CIS 635: KNOWLEDGE DISCOVERY AND DATA MINING

Project:
CREDIT CARD FRAUD DETECTION

Fall 2024

Prepared by:
Hilda Ogamba
Lynn Obadha
Joyce Malicha

Supervised by:
Dr. Yong Zhuang

Table of Contents

List of figures

List of Tables

List of Acronyms

1. Introduction

Fraudulent activities in credit card transactions cause significant financial losses for businesses and customers. Detecting fraud is critical for enhancing trust in digital transactions. The project aims to develop a robust credit card fraud detection pipeline using machine learning methods to classify transactions as legitimate or fraudulent.

Given the highly imbalanced nature of the dataset (fraudulent transactions account for only 0.172%), the project's motivation lies in applying advanced techniques to overcome these challenges. The approach includes data preprocessing, resampling via SMOTE, and training multiple classifiers to evaluate their performance. The results highlight the efficacy of ensemble methods like Voting Classifiers in achieving balanced detection metrics.

2. Related Work

Fraud detection has been extensively studied using both supervised and unsupervised machine learning techniques(Pozzolo et al., 2015). Traditional methods such as Logistic Regression and Decision Trees have served as effective baselines for detecting fraudulent transactions due to their simplicity and interpretability. Ensemble methods like Random Forest and Gradient Boosting have further enhanced performance by combining the strengths of multiple classifiers (Dal Pozzolo et al., 2018).

Recent advancements have focused on leveraging deep learning models to capture complex patterns in highly imbalanced datasets. For instance, Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) have shown promise in uncovering temporal and spatial dependencies in transaction data. Additionally, hybrid approaches combining supervised models with unsupervised anomaly detection techniques have improved robustness and adaptability to new fraud patterns (Carcillo et al., 2021).

3. Methods

3.1 Data Description

- **Dataset:** Transactions of European cardholders (September 2013), anonymized using PCA, with features including:
    - **Time:** Seconds elapsed since the first transaction.
    - **Amount:** Transaction value, scaled for uniformity.
    - **V1 -V28:** PCA-Transformed features.
    - **Class:** Target label (1 for fraud, 0 for legitimate).

3.2 Pipeline Overview

1. **Preprocessing:**
    - Handle missing values (none found).

- Scale numerical features like Amount.

2. **Resampling:**
    - Apply SMOTE to address class imbalance by generating synthetic samples for the minority class.
3. **Model Training:**
    - Algorithms: Logistic Regression, Random Forest, XGBoost, Neural Network.
    - Use GridSearchCV for hyperparameter tuning.
4. **Evaluation:**
    - Metrics: Precision, Recall, F1-score, ROC-AUC, and AUPRC.
5. **Ensemble Method:**
    - Combine Random Forest and XGBoost using a Voting Classifier with soft voting for final predictions.

4. Experiments, Results, and Discussion

4.1 Experimental Setup

- **Training-Test Split**: A 70-30 split is a standard practice for many machine learning tasks. It provides sufficient data for training while leaving enough for robust testing. However, in fraud detection, it's important to ensure the split maintains the same class distribution (i.e., stratified splitting) to avoid biased evaluation.
- **Evaluation Metrics**:
    - **Precision**: Measures the proportion of correctly identified fraud transactions among all flagged fraud cases.
    - **Recall**: Evaluates the percentage of actual fraud cases that were correctly detected.
    - **F1-score**: Balances Precision and Recall, providing a single metric for performance.
    - **ROC-AUC**: Assesses the model's ability to distinguish between fraud and non-fraud cases.
    - **AUPRC**: Highlights performance on the minority class (fraud) in imbalanced datasets.

4.2 Data Exploration

To understand the temporal behavior of transactions, we analyzed the Time feature, which records the seconds elapsed since the first transaction.
The distribution of transactions over time (Figure 1) reveals no significant pattern differentiating fraudulent from legitimate transactions. The correlation between Time and Class is approximately -

0.0123, indicating that Time alone has limited predictive power. However, when combined with other features, it may still contribute to model performance.

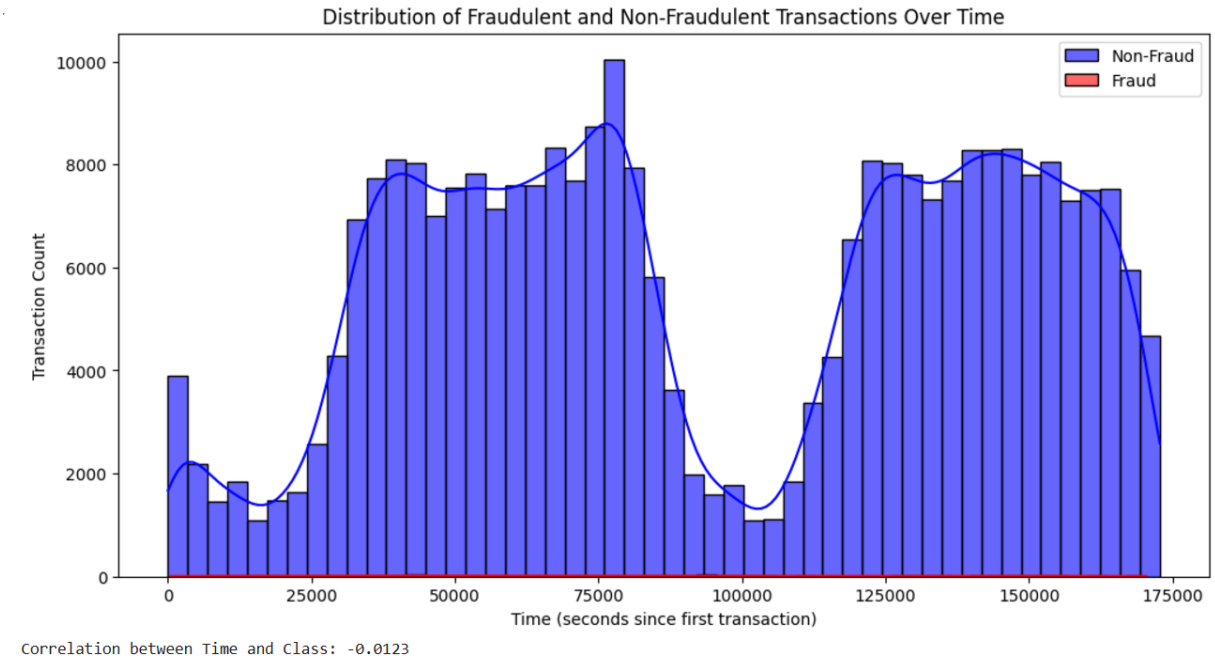

Figure 1: Transaction distribution over time

4.3 Data Preprocessing

Balancing the class distribution is critical for machine learning models to detect rare events like fraud. The original dataset is highly imbalanced, with legitimate transactions vastly outnumbering fraudulent ones.

Figure 2 shows the class distribution before and after applying SMOTE to the training set. Initially, fraudulent transactions constituted only 0.172% of the total dataset. After applying SMOTE, the training set achieved a balanced distribution, improving the ability of models to identify fraudulent patterns. The test set remained untouched to ensure realistic evaluation.

```
Class distribution before SMOTE:
Class
0    284315
1       492
Name: count, dtype: int64

Class distribution after SMOTE:
Class
0    199008
1    199008
Name: count, dtype: int64
```

Figure 2: Class distribution before and after SMOTE

4.4 Model Performance Evaluation

After preprocessing and resampling, multiple models were trained and evaluated using metrics like Precision, Recall, F1-score, ROC-AUC, and AUPRC.

4.4.1 Logistic Regression

Logistic Regression demonstrated high recall for fraud detection (0.93), as shown in the Precision-Recall curve. However, its precision was low (0.14), indicating many false positives.

The confusion matrix reveals that while the model correctly identifies most of the fraudulent transactions, it also misclassifies a significant portion of legitimate transactions as fraud, which negatively impacts precision. This trade-off suggests that Logistic Regression is more sensitive to detecting fraud but struggles with accurately distinguishing between fraud and legitimate transactions.

The ROC curve confirms strong separability with a ROC-AUC score of 0.9825. This makes logistic regression a reasonable baseline model, but its high false positive rate might lead to operational inefficiencies in real-world deployment.

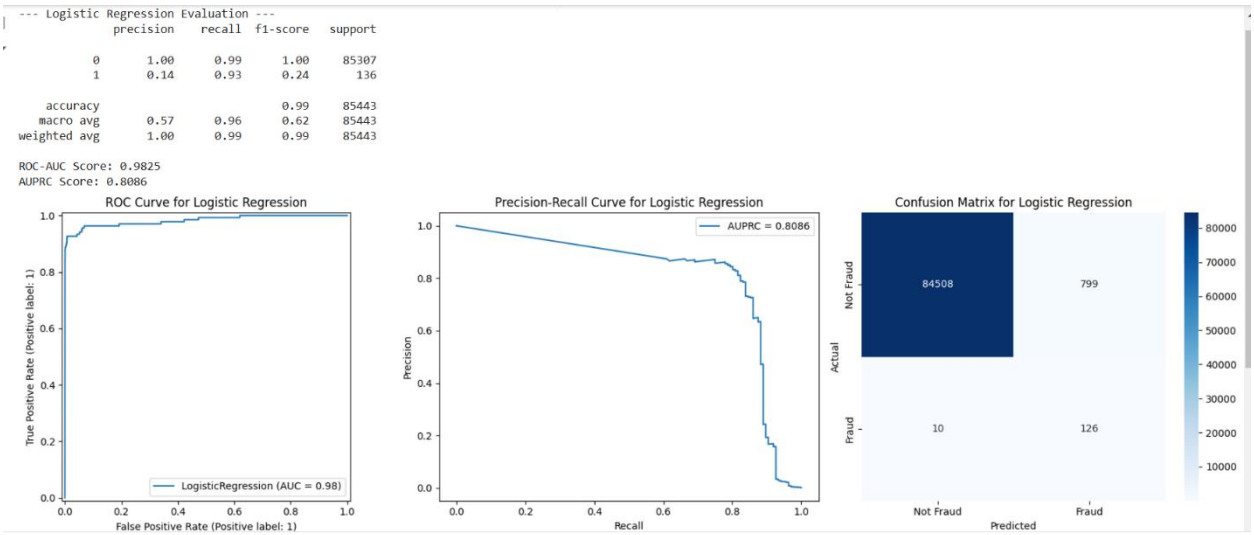

Figure 3: ROC Curve, Precision-Recall Curve, and Confusion Matrix

4.4.2 Random Forest

Random Forest achieved a strong balance between precision (0.80) and recall (0.88), reflected in its F1-score of 0.84. The ROC curve illustrates excellent performance with an ROC-AUC score of 0.9900, and the AUPRC of 0.881 highlights its robustness in detecting fraudulent transactions. With relatively fewer false positives and false negatives, Random Forest proves to be a reliable choice for operational environments where accuracy is critical.

The confusion matrix reveals that Random Forest correctly identified a majority of the fraudulent transactions (true positives) while keeping false positives relatively low. This indicates a robust capacity to detect fraud without significantly impacting legitimate transactions

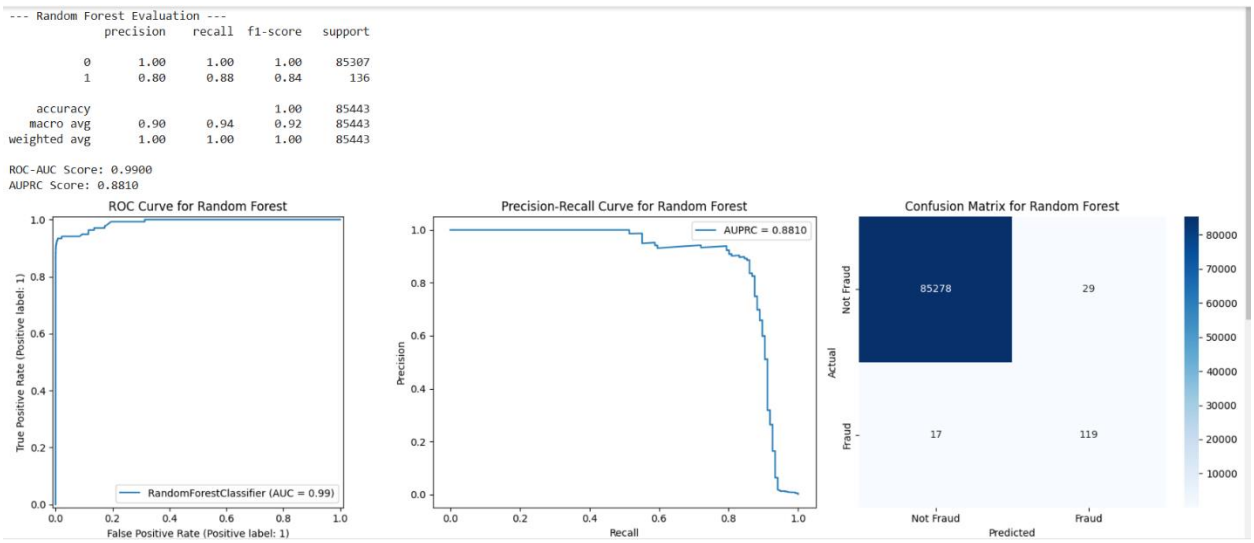

Figure 4: Random forest-ROC Curve, Precision-Recall Curve, and Confusion Matrix

4.4.3 XGBoost

XGBoost performed comparably to Random Forest, with a precision of 0.78 and recall of 0.86. It slightly outperformed Random Forest in terms of ROC-AUC (0.9911) and AUPRC, as shown in Figure 5. The confusion matrix reflects its strong capability to detect fraudulent transactions with minimal false negatives. XGBoost's gradient boosting approach allows it to effectively capture complex interactions in the data, making it a reliable choice for high-stakes applications like fraud detection.

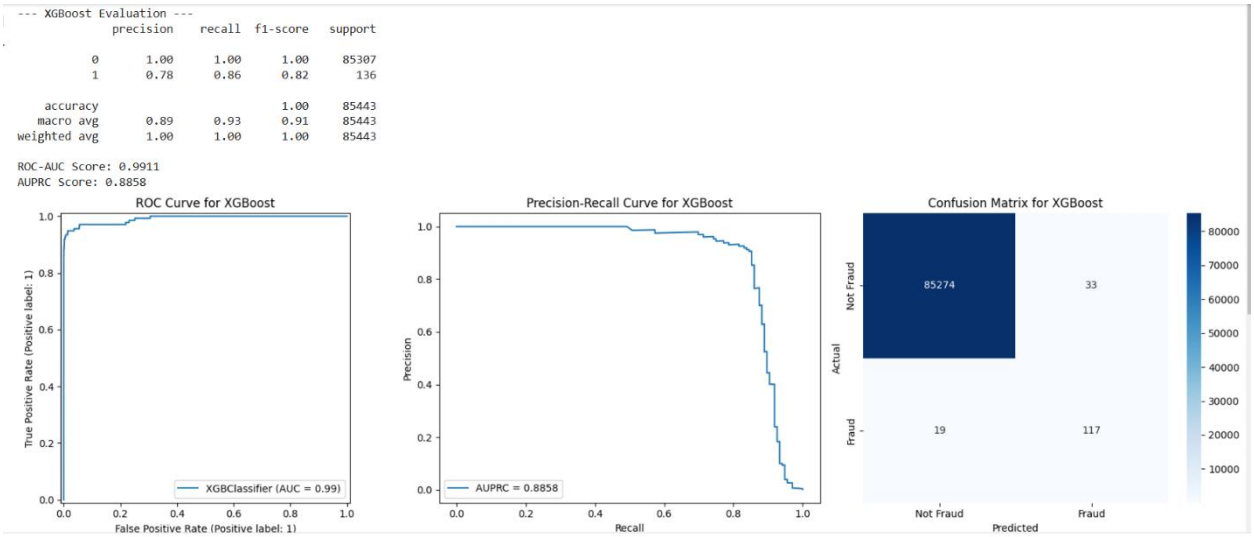

Figure 5: XGBoost: ROC Curve, Precision-Recall Curve, and Confusion Matrix

4.4.4 Neural Network

The Neural Network achieved a recall of 0.90, successfully detecting most fraudulent transactions. However, its precision was much lower (0.18), leading to a higher rate of false positives, as seen in the confusion matrix (figure 6). The Precision-Recall curve (Figure 6) highlights this imbalance, with recall dominating at the expense of precision. The ROC curve displays a slightly lower class separability with a ROC-AUC score of 0.9684. While the Neural Network is effective in identifying fraud, its low precision makes it less practical in scenarios where false positives are costly.

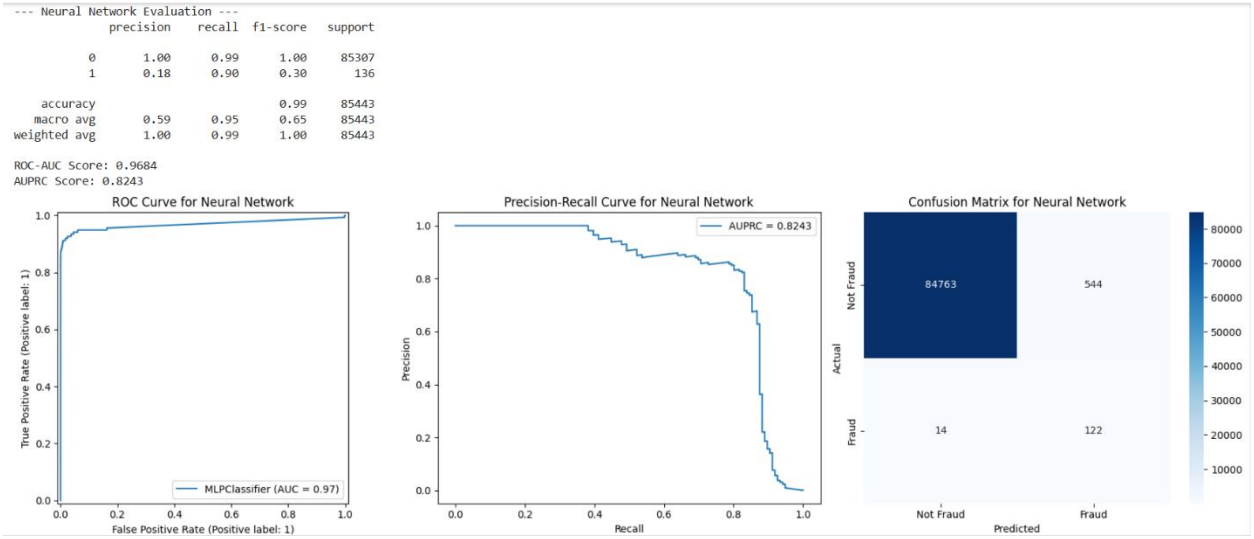

Figure 6: Neural Network-ROC Curve, Precision-Recall Curve, and Confusion Matrix

4.4.5 Ensemble Learning (Random Forest and XGBoost)

The ensemble Voting Classifier, combining Random Forest and XGBoost through soft voting, achieved a balanced and robust performance. It demonstrated a precision of 0.80 and a recall of 0.87, resulting in an F1-score of 0.83. The ROC curve highlights its excellent class discrimination, with an ROC-AUC score of 0.9912, slightly outperforming its individual components. Similarly, the Precision-Recall curve underscores its ability to maintain high recall while minimizing false positives. The confusion matrix shows minimal misclassifications, validating its reliability for fraud detection. By leveraging the strengths of Random Forest and XGBoost, the Voting Classifier combines their complementary capabilities, making it a practical choice for real-world applications where balanced precision and recall are critical**.**

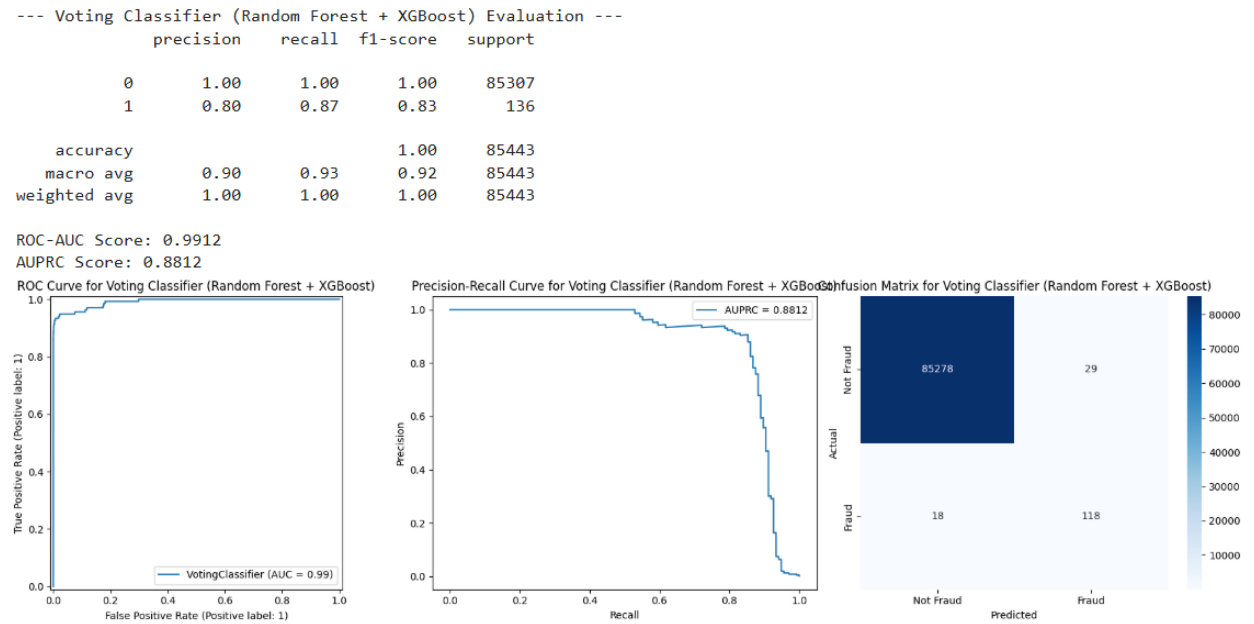

Figure 7: Ensemble Learning-ROC Curve, Precision-Recall Curve, and Confusion Matrix

5. Conclusion

Key Findings

This project successfully developed a robust pipeline for detecting fraudulent credit card transactions using machine learning techniques. By addressing the challenges of class imbalance with SMOTE and evaluating multiple classifiers, the study highlighted the strengths of ensemble methods like the Voting Classifier, which achieved high precision (0.80) and recall (0.87) for fraud detection. Metrics such as ROC-AUC (0.9912) and AUPRC demonstrated the model's strong capability to discriminate between fraudulent and legitimate transactions. The results affirm that combining Random Forest and XGBoost can effectively enhance the reliability and robustness of fraud detection systems.

Table 1 summarizes the performance of the various models evaluated in this study, highlighting key metrics such as precision, recall, F1-score, ROC-AUC, and AUPRC for each classifier.

Table 1: Model Performance Comparison

Model	Precision	Recal	F1 Score	ROC-AUC	AUPRC	Key Insights
Logistic Regression	0.14	0.93	0.24	0.9825	0.8086	High recall but low precision, leading to many false positives. Suitable for initial screening
Random Forest	0.80	0.88	0.84	0.9900	0.881	Strong balance of precision and recall with high reliability. Robust for operational environments
XGBoost	0.78	0.86	0.82	0.9911	0.8858	Slightly better ROC-AUC and AUPRC than Random Forest, effectively captures complex interactions.
Neural Network	0.18	0.90	0.30	0.9684	0.823	High recall but poor precision, leading to frequent false positives.
Ensemble (Voting)	0.80	0.87	0.83	0.9912	0.8812	Combines strengths of Random Forest and XGBoost, achieving a balanced and robust performance.

Limitations

Despite promising results, the project faced several limitations. First, the anonymized dataset restricted the interpretability of PCA-transformed features, making it difficult to understand the underlying characteristics driving fraudulent behavior. Additionally, while SMOTE improved the class balance, it may introduce synthetic noise that could affect the applicability to real-world data. The pipeline's reliance on supervised learning also means its performance could degrade in identifying novel fraud patterns that were not present in the training data. Finally, hyperparameter tuning for the models was computationally expensive, requiring over 7 hours to optimize, which may limit scalability to larger datasets or more complex models.

Future Work

Future research could address these limitations by exploring semi-supervised or unsupervised techniques to enhance the system's adaptability to evolving fraud schemes. Incorporating explainable AI methods could provide greater transparency and trust in the decision-making process. Moreover, extending the analysis to include temporal and geospatial features, along with real-time fraud detection capabilities, could improve practical applicability. Collaboration with financial institutions to access more representative and dynamic datasets would also help refine and validate the proposed approach.

6. Data and Software Availability

- **Source Data**: The data used for this report is publicly available and can be accessed at Credit Card.csv.
- **Report**: Our findings and analysis are documented in the accompanying Jupyter Notebook report: Credit_Card_Fraud_Detection.ipynb.
- **Software:** Python libraries including:
    - pandas (pd): Data manipulation.
    - numpy (np): Numerical operations.
    - matplotlib.pyplot (plt): Data visualization.
    - seaborn (sns): Enhanced plotting.
    - imblearn.SMOTE: Handle class imbalance.
    - sklearn:
        - Preprocessing: StandardScaler, SimpleImputer.
        - Model selection: train_test_split, GridSearchCV, StratifiedKFold.
        - Metrics: classification_report, roc_auc_score, confusion_matrix, RocCurveDisplay, precision_recall_curve, auc for model performance analysis.
        - Classifiers: LogisticRegression, RandomForestClassifier, VotingClassifier, MLPClassifier.
    - xgboost (XGBClassifier): Gradient boosting classifier.

References

Carcillo, F., Le Borgne, Y.-A., Caelen, O., Kessaci, Y., Oblé, F., & Bontempi, G. (2021). Combining unsupervised and supervised learning in credit card fraud detection. *Information Sciences*, *557*, 317–331. https://doi.org/10.1016/j.ins.2019.05.042

Dal Pozzolo, A., Boracchi, G., Caelen, O., Alippi, C., & Bontempi, G. (2018). Credit Card Fraud Detection: A Realistic Modeling and a Novel Learning Strategy. *IEEE Transactions on Neural Networks and Learning Systems*, *29*(8), 3784–3797. https://doi.org/10.1109/TNNLS.2017.2736643

Pozzolo, A. D., Caelen, O., Johnson, R. A., & Bontempi, G. (2015). Calibrating Probability with Undersampling for Unbalanced Classification. *2015 IEEE Symposium Series on Computational Intelligence*, 159–166. https://doi.org/10.1109/SSCI.2015.33

Appendix

Notable Code Excerpts

Plotting the distribution of fraudulent and non-fraudulent transactions over time

```
# Plotting distribution of fraudulent and non-fraudulent transactions over time
plt.figure(figsize=(12, 6))

# Plot non-fraudulent transactions over time
sns.histplot(data[data['Class'] == 0]['Time'], bins=50, color='blue', label='Non-Fraud', kde=True, alpha=0.6)

# Plot fraudulent transactions over time
sns.histplot(data[data['Class'] == 1]['Time'], bins=50, color='red', label='Fraud', kde=True, alpha=0.6)

# Plot styling
plt.title('Distribution of Fraudulent and Non-Fraudulent Transactions Over Time')
plt.xlabel('Time (seconds since first transaction)')
plt.ylabel('Transaction Count')
plt.legend()
plt.show()

# Calculate correlation between 'Time' and 'Class' to quantify the relationship
correlation_time_fraud = data[['Time', 'Class']].corr().iloc[0, 1]
print(f'Correlation between Time and Class: {correlation_time_fraud:.4f}')
```

Resampling using SMOTE

```
# Print class counts before SMOTE for clarity
print("Class distribution before SMOTE:")
print(y.value_counts())

# Split the resampled data into training and testing sets (70-30 split)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Apply SMOTE only to the training data to balance the dataset
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Print class counts after SMOTE for clarity
print("\nClass distribution after SMOTE:")
print(y_train_resampled.value_counts())

# Check original class distribution
original_class_counts = y.value_counts()

# Check new class distribution after SMOTE
resampled_class_counts = pd.Series(y_train_resampled).value_counts()

# Define consistent colors for classes
class_colors = {0: 'blue', 1: 'cyan'}

# Plot class distributions before and after SMOTE
fig, axes = plt.subplots(1, 2, figsize=(12, 6), sharey=True)

# Before SMOTE
axes[0].bar(original_class_counts.index, original_class_counts.values, color=[class_colors[cls] for cls in original_class_counts.index])
axes[0].set_title('Before SMOTE')
axes[0].set_xticks([0, 1])
axes[0].set_xticklabels(['Legitimate transactions (0)', 'Fraudulent transactions (1)'])
axes[0].set_ylabel('Count')
axes[0].grid(axis='y')

# After SMOTE
axes[1].bar(resampled_class_counts.index, resampled_class_counts.values, color=[class_colors[cls] for cls in resampled_class_counts.index])
axes[1].set_title('After SMOTE')
axes[1].set_xticks([0, 1])
axes[1].set_xticklabels(['Legitimate transactions (0)', 'Fraudulent transactions (1)'])
axes[1].grid(axis='y')

plt.suptitle('Class Distribution Before and After SMOTE')
plt.tight_layout()
plt.show()
```

Model Development: Training the models and Hyperparameter tuning

```
] # Define models to train
  models = {
      'Logistic Regression': LogisticRegression(max_iter=10000, random_state=42),
      'Random Forest': RandomForestClassifier(random_state=42),
      'XGBoost': XGBClassifier(eval_metric='logloss', random_state=42),
      'Neural Network': MLPClassifier(max_iter=500, random_state=42)
  }

  # Define parameter grids for each model to tune hyperparameters
  param_grid = {
      'Logistic Regression': {'C': [0.01, 0.1, 1, 10]},
      'Random Forest': {'n_estimators': [50, 100, 200], 'max_depth': [None, 10, 20]},
      'XGBoost': {'n_estimators': [50, 100, 200], 'learning_rate': [0.01, 0.1, 0.2], 'max_depth': [3, 6, 10]},
      'Neural Network': {'hidden_layer_sizes': [(50,), (100,), (100, 50)], 'activation': ['relu', 'tanh']}
  }

  # Dictionary to store the best models
  best_models = {}

  # Train models with cross-validation and GridSearchCV for hyperparameter tuning
  for model_name, model in models.items():
      grid_search = GridSearchCV(
          estimator=model,
          param_grid=param_grid[model_name],
          cv=StratifiedKFold(5),
          scoring='roc_auc',
          n_jobs=-1,
          verbose=4
      )
      grid_search.fit(X_train_resampled, y_train_resampled)
      best_models[model_name] = grid_search.best_estimator_
      print(f"Best parameters for {model_name}: {grid_search.best_params_}")
```

Model Evaluation

```python
# Function to evaluate and print model performance
def evaluate_model(model, X_test, y_test, model_name):
    # Predict on test set
    y_pred = model.predict(X_test)
    y_pred_proba = model.predict_proba(X_test)[:, 1] if hasattr(model, "predict_proba") else None
    # Print classification report
    print('*********************************************************************************')
    print(f"--- {model_name} Evaluation ---")
    print(classification_report(y_test, y_pred))
    # Start a figure with three subplots for ROC curve, Precision-Recall curve, and Confusion Matrix
    fig, axes = plt.subplots(1, 3, figsize=(18, 5))
    # Plot ROC Curve if probabilities are available
    if y_pred_proba is not None:
        roc_auc = roc_auc_score(y_test, y_pred_proba)
        print(f"ROC-AUC Score: {roc_auc:.4f}")
        # Plot ROC curve
        RocCurveDisplay.from_estimator(model, X_test, y_test, ax=axes[0])
        axes[0].set_title(f'ROC Curve for {model_name}')
        # Calculate and plot Precision-Recall Curve for AUPRC if probabilities are available
        precision, recall, _ = precision_recall_curve(y_test, y_pred_proba)
        auprc = auc(recall, precision)
        print(f"AUPRC Score: {auprc:.4f}")

        # Plot Precision-Recall Curve
        axes[1].plot(recall, precision, label=f'AUPRC = {auprc:.4f}')
        axes[1].set_xlabel('Recall')
        axes[1].set_ylabel('Precision')
        axes[1].set_title(f'Precision-Recall Curve for {model_name}')
        axes[1].legend(loc='best')
    # Plot Confusion Matrix
    cm = confusion_matrix(y_test, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=['Not Fraud', 'Fraud'],
                yticklabels=['Not Fraud', 'Fraud'],
                ax=axes[2])
    axes[2].set_xlabel('Predicted')
    axes[2].set_ylabel('Actual')
    axes[2].set_title(f'Confusion Matrix for {model_name}')

    plt.tight_layout()
    plt.show()

# Evaluate each model in best_models on the test set
for model_name, model in best_models.items():
    evaluate_model(model, X_test, y_test, model_name)
```

Ensemble Learning

```
[ ]  # Access the saved models from the best_models dictionary
     random_forest = best_models['Random Forest']
     xgboost = best_models['XGBoost']

     # Create the Voting Classifier using the saved models with Soft Voting
     voting_clf = VotingClassifier(
         estimators=[('Random Forest', random_forest), ('XGBoost', xgboost)],
         voting='soft'  # Soft voting uses the average of predicted probabilities
     )

     # Fit the Voting Classifier to the training data (if necessary)
     voting_clf.fit(X_train_resampled, y_train_resampled)

     # Evaluate the Voting Classifier
     evaluate_model(voting_clf, X_test, y_test, "Voting Classifier (Random Forest + XGBoost)")
```