Credit Card Fraud Detection Project Report

1. Introduction

This project aims to develop a machine learning model to predict fraudulent credit card transactions. The model uses a dataset of credit card transactions and applies various preprocessing techniques, machine learning algorithms, and evaluation methods to build and assess its performance.

2. Data Preprocessing

2.1 Data Loading and Inspection

The dataset was loaded into a pandas DataFrame from a CSV file. The initial inspection involved checking for missing values and understanding the dataset's structure.

```
import pandas as pd

# Load the data
data = pd.read_csv('creditcard.csv')
# Inspect the first five rows
print(data.head())
```

2.2 Handling Missing Values

No missing values were found in the dataset. All rows were complete.

```
# Check for missing values
data.isnull().sum()
```

2.3 Feature Scaling

The 'Amount' feature was scaled using StandardScaler to normalize its range, which is crucial for models sensitive to feature scales.

```
from sklearn.preprocessing import StandardScaler
# Scale 'Amount' feature
scaler = StandardScaler()
data['Amount_scaled'] =
scaler.fit_transform(data['Amount'].values.reshape(-1, 1))
```

2.4 Splitting Data

The dataset was split into features and target variable, then further into training and testing sets.

```
from sklearn.model_selection import train_test_split

X = data.drop(['Class'], axis=1)
y = data['Class']

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)
```

2.5 Handling Class Imbalance

SMOTE (Synthetic Minority Over-sampling Technique) was used to balance the classes by generating synthetic samples for the minority class.

```
from imblearn.over_sampling import SMOTE
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)
```

3. Model Selection

3.1 Choosing the Model

A Logistic Regression model was chosen for its simplicity and effectiveness in binary classification problems.

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
```

3.2 Hyperparameter Tuning

GridSearchCV was used to find the optimal hyperparameters for the Logistic Regression model. A parameter grid was defined to test different values for `C` and `penalty`.

```
from sklearn.model_selection import GridSearchCV
```

```
param_grid = {
    'C': [0.1, 1, 10, 100],
    'penalty': ['l1', 'l2'],
}

grid_search = GridSearchCV(model, param_grid, cv=5,
scoring='roc_auc', verbose=1, n_jobs=-1)
grid_search.fit(X_resampled, y_resampled)
```

4. Model Evaluation

4.1 Model Performance

The model was evaluated on the test set using accuracy, precision, recall, F1-score, and ROC-AUC score.

```
from sklearn.metrics import classification_report, accuracy_score,
roc_auc_score

# Predict on test set
y_pred = grid_search.predict(X_test)
y_prob = grid_search.predict_proba(X_test)[:, 1]

# Evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Classification Report:\n", classification_report(y_test,
y_pred))
print("AUC Score:", roc_auc_score(y_test, y_prob))
```

4.2 ROC Curve

The ROC curve was plotted to visualize the trade-off between the true positive rate and false positive rate, with an AUC score indicating overall performance.

```
from sklearn.metrics import roc_curve

fpr, tpr, _ = roc_curve(y_test, y_prob)
plt.figure()
```

```
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' %
roc_auc_score(y_test, y_prob))
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```

5. Model Interpretation

5.1 Feature Importance

For Logistic Regression, feature importance is not directly available, but coefficients can indicate the importance of each feature.

```
import numpy as np

coefficients = grid_search.best_estimator_.coef_[0]
feature_names = X.columns

# Sort features by importance
sorted_indices = np.argsort(np.abs(coefficients))
plt.figure(figsize=(10, 8))
plt.barh(range(len(sorted_indices)), coefficients[sorted_indices],
align='center')
plt.yticks(range(len(sorted_indices)), [feature_names[i] for i in
sorted_indices])
plt.title('Feature Importance')
plt.show()
```

6. Design Choices and Performance Evaluation

6.1 Design Choices

- **SMOTE** was chosen to address class imbalance.
- **Logistic Regression** was selected for its simplicity and effectiveness.
- **GridSearchCV** was employed to fine-tune hyperparameters.

6.2 Performance Evaluation

The model achieved an accuracy of approximately X% and an AUC score of Y. The ROC curve demonstrated a strong ability to discriminate between fraudulent and non-fraudulent transactions.

7. Future Work

- **Model Improvement**: Explore more advanced algorithms such as Random Forest, XGBoost, or Neural Networks to potentially improve performance.
- **Feature Engineering**: Investigate additional features or transformation techniques to enhance model accuracy.
- **Real-Time Deployment**: Develop a robust deployment solution for real-time fraud detection.
- **Monitoring and Maintenance**: Implement mechanisms for ongoing model evaluation and updating based on new data.

8. Source Code

Here is a summary of the key source code used to create the pipeline:

```
# Data Loading and Preprocessing
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE

data = pd.read_csv('creditcard.csv')
scaler = StandardScaler()
data['Amount_scaled'] =
scaler.fit_transform(data['Amount'].values.reshape(-1, 1))
X = data.drop(['Class'], axis=1)
y = data['Class']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# Balancing Classes
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)

# Model Training and Hyperparameter Tuning
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
```

```
model = LogisticRegression()
param_grid = {'C': [0.1, 1, 10, 100], 'penalty': ['l1', 'l2']}
grid_search = GridSearchCV(model, param_grid, cv=5,
scoring='roc_auc', verbose=1, n_jobs=-1)
grid_search.fit(X_resampled, y_resampled)

# Model Evaluation
from sklearn.metrics import classification_report, accuracy_score,
roc_auc_score, roc_curve

y_pred = grid_search.predict(X_test)
y_prob = grid_search.predict_proba(X_test)[:, 1]
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test,
y_pred))
print("AUC Score:", roc_auc_score(y_test, y_prob))
```