:%0.4f"%gmean,"\troc auc:%0.4f"%auc,"\tMCC:%0.4f"%mcc)

"""**Decision Trees**"""

from sklearn.tree import DecisionTreeClassifier
dt=DecisionTreeClassifier()
dt.fit(X_train,y_train)
y_pred=dt.predict(X_test)
yhat=dt.predict_proba(X_test)
print(classification_report(y_test,y_pred))

#Decision tree is overfitting
y_pred_t=dt.predict(X_train)
print(classification_report(y_train,y_pred_t))

from sklearn.model_selection import RandomizedSearchCV
param_grid={'criterion':['gini','entropy'],'min_samples_split':[1,3,5,8], '
    min_samples_leaf':[3,6,9,12]}
grid_dt= RandomizedSearchCV(dt,param_grid,scoring='roc_auc',verbose=True,cv=5,
    n_jobs=-1)
best_dt=grid_dt.fit(X_train,y_train)
```

```
print(best_dt.best_params_)

dt_hyp=DecisionTreeClassifier(criterion='entropy',min_samples_split=5,
    min_samples_leaf=12)
dt_hyp.fit(X_train,y_train)
y_pred=dt_hyp.predict(X_test)
yhat=dt_hyp.predict_proba(X_test)
#There is an increase in precision and recall
print(classification_report(y_test,y_pred))

y_pred_tr=dt_hyp.predict(X_train)
print(classification_report(y_train,y_pred_tr))

yhat=dt_hyp.predict_proba(X_test)
average_precision = average_precision_score(y_test, yhat[:,1])
dthyp_pr_curve = plot_precision_recall_curve(dt_hyp, X_test, y_test)
dthyp_pr_curve.ax_.set_title('Precision-Recall curve: '
                   'Avearge precision={0:0.2f}'.format(average_precision))

from sklearn.metrics import roc_curve, roc_auc_score
false_positive_rate, true_positive_rate, threshold = roc_curve(y_test, yhat
    [:,1])
fig, ax = plt.subplots(figsize=(8,5))
plt.title('ROC Curve for Decision tree')
plt.plot(false_positive_rate, true_positive_rate)
plt.plot([0, 1], ls="--")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()

#Test dataset
yhat=dt_hyp.predict(X_test)
auc=roc_auc_score(y_test, yhat)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
gmean=geometric_mean_score(y_test,y_pred)
mcc=matthews_corrcoef(y_test,y_pred)
print("\tprecision:%0.4f"%prec,"\trecall:%0.4f"%rec,"\tF1-score:%0.4f"%f1,"\
    tgeometric mean:%0.4f"%gmean,"\troc auc:%0.4f"%auc,"\tMCC:%0.4f"%mcc)

"""**Decision Tree Bootstrap Estimate**"""

estimator=DecisionTreeClassifier(criterion='gini',min_samples_split=10,
    min_samples_leaf=5)
est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=precision_score)
```

```
print(f"Precision estimate on test dataset: {est:.4f}, confidence interval: [{
    low:.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=recall_score)
print(f"recall estimate on test dataset: {est:.4f}, confidence interval: [{low
    :.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=f1_score)
print(f"f1 estimate on test dataset: {est:.4f}, confidence interval: [{low:.4f
    }, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=geometric_mean_score)
print(f"geometric mean estimate on test dataset: {est:.4f}, confidence
    interval: [{low:.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=roc_auc_score)
print(f"roc auc estimate on test dataset: {est:.4f}, confidence interval: [{
    low:.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=matthews_corrcoef)
print(f"MCC estimate on test dataset: {est:.4f}, confidence interval: [{low:.4
    f}, {up:.4f}], " f"standard error: {stderr:.4f}")

"""**Decision Tree for One-Sided Sampling**"""

from sklearn.tree import DecisionTreeClassifier
dt_bal=DecisionTreeClassifier().fit(X_train,y_train)
yhat=dt_bal.predict(X_train)
print(classification_report(y_train,yhat))

y_pred=dt_bal.predict(X_test)
print(classification_report(y_test,y_pred))

from sklearn.model_selection import RandomizedSearchCV
param_grid={'criterion':['gini','entropy'],'max_depth':[3,5,7,9],'
    min_samples_split':[2,4,6], 'min_samples_leaf':[1,3,5]}
grid_dt= RandomizedSearchCV(dt_bal,param_grid,scoring='roc_auc',verbose=True,
    cv=5,n_jobs=-1)
best_dt=grid_dt.fit(X_train,y_train)

print(best_dt.best_params_)
```

```
dt_hyp=DecisionTreeClassifier(criterion='gini',min_samples_split=6,
    min_samples_leaf=5,max_depth=9)
dt_hyp.fit(X_train,y_train)
y_pred=dt_hyp.predict(X_test)
yhat=dt_hyp.predict_proba(X_test)
#There is an increase in precision and recall
print(classification_report(y_test,y_pred))

y_pred_tr=dt_hyp.predict(X_train)
print(classification_report(y_train,y_pred_tr))

yhat=dt_hyp.predict_proba(X_test)
average_precision = average_precision_score(y_test, yhat[:,1])
dthyp_pr_curve = plot_precision_recall_curve(dt_hyp, X_test, y_test)
dthyp_pr_curve.ax_.set_title('Precision-Recall curve: '
                   'Avearge precision={0:0.2f}'.format(average_precision))

from sklearn.metrics import roc_curve, roc_auc_score
false_positive_rate, true_positive_rate, threshold = roc_curve(y_test, yhat
    [:,1])
fig, ax = plt.subplots(figsize=(8,5))
plt.title('ROC Curve for Decision tree')
plt.plot(false_positive_rate, true_positive_rate)
plt.plot([0, 1], ls="--")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()

#Test dataset
yhat=dt_hyp.predict(X_test)
auc=roc_auc_score(y_test, yhat)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
gmean=geometric_mean_score(y_test,y_pred)
mcc=matthews_corrcoef(y_test,y_pred)
print("\tprecision:%0.4f"%prec,"\trecall:%0.4f"%rec,"\tF1-score:%0.4f"%f1,"\
    tgeometric mean:%0.4f"%gmean,"\troc auc:%0.4f"%auc,"\tMCC:%0.4f"%mcc)

"""**Decision Tree Bootstrap Point Estimate for One-Sided Sampled Dataset**"""

estimator=DecisionTreeClassifier(criterion='gini',min_samples_split=6,
    min_samples_leaf=5,max_depth=9)
est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=precision_score)
print(f"Precision estimate on test dataset: {est:.4f}, confidence interval: [{
    low:.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")
```

```
est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=recall_score)
print(f"recall estimate on test dataset: {est:.4f}, confidence interval: [{low
    :.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=f1_score)
print(f"f1 estimate on test dataset: {est:.4f}, confidence interval: [{low:.4f
    }, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=geometric_mean_score)
print(f"geometric mean estimate on test dataset: {est:.4f}, confidence
    interval: [{low:.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=roc_auc_score)
print(f"roc auc estimate on test dataset: {est:.4f}, confidence interval: [{
    low:.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=matthews_corrcoef)
print(f"MCC estimate on test dataset: {est:.4f}, confidence interval: [{low:.4
    f}, {up:.4f}], " f"standard error: {stderr:.4f}")

"""**Random undersampling**"""

dt_rus=DecisionTreeClassifier().fit(X_train,y_train)
yhat=dt_rus.predict(X_train)
print(classification_report(y_train,yhat))

y_pred=dt_rus.predict(X_test)
print(classification_report(y_test,y_pred))

from sklearn.tree import DecisionTreeClassifier
#K-Fold cross validation
kf = StratifiedKFold(n_splits=10,shuffle=True,random_state=42)
auc_score =[]
recall_scor=[]
precision_scor=[]
geo_score=[]
f1=[]
mcc=[]
i=1
for train_index,test_index in kf.split(X_rus,y_rus):
    #print('{} of KFold {}'.format(i,kf.n_splits))
    xtrain,xtest = X_rus.iloc[train_index],X_rus.iloc[test_index]
    ytrain,ytest = y_rus.iloc[train_index],y_rus.iloc[test_index]
```

```
    #model
    dt_rus=DecisionTreeClassifier().fit(xtrain,ytrain)
    score = roc_auc_score(ytest,dt_rus.predict(xtest))
    score1 = recall_score(ytest,dt_rus.predict(xtest))
    score2 = precision_score(ytest,dt_rus.predict(xtest))
    score3=geometric_mean_score(ytest,dt_rus.predict(xtest))
    score4=f1_score(ytest,dt_rus.predict(xtest))
    score5=matthews_corrcoef(ytest,dt_rus.predict(xtest))
    #
        ##########################################################################################

    auc_score.append(score)
    recall_scor.append(score1)
    precision_scor.append(score2)
    geo_score.append(score3)
    f1.append(score4)
    mcc.append(score5)

    i+=1

print("\tprecision:%0.4f"%mean(precision_scor),"\trecall:%0.4f"%mean(
    recall_scor),"\tF1-score:%0.4f"%mean(f1),"\tgeometric mean:%0.4f"%mean(
    geo_score),"\troc auc:%0.4f"%mean(auc_score),"\tMCC:%0.4f"%mean(mcc))

y_pred=dt_rus.predict(X_test)
auc=roc_auc_score(y_test,y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
gmean=geometric_mean_score(y_test,y_pred)
mcc=matthews_corrcoef(y_test,y_pred)
print("\tprecision:%0.4f"%prec,"\trecall:%0.4f"%rec,"\tF1-score:%0.4f"%f1,"\
    tgeometric mean:%0.4f"%gmean,"\troc auc:%0.4f"%auc,"\tMCC:%0.4f"%mcc)

"""**SMOTE and TomekLink**"""

from sklearn.tree import DecisionTreeClassifier
dt_smote=DecisionTreeClassifier().fit(X_train,y_train)
yhat=dt_smote.predict(X_train)
print(classification_report(y_train,yhat))

y_pred=dt_smote.predict(X_test)
print(classification_report(y_test,y_pred))

y_pred=dt_smote.predict(X_test)
auc=roc_auc_score(y_test,y_pred)
prec = precision_score(y_test, y_pred)
```

```
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
gmean=geometric_mean_score(y_test,y_pred)
mcc=matthews_corrcoef(y_test,y_pred)
print("\tprecision:%0.4f"%prec,"\trecall:%0.4f"%rec,"\tF1-score:%0.4f"%f1,"\
    tgeometric mean:%0.4f"%gmean,"\troc auc:%0.4f"%auc,"\tMCC:%0.4f"%mcc)

"""
    ##########################################################################################################


**Multilayer Perceptron**
"""

from sklearn.neural_network import MLPClassifier
mpc=MLPClassifier().fit(X_train,y_train)
y_hat=mpc.predict(X_train)
print(classification_report(y_train,y_hat))

y_mpc=mpc.predict(X_test)
print(classification_report(y_test,y_mpc))

from sklearn.model_selection import RandomizedSearchCV
# defining parameter range
param_grid = {
     'hidden_layer_sizes': [(10,30,10),(20,)],
     'activation': ['tanh', 'relu'],
     'solver': ['sgd', 'adam'],
     'alpha': [0.01,0.03, 0.05],
     'learning_rate': ['constant','adaptive']}
rs_mpc=RandomizedSearchCV(MLPClassifier(), param_grid,scoring='roc_auc',
    verbose=True,cv=5,n_jobs=-1)
best_mpc=rs_mpc.fit(X_train,y_train)

print(best_mpc.best_params_)

from sklearn.neural_network import MLPClassifier
mpc_hyp=MLPClassifier(activation="relu",alpha=0.03,learning_rate="adaptive",
    hidden_layer_sizes=(10,30,10),solver="adam")
mpc_hyp.fit(X_train,y_train)
yhat=mpc_hyp.predict_proba(X_test)

#There is an increase in precision and recall
y_pred_train=mpc_hyp.predict(X_train)
print(classification_report(y_train,y_pred_train))

y_pred=mpc_hyp.predict(X_test)
print(classification_report(y_test,y_pred))
```

```
yhat=mpc_hyp.predict_proba(X_test)
average_precision = average_precision_score(y_test, yhat[:,1])
mpchyp_pr_curve = plot_precision_recall_curve(mpc_hyp, X_test, y_test)
mpchyp_pr_curve.ax_.set_title('Precision-Recall curve: '
                   'Avearge precision={0:0.2f}'.format(average_precision))

from sklearn.metrics import roc_curve, roc_auc_score
false_positive_rate, true_positive_rate, threshold = roc_curve(y_test, yhat
    [:,1])
fig, ax = plt.subplots(figsize=(8,5))
plt.title('ROC Curve for Multilayer perceptron')
plt.plot(false_positive_rate, true_positive_rate)
plt.plot([0, 1], ls="--")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()

#Test dataset from the model
auc=roc_auc_score(y_test, mpc_hyp.predict(X_test))
prec = precision_score(y_test, mpc_hyp.predict(X_test))
rec = recall_score(y_test, mpc_hyp.predict(X_test))
f1 = f1_score(y_test, mpc_hyp.predict(X_test))
gmean=geometric_mean_score(y_test,mpc_hyp.predict(X_test))
mcc=matthews_corrcoef(y_test,mpc_hyp.predict(X_test))

print("\tprecision:%0.4f"%prec,"\trecall:%0.4f"%rec,"\tF1-score:%0.4f"%f1,"\
    tgeometric mean:%0.4f"%gmean,"\troc auc:%0.4f"%auc,"\tMCC:%0.4f"%mcc)

"""**Multilayer Perceptron Bootstrap Estimate**

"""

mpc_hyp=MLPClassifier(activation="relu",alpha=0.03,learning_rate="adaptive",
    hidden_layer_sizes=(10,30,10),solver="adam")
est, low, up, stderr = bootstrap_estimate_and_ci(mpc_hyp, xtest, ytest,
    scoring_func=precision_score)
print(f"precision estimate on test dataset: {est:.4f}, confidence interval: [{
    low:.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(mpc_hyp, xtest, ytest,
    scoring_func=recall_score)
print(f"recall estimate on test dataset: {est:.4f}, confidence interval: [{low
    :.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(mpc_hyp, xtest, ytest,
    scoring_func=f1_score)
```

```
print(f"f1 estimate on test dataset: {est:.4f}, confidence interval: [{low:.4f
    }, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(mpc_hyp, xtest, ytest,
    scoring_func=geometric_mean_score)
print(f"geometric mean estimate on test dataset: {est:.4f}, confidence
    interval: [{low:.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(mpc_hyp, xtest, ytest,
    scoring_func=roc_auc_score)
print(f"roc auc estimate on test dataset: {est:.4f}, confidence interval: [{
    low:.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(mpc_hyp, xtest, ytest,
    scoring_func=matthews_corrcoef)
print(f"MCC estimate on test dataset: {est:.4f}, confidence interval: [{low:.4
    f}, {up:.4f}], " f"standard error: {stderr:.4f}")

"""**Multilayer Perceptron One_Sided Sampling**


"""

from sklearn.neural_network import MLPClassifier
mpc_bal=MLPClassifier().fit(X_train,y_train)
yhat=mpc_bal.predict(X_train)
print(classification_report(y_train,yhat))

yhat=mpc_bal.predict(X_test)
print(classification_report(y_test,yhat))

from sklearn.model_selection import RandomizedSearchCV
# defining parameter range
param_grid = {
    'hidden_layer_sizes': [(10,30,10),(20,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.01,0.03, 0.05],
    'learning_rate': ['constant','adaptive']}
rs_mpc=RandomizedSearchCV(MLPClassifier(), param_grid,scoring='roc_auc',
    verbose=True,cv=5,n_jobs=-1)
best_mpc=rs_mpc.fit(X_train,y_train)

print(best_mpc.best_params_)

from sklearn.neural_network import MLPClassifier
mpc_hyp=MLPClassifier(activation="relu",alpha=0.01,learning_rate="constant",
    hidden_layer_sizes=(20,),solver="adam")
```

```
mpc_hyp.fit(X_train,y_train)
yhat=mpc_hyp.predict_proba(X_test)

#There is an increase in precision and recall
y_pred_train=mpc_hyp.predict(X_train)
print(classification_report(y_train,y_pred_train))

y_pred=mpc_hyp.predict(X_test)
print(classification_report(y_test,y_pred))

yhat=mpc_hyp.predict_proba(X_test)
average_precision = average_precision_score(y_test, yhat[:,1])
mpchyp_pr_curve = plot_precision_recall_curve(mpc_hyp, X_test, y_test)
mpchyp_pr_curve.ax_.set_title('Precision-Recall curve: '
                    'Avearge precision={0:0.2f}'.format(average_precision))

from sklearn.metrics import roc_curve, roc_auc_score
false_positive_rate, true_positive_rate, threshold = roc_curve(y_test, yhat
    [:,1])
fig, ax = plt.subplots(figsize=(8,5))
plt.title('ROC Curve for Multilayer perceptron')
plt.plot(false_positive_rate, true_positive_rate)
plt.plot([0, 1], ls="--")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()

#Test dataset from the model
auc=roc_auc_score(y_test, mpc_hyp.predict(X_test))
prec = precision_score(y_test, mpc_hyp.predict(X_test))
rec = recall_score(y_test, mpc_hyp.predict(X_test))
f1 = f1_score(y_test, mpc_hyp.predict(X_test))
gmean=geometric_mean_score(y_test,mpc_hyp.predict(X_test))
mcc=matthews_corrcoef(y_test,mpc_hyp.predict(X_test))
print("\tprecision:%0.4f"%prec,"\trecall:%0.4f"%rec,"\tF1-score:%0.4f"%f1,"\
    tgeometric mean:%0.4f"%gmean,"\troc auc:%0.4f"%auc,"\tMCC:%0.4f"%mcc)

"""**MLP Bootstrap Point Estimate for One-Sided Samplling**"""

mpc_hyp=MLPClassifier(activation="relu",alpha=0.01,learning_rate="constant",
    hidden_layer_sizes=(20,),solver="adam")
est, low, up, stderr = bootstrap_estimate_and_ci(mpc_hyp, xtest, ytest,
    scoring_func=precision_score)
print(f"precision estimate on test dataset: {est:.4f}, confidence interval: [{
    low:.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(mpc_hyp, xtest, ytest,
    scoring_func=recall_score)
```

```
print(f"recall estimate on test dataset: {est:.4f}, confidence interval: [{low
    :.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(mpc_hyp, xtest, ytest,
    scoring_func=f1_score)
print(f"f1 estimate on test dataset: {est:.4f}, confidence interval: [{low:.4f
    }, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(mpc_hyp, xtest, ytest,
    scoring_func=geometric_mean_score)
print(f"geometric mean estimate on test dataset: {est:.4f}, confidence
    interval: [{low:.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(mpc_hyp, xtest, ytest,
    scoring_func=roc_auc_score)
print(f"roc auc estimate on test dataset: {est:.4f}, confidence interval: [{
    low:.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(mpc_hyp, xtest, ytest,
    scoring_func=matthews_corrcoef)
print(f"MCC estimate on test dataset: {est:.4f}, confidence interval: [{low:.4
    f}, {up:.4f}], " f"standard error: {stderr:.4f}")

"""**Random undersampling**"""

X_train, X_test,y_train,y_test=train_test_split(X_rus,y_rus,test_size=0.3,
    stratify=y_rus,random_state=42)

mpc_rus=MLPClassifier().fit(X_train,y_train)
yhat=mpc_rus.predict(X_train)
print(classification_report(y_train,yhat))

y_hat=mpc_rus.predict(X_test)
print(classification_report(y_test,y_hat))

from sklearn.neural_network import MLPClassifier
#K-Fold cross validation
kf = StratifiedKFold(n_splits=10,shuffle=True,random_state=42)
auc_score =[]
recall_scor=[]
precision_scor=[]
geo_score=[]
f1=[]
mcc=[]
i=1
for train_index,test_index in kf.split(X_rus,y_rus):
    #print('{} of KFold {}'.format(i,kf.n_splits))
    xtrain,xtest = X_rus.iloc[train_index],X_rus.iloc[test_index]
```

```
    ytrain,ytest = y_rus.iloc[train_index],y_rus.iloc[test_index]

    #model
    mpc_rus=MLPClassifier().fit(xtrain,ytrain)
    score = roc_auc_score(ytest,mpc_rus.predict(xtest))
    score1 = recall_score(ytest,mpc_rus.predict(xtest))
    score2 = precision_score(ytest,mpc_rus.predict(xtest))
    score3=geometric_mean_score(ytest,mpc_rus.predict(xtest))
    score4=f1_score(ytest,mpc_rus.predict(xtest))
    score5=matthews_corrcoef(ytest,mpc_rus.predict(xtest))
    #
        ##########################################################################################

    auc_score.append(score)
    recall_scor.append(score1)
    precision_scor.append(score2)
    geo_score.append(score3)
    f1.append(score4)
    mcc.append(score5)

    i+=1

print("\tprecision:%0.4f"%mean(precision_scor),"\trecall:%0.4f"%mean(
    recall_scor),"\tF1-score:%0.4f"%mean(f1),"\tgeometric mean:%0.4f"%mean(
    geo_score),"\troc auc:%0.4f"%mean(auc_score),"\tMCC:%0.4f"%mean(mcc))

"""**SMOTE and TomekLink**"""

from sklearn.neural_network import MLPClassifier
mpc_smote=MLPClassifier().fit(X_train,y_train)
yhat=mpc_smote.predict(X_train)
print(classification_report(y_train,yhat))

y_hat=mpc_smote.predict(X_test)
print(classification_report(y_test,y_hat))

auc=roc_auc_score(y_test, y_hat)
prec = precision_score(y_test,y_hat)
rec = recall_score(y_test, y_hat)
f1 = f1_score(y_test,y_hat)
gmean=geometric_mean_score(y_test,y_hat)
mcc=matthews_corrcoef(y_test,y_hat)
print("\tprecision:%0.4f"%prec,"\trecall:%0.4f"%rec,"\tF1-score:%0.4f"%f1,"\
    tgeometric mean:%0.4f"%gmean,"\troc auc:%0.4f"%auc,"\tMCC:%0.4f"%mcc)

"""
    ##########################################################################################
```

```
**Principal Component Analysis (Feature Extraction)**
"""

#Using elbow-plot variance/dimensions
from sklearn.decomposition import PCA
pca = PCA()
pca.fit(X_train)
cumsum = np.cumsum(pca.explained_variance_ratio_)*100
d = [n for n in range(len(cumsum))]
plt.figure(figsize=(6, 6))
plt.plot(d,cumsum, color = 'red',label='cumulative explained variance')
plt.title('Cumulative Explained Variance as\n a Function of the Number of
    Components.')
plt.ylabel('Cumulative Explained variance')
plt.xlabel('Principal components')
plt.axhline(y = 95, color='k', linestyle='--', label = '95% Explained Variance
    ')
plt.legend(loc='best')

from sklearn.model_selection import RandomizedSearchCV
# defining parameter range
n_components=[x for x in range(1,31)]
param_grid = {'n_components':n_components}
rs_pca = RandomizedSearchCV(PCA(),param_grid, scoring='recall', verbose = 2,
    n_jobs=-1, cv=5)
rs_pca.fit(X_train)

print(rs_pca.best_params_)

pca = PCA(n_components=17)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
y=pd.concat([y_train,y_test])
print('Variance explained is ratio {}%'.format(((pca.explained_variance_ratio_
    ).sum()).round(4)))

X_train_pca.shape

"""**Support Vector Machine**"""

#Import svm model
from sklearn import svm
#Create a svm Classifier
svm_clf = svm.SVC()
#Train the model using the training sets
svm_clf .fit(X_train_pca, y_train)
```

```
y_pred=svm_clf.predict(X_test_pca)
print(classification_report(y_test,y_pred))

y_pred_tr=svm_clf.predict(X_train_pca)
print(classification_report(y_train,y_pred_tr))

from sklearn.model_selection import RandomizedSearchCV
# defining parameter range
param_grid = {'C':[ 0.8,0.85,0.9,0.95],'kernel':['linear','rbf','poly','
    sigmoid']}
rs_svm = RandomizedSearchCV(svm.SVC(max_iter=200, tol=2,probability=True),
    param_grid, scoring='roc_auc', verbose = True, n_jobs=-1, cv=5)
rs_svm.fit(X_train_pca, y_train)

print(rs_svm.best_params_)

from sklearn import svm
svm_hyp=svm.SVC(C=0.95,kernel='rbf',max_iter=200, tol=2,probability=True).fit(
    X_train_pca,y_train)
#Predict the response for test dataset
y_pred = svm_hyp.predict_proba(X_test_pca)
y_hat=svm_hyp.predict(X_test_pca)
print(classification_report(y_test,y_hat))

#Predict the response for train dataset
y_pred_tr = svm_hyp.predict(X_train_pca)
print(classification_report(y_train,y_pred_tr))

average_precision = average_precision_score(y_test, y_pred[:,1])
svm_pr_curve = plot_precision_recall_curve(svm_hyp, X_test_pca, y_test)
svm_pr_curve.ax_.set_title('Precision-Recall curve: '
                   'Avearge precision={0:0.2f}'.format(average_precision))

from sklearn.metrics import roc_curve, roc_auc_score
prob=svm_hyp.predict_proba(X_test_pca)[:,1]
false_positive_rate, true_positive_rate, threshold = roc_curve(y_test, prob)
fig, ax = plt.subplots(figsize=(8,5))
plt.title('ROC Curve for SVM')
plt.plot(false_positive_rate, true_positive_rate)
plt.plot([0, 1], ls="--")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()

y_hat=svm_hyp.predict(X_test_pca)
auc=roc_auc_score(y_test,y_hat)
prec = precision_score(y_test, y_hat)
rec = recall_score(y_test, y_hat)
```

```
f1 = f1_score(y_test,y_hat)
gmean=geometric_mean_score(y_test,y_hat)
mcc=matthews_corrcoef(y_test,y_hat)
print("\tprecision:%0.4f"%prec,"\trecall:%0.4f"%rec,"\tF1-score:%0.4f"%f1,"\
    tgeometric mean:%0.4f"%gmean,"\troc auc:%0.4f"%auc,"\tMCC:%0.4f"%mcc)

"""**SVM bootstrap estimate**"""

xtest=X_test_pca
ytest=y_test.values #Make it an array
ytest

# Calculate a bootstrap estimate for recall and a 95% confidence interval
estimator=svm.SVC(C=0.9,kernel='rbf',max_iter=200, tol=2,probability=True)
est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=precision_score)
print(f"precision estimate on test dataset: {est:.4f}, confidence interval: [{
    low:.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=recall_score)
print(f"recall estimate on test dataset: {est:.4f}, confidence interval: [{low
    :.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=f1_score)
print(f"f1 estimate on test dataset: {est:.4f}, confidence interval: [{low:.4f
    }, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=geometric_mean_score)
print(f"geometric mean estimate on test dataset: {est:.4f}, confidence
    interval: [{low:.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=roc_auc_score)
print(f"AUC estimate on test dataset: {est:.4f}, confidence interval: [{low:.4
    f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=matthews_corrcoef)
print(f"MCC estimate on test dataset: {est:.4f}, confidence interval: [{low:.4
    f}, {up:.4f}], " f"standard error: {stderr:.4f}")

"""**Random Undersampling with SVM**"""

from imblearn.under_sampling import RandomUnderSampler
X=creditdata_normalized.drop('Class',axis=1)
```

```
y=creditdata_normalized['Class']
rus = RandomUnderSampler(random_state=42)
X_rus, y_rus= rus.fit_resample(X, y)

from sklearn import svm
X_train, X_test, y_train, y_test=train_test_split(X_rus,y_rus,test_size=0.3,
    stratify=y_rus, random_state=42)
svm_bal=svm.SVC(kernel='linear',probability=True).fit(X_train,y_train)

yhat=svm_bal.predict(X_train)
print(classification_report(y_train,yhat))

yhat=svm_bal.predict(X_test)
print(classification_report(y_test,yhat))

"""
    ##########################################################################################


**K-nearest neighbor(KNN)**
"""

from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier().fit(X_train_pca,y_train)
y_pred=knn.predict(X_test_pca)

print(classification_report(y_test,y_pred))

y_tr=knn.predict(X_train_pca)

print(classification_report(y_train,y_tr))

"""**KNN Random Search**"""

#Knn rs search
from sklearn.model_selection import RandomizedSearchCV
n_neighbors=[i for i in range(2,9)]
p=[1,2]
param_grid={'n_neighbors':n_neighbors,'p':p}
rs_knn=RandomizedSearchCV(KNeighborsClassifier(),param_grid,cv=5,n_jobs=-1,
    verbose=True, scoring='roc_auc')
best_knn=rs_knn.fit(X_train_pca,y_train)

print(best_knn.best_params_)

from sklearn.neighbors import KNeighborsClassifier
knn_hyp = KNeighborsClassifier(n_neighbors=6,p=1).fit(X_train_pca,y_train)
predict=knn_hyp.predict(X_test_pca)
```

```
#yhat=knn_hyp.predict_proba(X_test)
print(classification_report(y_test,predict))

predict_train=knn_hyp.predict(X_train_pca)

print(classification_report(y_train,predict_train))

Knn_hat=knn_hyp.predict_proba(X_test_pca)[:,1]

average_precision = average_precision_score(y_test, Knn_hat)
knn_pr_curve = plot_precision_recall_curve(knn_hyp, X_test_pca, y_test)
knn_pr_curve.ax_.set_title('Precision-Recall curve: '
                   'Avearge precision={0:0.2f}'.format(average_precision))

from sklearn.metrics import roc_curve, roc_auc_score
false_positive_rate, true_positive_rate, threshold = roc_curve(y_test, Knn_hat
    )
fig, ax = plt.subplots(figsize=(8,5))
plt.title('ROC Curve for KNN')
plt.plot(false_positive_rate, true_positive_rate)
plt.plot([0, 1], ls="--")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()

#Test dataset
auc=roc_auc_score(y_test,predict)
prec = precision_score(y_test, predict)
rec = recall_score(y_test, predict)
f1 = f1_score(y_test, predict)
gmean=geometric_mean_score(y_test,predict)
mcc=matthews_corrcoef(y_test,predict)
print("\tprecision:%0.4f"%prec,"\trecall:%0.4f"%rec,"\tF1-score:%0.4f"%f1,"\
    tgeometric mean:%0.4f"%gmean,"\troc auc:%0.4f"%auc,"\tMCC:%0.4f"%mcc)

"""**KNN Bootstrap Estimate**"""

xtest=X_test_pca
ytest=y_test.values

# Calculate a bootstrap estimate for recall and a 95% confidence interval
estimator=KNeighborsClassifier(n_neighbors = 5,p=2)
est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=precision_score)
print(f"precision estimate on test dataset: {est:.4f}, confidence interval: [{
    low:.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")
```

```
est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=recall_score)
print(f"recall estimate on test dataset: {est:.4f}, confidence interval: [{low
    :.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=f1_score)
print(f"f1 estimate on test dataset: {est:.4f}, confidence interval: [{low:.4f
    }, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=geometric_mean_score)
print(f"geometric mean estimate on test dataset: {est:.4f}, confidence
    interval: [{low:.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=roc_auc_score)
print(f"roc_auc estimate on test dataset: {est:.4f}, confidence interval: [{
    low:.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=matthews_corrcoef)
print(f"MCC estimate on test dataset: {est:.4f}, confidence interval: [{low:.4
    f}, {up:.4f}], " f"standard error: {stderr:.4f}")

"""**Random Undersampling with kNN**"""

knn=KNeighborsClassifier().fit(X_train,y_train)
y_pred=knn.predict(X_train)
print(classification_report(y_train,y_pred))

y_pred=knn.predict(X_test)
print(classification_report(y_test,y_pred))

"""
    ##########################################################################################


Logistic Regression
"""

from sklearn.linear_model import LogisticRegression
#Basic Logistic regression on raw dataset
lr = LogisticRegression().fit(X_train_pca, y_train)
y_pred=lr.predict(X_test_pca)
yhat=lr.predict_proba(X_test_pca)
print(classification_report(y_test,y_pred))
```

```
y_pred_train=lr.predict(X_train_pca)
print(classification_report(y_train,y_pred_train))

#Logistic regression random search
from sklearn.model_selection import RandomizedSearchCV
param_grid={'C':[0.1,0.3,0.5,0.7,0.9],'solver':['lbfgs','newton-cg','sag','
    saga'], 'penalty':['l2','l1','elasticnet','none'],'class_weight'
    :[{0:1,1:3},{0:1,1:5},{0:1,1:8},{0:1,1:10}]}
rs_lr=RandomizedSearchCV(LogisticRegression(max_iter=1000),param_grid,cv=5,
    n_jobs=-1,verbose=True, scoring='roc_auc')
rs_lr.fit(X_train_pca,y_train)

print(rs_lr.best_params_)

from sklearn.linear_model import LogisticRegression
lr_hyp = LogisticRegression(C=0.1,solver='newton-cg',penalty='l2',max_iter
    =1000,class_weight={0:1,1:5}).fit(X_train_pca, y_train)
y_pred=lr_hyp.predict(X_test_pca)
print(classification_report(y_test,y_pred))

y_pred_tr=lr_hyp.predict(X_train_pca)
print(classification_report(y_train,y_pred_tr))

yhat=lr_hyp.predict_proba(X_test_pca)
average_precision = average_precision_score(y_test, yhat[:,1])
lr_pr_curve = plot_precision_recall_curve(lr_hyp, X_test_pca, y_test)
lr_pr_curve.ax_.set_title('Precision-Recall curve: '
                  'Avearge precision={0:0.2f}'.format(average_precision))

from sklearn.metrics import roc_curve, roc_auc_score
yhat=lr_hyp.predict_proba(X_test_pca)
false_positive_rate, true_positive_rate, threshold = roc_curve(y_test, yhat
    [:,1])
fig, ax = plt.subplots(figsize=(8,5))
plt.title('ROC Curve for LR')
plt.plot(false_positive_rate, true_positive_rate)
plt.plot([0, 1], ls="--")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()

y_pred=lr_hyp.predict(X_test_pca)
auc=roc_auc_score(y_test,y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
gmean=geometric_mean_score(y_test,y_pred)
mcc=matthews_corrcoef(y_test,y_pred)
```

```
print("\tprecision:%0.4f"%prec,"\trecall:%0.4f"%rec,"\tF1-score:%0.4f"%f1,"\
    tgeometric mean:%0.4f"%gmean,"\troc auc:%0.4f"%auc,"\tMCC:%0.4f"%mcc)

"""**Logistic regression Bootstrap Estimate**"""

estimator=LogisticRegression(C=0.1,solver='newton-cg',penalty='l2',max_iter
    =1000,class_weight={0:1,1:5})
est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=precision_score)
print(f"precision estimate on test dataset: {est:.2f}, confidence interval: [{
    low:.2f}, {up:.2f}], " f"standard error: {stderr:.2f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=recall_score)
print(f"recall estimate on test dataset: {est:.4f}, confidence interval: [{low
    :.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=f1_score)
print(f"f1 estimate on test dataset: {est:.4f}, confidence interval: [{low:.4f
    }, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=geometric_mean_score)
print(f"Geomtric mean estimate on test dataset: {est:.4f}, confidence interval
    : [{low:.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=roc_auc_score)
print(f"AUC estimate on test dataset: {est:.4f}, confidence interval: [{low:.4
    f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=matthews_corrcoef)
print(f"MCC estimate on test dataset: {est:.4f}, confidence interval: [{low:.4
    f}, {up:.4f}], " f"standard error: {stderr:.4f}")

"""**Random Undersampling**"""

from sklearn.linear_model import LogisticRegression
lr_bal = LogisticRegression().fit(X_train, y_train)
y_pred=lr_bal.predict(X_train)
print(classification_report(y_train,y_pred))

y_pred=lr_bal.predict(X_test)
print(classification_report(y_test,y_pred))
```

```
"""
    ##########################################################################################################


**Decision Tree**
"""

from sklearn.tree import DecisionTreeClassifier
dt=DecisionTreeClassifier()
dt.fit(X_train_pca,y_train)
y_pred=dt.predict(X_test_pca)
yhat=dt.predict_proba(X_test_pca)
print(classification_report(y_test,y_pred))

#Decision tree is overfitting
y_pred_t=dt.predict(X_train_pca)
print(classification_report(y_train,y_pred_t))

"""**Decision Tree Random Search**"""

from sklearn.model_selection import RandomizedSearchCV
param_grid={'criterion':['gini','entropy'],'max_depth':[7,9,11,13],'
    min_samples_split':[2,4,6], 'min_samples_leaf':[1,3,5]}
grid_dt= RandomizedSearchCV(dt,param_grid,scoring='roc_auc',verbose=True,cv=5,
    n_jobs=-1)
best_dt=grid_dt.fit(X_train_pca,y_train)

print(best_dt.best_params_)

dt_hyp=DecisionTreeClassifier(criterion='gini',min_samples_split=4,
    min_samples_leaf=5,max_depth=11)
dt_hyp.fit(X_train_pca,y_train)
y_pred=dt_hyp.predict(X_test_pca)
yhat=dt_hyp.predict_proba(X_test_pca)
#There is an increase in precision and recall
print(classification_report(y_test,y_pred))

y_pred_tr=dt_hyp.predict(X_train_pca)
print(classification_report(y_train,y_pred_tr))

yhat=dt_hyp.predict_proba(X_test_pca)
average_precision = average_precision_score(y_test, yhat[:,1])
dthyp_pr_curve = plot_precision_recall_curve(dt_hyp, X_test_pca, y_test)
dthyp_pr_curve.ax_.set_title('Precision-Recall curve: '
                  'Avearge precision={0:0.2f}'.format(average_precision))

from sklearn.metrics import roc_curve, roc_auc_score
```

```
false_positive_rate, true_positive_rate, threshold = roc_curve(y_test, yhat
    [:,1])
fig, ax = plt.subplots(figsize=(8,5))
plt.title('ROC Curve for Decision tree')
plt.plot(false_positive_rate, true_positive_rate)
plt.plot([0, 1], ls="--")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()

#Test dataset
yhat=dt_hyp.predict(X_test_pca)
auc=roc_auc_score(y_test, yhat)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
gmean=geometric_mean_score(y_test,y_pred)
mcc=matthews_corrcoef(y_test,y_pred)
print("\tprecision:%0.4f"%prec,"\trecall:%0.4f"%rec,"\tF1-score:%0.4f"%f1,"\
    tgeometric mean:%0.4f"%gmean,"\troc auc:%0.4f"%auc,"\tMCC:%0.4f"%mcc)

"""**Decision Tree Bootstrap Estimate**"""

estimator=DecisionTreeClassifier(criterion='entropy',min_samples_split=4,
    min_samples_leaf=5,max_depth=5)
est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=precision_score)
print(f"Precision estimate on test dataset: {est:.4f}, confidence interval: [{
    low:.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=recall_score)
print(f"recall estimate on test dataset: {est:.4f}, confidence interval: [{low
    :.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=f1_score)
print(f"F1 estimate on training dataset: {est:.4f}, confidence interval: [{low
    :.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=geometric_mean_score)
print(f"Geometric mean estimate on test dataset: {est:.4f}, confidence
    interval: [{low:.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=roc_auc_score)
```

```
print(f"AUC measure estimate on test dataset: {est:.4f}, confidence interval:
    [{low:.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(estimator, xtest, ytest,
    scoring_func=matthews_corrcoef)
print(f"MCC measure estimate on test dataset: {est:.4f}, confidence interval:
    [{low:.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

"""**Random Undersampling with Decision Tree**"""

from sklearn.tree import DecisionTreeClassifier
dt_bal=DecisionTreeClassifier().fit(X_train,y_train)
y_pred=dt_bal.predict(X_train)
print(classification_report(y_train,y_pred))

y_pred=dt_bal.predict(X_test)
print(classification_report(y_test,y_pred))

"""
    ##########################################################################################################


**Multilayer Perceptron Classifier**
"""

from sklearn.neural_network import MLPClassifier
mpc=MLPClassifier().fit(X_train_pca,y_train)
y_hat=mpc.predict(X_train_pca)
print(classification_report(y_train,y_hat))

y_mpc=mpc.predict(X_test_pca)
print(classification_report(y_test,y_mpc))

from sklearn.model_selection import RandomizedSearchCV
# defining parameter range
param_grid = {
    'hidden_layer_sizes': [(10,20,30),(15,30,40)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.001,0.003, 0.004,0.005],
    'learning_rate': ['constant','adaptive']}
rs_mpc=RandomizedSearchCV(MLPClassifier(tol=0.001), param_grid,scoring='
    roc_auc',verbose=True,cv=5,n_jobs=-1)
best_mpc=rs_mpc.fit(X_train_pca,y_train)

print(best_mpc.best_params_)

from sklearn.neural_network import MLPClassifier
```

```
mpc_hyp=MLPClassifier(activation="tanh",alpha=0.004,learning_rate="constant",
    hidden_layer_sizes=(15,30,40),solver="adam",tol=0.001)
mpc_hyp.fit(X_train_pca,y_train)
yhat=mpc_hyp.predict_proba(X_test_pca)
#There is an increase in precision and recall

y_pred_train=mpc_hyp.predict(X_train_pca)
print(classification_report(y_train,y_pred_train))

y_pred=mpc_hyp.predict(X_test_pca)
print(classification_report(y_test,y_pred))

yhat=mpc_hyp.predict_proba(X_test_pca)
average_precision = average_precision_score(y_test, yhat[:,1])
mpchyp_pr_curve = plot_precision_recall_curve(mpc_hyp, X_test_pca, y_test)
mpchyp_pr_curve.ax_.set_title('Precision-Recall curve: '
                    'Avearge precision={0:0.2f}'.format(average_precision))

from sklearn.metrics import roc_curve, roc_auc_score
false_positive_rate, true_positive_rate, threshold = roc_curve(y_test, yhat
    [:,1])
fig, ax = plt.subplots(figsize=(8,5))
plt.title('ROC Curve for Multilayer perceptron')
plt.plot(false_positive_rate, true_positive_rate)
plt.plot([0, 1], ls="--")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()

#Test dataset from the model
auc=roc_auc_score(y_test, mpc_hyp.predict(X_test_pca))
prec = precision_score(y_test, mpc_hyp.predict(X_test_pca))
rec = recall_score(y_test, mpc_hyp.predict(X_test_pca))
f1 = f1_score(y_test, mpc_hyp.predict(X_test_pca))
gmean=geometric_mean_score(y_test,mpc_hyp.predict(X_test_pca))
mcc=matthews_corrcoef(y_test,mpc_hyp.predict(X_test_pca))
print("\tprecision:%0.4f"%prec,"\trecall:%0.4f"%rec,"\tF1-score:%0.4f"%f1,"\
    tgeometric mean:%0.4f"%gmean,"\troc auc:%0.4f"%auc,"\tMCC:%0.4f"%mcc)

xtest=X_test_pca
ytest=y_test.values

mpc_hyp=MLPClassifier(activation="tanh",alpha=0.004,learning_rate="constant",
    hidden_layer_sizes=(15,30,40),solver="adam",tol=0.001)
est, low, up, stderr = bootstrap_estimate_and_ci(mpc_hyp, xtest, ytest,
    scoring_func=precision_score)
print(f"precision estimate on test dataset: {est:.4f}, confidence interval: [{
    low:.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")
```

```
est, low, up, stderr = bootstrap_estimate_and_ci(mpc_hyp, xtest, ytest,
    scoring_func=recall_score)
print(f"recall estimate on test dataset: {est:.4f}, confidence interval: [{low
    :.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(mpc_hyp, xtest, ytest,
    scoring_func=f1_score)
print(f"f1 estimate on test dataset: {est:.4f}, confidence interval: [{low:.4f
    }, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(mpc_hyp, xtest, ytest,
    scoring_func=geometric_mean_score)
print(f"geometric mean estimate on test dataset: {est:.4f}, confidence
    interval: [{low:.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(mpc_hyp, xtest, ytest,
    scoring_func=roc_auc_score)
print(f"roc auc estimate on test dataset: {est:.4f}, confidence interval: [{
    low:.4f}, {up:.4f}], " f"standard error: {stderr:.4f}")

est, low, up, stderr = bootstrap_estimate_and_ci(mpc_hyp, xtest, ytest,
    scoring_func=matthews_corrcoef)
print(f"MCC estimate on test dataset: {est:.4f}, confidence interval: [{low:.4
    f}, {up:.4f}], " f"standard error: {stderr:.4f}")

"""
    ##########################################################################################


**Ramdom undersampling**
"""

from sklearn.neural_network import MLPClassifier
mpc_bal=MLPClassifier().fit(X_train,y_train)
yhat=mpc_bal.predict(X_train)
print(classification_report(y_train,yhat))

yhat=mpc_bal.predict(X_test)
print(classification_report(y_test,yhat))

"""
    ##########################################################################################
    """

D['Class'].value_counts()

"""**Future Work $\cdots$**
```

```
**Support Vector Machine**
"""

from sklearn.utils import shuffle
from statistics import mean
from sklearn import svm
#Create a svm Classifier
pr_score=[]
re_score=[]
f1=[]
G_mean=[]
AUC=[]
mcc=[]
index=448
for i,j in zip(range(0,627),range(1,628)):
  non_fraud_cases_indexed=non_fraud_cases.iloc[i*index:j*index]
  balanced_dataset=pd.concat([non_fraud_cases_indexed,fraud_cases])
  shuffle(balanced_dataset)
  X = balanced_dataset.drop('Class', axis=1)
  y = balanced_dataset['Class']
  X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3,
      random_state=42) # 70% training and 30% test
  svm_clf = svm.SVC().fit(X_train, y_train)
  pr=precision_score(y_test,svm_clf.predict(X_test))
  r=recall_score(y_test,svm_clf.predict(X_test))
  f=f1_score(y_test,svm_clf.predict(X_test))
  g=geometric_mean_score(y_test,svm_clf.predict(X_test))
  auc=roc_auc_score(y_test,svm_clf.predict(X_test))
  m=matthews_corrcoef(y_test,svm_clf.predict(X_test))
  pr_score.append(pr)
  re_score.append(r)
  f1.append(f)
  G_mean.append(g)
  AUC.append(auc)
  mcc.append(m)

print("Mean precision score",mean(pr_score))
print("Mean recall score",mean(re_score))
print("Mean f1 score",mean(f1))
print("Mean G-mean score",mean(G_mean))
print("Mean AUC score",mean(AUC))
print("Mean MCC score",mean(mcc))

#Create a svm Classifier
pr_score=[]
re_score=[]
f1=[]
```

```
G_mean=[]
AUC=[]
mcc=[]
index=448
for i,j in zip(range(0,627),range(1,628)):
  non_fraud_cases_indexed=non_fraud_cases.iloc[i*index:j*index]
  balanced_dataset=pd.concat([non_fraud_cases_indexed,fraud_cases])
  shuffle(balanced_dataset)
  X = balanced_dataset.drop('Class', axis=1)
  y = balanced_dataset['Class']
  X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3,
      random_state=42) # 70% training and 30% test
  svm_clf = svm.SVC().fit(X_train, y_train)
  pr=precision_score(y_test,svm_clf.predict(X_test))
  r=recall_score(y_test,svm_clf.predict(X_test))
  f=f1_score(y_test,svm_clf.predict(X_test))
  g=geometric_mean_score(y_test,svm_clf.predict(X_test))
  auc=roc_auc_score(y_test,svm_clf.predict(X_test))
  m=matthews_corrcoef(y_test,svm_clf.predict(X_test))
  pr_score.append(pr)
  re_score.append(r)
  f1.append(f)
  G_mean.append(g)
  AUC.append(auc)
  mcc.append(m)

print("Mean precision score",mean(pr_score))
print("Mean recall score",mean(re_score))
print("Mean f1 score",mean(f1))
print("Mean G-mean score",mean(G_mean))
print("Mean AUC score",mean(AUC))
print("Mean MCC score",mean(mcc))

"""**k-Nearst Neighbour**"""

from sklearn.neighbors import KNeighborsClassifier
pr_score=[]
re_score=[]
f1=[]
G_mean=[]
AUC=[]
mcc=[]
index=448
for i,j in zip(range(0,627),range(1,628)):
  non_fraud_cases_indexed=non_fraud_cases.iloc[i*index:j*index]
  balanced_dataset=pd.concat([non_fraud_cases_indexed,fraud_cases])
  shuffle(balanced_dataset)
  X = balanced_dataset.drop('Class', axis=1)
```

```
    y = balanced_dataset['Class']
    X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3,
        random_state=42) # 70% training and 30% test
    knn_clf =  KNeighborsClassifier().fit(X_train, y_train)
    pr=precision_score(y_test,knn_clf .predict(X_test))
    r=recall_score(y_test,knn_clf .predict(X_test))
    f=f1_score(y_test,knn_clf .predict(X_test))
    g=geometric_mean_score(y_test,knn_clf .predict(X_test))
    auc=roc_auc_score(y_test,knn_clf .predict(X_test))
    m=matthews_corrcoef(y_test,knn_clf .predict(X_test))
    pr_score.append(pr)
    re_score.append(r)
    f1.append(f)
    G_mean.append(g)
    AUC.append(auc)
    mcc.append(m)

print("Mean precision score",mean(pr_score))
print("Mean recall score",mean(re_score))
print("Mean f1 score",mean(f1))
print("Mean G-mean score",mean(G_mean))
print("Mean AUC score",mean(AUC))
print("Mean MCC score",mean(mcc))

"""**Logistic regression**"""

from sklearn.linear_model import LogisticRegression
pr_score=[]
re_score=[]
f1=[]
G_mean=[]
AUC=[]
mcc=[]
index=448
for i,j in zip(range(0,627),range(1,628)):
    non_fraud_cases_indexed=non_fraud_cases.iloc[i*index:j*index]
    balanced_dataset=pd.concat([non_fraud_cases_indexed,fraud_cases])
    shuffle(balanced_dataset)
    X = balanced_dataset.drop('Class', axis=1)
    y = balanced_dataset['Class']
    X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3,
        random_state=42) # 70% training and 30% test
    lr_clf = LogisticRegression().fit(X_train, y_train)
    pr=precision_score(y_test,lr_clf.predict(X_test))
    r=recall_score(y_test,lr_clf.predict(X_test))
    f=f1_score(y_test,lr_clf.predict(X_test))
    g=geometric_mean_score(y_test,lr_clf.predict(X_test))
    auc=roc_auc_score(y_test,lr_clf.predict(X_test))
```

```
    m=matthews_corrcoef(y_test,lr_clf.predict(X_test))
    pr_score.append(pr)
    re_score.append(r)
    f1.append(f)
    G_mean.append(g)
    AUC.append(auc)
    mcc.append(m)

print("Mean precision score",mean(pr_score))
print("Mean recall score",mean(re_score))
print("Mean f1 score",mean(f1))
print("Mean G-mean score",mean(G_mean))
print("Mean AUC score",mean(AUC))
print("Mean MCC score",mean(mcc))

"""**Decision Tree**"""

from sklearn.tree import DecisionTreeClassifier
pr_score=[]
re_score=[]
f1=[]
G_mean=[]
AUC=[]
mcc=[]
index=448
for i,j in zip(range(0,627),range(1,628)):
    non_fraud_cases_indexed=non_fraud_cases.iloc[i*index:j*index]
    balanced_dataset=pd.concat([non_fraud_cases_indexed,fraud_cases])
    shuffle(balanced_dataset)
    X = balanced_dataset.drop('Class', axis=1)
    y = balanced_dataset['Class']
    X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3,
        random_state=42) # 70% training and 30% test
    dt_clf = DecisionTreeClassifier().fit(X_train, y_train)
    pr=precision_score(y_test,dt_clf.predict(X_test))
    r=recall_score(y_test,dt_clf.predict(X_test))
    f=f1_score(y_test,dt_clf.predict(X_test))
    g=geometric_mean_score(y_test,dt_clf.predict(X_test))
    auc=roc_auc_score(y_test,dt_clf.predict(X_test))
    m=matthews_corrcoef(y_test,dt_clf.predict(X_test))
    pr_score.append(pr)
    re_score.append(r)
    f1.append(f)
    G_mean.append(g)
    AUC.append(auc)
    mcc.append(m)

print("Mean precision score",mean(pr_score))
```

```
print("Mean recall score",mean(re_score))
print("Mean f1 score",mean(f1))
print("Mean G-mean score",mean(G_mean))
print("Mean AUC score",mean(AUC))
print("Mean MCC score",mean(mcc))

"""**Multilayer Perceptron**"""

from sklearn.neural_network import MLPClassifier
pr_score=[]
re_score=[]
f1=[]
G_mean=[]
AUC=[]
mcc=[]
index=448
for i,j in zip(range(0,627),range(1,628)):
  non_fraud_cases_indexed=non_fraud_cases.iloc[i*index:j*index]
  balanced_dataset=pd.concat([non_fraud_cases_indexed,fraud_cases])
  shuffle(balanced_dataset)
  X = balanced_dataset.drop('Class', axis=1)
  y = balanced_dataset['Class']
  X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3,
      random_state=42, stratify=y) # 70% training and 30% test
  mlp_clf = MLPClassifier().fit(X_train, y_train)
  pr=precision_score(y_test,mlp_clf.predict(X_test))
  r=recall_score(y_test,mlp_clf.predict(X_test))
  f=f1_score(y_test,mlp_clf.predict(X_test))
  g=geometric_mean_score(y_test,mlp_clf.predict(X_test))
  auc=roc_auc_score(y_test,mlp_clf.predict(X_test))
  m=matthews_corrcoef(y_test,mlp_clf.predict(X_test))
  pr_score.append(pr)
  re_score.append(r)
  f1.append(f)
  G_mean.append(g)
  AUC.append(auc)
  mcc.append(m)

print("Mean precision score",mean(pr_score))
print("Mean recall score",mean(re_score))
print("Mean f1 score",mean(f1))
print("Mean G-mean score",mean(G_mean))
print("Mean AUC score",mean(AUC))
print("Mean MCC score",mean(mcc))

"""**Artifical Neural Network**"""

X=creditdata_normalized.drop('Class',axis=1)
```

```
y=creditdata_normalized['Class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
    stratify=y, random_state=42)

from keras.wrappers.scikit_learn import KerasClassifier
from keras.models import Sequential
import tensorflow as tf
from tensorflow import keras
from keras.layers import Dense, Dropout, Activation

def make_model(layers, activation):
  model= Sequential()
  for i,nodes in enumerate(layers):
    if i==0:
      model.add(Dense(nodes, input_dim = X_train.shape[1]))
      model.add(Activation(activation))
    else:
      model.add(Dense(nodes))
      model.add(Activation(activation))
  model.add(Dense(1, activation='sigmoid'))
  model.compile(optimizer='nadam', loss='binary_crossentropy', metrics=['
      accuracy'])
  return model

model=KerasClassifier(build_fn=make_model,verbose=0)

loss=[]
optimizer=[]

layers=[[20],[40,20],[45, 30, 25]]
activations=['sigmoid','relu']
param_grid={'layers':layers,'activation':activations,'batch_size':[30,60],'
    epochs':[15]}
ann_grid=GridSearchCV(estimator=model, param_grid=param_grid)
ann_grid.fit(X_train,y_train)

print(ann_grid.best_params_)

model = Sequential([
    Dense(units=20, input_dim = X_train.shape[1], activation='relu'),
    Dense(units=45,activation='relu'),
    Dropout(0.2),
    Dense(units=30,activation='relu'),
    Dense(units=25,activation='relu'),
    Dense(1, activation='sigmoid')
])
```

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=[tf.keras.
    metrics.Recall()])
model.fit(X_train, y_train,batch_size=30, epochs=15)

y_pred = model.predict(X_test)
print(classification_report(y_test,y_pred.round()))

y_train_pred = model.predict(X_train)
print(classification_report(y_train,y_train_pred.round()))

#K-Fold cross validation
kf = StratifiedKFold(n_splits=10,shuffle=True,random_state=42)
auc_score =[]
recall_scor=[]
precision_scor=[]
geo_score=[]
f1=[]
mcc=[]
i=1
for train_index,test_index in kf.split(X,y):
    #print('{} of KFold {}'.format(i,kf.n_splits))
    xtrain,xtest = X.iloc[train_index],X.iloc[test_index]
    ytrain,ytest = y.iloc[train_index],y.iloc[test_index]

    #model
    model.fit(xtrain, ytrain,batch_size=30, epochs=15)
    score = roc_auc_score(ytest,model.predict(xtest).round())
    score1 = recall_score(ytest,model.predict(xtest).round())
    score2 = precision_score(ytest,model.predict(xtest).round())
    score3=geometric_mean_score(ytest,model.predict(xtest).round())
    score4=f1_score(ytest,model.predict(xtest).round())
    score5=matthews_corrcoef(ytest,model.predict(xtest).round())
    #
        ##########################################################################

    auc_score.append(score)
    recall_scor.append(score1)
    precision_scor.append(score2)
    geo_score.append(score3)
    f1.append(score4)
    mcc.append(score5)

    i+=1

print('ANN auc mean score:',mean(auc_score))
print('ANN recall mean score:',mean(recall_scor))
print('ANN  precision mean score:',mean(precision_scor))
print('ANN  geometric mean score:',mean(geo_score))
```

```
print('ANN f1 measure score:',mean(f1))
print('ANN  MCC mean score:',mean(mcc))

#Test dataset from the model
auc=roc_auc_score(y_test, y_pred)
prec = precision_score(y_test, y_pred.round())
rec = recall_score(y_test, y_pred.round())
f1 = f1_score(y_test, y_pred.round())
gmean=geometric_mean_score(y_test,y_pred.round())
mcc=matthews_corrcoef(y_test,y_pred.round())
print("\tprecision:%0.4f"%prec,"\trecall:%0.4f"%rec,"\tF1-score:%0.4f"%f1,"\
    tgeometric mean:%0.4f"%gmean,"\troc auc:%0.4f"%auc,"\tMCC:%0.4f"%mcc)

"""**Feature Selection**"""

from sklearn.decomposition import PCA
X = creditdata_normalized.drop('Class', axis=1)
y = creditdata_normalized['Class']
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3,
    stratify=y, random_state=42) #70% training and 30% test
print(X_train.shape)
print(X_test.shape)

"""**Feature Selection Using Decision Tree**"""

from sklearn.pipeline import Pipeline
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.feature_selection import RFE

model=DecisionTreeClassifier(criterion='entropy',max_depth=3,min_samples_leaf
    =1,min_samples_split=2)
rfe = RFE(estimator=DecisionTreeClassifier(criterion='entropy',max_depth=3,
    min_samples_leaf=1,min_samples_split=2))
pipe = Pipeline([('Feature Selection', rfe), ('Model', model)])
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=5, random_state=146)
n_scores = cross_val_score(pipe, X_train, y_train, scoring='recall', cv=cv,
    n_jobs=-1)
np.mean(n_scores)

pipe.fit(X_train, y_train)
print("Num Features: %d" % rfe.n_features_)
print("Selected Features: %s" % rfe.support_)
print("Feature Ranking: %s" % rfe.ranking_)
```

credit_card_dataset_analysis.py