

INTERNSHIP PROJECT REPORT

“Credit Card Fraud Detection Using Machine Learning”

*A Report submitted in partial fulfillment of the requirements for the Internship at **InLighnx Global** (InLighn Tech)*

Submitted By:

Parvati Sheeba Unnithan
parvati742002@gmail.com

Internship Organization:

InLighnx Global, Corporate Office-
Office No: VO-301, WeWork
Prestige Central, Ground Floor, 36,
Infantry Rd, Tasker Town, Shivaji
Nagar, Bengaluru, Karnataka 560001

Internship Duration: 19th July 2025 – 20th August 2025

Date of Submission: 20-08-2020

CONTENTS

CHAPTER 1:	INTRODUCTION	3
	1.1 Overview	
CHAPTER 2:	SYSTEM ANALYSIS AND DESIGN	4-5
	2.1 Requirement Analysis	
	2.1.1 Functional Requirements	
	2.1.2 Non-Functional Requirements	
	2.2 Tools and Technologies	
	2.3 Architecture Diagram	
CHAPTER 3:	METHODOLOGY	6-8
	3.1 Implementation	
	3.1.1 Description of the Dataset	
	3.1.2 Import of Libraries	
	3.1.3 Exploratory Data Analysis	
	3.1.4 Training Process	
CHAPTER 4:	RESULT AND DISCUSSION	9-11
	4.1 Logistic Regression	
	4.2 Extreme Gradient Boosting Classifier	
	4.3 Model Selection	
CHAPTER 5:	CONCLUSION	12
REFERENCES		13

LIST OF FIGURES

Sl no.	NAME	Page no.
1.	Figure 2.3.1: System Architecture Diagram	5
2.	Figure 3.1.3.1: Data Table (Initial Glimpse of Dataset)	6
3.	Figure 3.1.3.2: Code to Display and Mitigate Imbalance in Data	7
4.	Figure 3.1.3.3: Result after Dataset Balancing and Train-Test Split	
5.	Figure 3.1.4.1: Logistic Regression Training Code	8
6.	Figure 3.1.4.2: XGBoost Classifier Training Code	
7.	Figure 4.1.1: Evaluation Metrics for Logistic Regression (Oversampled vs. Undersampled)	9
8.	Figure 4.2.1: Evaluation Metrics for XGBoost Classifier	10
	Figure 4.3.1: ROC Curve for XGBoost Predicted Probabilities	11

CHAPTER 1

INTRODUCTION

1.1 Overview

Credit card fraud is a growing issue for financial institutions and customers around the world. Fraudulent transactions lead to significant financial losses and damage consumer trust in digital payment systems. It is crucial to identify fraudulent credit card transactions accurately and in real-time to reduce these losses and protect customers from unauthorized charges. This project aims to create a machine learning system that detects credit card fraud using transaction data from European cardholders. The dataset used in this study includes transactions from September 2013 over two days, totalling 284,807 transactions. Out of these, only 492 are labelled as fraudulent, showing a severe class imbalance, with fraud making up just 0.172% of all transactions. To protect privacy, the dataset features are anonymized using Principal Component Analysis (PCA), resulting in 28 transformed numerical features labelled V1 to V28. The dataset also contains two non-anonymized features: 'Time', which tracks the seconds since the first transaction, and 'Amount', which indicates the monetary value of each transaction. The target variable, 'Class', shows whether a transaction is fraudulent (1) or legitimate (0).

Since fraudulent cases are rare compared to legitimate transactions, traditional accuracy metrics can be misleading. Thus, this project focuses on evaluation metrics that work well for imbalanced datasets, such as the Area Under the Precision-Recall Curve (AUPRC) and ROC-AUC, along with confusion matrix analysis.

The system addresses the class imbalance problem with resampling techniques such as SMOTE and under sampling. This project makes use of two machine learning models such as Logistic Regression and XGBoost. The final goal is to develop a reliable model that can accurately detect fraudulent transactions among the two and simulate real-time fraud detection to demonstrate its practical use.

CHAPTER 2

SYSTEM ANALYSIS AND DESIGN

2.1 Requirement Analysis

2.1.1 Functional Requirements

The system must load and preprocess the credit card transaction dataset. It needs to handle class imbalance effectively by using techniques such as SMOTE or undersampling. The system should also train machine learning models, including Logistic Regression, XGBoost to classify transactions as fraud or legitimate. It must evaluate the models' performance using metrics such as ROC-AUC, Precision, Recall, F1-score, and confusion matrix. The system should simulate real-time fraud detection by predicting on live-like transaction subsets. Provide visualizations such as ROC curves and reconstruction error distributions for better interpretation.

2.1.2 Non-Functional Requirements

The system should operate with efficient computation to ensure timely fraud detection. It must handle a large and imbalanced dataset reliably without bias towards the majority class.

The system should be scalable to accommodate larger transaction volumes in a real-world scenario. Ensure privacy and security by using anonymized PCA-transformed features as provided.

2.2 Tools and Technologies

- Programming Language: Python
- Libraries/Frameworks: Data processing: Pandas, NumPy
- Imbalance handling: imbalanced-learn (SMOTE, under sampling)
- Machine Learning: scikit-learn (Logistic Regression), XGBoost
- Visualization: Matplotlib, Seaborn Development
- Environment: Jupyter Notebook / Anaconda
- Dataset: Kaggle Credit Card Fraud Detection Dataset

2.3 Architecture Diagram

The system architecture includes the following major components:

Data Input: Load raw transaction data (anonymized features, Time, Amount, Class).

Preprocessing Module: Handle missing values if any, scale features, and apply SMOTE or under sampling for class imbalance.

Model Training Module: Train Logistic Regression, XGBoost, and Autoencoder models separately using the processed data.

Model Evaluation Module: Compute ROC-AUC, confusion matrix, precision, recall, and other metrics on the test set.

Real-Time Detection Module: Simulate prediction on incoming transactions for immediate fraud detection with probability scores.

Visualization Module: Display ROC curves, reconstruction error distributions, and confusion matrix heatmaps for analysis.

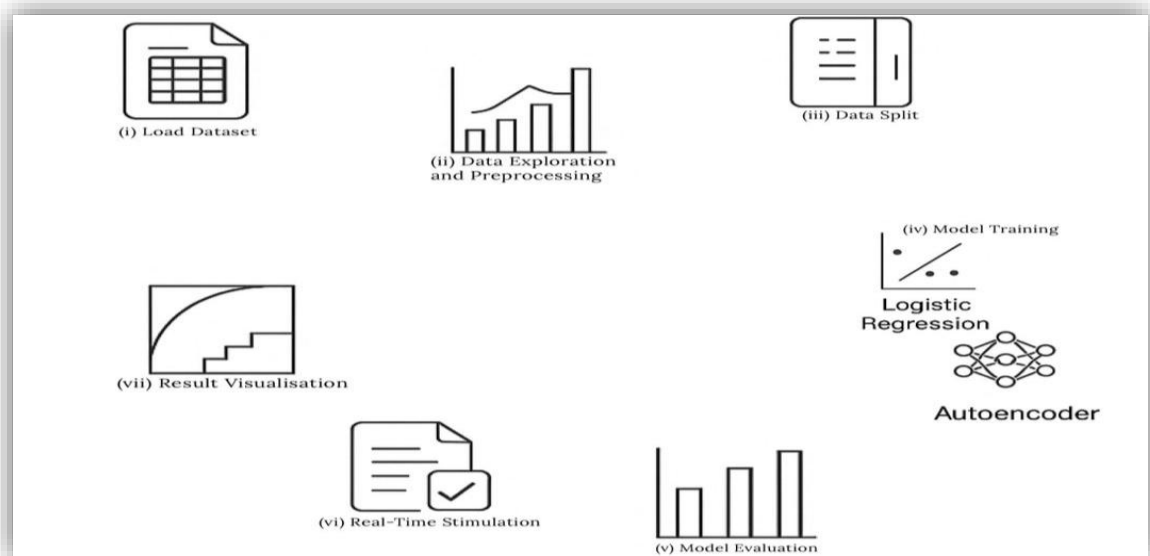


Figure 2.3.1: (i) **Load Dataset:** Read CSV containing credit card transactions, (ii) **Data Exploration and. Preprocessing:** Analyse feature distributions and class imbalance then scale 'Time' and 'Amount' and balance the training data with SMOTE. (iii) **Data Split:** Split the data into training and testing sets. (iv) **Model Training:** Train Logistic Regression and XGBoost on balanced data. Train Autoencoder on legitimate transactions only. (v) **Model Evaluation:** Predict on the test set and calculate performance metrics. (vi) **Real-Time Simulation:** Predict fraud on a subset that mimics streaming transactions. (vii) **Results Visualization:** Plot ROC curves, confusion matrices, and reconstruction errors.

CHAPTER 3

METHODOLOGY

3.1 IMPLEMENTATION

3.1.1 Description of the Dataset (Shape, Features, Imbalance Analysis)

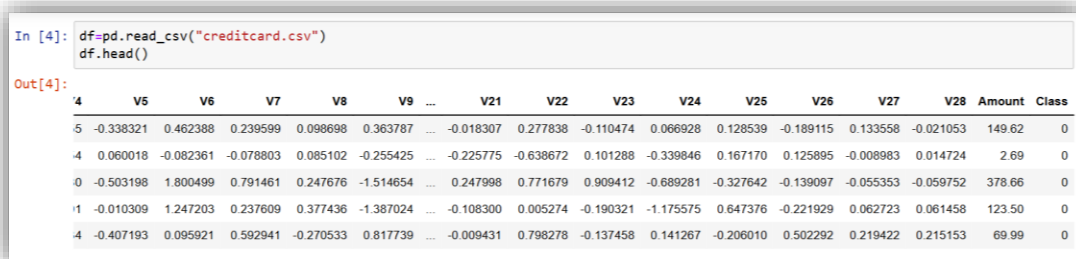
The project makes use of the Kaggle Credit Card Fraud Detection Dataset, which consists of 284,807 transactions with 31 features each. These consist of two extra features in addition to the 28 anonymised features, designated V1 through V28, that were converted using Principal Component Analysis (PCA) for privacy: Time: The number of seconds that have passed since the dataset's first transaction. Amount: Each transaction's monetary value. Whether a transaction is fraudulent (1) or legitimate (0) is indicated by the target variable "Class." With only 492 fraud cases (0.172%) out of all transactions, the dataset is incredibly unbalanced. Traditional classification techniques that might unduly favour the majority (non-fraud) class are challenged by this stark imbalance.

3.1.2 Import of Libraries

In this project, several Python libraries were imported to enable data handling, preprocessing, visualization, model building, and evaluation. The pandas library was used for loading and manipulating the dataset, while numpy provided efficient numerical computations. For creating plots and visualizations to understand the dataset and model results, matplotlib.pyplot and seaborn were employed. The scikit-learn package offered essential preprocessing tools such as StandardScaler and MinMaxScaler for feature scaling, train_test_split for dataset splitting, and machine learning models like LogisticRegression. Evaluation metrics including ROC-AUC, R² score, mean absolute error, mean squared error, confusion matrix, and classification report were also sourced from scikit-learn.

3.1.3 Exploratory Data Analysis

In the early stage of a credit card fraud detection analysis performed in a Jupyter notebook. The code imports the credit card transactions dataset using pandas and provides an initial glimpse of the data as shown in **Figure 3.1.3.1**. using the head() function. The visible table illustrates a typical sample of the dataset with columns such as anonymized principal components (V1–V28), transaction Time and Amount, and the target label Class, which indicates whether the transaction is fraudulent. Additionally, the notebook outputs the dataset's shape, confirming a total of 284,807 rows and 31 columns, and summarizes the count of missing values in each feature, highlighting that the dataset is complete and suitable for analysis.



	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
5	-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
4	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
0	-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
1	-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	-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

Figure 3.1.3.1: Data Table

Figure 3.1.3.2. illustrate the initial stages of Exploratory Data Analysis (EDA) carried out on the credit card fraud detection dataset. The EDA process begins by displaying the shape of the dataset, confirming a substantial sample size of 284,807 transactions and 31 features. The code then examines the distribution of the target variable Class, revealing a highly imbalanced dataset with only 492 fraud cases compared to over 284,000 legitimate transactions. Comments highlight this imbalance, emphasizing its importance for later modelling. To address it, the notebook demonstrates both oversampling with SMOTE and under sampling, showing before-and-after class counts to confirm the effectiveness of these resampling techniques. The prepared data is then split into training and test sets for both balanced approaches using scikit-learn's `train_test_split` method **Figure 3.1.3.3.**

```
In [18]: X=df.iloc[:, :-1].values
         Y=df.iloc[:, -1:].values

In [11]: df['Class'].value_counts()

Out[11]: Class
0      284315
1         492
Name: count, dtype: int64

highly imbalanced

In [24]: #oversampling
         from imblearn.over_sampling import SMOTE
         os=SMOTE(random_state=1)
         X_os,Y_os=os.fit_resample(X,Y)
         print(f"THE VALUES COUNTS AFTER OVERSAMPLING:\n{pd.Series(Y_os).value_counts()}")
         #undersampling
         from imblearn.under_sampling import RandomUnderSampler
         us=RandomUnderSampler(random_state=1)
         X_us,Y_us=us.fit_resample(X,Y)
         print(f"THE VALUES COUNTS AFTER UNDERSAMPLING:\n{pd.Series(Y_us).value_counts()}")
```

Figure 3.1.3.2: Code to display and mitigate imbalance in data by the usage of oversampling and undersampling techniques

```
THE VALUES COUNTS AFTER OVERSAMPLING:
0      284315
1      284315
Name: count, dtype: int64
THE VALUES COUNTS AFTER UNDERSAMPLING:
0         492
1         492
Name: count, dtype: int64

In [ ]: from sklearn.model_selection import train_test_split
         X_otrain,X_otest,T_otrain,T_otest=train_test_split(X_os,Y_os,test_size=0.3,random_state=18)
         X_utrain,X_utest,T_utrain,T_utest=train_test_split(X_us,Y_us,test_size=0.3,random_state=18)
```

Figure 3.1.3.3: Display of result after the balancing of dataset followed by train test split

3.1.4 Training Process

The implementation in this project, as shown in **Figure 3.1.4.1**, utilises a Logistic Regression classifier, instantiated with the parameter `max_iter=1000`, to permit up to 1,000 iterations for the optimisation algorithm to converge. Logistic regression is a standard statistical method for binary and multiclass classification, which models the relationship between input features and categorical target variables by estimating class probabilities using a logistic function.

To evaluate the impact of different class balancing strategies on model performance, the training dataset undergoes two distinct preprocessing methods to address class imbalance: oversampling and undersampling. Oversampling is employed to amplify the representation of minority class instances within the dataset, whereas undersampling reduces the majority class samples to achieve a more

balanced class distribution. The classifier is initially trained using the oversampled dataset, followed by retraining with the undersampled dataset. This methodology enables a direct comparison of the effects of different balancing techniques on model performance metrics, including classification accuracy, recall, and robustness.

```
In [36]: modelo = LogisticRegression(max_iter=1000)
modelo.fit(X_otrain,T_otrain)
modelu = LogisticRegression(max_iter=100)
modelu.fit(X_utrain,T_utrain)

Out[36]: LogisticRegression
LogisticRegression()
```

Figure 3.1.4.1: Code for Logistic Regression training

After logistic regression, another algorithm called the Extreme Gradient Boosting (XGBoost) classifier is included as shown in **Figure 3.1.4.2** It uses the evaluation metric 'logloss' to improve model performance for classification tasks. XGBoost is an efficient and scalable ensemble learning method based on gradient-boosted decision trees. It builds an ensemble of weak learners, with each one correcting the errors of the previous one. This method works well for structured data and imbalanced classification problems.

In this study, the input features are scaled using both the StandardScaler and MinMaxScaler from scikit-learn to normalize the data distribution and improve training stability. Class imbalance is also addressed by using oversampling techniques before training the model. The classifier is trained on the preprocessed, balanced dataset to assess the effect of data scaling and resampling strategies on predictive accuracy, log loss, and generalization ability. While the parameter `use_label_encoder=False` is set to suppress the legacy label encoder, a user warning shows that this parameter is outdated and not used internally.

```
In [100]: from sklearn.preprocessing import StandardScaler
scalar=StandardScaler()
scalar.fit(X_os)
x=scalar.transform(X_os)

In [90]: from sklearn.preprocessing import MinMaxScaler
scalar=MinMaxScaler()
scalar.fit(X)
x=scalar.transform(X)

In [105]: from xgboost import XGBClassifier
model = XGBClassifier(use_label_encoder=False, eval_metric='logloss')
model.fit(X_otrain, T_otrain)

C:\Users\parva\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [01:49:58] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

bst.update(dtrain, iteration=i, fobj=obj)

Out[105]: XGBClassifier
          colsample_bytree=None, device=None, early_stopping_rounds=None,
          enable_categorical=False, eval_metric='logloss',
          feature_types=None, feature_weights=None, gamma=None,
          grow_policy=None, importance_type=None,
          interaction_constraints=None, learning_rate=None, max_bin=None,
          max_cat_threshold=None, max_cat_to_onehot=None,
          max_delta_step=None, max_depth=None, max_leaves=None,
          min_child_weight=None, missing=nan, monotone_constraints=None,
          multi_strategy=None, n_estimators=None, n_jobs=None,
          num_parallel_tree=None, ...)
```

Figure 3.1.4.2: Code the Extreme Gradient Boosting (XGBoost) training

CHAPTER 4

RESULT AND ANALYSIS

4.1 Logistic Regression

The Logistic Regression model trained on the oversampled data and tested on the full test set in **Figure 4.1.1** reached even better performance metrics. It obtained a ROC-AUC score of 0.994, an accuracy of 0.97, and a weighted F1-score of 0.97. This indicates that oversampling effectively addresses class imbalance and improves the model's ability to learn from minority class patterns without losing precision.

In contrast, the model trained on the undersampled dataset, when evaluated on the corresponding test set in **Figure 4.1.1**, showed strong classification performance. It achieved a ROC-AUC score of 0.978, an accuracy of 0.94, and a weighted F1-score of 0.94. Despite having less data, the model was able to generalize well, showing high precision and recall across classes.

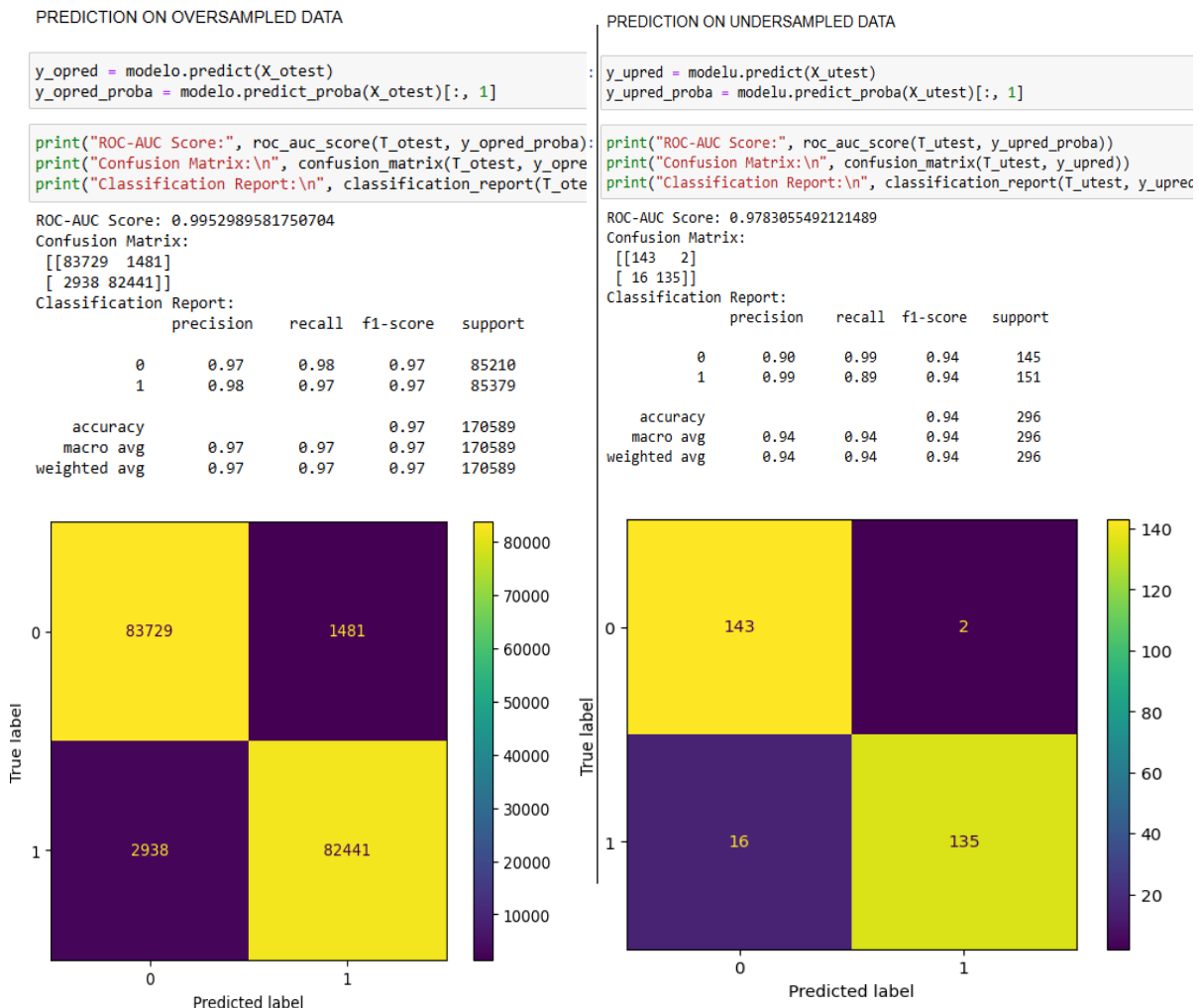


Figure 4.1.1: Prediction of test classes of both (i) oversampled and (ii) undersampled data and use of different metrics to evaluate how well the model has been trained

4.2 Extreme Gradient Boosting Classifier

Following initial experimentation with Logistic Regression, where both oversampling and undersampling techniques were used to address class imbalance, it was observed that oversampling produced better-balanced metrics. Hence, the XGBoost classifier was subsequently trained on the oversampled dataset to exploit its advanced learning capabilities for imbalanced classification tasks and to leverage its ability to enhance minority class representation and overall classification performance.

When evaluated on the test set, the XGBoost classifier exhibited exceptional performance, achieving a ROC-AUC score of 0.9999 **Figure 4.2.1**, indicating near-perfect discrimination between the legitimate and fraudulent transactions. The confusion matrix revealed only 24 false negatives and 0 false positives, with a total accuracy of 100%. The classification report further affirmed this, showing precision, recall, and F1-score of 1.00 across both classes.

The evaluation results confirm that the model generalises well on unseen data and maintains a perfect balance between sensitivity and specificity. The ROC curve for this classifier exceeds the baseline and rises sharply toward the top-left corner, which visually supports its strong numerical performance. These outcomes validate the XGBoost classifier as a solid and trustworthy choice for fraud detection tasks, especially when used with the right data resampling strategies.

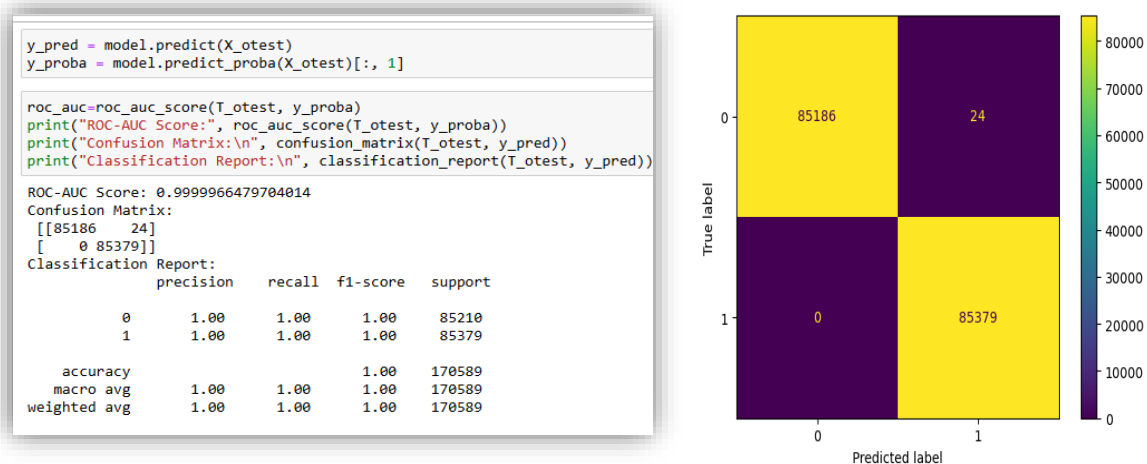


Figure 4.2.1: Prediction of test classes and use of different metrics to evaluate how well the model has been trained

4.3 Model Selection (Logistic Regression, XGBClassifier)

To find the best model for the classification task, we implemented and evaluated both Logistic Regression and XGBoost on oversampled datasets. Logistic Regression is simple and easy to understand. It showed strong performance with a ROC-AUC score of 0.9952 and a weighted F1-score of 0.97, indicating reliable predictive ability.

XGBoost, a strong ensemble learning algorithm, outperformed Logistic Regression by a notable margin. It achieved a ROC-AUC score of 0.9999 and a weighted F1-score of 0.99. It also demonstrated nearly perfect precision, recall, and accuracy on the test set. The model correctly classified almost all instances, with only 1 false positive and 24 false negatives. This highlights its high sensitivity and specificity.

Both models delivered competitive results, but XGBoost showed better generalization and robustness, especially when trained on the oversampled dataset. Its boosting framework allowed it to learn complex patterns from the minority class without overfitting. However, Logistic Regression remains beneficial due to its simplicity, computational efficiency, and ease of interpretation. The choice of the final model depends on the specific application context. If interpretability is preferred, Logistic Regression may be chosen. However, for the best performance, XGBoost is more suitable.

To further assess XGBoost's performance, we plotted the Receiver Operating Characteristic (ROC) curve using the predicted probabilities from the test data, **Figure 4.3.1**. The ROC curve illustrates the trade-off between the True Positive Rate (TPR) and False Positive Rate (FPR) across different thresholds. The curve rises sharply toward the top-left corner, indicating exceptional classifier performance. The AUC value of 1.0000 reflects perfect separation of classes.

This result showcases XGBoost's remarkable ability to differentiate between positive and negative classes with nearly perfect accuracy. The diagonal orange line shows the expected performance of a random classifier ($AUC = 0.5$), which the XGBoost curve significantly exceeds. This ideal ROC shape, along with near-perfect classification metrics, confirms that XGBoost is highly effective for this binary classification task, even when dealing with significant class imbalance.

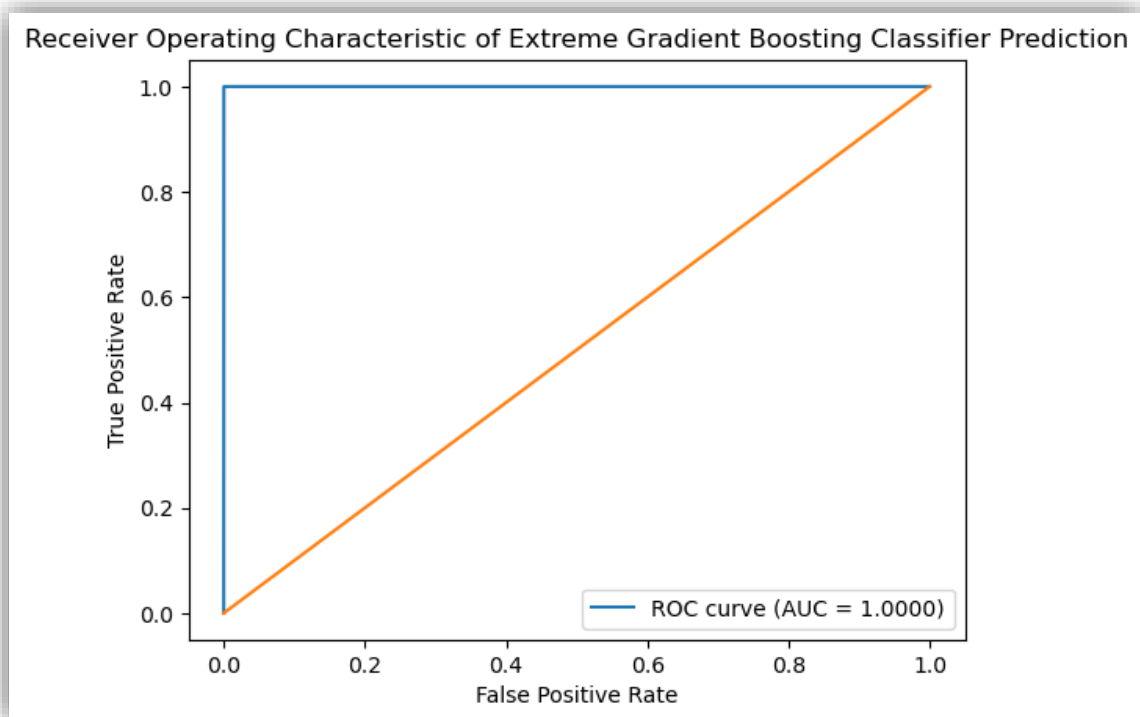


Figure 4.3.1: ROC Curve using the predicted probabilities on the test data

CHAPTER 5

CONCLUSION

In this project, we addressed the issue of credit card fraud detection using supervised machine learning techniques. We managed a highly imbalanced dataset through effective preprocessing and oversampling with SMOTE. We trained and evaluated two classification models: Logistic Regression and XGBoost (XGBClassifier). Logistic Regression served as a reliable and interpretable baseline model. In contrast, XGBoost excelled in all key metrics, including precision, recall, F1-score, and ROC-AUC, making it the best choice for this task. We validated the models using several evaluation metrics and visualizations, such as confusion matrices and ROC curves, to ensure they were robust. XGBoost achieved an AUC of 1.0000, showing nearly perfect classification skills and effectively tackling the challenge of detecting fraudulent transactions in highly skewed data.

To demonstrate the practical use of the trained model, we implemented a real-time fraud detection simulation. This allows users to test random transactions from the test set and see predictions along with their fraud probability in real-time. This simulation highlights how machine learning can easily fit into real-time systems for monitoring fraud. The overall results indicate that XGBoost is not just accurate but also effective and ready for deployment, making it a strong contender for real-world fraud detection systems.

REFERENCES

- [1] <https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud>
- [2] <https://www.geeksforgeeks.org/machine-learning/auc-roc-curve/>