BOOSTING FOR IMBALANCED DATASETS WITH XGBOOST – CREDIT CARD FRAUD DETECTION

Submitted by: PRIYANKA B

Roll Number: 727723EUAI093

Class: III Year, AI&DS - B

PROJECT DOCUMENTATION

TITLE:

Boosting for Imbalanced Datasets with XGBoost – Credit Card Fraud Detection

ABSTRACT:

In many real-world machine learning applications, datasets are highly imbalanced, where the number of samples belonging to one class significantly exceeds the other. Credit card fraud detection is a classic example, where fraudulent transactions form only a small fraction of total transactions. Traditional machine learning models often fail to detect such rare events effectively.

This project focuses on applying XGBoost (Extreme Gradient Boosting) to handle imbalanced datasets efficiently. By incorporating boosting techniques, class weighting, and SMOTE oversampling, the proposed model improves fraud detection performance. The effectiveness of the approach is evaluated using ROC-AUC and Precision–Recall curves, which are more suitable for imbalanced datasets than accuracy.

PROBLEM STATEMENT:

Handling imbalanced datasets is a major challenge in machine learning. Standard classifiers such as Support Vector Machines (SVM) and Random Forests tend to be biased toward the majority class, leading to poor detection of minority class samples.

The Credit Card Fraud Detection dataset is extremely imbalanced, with fraudulent transactions representing less than 0.2% of the total data. The problem is to build a robust classification model that can accurately identify fraudulent transactions while minimizing false negatives.

OBJECTIVES:

- To apply XGBoost for classification on an imbalanced dataset
- To handle class imbalance using:
  - Class weighting (scale_pos_weight)
  - SMOTE oversampling technique
- To tune XGBoost hyperparameters for improved performance
- To evaluate the model using metrics suitable for imbalanced data:
  - Precision
  - Recall
  - F1-score
  - ROC-AUC
  - Precision–Recall Curve

DATASET DESCRIPTION:

Dataset Name: Credit Card Fraud Detection

Source: Kaggle

Dataset Details

- Total Transactions: 284,807
- Fraudulent Transactions: 492
- Legitimate Transactions: 284,315

Attributes

- V1 to V28: Anonymized numerical features obtained using PCA
- Time: Time elapsed between transactions
- Amount: Transaction amount
- Class: Target variable
  - 0 → Legitimate transaction
  - 1 → Fraudulent transaction

Nature of Dataset

The dataset is highly imbalanced, making it ideal for testing boosting-based models and imbalance-handling techniques.

SYSTEM REQUIREMENTS:

Hardware Requirements

- Processor: Intel i3 or higher
- RAM: Minimum 4 GB
- Storage: 2 GB free space

Software Requirements

- Python 3.x
- Google Colab (cloud-based execution)
- Libraries:
  - pandas
  - numpy
  - scikit-learn
  - xgboost

- imbalanced-learn
- matplotlib

METHODOLOGY:

Step 1: Data Loading

The dataset is loaded into the environment and separated into features and target variables.

Step 2: Data Preprocessing

- Standardization is applied to Time and Amount
- Stratified train-test split is performed to preserve class distribution

Step 3: Handling Class Imbalance

- SMOTE is applied to generate synthetic minority class samples
- Class weighting is used in XGBoost to penalize misclassification of fraud cases

Step 4: Model Training

- XGBoost classifier is trained using boosting techniques to focus on misclassified samples**.**

Step 5: Hyperparameter Tuning

- GridSearchCV is used to optimize parameters such as learning rate, maximum depth, and number of estimators.

Step 6: Model Evaluation

- Performance is evaluated using ROC-AUC score and Precision–Recall curve.

ALGORITHM USED:

XGBoost (Extreme Gradient Boosting)

XGBoost is an ensemble learning algorithm based on gradient boosting. It builds multiple decision trees sequentially, where each new tree corrects the errors of the previous ones. XGBoost supports:

- Regularization
- Parallel processing
- Handling of imbalanced data using class weights

EVALUATION METRICS:

Because accuracy is misleading for imbalanced datasets, the following metrics are used:

- Precision: Measures correctness of predicted frauds
- Recall: Measures ability to detect actual frauds
- F1-score: Balance between precision and recall
- ROC-AUC: Measures overall discrimination ability
- Precision–Recall Curve: Highlights minority class performance

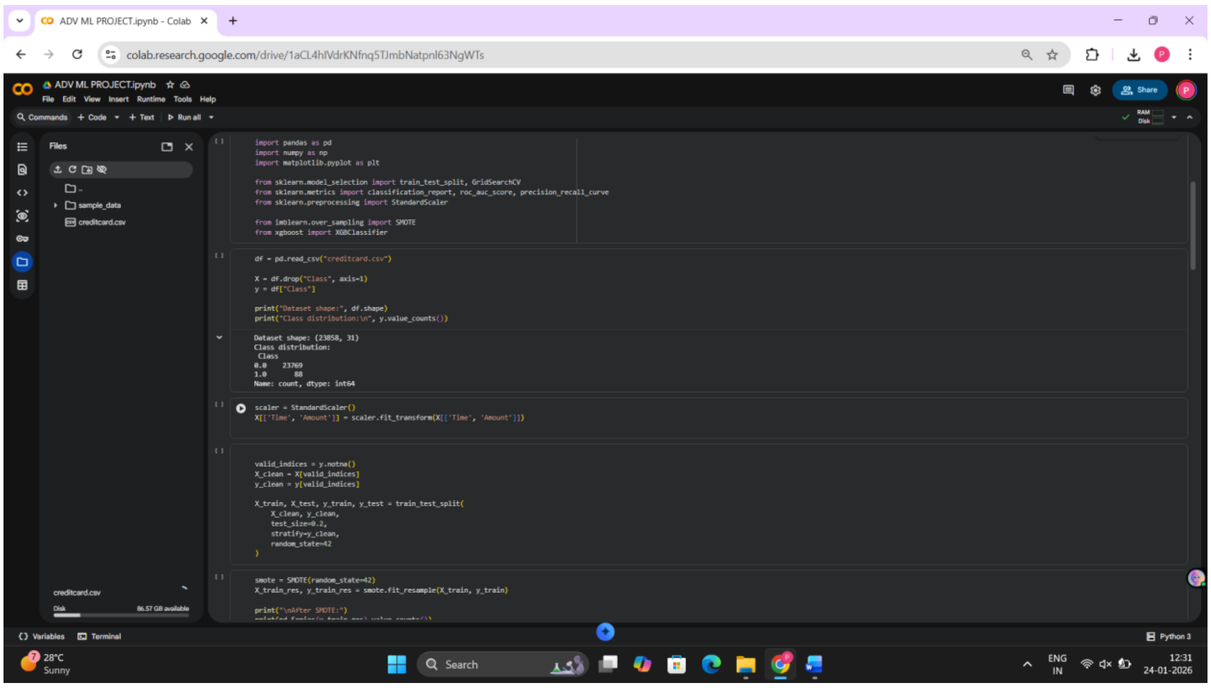
RESULTS AND DISCUSSION:

- XGBoost significantly improves fraud detection recall
- SMOTE helps the model learn minority class patterns
- ROC-AUC score indicates strong classification performance
- Precision–Recall curve shows better detection of rare fraud cases compared to traditional classifiers

OUTCOMES:

- Successful handling of extreme class imbalance
- Improved minority class detection
- Robust and scalable fraud detection model
- Industry-relevant machine learning pipeline

SCREENSHOTS:


ADV ML PROJECT.ipynb - Colab
colab.research.google.com/drive/1aCL4hIVdrKNfnq5TJmbNatpnl63NgWTs
ADV ML PROJECT.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
Files
sample_data
creditcard.csv
Variables Terminal
Python 3

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import classification_report, roc_auc_score, precision_recall_curve
from sklearn.preprocessing import StandardScaler

from imblearn.over_sampling import SMOTE
from xgboost import XGBClassifier
```

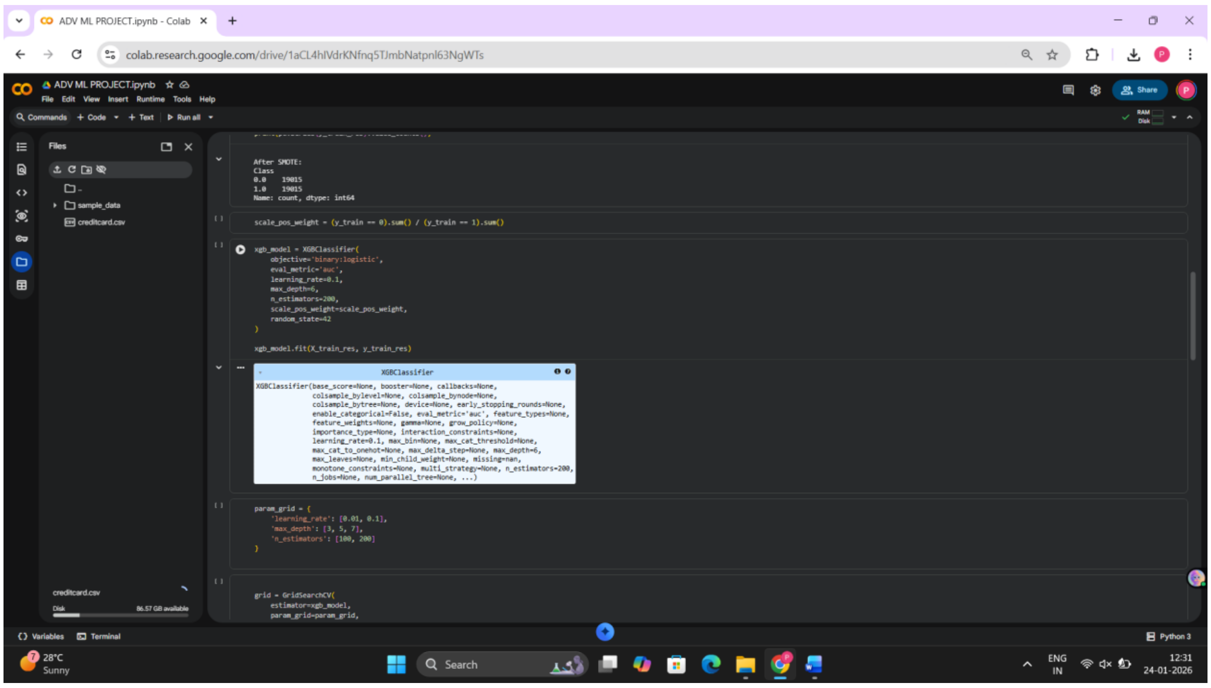
```
df = pd.read_csv("creditcard.csv")

X = df.drop("Class", axis=1)
y = df["Class"]

print("Dataset shape:", df.shape)
print("Class distribution:\n", y.value_counts())
```

```
Dataset shape: (23858, 31)
Class distribution:
 Class
0.0    23769
1.0       88
Name: count, dtype: int64
```

```
scaler = StandardScaler()
X[['Time', 'Amount']] = scaler.fit_transform(X[['Time', 'Amount']])
```

```
valid_indices = y.notna()
X_clean = X[valid_indices]
y_clean = y[valid_indices]

X_train, X_test, y_train, y_test = train_test_split(
    X_clean, y_clean,
    test_size=0.2,
    stratify=y_clean,
    random_state=42
)
```

```
smote = SMOTE(random_state=42)
X_train_res, y_train_res = smote.fit_resample(X_train, y_train)

print("\nAfter SMOTE:")
```



```
After SMOTE:
Class
0.0    19015
1.0    19015
Name: count, dtype: int64
```

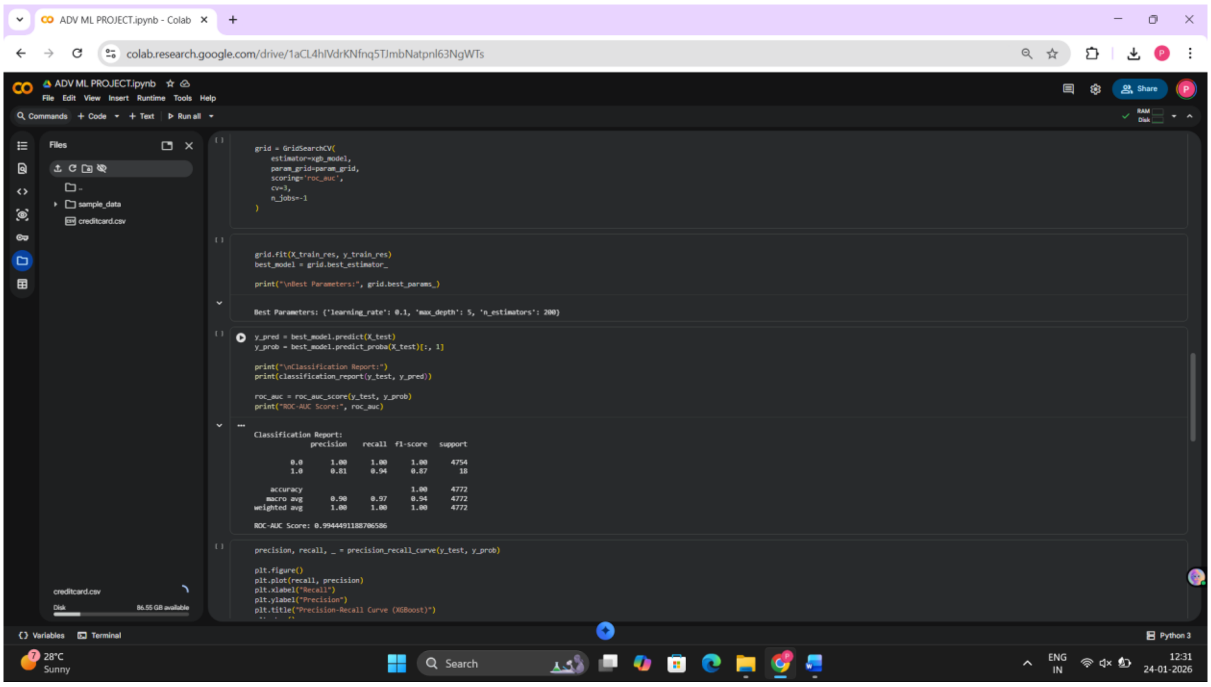
```
scale_pos_weight = (y_train == 0).sum() / (y_train == 1).sum()
```

```
xgb_model = XGBClassifier(
    objective='binary:logistic',
    eval_metric='auc',
    learning_rate=0.1,
    max_depth=6,
    n_estimators=200,
    scale_pos_weight=scale_pos_weight,
    random_state=42
)

xgb_model.fit(X_train_res, y_train_res)
```

```
XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='auc', feature_types=None,
              feature_weights=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=0.1, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=6,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, multi_strategy=None, n_estimators=200,
              n_jobs=None, num_parallel_tree=None, ...)
```

```
param_grid = {
    'learning_rate': [0.01, 0.1],
    'max_depth': [3, 5, 7],
    'n_estimators': [100, 200]
}
```

```
grid = GridSearchCV(
    estimator=xgb_model,
    param_grid=param_grid,
```



```
grid = GridSearchCV(
    estimator=xgb_model,
    param_grid=param_grid,
    scoring='roc_auc',
    cv=3,
    n_jobs=-1
)
```

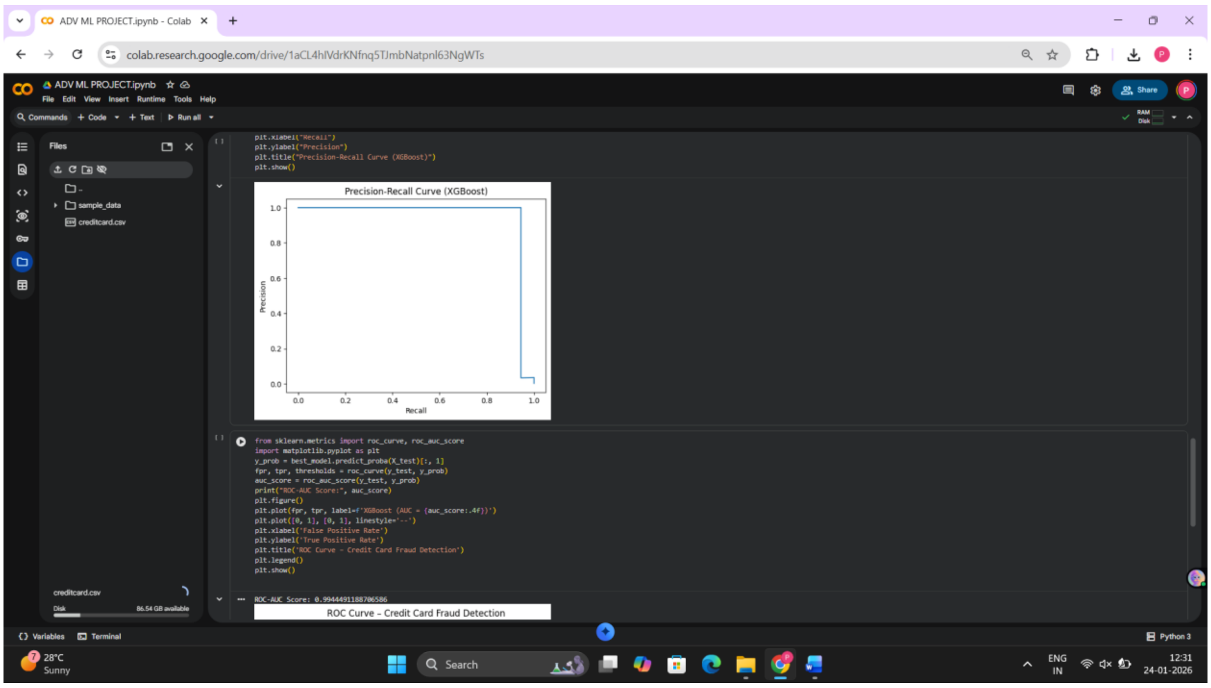
```
grid.fit(X_train_res, y_train_res)
best_model = grid.best_estimator_

print("\nBest Parameters:", grid.best_params_)
```

```
Best Parameters: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 200}
```

```
y_pred = best_model.predict(X_test)
y_prob = best_model.predict_proba(X_test)[:, 1]

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

roc_auc = roc_auc_score(y_test, y_prob)
print("ROC-AUC Score:", roc_auc)
```

```
Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      4754
         1.0       0.81      0.94      0.87        18

    accuracy                           1.00      4772
   macro avg       0.90      0.97      0.94      4772
weighted avg       1.00      1.00      1.00      4772

ROC-AUC Score: 0.9944491188706586
```

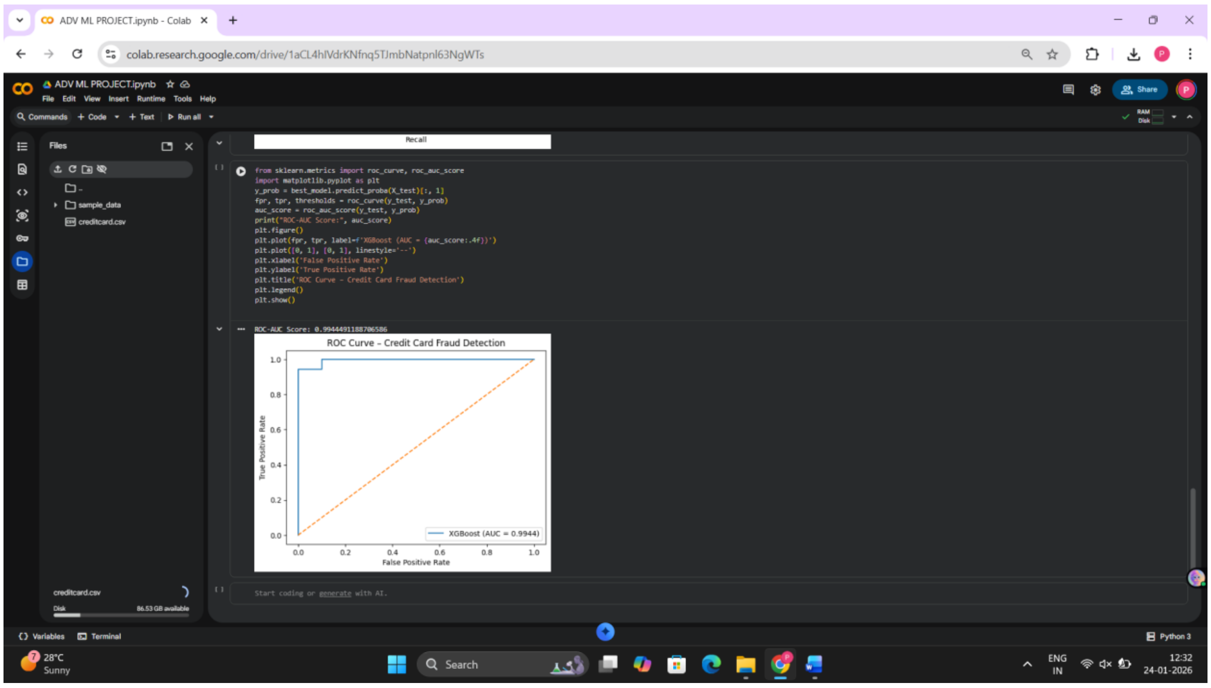
```
precision, recall, _ = precision_recall_curve(y_test, y_prob)

plt.figure()
plt.plot(recall, precision)
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve (XGBoost)")
```



```python
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve (XGBoost)")
plt.show()
```

```python
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt
y_prob = best_model.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
auc_score = roc_auc_score(y_test, y_prob)
print("ROC-AUC Score:", auc_score)
plt.figure()
plt.plot(fpr, tpr, label=f'XGBoost (AUC = {auc_score:.4f})')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Credit Card Fraud Detection')
plt.legend()
plt.show()
```

```
ROC-AUC Score: 0.9944491188706586
```



```python
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt
y_prob = best_model.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
auc_score = roc_auc_score(y_test, y_prob)
print("ROC-AUC Score:", auc_score)
plt.figure()
plt.plot(fpr, tpr, label=f'XGBoost (AUC = {auc_score:.4f})')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Credit Card Fraud Detection')
plt.legend()
plt.show()
```

```
ROC-AUC Score: 0.9944491188706586
```

CONCLUSION:

This project demonstrates that XGBoost is an effective solution for imbalanced classification problems such as credit card fraud detection. By combining boosting, class weighting, and SMOTE, the model achieves improved performance on rare event detection. The approach can be extended to other real-world domains involving imbalanced data.

FUTURE ENHANCEMENTS:

- Compare performance with Logistic Regression and Random Forest
- Implement threshold tuning to reduce false negatives
- Deploy the model using Flask or FastAPI
- Apply real-time fraud detection techniques

REFERENCES:

1. Kaggle – Credit Card Fraud Detection Dataset
2. XGBoost Official Documentation
3. Scikit-learn Documentation
4. Imbalanced-learn Documentation