Fraud Detection in Banking Data by Machine Learning Techniques

PROJECT SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF

BACHELOR OF TECHNOLOGY IN INFORMATION TECHNOLOGY

OF THE ANNA UNIVERSITY

MINI PROJECT WORK

2023

SUBMITTED BY

AJMAL FARIZ M
2018102

GOKUL R
2018113

ARAVIND M
2018L05

KIRAN R V
2018L12

Under the guidance of

Dr. S. RATHI, M.E., Ph.D.,

Head of the Department

DEPARTMENT OF INFORMATION TECHNOLOGY

GOVERNMENT COLLEGE OF TECHNOLOGY

(An Autonomous institution affiliated to Anna University)

COIMBATORE – 641 013

DEPARTMENT OF INFORMATION TECHNOLOGY

GOVERNMENT COLLEGE OF TECHNOLOGY

(An Autonomous Institution affiliated to Anna University)

COIMBATORE – 641013

MINI PROJECT WORK

DECEMBER 2023

This is to certify that this project work entitled

Fraud Detection in Banking Data by Machine Learning Techniques

is the bonafide record of project work done by

AJMAL FARIZ M
2018102

GOKUL R
2018113

ARAVIND M
2018L05

KIRAN R V
2018L12

of B.Tech. (Information Technology) during the year **2023 – 2024**

DR. S. RATHI M.E, Ph.D.,	**DR. S. RATHI M.E, Ph.D.,**
Project Guide	Head of the Department

Submitted for the Project Viva-Voce examination held on _____

Internal Examiner	**External Examiner**

ACKNOWLEDGEMENT

Great achievements are not possible without standing on the shoulders of giants. without the active involvement of the following experts this project would not have been a reality.

We express our sincere gratitude to **Dr.K.Manonmani**, **M.E., Ph.D.,** Principal, Government College of Technology, Coimbatore for providing us all facilities that we needed for the completion of this project.

We whole-heartedly express our thankfulness and gratitude to **Dr.S.Rathi, M.E., Ph.D.,** Professor and Head of the Department of Information Technology, Government College of Technology, for helping us to successfully carry out this project.

Our thankfulness and gratitude to our respectable project guide, **Dr.S.Rathi, M.E., Ph.D.,** Professor and Head of the Department of Information Technology who has been immense help through the various phases of the project, with her potent ideas and excellent guidance, we were able to comprehend the essential aspects involved.

We would like to thank our faculty advisor **Dr.S.Gladson Oliver, M.Tech., Ph.D.,** Assistant Professor for his continuous support and encouragement throughout this project.

We extend our sincere thanks to the staff members of Information Technology department, **Dr.R.Devi, M.Tech., Ph.D., Assistant Professor, Prof.C.Aswini, M.E., Assistant Professor, Dr.T.Suguna, M.Tech., Ph.D., Assistant Professor, Prof.M.Jeyanthi, M.Tech., Assistant Professor, Dr.R.Malavika, M.Tech., Ph.D., Assistant Professor, Dr.M .Blessy Queen Mary, M.E., Ph.D., Assistant Professor,** for rendering their help for the completion of this project. we also thank all our friends for their cooperation and suggestions towards the successful completion of this project.

SYNOPSIS

In recent years, there has been a significant increase in the volume of financial transactions due to the expansion of financial institutions and the popularity of web-based ecommerce. Fraudulent transactions have become a growing problem in online banking, and fraud detection has always been challenging. Along with credit card development, the pattern of credit card fraud has always been updated. Fraudsters do their best to make it look legitimate, and credit card fraud has always been updated. Fraudsters do their best to make it look legitimate. They try to learn how fraud detection systems work and continue to stimulate these systems, making fraud detection more complicated. Fraud detection in banking is considered a binary classification problem in which data is classified as legitimate or fraudulent. Because banking data is large in volume and with datasets containing a large amount of transaction data, manually reviewing and finding patterns for fraudulent transactions is either impossible or takes a long time. Therefore, machine learning-based algorithms play a pivotal role in fraud detection and prediction. Machine learning algorithms and high processing power increase the capability of handling large datasets and fraud detection in a more efficient manner. Machine learning algorithms and deep learning also provide fast and efficient solutions to real-time problems.

CONTENTS

CHAPTER 1

INTRODUCTION

1.1 DESCRIPTION

As technology advanced and e-commerce services expanded, credit cards became one of the most popular payment methods, resulting in an increase in the volume of banking transactions. Furthermore, the significant increase in fraud requires high banking transaction costs. People who commit fraud usually use security, control, and monitoring weaknesses in commercial applications to achieve their goals. However, technology can be a tool to combat fraud. Fraud detection in banking is considered a binary classification problem in which data is classified as legitimate or fraudulent. Therefore, machine learning-based algorithms play a pivotal role in fraud detection and prediction.

1.2 EXISTING SYSTEM

Various algorithms, including Logistic Regression, LightGBM, XGBoost. CatBoost and Majority Voting ensemble learning method were implemented and evaluated. Bayesian optimization and class weight tuning were performed for each model. The model was trained and tested using a 5-fold cross-validation technique.

1.3 PROBLEM DEFINITION

According to the Global Fraud Landscape, Fraud losses are expected to reach $343 billion by 2026, with credit cards being the most targeted payment instrument. Card-not-present (CNP) fraud is the fastest-growing segment, accounting for 43% of all fraud losses in 2021.Fraudsters are becoming more sophisticated, using social engineering, malware, and deepfakes to bypass traditional security measures. The objective is to identify fraud transactions.

1.4 PROPOSED SYSTEM

Proposed algorithms exhibit higher performance when compared to traditional techniques.

1.5 ORGANIZATION OF THE PROJECT

- Literature reviews of already existing proposals are discussed in chapter 2.
- Chapter 3 has system specification which tells about the software requirements.
- Chapter 4 discusses training algorithm which are used in this project.
- Chapter 5 discusses the overall project and design which tells the brief description of each of the modules in this project.
- Chapter 6 has the implementation and experimental result of the project.
- Chapter 7 deals with the conclusion and
- Finally, chapter 8 deals with the reference

CHAPTER 2

LITERATURE SURVEY

2.1 An Intelligent Approach to Credit Card Fraud Detection Using an Optimized Light Gradient Boosting Machine [2020]

2.1.1 DESCRIPTION

The detection of fraudulent transactions has become a significant factor affecting the greater utilization of electronic payment. Thus, there is a need for efficient and effective approaches for detecting fraud in credit card transactions. This paper proposes an intelligent approach for detecting fraud in credit card transactions using an optimized light gradient boosting machine (LightGBM). In the proposed approach, a Bayesian-based hyperparameter optimization algorithm is intelligently integrated to tune the parameters of a light gradient boosting machine (LightGBM). To demonstrate the effectiveness of proposed LightGBM for detecting fraud in credit card transactions, experiments were performed using two real-world public credit card transaction data sets consisting of fraudulent transactions and legitimate ones. Based on a comparison with other approaches using the two data sets, the proposed approach outperformed the other approaches and achieved the highest performances.

2.1.2 MERIT

- LightGBM algorithm surpasses other methods in credit card fraud detection, achieving high accuracy, AUC, precision, and recall.

- Rigorous evaluation using real-world datasets, 5-fold cross-validation, and feature optimization enhances the study's reliability.

2.1.3 DEMERIT

- Lack of comparison with deep learning methods restricts understanding of potential advantages in fraud detection.

- It lacks insights into the scalability of LightGBM, especially in terms of computational efficiency and real-time processing.

2.2 Card Fraud Prediction and Classification using Deep Neural Network and Ensemble Learning [2020]

2.2.1 DESCRIPTION

This paper explores the surging use of credit cards and the escalating challenge of fraud. It focuses on predicting faulty transactions using Ensemble learning, comparing four algorithms: Naïve Bayes, Logistic Regression, Decision Trees, and Deep Belief Network. The study demonstrates the significance of the deep belief network algorithm and asserts that the proposed ensemble model, incorporating all four algorithms, excels in predicting credit card defaults with exceptional precision.

2.2.2 MERIT

- Logistic Regression, Naïve Bayes, Decision Tree, and Deep Belief Network in an ensemble approach significantly boosts fraud detection accuracy.

- Prioritizing Deep Belief Network showcases a commitment to advanced techniques, enhancing overall model performance.

2.2.3 DEMERIT

- Binarizing features for Deep Belief Network adds complexity to preprocessing, requiring careful consideration.

- A weakness in precision, leading to more false positives, may impact real-world reliability.

2.3 Automatic tuning of hyperparameters using Bayesian optimization [2020]

2.3.1 DESCRIPTION

Deep learning is a field in artificial intelligence that works well in computer vision, natural language processing and audio recognition. Deep neural network architectures have number of layers to conceive the features well, by itself. The hyperparameter tuning plays a major role in every dataset which has major effect in the performance of the training model. Due to the large dimensionality, it is impossible to tune the parameters by human expertise. In this paper, the CIFAR-10 Dataset is used and applied the Bayesian hyperparameter optimization algorithm to enhance the performance of the model. Bayesian optimization can be used for any noisy black box function for hyperparameter tuning. In this work Bayesian optimization clearly obtains. optimized values for all hyperparameters which saves time and improves performance. The results also show that the error has been reduced in graphical processing unit than in CPU by 6.2% in the validation. Achieving global optimization in the trained model helps transfer learning across domains as well.

2.3.2 MERIT

- Demonstrates Bayesian optimization's efficiency for tuning deep neural network hyperparameters, enhancing model performance and reducing training time.

- Successfully achieves global optima, especially in scenarios with expensive function evaluations and extensive search spaces.

2.3.3. DEMERIT

- Potential limitation in Bayesian optimization, as the acquisition function may set the search space early, risking the omission of crucial features.

- Recognizes the trade-off between model training and hyperparameter tuning time, but more insights or solutions could be beneficial.

2.4 Deep Learning Detecting Fraud in Credit Card Transactions [2018]

2.4.1 DESCRIPTION

Credit card fraud incurred a $3 billion loss for North American financial institutions in 2017, with digital payment systems expected to amplify this threat. Deep Learning emerges as a promising solution for fraud detection, offering comparable results to existing methods. This study evaluates various Deep Learning topologies and parameters using a dataset of 80 million credit card transactions. Leveraging a high-performance cloud computing environment, the analysis addresses common fraud detection challenges and provides a guide for sensitivity analysis and parameter tuning. The goal is to equip financial institutions with tools to minimize losses by preventing fraudulent activities.

2.4.2 MERIT

- Evaluates multiple deep learning topologies on a large dataset for credit card fraud detection.
- Uses account-level undersampling to address class imbalance, enhancing model reliability.

2.4.3 DEMERIT

- Focuses on specific hyperparameters, potentially missing nuances in model sensitivity.
- Acknowledges uncertainties regarding when model performance improvement plateaus with network size.

2.5 Detecting credit card fraud using selected machine learning algorithms[2019]

2.5.1 DESCRIPTION

Due to the immense growth of e-commerce and increased online based payment possibilities, credit card fraud has become deeply relevant global issue. Recently, there has been major interest for applying machine learning algorithms as data mining technique for credit card fraud detection. However, number of challenges appear, such as lack of publicly available data sets, highly imbalanced class sizes, variant fraudulent behavior etc.

In this paper, the performance of three machine learning algorithms Random Forest, Support Vector Machine, and Logistic Regression is compared in detecting fraud using real-life data containing credit card transactions. The issue of ever-changing fraud patterns is addressed by employing incremental learning with the selected ML algorithms in experiments. The techniques' performance is evaluated based on the commonly accepted metrics: precision and recall.

2.5.2 MERITS

- SMOTE is utilized to tackle class imbalance by generating synthetic samples for the minority class.
- It conducts thorough experiments, comparing static and incremental learning approaches, providing insights into the algorithms' performance on a real-world dataset.

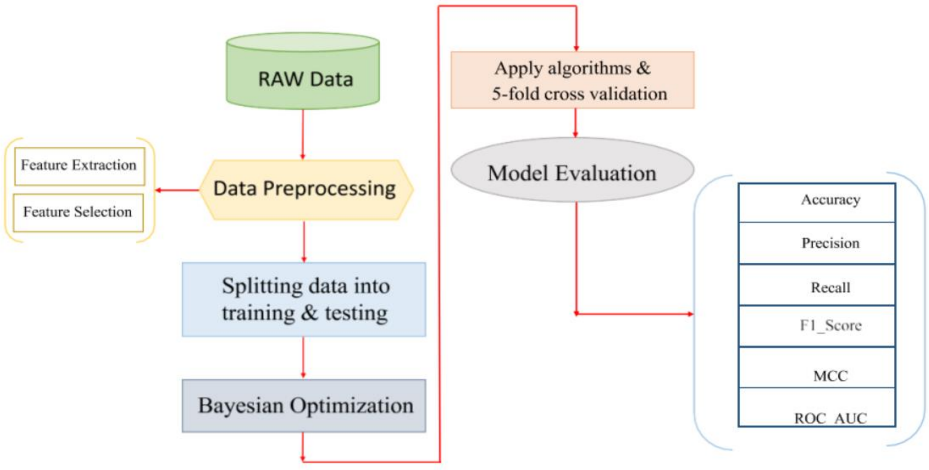
2.5.3 DEMERITS

- Overlapping performance, notably with SVM, introduces uncertainty in determining a clear algorithmic superiority, reducing practical applicability.
- Acknowledgment of static training limitations lacks a detailed exploration of challenges with incremental learning, offering a less nuanced understanding of proposed solutions.

CHAPTER – 3

PROJECT DESIGN

3.1 FLOW CHART



3.2 DATASET

It contains only numerical input variables which are the result of a PCA transformation. Features V1, V2, … V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise**.**

3.3 DATA PRE-PROCESSING

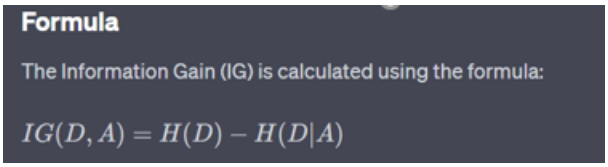
- ✓ Remove Null values.
- ✓ Remove Duplicate rows from the dataset.

3.3.1 FEATURE EXTRACTION

The feature extraction process involves expanding the "time" feature to extract additional temporal information. Specifically, we derive two new features: "hour" and "second". This extraction allows us to gain more detailed insights into the timing of transactions.

3.3.2 FEATURE SELECTION

The Information Gain method is a commonly used technique in the field of machine learning and decision tree-based algorithms. It is employed to determine the relevance of a feature in a dataset when constructing decision trees or making decisions based on data.



3.4 BAYESIAN OPTIMIZATION:

Bayesian optimization is a powerful technique used primarily in the field of machine learning and optimization for finding the optimal set of hyperparameters for complex, computationally expensive functions. It is particularly useful in scenarios where the objective function is costly to evaluate, as it aims to minimize the number of function evaluations required to find the best parameter configuration.

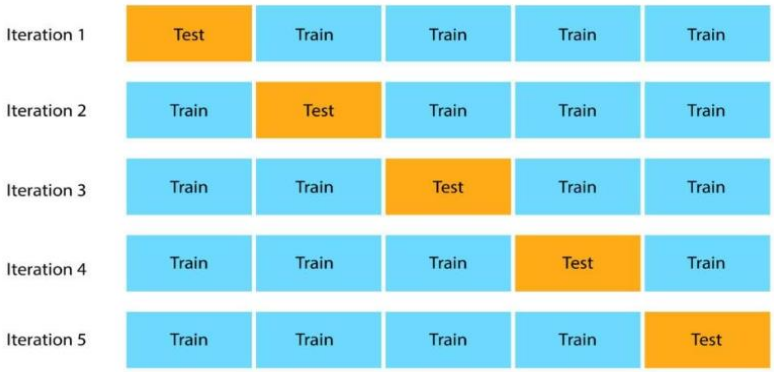
3.5 ALGORITHMS:

Below mentioned classifier models are implemented.

- Logistic Regression
- LightGBM
- XGBoost
- CatBoost
- Majority Voting
- Deep Learning

3.5.1 FIVE-FOLD CROSS VALIDATION

K-fold cross-validation is a superior technique to validate the performance of our model. It evaluates the model using different chunks of the data set as the validation set. We divide our data set into K-folds. K represents the number of folds into which you want to split your data. If we use 5-folds, the data set divides into five sections. In different iterations, one part becomes the validation set.



3.6 MODEL EVALUATION

We use the stratified 5-fold cross validation method and the boosting algorithms with the Bayesian optimization method to evaluate the performance of the proposed framework. We extract the hyperparameters and evaluate each algorithm individually before using the majority voting method. We examine the algorithms in triple and double precision.

- **Accuracy**
- **AUC**
- **Recall**
- **Precision**
- **F1-Score**
- **MCC**

CHAPTER - 4

SYSTEM SPECIFICATION

4.1 SYSTEM REQUIREMENTS

4.1.1 SOFTWARE REQUIREMENTS

- Operating System : Windows 10.
- Coding Language : PYTHON.
- Tool : ANACONDA.

4.1.2 ANACONDA SPECIFICATION

- System : ANACONDA.
- Hard Disk : 300GB.
- RAM : 8GB.

4.1.3 JUPYTER NOTEBOOK SPECIFICATION

- System : JUPYTER NOTEBOOK.
- Hard Disk : 1GB.
- RAM : 1GB.

CHAPTER 5

TRAINING ALGORITHM

5.1 LOGISTIC REGRESSION

5.1.1 OVERVIEW

Introduction to Logistic Regression in Fraud Detection: Logistic Regression, a fundamental statistical analysis technique, is widely used for binary classification problems, such as detecting fraudulent and legitimate transactions in credit card fraud detection. Its simplicity, interpretability, and effectiveness in binary classification make it a valuable tool in the initial stages of fraud detection systems.

5.1.2 KEY FEATURES

Logistic Regression is inherently designed for binary classification tasks, making it adept at distinguishing between two classes - in this case, fraudulent and legitimate transactions. One of the strongest advantages of Logistic Regression is its interpretability. The model provides clear insights into how different features influence the probability of a transaction being fraudulent, which is crucial for understanding and improving fraud detection mechanisms. Logistic Regression models are computationally efficient, requiring less computational resources for training and prediction, an essential feature for systems processing large numbers of transactions. This model is particularly effective when the relationship between the predictive features and the probability of fraud is linear, making it suitable for datasets where such linear relationships are prevalent.

5.1.3 APPLICATION IN PROJECT

In this project, Logistic Regression is utilized to analyze credit card transaction data. The model is trained on a dataset composed of various features labeled as either fraudulent or legitimate.

5.2 LIGHTGBM

5.2.1 OVERVIEW

Introduction to LightGBM in Fraud Detection: LightGBM, a gradient boosting framework developed by Microsoft, is particularly effective for handling large datasets with high efficiency and speed. In credit card fraud detection, LightGBM's ability to deal with unbalanced data, alongside its high processing speed, makes it an excellent choice for identifying fraudulent transactions.

5.2.2 KEY FEATURES

Credit card transaction datasets are typically large and complex. LightGBM is optimized for memory and speed, allowing for faster training on extensive datasets, which is essential for real-time fraud detection. Fraudulent transactions are relatively rare compared to legitimate ones, resulting in unbalanced datasets. LightGBM performs exceptionally well in such scenarios, effectively distinguishing between the minority class (fraudulent transactions) and the majority class (legitimate transactions). Compared to other machine learning algorithms, LightGBM requires less memory and has faster training times, which is crucial for deploying models quickly in response to evolving fraud tactics. LightGBM uses advanced techniques like Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB), which contribute to its high accuracy and precision, significantly reducing false positives and negatives in fraud detection.

5.2.3 APPLICATION IN PROJECT

This project employs LightGBM to develop an efficient and accurate model for detecting credit card fraud. The model processes a comprehensive dataset, including varieties of features.

5.3 XGBOOST

5.3.1 OVERVIEW

Introduction to XGBoost in Fraud Detection: XGBoost (Extreme Gradient Boosting) is a highly efficient and scalable implementation of gradient boosting. It has gained popularity in the machine learning community for its performance in classification tasks, particularly in scenarios with imbalanced datasets like credit card fraud detection.

5.3.2 KEY FEATURES

Credit card fraud detection typically involves working with highly imbalanced datasets. XGBoost's ability to give more focus to the minority class makes it highly effective in identifying fraudulent transactions, which are usually the minority in transaction datasets. XGBoost provides built-in methods for assessing feature importance, which is crucial in understanding and identifying the key attributes that signal fraudulent activity in transactions. Known for its computational efficiency, XGBoost can handle large-scale datasets with ease, a vital feature for processing the voluminous data generated by credit card transactions. XGBoost includes L1 and L2 regularization, which helps prevent overfitting, a common challenge in fraud detection where models may become too attuned to the training data and fail to generalize to unseen data.

5.3.3 APPLICATION IN PROJECT

In this project, XGBoost is applied to develop a robust model for detecting potential credit card fraud. The algorithm is trained on a dataset encompassing various features, including transaction amounts.

5.4 CATBOOST

5.4.1 OVERVIEW

CatBoost, an advanced machine learning algorithm, has proven to be highly effective in credit card fraud detection. It stands out due to its ability to process categorical data efficiently and its robustness against overfitting, which are crucial in analyzing complex and often deceptive patterns in financial transactions.

5.4.2 KEY FEATURES

CatBoost's innate ability to process such diverse data types without extensive pre-processing makes it a powerful tool in detecting anomalies and suspicious patterns. In credit card fraud detection, it's critical to minimize false positives and false negatives. CatBoost's advanced algorithms, including ordered boosting, significantly reduce the risk of overfitting, thereby improving the precision of fraud detection models. Understanding the reasoning behind a model's decision is crucial in fraud detection to ensure trust and compliance. CatBoost provides tools for model interpretation, which helps in understanding the factors influencing the detection of fraudulent transactions.

5.4.3 APPLICATION IN PROJECT

The CatBoost model is trained on historical transaction data, labeled as fraudulent or legitimate. Through its advanced feature engineering capabilities, the model learns to distinguish between normal and anomalous transaction patterns. Post-training, the model is evaluated for accuracy, precision, and recall, ensuring that it effectively minimizes false positives (legitimate transactions flagged as fraudulent) and false negatives (fraudulent transactions not detected).

5.5 MAJORITY VOTING

5.5.1 OVERVIEW

Introduction to Majority Voting in Fraud Detection: Majority Voting, a fundamental ensemble technique, involves combining the predictions from multiple models to make a final decision. In credit card fraud detection, employing a majority voting system can significantly enhance the accuracy and robustness of the detection process by leveraging the strengths of different algorithms.

.

5.5.2 KEY FEATURES

Majority voting aggregates the decisions from various models. This diversification reduces the risk associated with relying on a single model, especially important in the nuanced and ever-evolving field of fraud detection. By taking into account the predictions from different models, majority voting tends to offer more accurate and robust results than individual models, particularly in complex and imbalanced datasets like those in credit card fraud. Each model comes with its own set of biases and variances. It helps in mitigating these individual biases, leading to a more balanced and reliable fraud detection system. It allows for the combination of diverse models, from simple logistic regression to complex algorithms like XGBoost or neural networks, catering to the specific requirements of the fraud detection task.

5.5.3 APPLICATION IN PROJECT

In this project, a majority voting system is developed by integrating various predictive models, including Catboost, XGBoost, and LightGBM. Each model independently analyzes transaction data and casts a 'vote' on whether a transaction is fraudulent or legitimate.

5.6 DEEP LEARNING - ANN

5.6.1 OVERVIEW

Introduction to ANN in Fraud Detection: Artificial Neural Networks (ANN), inspired by the biological neural networks in human brains, are a cornerstone of modern machine learning. In credit card fraud detection, ANNs excel due to their ability to learn and model non-linear and complex relationships between variables, which is essential for identifying subtle and sophisticated fraudulent patterns in transaction data.

5.6.2 KEY FEATURES

ANNs are particularly adept at recognizing complex and non-linear patterns, a common characteristic of fraudulent transactions, which often deviate subtly from legitimate behavior. ANNs have the ability to learn and adapt from the data fed into them. This feature is crucial in fraud detection, where fraud patterns continuously evolve, requiring a dynamic and adaptive approach. Credit card transaction datasets often contain outliers or irrelevant data. ANNs can handle such noise effectively, making them suitable for real-world transaction data which is seldom clean or perfectly structured. ANNs can scale with the dataset size and complexity, making them ideal for processing the large volumes of data typically involved in credit card transactions.

5.6.3 APPLICATION IN PROJECT

In this project, an ANN-based model is developed to analyze and predict fraudulent transactions. The network architecture is designed with multiple layers, including input, hidden, and output layers, to effectively capture the intricate patterns in the data.

CHAPTER 6

IMPLEMENTATION AND RESULTS

6.1 IMPLEMENTATION

6.1.1 IMPORT LIBRARIES

```
import pandas as pd
import cv2
import numpy as np
from collections import Counter
import matplotlib.pyplot as plt
import scipy.stats as stats
import seaborn as sns
```

6.1.2 LOAD THE DATASET

This real dataset contains 284,807 records of two days of transactions made by credit card holders in September 2013.

```
df = pd.read_csv('credit_card.csv')
df.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0

Class Distribution in the Dataset

```
print('Non Fraud transactions', round(df['Class'].value_counts()[0]/len(df) * 100,2), '% of the dataset')

print('Frauds transactions', round(df['Class'].value_counts()[1]/len(df) * 100,2), '% of the dataset')
```

```
Non Fraud transactions 99.83 % of the dataset
Frauds transactions 0.17 % of the dataset
```

6.1.3 DATA PRE-PROCESSING

Check For Null Values

df.isnull().sum().max()

There is no null values in the dataset

Removing duplicate rows

```
duplicate_rows = df[df.duplicated()]
if not duplicate_rows.empty:
    print("Duplicate rows found. Number of duplicates:", len(duplicate_rows))
    df.drop_duplicates(inplace=True)
    print("Duplicate rows removed.")
else:
    print("No duplicate rows found.")
```

```
Duplicate rows found. Number of duplicates: 1081
Duplicate rows removed.
```

6.1.4 FEATURE EXTRACTION

The ‘‘time’’ feature includes the time (in seconds) elapsed between each transaction and the first transaction. Extract the transaction hour feature, which gives us more information than the time feature itself.

```
df['hour']=((df['Time']//3600)//2.).astype('int')
df['second']=(df['Time']%3600).astype('int')
df.drop('Time',axis=1,inplace=True)
df.head(5)
```

V7	V8	V9	V10	...	V23	V24	V25	V26	V27	V28	Amount	Class	hour	second
0.239599	0.098698	0.363787	0.090794	...	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0	0	0
-0.078803	0.085102	-0.255425	-0.166974	...	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0	0	0
0.791461	0.247676	-1.514654	0.207643	...	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0	0	1
0.237609	0.377436	-1.387024	-0.054952	...	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0	0	1
0.592941	-0.270533	0.817739	0.753074	...	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0	0	2

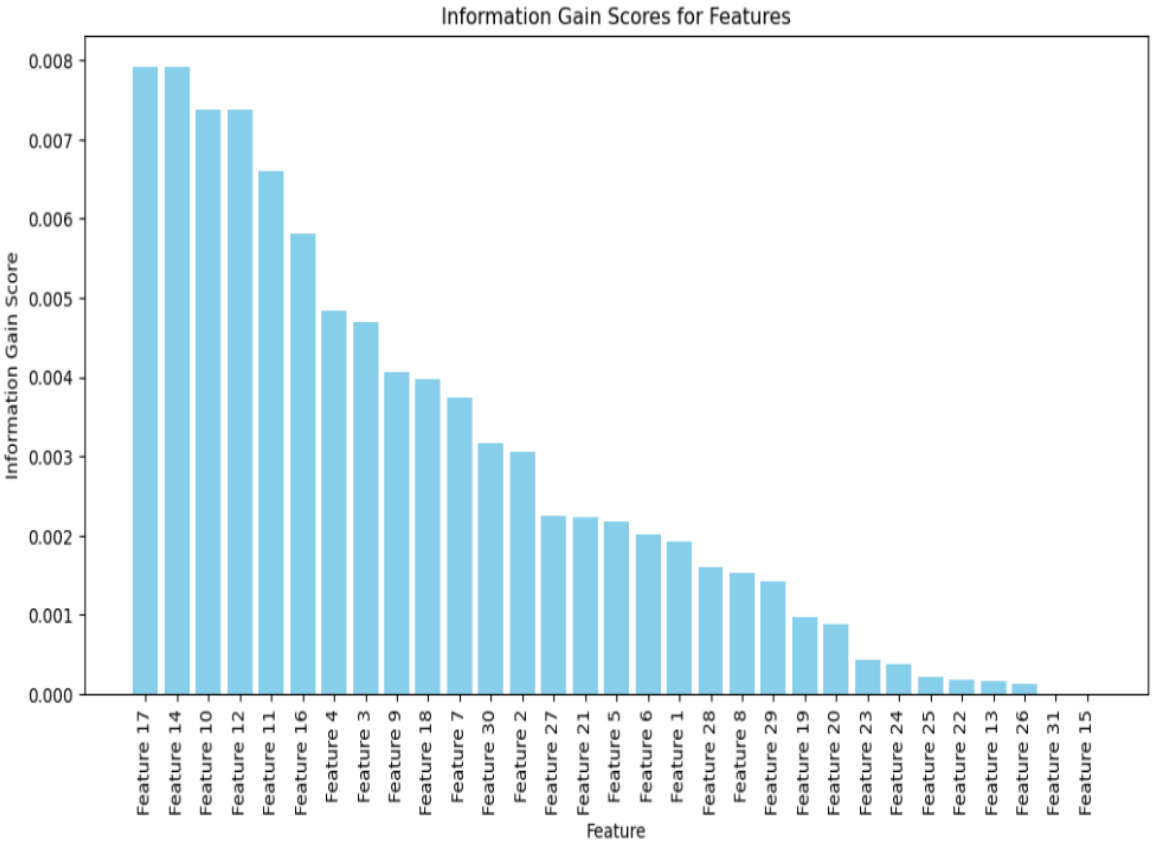
6.1.5 FEATURE SELECTION

The information gain (IG) method is used to select the most important features that lead to a dimension reduction of the training data. The top six features extracted by this method have been used to evaluate the proposed algorithm.

```
infogain_classif = SelectKBest(score_func=mutual_info_classif, k=6)
infogain_classif.fit(X_train, y_train)

feature_names = X_train.columns

for i in range(len(infogain_classif.scores_)):
    print(f'Feature "{feature_names[i]}" : {round(infogain_classif.scores_[i], 4)}')
```

```
Feature "V1" : 0.0019
Feature "V2" : 0.0031
Feature "V3" : 0.0047
Feature "V4" : 0.0048
Feature "V5" : 0.0022
Feature "V6" : 0.002
Feature "V7" : 0.0037
Feature "V8" : 0.0015
Feature "V9" : 0.0041
Feature "V10" : 0.0074
Feature "V11" : 0.0066
Feature "V12" : 0.0074
Feature "V13" : 0.0002
Feature "V14" : 0.0079
Feature "V15" : 0.0
Feature "V16" : 0.0058
Feature "V17" : 0.0079
Feature "V18" : 0.004
Feature "V19" : 0.001
Feature "V20" : 0.0009
Feature "V21" : 0.0022
Feature "V22" : 0.0002
Feature "V23" : 0.0004
Feature "V24" : 0.0004
Feature "V25" : 0.0002
Feature "V26" : 0.0001
Feature "V27" : 0.0022
Feature "V28" : 0.0016
Feature "Amount" : 0.0014
Feature "hour" : 0.0031
Feature "second" : 0.0
```

```
feature_indices = range(len(infogain_classif.scores_))
scores = infogain_classif.scores_
sorted_indices = np.argsort(scores)[::-1]
sorted_scores = [scores[i] for i in sorted_indices]
sorted_features = [f'Feature {i+1}' for i in sorted_indices]
plt.figure(figsize=(12, 6))
plt.bar(sorted_features, sorted_scores, color='skyblue', align='center')
plt.xlabel('Feature')
plt.ylabel('Information Gain Score')
plt.xticks(rotation=90)
plt.title('Information Gain Scores for Features')
plt.show()
```




```
mask=infogain_classif.get_support()
not_mask=np.logical_not(mask)
all_features=np.array(list(X))
best_features=all_features[mask]
```

```
Best Features : ['V10' 'V11' 'V12' 'V14' 'V16' 'V17']
```

6.1.6 ALGORITHMS

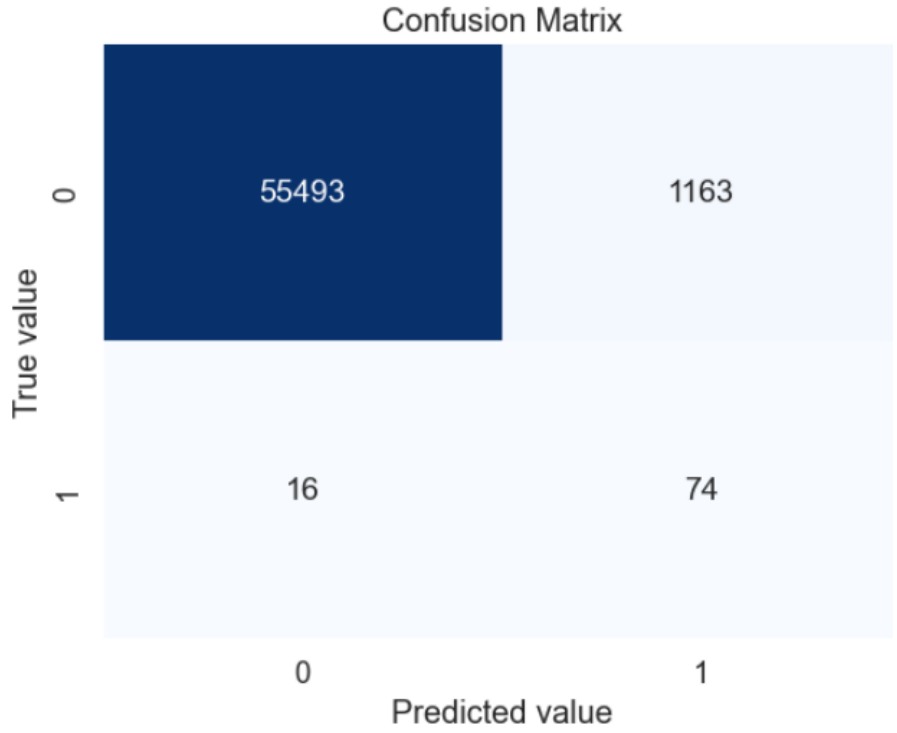
6.1.7 LOGISTIC REGRESSION

```
pbounds = {
'max_iter': (300, 3000)
}
logistic_model =
LogisticRegression(random_state=42,class_weight='balanced')
def objective_function(max_iter)
logistic_model.set_params(max_iter=int(max_iter))
cv = StratifiedKFold(n_splits=5)
scores = cross_val_score(logistic_model, X_train, y_train, cv=cv,
scoring='neg_log_loss')
return -np.mean(scores)

optimizer = BayesianOptimization(
f=objective_function,
pbounds=pbounds,
random_state=42,
)

optimizer.maximize(
init_points=5,
n_iter=8,
)
optimal_max_iter = optimizer.max['params']['max_iter']
print(f"Optimal Number of Iterations: {optimal_max_iter}")
```

```
|   iter    |  target   | max_iter  |
-------------------------------------
| 1         | 0.1342    | 1.311e+03 |
| 2         | 0.1342    | 2.867e+03 |
| 3         | 0.1342    | 2.276e+03 |
| 4         | 0.1342    | 1.916e+03 |
| 5         | 0.1342    | 721.3     |
| 6         | 0.1342    | 300.0     |
| 7         | 0.1342    | 590.2     |
| 8         | 0.1342    | 1.206e+03 |
| 9         | 0.1342    | 2.999e+03 |
| 10        | 0.1342    | 3e+03     |
| 11        | 0.1342    | 300.0     |
| 12        | 0.1342    | 454.7     |
| 13        | 0.1342    | 2.177e+03 |
=====================================
Optimal Number of Iterations: 1311.2583208878787
```

s

```
log_reg_output=cv_results(logistic_model, output_type='list')
logistic_model.fit(X_train, y_train)
y_pred = logistic_model.predict(X_test)
conf_matrix = confusion_matrix(y_test, y_pred)
sns.set(font_scale=1.2)
sns.heatmap(conf_matrix, annot=True, fmt='d',
cmap='Blues',annot_kws={"size": 14}, cbar=False)
plt.xlabel('Predicted value')
plt.ylabel('True value')
plt.title('Confusion Matrix')
plt.show()
```



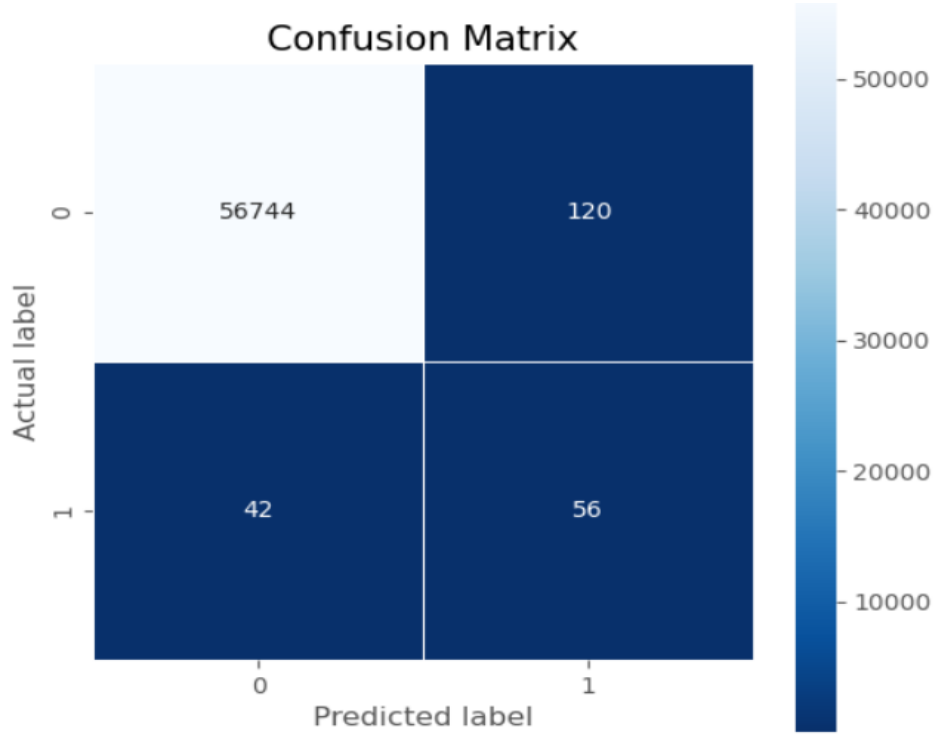
Model Name	f1	precision	recall	accuracy	roc_auc	mcc
Logistic Regression	0.11428496	0.06174842	0.87301232	0.97476791	0.95784261	0.22481584

6.1.8 LIGHTGBM

```
import numpy as np
import time
from lightgbm import LGBMClassifier
from sklearn.model_selection import cross_validate, StratifiedKFold
from bayes_opt import BayesianOptimization, UtilityFunction


def lgbm_cv(learning_rate, max_depth, num_leaves):
    model = LGBMClassifier(learning_rate=learning_rate,
                           num_leaves=int(round(num_leaves)),


    max_depth=int(round(max_depth)),
                           class_weight='balanced',
                           verbose = -1
                          )

cv = StratifiedKFold(n_splits=5)
scores = cross_validate(model, X_train, y_train, cv=cv, scoring='neg_log_loss')
return np.mean(scores['test_score'])

params = {
    'learning_rate': (0.001, 0.2),
    'max_depth': (-1, 8),
    'num_leaves': (2, 250)
}
 lgbmBO = BayesianOptimization(lgbm_cv, params)
 utility = UtilityFunction(kind='ei', kappa=2.5, xi=0.0)
 start = time.time()
lgbmBO.maximize(init_points=5, n_iter=8, acquisition_function=utility)
print('Optimization takes %s minutes' % ((time.time() - start)/60))
params_lgbm = lgbmBO.max['params']
params_lgbm['max_depth'] = round(params_lgbm['max_depth'])
params_lgbm['num_leaves'] = round(params_lgbm['num_leaves'])
print(params_lgbm)
```

```
|   iter    |  target   | learni... | max_depth | num_le... |
-------------------------------------------------------------
|  1        | -0.0385   |  0.05394  |  5.859    |  85.04    |
|  2        | -0.1525   |  0.02298  |  3.542    |  121.4    |
|  3        | -0.0105   |  0.1213   |  6.235    |  94.25    |
|  4        | -0.005794 |  0.18     | -0.6813   |  45.42    |
|  5        | -0.005467 |  0.109    | -0.5032   |  232.2    |
|  6        | -0.0202   |  0.03453  | -0.3525   |  230.8    |
|  7        | -0.007284 |  0.2      | -1.0      |  236.9    |
|  8        | -0.01551  |  0.2      |  3.962    |  235.1    |
|  9        | -0.007212 |  0.08437  | -0.6308   |  38.17    |
|  10       | -0.006042 |  0.1781   |  6.229    |  42.25    |
|  11       | -0.008241 |  0.1386   |  6.198    |  51.09    |
|  12       | -0.005409 |  0.182    |  7.661    |  32.61    |
|  13       | -0.006229 |  0.1313   |  0.1856   |  27.54    |
=============================================================
It takes 6.197216959794362 minutes
{'learning_rate': 0.18195577305404498, 'max_depth': 8, 'num_leaves': 33}
```

```
import lightgbm as lgb
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, matthews_corrcoef, confusion_matrix, roc_auc_score
import matplotlib.pyplot as plt
import seaborn as sns

model = lgb.LGBMClassifier(verbose=-1, learning_rate =
0.18195577305404498, max_depth = 8. num_leaves =33)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
y_pred_proba = model.predict_proba(X_test)[:, 1]
        accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
mcc = matthews_corrcoef(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred_proba)
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6,6))
sns.heatmap(cm, annot=True, fmt="d", linewidths=.5, square=True,
cmap='Blues_r')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.title('Confusion Matrix', size=15)
plt.show()
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
print("Matthews Correlation Coefficient:", mcc)
print("AUC-ROC:", auc)
```



Model Name	f1	precision	recall	accuracy	roc_auc	mcc
Lightgbm	0.75495483	0.72187393	0.80537514	0.99910477	0.9439948	0.75882762

6.1.9 XGBOOST

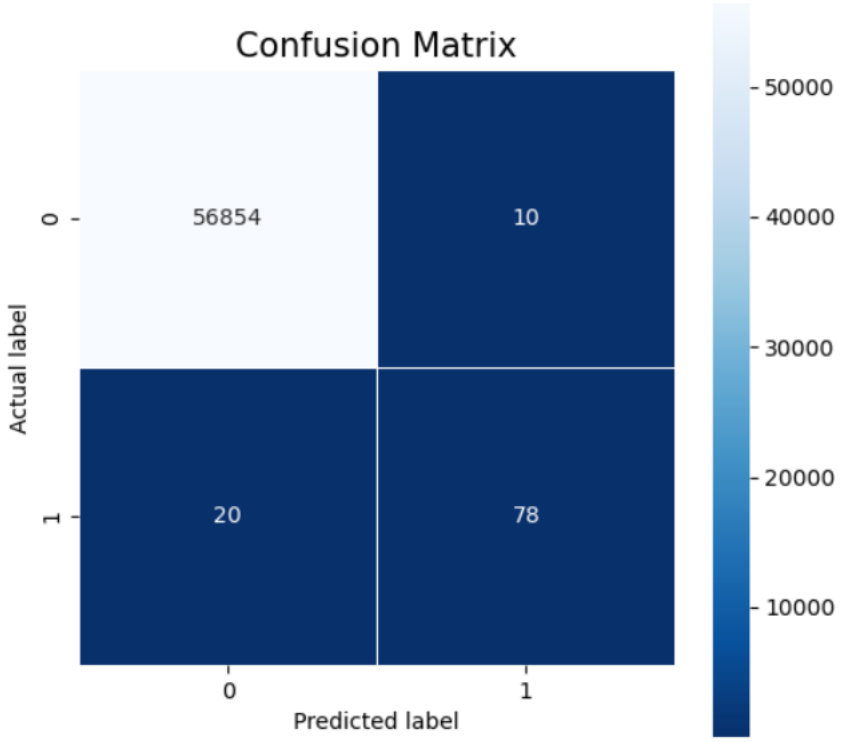
```
import time
import numpy as np
from xgboost import XGBClassifier
from sklearn.model_selection import cross_validate, StratifiedKFold
from bayes_opt import BayesianOptimization, UtilityFunction

# Placeholder function for XGBoost cross-validation
# Replace with your actual function
def xgb_cv(learning_rate, max_depth, n_estimators):
    model = XGBClassifier(learning_rate=learning_rate,
                          max_depth=int(round(max_depth)),
                          n_estimators=int(round(n_estimators)),
```

```
                    eval_metric='logloss')

    cv = StratifiedKFold(n_splits=5)
    # Assuming X_train and y_train are defined
    scores = cross_validate(model, X_train, y_train, cv=cv,
scoring='neg_log_loss')
    return np.mean(scores['test_score'])

# Define the parameter space to explore
params = {
    'learning_rate': (0.01, 0.2),
    'max_depth': (3, 10),
    'n_estimators': (50, 100)
}

# Initialize Bayesian Optimization
xgbBO = BayesianOptimization(xgb_cv, params)

# Create an instance of UtilityFunction
utility = UtilityFunction(kind='ei', kappa=2.5, xi=0.0)

# Start the optimization process
start = time.time()
xgbBO.maximize(init_points=5, n_iter=8, acquisition_function=utility)

# Calculate and print the optimization time
elapsed_time = (time.time() - start) / 60
print(f'Optimization takes {elapsed_time:.2f} minutes')

# Output the best parameters found
params_xgb = xgbBO.max['params']
params_xgb['max_depth'] = int(round(params_xgb['max_depth']))
params_xgb['n_estimators'] = int(round(params_xgb['n_estimators']))
print(params_xgb)
```

```
|   iter    |  target   | learni... | max_depth | n_esti... |
-------------------------------------------------------------
| 1         | -0.003233 | 0.1827    | 4.896     | 84.9      |
| 2         | -0.003139 | 0.08842   | 7.685     | 78.35     |
| 3         | -0.02658  | 0.02625   | 8.758     | 61.24     |
| 4         | -0.003238 | 0.06822   | 5.141     | 86.56     |
| 5         | -0.003491 | 0.08255   | 6.925     | 62.82     |
| 6         | -0.003226 | 0.1664    | 4.848     | 85.72     |
| 7         | -0.003122 | 0.1156    | 5.849     | 63.75     |
| 8         | -0.007704 | 0.03678   | 7.147     | 85.32     |
| 9         | -0.01054  | 0.04132   | 7.34      | 64.67     |
| 10        | -0.003128 | 0.1353    | 5.682     | 62.58     |
| 11        | -0.003293 | 0.06911   | 3.09      | 84.81     |
| 12        | -0.007438 | 0.03666   | 3.265     | 87.22     |
| 13        | -0.003149 | 0.1749    | 3.73      | 82.96     |
=============================================================
Optimization takes 0.86 minutes
{'learning_rate': 0.11560216784330198, 'max_depth': 6, 'n_estimators': 64}
```

```
model = xgb.XGBClassifier(learning_rate = 0.11560216784330198, max_depth
= 6, n_estimators = 6)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
y_pred_proba = model.predict_proba(X_test)[:, 1]

precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
mcc = matthews_corrcoef(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred_proba)


cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6,6))
sns.heatmap(cm, annot=True, fmt="d", linewidths=.5, square=True,
cmap='Blues_r')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.title('Confusion Matrix', size=15)
plt.show()


print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
print("Matthews Correlation Coefficient:", mcc)
print("AUC-ROC:", auc)
```

Model Name	f1	precision	recall	accuracy	roc_auc	mcc
Xgboost	0.45650138	0.33048142	0.80114222	0.99644728	0.94975002	0.50569031



6.1.10 CATBOOST

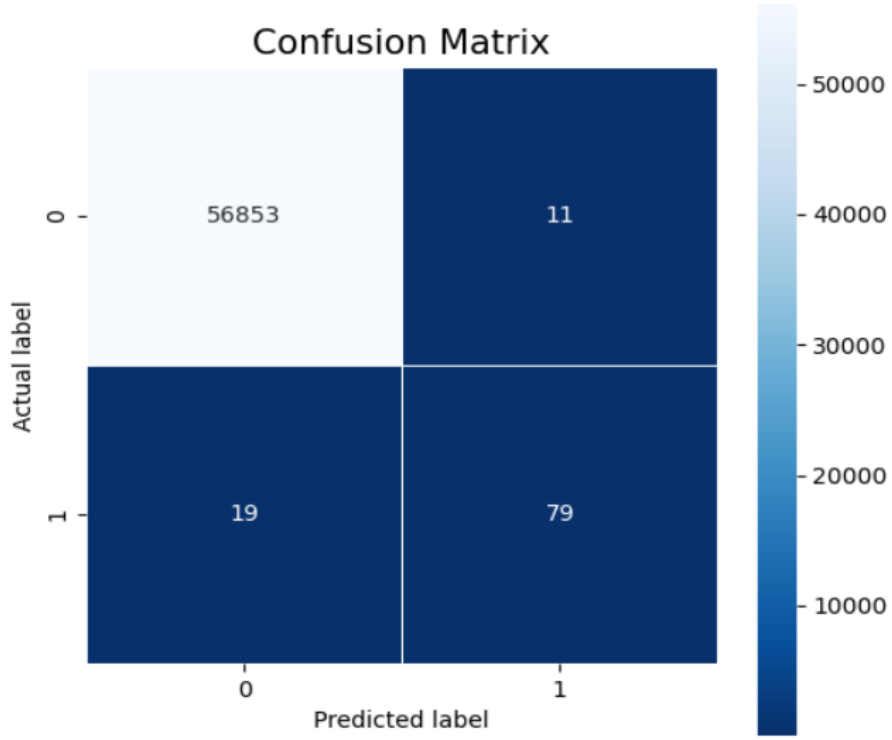
```
import numpy as np
import time
from catboost import CatBoostClassifier
from sklearn.model_selection import cross_validate, StratifiedKFold
from bayes_opt import BayesianOptimization, UtilityFunction

# Create purpose function
def cat_cv(learning_rate, depth, iterations):
    model = CatBoostClassifier(learning_rate=learning_rate,
                               depth=int(round(depth)),
                               iterations=int(round(iterations)),
                               class_weights={0: 1, 1: 592},
                               verbose=False)
    cv = StratifiedKFold(n_splits=5)
    scores = cross_validate(model, X_train, y_train, cv=cv,
scoring='neg_log_loss', verbose=False)
    return np.mean(scores['test_score'])

# Interval to be explored for input values
params = {
    'learning_rate': (0.001, 0.2),
    'depth': (6, 16),
    'iterations': (50, 200)
}
```

```
catBO = BayesianOptimization(cat_cv, params)

# Create an instance of UtilityFunction
utility = UtilityFunction(kind='ei', kappa=2.5, xi=0.0)

# Start the optimization process
start = time.time()
catBO.maximize(init_points=4, n_iter=8, acquisition_function=utility)

# Calculate and print the optimization time
elapsed_time = (time.time() - start) / 60
print(f'It takes {elapsed_time:.2f} minutes')

# Output the best parameters found
params_cat = catBO.max['params']
params_cat['depth'] = round(params_cat['depth'])
params_cat['iterations'] = round(params_cat['iterations'])
print(params_cat)
```

```
|   iter    |  target   |   depth   | iterat... | learni... |
-------------------------------------------------------------
|  1        | -0.1233   |  7.872    |  112.0    |  0.02614  |
|  2        | -0.01612  |  15.35    |  58.46    |  0.1311   |
|  3        | -0.008942 |  10.66    |  90.54    |  0.1884   |
|  4        | -0.01208  |  10.78    |  181.3    |  0.06847  |
|  5        | -0.008999 |  10.78    |  89.55    |  0.1908   |
|  6        | -0.008773 |  12.93    |  194.5    |  0.07313  |
|  7        | -0.6439   |  6.0      |  70.56    |  0.001    |
|  8        | -0.0109   |  16.0     |  51.12    |  0.2      |
|  9        | -0.04287  |  6.231    |  189.4    |  0.07266  |
|  10       | -0.006239 |  16.0     |  187.1    |  0.2      |
|  11       | -0.006342 |  16.0     |  169.6    |  0.2      |
|  12       | -0.5859   |  6.0      |  166.3    |  0.001    |
=============================================================
It takes 90.66689343452454 minutes
{'depth': 16, 'iterations': 187, 'learning_rate': 0.2}
```

```
import catboost as cb
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, matthews_corrcoef, confusion_matrix, roc_auc_score
import matplotlib.pyplot as plt
import seaborn as sns
model = cb.CatBoostClassifier(verbose=0)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
y_pred_proba = model.predict_proba(X_test)[:, 1]
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
mcc = matthews_corrcoef(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred_proba)
```

```
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6,6))
sns.heatmap(cm, annot=True, fmt="d", linewidths=.5, square=True, cmap='Blues_r')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.title('Confusion Matrix', size=15)
plt.show()

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
print("Matthews Correlation Coefficient:", mcc)
print("AUC-ROC:", auc)
```



Model Name	f1	precision	recall	accuracy	roc_auc	mcc
Catboost	0.70975379	0.64950148	0.80960806	0.99881928	0.93856847	0.71897966

6.1.11 MAJORITY VOTING

```
import lightgbm as lgb
from xgboost import XGBClassifier
from catboost import CatBoostClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, matthews_corrcoef
import pandas as pd


lightgbm = lgb.LGBMClassifier(learning_rate=0.182, max_depth=8,
num_leaves=33, class_weight='balanced')
xgboost = XGBClassifier(scale_pos_weight=592, learning_rate=0.1109,
max_depth=6, n_estimators=64)
catboost = CatBoostClassifier(scale_pos_weight=592, verbose=False,
depth=16, learning_rate=0.2)


Model1 = [('lightgbm', lightgbm), ('xgboost', xgboost), ('catboost', catboost)]
Model2 = [('lightgbm', lightgbm), ('xgboost', xgboost)]
Model3 = [('catboost', catboost), ('xgboost', xgboost)]
Model4 = [('lightgbm', lightgbm), ('catboost', catboost)]

voting1 = VotingClassifier(estimators=Model1, voting='soft')
voting2 = VotingClassifier(estimators=Model2, voting='soft')
voting3 = VotingClassifier(estimators=Model3, voting='soft')
voting4 = VotingClassifier(estimators=Model4, voting='soft')


models = [lightgbm, xgboost, catboost, voting1, voting2, voting3, voting4]
names = ["LGBM", "XGB", "CatBoost", "Vot_Lg,Xg,Ca", "Vot_Lg,Xg",
"Vot_Xg,Ca", "Vot_Lg,Ca"]


model_accuracy, model_precision, model_recall, model_f1, model_mcc = [], [],
[], [], []


for model in models:

  y_pred = cross_val_predict(model, X, y, cv=5)
  model_accuracy.append(accuracy_score(y, y_pred))
  model_precision.append(precision_score(y, y_pred))
  model_recall.append(recall_score(y, y_pred))
  model_f1.append(f1_score(y, y_pred))
  model_mcc.append(matthews_corrcoef(y, y_pred))
  model_results = pd.DataFrame({
```

```
        'Model': names,
        'Accuracy': model_accuracy,
        'Precision': model_precision,
        'Recall': model_recall,
        'F1 Score': model_f1,
        'MCC': model_mcc
    })

    print(model_results)
```

	Model	Accuracy	Precision	Recall	F1	MCC	AUC
0	Vot_Lg,Xg	0.998967	0.672389	0.80748	0.727831	0.733337	0.951209
1	Vot_Xg,Ca	0.998851	0.645653	0.807503	0.70975	0.717543	0.950589
2	Vot_Lg,Ca	0.999105	0.723251	0.809608	0.75672	0.761114	0.942206
3	Vot_Lg,Xg,Ca	0.998967	0.672389	0.80748	0.727831	0.733337	0.951209

6.1.12 DEEPLEARNING - ANN

```
def generate_model(batch_size, epochs, neuronPct):
model = Sequential()
neurons = int(neuronPct * 100)
# So long as there would have been at least 20 neurons and fewer than 5layers,
create a new layer.
layer = 0
while  round(neurons)>20 and layer <5:
# The first (0th) layer needs an input input_dim(neuronCount)
if layer==0:
model.add(Dense(neurons,input_dim=31 , activation= 'relu',
kernel_initializer='he_uniform'))
else:
model.add(Dense(neurons, activation='relu'))

layer += 1
neurons = round((neurons +1)/2)

model.add(Dense(1,activation='sigmoid')) # Output
return model

def keras_cv(batch_size, epochs, neuronPct):
 cv = StratifiedKFold(n_splits=5)
 num = 0
 for train, test in cv.split(X_train1, y_train1):
```

```
num+=1
# Split train and test
x_train = X_train1[train]
Y_train = y_train1[train]
x_test = X_train1[test]
Y_test = y_train1[test]
model = generate_model(batch_size, epochs, neuronPct)
opt = tf.keras.optimizers.Adam() #optimizer
model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
callback = tf.keras.callbacks.EarlyStopping(monitor="val_loss")
params_dl['batch_size'] = round(batch_size)
params_dl['epochs'] = round(epochs)
params_dl['neuronPct'] = neuronPct
model.fit(x_train,Y_train,validation_split=0.1
,class_weight={0:0.25, 1:148}
,callbacks=[callback]
,verbose=0,epochs=round(epochs), batch_size=round(batch_size))

# Predict on the out of boot (validation)
proba = model.predict(x_test)
pred = (model.predict(x_test)> 0.5)
score = metrics.log_loss(Y_test, pred)
        tensorflow.keras.backend.clear_session()
return (-(np.mean(score)))

params_dl={'batch_size' : (1000,3500),
'epochs' : (25,150),
'neuronPct': (0.51, 1)
}

dlBO = BayesianOptimization(keras_cv, params_dl, verbose=2)
# verbose = 1 prints only when a maximum
# is observed, verbose = 0 is silent
start = time.time()
dlBO.maximize(init_points=6, n_iter =10, acq='ei')

print('It takes %s minutes' % ((time.time() - start)/60))

params_dl = dlBO.max['params']
params_dl['epochs'] = round(params_dl['epochs'])
params_dl['batch_size'] = round(params_dl['batch_size'])
params_dl['neuronPct'] = round((params_dl['neuronPct']), 2)
print(params_dl)
```

iter	target	batch_...	epochs	neuronPct
1	-0.8651	1.809e+0	121.8	0.5771
2	-0.9267	1.341e+0	84.61	0.5109
3	-0.6224	1.495e+0	90.3	0.9427
4	-1.926	2.808e+0	128.1	0.9518
5	-1.763	1.517e+0	55.41	0.5107
6	-0.9252	2.213e+0	78.43	0.9365
7	-2.414	2.211e+0	81.87	0.6905
8	-0.8636	2.371e+0	28.25	0.8228
9	-1.993	3.181e+0	120.4	0.9661
10	-0.7647	3.315e+0	39.42	0.6542
11	-1.135	2.17e+03	104.7	0.6396
12	-1.061	2.776e+0	36.69	0.8304
13	-1.171	2.085e+0	32.66	0.7278
14	-0.4953	1.563e+0	117.4	0.8648
15	-1.216	1.356e+0	135.9	0.7216
16	-1.457	2.721e+0	140.6	0.643

```
It takes 8.144162921110789 minutes
{'batch_size': 1563, 'epochs': 117, 'neuronPct': 0.86}
```

```
model = generate_model(batch_size= 1563 ,epochs=117 ,neuronPct=0.86)
model.summary()
opt = tf.keras.optimizers.Adam() #optimizer
model.compile(optimizer=opt,          loss=tf.keras.losses.BinaryCrossentropy(),
metrics=['accuracy'])
history = model.fit(X_train1, y_train1, epochs =117 ,
batch_size=1563,validation_split = 0.1
,class_weight={0:0.25, 1:148}
,verbose = 0)
history_dict = history.history
y_prob = model.predict(X_test1)
y_pred = (model.predict(X_test1)> 0.5).astype("int32")


print(classification_report(y_test1 , y_pred))
print(confusion_matrix(y_test1 , y_pred))

print('\n')
print('--------- Summary ----------')
print('\n')
print('accuracy: {}'.format(round(accuracy_score(y_test1, y_pred),4)))
print('Precision: {}'.format(round(precision_score(y_test1, y_pred),4)))
print('f1: {}'.format(round(metrics.f1_score(y_test1, y_pred),4)))
print('Recall: {}'.format(round(recall_score(y_test1, y_pred),4)))
print('AUC: {}'.format(round(roc_auc_score(y_test1, y_prob),4)))
print('Precision-Recall: {}'.format(round(average_precision_score
      (y_test1, y_prob),4)))
print('MCC: {}'.format((matthews_corrcoef(y_test1, y_pred))))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56656
           1       0.80      0.82      0.81        90

    accuracy                           1.00     56746
   macro avg       0.90      0.91      0.91     56746
weighted avg       1.00      1.00      1.00     56746

[[56638    18]
 [   16    74]]


--------- Summary ------------


accuracy: 0.9994
Precision: 0.8043
f1: 0.8132
Recall: 0.8222
AUC: 0.9401
Precision-Recall: 0.7922
MCC: 0.8129360313935904
```

CHAPTER 7

CONCLUSION

6.1 CONCLUSION

Here, in our comprehensive study on credit card fraud detection using an imbalanced dataset, we deployed advanced ensemble methods and scrutinized their efficacy through meticulous analysis. The side-by-side comparison of the models, as visually represented in our results, indicates a nuanced understanding of how different algorithms capture fraud patterns. Notably, the incorporation of class weight tuning in LightGBM catalyzed a substantial leap in detecting fraudulent transactions, revealing an enhancement in detection rates over previous benchmarks.The amalgamation of individual model strengths via a majority voting system not only consolidated detection accuracy but also imparted resilience to our evaluation metrics. This strategy's merit is evident in the substantial uplift in the F1-score, which signifies a more balanced approach between precision and recall—a critical factor in the context of fraud detection. Our empirical evidence suggests that the Matthew's Correlation Coefficient (MCC) emerges as a robust indicator, especially beneficial in the realm of heavily skewed datasets. Furthermore, hybridizing LightGBM and XGBoost with deep learning approaches exhibited promising prospects, registering MCC scores that underscore the potential of deep learning in this domain.

CHAPTER 8

REFERENCES

8.1 REFERENCES

[1] J. Nanduri, Y.-W. Liu, K. Yang, and Y. Jia, ‘‘Ecommerce fraud detection through fraud islands and multi-layer machine learning model,’’ in *Proc. Future Inf. Commun. Conf.*, in Advances in Information and Communica- tion. San Francisco, CA, USA: Springer, 2020, pp. 556–570.

[2]I. Matloob, S. A. Khan, R. Rukaiya, M. A. K. Khattak, and

A. Munir, ‘‘A sequence mining-based novel architecture for detecting fraudulent transactions in healthcare systems,’’ *IEEE Access*, vol. 10, pp. 48447–48463, 2022.

[3]H. Feng, ‘‘Ensemble learning in credit card fraud detection using boosting methods,’’ in *Proc. 2nd Int. Conf. Comput. Data Sci. (CDS)*, Jan. 2021, pp. 7–11.

[4]M. S. Delgosha, N. Hajiheydari, and S. M. Fahimi, ‘‘Elucidation of big data analytics in banking: A four-stage delphi study,’’ *J. Enterprise Inf. Manage.*, vol. 34, no. 6, pp. 1577–1596, Nov. 2021.

[5]M. Puh and L. Brkić, ‘‘Detecting credit card fraud using selected machine learning algorithms,’’ in *Proc. 42nd Int. Conv. Inf. Commun. Technol., Electron. Microelectron. (MIPRO)*, May 2019, pp. 1250–1255.

[6]K. Randhawa, C. K. Loo, M. Seera, C. P. Lim, and A. K. Nandi, ‘‘Credit card fraud detection using AdaBoost and majority voting,’’ *IEEE Access*, vol. 6, pp. 14277–14284, 2018.

[7]N. Kumaraswamy, M. K. Markey, T. Ekin, J. C. Barner, and K. Rascati, ‘‘Healthcare fraud data mining methods: A look back and look ahead,’’ *Perspectives Health Inf. Manag.*, vol. 19, no. 1, p. 1, 2022.

[8]E. F. Malik, K. W. Khaw, B. Belaton, W. P. Wong, and X. Chew, ‘‘Credit card fraud detection using a new hybrid machine learning architecture,’’ *Mathematics*, vol. 10, no. 9, p. 1480, Apr. 2022.

[9]K. Gupta, K. Singh, G. V. Singh, M. Hassan, G. Himani, and U. Sharma, ‘‘Machine learning based credit card fraud detection—A review,’’ in *Proc. Int. Conf. Appl. Artif. Intell. Comput. (ICAAIC)*, 2022, pp. 362–368.

[10] R. Almutairi, A. Godavarthi, A. R. Kotha, and E. Ceesay, ‘‘Analyzing credit card fraud detection based on machine learning models,’’ in *Proc. IEEE Int. IoT, Electron. Mechatronics Conf. (IEMTRONICS)*, Jun. 2022, pp. 1–8.