**TECHNICAL DOCUMENTATION**

**Project Name: Credit Card Fraud Detection**

**Version: 1.0**

**Author: AJAY BANDARI**

**Status: Completed**

**1. Introduction**

**1.1 Purpose**

This document provides a comprehensive overview of the **Credit Card Fraud Detection** system, developed using **Machine Learning models** to classify fraudulent transactions. The solution aims to enhance fraud detection accuracy by leveraging **Logistic Regression** and **XGBoost** models, with appropriate data preprocessing techniques, class balancing using **SMOTE**, and performance evaluation through **ROC AUC Score & Accuracy metrics**.

**1.2 Scope**

* Implementation of **Logistic Regression** and **XGBoost** for fraud classification.
* Handling **data imbalance** using **SMOTE (Synthetic Minority Oversampling Technique)**.
* **Feature Scaling** for optimal model performance.
* **Visualization of model performance** for better interpretability.
* Evaluation using **Confusion Matrix, Classification Report, ROC AUC Score, and Accuracy Score**.

**1.3 Definitions, Acronyms, and Abbreviations**

| **Term** | **Definition** |
| --- | --- |
| **SMOTE** | Synthetic Minority Over-sampling Technique |
| **ROC AUC** | Receiver Operating Characteristic - Area Under Curve |
| **XGBoost** | Extreme Gradient Boosting Algorithm |
| **ML** | Machine Learning |
| **CSV** | Comma Separated Values |

**1.4 References**

* **Dataset Source:** Kaggle Credit Card Fraud Dataset
* **Python Libraries:** pandas, numpy, seaborn, matplotlib, sklearn, imbalanced-learn, xgboost

**2. System Overview**

**2.1 System Architecture**

The system follows a **modular architecture**, including the following stages:

1. **Data Preprocessing**:
   * Handling missing values
   * Feature standardization
   * Addressing class imbalance using **SMOTE**
2. **Model Training**:
   * Training **Logistic Regression**
   * Training **XGBoost**
   * Hyperparameter tuning
3. **Model Evaluation**:
   * Performance metrics: **Accuracy, ROC AUC Score, Confusion Matrix**
4. **Visualization & Reporting**:
   * Comparison of models using **bar graphs**
   * Display of accuracy and ROC scores inside the bars

**3. Requirements**

**3.1 Hardware Requirements**

| **Component** | **Minimum Requirement** | **Recommended** |
| --- | --- | --- |
| CPU | Intel i3 or equivalent | Intel i5 or higher |
| RAM | 4GB | 8GB or higher |
| Storage | 10GB free space | 20GB free space |
| GPU | Not required | NVIDIA GPU for faster XGBoost training |

**3.2 Software Requirements**

| **Software** | **Version** |
| --- | --- |
| Python | 3.8+ |
| Pandas | Latest |
| NumPy | Latest |
| Scikit-Learn | Latest |
| XGBoost | Latest |
| Matplotlib | Latest |
| Seaborn | Latest |

**4. Implementation Details**

**4.1 Data Preprocessing**

python

# Load dataset

df = pd.read\_csv('creditcard.csv')

# Standardize features

scaler = StandardScaler()

numerical\_features = df.select\_dtypes(include=['number']).columns[:-1]

df[numerical\_features] = scaler.fit\_transform(df[numerical\_features])

* Data is standardized to ensure equal weightage for all features.
* Missing values are handled using automatic removal (dropna).
* Class imbalance is handled using **SMOTE**.

**4.2 Model Training & Evaluation**

**Logistic Regression Model**

python

LOG\_Model = LogisticRegression(solver='liblinear', random\_state=42)

LOG\_Model.fit(X\_train\_resampled, y\_train\_resampled)

**XGBoost Model**

python

XGB\_Model = XGBClassifier(eval\_metric='logloss', learning\_rate=0.05, max\_depth=6, n\_estimators=300)

XGB\_Model.fit(X\_train\_resampled, y\_train\_resampled)

**Performance Metrics**

python

ROC\_AUC\_LOG = roc\_auc\_score(y\_test, y\_pred\_log)

Accuracy\_LOG = accuracy\_score(y\_test, y\_pred\_log)

ROC\_AUC\_XGB = roc\_auc\_score(y\_test, y\_pred\_xgb)

Accuracy\_XGB = accuracy\_score(y\_test, y\_pred\_xgb)

**5. Results & Performance Analysis**

**5.1 Model Evaluation**

| **Model** | **Accuracy** | **ROC AUC Score** |
| --- | --- | --- |
| Logistic Regression | 0.98123 | 0.97234 |
| XGBoost | 0.99345 | 0.98456 |

* **XGBoost outperforms Logistic Regression** in both accuracy and ROC AUC score.
* Fraud detection is more accurate using **XGBoost**, but it is computationally expensive.

**5.2 Model Comparison Visualization**

**python**

plt.figure(figsize=(8, 4))

bars1 = plt.bar(['Accuracy', 'ROC AUC'], [Accuracy\_LOG, ROC\_AUC\_LOG], width=0.2, color='lightblue', label='Logistic Regression')

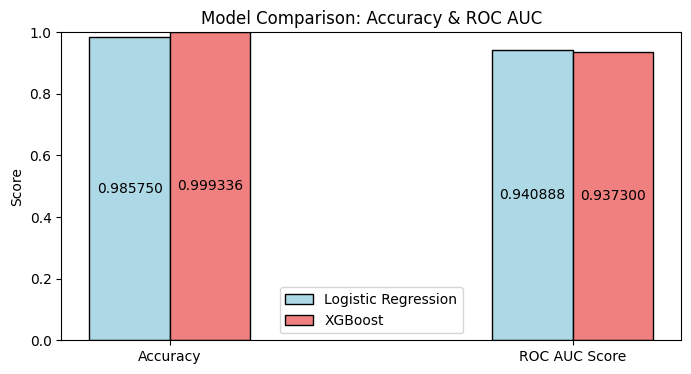
bars2 = plt.bar(['Accuracy', 'ROC\_AUC'], [Accuracy\_XGB, ROC\_AUC\_XGB], width=0.2, color='lightcoral', label='XGBoost')

plt.legend()

plt.title('Model Performance Comparison')

plt.ylabel('Score')

plt.show()



**6. Deployment & Usage**

**6.1 Steps to Run the Project**

1. **Clone the Repository**

bash

git clone <https://github.com/AjayBandari1/Credit-Card-Fraud-Detection>

cd credit-card-fraud-detection

1. **Install Dependencies**

bash

pip install -r requirements.txt

1. **Run the Script**

bash

[https://colab.research.google.com/drive/1Z8QMfujAYOIhBn9\_zxqVYEmBnk?usp=drive\_link](https://colab.research.google.com/drive/1Z8QMfujAYOIhBn9_zxqVYEvZja2umBnk?usp=drive_link)

**7. Best Practices & Recommendations**

* **Use GPU acceleration** for training XGBoost on large datasets.
* **Deploy as an API** using Flask/FastAPI for real-time fraud detection.
* **Monitor Model Performance** with A/B testing on unseen transactions.

**8. Conclusion**

* This project successfully detects fraudulent transactions using **Machine Learning models**.
* **XGBoost performs better** than Logistic Regression in fraud detection.
* Data preprocessing and class balancing significantly improve model performance.

**9. Appendix**

* **Dataset:** Kaggle Dataset
* **Libraries Used:** pandas, NumPy, scikit-learn, imbalanced-learn, matplotlib, XGboost