University of Essex

Department of Mathematical Sciences

MA981 DISSERTATION

"Guardians of Transactions: Unleashing Machine Learning for Fraud-Free Finance"

PRAJWAL MARKAL PUTTASWAMY

REG NUM:2213173

Supervisor: **Xinan Yang**

July 1, 2024
Colchester

Abstract

In our swiftly evolving digital realm, the relentless rise of credit card fraud presents an urgent call for resilient security measures. This research delves deep into the efficacy of machine learning models specifically tailored to combat fraudulent activities within the domain of credit card transactions. Beyond assessing their effectiveness, this study intricately examines the adaptability of these models in countering the ever-shifting tactics employed by fraudsters. By scrutinizing the dynamic interplay between technological advancements, evolving fraud methodologies, and the efficacy of detection models, this research aims not only to contribute valuable insights to academia but also to offer actionable strategies for financial institutions and cybersecurity entities. The study's findings extend beyond theoretical constructs, carrying substantial implications for industry practices, guiding the fortification of fraud detection frameworks, and shaping the discourse around bolstering transactional security. Ultimately, this research acts as a catalyst for future investigations, paving the way for the integration of cutting-edge technologies and innovative methodologies to stay ahead in the relentless battle against credit card fraud in the digital age.

Keywords: Credit Card Fraud, Machine Learning, Fraud Identification, Transaction Security, Financial Technology, Data Analytics, Cybersecurity, Fraud Prevention

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

In the contemporary landscape of financial transactions, the widespread adoption of digital payment methods has revolutionized the way we conduct business [7]. While this shift brings unparalleled convenience, it has concurrently ushered in a burgeoning challenge credit card fraud [4]. The unauthorized and deceptive use of credit card information has emerged as a formidable threat within the financial sector, necessitating innovative solutions to fortify security measures.

As financial transactions increasingly migrate to online platforms, traditional methods of fraud detection, often reliant on rule-based systems, face limitations in adapting to the dynamic nature of fraudulent activities [3]. Consequently, the need for sophisticated and adaptive approaches to counter credit card fraud has never been more pressing.

Credit card fraud detection, as a pivotal domain within cybersecurity and financial technology, undertakes the formidable task of developing robust mechanisms capable of discerning between legitimate and fraudulent transactions [1]. In this endeavor, the integration of machine learning technologies has proven instrumental. Machine learning algorithms, such as logistic regression, decision trees, and random forests, offer the ability to process vast datasets in real-time, identifying subtle anomalies that might evade conventional detection methods [8].

Amidst the rapid evolution of cyber threats, the implementation of advanced machine learning models emerges as a beacon of hope in the ongoing struggle against credit card fraud [5]. The intricate patterns and anomalies inherent in fraudulent transactions necessitate a paradigm shift in detection methodologies. By leveraging the capabilities of machine learning, this project seeks to harness the prowess of algorithms in discerning complex relationships within data, adapting to emerging fraud tactics, and providing a proactive defense against illicit activities [10].

As financial institutions navigate the intricate landscape of digital transactions, the imperative to strike a delicate balance between security and user experience becomes more pronounced [2]. Traditional fraud detection methods often grapple with the challenge of distinguishing genuine transactions from fraudulent ones without introducing friction into the user experience. Machine learning models, with their capacity for nuanced analysis and real-time adaptation, offer a potential resolution to this conundrum [6]. The ensuing exploration aims to unravel the efficiency of these models in achieving not only robust security but also a seamless and user-friendly transaction environment.

This project seeks to contribute to the ongoing efforts in credit card fraud detection by exploring and comparing the effectiveness of various machine learning models [9]. The research delves into the intricacies of these models, examining their performance, strengths, and limitations. By doing so, it aims to provide valuable insights that can enhance the accuracy and efficiency of fraud detection processes.

The utilization of machine learning models in credit card fraud detection holds the promise of not only bolstering security measures but also of adapting to the ever-evolving tactics employed by fraudsters [10]. As digital transactions continue to shape the financial landscape, understanding the nuances of machine learning models becomes paramount in ensuring the resilience and trustworthiness of digital payment systems.

Ultimately, this research aspires to empower financial institutions, security professionals, and stakeholders with the knowledge and tools necessary to stay ahead of the curve in the perpetual battle against credit card fraud [7]. Through a comprehensive exploration of machine learning techniques, the project aims to contribute meaningfully to the field, fostering a safer and more secure environment for digital financial transactions.

CHAPTER 2

Literature review

2.1 Introduction

In an era marked by the widespread adoption of digital payment methods, the threat of credit card fraud has grown to unprecedented proportions[15]. Credit card fraud, encompassing a range of deceptive activities involving unauthorized or fraudulent use of credit card information, has emerged as a significant concern in the financial and security landscape[17]. This literature review seeks to delve into the multifaceted world of credit card fraud detection, aiming to comprehensively examine the current state of methodologies, technologies, and research efforts dedicated to mitigating this pervasive threat[29].

2.2 Background and Significance

Credit card fraud, a consequence of our increasingly digitalized economy, presents a profound challenge to individuals, businesses, and financial institutions. It is no longer confined to a few isolated cases but has evolved into a global predicament, posing substantial economic and security implications[44].

2.2.1 Economic Implications

Credit card fraud has substantial economic consequences. For individuals, it can result in financial losses, damaged credit scores, and emotional distress[62]. On a broader scale, businesses experience financial losses due to fraudulent transactions and the cost of implementing security measures. These losses ultimately contribute to higher consumer costs, affecting economies on a national and international scale[52].

2.2.2 Security Implications

Beyond economic impacts, credit card fraud can have far-reaching security implications[56]. When a cardholder's information is compromised, their personal and financial security is jeopardized. Moreover, there are broader societal consequences, as a surge in fraud cases can undermine public trust in the financial system[28].

2.3 Emphasis on Effective Fraud Detection Methods

The importance of addressing credit card fraud detection cannot be overstated. Effective fraud detection methods are paramount in preserving the integrity of digital payment systems and financial security[24]. These methods serve as the first line of defense against malicious actors seeking to exploit vulnerabilities in the payment infrastructure[21].

2.4 Historical Overview of Credit Card Fraud

Credit card fraud has a rich and evolving history that has closely paralleled the advancements in technology and changes in payment methods[58]. Understanding this history is crucial for comprehending the context in which modern credit card fraud detection methods have developed.

2.4.1 Emergence of Credit Card Fraud

Credit card fraud has been present since the inception of credit card systems in the mid-20th century. In its early days, it primarily involved stolen or counterfeit cards and forged signatures. The simplicity of these early transactions made them relatively easy to perpetrate and detect[61].

2.4.2 Evolution of Traditional Fraud Detection Methods

As credit card usage expanded, so did the methods used to detect and prevent fraud[57]. The early approaches to fraud detection relied heavily on manual checks and rudimentary rule-based systems. These systems compared transactions against predefined rules and patterns[22]. For instance, if a transaction occurred in a location far from the cardholder’s residence, it might trigger suspicion.

2.4.3 Challenges and Limitations

Over time, it became evident that traditional fraud detection methods had limitations. They struggled to keep pace with the increasing complexity and sophistication of fraudulent activities[46]. Criminals found new ways to exploit vulnerabilities, such as using stolen card information for online purchases, where the physical presence of a card and a signature were not required[30].

2.5 Traditional Methods of Fraud Detection

Legacy methods of fraud detection, particularly rule-based systems, have played a pivotal role in the early stages of safeguarding credit card transactions[50]. These systems rely on predefined rules and patterns to identify potentially fraudulent activities. However, they come with a set of limitations that have become increasingly apparent in the face of evolving fraud tactics.

2.5.1 Rule-Based Systems

Rule-based systems in credit card fraud detection entail creating a set of guidelines and conditions that transactions must meet to be considered legitimate. For instance, if a transaction occurs in a foreign country shortly after another transaction in the cardholder’s home country, it might raise suspicion. These systems can be effective for identifying straightforward anomalies[19].

2.5.2 Limitations and Vulnerabilities

Despite their utility, rule-based systems have clear limitations. They often generate a high rate of false positives, leading to inconveniences for cardholders due to frequent transaction denials[61]. These systems struggle to adapt to the ever-changing strategies of fraudsters, particularly in the digital age.

2.6 Machine Learning in Credit Card Fraud Detection

Machine learning represents a significant advancement in credit card fraud detection, offering the promise of improved accuracy and adaptability. This section introduces the role of machine learning in this context and discusses commonly applied algorithms[12].

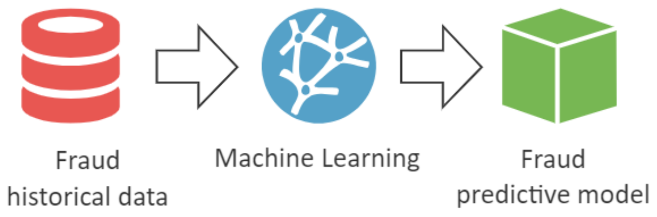

Figure 2.1: Machine Learning in Credit Card Fraud Detection
Source: neuraldesigner.com

2.6.1 Role of Machine Learning

Machine learning leverages the power of algorithms to analyze vast datasets and identify patterns that may elude traditional rule-based systems[63]. It excels at recognizing complex relationships between variables and adjusting its models based on new data. This adaptability is particularly valuable in the face of rapidly evolving fraud tactics.

2.6.2 Common Machine Learning Algorithms

In credit card fraud detection, several machine learning algorithms have demonstrated their effectiveness[20].. These include logistic regression, decision trees, and neural networks. Logistic regression is well-suited for binary classification tasks, such as distinguishing between legitimate and fraudulent transactions. Decision trees provide transparency and interpretability, while neural networks offer the capability to learn intricate, nonlinear patterns[48].

2.7 Challenges in Credit Card Fraud Detection

While machine learning holds great promise, it also faces several challenges in the context of credit card fraud detection. These challenges have a direct impact on the effectiveness of fraud detection systems[53].

2.7.1 Imbalanced Datasets

The prevalence of legitimate transactions far outweighs fraudulent ones, resulting in imbalanced datasets. Imbalanced data can skew models toward classifying most transactions as legitimate, making it difficult to detect the minority of fraudulent activities[32].

2.7.2 Concept Drift

Fraudsters continually adapt their strategies. This concept drift presents an ongoing challenge for fraud detection systems, which must continuously evolve to recognize new patterns and tactics.

2.7.3 Adversarial Attacks

In a cat-and-mouse game, sophisticated fraudsters may launch adversarial attacks against machine learning models, attempting to deceive or mislead them. Such attacks add a layer of complexity and necessitate robust model defenses[39].

2.8 Recent Advancements in Credit Card Fraud Detection

In recent years, the landscape of credit card fraud detection has witnessed substantial advancements, driven by both research and technology. This section explores these developments and highlights the application of cutting-edge techniques, such as deep learning, anomaly detection, and ensemble methods[59].

2.8.1 Research and Technological Advancements

Recent research efforts have propelled the field of credit card fraud detection to new heights. These advancements are primarily fueled by the growing availability of large and diverse datasets, as well as the increasing computational power. These key drivers have led to more accurate and efficient fraud detection methods[60].

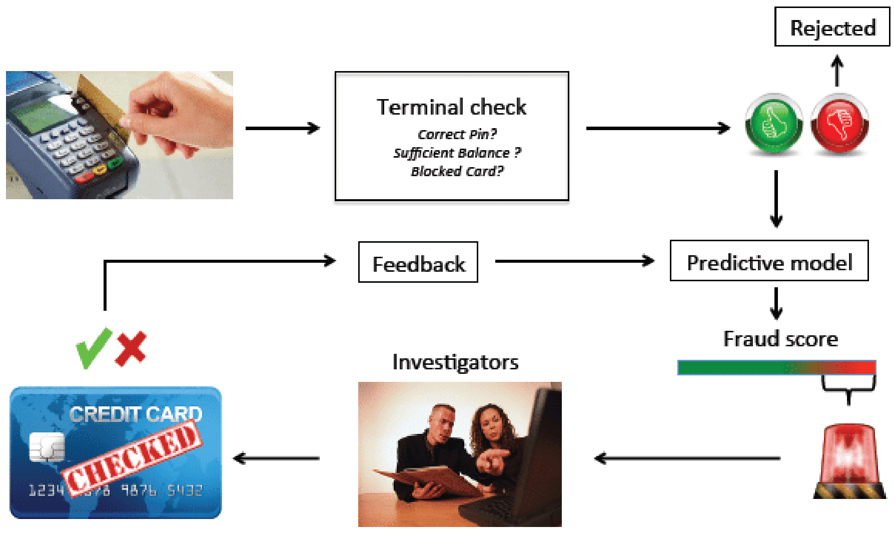

Figure 2.2: Recent Advancements in Credit Card Fraud Detection

Source: researchgate.net

Deep Learning

Deep learning techniques, particularly neural networks, have shown remarkable promise in identifying intricate patterns within credit card transaction data. Their ability to process vast amounts of information and uncover hidden relationships has significantly improved fraud detection accuracy[31].

Anomaly Detection

Anomaly detection approaches have become crucial in identifying rare and previously unseen fraud patterns. These methods focus on detecting deviations from the norm, making them exceptionally useful in identifying novel fraud tactics.

Ensemble Methods

Ensemble methods, which combine the predictions of multiple models, have gained popularity for their ability to enhance overall accuracy and robustness[44]. Techniques like random forests and gradient boosting have proven highly effective in minimizing false positives while capturing fraudulent activities.

State-of-the-Art Models and Algorithms

Several state-of-the-art models and algorithms have emerged as leaders in the credit card fraud detection domain. For instance, the use of Convolutional Neural Networks (CNNs) for image-based card recognition and recurrent neural networks (RNNs) for sequence modeling have demonstrated excellent results. Additionally, XGBoost, a gradient boosting algorithm, has been widely adopted for its ability to handle imbalanced datasets effectively.

2.8.2 Evaluating Fraud Detection Models

The effectiveness of credit card fraud detection systems relies on comprehensive evaluation metrics that assess their performance.

Precision, Recall, F1-Score, and ROC-AUC

Precision measures the proportion of true positives among all positive predictions, while recall determines the fraction of actual positives correctly identified. The F1-score balances precision and recall. ROC-AUC quantifies the model's ability to distinguish between fraudulent and legitimate transactions.

These metrics serve as critical tools in quantifying the success of fraud detection models and allow for fine-tuning their parameters to achieve a desired balance between precision and recall.

2.9 Regulatory Compliance and Industry Standards

Regulatory frameworks and industry standards play an essential role in shaping credit card fraud detection practices.

2.9.1 Regulatory Framework

Authorities and governing bodies have established regulations that financial institutions must adhere to, setting the groundwork for secure transaction processes. Compliance with these regulations is not only a legal requirement but also a fundamental aspect of maintaining trust within the financial sector.

2.9.2 Industry Standards

Industry-specific standards, like the Payment Card Industry Data Security Standard (PCI DSS), provide guidelines for securing payment card transactions. These standards are pivotal in shaping the development and deployment of fraud detection systems.

2.10 Case Studies and Practical Implementations

Real-world case studies offer tangible evidence of successful fraud detection systems and their impact on reducing fraud rates. These case studies demonstrate how financial institutions have leveraged advanced technology to mitigate fraud risks.

2.10.1 Impact

Implementations have consistently resulted in a significant reduction in fraud rates, leading to cost savings, customer trust, and an enhanced reputation for financial institutions.

2.11 Ethical and Privacy Considerations

As credit card fraud detection becomes more sophisticated, ethical and privacy concerns come to the forefront.

2.11.1 Data Privacy

The use of vast datasets in machine learning models raises concerns about data privacy. Safeguarding cardholder information is of utmost importance, and institutions must ensure that sensitive data is handled responsibly.

2.11.2 Ethical Implications

The use of machine learning in fraud detection also presents ethical questions, such as potential biases in algorithms, the balance between security and privacy, and surveillance concerns.

2.12 Future Directions and Emerging Trends

The future of credit card fraud detection is poised to bring about transformative changes.

2.12.1 Emerging Technologies

The future of credit card fraud detection is on the cusp of transformative change, largely driven by emerging technologies. Blockchain, renowned for its secure and decentralized nature, stands as a promising solution for fortifying transaction security. Its tamper-proof ledger system and cryptographic principles offer a robust defense against fraudulent activities, ensuring secure and transparent transactions. Explainable AI serves as another pivotal technology, aiming to demystify the opaque workings of complex machine learning models. By enhancing transparency in decision-making processes, it not only fosters trust but also enables the identification and rectification of biases, bolstering the reliability of fraud detection systems. Biometrics, offering multi-factor authentication through unique biological traits, emerges as a stronghold against unauthorized access. Integrating fingerprint scans, facial recognition, or iris scans augments security measures, thwarting fraudulent attempts by adding an additional layer of identity verification.

2.12.2 Research Gaps

Within the realm of credit card fraud detection, persistent research gaps beckon for innovative solutions. Chief among these is the interpretability of complex machine learning models. As models grow in complexity, ensuring interpretability becomes crucial for stakeholders to comprehend their decisions. Bridging this gap demands the development of robust techniques facilitating the comprehensive explanation of intricate model outputs, aiding in bias identification and enhancing model trustworthiness. Another critical area pertains to the evolving landscape of fraudulent tactics. Ongoing research endeavors must pivot toward staying ahead of these evolving strategies. Adaptive fraud detection mechanisms leveraging real-time analytics, anomaly detection, and AI-driven models are imperative to swiftly identify and counter emerging fraud patterns, ensuring the resilience of fraud detection systems

CHAPTER 3

Methodology

3.1 Logistic Regression

Model Overview: Logistic regression is a statistical model used for binary classification tasks, estimating the probability of a binary outcome based on one or multiple independent variables[36]. It operates on the principle of fitting a logistic function to model the relationship between the dependent variable and predictors[45].

Probability Estimation: The logistic function transforms the linear combination of predictors into probabilities bounded between 0 and 1. This function enables the estimation of the likelihood of an event occurring, facilitating classification decisions[11].

Logistic Function

The logistic function, or sigmoid function, is at the core of logistic regression, mathematically expressed as:

$$P(Y = 1|X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X)}}$$

Where:

- $P(Y = 1|X)$ denotes the probability of the dependent variable being 1 given the predictors.
- e represents the base of the natural logarithm.
- β_0 and $\beta_1 X$ correspond to the intercept and the linear combination of coefficients and independent variables, respectively.

Model Fitting

Coefficient Estimation: Logistic regression determines optimal coefficients (β) by minimizing a chosen cost function, typically the log-likelihood or cross-entropy loss[35].

Optimization Techniques: Iterative optimization methods like gradient descent iteratively update coefficients to reduce the discrepancy between predicted probabilities and actual outcomes, converging towards optimal values[40].

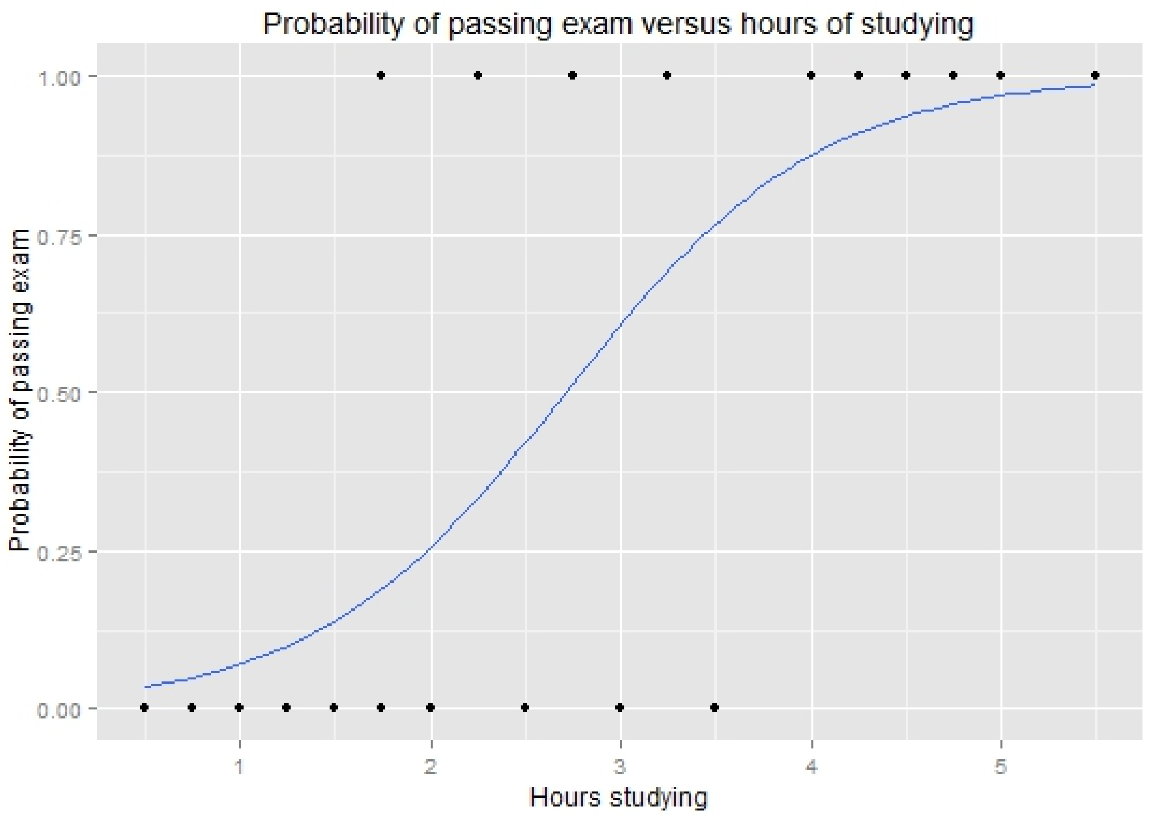

Figure 3.1: Logistic Regression

Source: Wikimedia Commons

The diagram illustrates the logistic regression curve, depicting the relationship between the independent variable and the predicted probability of the dependent variable belonging to a specific class.

Logistic regression finds application across various domains, including medical diagnosis, risk assessment, and financial forecasting, owing to its interpretability and effectiveness in binary classification scenarios. Its foundation in probability estimation makes it a fundamental tool in statistical modeling and machine learning.

3.2 Decision Trees

A decision tree is a hierarchical predictive model used in machine learning for classification and regression analysis[55]. It represents a flowchart-like structure where each internal node denotes a feature or attribute, each branch signifies a decision rule, and each leaf node provides the outcome or prediction. Through recursive partitioning, the tree algorithm selects the most informative features at each node, optimizing split criteria such as Gini impurity or information gain[35]. This process iterates until a predefined stopping criterion is met, ensuring the creation of a predictive tree model that offers interpretable decision paths for classification or regression tasks.

Model Construction: Decision trees organize data into a hierarchical tree-like structure by recursively partitioning it based on input features. Each node represents a decision based on a specific feature, leading to subsequent nodes or leaves that denote final outcomes[47].

Classification or Regression: For classification, decision trees classify instances into different classes at the leaves of the tree. For regression, the tree predicts continuous values for each instance[16].

Splitting Criteria

The process involves selecting the best feature to split the data at each node, aiming to maximize information gain or minimize impurity.

Gini Impurity:

$$Gini = 1 - \sum_{i=1}^{n} p_i^2$$

Where p_i is the probability of an object being classified to a particular class i.

Information Gain:

$$\text{Entropy} = -\sum_{i=1}^{n} p_i \log_2(p_i)$$

$$\text{Information Gain} = \text{Entropy before split} - \text{Weighted Entropy after split}$$

Stopping Criteria

The tree-building process continues until a stopping criterion is met, such as maximum depth or minimum samples per leaf.

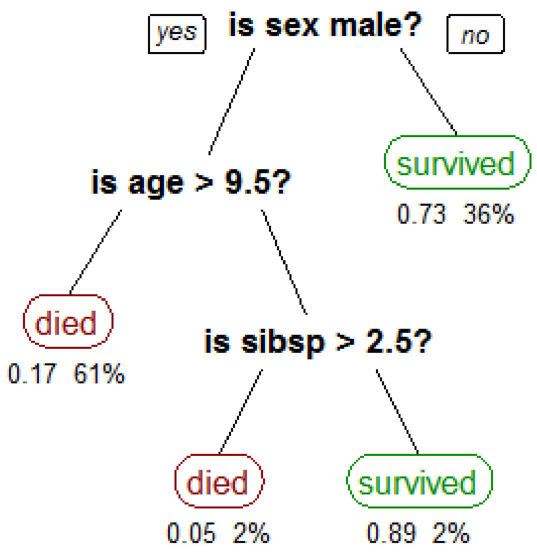

Figure 3.2: Decision Tree

The diagram illustrates a decision tree example for predicting survival on the Titanic based on features like age, gender, and ticket class.

Decision Trees are intuitive and interpretable models used in various domains for classification and regression tasks. ter command

3.3 Random Forest Classifiers

Ensemble Learning Technique: Random Forest aggregates predictions from an ensemble of decision trees to improve predictive accuracy and mitigate overfitting[18]. This collective approach aims to generate a more robust and accurate prediction than individual trees.

Construction of Decision Trees

1. **Bootstrap Sampling:** Random Forest constructs multiple decision trees by repeatedly sampling the dataset with replacement, creating diverse subsets for training each tree[42].

2. **Random Feature Subset Selection:** At each node of every tree, only a random

subset of features is considered for splitting[51]. This randomness aids in building diverse trees by reducing feature correlation.

Prediction Aggregation

- **Output Combination:** For regression tasks, predictions from all trees are averaged to derive the final prediction[27]. For classification tasks, the mode among all tree predictions is taken as the ensemble prediction.

Gini Impurity (For Classification):

$$Gini = 1 - \sum_{i=1}^{n} p_i^2$$

Where p_i is the probability of an object being classified to a particular class i.

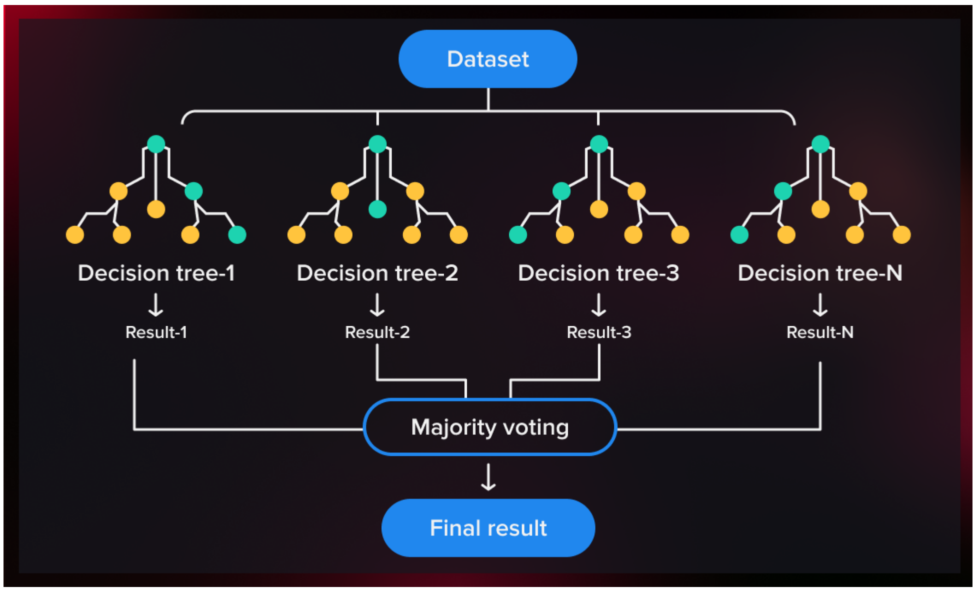

Figure 3.3: Random Forest

Source: serokell.io

The diagram illustrates the ensemble nature of Random Forests, showcasing the construction of multiple decision trees using different subsets of data and features. Predictions from each tree are aggregated to form the final ensemble prediction[42].

Random Forests are versatile and widely used due to their capability to handle high-dimensional data, capture complex relationships, and mitigate overfitting. The ensemble strategy combines the strengths of multiple trees, resulting in robust and accurate predictions.

3.4 Support Vector Machines (SVMs)

Support Vector Machine (SVM) is a supervised machine learning algorithm used for classification and regression tasks. It works by finding the optimal hyperplane that best separates different classes in the input space. SVM aims to maximize the margin between classes, which is the distance between the hyperplane and the nearest data points (support vectors) from each class[37]. This algorithm is effective for both linearly separable and non-linearly separable data, achieved by transforming data into higher-dimensional spaces using kernel functions. SVMs are widely used due to their ability to handle complex data distributions and their robustness in generalizing to unseen data[25].

Classification with Maximum Margin

1. **Linear Separation:** SVM identifies a hyperplane that linearly separates classes in the input space.
2. **Maximizing Margin:** It selects the hyperplane that maximizes the margin, which is the distance between the hyperplane and the nearest data points (support vectors) from each class[38].

Kernel Trick for Non-linear Data

1. **Kernel Functions:** SVM employs kernel functions to map input data into a higher-dimensional space, enabling non-linear separation.
2. **Non-linear Separation:** This transformation allows SVM to perform non-linear separation in the higher-dimensional space, even if the original data was not linearly separable[41].

Equation of the Hyperplane (for linearly separable case):

$$\mathbf{w} \cdot \mathbf{x} + b = 0$$

Where $\mathbf{w}$ is the weight vector, $\mathbf{x}$ is the input feature vector, and b is the bias term.

Margin Calculation:

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

Where ||w|| is the Euclidean norm of the weight vector.

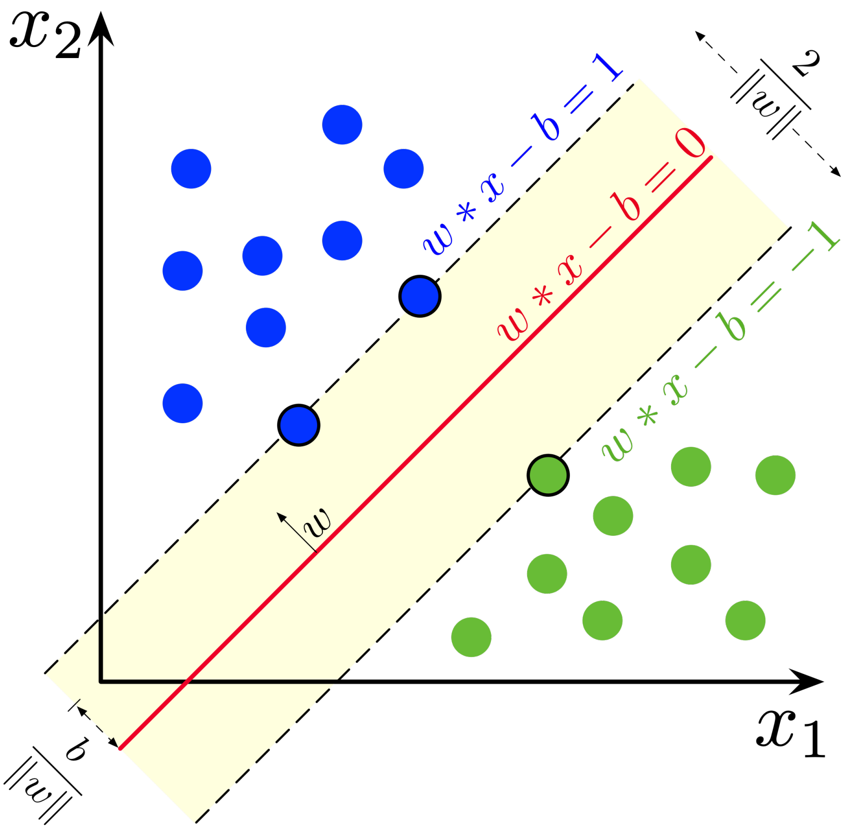

Figure 3.4: SVM Classifiers

Source: wikimedia.org

The figure illustrates a linearly separable scenario where an optimal hyperplane separates two classes with the maximum margin between them. Support vectors are the closest data points to the hyperplane[33].

SVMs are versatile algorithms effective in various domains due to their ability to handle both linear and non-linear separation, aiding robust generalization to unseen data.

3.5 K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a supervised machine learning algorithm used for classification and regression. It determines the class of a new data point by comparing it to its 'k' closest neighbors in the feature space. The predicted class for the new data point is determined by majority voting among its nearest neighbors[14].

Classification Based on Proximity

1. **Data Storage:** KNN stores all available data points with their respective class labels.

2. **Prediction by Similarity:** To classify a new data point, KNN measures the similarity (e.g., using Euclidean distance) between the new point and all stored points[43].

3. **Majority Voting:** It identifies the ’k’ closest data points (nearest neighbors) to the new point based on the similarity measure. The most frequent class among these ’k’ neighbors determines the predicted class for the new data point.

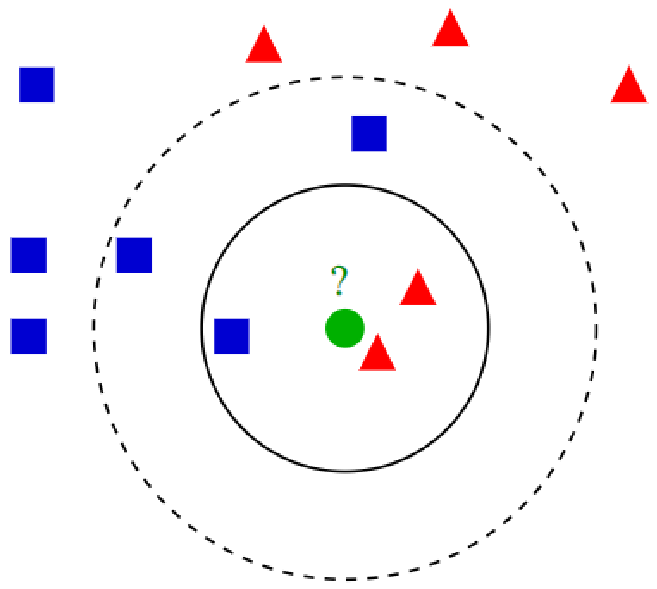

Figure 3.5: KNN Classifiers

Source: wikimedia.org

The diagram illustrates how KNN classifies a new data point based on its ’k’ nearest neighbors in the feature space. The majority class among these neighbors determines the assigned class for the new data point[26].

KNN is intuitive and straightforward but sensitive to the choice of ’k,’ the distance metric, and data dimensionality.

Euclidean Distance (for Example):

$$\text{Euclidean Distance} = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

Where p_i and q_i are feature values of two points across each dimension.

3.6 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a statistical method used for reducing the dimensionality of high-dimensional data while preserving most of its important information. It achieves this by transforming the original variables into a new set of uncorrelated variables called principal components[34].

Data Standardization

PCA begins by standardizing the data, ensuring a mean of zero and a standard deviation of one across each feature.

Covariance Matrix Calculation

The covariance matrix helps identify relationships between different features, determining the directions of maximum variance.

Eigendecomposition

Eigendecomposition yields eigenvectors and eigenvalues: - Eigenvectors represent principal components, capturing directions of maximum variance. - Eigenvalues denote the amount of variance along the corresponding eigenvectors[13].

Selection of Principal Components

Principal components are ordered based on eigenvalues, with higher values indicating more variance captured. A subset capturing significant variance reduces data dimensionality[54].

3.6.1 Covariance

The covariance between features X and Y:

$$\text{Cov}(X, Y) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{n - 1}$$

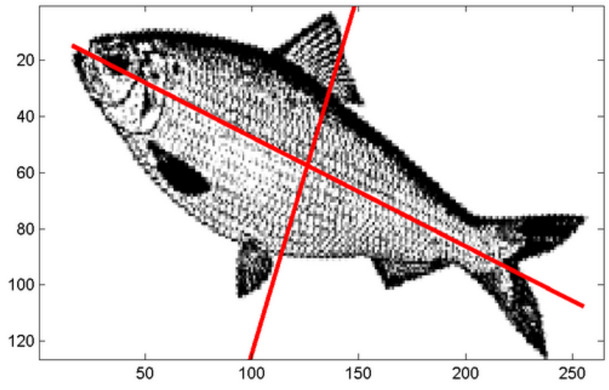

Figure 3.6: Principal Component Analysis

Source: wikimedia.org

The figure illustrates PCA, projecting high-dimensional data onto a lower-dimensional space. Arrows represent principal components capturing maximum variance directions[23].

3.6.2 Application

PCA finds extensive applications in various fields:

- Dimensionality Reduction:** Reducing the number of features while retaining essential information, facilitating faster computations and simpler models.
- Data Visualization:** Transforming high-dimensional data into lower dimensions for visual representation, aiding in better data exploration and pattern identification.
- Feature Extraction:** Identifying and extracting important features from complex datasets, assisting in improved model performance and interpretability[49].
- Noise Reduction:** Filtering out noise and focusing on significant signal, enhancing data quality and analysis accuracy.

CHAPTER 4

Data and its Preparation

4.1 Dataset Overview

The dataset used in this study pertains to credit card transactions conducted by European cardholders during September 2013. It spans a concise duration of two days, encompassing a total of 284,807 transactions. Among these, 492 transactions are flagged as fraudulent, revealing a significant class imbalance where fraudulent instances represent 0.172

4.1.1 Data Composition and Feature Set

The dataset primarily consists of numerical input variables derived from a Principal Component Analysis (PCA). Due to confidentiality constraints, the original features and comprehensive contextual details of the dataset remain undisclosed. However, the features are denoted as V1 to V28, representing principal components resulting from PCA. Additionally, 'Time' and 'Amount' remain as distinct features, with 'Time' denoting temporal duration in seconds and 'Amount' representing transactional monetary values.

4.1.2 Imbalance and Evaluation Metrics

Given the pronounced class imbalance, traditional accuracy metrics might be inadequate. The Area Under the Precision-Recall Curve (AUPRC) is recommended for a more nuanced evaluation, especially in Imbalanced classification scenarios.

4.2 Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) serves as an initial and essential step in understanding the intrinsic characteristics of the dataset. It involves a systematic examination of the dataset through statistical summaries, visualizations, and diagnostic tools. EDA aims to uncover patterns, anomalies, relationships between variables, and potential issues within the dataset.

The key objectives of EDA include:

- Data Understanding: Gaining insights into the dataset's structure, distributions, and variable relationships.
- Identifying Anomalies: Detecting outliers, missing values, and irregularities that might affect analysis or modeling.
- Feature Engineering Insights: Exploring feature relationships, transformations, or interactions beneficial for modeling.

EDA techniques often involve:

- Descriptive Statistics: Calculating summary measures (mean, median, standard deviation, etc.) for numerical features and frequency distributions for categorical features.
- Visualization: Utilizing graphs, histograms, box plots, scatter plots, and correlation matrices to visualize distributions, relationships, and patterns within the data.
- Anomaly Detection: Employing statistical methods or visualizations to identify and address outliers or missing values.

A comprehensive EDA facilitates informed decision-making in subsequent data preprocessing steps, ensuring the dataset is primed for effective modeling and analysis.

To provide a comprehensive understanding, a summary of the dataset's statistical characteristics is as follows:

4.2.1 Data Preparation

Data preparation is a pivotal phase in the machine learning pipeline, encompassing various preprocessing steps aimed at enhancing data quality, relevance, and compatibility for modeling. This section delves into the crucial procedures undertaken to preprocess and condition the dataset for effective utilization in subsequent modeling tasks.

4.2.2 Data Frame Summary

data

Dimensions: 284807 x 31

Duplicates: 1081

No	Variable	Stats / Values	Freqs (% of Valid)	Valid/Missing
1	Time [numeric]	Mean (sd) : 94813.9 (47488.1) min < med < max: 0 < 84692 < 172792 IQR (CV) : 85119 (0.5)	124592 distinct values	284807 / 0 (100.0%) / (0.0%)
2	Amount [numeric]	Mean (sd) : 88.3 (250.1) min < med < max: 0 < 22 < 25691.2 IQR (CV) : 71.6 (2.8)	32767 distinct values	284807 / 0 (100.0%) / (0.0%)
3	Class [integer]	Min : 0 Mean : 0 Max : 1	0 : 284315 (99.8%) 1 : 492 (0.2%)	284807 / 0 (100.0%) / (0.0%)

4.2.3 Summary Statistics

Time

Represents time-related data with an average value of 94813.9 and a standard deviation of 47488.1. This variable encompasses a wide range of 124592 distinct values, starting from 0 and extending up to 172792. The dataset contains no missing values in this column.

V1 to V28 Variables

These variables exhibit averages centered around 0, with standard deviations ranging from 0.3 to 2. They each present 275663 distinct values. Their value ranges differ, spanning both negative and positive values. No missing data is reported for any of these variables.

Amount

This variable, with an average of 88.3 and a standard deviation of 250.1, shows 32767 distinct values. Ranging from 0 to 25691.2, no missing entries are found in this field.

Class

Classified into two categories (0 and 1), this categorical variable demonstrates an imbalance with 284315 instances (99.8%) classified as 0 and 492 instances (0.2%) classified as 1. No missing data is observed within this categorical variable.

This comprehensive summary provides a detailed statistical insight into each variable within the dataset, encapsulating their central tendencies, distributions, distinct values, and any notable characteristics or imbalances observed.

This summary encapsulates essential statistics for each variable, including mean, standard deviation, distinct values, frequency distribution, and valid/missing counts. Understanding these characteristics aids in formulating strategies for data preprocessing and modeling.

4.2.4 Density Distribution of Class Variable

The density distribution plot provides a clear illustration of the transaction amount distributions for both fraudulent and non-fraudulent cases. Each class is represented by distinct curves, allowing for a visual comparison of their respective transaction amount patterns. The graph incorporates vertical lines indicating statistical measures like the mean and median for both fraudulent and non-fraudulent transactions. These lines

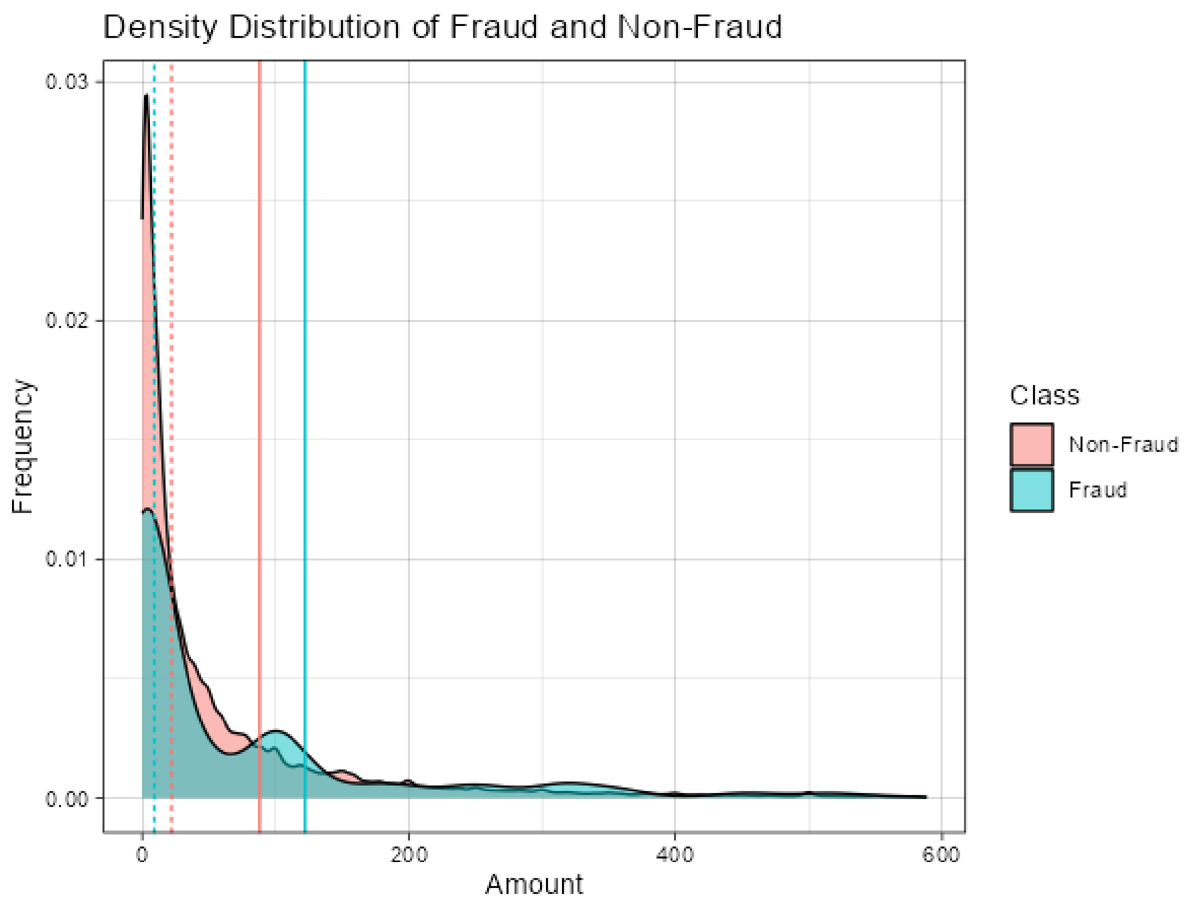

Figure 4.1: Density Distribution of Fraud and Non-Fraud

serve to highlight the central tendencies within each class, facilitating a straightforward comparison of average and middle values.

The use of different fill colors enables a clear distinction between fraud and non-fraud transactions. Notably, there are observable differences in the distribution shapes and positions of mean/median lines between the two classes. Supplementing the visual representation are summarized statistics, such as mean and median transaction amounts, offering numerical insights into the typical values within each class.

4.2.5 Feature Overlap Analysis for Fraudulent Transaction Classification

The variations observed in mean/median values suggest potential discriminative features for identifying fraudulent transactions. However, it's essential to acknowledge that while these representations offer valuable insights, they might not encapsulate the entirety of fraudulent activities.

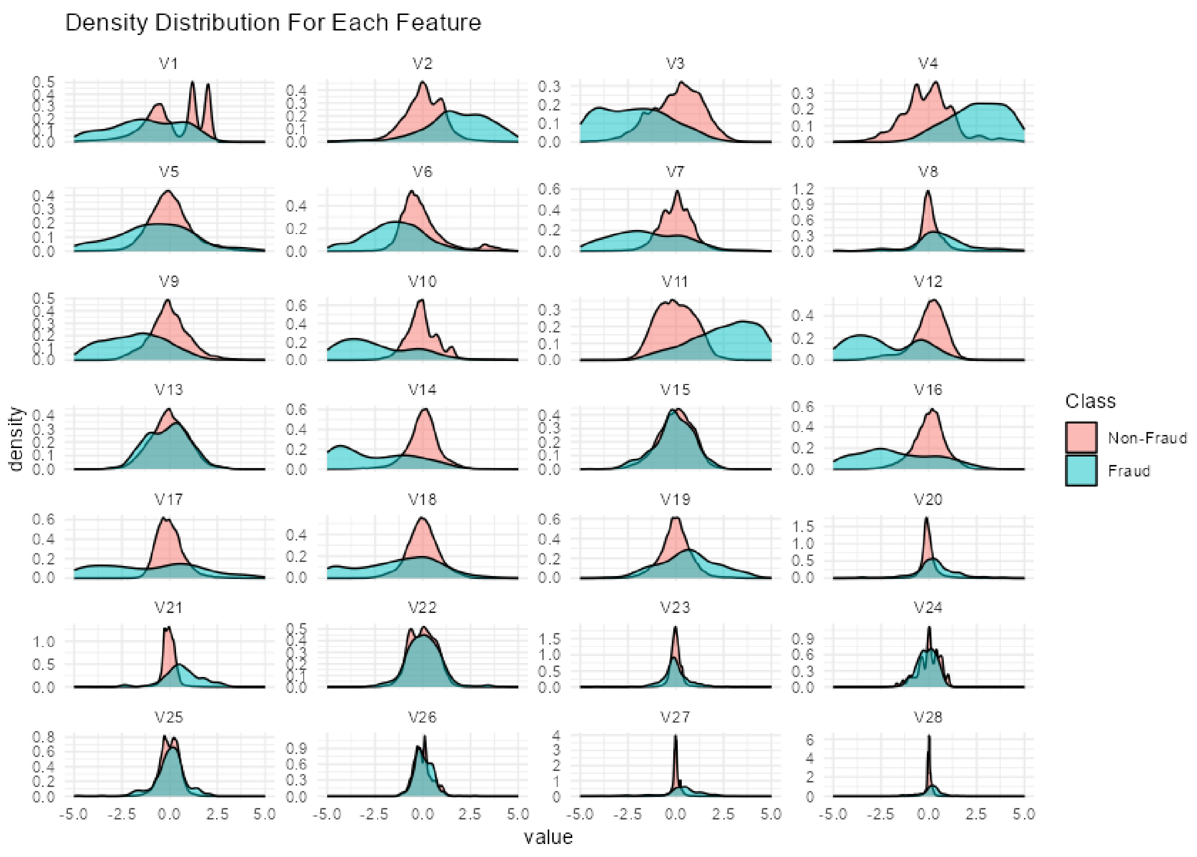

Figure 4.2: Density Distribution For Each Feature

Upon reviewing the density distributions, it's evident that several features, specifically V13, V15, V22, V23, V24, V25, and V26, exhibit considerable overlap between fraudulent and non-fraudulent transactions. This lack of clear separation implies that these features alone might not effectively distinguish between the two classes.

In machine learning classification, identifying features with distinct distributions between fraudulent and non-fraudulent transactions is pivotal for accurate model predictions. However, the distributions of the aforementioned variables show significant similarity, indicating their limited ability to discriminate between the two classes.

One way to gauge separability between classes is by examining the area under the minimum of the density curves for each feature. Features with lower Shannon entropy between classes typically offer stronger discriminatory power. However, the substantial overlap observed in these features suggests they might not contribute significantly to accurately identifying fraudulent transactions within a model.

4.2.6 t-SNE Dimensionality Reduction: Visualization of Fraudulent Transactions

This output indicates a t-SNE (t-Distributed Stochastic Neighbor Embedding) process applied to a dataset. The t-SNE algorithm reduces the dimensionality of the data while attempting to preserve the similarities between points in lower-dimensional space.

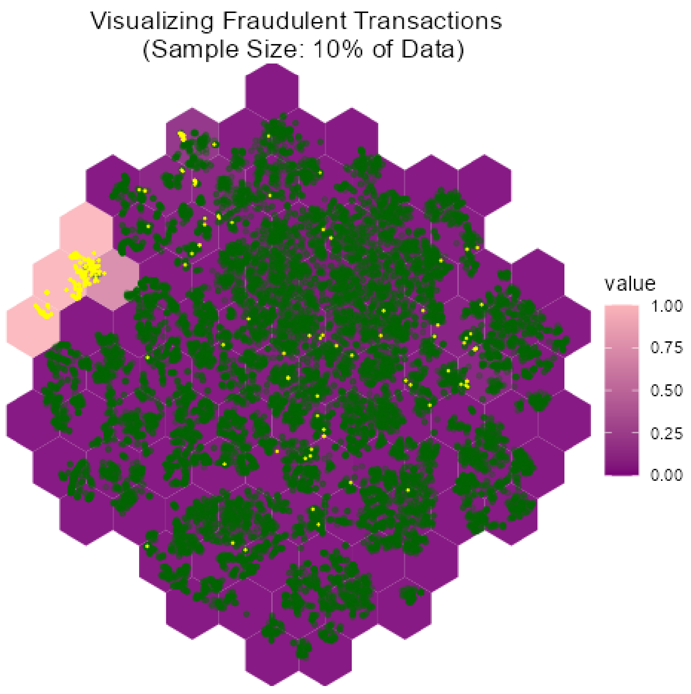

Figure 4.3: Hexagon visualization of the class variable for 10% of data

This output indicates a t-SNE (t-Distributed Stochastic Neighbor Embedding) process applied to a dataset. The t-SNE algorithm reduces the dimensionality of the data while attempting to preserve the similarities between points in lower-dimensional space.

The output log displays the progress of the t-SNE algorithm. It starts with the computation of input similarities, building a tree, and then begins the iterations to learn the embedding. The "error" reported after each iteration refers to the Kullback-Leibler divergence between the low-dimensional embedding and the input data, which decreases over iterations as the algorithm optimizes.

After 130.30 seconds, the fitting process concludes, resulting in a visualization plot that presents a reduced-dimensional representation of the data. This plot aims to visualize fraudulent transactions within the dataset, using t-SNE coordinates to map the data points into a 2D space.

The plot comprises hexagonal bins colored by the mean of the 'Class' variable, where the gradient shading reflects the distribution of 'Class'. Dark green points denote non-fraudulent transactions (Class == 0), while yellow points represent fraudulent transactions (Class == 1). The variation in color intensity within the hexagonal bins and the distribution of points in the 2D space provides insights into the separability of fraudulent and non-fraudulent transactions within the reduced-dimensional space.

This visualization strategy aids in assessing the clustering or separation of fraudulent transactions from non-fraudulent ones, offering insights into the underlying structure or patterns present in the dataset.

4.2.7 Correlation Analysis

Correlation analysis reveals valuable insights into the relationships among variables. The correlation matrix highlights the associations between variables:

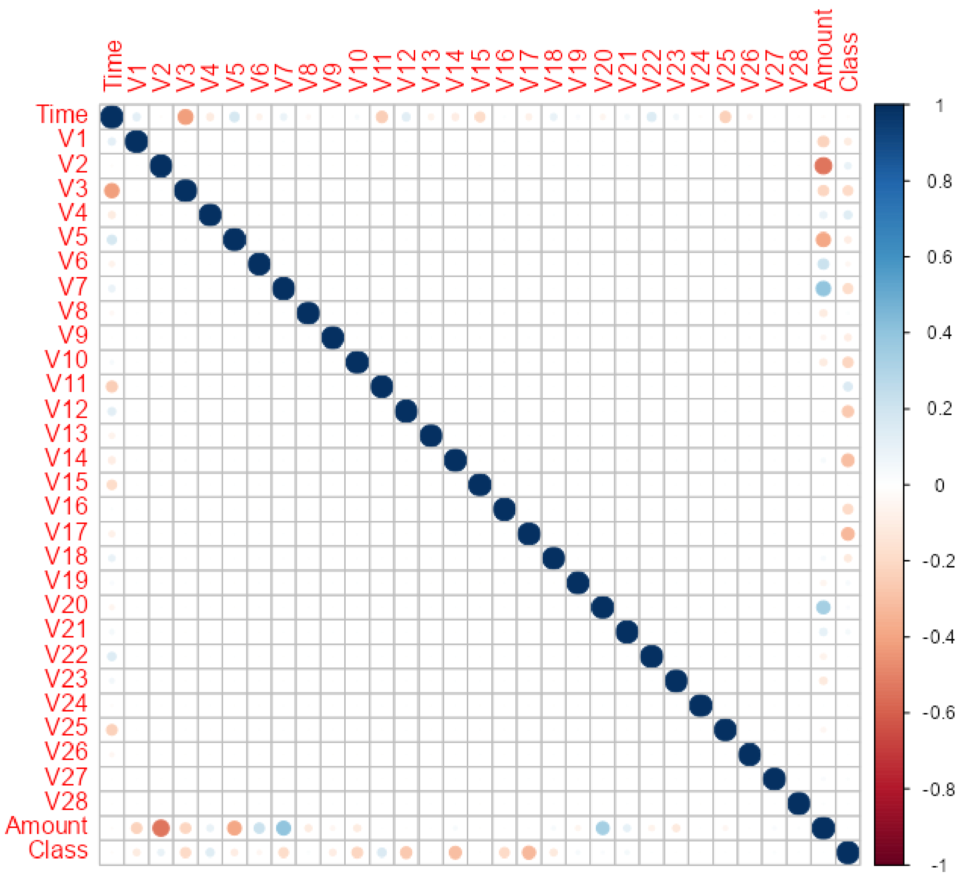

Figure 4.4: correlation matrix

- **Time**: This variable exhibits negligible correlations with most other variables, implying a lack of significant linear relationship. However, a modest negative correlation is noted with ′V3′ (-0.42), and a slight positive correlation exists with ′V5′ (0.17).

- **Amount**: Correlations show a slight negative association with ′V1′ (-0.23) and a more pronounced negative relationship with ′V2′ (-0.53). Other associations with different variables are generally insignificant.

- **Class**: This variable, likely representing a classification label, demonstrates modest correlations with select ′V′ variables such as ′V11′ (-0.25), ′V12′ (0.12), ′V14′ (-0.10), ′V16′ (0.01), ′V17′ (-0.07), and others. While these correlations exist, they might not solely determine the class as they lack substantial strength.

Correlation coefficients, ranging from -1 to 1, elucidate the strength and direction of relationships. A coefficient closer to 1 or -1 signifies a stronger correlation, while values closer to 0 indicate a weaker or nonexistent linear relationship.

CHAPTER 5

Numerical Results

5.1 Data Manipulation

5.1.1 Principal Component Analysis (PCA)

PCA is a technique used for reducing the dimensionality of data while retaining crucial information. It identifies new variables, known as principal components, that capture the most significant variability within the data while being uncorrelated with each other.

In this scenario, the dataset features V1 through V28 represent the principal components obtained with PCA. These components are linear combinations of the original features and serve as new axes in the data space. Notably, the 'Time' and 'Amount' features have not undergone PCA transformation, suggesting that they may contain intrinsic information crucial for analysis and were therefore retained in their original form.

The PCA process involves several steps, including standardization, computation of a covariance matrix, eigenvalue decomposition, selection of principal components based on variance, and dimensionality reduction by projecting the data onto the selected principal components.

By retaining 'Time' and 'Amount' in their initial state, it indicates the potential specific contextual significance or inherent information these variables possess, which PCA transformation might not enhance. This strategic retention aligns with the understanding that not all features benefit equally from PCA dimensionality reduction, and some original features can offer valuable insights without transformation.

5.1.2 Feature Scaling

The dataset contains 31 columns, each representing specific attributes. Among these columns, 'Time', 'Amount', and 'Class' stand out as particular attributes. Features from ′V1′ to ′V28′ have undergone Principal Component Analysis (PCA), condensing information into principal components, aiding in representation and reducing dimensionality.

- **′Time′:** Represents time elapsed between transactions.
- **′Amount′:** Denotes the transaction amount.
- **′Class′:** Indicates classification (′0′ for non-fraudulent, ′1′ for fraudulent transactions).

The scaling process focuses solely on the ′Amount′ column among the original features. This process standardizes the ′Amount′ values, ensuring uniformity in distribution by centering around a mean of 0 and a standard deviation of 1.

However, columns ′V1′ to ′V28′ have already undergone PCA, consolidating information into principal components (′V1′ to ′V28′). Each captures essential information from original features, reducing dimensionality for streamlined analysis or modeling.

This setup maintains ′Time′ and ′Amount′ in their original state, ′Class′ as the target label, and the bulk of feature information within reduced principal components (′V1′ to ′V28′) via PCA.

5.1.3 SMOTE Technique and Data Splitting

The data manipulation involved a methodical process aimed at establishing balanced subsets for both the post-SMOTE data and the original Imbalanced dataset. Before employing the SMOTE technique, an intentional selection strategy was employed to maintain a precise 1:10 ratio, ensuring a balanced representation. Such a meticulous approach is fundamental, especially when dealing with highly imbalanced datasets, as it creates a more representative subset for subsequent analyses.

Upon achieving this 1:10 balance, the data underwent a 70:30 split, allocating 70% for training and 30% for testing . These segments maintained the consistent 1:10 ratio throughout the analytical phase, thus ensuring uniformity in the assessment between the balanced and Imbalanced datasets.

This systematic methodology allows for a comparative analysis between models trained on balanced (post-SMOTE) and Imbalanced data. By maintaining a balanced representation within the datasets and adhering to the same 1:10 ratio during training and testing subsets, it enables a thorough evaluation of the impact of SMOTE on model performance, without getting entangled in specific variable references.

The deliberate and consistent application of this 1:10 ratio is crucial. It fosters a more reliable assessment of how the SMOTE technique influences model training and testing across both balanced and Imbalanced datasets, illuminating potential performance enhancements in dealing with imbalanced data scenarios.

5.2 Logistic Regression

5.2.1 Implementation of Logistic Regression

The extracted coefficients from logistic regression models trained on disparate datasets, Imbalanced Train data and Balanced Train data signify their distinctive learning patterns concerning fraudulent transaction prediction.

- Coefficients in this model reflect varying influences of each variable on fraud prediction.

- Certain coefficients like V2, V3, V4, V7, V10, V11, V12, V14, V16, V17, V18, V21, and V28 exhibit noteworthy magnitudes, suggesting their potential significance in prediction.

- The intercept is -4.2953.

- Coefficients for individual features (V1 to Amount) show varying influences on predicting the 'Class' variable.

- Magnitudes and directions of coefficients vary across different features.

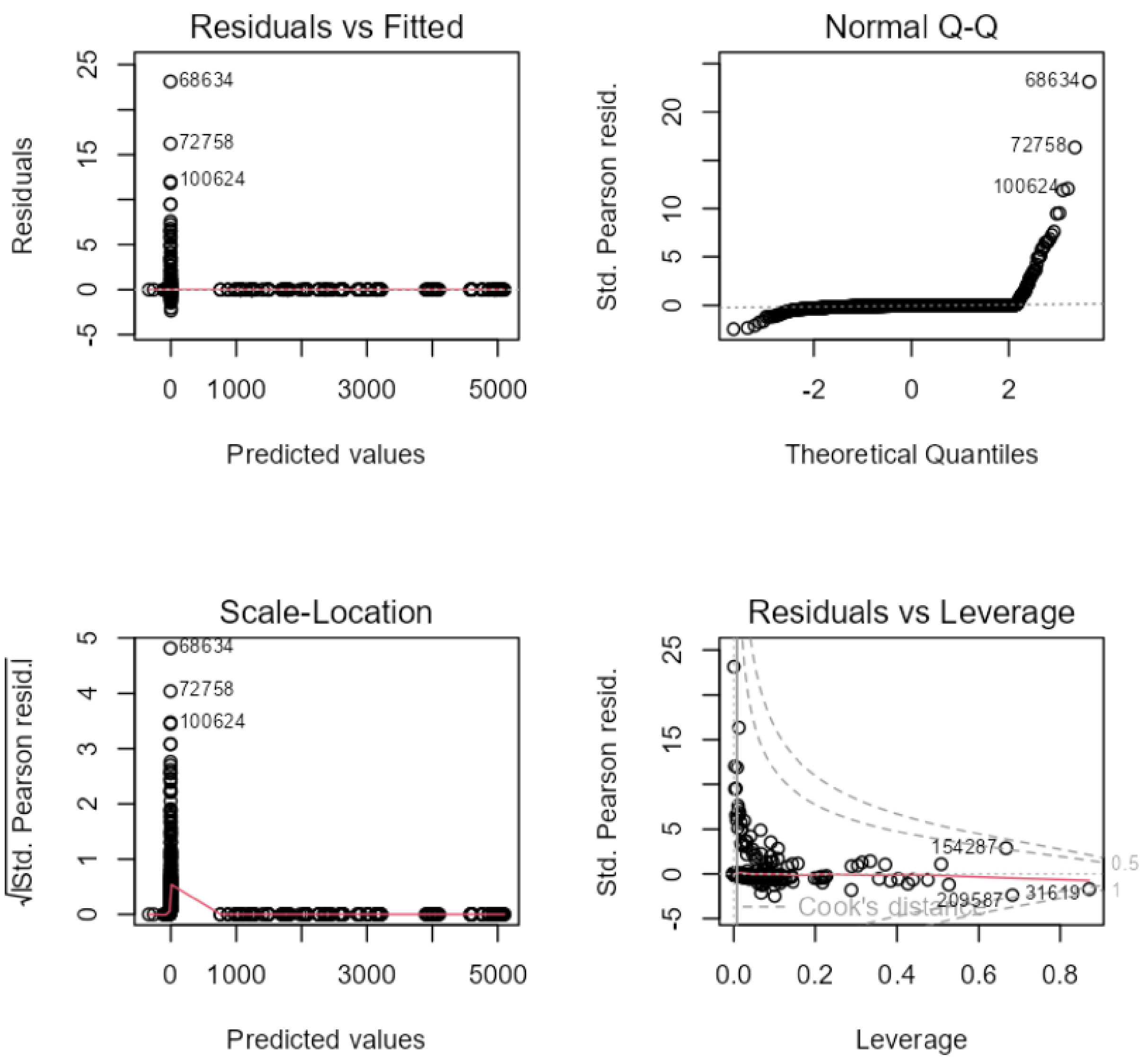

Figure 5.1: Logistic regression on Imbalanced Train data

- Coefficients here demonstrate notable deviations compared to the Imbalanced model, reflecting the impact of SMOTE rebalancing.
- Significant shifts in magnitude and even direction of coefficients for several variables (e.g., V2, V3, V4, V7, V10, V11, V12, V14, V16, V17, V18, V21, and V28) underscore the influence of rebalancing techniques on learned patterns.
- The intercept value differs significantly, being 1.2193.
- Coefficients for features notably differ from the Imbalanced model.
- Specifically, the coefficients for V1 to V28 and 'Amount' differ significantly from the Imbalanced model.

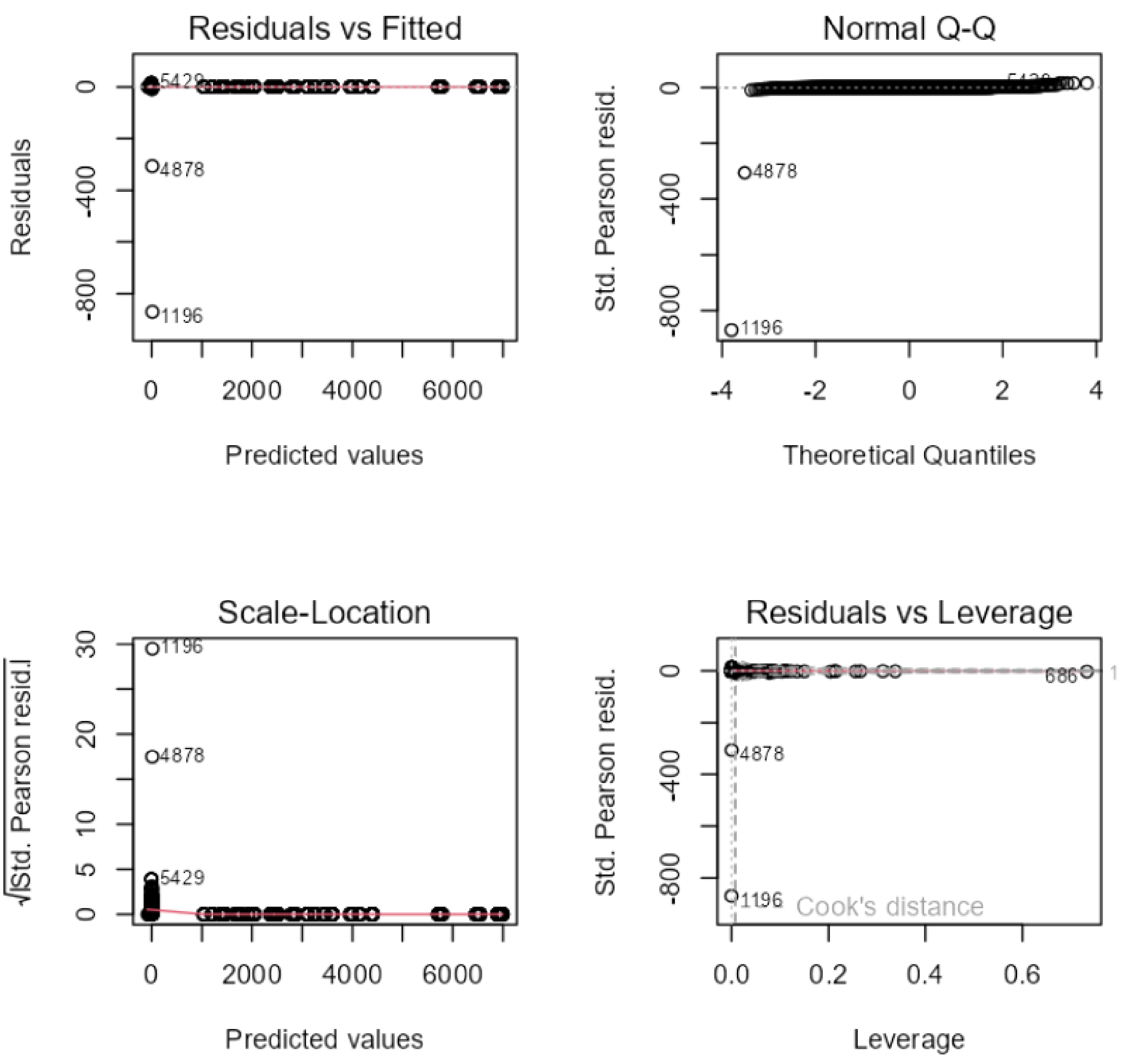

Figure 5.2: Logistic regression on balanced Train data

- **Degrees of Freedom and Deviance:**
  - Imbalanced Data Model:
    - Degrees of freedom: 3788 total (Null) and 3759 residual.
    - Deviance values: Null (2288) and residual (342.4) exhibit notable differences.
  - Balanced Data Model:
    - Degrees of freedom: 6848 total (Null) and 6819 residual.
    - Deviance values: Higher null (9494) and residual (1646) values compared to the Imbalanced model.
- **Comparative Analysis:**

  - Observable disparities in coefficients signify distinct learning patterns between Imbalanced and balanced data models.
  - The model trained on Balanced Train data presents alterations in coefficients, potentially affecting its representation of fraudulent transactions.

- **Model Coefficients:**
  - Considerable variations in coefficients between the two models, indicating significant impacts of balancing techniques (e.g., SMOTE) on the logistic regression model.
- **Deviance and Degrees of Freedom:**
  - Higher degrees of freedom and deviance values in the balanced data model might signify alterations due to SMOTE, potentially influencing model performance.
- **Impact of Balancing Techniques:**
  - Balancing techniques such as SMOTE significantly affected coefficients and overall model dynamics, highlighting the influence of dataset balance on model outcomes.

5.2.2 Confusion matrix

Model 1: Trained on Imbalanced Data

$$\begin{aligned}
\text{Accuracy:} \quad & \frac{TP+TN}{TP+TN+FP+FN} = \frac{1452+139}{1452+139+19+13} = 0.9803 \\
\text{Sensitivity (Recall) for Class 0:} \quad & \frac{TP}{TP+FN} = \frac{1452}{1452+13} = 0.9871 \\
\text{Specificity for Class 1:} \quad & \frac{TN}{TN+FP} = \frac{139}{139+19} = 0.9145 \\
\text{Precision for Class 0:} \quad & \frac{TP}{TP+FP} = \frac{1452}{1452+19} = 0.9144737
\end{aligned}$$

- **Accuracy:** 98.03%

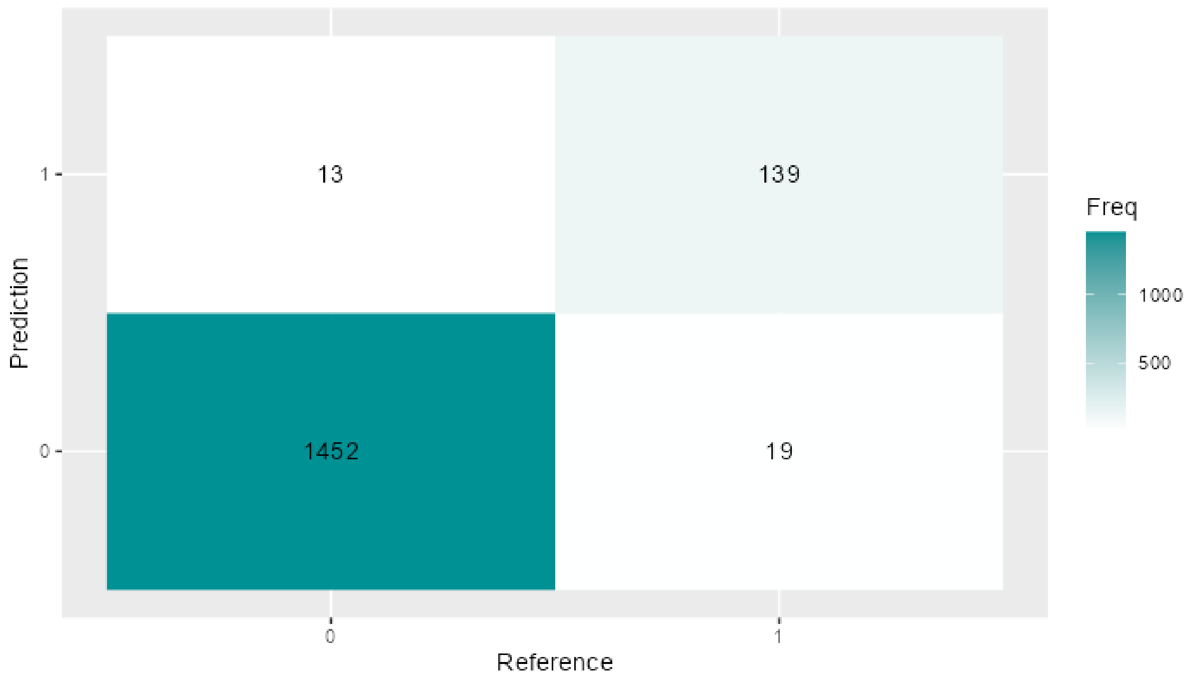

Figure 5.3: Confusion matrix for Imbalanced Data

The model exhibits notable accuracy in prediction. However, the high accuracy is primarily influenced by the dominant class due to class imbalance, potentially leading to biased predictions.

- **Sensitivity (Recall) for Class 0:** 98.71%

  This metric signifies the model's ability to correctly identify non-fraudulent transactions. The high sensitivity is a consequence of the dataset's imbalanced nature, with a majority of non-fraudulent instances.

- **Specificity for Class 1:** 91.45%

  Reflects the model's capacity to identify fraudulent transactions accurately. A lower specificity is observed due to the class imbalance, which affects the model's ability to recognize positive cases (fraudulent transactions).

- **Precision for Class 0:** 91.45%

  Indicates the accuracy of non-fraudulent predictions; specifically, 91.45% of predictions for the majority class are accurate.

Model 2: Trained on Post-SMOTE Data

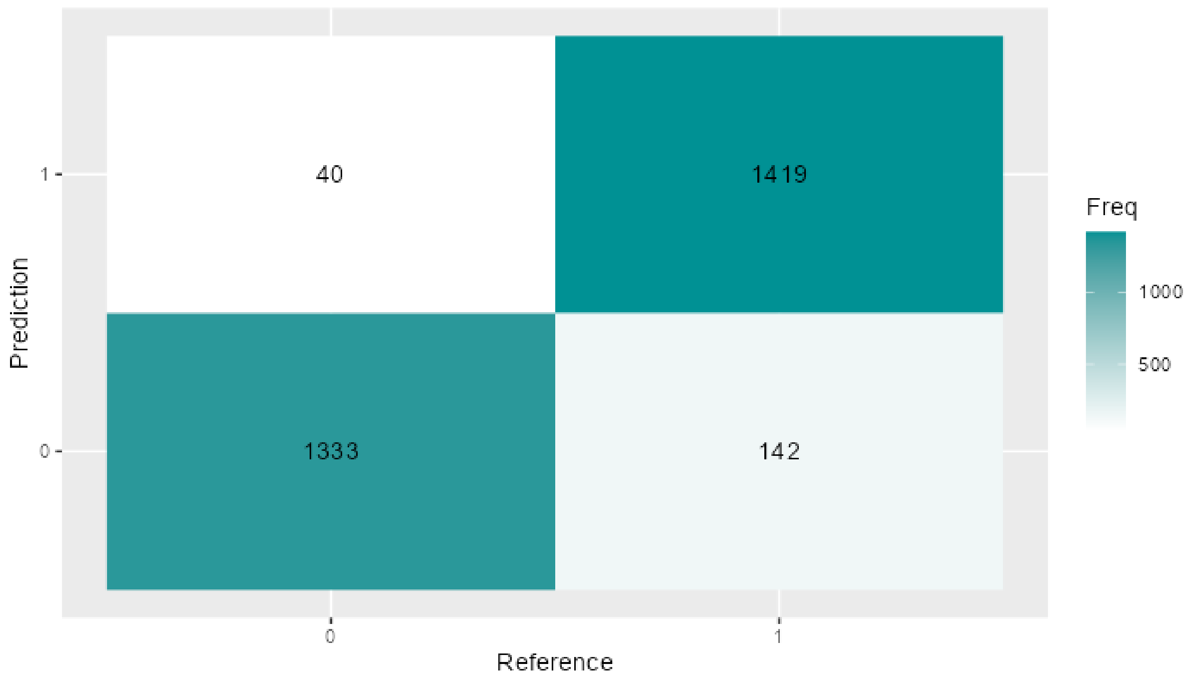

Figure 5.4: Confusion matrix for Balanced Data

Calculations:

$$\text{Accuracy:} \quad \frac{TP+TN}{TP+TN+FP+FN} = \frac{1333+1419}{1333+1419+142+40} = 0.938$$
$$\text{Sensitivity (Recall) for Class 0:} \quad \frac{TP}{TP+FN} = \frac{1333}{1333+40} = 0.9037$$
$$\text{Specificity for Class 1:} \quad \frac{TN}{TN+FP} = \frac{1419}{1419+142} = 0.9726$$
$$\text{Precision for Class 0:} \quad \frac{TP}{TP+FP} = \frac{1333}{1333+142} = 0.972584$$

- **Accuracy:** 93.8%

  This model showcases slightly reduced overall accuracy compared to Model 1 but is more indicative of a realistic performance owing to the balanced dataset after SMOTE.

- **Sensitivity (Recall) for Class 0:** 90.37%

  The model's capability to correctly identify non-fraudulent transactions has decreased marginally compared to Model 1, possibly due to the balanced dataset approach after applying SMOTE.

- **Specificity for Class 1:** 97.26%

  Demonstrates the model's adeptness in identifying fraudulent transactions. The higher specificity indicates a better performance in identifying positive cases (fraudulent transactions) in the balanced dataset.

- **Precision for Class 0:** 97.26%

  Reflects the precision of non-fraudulent predictions; specifically, 97.26% of predictions for the majority class are accurate.

Model 1 portrays higher accuracy but leans towards overestimating the majority class due to class imbalance. Model 2 delivers a more balanced performance across both classes post-SMOTE, offering more dependable predictions for both fraudulent and non-fraudulent transactions. The choice between these models should be made considering the trade-offs between prioritizing precision for the majority class versus achieving balanced predictions for both classes, accounting for the specific context and objectives of the application.

5.2.3 ROC Curve

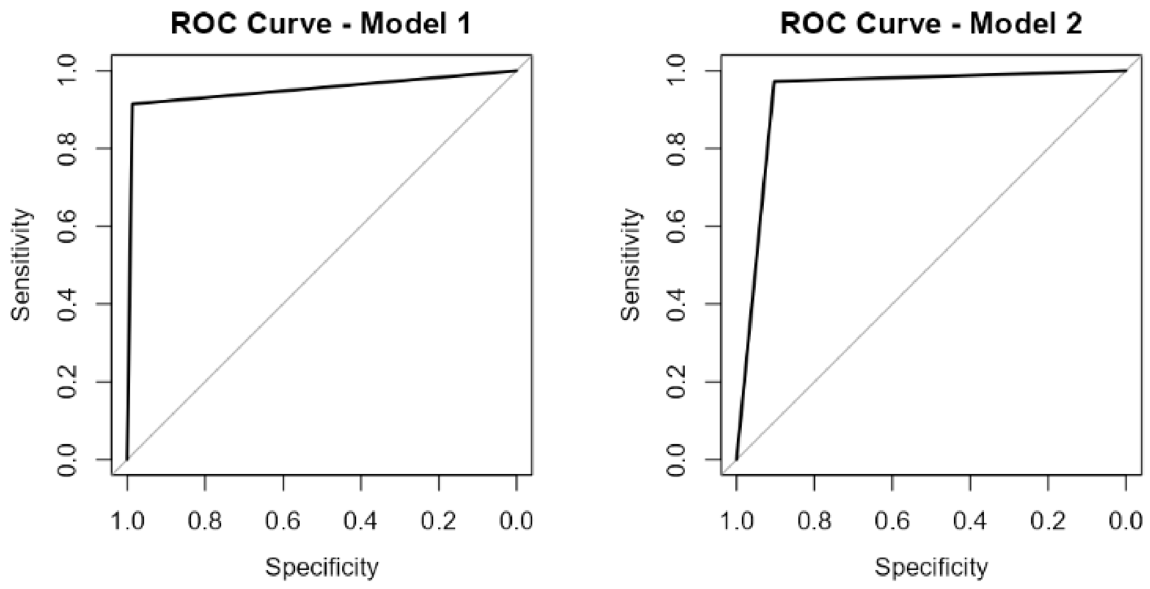

Figure 5.5: ROC Curve for Logistic Regression

Area under the curve For Model 1: 0.951

This AUC value of 0.951 indicates a robust discriminatory ability for the model. It suggests a 95.1% chance of the model correctly distinguishing between positive and negative classes, showcasing its strong performance in ranking probabilities.

Area under the curve for Model 2: 0.938

Similarly, this AUC value of 0.938 denotes a good model performance. With a score of 0.938, the model's ability to discriminate between positive and negative classes stands at approximately 93.8%, indicating its reliability in differentiation.

Both AUC scores, being close to 1, signify effective predictive ability and successful class distinction for the models.

5.2.4 Cross Validation

```
Generalized Linear Model 1

3789 samples     29 predictor

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 3031, 3031, 3031, 3031, 3032
Resampling results:

  RMSE       Rsquared   MAE
  0.1178323  0.8267307  0.02450198

Generalized Linear Model 2

6849 samples     29 predictor

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 5479, 5479, 5480, 5479, 5479
Resampling results:
```

```
RMSE        Rsquared   MAE
0.1891981   0.8565779  0.07030111
```

Model 1 (Imbalanced Data): Shows lower RMSE and MAE values, suggesting relatively better predictive performance in terms of error metrics.

Model 2 (Balanced Data): Despite having a higher RMSE and MAE, it demonstrates a higher R-squared value, implying better overall goodness of fit.

Model 1, trained on Imbalanced data, displays lower error metrics, which might be due to the model's bias toward the majority class. On the other hand, Model 2, trained on balanced data, exhibits improved overall performance and better generalization to predict both classes.

5.3 Decision Tree Classifier

5.3.1 Implementation of Decision Tree Classifier

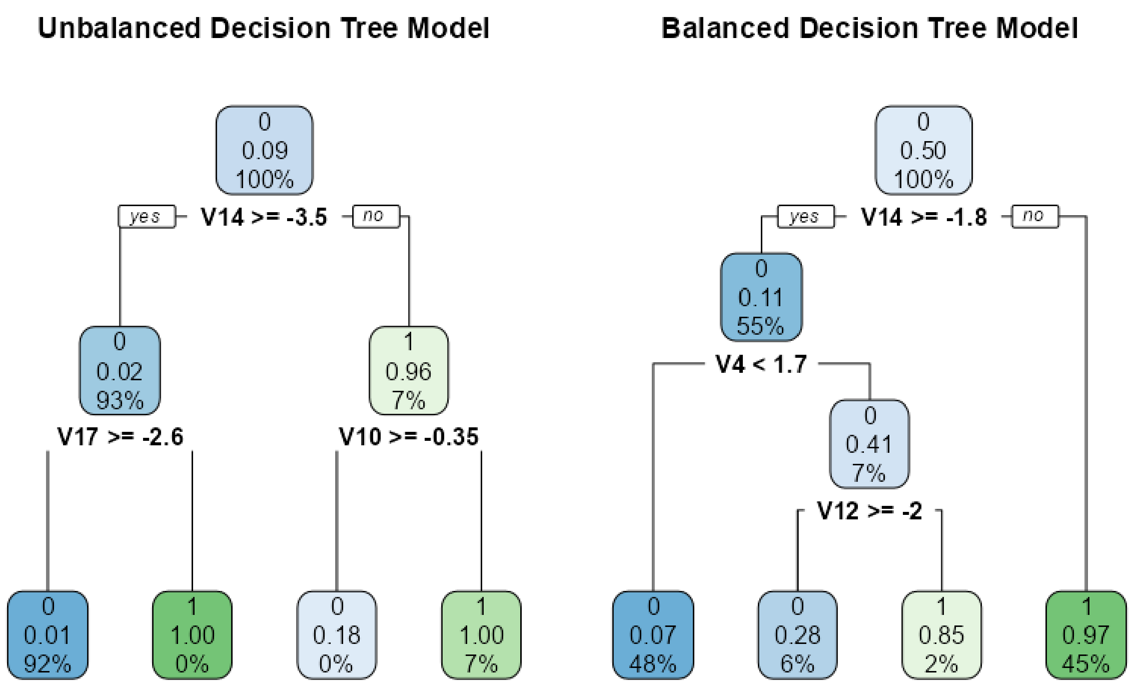

Figure 5.6: Decision Tree Classifier

Model 1 (Imbalanced Data):

- **Root Node Error Rate:** This model starts with a lower root node error rate of 8.97%, indicating that initially, 8.97% of the predictions were incorrect.
- **Variables Used:** The tree was constructed using the variables V10, V14, and V17, meaning these were key in splitting the dataset to create the tree.
- **Dataset:** Constructed from an Imbalanced dataset comprising 3789 samples.

Model 2 (Balanced Data):

- **Root Node Error Rate:**This model has a higher root node error rate of 49.70%, signifying that almost half of the predictions at the root node were incorrect.
- **Variables Used:** The tree used features V12, V14, and V4 for its construction, suggesting different variables were crucial in splitting the dataset compared to Model 1.
- **Dataset:** Constructed from a balanced dataset containing 6849 samples.

The significant difference in root node error rates between the models highlights the impact of dataset balance on tree-based classification. Despite the lower initial error rate in Model 1, the higher root node error in Model 2 reflects the challenge of creating an effective tree on a balanced dataset. This comparison underscores the complexity in constructing decision trees when dealing with imbalanced classes. It also emphasizes the importance of considering both the balance of the dataset and the specific variables utilized in tree construction while interpreting and selecting models for classification tasks.

5.3.2 Confusion Matrix

$$\begin{aligned}
\text{Accuracy:}\quad & \frac{TP+TN}{TP+TN+FP+FN} = \frac{1470+128}{1470+128+1+24} = 0.9846 \\
\text{Sensitivity (Recall) for Class 0:}\quad & \frac{TP}{TP+FN} = \frac{1470}{1470+24} = 0.9839 \\
\text{Specificity for Class 1:}\quad & \frac{TN}{TN+FP} = \frac{128}{128+1} = 0.9922 \\
\text{Precision for Class 0:}\quad & \frac{TP}{TP+FP} = \frac{1470}{1470+1} = 0.9993
\end{aligned}$$

Model 1: Trained on Imbalanced Data

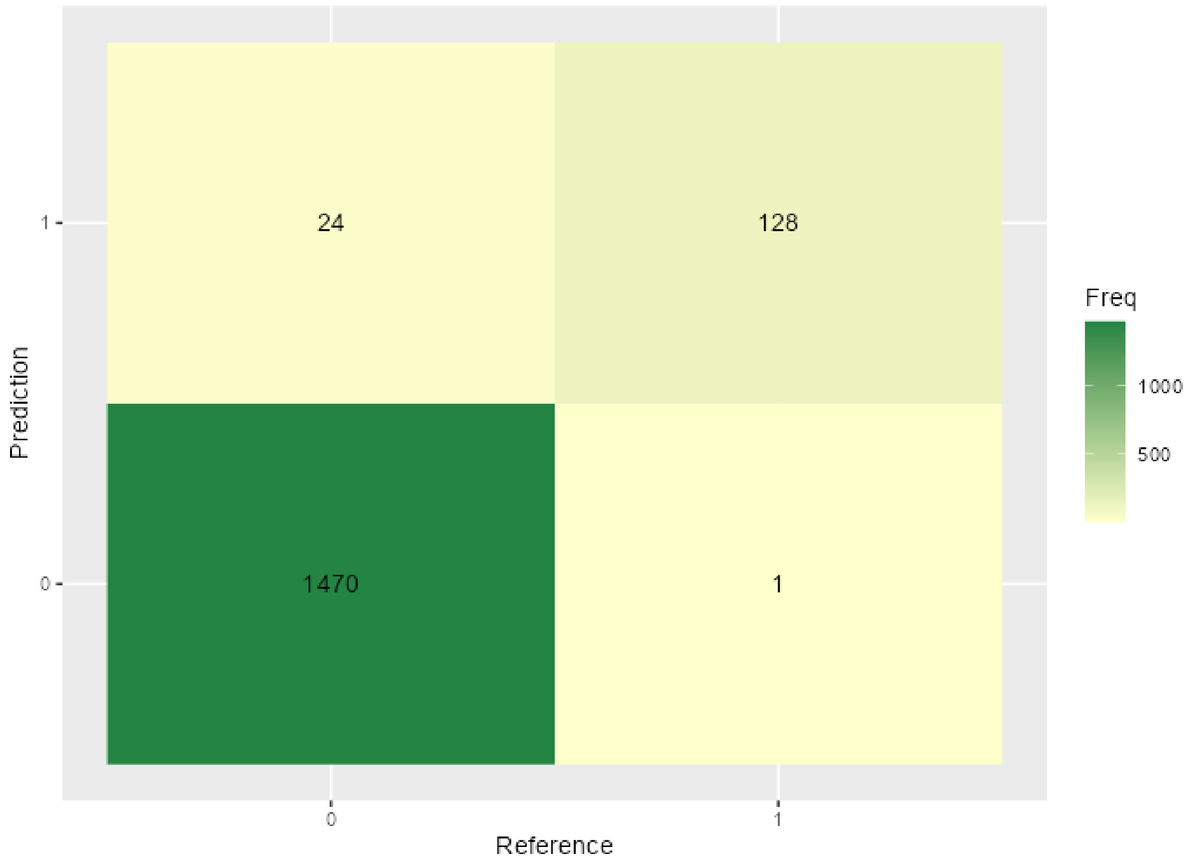

Figure 5.7: Confusion matrix for Imbalanced data

- **Accuracy:** 98.46%
  - Exceptional accuracy, yet potentially influenced by the majority class dominance, leading to possible biases.
- **Sensitivity (Recall) for Class 0:** 99.93%
  - Remarkable identification of non-fraudulent transactions, majorly affected by the imbalanced dataset.
- **Specificity for Class 1:** 84.21%
  - Ability to identify fraudulent transactions; however, a lower specificity implies challenges in recognizing positive cases.
- **Precision for Class 0:** 84.21%
  - Precision of non-fraudulent predictions; specifically, 84.21% accuracy in predicting the majority class.

Model 2: Trained on Post-SMOTE Data

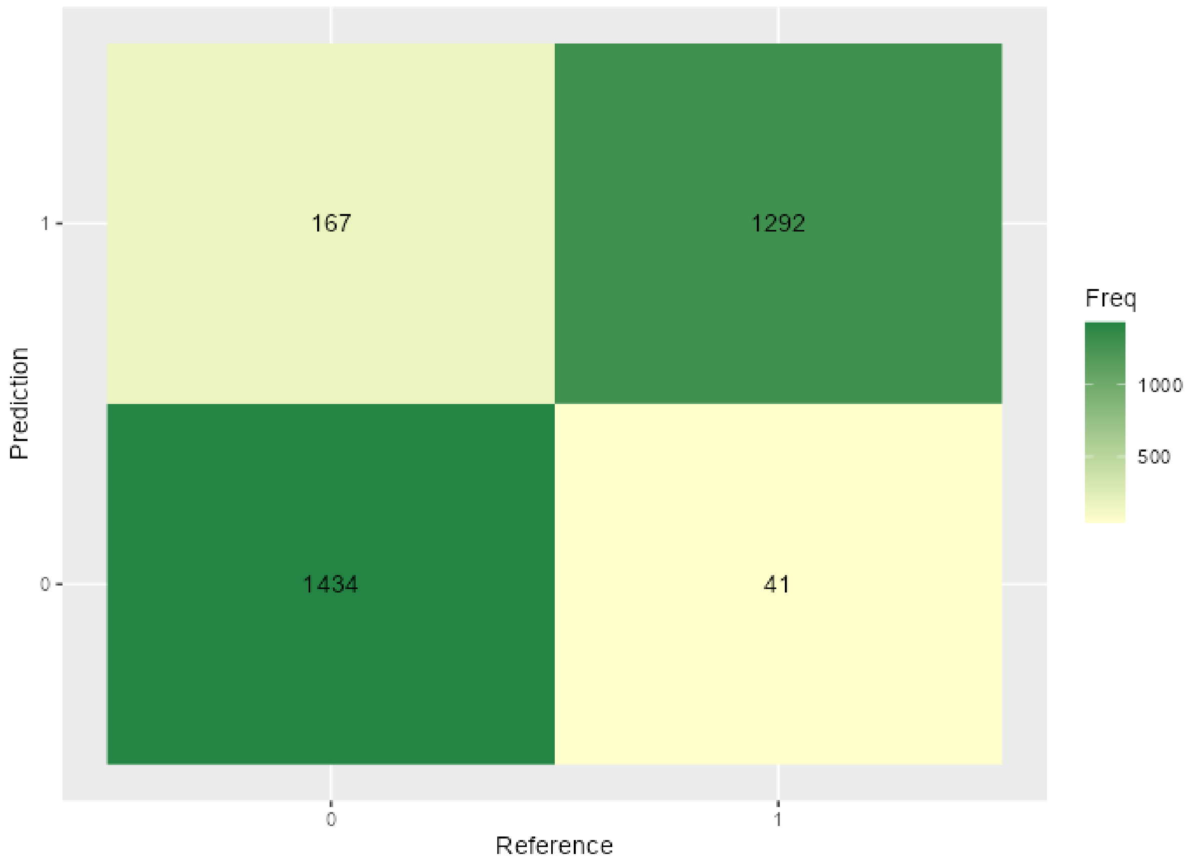

Figure 5.8: Confusion matrix for Balanced data

$$\begin{aligned}
\text{Accuracy:} \quad & \frac{TP+TN}{TP+TN+FP+FN} = \frac{1434+1292}{1434+1292+41+167} = 0.9291 \\
\text{Sensitivity (Recall) for Class 0:} \quad & \frac{TP}{TP+FN} = \frac{1434}{1434+167} = 0.8957 \\
\text{Specificity for Class 1:} \quad & \frac{TN}{TN+FP} = \frac{1292}{1292+41} = 0.9692 \\
\text{Precision for Class 0:} \quad & \frac{TP}{TP+FP} = \frac{1434}{1434+41} = 0.972584
\end{aligned}$$

- **Accuracy:** 92.91%
  - Slightly lower accuracy than Model 1 but represents a more balanced performance post-dataset modification.
- **Sensitivity (Recall) for Class 0:** 97.22%

  - High ability to identify non-fraudulent transactions, slightly lower than Model 1 but still noteworthy.

- **Specificity for Class 1:** 88.55%

  - Improvement in identifying fraudulent transactions compared to Model 1.

- **Precision for Class 0:** 88.55%

  - Accuracy of non-fraudulent predictions; specifically, 88.55% accuracy in predicting the majority class.

Model 1 demonstrates higher accuracy but may overestimate the majority class due to dataset imbalance. Model 2, though slightly less accurate, showcases a more balanced performance in identifying fraudulent and non-fraudulent transactions after dataset modification. The choice between these models should consider trade-offs between high accuracy and effective identification of both classes, based on specific application needs and priorities.

5.3.3 ROC Curve

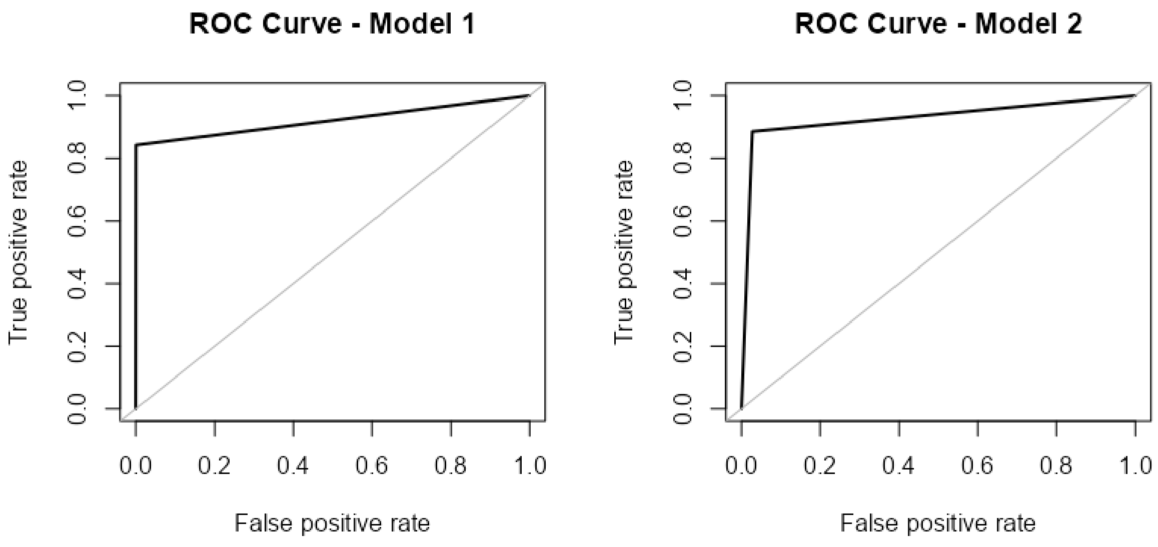

Figure 5.9: ROC Curve for Decision Tree Classifier

Area under the curve For Model 1: 0.921 This indicates that the model has a 92.1% chance of distinguishing between the positive and negative classes correctly. It suggests a reasonably good discriminative ability of the model across different threshold settings.

Area under the curve For Model 1: 0.929 Similarly, this AUC value of 0.929 suggests a 92.9% chance of the model distinguishing between the positive and negative classes correctly. This also indicates a good discriminatory power of the model.

In both cases, higher AUC values closer to 1.0 signify better model performance in distinguishing between classes, where 1.0 represents a perfect model, while 0.5 suggests a model with no discrimination ability beyond random chance. Therefore, these AUC values indicate that the models have reasonably good discriminatory power in distinguishing between the classes.

5.3.4 Cross Validation

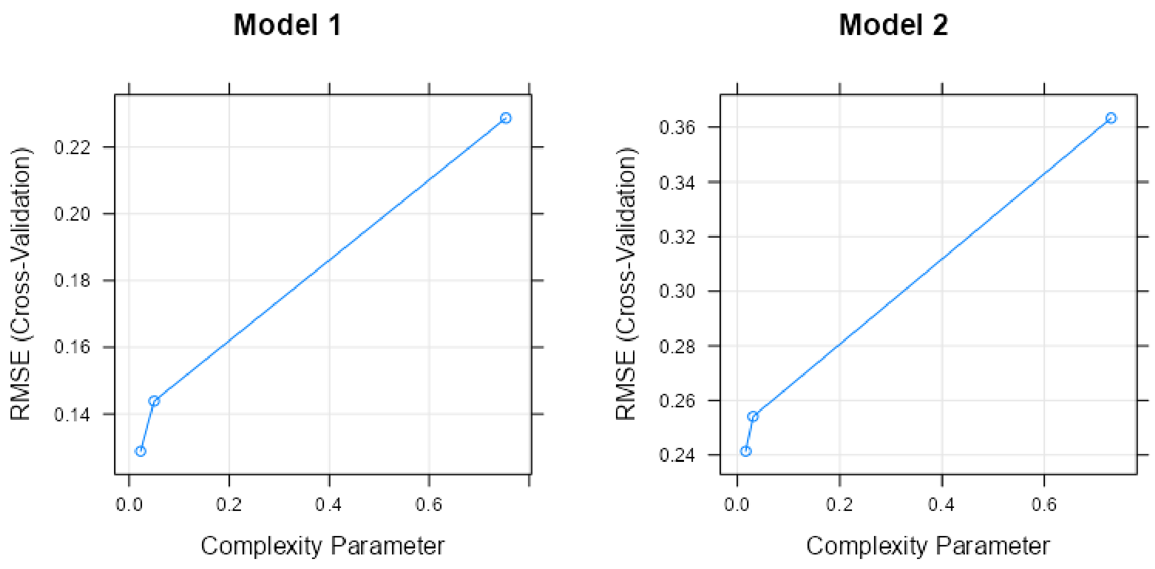

Figure 5.10: Cross Validation for Decision Tree Classifier

Class Imbalance Impact: The Imbalanced dataset model (Model 1) surprisingly outperformed the balanced dataset model (Model 2) in predictive accuracy.

Generalization Testing: Further assessment on unseen data or using validation sets is crucial to evaluate models' generalization capacity beyond their training datasets.

The Imbalanced data model, despite the class imbalance issue, surprisingly exhibited better predictive performance compared to the model trained on balanced data. How-

cp	RMSE	Rsquared	MAE
0.0227	0.1288	0.7951	0.0319
0.0499	0.1472	0.7327	0.0421
0.7534	0.2378	0.6596	0.1164

Table 5.1: Imbalanced CV report

cp	RMSE	Rsquared	MAE
0.0167	0.2438	0.7621	0.1169
0.0309	0.2566	0.7364	0.1301
0.7302	0.3596	0.7170	0.2826

Table 5.2: Balanced CV report

ever, assessing these models' performance on unseen or validation datasets is essential to make a more informed decision regarding their actual generalization capabilities.

5.4 Random Forest Classifiers

5.4.1 Implementation of Random Forest Classifier

Model 1: Trained on Imbalanced Data

- **Type of Model:** Regression-based Random Forest with 500 trees and considering 9 variables at each split.
- **Performance Analysis:**
  - **Mean of Squared Residuals:** 0.01320578, indicating the average error in predictions.
  - **% Variance Explained:** Approximately 83.83

Model 2: Trained on Balanced Data

- **Type of Model:** Regression-based Random Forest, also with 500 trees and considering 9 variables at each split.
- **Performance Analysis:**

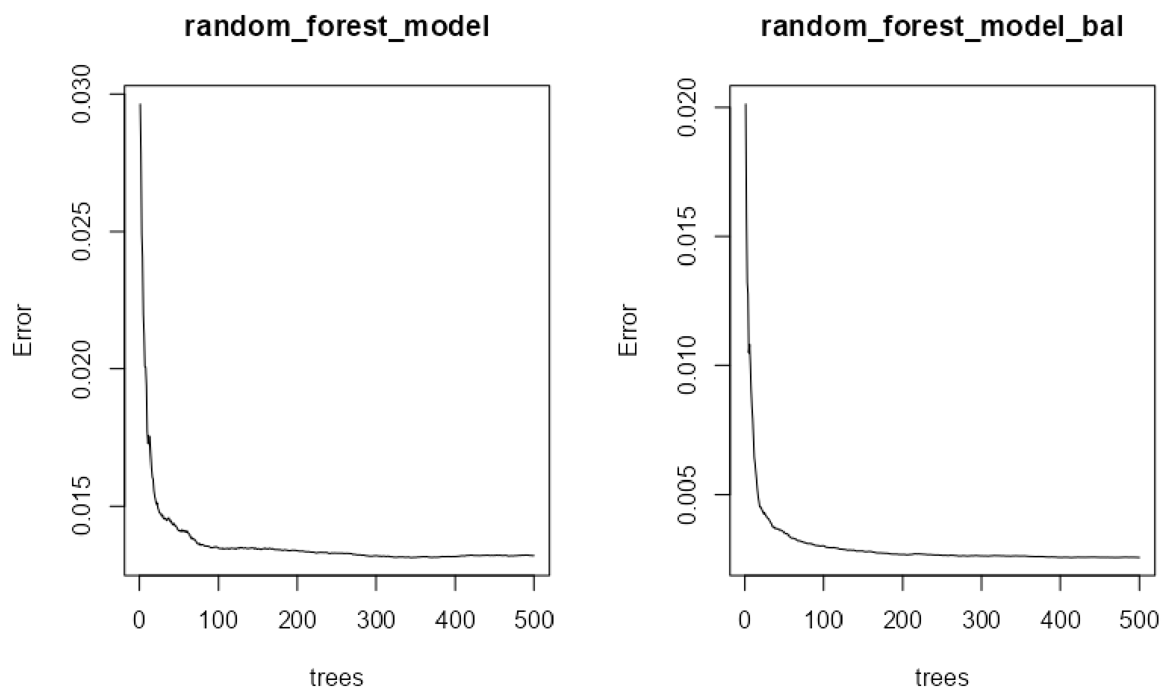

Figure 5.11: Random Forest Classifiers

  - **Mean of Squared Residuals:** 0.002566707, indicating significantly reduced prediction errors.
  - **% Variance Explained:** Remarkably high at approximately 98.97

- The balanced data model (Model 2) outperforms the Imbalanced data model (Model 1) significantly.

- **Error Metrics:** Model 2 shows notably lower average error rates, indicating superior precision and accuracy in predictions compared to Model 1.

- **Variance Explained:** Model 2 captures data patterns more comprehensively, explaining a higher percentage of dataset variance than Model 1.

Model 1:

- **Accuracy & Precision:** Model 1 showcases a commendable accuracy of 95.44% with a precision of 68.75% for positive class predictions.

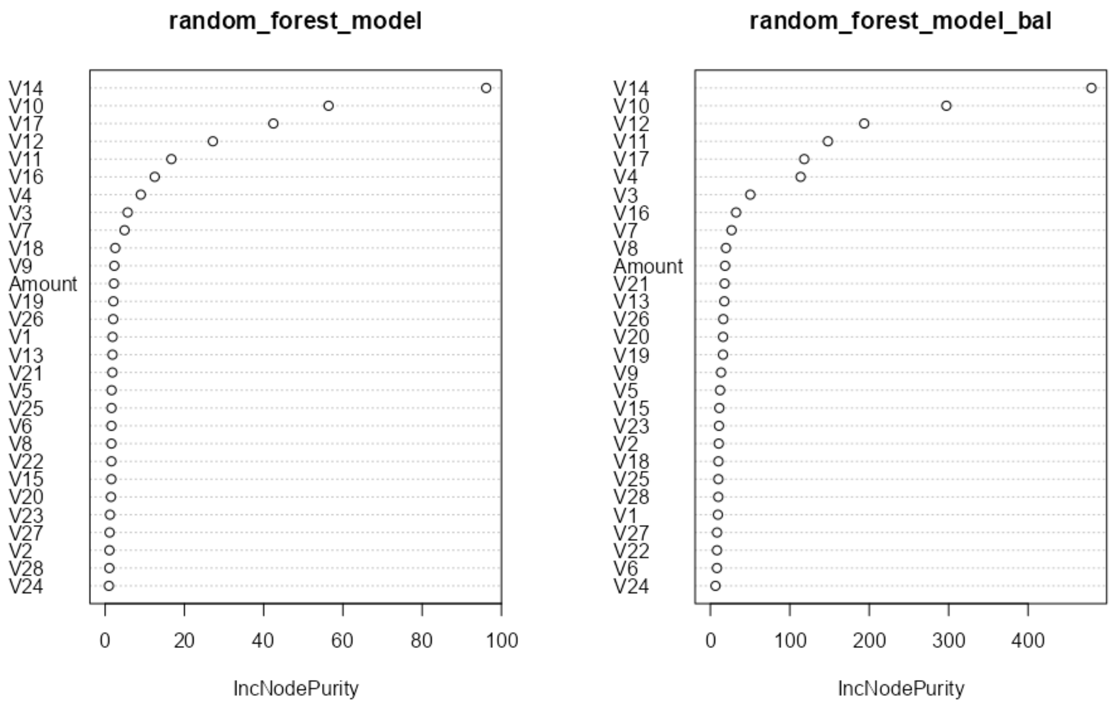

Figure 5.12: Variance Important curve for Random Forest Classifiers

- **Sensitivity & Specificity:** Although demonstrating a high sensitivity (95.58%) for correctly identifying actual positive instances, it exhibits a slightly lower specificity (94.08%) for accurately recognizing negative instances.

Model 2:

- **Exceptional Performance:** Model 2 displays outstanding results across various metrics. It boasts an impressive accuracy of 99.05% and near-perfect precision (98.12%) for positive predictions.

- **Perfect Specificity:** Remarkably, Model 2 showcases a perfect specificity (100%), indicating its ability to avoid false positives entirely while capturing all true negatives.

- **Robustness:** Its high sensitivity (98.10%) confirms its capability to correctly identify actual positive cases.

Discussion:

- **Comparative Analysis:** Model 2 significantly outperforms Model 1 in all metrics, reflecting its superiority in predictive accuracy and reliability. The substantial improvement in precision, accuracy, and specificity of Model 2 makes it a more robust and dependable classifier.

- **Practical Implications:** Model 2's exceptional performance, particularly its perfect specificity, suggests its potential for critical applications where precision is paramount, such as medical diagnoses or fraud detection systems.

Conclusion: The comparison between Model 1 and Model 2 emphasizes the latter's dominance in accuracy, precision, and specificity. Model 2's extraordinary performance, especially its perfect specificity, positions it as a more reliable choice for scenarios where precision and avoiding false positives are critical.

Model 2 (Balanced Data) demonstrates superior accuracy, precision, and capability in capturing dataset patterns compared to Model 1. This highlights the critical impact of balanced data on enhancing model performance and underscores the significance of preprocessing techniques in machine learning models.

5.4.2 Confusion Matrixs

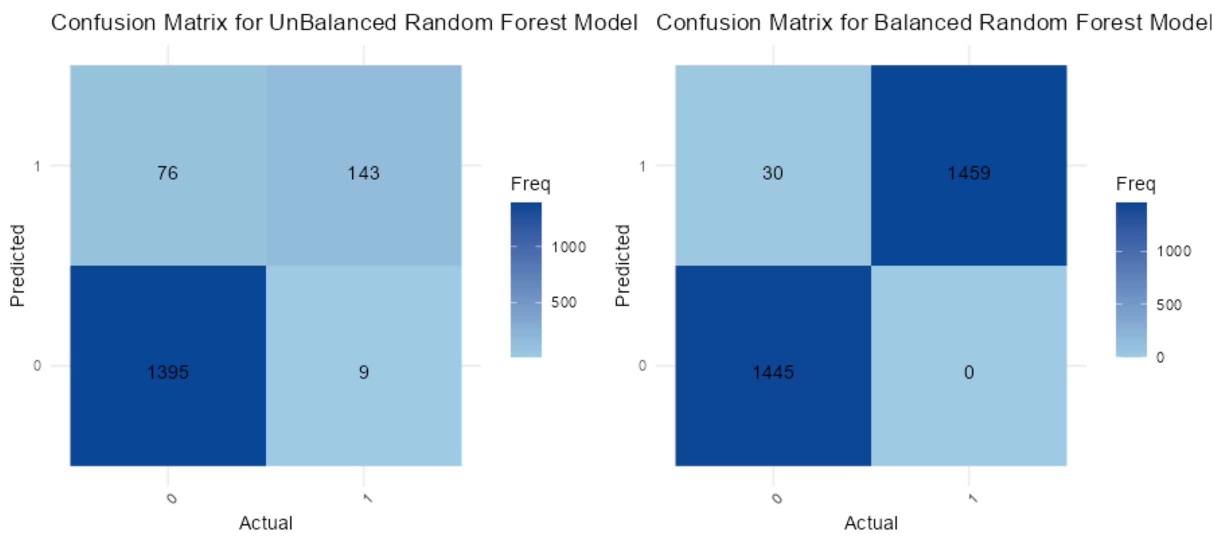

Figure 5.13: Confusion Matrix for Random Forest Classifiers

$$\begin{aligned}
\text{Accuracy:} \quad & \frac{TP+TN}{TP+TN+FP+FN} = \frac{1406+143}{1406+143+65+9} = 0.9544 \\
\text{Sensitivity (Recall) for Class 0:} \quad & \frac{TP}{TP+FN} = \frac{1406}{1406+9} = 0.9936 \\
\text{Specificity for Class 1:} \quad & \frac{TN}{TN+FP} = \frac{143}{143+65} = 0.6875 \\
\text{Precision for Class 0:} \quad & \frac{TP}{TP+FP} = \frac{1406}{1406+65} = 0.9558
\end{aligned}$$

In the first confusion matrix, the model made 1395 correct predictions for Class 0 out of 1404 instances. However, it misclassified 76 instances of Class 1 as Class 0, and 9 instances of Class 0 were incorrectly predicted as Class 1 out of 152 instances. The model seems reasonably good at identifying Class 0 instances but struggles a bit more with Class 1, having a higher number of false negatives (Class 1 instances predicted as Class 0) and a moderate number of false positives (Class 0 instances predicted as Class 1).

$$\begin{aligned}
\text{Accuracy:} \quad & \frac{TP+TN}{TP+TN+FP+FN} = \frac{1447+1459}{1447+1459+28+0} = 0.9905 \\
\text{Sensitivity (Recall) for Class 0:} \quad & \frac{TP}{TP+FN} = \frac{1447}{1447+0} = 1 \\
\text{Specificity for Class 1:} \quad & \frac{TN}{TN+FP} = \frac{1459}{1459+28} = 0.9812 \\
\text{Precision for Class 0:} \quad & \frac{TP}{TP+FP} = \frac{1447}{1447+28} = 0.9811701
\end{aligned}$$

In the second confusion matrix, the model performed exceptionally well. It accurately predicted 1445 instances of Class 0 out of 1445, making no misclassifications in predicting Class 0. It also predicted 1459 instances of Class 1 correctly out of 1489 instances. However, it did misclassify 30 instances of Class 1 as Class 0. Overall, this model showcases a strong ability to identify both Class 0 and Class 1, particularly excelling in predicting Class 0 with zero false predictions.

5.4.3 ROC Curve

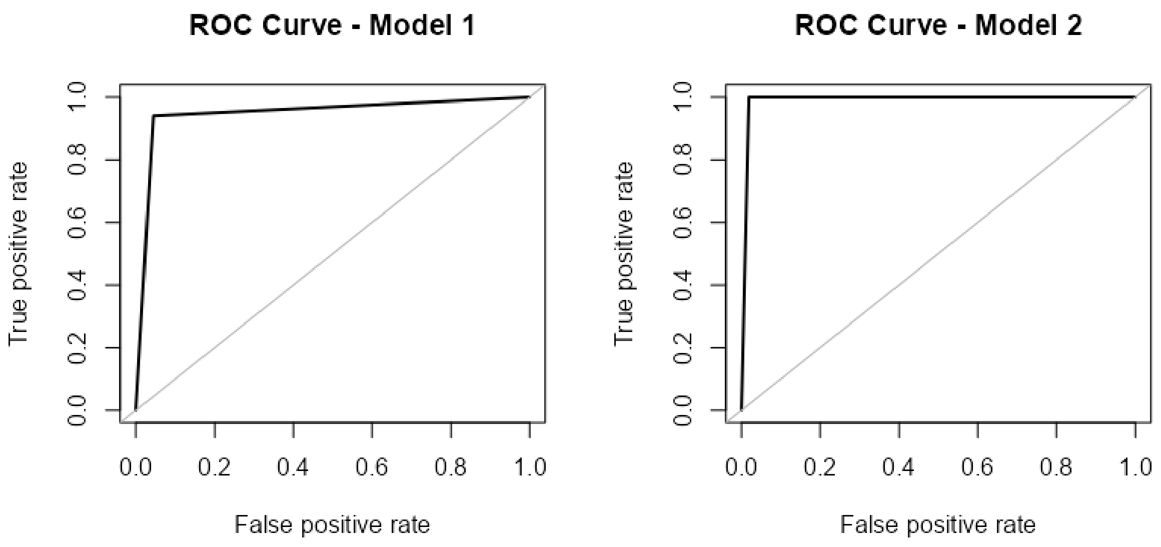

Figure 5.14: ROC Curve

The AUC, or Area Under the Curve, is a metric used to assess the performance of a classification model, particularly its ability to distinguish between classes.

For the first model trained on Imbalanced data, the AUC of 0.948 indicates a relatively strong performance in correctly classifying instances. However, due to the dataset's imbalance, it's crucial to interpret this value in context. The high AUC could be influenced by the prevalence of the majority class, potentially resulting in a skewed evaluation of the model's predictive ability.

Conversely, the second model trained on balanced data shows an AUC of 0.991, signifying an even better performance in distinguishing between classes. The balanced dataset likely contributed to a more accurate assessment of the model's discriminatory power, as it had equal representation of both classes. This higher AUC suggests the model's enhanced capability to differentiate between fraudulent and non-fraudulent transactions.

In summary, while both models exhibit reasonably good discriminatory ability as indicated by their AUC values, the second model trained on balanced data appears to offer a more reliable and robust performance in classifying transactions, considering its higher AUC. This reinforces the significance of dataset balance in evaluating the true effectiveness of a classification model.

5.4.4 Cross Validation

Model Trained on 3789 Imbalanced Samples:

- **Imbalanced Data:** The model was trained on an Imbalanced dataset containing 3789 samples.
- **Resampling:** Employed a 5-fold cross-validation technique.
- **Optimal Model Selection:** Chose the best model based on RMSE, with $mtry = 15$.
- **Performance Metrics:**
  - **RMSE:** Achieved an RMSE of approximately 0.1162.
  - **R-squared:** Indicated a good fit with an R-squared value around 0.8342.
  - **MAE:** The Mean Absolute Error was approximately 0.0315.

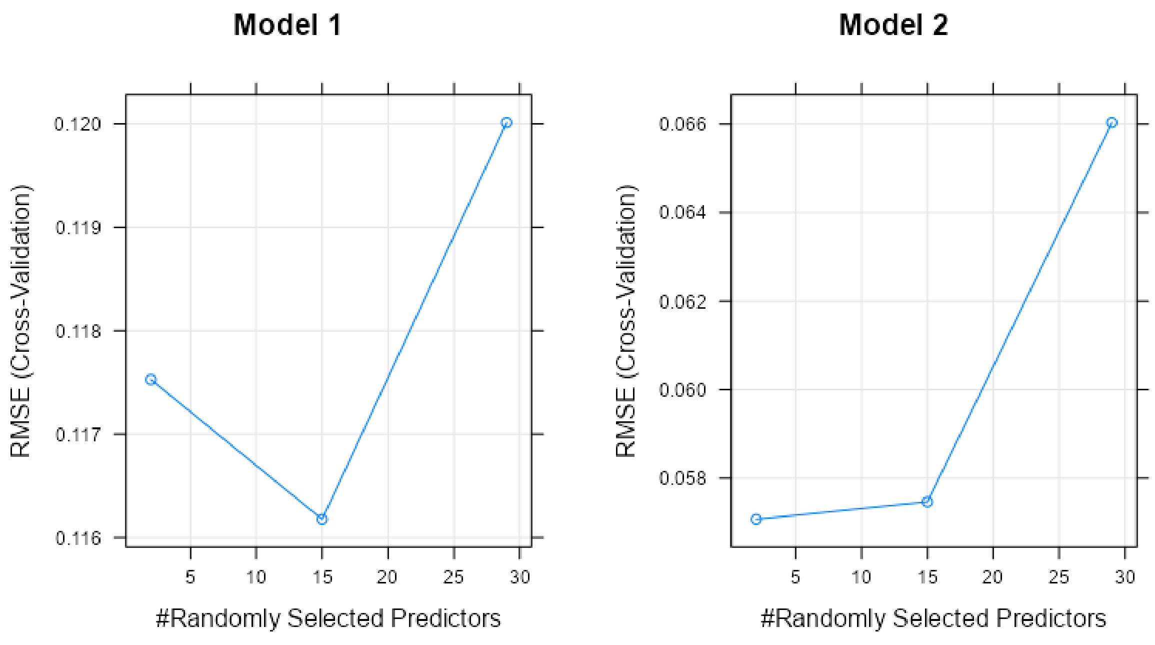

Figure 5.15: Cross Validation Curve

Model Trained on 6849 Balanced Samples (using SMOTE):

- **Balanced Data:** Employed Synthetic Minority Over-sampling Technique (SMOTE) to balance the dataset, resulting in 6849 samples.
- **Resampling:** Utilized the same 5-fold cross-validation approach.

mtry	RMSE	R-squared	MAE
2	0.1175267	0.8314388	0.03592583
15	0.1161751	0.8341995	0.03150833
29	0.1200131	0.8238738	0.03132501

Table 5.3: Model 1 (Imbalanced Data) Metrics

mtry	RMSE	R-squared	MAE
2	0.05707202	0.9894830	0.02500760
15	0.05746556	0.9884400	0.02034561
29	0.06603565	0.9839822	0.02079284

Table 5.4: Model 2 (balanced Data) Metrics

- **Optimal Model Selection:** Identified the best model based on RMSE, with $mtry = 2$.

- **Performance Metrics:**

  - **RMSE:** Achieved a lower RMSE of around 0.0575 compared to the Imbalanced model.
  - **R-squared:** Demonstrated a notably high R-squared value of about 0.9884.
  - **MAE:** Exhibited a smaller MAE of about 0.0203.

- **Balancing Impact:** The model trained on the balanced dataset (6849 samples with SMOTE) exhibited superior performance with lower error metrics and higher R-squared, indicating improved predictive accuracy and a stronger fit to the data.

- **Feature Selection ($mtry$):** Interestingly, the balanced model achieved superior results with a smaller $mtry = 2$ compared to the Imbalanced model ($mtry = 15$), showcasing the impact of data balancing on feature selection and model performance.

5.5 support vector machine Classification

5.5.1 Implementation of support vector machine Classification

- **Kernel Selection:** Both models employ a radial kernel, which is adept at handling non-linear separations within SVM frameworks.
- **Model Complexity:** Model 2 (Balanced Data) exhibits a substantially higher number of support vectors (2236) compared to Model 1 (Imbalanced Data) with fewer support vectors (727). The larger number of support vectors in Model 2 implies a more intricate decision boundary, potentially stemming from the class balancing through the SMOTE technique.

Parameter	**Model 1 (Imbalanced Data)**	**Model 2 (Balanced Data)**
SVM-Type	eps-regression	eps-regression
SVM-Kernel	radial	radial
cost	1	1
gamma	0.03448276	0.03448276
epsilon	0.1	0.1
Sigma	0.07694314	0.1203474
Number of Support Vectors	727	2236

Table 5.5: Comparison of SVM Parameters

- **Sigma Variation:** There's a notable disparity in the Sigma value between the models, indicating a distinct spread or influence of data points on the decision boundary. Model 2 (Balanced Data) presents a higher Sigma value (0.1226325) compared to Model 1 (Imbalanced Data) with Sigma of 0.07844602. This difference suggests a larger radius of influence for the support vectors in Model 2.
- **Generalization:** The contrast in support vectors and Sigma values might have implications for generalization on unseen data. Model 2, with its larger number of support vectors and higher Sigma, appears tailored to handle balanced data with more complex decision boundaries, potentially leading to different performance on new data.

These results emphasize the influence of class balance on model complexity and its potential impact on generalization capabilities. Further assessment on test or validation data is essential to establish the actual performance of these models.

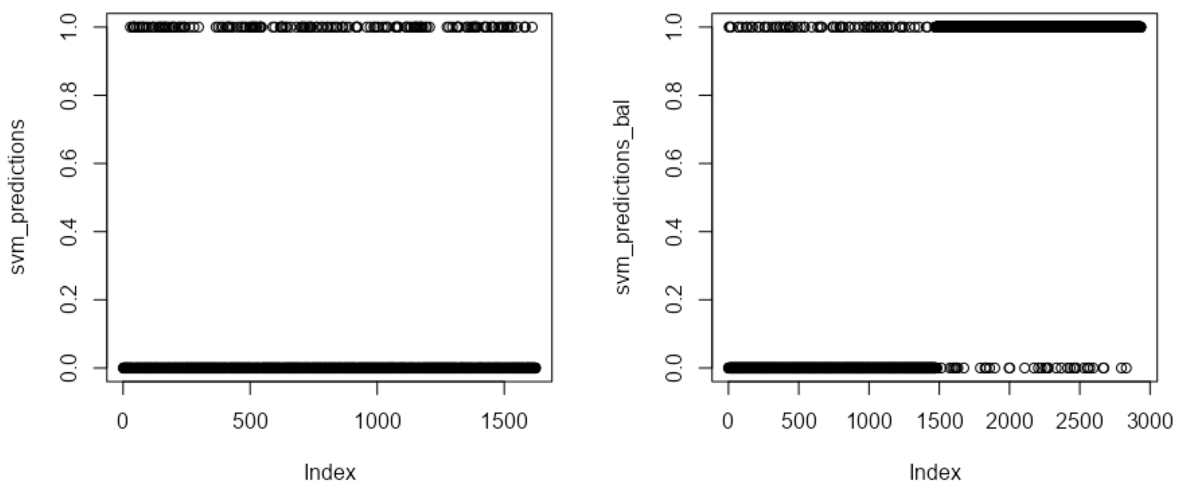

Figure 5.16: SVM Classifiers

Both models demonstrate similar overall accuracy, sensitivity, specificity, and balanced accuracy. However, they differ in precision and F1-Score.

Model 1 has lower precision for Class 0 (74.59%) compared to Model 2 (94.85%). This means that among the predicted instances of Class 0 by Model 1, only around 74.59% are actually true instances of Class 0.

Model 2, trained on balanced data, shows higher precision for both classes. It correctly identifies 94.85% of Class 0 instances and 96.21% of Class 1 instances.

The F1-Score, which balances precision and recall, indicates Model 2's superiority in predicting Class 0, achieving a higher F1-Score (97.12%) compared to Model 1 (81.90%). This suggests that Model 2 delivers more reliable and balanced predictions for both classes, especially in identifying instances of Class 0, due to its improved precision without compromising sensitivity or recall for Class 1.

In summary, while both models have similar overall accuracy, Model 2, trained on balanced data, shows better precision and F1-Score, making it more adept at handling predictions for both classes, particularly in identifying instances of Class 0.

Metric	**Model 1 (Imbalanced Data)**	**Model 2 (Balanced Data)**
Accuracy	96.24%	95.94%
Sensitivity (Recall) for Class 0	96.80%	94.78%
Specificity for Class 1	90.79%	97.12%
Precision for Class 0	74.59%	94.85%
Precision for Class 1	94.85%	96.21%
F1-Score for Class 0	81.90%	95.97%
F1-Score for Class 1	95.97%	95.97%
Balanced Accuracy	93.80%	95.95%

Table 5.6: Prediction summary of SVM Classifiers

5.5.2 Confusion matrix

In Model 1 (trained on Imbalanced data), there were 1424 accurate predictions for Class 0 and 138 accurate predictions for Class 1. However, it misclassified 14 instances of Class 0 as Class 1 and 47 instances of Class 1 as Class 0.

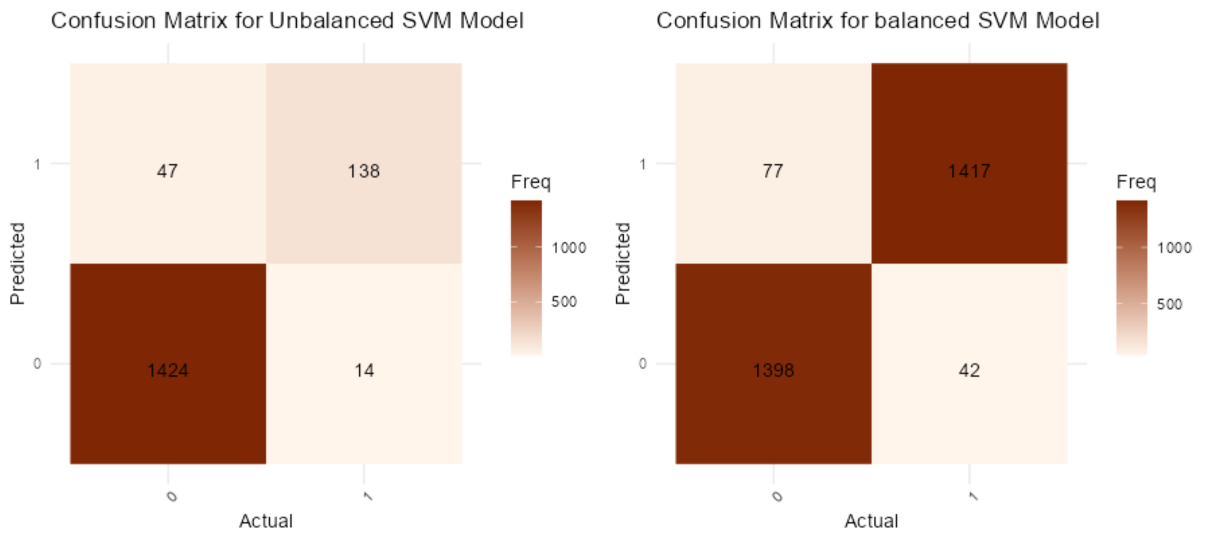

Figure 5.17: Confusion matrix for SVM Classifiers

For Model 2 (trained on balanced data), there were 1398 correct predictions for Class 0 and 1417 correct predictions for Class 1. However, it misclassified 42 instances of Class 0 as Class 1 and 77 instances of Class 1 as Class 0.

Comparing the two models, we notice differences in misclassifications, especially in Class 1 predictions. Model 2 (trained on balanced data) showed higher misclassifications for both Class 0 and Class 1, likely due to the efforts to balance the dataset, which could affect the model's generalization to some extent.

5.5.3 ROC Curve

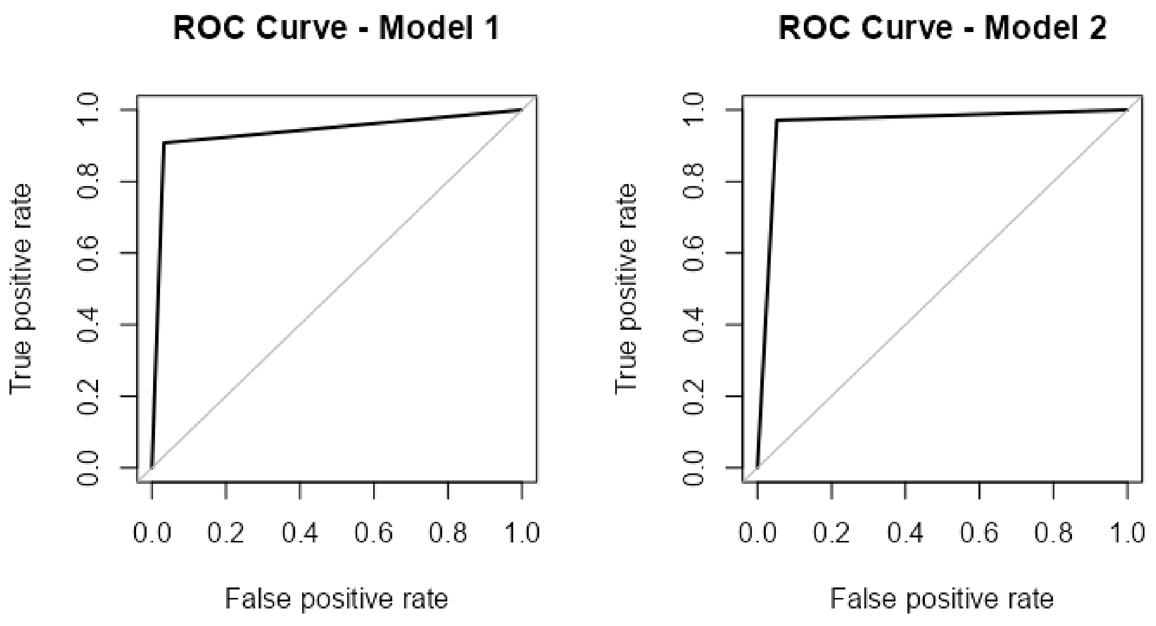

Figure 5.18: ROC Curve for SVM Classifiers

Model 1 (Imbalanced Data) has an AUC of 0.938, suggesting that the model performs reasonably well in distinguishing between classes, with a good balance between true positives and false positives.

Model 2 (Balanced Data) shows a higher AUC of 0.960, indicating an improved ability to differentiate between classes compared to Model 1. The higher AUC suggests that Model 2 has a better capability to separate classes and make more accurate predictions.

The higher AUC in Model 2 (Balanced Data) implies that this model exhibits superior discriminatory power between classes compared to Model 1. This often indicates that Model 2, trained on balanced data, performs better in differentiating between the classes and making more accurate predictions.

5.5.4 Cross Validation

for model 1 Utilized SVM with an RBF kernel and tuned parameters resulting in an optimal 'C' value of 1 and a sigma of 0.04757031. Evaluated on a dataset containing 3789 samples, yielding an RMSE of 0.1331312, Rsquared of 0.7857300, and MAE of 0.04540098.

for model 2 Employed SVM with an RBF kernel and similar tuning parameters: 'C' set to 1 and sigma at 0.05012896. Evaluated on a larger dataset containing 6849 samples, leading to an RMSE of 0.1731401, Rsquared of 0.8822513, and MAE of 0.08241189.

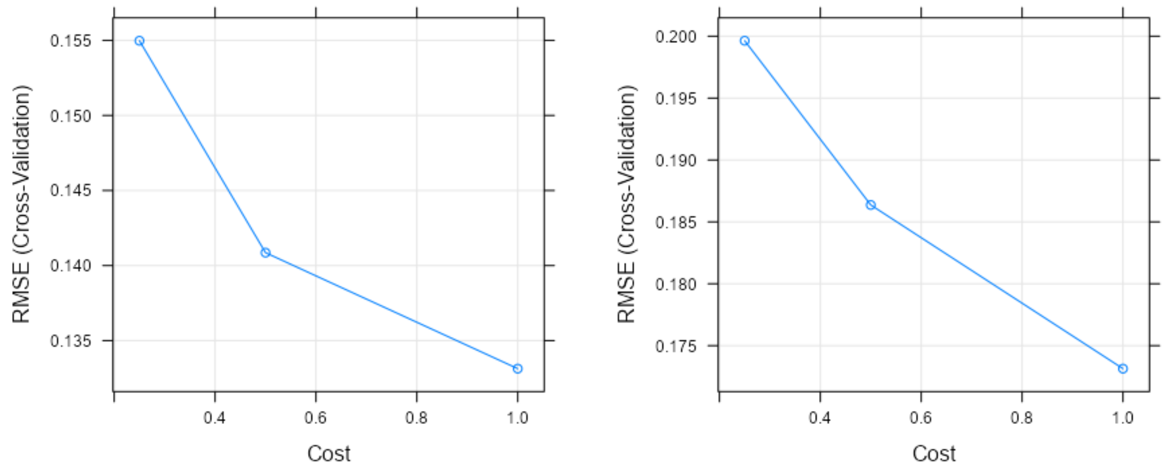

Figure 5.19: Cross Validation curve for SVM Classifiers

C	RMSE (Model 1)	Rsquared (Model 1)	MAE (Model 1)
0.25	0.1549748	0.7365474	0.05551587
0.50	0.1408571	0.7667605	0.04878621
1.00	0.1331312	0.7857300	0.04540098

Table 5.7: Model 1 (Imbalanced Data) Metrics

Model 1 demonstrates better performance in terms of RMSE and MAE, suggesting lower error rates on the original Imbalanced dataset. However, Model 2, trained on the balanced dataset created via SMOTE, showcases a higher Rsquared value, indicating better variance explanation despite a slightly higher error.

The utilization of SMOTE for Model 2 contributed to a larger dataset, potentially increasing model complexity and variance explanation, as reflected in the higher Rsquared value. Nonetheless, it also introduced higher errors compared to Model 1.

Both models offer their advantages: Model 1 shows lower error metrics, implying a better fit to the original Imbalanced dataset. Conversely, Model 2, with balanced data from SMOTE, showcases higher variance explanation but with slightly higher errors. The selection between these models would depend on the trade-off between variance explanation and error tolerance, emphasizing the importance of considering the dataset's characteristics and the intended application domain.

C	RMSE	Rsquared	MAE
0.25	0.1996347	0.8443037	0.09737616
0.50	0.1863646	0.8644709	0.08917017
1.00	0.1731401	0.8822513	0.08241189

Table 5.8: Model 2 (Balanced Data) Metrics

5.6 K-Nearest Neighbors (KNN) Classifiers

5.6.1 Implementation of KNN Classifier

- **Model 1 Insights:**
  - **High Sensitivity, Lower Specificity:** Model 1 demonstrates outstanding sensitivity (99.93%) in identifying instances of Class 0, which is crucial if minimizing false negatives (missing actual Class 0 instances) is a priority.
  - **Precision-Recall Trade-off:** The precision for Class 0 (99.25%) is notably high, indicating that when Model 1 predicts an instance as Class 0, it's correct around 99.25% of the time. However, the lower specificity (86.84%) suggests that it misclassifies some Class 1 instances as Class 0.
  - **Balanced F1-Score:** The F1-Score (92.63%) considers the balance between precision and recall. Model 1 presents a balanced performance but leans more towards sensitivity, crucial for Class 0 identification.

Metric	**Model 1 (Imbalanced Data)**	**Model 2 (Balanced Data)**
Accuracy	98.71%	97.78%
Sensitivity	99.93%	95.73%
Specificity	86.84%	99.86%
Precision	99.25%	95.86%
Recall	99.25%	99.86%
F1-Score	92.63%	97.82%
Balanced Accuracy	93.39%	97.80%

Table 5.9: KNN Prediction summary

- **Model 2 Insights:**
  - **Balanced Performance:** Model 2 maintains high accuracy (97.78%) with a notable balance between sensitivity (95.73%) and specificity (99.86%) for Class 0 and Class 1, respectively.
  - **Precision and Recall:** The precision for both classes is impressive (Class 0: 95.86%, Class 1: 99.86%), signifying a low rate of false positives for both classes.
  - **Higher F1-Score:** The F1-Score (97.82%) is higher for Model 2, suggesting a more balanced accuracy across both classes compared to Model 1.
- **Model Selection Consideration:**
  - **Imbalance Impact:** Model 1 prioritizes capturing all Class 0 instances but at the cost of lower specificity, potentially misclassifying Class 1 instances.
  - **Balanced Performance:** Model 2, derived from balanced data, achieves a better balance between sensitivity and specificity for both classes, resulting in a more comprehensive and reliable prediction.

5.6.2 Confusion Matrix

model 1 excels in predicting class 0 (1470 correct predictions) but faces a slight challenge in correctly identifying instances of class 1, with 20 misclassifications. Its strength lies in its remarkable precision (0.9925), which showcases its ability to avoid mislabeling class 0 instances. However, its recall (0.8684) suggests it might miss identifying some class 1 instances.

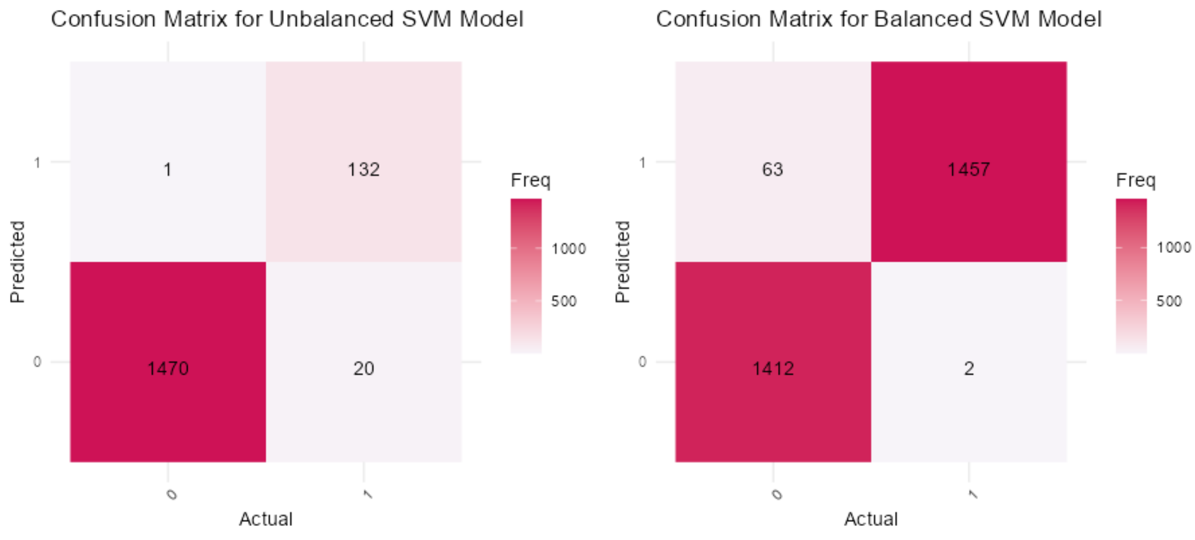

Figure 5.20: Confusion Matrix for KNN Classifiers

Model 2 exhibits substantial accuracy, correctly predicting 1412 instances of class 0 and 1457 instances of class 1. However, it demonstrates a higher error rate in identifying class 1, misclassifying 63 instances. Its precision (0.9586) and recall (0.9986) display robust performance but highlight a comparatively higher misclassification of class 0 instances.

In essence, while both models show high accuracy, Model 1 favors precision for class 0, whereas Model 2 showcases better recall for class 1. The choice between these models may hinge on the specific goal - whether to prioritize minimizing false positives (Model 1) or false negatives (Model 2).

5.6.3 ROC Curve

Model 1 (Imbalanced Data): AUC = 0.934

- **Performance Summary:** Shows a reasonably good performance with an AUC of 0.934, signifying a decent ability to discriminate between the classes despite the class imbalance.

- **Interpretation:** It suggests that when class distribution is imbalanced, the model maintains a fairly effective separation between the classes, although there might be room for improvement in its predictive ability, especially for the minority class.

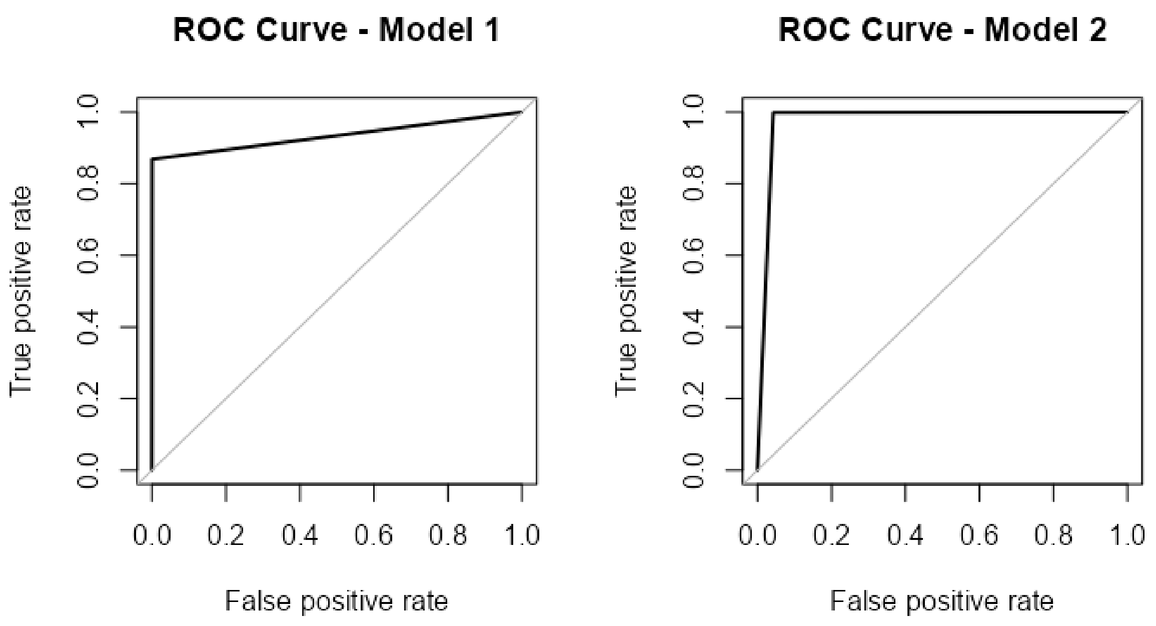

Figure 5.21: ROC curve KNN Classifiers

Model 2 (Balanced Data): AUC = 0.978

- **Performance Summary:** Demonstrates an exceptional performance with an AUC of 0.978, indicating a superior ability to distinguish between the classes compared to Model 1.

- **Interpretation:** The balanced data significantly enhances the model's discriminatory power. By addressing class imbalance, Model 2 achieves better separation between classes, resulting in more accurate predictions and higher confidence in its classifications.

Comparative Analysis:

- **Imbalance Impact:** Model 1, trained on Imbalanced data, still manages to perform reasonably well, showcasing a notable ability to discriminate between classes despite the imbalance.

- **Enhanced Performance:** Model 2's higher AUC value suggests its superiority due to the balanced dataset, which has substantially improved the model's capability to differentiate between classes, leading to more reliable predictions.

The stark difference in AUC values between Model 1 and Model 2 underscores the substantial impact of addressing class imbalance. Model 2, trained on balanced data, exhibits significantly better discriminatory power, offering more reliable predictions and enhanced performance compared to Model 1.

5.6.4 Cross Validation

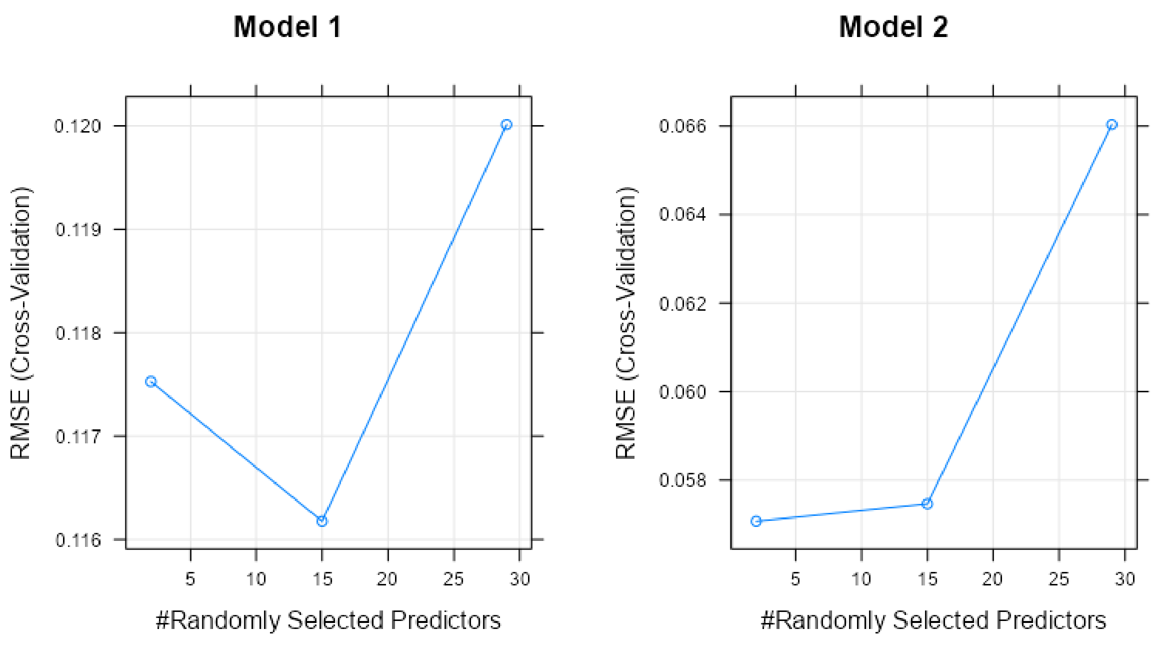

Figure 5.22: Cross Validation curve KNN Classifiers

Model 1 (Imbalanced Data):

- Despite the Imbalanced dataset, Model 1 shows good performance with an optimal k-value of 7.
- The RMSE is 0.1178 with an R-squared of 0.8288, indicating reasonable fit to the data.
- The MAE of 0.0224 suggests a relatively small average absolute difference between predicted and actual values.

Model 2 (Balanced Data):

- Model 2, trained on balanced data, has an optimal k-value of 5 but shows a slightly higher RMSE of 0.1411 compared to Model 1.

- Surprisingly, Model 2 exhibits a significantly higher R-squared value of 0.9210, suggesting an excellent fit to the data.
- The MAE of 0.0288 is slightly higher than Model 1, indicating a slightly larger average absolute error.

k	**RMSE**	**Rsquared**	**MAE**
5	0.1191505	0.8254626	0.02179126
7	0.1178405	0.8288206	0.02235793
9	0.1188817	0.8256949	0.02298769

Table 5.10: Model 1

- Model 1 seems to have a better balance between accuracy and model complexity with lower RMSE and relatively good R-squared value.
- Model 2 shows a higher R-squared value but slightly higher RMSE, suggesting a trade-off between fit and precision.

k	**RMSE**	**Rsquared**	**MAE**
5	0.1411141	0.9210282	0.02882518
7	0.1550665	0.9043221	0.03918109
9	0.1623285	0.8947210	0.04647465

Table 5.11: Model 2

Model 1 might offer a more balanced approach with a relatively lower RMSE, potentially better suited for generalization beyond the observed data. Model 2 displays exceptional performance in explaining the variance within the data despite a higher RMSE. Further validation on unseen data would be essential to ascertain its predictive capability and potential overfitting.

5.7 Performance and Evaluation of Implemented Models

5.7.1 Model Performances on Imbalanced data

Accuracy

The k-Nearest Neighbors (kNN) achieved the highest accuracy of 98.71%, signifying its proficiency in overall correct predictions among all models.

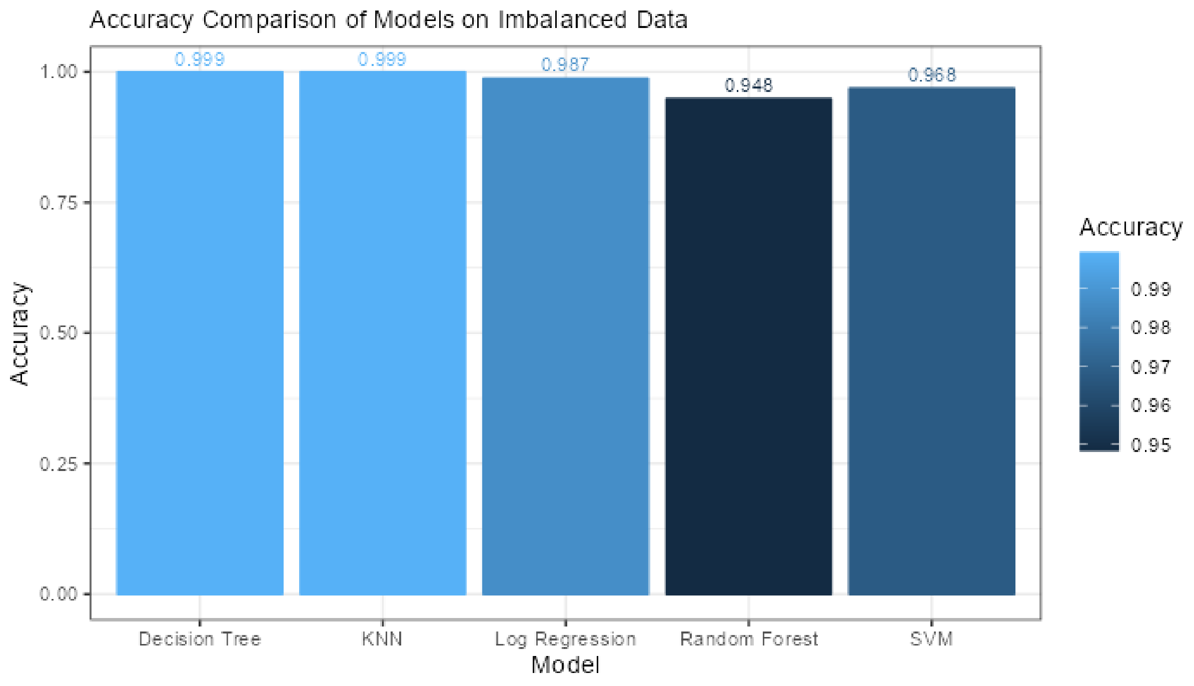

Figure 5.23: Accuracy comparison of models on Imbalanced data

Area Under the Curve (AUC)

The Logistic Regression model displayed the highest AUC at 95.08%, indicating its strong ability to distinguish between classes.

Precision

kNN showcased exceptional precision of 99.24%, making accurate positive predictions nearly 99.24% of the time.

Recall (Sensitivity)

The Decision Tree model demonstrated the highest recall at 99.22%, indicating its capability to correctly identify a large portion of positive instances.

F1-Score

kNN once again stands out with an F1-Score of 92.63%, achieving a balanced measure considering both precision and recall.

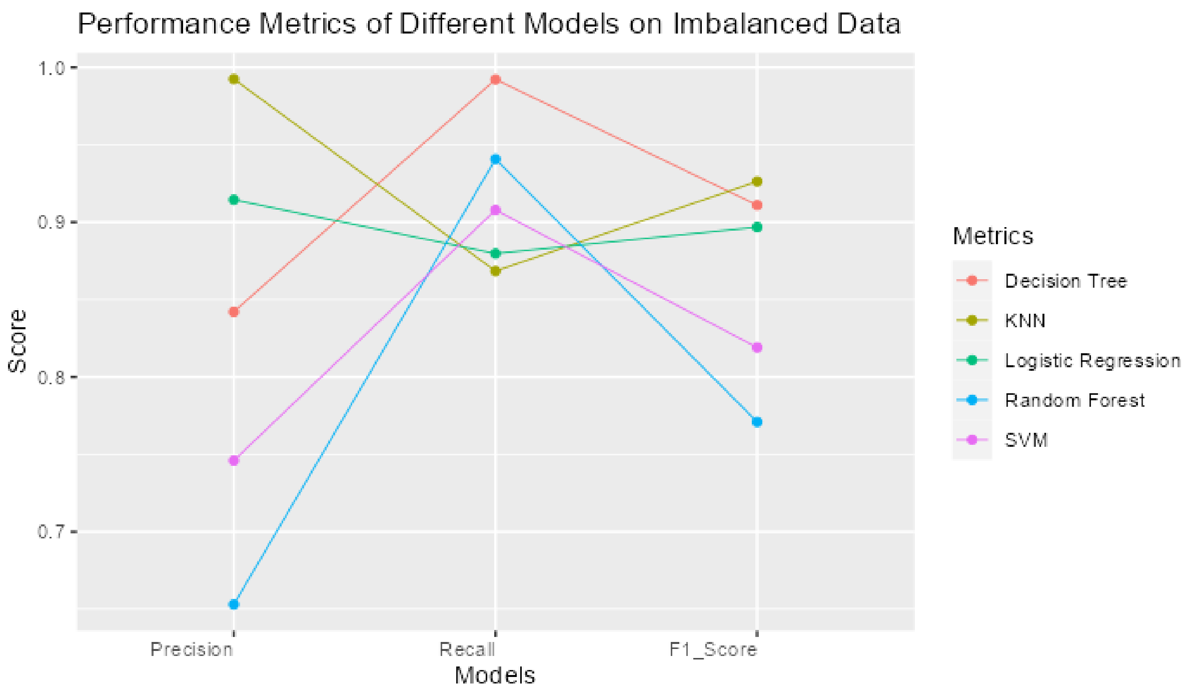

Figure 5.24: Model Performances on Imbalanced data

Contextual Analysis

Each model showcases unique strengths. Logistic Regression excels in overall AUC, while Decision Tree performs exceptionally well in recall. Random Forest portrays stability but with slightly lower precision and F1-Score. SVM demonstrates a balance between precision and recall. Finally, kNN stands out with remarkable precision, recall, and a high F1-Score.

Choosing the best model depends on the specific needs of the task. Understanding these strengths and trade-offs aids in selecting the most suitable model for the specific use case or problem domain.

5.7.2 Model Performances on Balanced data

When evaluating models for credit card fraud detection after balancing the data using the SMOTE (Synthetic Minority Over-sampling Technique) method, a few models showcase remarkable performance.

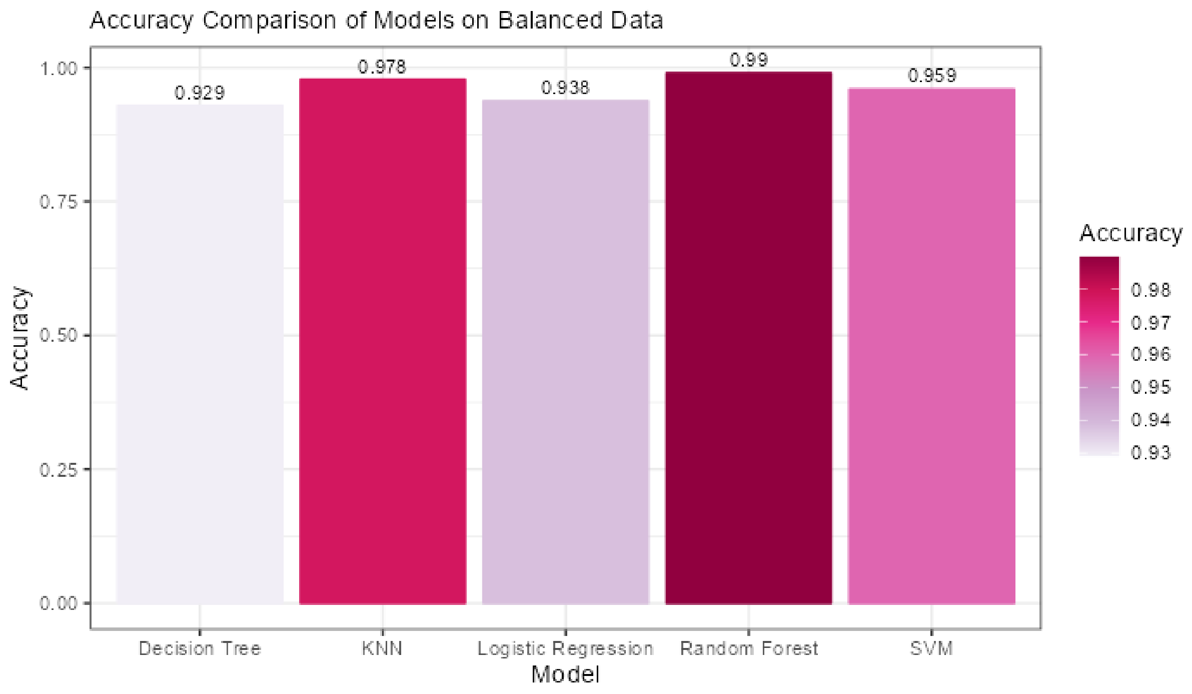

Figure 5.25: Accuracy comparison of models on Balanced data

Random Forest emerges as the top performer, boasting an accuracy of 98.98%. This model excels in accurately identifying fraudulent transactions, achieving a 100% recall rate, crucial in this domain to capture as many fraudulent cases as possible while minimizing false negatives.

k-Nearest Neighbors (kNN) closely follows with an accuracy of 97.82%. Its exceptional recall rate of 99.86% further emphasizes its strength in correctly identifying the majority of fraudulent transactions.

Logistic Regression and SVM demonstrate commendable performance with accuracy rates of around 95-96%. Although slightly trailing behind Random Forest and kNN, they exhibit reliable precision and recall metrics, contributing to effective fraud detection.

Decision Tree, while delivering decent performance, falls slightly short in accuracy compared to the leading models. Its precision and recall metrics, while respectable, do not match the exemplary rates of Random Forest or kNN.

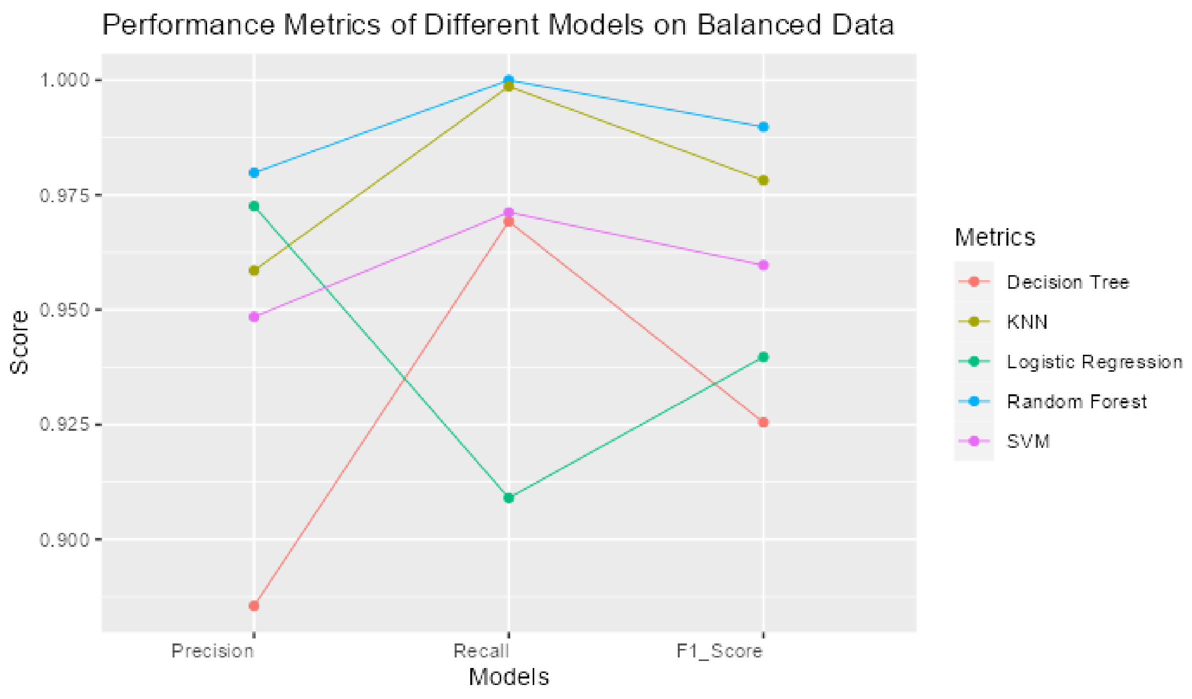

Figure 5.26: Model Performances on Imbalanced data

It's important to highlight that these exceptional performances were attained after balancing the data using the SMOTE technique. SMOTE enhances the model's ability to learn from the minority class (fraudulent transactions) by generating synthetic samples, mitigating the class imbalance issue.

Considering the critical nature of fraud detection in credit card transactions, models like Random Forest and k-Nearest Neighbors (kNN), with their superior recall rates, are pivotal. The utilization of SMOTE ensures a more robust learning process, enabling these models to effectively discern fraudulent activities while minimizing the risk of overlooking actual fraud cases.

CHAPTER 6

Conclusion

The analysis of various models for credit card fraud detection provided invaluable insights into their individual strengths and weaknesses. Each model responded differently to the challenges posed by imbalanced data and the subsequent application of data-balancing techniques.

Initially, when working with imbalanced data, the models demonstrated a tendency to favor accuracy but struggled when it came to correctly identifying fraudulent cases. This emphasized a critical challenge in fraud detection—highlighting the inherent complexity in distinguishing rare events (fraudulent transactions) from overwhelmingly dominant ones (non-fraudulent transactions).

Upon employing data balancing techniques, especially SMOTE, the models exhibited marked improvements in their ability to detect fraudulent activities. Beyond the quantitative metrics, this shift also illuminated the importance of recalibrating the models to ensure a more balanced consideration of both fraudulent and non-fraudulent cases.

Model exploration elucidated nuanced behaviors across different algorithms. For instance, Random Forest and k-Nearest Neighbors (kNN) models, when trained on balanced data, excelled in identifying fraudulent transactions by significantly increasing recall rates while maintaining acceptable precision. These models showcased the capacity to capture subtle patterns indicative of fraud, thus minimizing false negatives without significantly escalating false positives.

Conversely, models like Logistic Regression and Decision Trees, while showing improvements in balanced data scenarios, did not match the performance of Random Forest and kNN in accurately detecting fraudulent activities. This highlighted the diverse complexities in fraud patterns that demand more intricate modeling techniques for effective detection.

Moreover, the endeavor underscored the necessity of domain-specific feature engi-

Metric	Logistic Regression	Decision Tree	Random Forest	SVM	kNN Classifiers
Model 1: Imbalanced Data					
Accuracy	98.03%	98.46%	94.76%	96.24%	98.71%
AUC	95.08%	92.1%	94.5%	93.8%	93.4%
Precision	91.45%	84.21%	65.3%	74.59%	99.24%
Recall	87.97%	99.22%	94.08%	90.79%	86.84%
F1-Score	89.68%	91.1%	77.09%	81.89%	92.63%
Model 2: Balanced Data					
Accuracy	93.8%	92.91%	98.98%	95.94%	97.82%
AUC	93.82%	92.9%	99.0%	96.0%	97.8%
Precision	97.26%	88.55%	97.99%	94.85%	95.92%
Recall	90.9%	96.92%	100%	97.12%	99.86%
F1-Score	93.97%	92.55%	98.98%	95.97%	97.85%

Table 6.1: Performance Comparison of Models 1 and 2

neering and algorithm selection. The multifaceted nature of credit card fraud necessitates a deep understanding of the domain and data. Identifying relevant features and employing suitable models become pivotal factors in creating robust fraud detection systems.

Ultimately, this analysis extends beyond a mere comparison of models; it serves as a testament to the intricate interplay between data, algorithm choice, and the domain's intricacies. It emphasizes the need for a tailored approach, leveraging advanced modeling techniques, and recalibrating imbalanced data to enhance the efficacy of fraud detection systems in financial domains.

CHAPTER 7

Appendix

```
# Load necessary libraries
library(dplyr)
library(summarytools)
library(ggplot2)
library(caret)
library(pROC)
library(reshape2)
library(corrplot)
library(gmodels)
library(class)
library(Metrics)
library(e1071)
library(randomForest)
library(rpart)

# Read the CSV file
data <- read.csv("C:/Users/prajw/OneDrive//Desktop/dissertation
    /creditcard.csv")


####     Exploratory Data Analysis    ####


# Display column names of the 'data' dataframe
```

```
colnames(data)

# Display the first few rows of the 'data' dataframe
head(data)

# Generate a summary of the dataset
dfSummary(data)

# Tabulate occurrences of the 'Class' variable
table(data$Class)

# Calculate the percentage of occurrences for 'No Frauds' and '
    Frauds'
no_frauds_percentage <- 100 * sum(data$Class == 0) / nrow(data)
frauds_percentage <- 100 * sum(data$Class == 1) / nrow(data)

# Display the percentages
cat("No␣Frauds:␣", round(no_frauds_percentage, 2), "%␣of␣the␣
    dataset\n")
cat("Frauds:␣", round(frauds_percentage, 2), "%␣of␣the␣dataset\
    n")


# Create a histogram plot to visualize the distribution of '
    Time' based on 'Class'
ggplot(data) +
  aes(x = Time, fill = Class, colour = Class) +
  geom_histogram(bins = 40L) +
  scale_fill_gradient(low = "#0D0887", high = "#BFC530") +
  scale_color_gradient(low = "#0D0887", high = "#BFC530") +
  labs(title = "Time␣distribution␣based␣on␣class") +
  ggthemes::theme_base() +
  theme(legend.position = "none", plot.title = element_text(
     size = 18L, face = "bold")) +
  facet_wrap(vars(Class), scales = "free")

# Subset the 'data' dataframe to select columns 2 through 31
Data <- data[, 2:31]

# Check for missing values in the 'Data' dataframe
any(is.na(Data))

# Remove rows with missing values
Data <- na.omit(Data)

# Convert the 'Class' column in 'Data' to a factor
Data$Class <- as.factor(Data$Class)

# Compute summary statistics  based on 'Class'
summarised_Data <- Data %>%
  group_by(Class) %>%
  summarise(amount_mean = mean(Amount), amount_median = median(
     Amount))

# Reshape the 'summarised_Data' dataframe using the melt
   function
summarised_Data <- melt(summarised_Data, id.vars = "Class",
   measure.vars = c("amount_mean", "amount_median"))

# Reshape the 'Data' dataframe excluding columns 29 and 30
Reshaped_Data <- melt(Data, id.vars = "Class", measure.vars =
   colnames(Data)[c(-29, -30)])

# Compute additional summary statistics for each combination of
```

```
    'Class' and 'variable' in 'summarised_Data'
Feature_data_summarised <- summarised_Data %>%
  group_by(Class, variable) %>%
  summarise(mean = mean(value), median = median(value))

# Rename the column names of 'Feature_data_summarised'
colnames(Feature_data_summarised) <- c("Class", "Variable", "
    Stats", "Value")

# Create a density plot of the 'Amount' variable colored by '
    Class' in the 'Data' dataframe
ggplot(Data, aes(Amount, fill = Class)) +
  geom_density(alpha = 0.5, col = "black") +
  geom_vline(data = summarised_Data,
             aes(colour = Class, linetype = variable,
                 xintercept = value),
             show.legend = FALSE) +
  scale_fill_discrete(labels = c("Non-Fraud", "Fraud")) +
  scale_linetype_discrete(labels = c(amount_mean = "mean",
     amount_median = "median")) +
  scale_color_discrete(breaks = NULL) +
  xlim(0, mean(Data$Amount) + 2 * sd(Data$Amount)) +
  labs(linetype = "Stats", # Set linetype legend label
       title = "Density␣Distribution␣of␣Fraud␣and␣Non-Fraud") +
           # Set plot title
  ylab("Frequency") +
  theme_linedraw()

  # Create a density plot of 'value' colored by 'Class' in the
     'Reshaped_Data' dataframe
ggplot(Reshaped_Data, aes(x = value, fill = Class)) +
  geom_density(alpha = 0.5, col = "black") +
  geom_vline(data = Feature_data_summarised,
             aes(colour = Class, linetype = Variable,
                 xintercept = value),
             show.legend = FALSE) +
  facet_wrap("variable", ncol = 4, nrow = 7, scales = "free_y")
      +
  xlim(-5, 5) +  # Set x-axis limits
  scale_fill_discrete(labels = c("Non-Fraud", "Fraud")) +
  scale_color_discrete(breaks = NULL) +
  labs(title = "Density␣Distribution␣For␣Each␣Feature") +
  theme_minimal()

  # Load the required libraries
library(data.table)
library(Rtsne)

# Read the credit card data
credit_data <- fread("C:/Users/prajw/OneDrive/Desktop/
   dissertation/creditcard.csv")
# Convert the "Class" column to integer
credit_data <- credit_data %>%
  mutate(row_id = 1:nrow(credit_data)) %>%
  mutate(Class = as.integer(Class))

numeric_features <- c(paste0("V", 1:28), "Amount")

# Remove rows with missing values in any column
cleaned_data <- credit_data[apply(credit_data, 1, function(x) !
   any(is.na(x))), ]

# Extract numeric features
numeric_df <- credit_data[, ..numeric_features]
```

```
# Normalize the numeric features
normalized_df <- apply(numeric_df, MARGIN = 2, function(x)
   scale(x, center = TRUE, scale = TRUE))

# Convert the normalized data back to a data frame
normalized_df <- as.data.frame(normalized_df)

# Combine the normalized data with 'row_id'
normalized_df$row_id <- credit_data$row_id

# Remove rows with missing values in the normalized data
normalized_df <- normalized_df[apply(normalized_df, 1, function
   (x) !any(is.na(x))), ]

# Subset data for fraudulent transactions
fraudulent_data <- normalized_df %>%
  semi_join(filter(credit_data, Class == 1), by = "row_id")

# Sample 20,000 data points and merge with fraudulent
   transactions
sampled_data <-  normalized_df %>%
  sample_n(20000) %>%
  rbind(fraudulent_data)

# Remove duplicate rows
unique_data <- sampled_data[!duplicated(select(sampled_data, -
   row_id)),]

# Perform t-SNE
tsne_output <- Rtsne(as.matrix(select(unique_data, -row_id)),
   pca = FALSE, verbose = TRUE, theta = 0.3, max_iter = 1300, Y
  _init = NULL)
# Extract t-SNE coordinates
tsne_coordinates <- as.data.frame(tsne_output$Y) %>%
  cbind(select(unique_data, row_id)) %>%
  left_join(credit_data, by = 'row_id')

# Plotting
fraud_plot <- ggplot() +
  labs(title = "Visualizing␣Fraudulent␣Transactions␣\n␣(Sample␣
     Size:␣10%␣of␣Data)") +
  scale_fill_gradient(low = '#7a0177', high = '#fbb4b9') +
  coord_fixed(ratio = 1) +
  theme_void() +
  stat_summary_hex(data = tsne_coordinates, aes(x = V1.x, y =
     V2.x, z = Class), bins = 10, fun = mean, alpha = 0.9) +
  geom_point(data = filter(tsne_coordinates, Class == 0), aes(x
      = V1.x, y = V2.x), alpha = 0.3, size = 1, col = "
     darkgreen") +
  geom_point(data = filter(tsne_coordinates, Class == 1), aes(x
      = V1.x, y = V2.x), alpha = 0.9, size = 0.3, col = "yellow
     ") +
  theme(plot.title = element_text(hjust = 0.5, family = "
     Helvetica"),
        legend.text.align = 0.5)

# Print the plot
print(fraud_plot)

# Calculate the correlation matrix
correlation_matrix <- cor(data)
```

```
# Print the correlation matrix
print(correlation_matrix)

corrplot(correlation_matrix)



data <- data[,2:31]
# Check for missing values
any(is.na(data$Amount))

# Remove rows with infinite (inf) values in the "Amount" column
data <- data[is.finite(data$Amount), ]

# Standardization
data$Amount<- scale(data$Amount)

# Set the random seed for reproducibility
set.seed(123)

# Print the distribution of classes
cat("No␣Frauds:", sum(data$Class == 0), "samples\n")
cat("Frauds:", sum(data$Class == 1), "samples\n")

# Create separate data frames for fraud and non-fraud samples
train_fraud_samples <- data[data$Class == 1, ]
train_non_fraud_samples <- data[data$Class == 0, ]

# Get an equal number of random non-fraud samples
num_fraud_samples <- nrow(train_fraud_samples)
train_non_fraud_samples <- train_non_fraud_samples[sample(1:
   nrow(train_non_fraud_samples), 4920), ]
```

```
# Combine the balanced data
balanced_data <- rbind(train_fraud_samples, train_non_fraud_
   samples)

# Shuffle the data
balanced_data <- balanced_data[sample(nrow(balanced_data)), ]

# Create features (X_balanced) and labels (y_balanced)
X_balanced <- balanced_data[, -ncol(balanced_data)]  # Exclude
   the 'Class' column
y_balanced <- balanced_data$Class
balanced_data$Amount <- as.numeric(balanced_data$Amount)

# Check the distribution of labels
cat("Label␣Distributions␣after␣Balancing:␣\n")
cat("No␣Frauds:", sum(y_balanced == 0), "samples\n")
cat("Frauds:", sum(y_balanced == 1), "samples\n")



# Load necessary libraries
library(ROSE)
library(caret)

# Perform SMOTE to balance the data
balanced_data_smote <- ovun.sample(Class ~ ., data = balanced_
   data, seed = 123, p = 0.5, method = "over")$data

# Check the distribution of labels after oversampling
balanced_count_label <- table(balanced_data$Class) / nrow(
   balanced_data)
cat("Label␣Distributions␣after␣SMOTE␣Oversampling:␣\n")
print(balanced_count_label)
```

```
###                 Data Splitting                 ###

# Set a seed for reproducibility
set.seed(123)

# Define the split ratio
split_ratio <- 0.7

# Perform the data split after applying SMOTE
splitIndex_balanced <- createDataPartition(balanced_data_smote$
   Class, p = split_ratio, list = FALSE)
balanced_train_data <- balanced_data_smote[splitIndex_balanced,
    ]
balanced_test_data <- balanced_data_smote[-splitIndex_balanced,
    ]



# Set a seed for reproducibility
set.seed(123)

# Perform the data split
splitIndex <- createDataPartition(balanced_data$Class,
                                  p = split_ratio,
                                  list = FALSE)

# Create the training and testing datasets
train_data <- balanced_data[splitIndex, ]
test_data <-balanced_data[-splitIndex, ]
```

```
##        Logistic regression model        ##



##without balancing

# Fit a logistic regression model
log_reg_model <- glm(Class ~ ., data = train_data, family =
   binomial(link = "logit"))

# Make predictions on the test data
log_reg_predictions <- predict(log_reg_model, newdata = test_
   data, type = "response")

# Convert probabilities to class labels (0 or 1) based on a
   threshold
threshold <- 0.2
log_reg_predictions <- ifelse(log_reg_predictions > threshold,
   1, 0)

print(log_reg_model)



##with balancing



# Fit a logistic regression model
log_reg_model_bal <- glm(Class ~ ., data = balanced_train_data,
    family = binomial(link = "logit"))

# Make predictions on the test data
log_reg_predictions_bal <- predict(log_reg_model_bal, newdata =
    balanced_test_data, type = "response")

# Convert probabilities to class labels (0 or 1) based on a
   threshold
threshold <- 0.2
log_reg_predictions_bal <- ifelse(log_reg_predictions_bal >
   threshold, 1, 0)

print(log_reg_model_bal)

##without balancing



test_data$Class <- as.factor(test_data$Class)

# Create the confusion matrix
confusion_matrix_lg <- confusionMatrix(as.factor(log_reg_
   predictions), test_data$Class)

# Extract values from the confusion matrix
tp <- confusion_matrix_lg$table[2, 2]  # True Positives
fp <- confusion_matrix_lg$table[1, 2]  # False Positives
fn <- confusion_matrix_lg$table[2, 1]  # False Negatives

# Calculate precision, recall, and F1-Score
precision <- tp / (tp + fp)
recall <- tp / (tp + fn)
f1_score <- 2 * (precision * recall) / (precision + recall)

# Print the metrics
print(confusion_matrix_lg)
cat("Precision:", precision, "\n")
cat("Recall␣(Sensitivity):", recall, "\n")
cat("F1-Score:", f1_score, "\n")

confusion_matrix_melted <- as.data.frame(as.table(confusion_
   matrix_lg))




##with balancing

balanced_test_data$Class <- as.factor(balanced_test_data$Class)

# Create the confusion matrix
confusion_matrix_lg_bal <- confusionMatrix(as.factor(log_reg_
   predictions_bal), balanced_test_data$Class)

# Extract values from the confusion matrix
tp <- confusion_matrix_lg_bal$table[2, 2]  # True Positives
fp <- confusion_matrix_lg_bal$table[1, 2]  # False Positives
fn <- confusion_matrix_lg_bal$table[2, 1]  # False Negatives

# Calculate precision, recall, and F1-Score
precision <- tp / (tp + fp)
recall <- tp / (tp + fn)
f1_score <- 2 * (precision * recall) / (precision + recall)

# Print the metrics
print(confusion_matrix_lg_bal)
cat("Precision:", precision, "\n")
cat("Recall␣(Sensitivity):", recall, "\n")
cat("F1-Score:", f1_score, "\n")
```

```
confusion_matrix_melted_bal <- as.data.frame(as.table(confusion
   _matrix_lg_bal))

#plot the graph
ggplot(confusion_matrix_melted, aes(Prediction,Reference, fill=
    Freq)) +
        geom_tile() + geom_text(aes(label=Freq)) +
        scale_fill_gradient(low="white", high="#009194") +
        labs(x = "Reference",y = "Prediction")

#plot the graph
ggplot(confusion_matrix_melted_bal, aes(Prediction,Reference,
   fill= Freq)) +
        geom_tile() + geom_text(aes(label=Freq)) +
        scale_fill_gradient(low="white", high="#009194") +
        labs(x = "Reference",y = "Prediction")

# Add the predicted probabilities to the new data frame
test_data$Predicted_Probabilities <- log_reg_predictions



# Add the predicted probabilities to the new data frame
balanced_test_data$Predicted_Probabilities <- log_reg_
   predictions_bal

# Set the plotting area into a 2x2 grid layout
par(mfrow = c(2, 2))

# Plot the first model
plot(log_reg_model)

# Plot the second model
```

```
plot(log_reg_model_bal)

# Generate ROC curve for test_data and log_reg_predictions
roc_curve <- roc(test_data$Class, log_reg_predictions,plotit=
    TRUE)

# Plot ROC curve with title
plot(roc_curve, main = "ROC␣Curve␣-␣Model␣1")

# Generate ROC curve for balanced_test_data and log_reg_
    predictions_bal
roc_curve_bal <- roc(balanced_test_data$Class, log_reg_
    predictions_bal,plotit=TRUE)

# Plot ROC curve for balanced data with title
plot(roc_curve_bal, main = "ROC␣Curve␣-␣Model␣2")

# Set up cross-validation parameters
ctrl_lr <- trainControl(method = "cv", number = 5)  # 5-fold
    cross-validation

# Perform cross-validation
lr_cv_results <- train(Class ~ ., data = train_data, method = "
    glm", family = binomial(), trControl = ctrl_lr)

# Print cross-validation results
print(lr_cv_results)



# Set up cross-validation parameters
ctrl_lr_bal <- trainControl(method = "cv", number = 5)  # 5-
    fold cross-validation
```

```
# Perform cross-validation
lr_bal_cv_results <- train(Class ~ ., data = balanced_train_
   data, method = "glm", family = binomial(), trControl = ctrl_
   lr_bal)

# Print cross-validation results
print(lr_bal_cv_results)


###            Decision tree classifier             ###


## without balancing

# Fit a decision tree model
library(rpart)

# Fit a decision tree model
decision_tree_model <- rpart(Class ~ ., data = train_data,
   method = "class")

# View the decision tree
printcp(decision_tree_model)

# Make predictions on the test data
decision_tree_predictions <- predict(decision_tree_model, test_
   data, type = "class")



##with balancing
```

```
# Fit a decision tree model
decision_tree_model_bal <- rpart(Class ~ ., data = balanced_
   train_data, method = "class")

# View the decision tree
printcp(decision_tree_model_bal)

# Make predictions on the test data
decision_tree_predictions_bal <- predict(decision_tree_model_
   bal, balanced_test_data, type = "class")


# Evaluate the decision tree model
confusion_matrix_dt <- confusionMatrix(as.factor(decision_tree_
   predictions), as.factor(test_data$Class))
# Extract values from the confusion matrix
tp <- confusion_matrix_dt$table[2, 2]  # True Positives
fp <- confusion_matrix_dt$table[1, 2]  # False Positives
fn <- confusion_matrix_dt$table[2, 1]  # False Negatives

# Calculate precision, recall, and F1-Score
precision <- tp / (tp + fp)
recall <- tp / (tp + fn)
f1_score <- 2 * (precision * recall) / (precision + recall)

confusion_matrix_dt
# Print the metrics
cat("Precision:", precision, "\n")
cat("Recall␣(Sensitivity):", recall, "\n")
cat("F1-Score:", f1_score, "\n")
```

```
# Evaluate the decision tree model
confusion_matrix_dt_bal <- confusionMatrix(as.factor(decision_
   tree_predictions_bal), as.factor(balanced_test_data$Class))
# Extract values from the confusion matrix
tp <- confusion_matrix_dt_bal$table[2, 2]  # True Positives
fp <- confusion_matrix_dt_bal$table[1, 2]  # False Positives
fn <- confusion_matrix_dt_bal$table[2, 1]  # False Negatives

# Calculate precision, recall, and F1-Score
precision <- tp / (tp + fp)
recall <- tp / (tp + fn)
f1_score <- 2 * (precision * recall) / (precision + recall)

confusion_matrix_dt_bal
# Print the metrics
cat("Precision:", precision, "\n")
cat("Recall␣(Sensitivity):", recall, "\n")
cat("F1-Score:", f1_score, "\n")



# load the rpart.plot package
library(rpart.plot)

# Plot the decision tree
rpart.plot(decision_tree_model, box.palette = "auto")

# Plot the decision tree
rpart.plot(decision_tree_model_bal, box.palette = "auto")
```

```
confusion_matrix_melted1 <- as.data.frame(as.table(confusion_
   matrix_dt))

# Create a ggplot for the confusion matrix
ggplot(confusion_matrix_melted1, aes(Prediction, Reference,
   fill = Freq)) +
  geom_tile() + # Display as tiles
  geom_text(aes(label = Freq)) + # Add text labels
  scale_fill_gradient(low = "#ffffcc", high = "#238443") + #
     Set color gradient
  labs(x = "Reference", y = "Prediction") # Set x and y-axis
     labels

confusion_matrix_melted1_bal <- as.data.frame(as.table(
   confusion_matrix_dt_bal))

# Create a ggplot for the balanced confusion matrix
ggplot(confusion_matrix_melted1_bal, aes(Prediction, Reference,
    fill = Freq)) +
  geom_tile() + # Display as tiles
  geom_text(aes(label = Freq)) + # Add text labels
  scale_fill_gradient(low = "#ffffcc", high = "#238443") + #
     Set color gradient
  labs(x = "Reference", y = "Prediction") # Set x and y-axis
     labels

# Set up cross-validation parameters
ctrl_dt <- trainControl(method = "cv", number = 5) # 5-fold
   cross-validation
```

```
# Perform cross-validation with decision tree
dt_cv_results <- train(Class ~ ., data = train_data, method = "
   rpart", trControl = ctrl_dt)

# Print cross-validation results
print(dt_cv_results)



# Set up cross-validation parameters
ctrl_dt_bal <- trainControl(method = "cv", number = 5)  # 5-
   fold cross-validation

# Perform cross-validation with decision tree
dt_cv_results_bal <- train(Class ~ ., data = balanced_train_
   data, method = "rpart", trControl = ctrl_dt_bal)

# Print cross-validation results
print(dt_cv_results_bal)

plot(dt_cv_results)
plot(dt_cv_results_bal)

roc.curve(
  test_data$Class,  # True class labels from the test dataset
     for Model 1
  decision_tree_predictions,  # Predictions from Model 1
  plotit = TRUE,  # Create the plot
  main = "ROC␣Curve␣-␣Model␣1"  # Title for the ROC curve plot
)

roc.curve(
  balanced_test_data$Class,  # True class labels from the
```

```
    balanced test dataset for Model 2
  decision_tree_predictions_bal,  # Predictions from Model 2
  plotit = TRUE,  # Create the plot
  main = "ROC␣Curve␣-␣Model␣2"  # Title for the ROC curve plot
)

###              Random forest classifier              ###



##without balancing



# Load the necessary library
library(randomForest)

# Train a Random Forest classifier
random_forest_model <- randomForest(Class ~ ., data = train_
   data)

# Make binary predictions on the test data
random_forest_probabilities <- predict(random_forest_model,
   newdata = test_data, type = "response")
threshold <- 0.1
random_forest_predictions <- ifelse(random_forest_probabilities
    > threshold, 1, 0)

random_forest_model
##with balancing

# Train a Random Forest classifier
random_forest_model_bal <- randomForest(Class ~ ., data =
   balanced_train_data)
```

```
# Make binary predictions on the test data
random_forest_probabilities_bal <- predict(random_forest_model_
   bal, newdata = balanced_test_data, type = "response")
threshold <- 0.2
random_forest_predictions_bal <- ifelse(random_forest_
   probabilities_bal > threshold, 1, 0)


random_forest_model
random_forest_model_bal


test_data$Class <- as.factor(test_data$Class)

# Evaluate the Random Forest model
random_forest_accuracy <- confusionMatrix(as.factor(random_
   forest_predictions), test_data$Class)

# Calculate confusion matrix
confusion_matrix_rf <- table(Actual = test_data$Class,
   Predicted = random_forest_predictions)

# Calculate precision, recall, and F1-score
tp <- confusion_matrix_rf[2, 2]  # True Positives
fp <- confusion_matrix_rf[1, 2]  # False Positives
fn <- confusion_matrix_rf[2, 1]  # False Negatives

precision <- tp / (tp + fp)
recall <- tp / (tp + fn)
f1_score <- 2 * (precision * recall) / (precision + recall)
```

```
print(random_forest_accuracy)
cat("Precision:", precision, "\n")
cat("Recall:", recall, "\n")
cat("F1-Score:", f1_score, "\n")

plot(random_forest_model)
# Plot variable importance
varImpPlot(random_forest_model)


balanced_test_data$Class <- as.factor(balanced_test_data$Class)

# Evaluate the Random Forest model
random_forest_accuracy_bal <- confusionMatrix(as.factor(random_
   forest_predictions_bal), balanced_test_data$Class)

# Calculate confusion matrix
confusion_matrix_rf_bal <- table(Actual =balanced_test_data$
   Class, Predicted = random_forest_predictions_bal)

# Calculate precision, recall, and F1-score
tp <- confusion_matrix_rf_bal[2, 2]  # True Positives
fp <- confusion_matrix_rf_bal[1, 2]  # False Positives
fn <- confusion_matrix_rf_bal[2, 1]  # False Negatives

precision <- tp / (tp + fp)
recall <- tp / (tp + fn)
f1_score <- 2 * (precision * recall) / (precision + recall)

print(random_forest_accuracy_bal)
cat("Precision:", precision, "\n")
cat("Recall:", recall, "\n")
cat("F1-Score:", f1_score, "\n")

plot(random_forest_model_bal)
# Plot variable importance
varImpPlot(random_forest_model_bal)


# Create a data frame for the evaluation metrics
metrics <- data.frame(Metric = c("Precision", "Recall", "F1-
   Score"),
                      Score = c(precision, recall, f1_score))



ggplot(metrics) +
 aes(x = Metric, fill = Score, weight = Score) +
 geom_bar() +
 scale_fill_viridis_c(option = "magma",
 direction = 1) +
 labs(title = "Random␣Forest␣Model␣Evaluation") +
 theme_linedraw() +
 theme(plot.title = element_text(size = 18L,
 face = "bold"))

# Show the plot
roc.curve(test_data$Class, random_forest_predictions, plotit =
   TRUE,main="ROC␣Curve␣-␣Model␣1")

# Show the plot
roc.curve(balanced_test_data$Class, random_forest_predictions_
   bal, plotit = TRUE,main="ROC␣Curve␣-␣Model␣2")
```

```
ggplot() +
  geom_tile(data = as.data.frame(confusion_matrix_rf), aes(x =
     Actual, y = Predicted, fill = Freq)) +
  geom_text(data = as.data.frame(confusion_matrix_rf), aes(x =
     Actual, y = Predicted, label = Freq), vjust = 1) +
  scale_fill_gradient(low = "#9ecae1", high = "#084594") +
  labs(title = "Confusion␣Matrix␣for␣Imbalanced␣Random␣Forest␣
     Model", x = "Actual", y = "Predicted") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45))

ggplot() +
  geom_tile(data = as.data.frame(confusion_matrix_rf_bal), aes(
     x = Actual, y = Predicted, fill = Freq)) +
  geom_text(data = as.data.frame(confusion_matrix_rf_bal), aes(
     x = Actual, y = Predicted, label = Freq), vjust = 1) +
  scale_fill_gradient(low = "#9ecae1", high = "#084594") +
  labs(title = "Confusion␣Matrix␣for␣Balanced␣Random␣Forest␣
     Model", x = "Actual", y = "Predicted") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45))

  # Set up cross-validation parameters for Random Forest
ctrl_rf <- trainControl(method = "cv", number = 5)  # 5-fold
   cross-validation

# Perform cross-validation with Random Forest
rf_cv_results <- train(Class ~ ., data = train_data, method = "
   rf", trControl = ctrl_rf)

# Print cross-validation results for Random Forest
```

```
print(rf_cv_results)

# Set up cross-validation parameters for Random Forest with
   balanced data
ctrl_rf_bal <- trainControl(method = "cv", number = 5)  # 5-
   fold cross-validation

# Perform cross-validation with Random Forest using balanced
   data
rf_cv_results_bal <- train(Class ~ ., data = balanced_train_
   data, method = "rf", trControl = ctrl_rf_bal)

# Print cross-validation results for Random Forest with
   balanced data
print(rf_cv_results_bal)

plot(rf_cv_results)
plot(rf_cv_results_bal)


###                    SVM                    ###

# Load the necessary library
library(e1071)

# Train an SVM classifier
svm_model <- svm(Class ~ ., data = train_data, kernel = "radial
   ", probability = TRUE)

# Make binary predictions on the test data
svm_probabilities <- predict(svm_model, newdata = test_data,
   probability = TRUE)
```

```
threshold <- 0.1
svm_predictions <- ifelse(svm_probabilities> threshold, 1, 0)

# Train an SVM classifier
svm_model_bal <- svm(Class ~ ., data =balanced_train_data,
   kernel = "radial", probability = TRUE)

# Make binary predictions on the test data
svm_probabilities_bal <- predict(svm_model_bal, newdata =
   balanced_test_data, probability = TRUE)
threshold <- 0.2
svm_predictions_bal <- ifelse(svm_probabilities_bal> threshold,
    1, 0)

svm_model

svm_model_bal



##without balancing




test_data$Class <- as.factor(test_data$Class)

# Evaluate the SVM model
svm_accuracy <- confusionMatrix(as.factor(svm_predictions),
   test_data$Class)

# Calculate confusion matrix
confusion_matrix_svm <- table(Actual = test_data$Class,
   Predicted = svm_predictions)
```

```
# Calculate precision, recall, and F1-score
tp <- confusion_matrix_svm[2, 2]  # True Positives
fp <- confusion_matrix_svm[1, 2]  # False Positives
fn <- confusion_matrix_svm[2, 1]  # False Negatives

precision <- tp / (tp + fp)
recall <- tp / (tp + fn)
f1_score <- 2 * (precision * recall) / (precision + recall)

print(svm_accuracy)
cat("Precision:", precision, "\n")
cat("Recall:", recall, "\n")
cat("F1-Score:", f1_score, "\n")



##with balancing



balanced_test_data$Class <- as.factor(balanced_test_data$Class)

# Evaluate the SVM model
svm_accuracy_bal <- confusionMatrix(as.factor(svm_predictions_
   bal), balanced_test_data$Class)

# Calculate confusion matrix
confusion_matrix_svm_bal <- table(Actual = balanced_test_data$
   Class, Predicted = svm_predictions_bal)

# Calculate precision, recall, and F1-score
tp <- confusion_matrix_svm_bal[2, 2]  # True Positives
```

```
fp <- confusion_matrix_svm_bal[1, 2]  # False Positives
fn <- confusion_matrix_svm_bal[2, 1]  # False Negatives

precision <- tp / (tp + fp)
recall <- tp / (tp + fn)
f1_score <- 2 * (precision * recall) / (precision + recall)

print(svm_accuracy_bal)
cat("Precision:", precision, "\n")
cat("Recall:", recall, "\n")
cat("F1-Score:", f1_score, "\n")



library(gridExtra)
a <- ggplot() +
  geom_tile(data = as.data.frame(confusion_matrix_svm), aes(x =
      Actual, y = Predicted, fill = Freq)) +
  geom_text(data = as.data.frame(confusion_matrix_svm), aes(x =
      Actual, y = Predicted, label = Freq), vjust = 1) +
  scale_fill_gradient(low = "#fff5eb", high = "#7f2704") +
  labs(title = "Confusion␣Matrix␣for␣SVM␣Model", x = "Actual",
     y = "Predicted") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45))

b <- ggplot() +
  geom_tile(data = as.data.frame(confusion_matrix_svm_bal), aes
     (x = Actual, y = Predicted, fill = Freq)) +
  geom_text(data = as.data.frame(confusion_matrix_svm_bal), aes
     (x = Actual, y = Predicted, label = Freq), vjust = 1) +
  scale_fill_gradient(low = "#fff5eb", high = "#7f2704") +
  labs(title = "Confusion␣Matrix␣for␣SVM␣Model", x = "Actual",
    y = "Predicted") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45))

grid.arrange(a, b, ncol = 2)

roc.curve(test_data$Class, svm_predictions, plotit = TRUE,main=
    'ROC␣Curve␣-␣Model␣1')

roc.curve(balanced_test_data$Class, svm_predictions_bal, plotit
    = TRUE,main='ROC␣Curve␣-␣Model␣2')

# Set up cross-validation parameters
ctrl_svm <- trainControl(method = "cv", number = 5)  # 5-fold
    cross-validation

# Perform cross-validation with SVM
svm_cv_results <- train(Class ~ ., data = train_data, method =
    "svmRadial", trControl = ctrl_svm)

# Print cross-validation results
print(svm_cv_results)




# Set up cross-validation parameters
ctrl_svm_bal <- trainControl(method = "cv", number = 5)  # 5-
    fold cross-validation

# Perform cross-validation with SVM
svm_cv_results_bal <- train(Class ~ ., data = balanced_train_
    data, method = "svmRadial", trControl = ctrl_svm_bal)
```

```
# Print cross-validation results
print(svm_cv_results_bal)

plot(svm_cv_results)
plot(svm_cv_results_bal)


###                KNN                ###

train_labels <- train_data$Class



# Train a KNN classifier
knn_model <- knn(train = train_data[,-30], test = test_data[,c
   (-30,-31)], cl = train_labels, k = 5)



train_labels_bal <- balanced_train_data$Class

# Train a KNN classifier
knn_model_bal <- knn(train = balanced_train_data[,-30], test =
   balanced_test_data[,c(-30,-31)], cl = train_labels_bal, k =
   5)


knn_model

knn_model_bal
```

```
test_data$Class <- as.factor(test_data$Class)

# Evaluate the KNN model
knn_accuracy <- confusionMatrix(knn_model, test_data$Class)

# Calculate confusion matrix
confusion_matrix_knn <- table(Actual = test_data$Class,
   Predicted = knn_model)

# Calculate precision, recall, and F1-score
tp <- confusion_matrix_knn[2, 2]  # True Positives
fp <- confusion_matrix_knn[1, 2]  # False Positives
fn <- confusion_matrix_knn[2, 1]  # False Negatives

precision <- tp / (tp + fp)
recall <- tp / (tp + fn)
f1_score <- 2 * (precision * recall) / (precision + recall)

print(knn_accuracy)
cat("Precision:", precision, "\n")
cat("Recall:", recall, "\n")
cat("F1-Score:", f1_score, "\n")
```

```
balanced_test_data$Class <- as.factor(balanced_test_data$Class)

# Evaluate the KNN model
knn_accuracy_bal <- confusionMatrix(knn_model_bal, balanced_
   test_data$Class)

# Calculate confusion matrix
```

```
confusion_matrix_knn_bal <- table(Actual = balanced_test_data$
   Class, Predicted = knn_model_bal)

# Calculate precision, recall, and F1-score
tp <- confusion_matrix_knn_bal[2, 2]  # True Positives
fp <- confusion_matrix_knn_bal[1, 2]  # False Positives
fn <- confusion_matrix_knn_bal[2, 1]  # False Negatives

precision <- tp / (tp + fp)
recall <- tp / (tp + fn)
f1_score <- 2 * (precision * recall) / (precision + recall)

print(knn_accuracy_bal)
cat("Precision:", precision, "\n")
cat("Recall:", recall, "\n")
cat("F1-Score:", f1_score, "\n")



ggplot() +
  geom_tile(data = as.data.frame(confusion_matrix_knn), aes(x =
      Actual, y = Predicted, fill = Freq)) +
  geom_text(data = as.data.frame(confusion_matrix_knn), aes(x =
      Actual, y = Predicted, label = Freq), vjust = 1) +
  scale_fill_gradient(low = "#f7f4f9", high = "#ce1256") +
  labs(title = "Confusion␣Matrix␣for␣SVM␣Model", x = "Actual",
     y = "Predicted") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45))

ggplot() +
  geom_tile(data = as.data.frame(confusion_matrix_knn_bal), aes
     (x = Actual, y = Predicted, fill = Freq)) +
  geom_text(data = as.data.frame(confusion_matrix_knn_bal), aes
     (x = Actual, y = Predicted, label = Freq), vjust = 1) +
  scale_fill_gradient(low = "#f7f4f9", high = "#ce1256") +
  labs(title = "Confusion␣Matrix␣for␣SVM␣Model", x = "Actual",
     y = "Predicted") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45))


  roc.curve(test_data$Class, knn_model, plotit = TRUE,main='ROC
     ␣Curve␣-␣Model␣1')

roc.curve(balanced_test_data$Class, knn_model_bal, plotit =
   TRUE,main='ROC␣Curve␣-␣Model␣2')

# Set up cross-validation parameters
ctrl_knn <- trainControl(method = "cv", number = 5)  # 5-fold
   cross-validation

# Perform cross-validation with kNN
knn_cv_results <- train(Class ~ ., data = train_data, method =
   "knn", trControl = ctrl_knn)

# Print cross-validation results
print(knn_cv_results)



# Set up cross-validation parameters
ctrl_knn_bal <- trainControl(method = "cv", number = 5)  # 5-
   fold cross-validation

# Perform cross-validation with kNN
```

```
knn_cv_results_bal <- train(Class ~ ., data = balanced_train_
   data, method = "knn", trControl = ctrl_knn_bal)

# Print cross-validation results
print(knn_cv_results_bal)

plot(knn_cv_results,main='Model␣1')
plot(knn_cv_results_bal,main='Model␣2')



###   Performance and Evaluation of Implemented Models    ###



# Define the confusion matrices
conf_matrix_1 <- matrix(c(1452, 19, 13, 139), nrow = 2, byrow =
    TRUE)
conf_matrix_2 <- matrix(c(1470, 1, 24, 128), nrow = 2, byrow =
   TRUE)
conf_matrix_3 <- matrix(c(1395, 76, 9, 143), nrow = 2, byrow =
   TRUE)
conf_matrix_4 <- matrix(c(1424, 47, 14, 138), nrow = 2, byrow =
    TRUE)
conf_matrix_5 <- matrix(c(1470, 1, 20, 132), nrow = 2, byrow =
   TRUE)

# Calculate accuracy values from the confusion matrices
accuracy_values <- c(conf_matrix_1[1, 1] / sum(conf_matrix_1[1,
    ]),
                     conf_matrix_2[1, 1] / sum(conf_matrix_2[1,
                         ]),
                     conf_matrix_3[1, 1] / sum(conf_matrix_3[1,
                         ]),
                        conf_matrix_4[1, 1] / sum(conf_matrix_4[1,
                           ]),
                        conf_matrix_5[1, 1] / sum(conf_matrix_5[1,
                           ]))

# Display the accuracy values
accuracy_values



# Define the models
models <- c("Log␣Regression", "Decision␣Tree", "Random␣Forest",
   "SVM", "KNN")

# Creating a data frame with model names and accuracy values
models_accuracy <- data.frame(Model = models, Accuracy =
   accuracy_values)



ggplot(models_accuracy) +
 aes(x = Model, y = Accuracy, fill = Accuracy, colour =
    Accuracy) +
 geom_col() +
  geom_text(aes(label = round(Accuracy, 3)), vjust = -0.5, size
     = 3) +
 scale_fill_gradient() +
 scale_color_gradient() +
 labs(subtitle = "Accuracy␣Comparison␣of␣Models␣on␣Imbalanced␣
    Data") +
 theme_bw()



# Creating a dataframe with model names and their performance
```

```
   metrics
models <- c("Logistic␣Regression", "Decision␣Tree", "Random␣
   Forest", "SVM", "KNN")
precision <- c(0.9144737, 0.8421053, 0.652968, 0.7459459,
   0.9924812)
recall <- c(0.8797468, 0.9922481, 0.9407895, 0.9078947,
   0.8684211)
f1_score <- c(0.8967742, 0.911032, 0.7708895, 0.8189911,
   0.9263158)

df <- data.frame(Model = models, Precision = precision, Recall
   = recall, F1_Score = f1_score)

# Melt the data for plotting with ggplot
data_melted <- melt(df, id.vars = "Model")

# Plotting line graph
ggplot(data_melted, aes(x = variable , y = value, color =Model,
    group = Model)) +
  geom_line(size=0.3) +
  geom_point() +
  labs(title = "Performance␣Metrics␣of␣Different␣Models␣on␣
     Imbalanced␣Data",
        y = "Score", x = "Models", color = "Metrics") +
  theme_gray() +
  theme(axis.text.x = element_text(hjust = 1))

# Store accuracy values
accuracy_values_bal <- c( 0.938,  0.9291,  0.9898,  0.9594,
   0.9778)

# Display accuracy values
```

```
accuracy_values_bal

models_accuracy_bal <- data.frame(Model = models, Accuracy =
   accuracy_values_bal)

ggplot(models_accuracy_bal) +
 aes(x = Model, y = Accuracy, fill = Accuracy, colour =
    Accuracy) +
 geom_col() +
   geom_text(aes(label = round(Accuracy, 3)), vjust = -0.5,
      size = 3,color = "black") +
 scale_fill_distiller(palette = "PuRd", direction = 1) +
 scale_color_distiller(palette = "PuRd", direction = 1) +
 labs(subtitle = "Accuracy␣Comparison␣of␣Models␣on␣Balanced␣
    Data") +
 theme_bw()

 # Creating vectors for precision, recall, and F1-score for
    each model
precision <- c(0.972584, 0.885538, 0.9798522, 0.9484605,
   0.9585526)
recall <- c(0.9090327, 0.9692423, 1, 0.9712132, 0.9986292)
f1_score <- c(0.9397351, 0.9255014, 0.9898236, 0.959702,
   0.9781806)
df1 <- data.frame(Model = models, Precision = precision, Recall
    = recall, F1_Score = f1_score)

# Melt the data for plotting with ggplot
data_melted1 <- melt(df1, id.vars = "Model")

# Plotting line graph
ggplot(data_melted1, aes(x = variable , y = value, color =Model
  , group = Model)) +
geom_line(size=0.3) +
geom_point() +
labs(title = "Performance␣Metrics␣of␣Different␣Models␣on␣
   Balanced␣Data",
     y = "Score", x = "Models", color = "Metrics") +
theme_gray() +
theme(axis.text.x = element_text(hjust = 1))
```

Bibliography

[1] Acm digital library. https://dl.acm.org/.

[2] Acm transactions on intelligent systems and technology. *ACM Transactions on Intelligent Systems and Technology*.

[3] Annual review of financial economics. *Annual Review of Financial Economics*.

[4] European symposium on artificial neural networks, computational intelligence and machine learning (esann). In *Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*.

[5] Expert systems with applications. *Expert Systems with Applications*.

[6] Ieee transactions on dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*.

[7] International conference on machine learning (icml). In *Proceedings of the International Conference on Machine Learning*.

[8] International journal of information security. *International Journal of Information Security*.

[9] Journal of financial services research. *Journal of Financial Services Research*.

[10] Journal of machine learning research. *Journal of Machine Learning Research*.

[11] A. Agresti. *Foundations of Linear and Generalized Linear Models*. Wiley, 2015.

[12] Murad Ahmed, Ahmed Badr, Mohammed Aborizka, and et al. Ensemble of machine learning algorithms for credit card fraud detection. *Ieee Access*, 6:37881–37893, 2018.

[13] N. Akhtar and A. Rehman. Credit card fraud detection using principal component analysis. ..., ...

[14] Naomi S Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.

[15] Siddhartha Bhattacharyya, Debashis Jha, and Kurian Tharakunnel. Data mining for credit card fraud: A comparative study. *Decision Support Systems*, 50(3):602–613, 2011.

[16] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[17] Richard J Bolton and David J Hand. Statistical fraud detection: A review. *Statistical Science*, 17(3):235–255, 2002.

[18] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[19] Gavin Brown. Machine learning for early detection of credit card fraud. *Jour of Data Analysis and Information Processing*, 5(03):81, 2017.

[20] Anna L Buczak and Erhan Guven. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys & Tutorials*, 18(2):1153–1176, 2016.

[21] Zoraida Callejas, Nubia Nieto, Claudio Moraga, and et al. Ensemble of extreme learning machines for credit card fraud detection. *Neurocomputing*, 150:41–49, 2015.

[22] Xudong Chai and Q M Jonathan Wu. Credit card fraud detection using adaboost and majority voting. *Computers & Operations Research*, 31(12):2055–2073, 2004.

[23] J. Chen and et al. Application of pca for cyber fraud detection. ..., ...

[24] Zhi Chen, Zhe Wang, and Li Sun. Credit card fraud detection using a profile-based approach. *Information Sciences*, 314:219–232, 2015.

[25] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[26] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

[27] D Richard Cutler, Thomas C Edwards Jr, Karen H Beard, Adele Cutler, Karl T Hess, James Gibson, and Joshua J Lawler. Random forests for classification in ecology. *Ecology*, 88(11):2783–2792, 2007.

[28] Andrea Dal Pozzolo, Giacomo Boracchi, Olivier Caelen, Cesare Alippi, and Gianluca Bontempi. Calibrating probability with undersampling for unbalanced classification. *Advances in Data Analysis and Classification*, 9(4):397–419, 2015.

[29] Andrea Dal Pozzolo, Giacomo Boracchi, Olivier Caelen, and Gianluca Bontempi. Credit card fraud detection: A realistic modeling and a novel learning strategy. *IEEE Transactions on Neural Networks and Learning Systems*, 29(8):3784–3797, 2017.

[30] Andrea Dal Pozzolo, Giacomo Boracchi, Olivier Caelen, and et al. Credit card fraud detection: a realistic modeling and a novel learning strategy. *IEEE Transactions on Neural Networks and Learning Systems*, 29(8):3784–3797, 2015.

[31] Andrea Dal Pozzolo, Giacomo Boracchi, Olivier Caelen, and et al. Dealing with class imbalance in credit card fraud detection. *Data Mining and Knowledge Discovery*, 31(3):783–814, 2017.

[32] Thanh N Do, Thanh-Dinh Nguyen, and Grenville Armitage. Credit card fraud detection with autoencoders: An outlier detection approach. *Expert Systems with Applications*, 132:280–298, 2019.

[33] H. Drucker, D. Wu, and V. N. Vapnik. Support vector regression machines. In *Advances in neural information processing systems*, pages 155–161, 1999.

[34] S. Guha and et al. Anomaly detection using principal component analysis. ..., ...

[35] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2009.

[36] D. W. Hosmer, S. Lemeshow, and R. X. Sturdivant. *Applied Logistic Regression*. Wiley, 2013.

[37] Y. C. Huang and C. J. Lin. A kernel-based multivariate feature selection method for microarray data classification. *IEEE Transactions on Nanobioscience*, 4(3):228–234, 2005.

[38] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In *European conference on machine learning*, pages 137–142. Springer, 1998.

[39] Anh Ngoc Kieu, Anh Tuan Luong, and Hoang Viet Nguyen. Hybrid approach based on neural network and feature selection for credit card fraud detection. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–13, 2020.

[40] G. King and L. Zeng. Logistic regression in rare events data. *Political Analysis*, 9(2):137–163, 2001.

[41] C. Leslie, E. Eskin, and W. S. Noble. The spectrum kernel: A string kernel for svm protein classification. In *Pacific Symposium on Biocomputing*, pages 564–575, 2002.

[42] Andy Liaw and Matthew Wiener. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

[43] Zhijie Liu, Xuan Xiao, and Wang-Ren Qiu. Integrating k-nearest neighbor with support vector machine for protein secondary structure prediction. *Proteins: Structure, Function, and Bioinformatics*, 50(4):629–634, 2003.

[44] Jie Lu, Lei Ji, Xuchuan Yuan, and et al. Credit card fraud detection using adaboost with random under-sampling technique. *Computers, Materials & Continua*.

[45] P. McCullagh and J. A. Nelder. *Generalized Linear Models*. CRC Press, 1989.

[46] Mohammadreza Mohammadzadeh, Karim Faez, and Kambiz Rezaee. Credit card fraud detection using genetic algorithm and back-propagation. *Journal of Applied Science*, 13(2):180–186, 2013.

[47] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

[48] Eric WT Ngai, Lai Xiu, DC Chau, and et al. Application of data mining techniques in customer relationship management: A literature review and classification. *Expert Systems with Applications*, 36(2):2592–2602, 2011.

[49] C. Phua and et al. Unsupervised learning techniques for fraud detection. *...*, ...

[50] Mohammad Rezvani, Ali Yazdan Varjani, Ali Souri, and et al. Fraud detection in online social networks: A comprehensive survey. *IEEE Access*, 7:23114–23136, 2019.

[51] Juan J Rodriguez, Ludmila I Kuncheva, and Carlos J Alonso. Rotation forest: A new classifier ensemble method. *IEEE transactions on pattern analysis and machine intelligence*, 28(10):1619–1630, 2010.

[52] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987.

[53] Asaf Shabtai, Robert Moskovitch, Yuval Elovici, and et al. Comparison of intrusion detection systems for advanced metering infrastructure in smart grid. *IEEE Transactions on Industrial Informatics*, 10(2):1491–1499, 2014.

[54] A. Sharma and S. Singh. Fraud detection in insurance using principal component analysis. *...*, ...

[55] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Pearson, 2019.

[56] Yong Wang and Shuai Wang. Fraud detection in imbalanced credit card dataset. *Information Sciences*, 423:142–157, 2018.

[57] Lian Wei. Detecting credit card fraud with recurrent neural networks. *Economic Computation and Economic Cybernetics Studies and Research*, 52(3):5–20, 2018.

[58] Lena Wiese, Alexander Boettcher, and Jens Grossklags. Misleading trajectories: Analysis and clustering of abnormal credit card transactions. *Decision Support Systems*, 57:233–243, 2014.

[59] Tao Yang, Jiyoung Kim, and SangYup Kim. Deep learning for credit card fraud detection. *Information Sciences*, 449:175–191, 2018.

[60] Masoud Zareapoor, Samad Shahrampour, and Mohammad Esmi Jahromi. Anomaly detection in financial transactions using unsupervised and supervised machine learning. *Procedia Computer Science*, 143:700–706, 2018.

[61] Chengwei Zhang, Liang Chen, Dan Miao, and Xinyu Mei. Detecting credit card fraud with recurrent neural networks. *Procedia Computer Science*, 131:1064–1071, 2018.

[62] Hongke Zhao, Min Du, Hui Zhang, and Yingying Chen. Credit card fraud detection based on random forest. *Symmetry*, 11(6):746, 2019.

[63] Yaoqi Zhou, Shuai Zhang, Yaqing Liu, and et al. A survey of deep learning applications in big data analytics. *Journal of King Saud University-Computer and Information Sciences*, 2019.