Boosting for Imbalanced Datasets with XGBoost

Problem Statement

Imbalanced datasets are a common challenge in real-world machine learning applications such as fraud detection, medical diagnosis, and anomaly detection. In such datasets, one class significantly outnumbers the other, causing traditional machine learning algorithms like Support Vector Machines (SVM) and Random Forest to be biased toward the majority class. As a result, minority class predictions—often the most important—are inaccurate.

Boosting techniques aim to improve model performance by combining multiple weak learners into a strong learner. XGBoost (Extreme Gradient Boosting) is an advanced boosting algorithm that efficiently handles large datasets and complex patterns. It provides built-in mechanisms such as weighted loss functions and regularization, making it highly effective for imbalanced classification problems.

This project focuses on applying XGBoost to handle imbalanced datasets, improving classification performance by emphasizing minority class learning through class balancing techniques and hyperparameter optimization.

Objectives

The main objectives of this project are:

1. To implement an XGBoost classifier for imbalanced classification tasks.
2. To address class imbalance using techniques such as class weighting and SMOTE.
3. To tune XGBoost hyperparameters (learning rate, max depth, number of estimators) for improved performance.
4. To evaluate model performance using metrics suitable for imbalanced datasets such as Precision, Recall, F1-Score, ROC-AUC, and Precision-Recall curves.

Model Implementation :

XGBoost Classification:

XGBoost is an ensemble learning algorithm based on gradient boosting that builds decision trees sequentially, where each new tree corrects the errors made by previous trees. It optimizes a loss function using gradient descent and includes regularization to prevent overfitting.

Steps involved:

- Load the imbalanced dataset
- Split data into training and testing sets
- Apply class imbalance handling techniques
- Train an XGBoost classifier
- Evaluate model performance using appropriate metrics

Handling Class Imbalance:

Class Weighting:

Class weighting assigns higher importance to the minority class by penalizing misclassification errors more heavily. In XGBoost, this is achieved using the scale_pos_weight parameter, which helps the model focus more on minority class samples.

SMOTE (Synthetic Minority Oversampling Technique):

SMOTE is used to generate synthetic samples for the minority class instead of duplicating existing samples. This balances the dataset and helps the model learn better decision boundaries.

Steps involved:

- Apply SMOTE on the training dataset
- Ensure class balance before model training
- Train XGBoost on the resampled data

Hyperparameter Tuning:

Hyperparameter tuning plays a crucial role in improving model performance. The following parameters are optimized:

- **Learning Rate:** Controls the contribution of each tree
- **Max Depth:** Limits tree complexity
- **Number of Estimators:** Number of boosting rounds
- **Subsample and Colsample:** Prevent overfitting

Grid Search or Random Search techniques are used to find the optimal combination of parameters.

Performance Evaluation:

For imbalanced datasets, accuracy alone is not a reliable metric. Therefore, the following evaluation metrics are used:

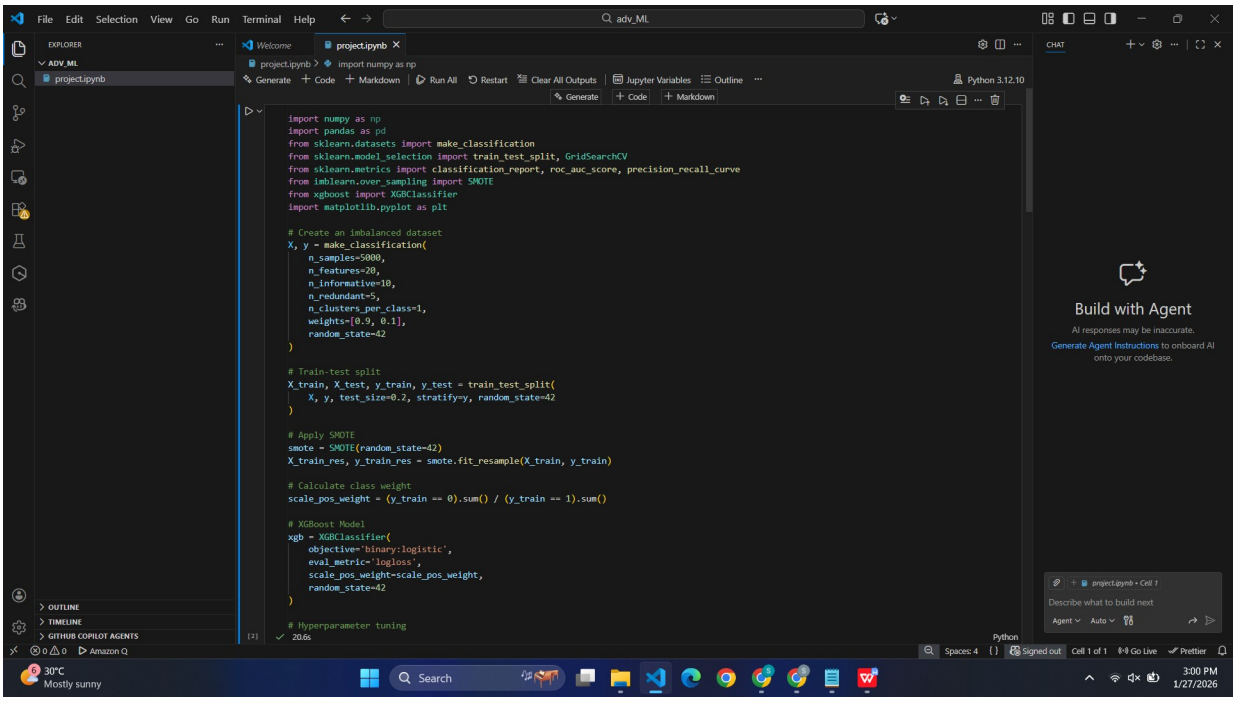
- **Precision:** Measures correctness of positive predictions
- **Recall:** Measures the ability to detect minority class samples
- **F1-Score:** Balances precision and recall
- **ROC-AUC Curve:** Evaluates class separation capability
- **Precision-Recall Curve:** Provides better insight for imbalanced datasets

These metrics demonstrate the effectiveness of XGBoost in improving minority class prediction.

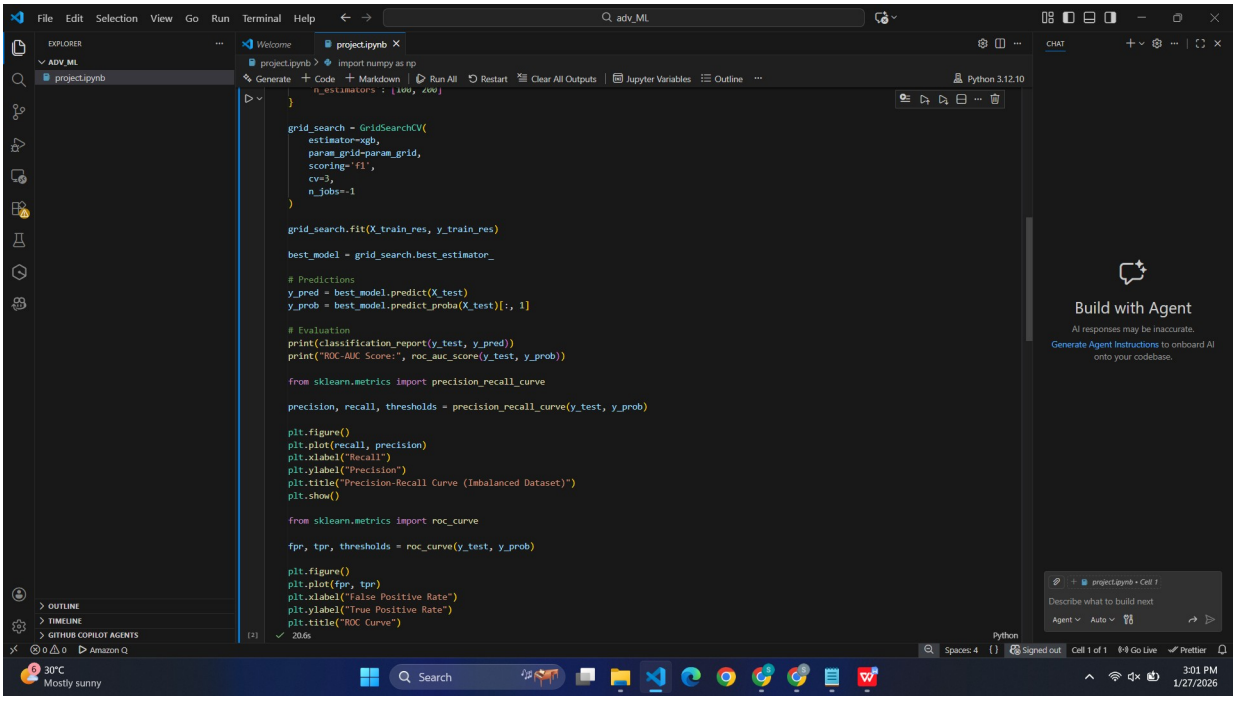
Results and Outcomes:

- Improved classification performance on imbalanced datasets
- Better recall and F1-score for the minority class
- Demonstrated superiority of XGBoost over traditional classifiers
- Clear visualization of model performance using ROC and Precision-Recall curves

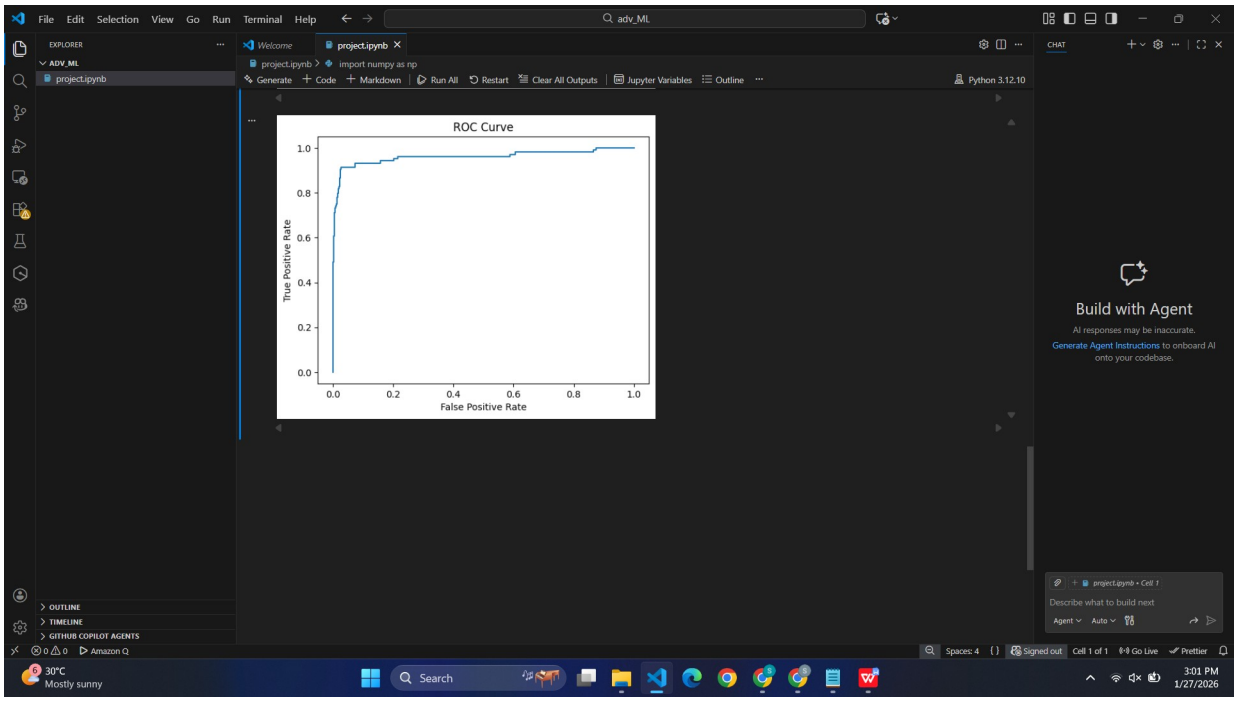
Output

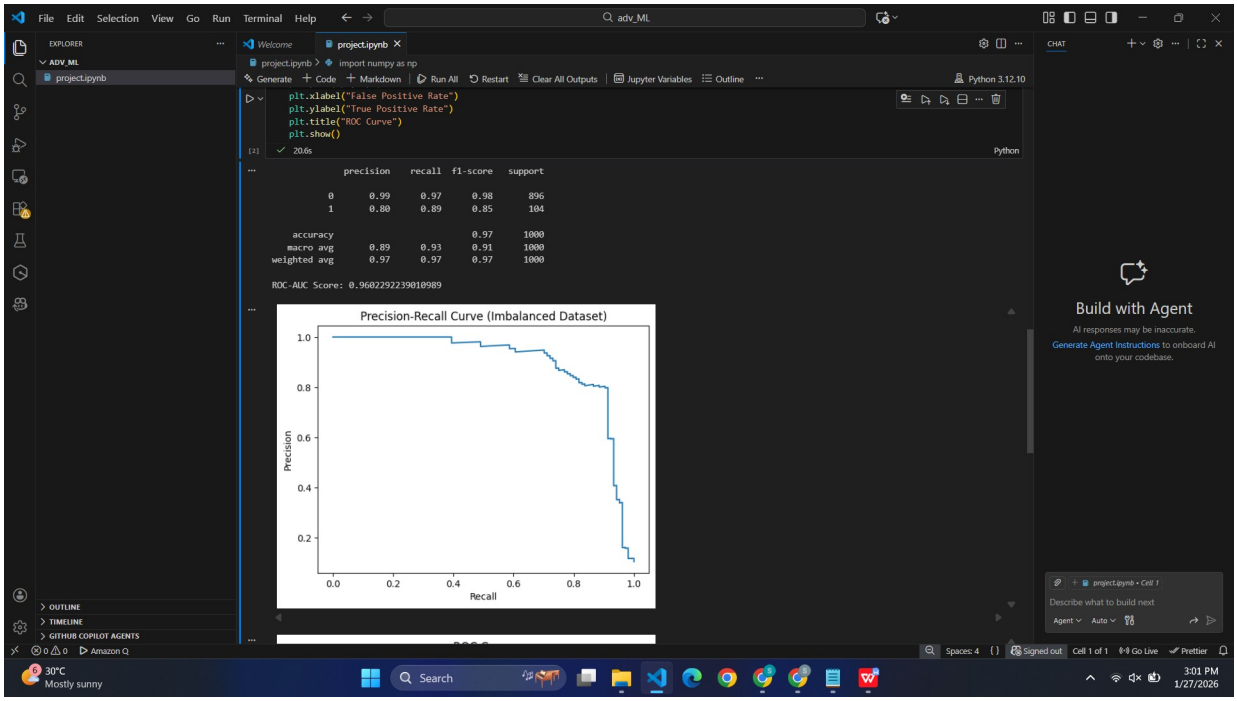


```
import numpy as np
import pandas as pd
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import classification_report, roc_auc_score, precision_recall_curve
from imblearn.over_sampling import SMOTE
from xgboost import XGBClassifier
import matplotlib.pyplot as plt

# Create an imbalanced dataset
X, y = make_classification(
    n_samples=5000,
    n_features=20,
    n_informative=10,
    n_redundant=5,
    n_clusters_per_class=1,
    weights=[0.9, 0.1],
    random_state=42
)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)

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

# Calculate class weight
scale_pos_weight = (y_train == 0).sum() / (y_train == 1).sum()

# XGBoost Model
xgb = XGBClassifier(
    objective='binary:logistic',
    eval_metric='logloss',
    scale_pos_weight=scale_pos_weight,
    random_state=42
)

# Hyperparameter tuning
```



```
    n_estimators : [100, 200]
}

grid_search = GridSearchCV(
    estimator=xgb,
    param_grid=param_grid,
    scoring='f1',
    cv=3,
    n_jobs=-1
)

grid_search.fit(X_train_res, y_train_res)

best_model = grid_search.best_estimator_

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

# Evaluation
print(classification_report(y_test, y_pred))
print("ROC-AUC Score:", roc_auc_score(y_test, y_prob))

from sklearn.metrics import precision_recall_curve

precision, recall, thresholds = precision_recall_curve(y_test, y_prob)

plt.figure()
plt.plot(recall, precision)
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve (Imbalanced Dataset)")
plt.show()

from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y_test, y_prob)

plt.figure()
plt.plot(fpr, tpr)
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
```


ROC Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Build with Agent
AI responses may be inaccurate.
Generate Agent Instructions to onboard AI onto your codebase.


plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.show()
precision recall f1-score support
0 0.99 0.97 0.98 896
1 0.80 0.89 0.85 104
accuracy 0.97 1000
macro avg 0.89 0.93 0.91 1000
weighted avg 0.97 0.97 0.97 1000
ROC-AUC Score: 0.960229223010989
Precision-Recall Curve (Imbalanced Dataset)
Precision
Recall
Build with Agent

Conclusion:

This project demonstrates that XGBoost is a powerful and effective boosting technique for handling imbalanced datasets. By combining class imbalance handling methods, hyperparameter tuning, and appropriate evaluation metrics, the model achieves significantly improved classification performance. The results highlight XGBoost's suitability for real-world imbalanced classification problems.