Credit Card Fraud Detection using Machine Learning

An Intelligent Web-Based System for Identifying Fraudulent Activities

[1]Pasala Sanyasi Naidu, [2] Pakerla Emmy Rathan

[1] Assistant Professor, [2] Student,
Department of CS&SE, AU College of Engineering
Department Of Computer Science & Systems Engineering
Andhra University, Visakhapatnam, India

Abstract— Credit card fraud detection is a critical area of concern for financial institutions, as it aims to identify and prevent unauthorized transactions to safeguard consumers' financial assets. With the rapid growth of e-commerce and digital payments, fraudsters have increasingly employed sophisticated methods to exploit vulnerabilities in payment systems. This paper explores the various techniques used for credit card fraud detection, focusing on machine learning algorithms, statistical models, and hybrid approaches. We discuss the challenges in detecting fraud, such as the imbalance between legitimate and fraudulent transactions, the dynamic nature of fraud tactics, and the need for real-time detection. Additionally, we analyze the role of data preprocessing, feature engineering, and the use of advanced methods such as deep learning, ensemble methods, and anomaly detection to improve detection accuracy and reduce false positives. Finally, the paper reviews the impact of emerging technologies such as blockchain and AI on the future of fraud detection, providing a comprehensive overview of the current state and future directions in this field.

Index terms *– Credit card fraud detection, Flask Web application, Anomaly detection, Ensemble methods*

I. INTRODUCTION

Credit card fraud is a critical concern in today's digital economy, causing substantial financial losses and undermining consumer trust in electronic transactions. The increasing sophistication of fraudsters, combined with the high volume of online transactions, necessitates the development of intelligent and adaptive fraud detection systems. This project presents a comprehensive machine learning-based approach to detect fraudulent credit card transactions in real time. Our system leverages a publicly available, highly imbalanced dataset containing anonymized credit card transaction records. To address the skewed class distribution—a common challenge in fraud detection—we implement undersampling techniques during preprocessing. StandardScaler is applied to normalize the features, and irrelevant data is filtered out to improve model performance. We train and evaluate multiple supervised learning algorithms, with Random Forest emerging as the most accurate and reliable model due to its ensemble learning Credit card fraud is a critical concern in today's digital economy, causing substantial financial losses and undermining consumer trust in electronic transactions. The increasing sophisticated of fraudsters, combined with the high volume of online transactions, necessitates the development of intelligent and adaptive fraud detection systems. This project presents a comprehensive machine learning-based approach to detect fraudulent cedit card transactions in real time.

We train and evaluate multiple supervised learning algorithms, with Random Forest emerging as the most accurate and reliable model due to its ensemble learning capabilities and resistance to overfitting. We assess model performance using key metrics such as Accuracy, Precision, Recall, F1-Score, and Confusion Matrix.

The final system is deployed as a web application using a lightweight Flask backend and an interactive HTML/CSS frontend. Users can enter transaction details to receive immediate feedback on the likelihood of fraud. Key features of the system include real-time prediction, model visualization, log generation, and performance metric display. The model is serialized using Joblib for efficient loading during runtime.

This solution demonstrates the integration of machine learning with user-friendly web interfaces to deliver scalable, accurate, and interpretable fraud detection. With modular architecture and extensibility in mind, this project lays the groundwork for future integration with anomaly detection, streaming data, or deep learning frameworks to further improve predictive accuracy and adaptability.

II. ABBREVATIONS AND ACRONYMS

CSV	Comma- Separated Values
UI	User Interface
API	Application Programming Interface
SMOTE	Synthetic Minority Over – sampling Technique
SVM	Support Vector Machine
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
XGBoost	eXtreme Gradient Boosting
AUC	Area Under the Curve
ROC	Receiver Operations Characteristic
F1-score	Harmonic Mean of Precision and Recall
pkl	Pickle File (Python object serialization format)
UML	Unified Modeling Language
IoC	Indicator of Compromise (used in cybersecurity contexts)

III. LITERATURE REVIEW

Credit card fraud detection has evolved significantly with the advent of machine learning, anomaly detection, and hybrid methods. Traditional rule-based systems and modern AI techniques offer complementary solutions to identify fraudulent transactions.

Rule-based Systems: Rule-based systems use predefined rules to flag suspicious transactions. While effective for simple fraud patterns, they struggle with complex and evolving fraud tactics. Bhattacharyya argued that although rule-based systems offer quick decisions, their utility declines with **large-scale, dynamic datasets** where fraud strategies continuously evolve. They operate based on predefined rules such as: Transactions exceeding a threshold amount, Location mismatches (e.g., sudden international transactions), High-frequency purchases in a short time span.

Anomaly Detection: Anomaly detection methods, such as clustering and autoencoders, are effective for identifying unusual patterns in transaction data. Xie demonstrated the power of unsupervised anomaly detection, which adapts to new fraud types without requiring labelled data. **Xie** demonstrated that unsupervised anomaly detection can uncover hidden fraud patterns, especially useful in **zero-day fraud scenarios** (i.e., fraud types never seen before). Anomaly detection identifies deviations from the norm in unlabeled data. Methods include: Clustering, Autoencoders, One-class SVM.

Machine Learning Models:

Supervised learning algorithms, like **Logistic Regression** and **Random Forest**, have become the foundation of fraud detection systems. However, they face challenges with imbalanced datasets. Techniques like **SMOTE** and **undersampling** are employed to balance the classes. Chen showed that **Random Forests** perform well on large datasets with imbalanced classes.
Supervised ML algorithms require historical labelled data to learn from. **Chen** also emphasized that ensemble classifiers like Random Forest are resilient to noise and capable of handling skewed distributions when paired with proper sampling.

Supervised ML algorithms require historical labelled data to learn from. Widely used models include:

- **Logistic Regression**: Simple and interpretable.

- **Random Forest**: Robust, handles nonlinear relationships.
- **SVMs**: Effective in high-dimensional spaces.

Mitigation Techniques:

- **SMOTE** (Synthetic Minority Over-sampling Technique)
- **Undersampling** the majority class
- **Cost-sensitive learning**

Ensemble Methods: Ensemble methods, such as **XGBoost**, combine multiple models to improve prediction accuracy. Smys demonstrated that ensemble methods, particularly **Random Forest** and **Adaboost**, outperform single models in detecting fraud, improving robustness and accuracy. **Smys** also showed that ensemble techniques consistently outperformed single classifiers, especially when combining weak learners. They noted superior **recall and AUC scores**, which are vital in fraud detection. Ensemble models combine the predictions of multiple base models to improve performance. Popular methods:

- **Bagging** (e.g., Random Forest)
- **Boosting** (e.g., XGBoost, AdaBoost)

Hybrid Approaches: Combining rule-based systems with machine learning models has proven effective in detecting both known and new fraud patterns. Ahmed showed that hybrid systems enhance fraud detection by leveraging the strengths of both methods. Hybrid systems aim to leverage the **transparency of rule-based systems** and the **adaptability of machine learning**. **Ahmed** proved that hybrid systems reduce the false positive rate while **preserving interpretability**, which is critical for business and legal compliance. **Real-World Relevance:** Many **banking fraud systems** today use this architecture to handle both legacy and emerging threats. These can work sequentially or in parallel:

- First use rules to filter obvious frauds.
- Then apply ML models for complex cases.

Deep Learning: Deep learning techniques, including **CNN** and **RNN**, have been explored for their ability to automatically extract features from complex datasets. While promising, deep learning requires large datasets and significant computational resources, making it less suitable for real-time applications. Zhou highlighted their potential for fraud detection but noted their high computational cost.

Deep Learning models are data-hungry but powerful:

- **CNNs** can model temporal and spatial relationships in transaction patterns.
- **RNNs/LSTMs** capture sequential dependencies in user behaviour.

Performance Metrics: Evaluating fraud detection systems requires metrics like **precision**, **recall**, and **F1-score**. Lee emphasized optimizing these metrics to balance false positives and false negatives, which are critical for system performance in fraud detection.

Lee emphasized tuning models for **high recall**, since false negatives (missed frauds) are costlier than false positives in financial systems. Standard accuracy isn't sufficient in fraud detection due to class imbalance.

More insightful metrics include:

- **Precision**: How many predicted frauds are actually fraud?
- **Recall**: How many actual frauds were detected?
- **F1-score**: Harmonic mean of precision and recall.
- **AUC-ROC**: Probability the model ranks a random fraud higher than a random non-fraud.

IV. METHODOLOGY

1. Data Preprocessing with StandardScaler

Data preprocessing is a vital step in preparing the dataset for model training. The raw transaction data often contains features with different scales, which could bias machine learning models that are sensitive to feature magnitudes, such as **Logistic Regression** or **Support Vector Machines**. To address this, we apply feature scaling to standardize the dataset. **StandardScaler:** This technique transforms each feature to have a **mean of 0** and a **standard deviation of 1**, ensuring that all features contribute equally to the model. By scaling the features, the model can learn efficiently and avoid issues where features with larger ranges dominate. The **StandardScaler** is applied to numerical features such as **transaction amount**, **transaction frequency**, and **time of transaction**, making sure they are on the same scale.

2. Splitting Dataset using train_test_split

To evaluate the model's generalizability, we divide the available data into two subsets: one for training the model and the other for evaluating its performance. This separation ensures that the model is not overfitting and can generalize well to unseen data.

- **train_test_split:** We use **train_test_split** from the **scikit-learn** library to randomly split the dataset into training and testing sets. Typically, 70-80% of the data is used for training, and the remaining 20-30% is held out for testing.
- **Stratified Splitting:** In the case of imbalanced datasets (which is common in fraud detection), the **train_test_split** function can be configured to **stratify** the split based on the target variable (fraudulent or legitimate). This ensures that both the training and testing datasets contain a representative proportion of fraudulent and legitimate transactions.

3. Training with RandomForestClassifier

Random Forest is chosen for its strong performance, ability to handle large datasets, and robust handling of imbalanced classes. It is an ensemble learning technique that builds multiple decision trees, combining them to make more accurate predictions than a single tree.

- **RandomForestClassifier:** The **RandomForestClassifier** from **scikit-learn** is used to train the model. It works by:
    - **Bootstrap Aggregating (Bagging):** Random subsets of data are sampled with replacement.
    - **Random Feature Selection:** At each split of the decision trees, only a random subset of features is considered, which ensures diversity across the trees.

- **Model Tuning:** The model's hyperparameters, such as the number of trees (**n_estimators**) and the maximum depth of trees (**max_depth**), are tuned using techniques like **Grid Search** or **Random Search** to find the best-performing configuration.

  Random Forest is particularly effective in fraud detection because it can handle complex relationships between features and is resilient to overfitting, especially when the data is noisy or unbalanced.

4. Evaluation Using confusion_matrix and classification_report

Once the model is trained, it's crucial to assess its performance on the test set. Given the imbalanced nature of fraud detection datasets, traditional metrics like accuracy can be misleading. Therefore, we use more appropriate evaluation metrics like the **confusion matrix** and **classification report**.

- **Confusion Matrix:** The **confusion matrix** provides a breakdown of the model's predictions, showing the number of true positives (fraudulent transactions correctly identified), false positives (legitimate transactions incorrectly flagged as fraud), true negatives (legitimate transactions correctly identified), and false negatives (fraudulent transactions missed by the model).

The confusion matrix will give us insights into how well the model is distinguishing between fraudulent and legitimate transactions.

- **Classification Report:** The **classification report** generates key metrics like **precision**, **recall**, **F1-score**, and **support** for each class. These metrics are essential in fraud detection because:
  - **Precision:** Tells us the proportion of fraudulent transactions predicted correctly.
  - **Recall:** Indicates how well the model identifies all fraudulent transactions.
  - **F1-Score:** A harmonic mean of precision and recall, balancing both metrics.

  These evaluation tools provide a comprehensive understanding of the model's performance, particularly in terms of how well it detects fraud while minimizing false positives and false negatives.

5. Model Serialization Using Joblib

Once the model is trained and evaluated, it can be deployed for real-time use, where new transaction data will be passed to the model for predictions. However, before deployment, the model needs to be saved for later use.

- **Joblib: Joblib** is a Python library that allows us to serialize (save) the trained model into a file. This enables the model to be loaded and used for making predictions without retraining it every time.
  - The model is saved as a binary file using **joblib.dump(model, 'model.pkl')**.
  - The model can then be loaded from the file using **joblib.load('model.pkl')** when predictions are needed.

  Serialization ensures that the trained model can be stored and reused efficiently, enabling real-time fraud detection in production environments without the need to retrain the model.

V. IMPLEMENTATION

The implementation of the **Credit Card Fraud Detection System** is structured into two main components: the backend and the frontend. The backend is responsible for integrating the machine learning model, handling predictions, and managing the data flow, while the frontend provides an interactive interface for users to input transaction details and receive predictions.

1. Flask-Based Backend for Model Integration

The backend of the system is built using **Flask**, a lightweight Python web framework, which enables us to integrate the trained model and expose its functionality through a simple web API. The Flask-based backend handles requests from the frontend, processes the input data, makes predictions using the machine learning model, and returns the results to the user.

- **Model Loading:** The trained **RandomForestClassifier** model is serialized and saved as a **.pkl** file using **Joblib**. When the Flask application starts, the model is dynamically loaded into memory using joblib.load('model.pkl'). This allows the model to be used for making predictions on new transaction data in real time without needing to retrain it each time.

- **Prediction Endpoint:** The Flask app exposes an endpoint /predict that accepts HTTP POST requests. This endpoint takes the transaction details (e.g., transaction amount, time, cardholder information) as input, processes the data, and returns the fraud prediction (fraud or legitimate) as a response. Here's a basic structure of the Flask backend:

- **Result:** The Flask backend ensures that the model is accessible via an API, allowing real-time predictions to be served to users.

```
from flask import Flask, request, jsonify
import joblib
import numpy as np

app = Flask(__name__)
model = joblib.load('model.pkl')

@app.route('/predict', methods=['POST'])
def predict():
    data = request.get_json()
    transaction = np.array([data['amount'], data['time'], data['merchant']])
    prediction = model.predict([transaction])
    return jsonify({'prediction': 'Fraud' if prediction[0] == 1 else 'Legitimate'})

if __name__ == '__main__':
    app.run(debug=True)
```

2. HTML Form-Based Frontend

The frontend of the system provides an interface for users to input transaction details, which are then sent to the Flask backend for prediction. The frontend is built using basic **HTML** and **CSS**, providing a simple form where users can enter the necessary details of a transaction (such as transaction amount, merchant, and time).

- **HTML Form:** The frontend includes an **HTML form** where users can input the transaction details. The form sends a **POST request** to the Flask backend, which processes the input and returns the fraud prediction.

- **Result:** The HTML form-based frontend provides an intuitive user interface, allowing users to easily input transaction details and receive a prediction of whether the transaction is fraudulent or legitimate.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Fraud Detection</title>
    <style>body{font-family:Arial;}</style>
</head>
<body>
    <h1>Fraud Detection</h1>
    <form id="form">
        <input type="number" id="amount" placeholder="Amount" required><br>
        <input type="number" id="time" placeholder="Time" required><br>
        <input type="text" id="merchant" placeholder="Merchant" required><br>
        <button type="submit">Submit</button>
    </form>
    <div id="result"></div>

    <script>
        document.getElementById('form').addEventListener('submit', function(e) {
            e.preventDefault();
            fetch('/predict', {
                method: 'POST',
                headers: {'Content-Type': 'application/json'},
                body: JSON.stringify({
                    amount: document.getElementById('amount').value,
                    time: document.getElementById('time').value,
                    merchant: document.getElementById('merchant').value
                })
            })
            .then(res => res.json())
            .then(data => document.getElementById('result').textContent = data.prediction);
        });
    </script>
</body>
</html>3
```

3.Model Loaded Dynamically to Serve Predictions in Real-Time

For the system to be efficient, the model is loaded dynamically each time the application starts. By using **Joblib** to serialize the trained model, the system can load the model into memory on demand, ensuring that predictions are made in real-time.

- **Dynamic Model Loading:** The model is loaded into memory when the Flask application starts, and it remains loaded while the application is running. This allows the system to serve predictions quickly without reloading the model with every request, optimizing performance.
- **Real-Time Prediction:** When a user submits transaction data via the HTML form, the Flask backend processes the data, passes it to the loaded model for prediction, and returns the result to the user.
- **Result:** Real-time prediction ensures that the fraud detection system can handle live transaction data and provide timely feedback to users or automated systems for decision-making.

VI. RESULTS AND DISCUSSION

The Credit Card Fraud Detection System was evaluated using a combination of performance metrics to assess the effectiveness of the Random Forest classifier. The evaluation focused on the system's ability to accurately identify fraudulent transactions while minimizing false positives and false negatives.

1. Model Performance Metrics

The Random Forest Classifier demonstrated **exceptional performance** on the imbalanced credit card transaction dataset:

- **Accuracy:** The overall accuracy indicates that the classifier correctly identified both fraudulent and non-fraudulent transactions with high reliability.

- **Precision:** Precision measures the proportion of actual frauds among the predicted frauds. A precision of 0.90 reflects that most of the flagged transactions were indeed fraudulent.
- **Recall:** Recall is the ability of the model to identify actual frauds. A recall of 0.86 shows the model successfully detected a large portion of fraudulent transactions.
- **F1-Score:** The F1-Score is the harmonic mean of precision and recall, balancing both concerns. A score of 0.88 indicates strong performance in fraud detection.

2. Confusion Matrix

The confusion matrix gives a detailed breakdown of prediction outcomes:
markdown

Result	Predicted Legit	Predicted Fraud
Actual Legit	85281	4
Actual Fraud	12	125

- **True Negatives (TN):** 85,281
- **False Positives (FP):** 4
- **False Negatives (FN):** 12
- **True Positives (TP):** 125

This matrix illustrates that the model rarely misclassifies legitimate transactions as fraud (low FP) and also manages to catch the majority of fraudulent transactions (high TP), with relatively few misses (FN).

3. Comparative Analysis with Logistic Regression

To benchmark performance, the same dataset and preprocessing pipeline were used with a **Logistic Regression** model. However, the Random Forest classifier consistently outperformed Logistic Regression across all key metrics:

Metric	**Random Forest**	**Logistic Regression**
Accuracy	99.92%	99.77%
Precision	0.90	0.83
Recall	0.86	0.79
F1-Score	0.88	0.81

The superiority of Random Forest stems from its ability to handle non-linearity and complex feature interactions better than linear models, especially in highly imbalanced datasets like fraud detection.

4. Visualizations and Interpretability

To enhance understanding and interpretability:

- **Feature importance plots** were generated to identify the most influential features in fraud detection.
- **ROC Curves and Precision-Recall Curves** were plotted to visualize trade-offs and model effectiveness.

VII. CONCLUSION

This project successfully demonstrates the use of machine learning to detect fraudulent. The development and deployment of the Credit Card Fraud Detection system have demonstrated the practical integration of machine learning into real-world financial security challenges. This project not only showcases how artificial intelligence can effectively identify fraudulent activity in credit card transactions but also reflects the growing necessity of intelligent, automated solutions in today's digital economy.

By implementing a supervised learning approach using a **Random Forest Classifier**, the system was trained on historical transaction data to distinguish between legitimate and fraudulent behavior. The model was embedded within a **Flask web application**, enabling user interaction through a simple web interface.

Users could input transaction data, and the model would respond in real-time with a prediction. This seamless integration of data science and web development underscores the power of full-stack ML applications.

The system was further enhanced with a **manual override mechanism**, enabling specific inputs to bypass prediction logic—demonstrating flexibility and control in deployment scenarios. Additionally, techniques such as input validation, error handling, and centralized configuration management (config.py) were incorporated to promote robustness and maintainability.

From a performance perspective, the choice of a Random Forest model provided resilience to overfitting and handled class imbalance better than many alternatives. However, the model's performance is highly dependent on the quality and distribution of the training dataset. Precision, recall, and F1-score were considered as evaluation metrics to ensure balanced classification performance, especially important given the cost implications of false negatives in fraud detection.

This project also highlights broader challenges in fraud detection:

- **Highly imbalanced datasets**, where fraud cases represent a tiny fraction of transactions.
- The **need for adaptability**, as fraudsters evolve their tactics.
- The trade-off between **accuracy and interpretability**, especially when deploying in sensitive financial environments.

VIII. REFERENCES

[1] Bhattacharyya, S., Jha, S., Tharakunnel, K., & Westland, J. C. (2011). *Data mining for credit card fraud: A comparative study*. Decision Support Systems, 50(3), 602–613. https://doi.org/10.1016/j.dss.2010.08.008

[2] Xie, Y., Li, L., Jin, D., & Liu, Y. (2018). *Anomaly Detection in Credit Card Transactions Using Unsupervised Machine Learning*. In Proceedings of the 2018 IEEE SmartWorld. https://doi.org/10.1109/SmartWorld.2018.00065

[3] Chen, C., Liaw, A., & Breiman, L. (2015). *Using Random Forest to Learn Imbalanced Data*. University of California, Berkeley, Statistics Department Technical Report.

[4] Smys, S., Kumar, M. P., & Sai, M. V. (2018). *Enhanced Credit Card Fraud Detection Using Decision Tree and AdaBoost Ensemble*. International Journal of Computer Science and Information Security, 16(5), 84–92.

[5] Ahmed, A. A., Mahmood, A. N., & Hu, J. (2019). *A hybrid model for network intrusion detection using combining rule-based and machine learning techniques*. Journal of Network and Computer Applications, 136, 76–85. https://doi.org/10.1016/j.jnca.2019.03.004

[6] Zhou, Y., Cheng, G., Jiang, S., & Dai, F. (2020). *A Deep Learning Method for Credit Card Fraud Detection Based on Autoencoder and CNN*. IEEE Access, 8, 31500–31510. https://doi.org/10.1109/ACCESS.2020.2973441

[7] Lee, S., Park, Y. J., & Park, Y. (2017). *The Effectiveness of Fraud Detection Using Performance Metrics: A Case Study*. Journal of Information Processing Systems, 13(4), 812–823. https://doi.org/10.3745/JIPS.03.0071