<u>ONLINE PAYMENTS FRAUD DETECTION USING WITH MACHINE LEARNING</u>:

To build an application that can detect the legitimacy of the transaction in real-time and increase the security to prevent fraud.

By

(Marri yashmitha)

(Manthina raja rishika)

(Kutagulla safa)

Guided by

Prof. Ms swetha raj

A Dissertation Submitted to SRI VENKATESWARA COLLEGE OF ENGINEERING AND TECHNOLOGY, An Autonomous Institution affiliated to 'JNTU Ananthapur' in Partial Fulfilment of the Bachelor of Technology branch of ***Computer science and Engineering***

May 2024

SRI VENKATESWARA COLLEGE OF ENGINEERING AND TECHNOLOGY

R.V.S. Nagar Tirupathi Road, Andhra Pradesh– 517127

Model optimization and turning phase report for online fraud detection using machine learning

Introduction:

This report outlines the optimization and tuning processes applied to a machine learning model developed for online fraud detection. The goal is to enhance the model's performance, ensuring high accuracy and robustness in identifying fraudulent transactions.

Data Overview

The dataset used for this task includes transaction records with various features such as transaction amount, time, user details, and transaction location. The target variable indicates whether a transaction is fraudulent.

Preprocessing

Data Cleaning:

-Missing values handled by imputation.

-Removal of duplicate records.

Feature Engineering:

-Creation of new features like transaction frequency per user.

-Encoding categorical variables using techniques like One-Hot encoding

Normalization/Scaling:

-Applied Min-Max scaling to ensure all features are on a similar scale

Model Selection:
Several machine learning algorithms were initially considered:

Logistic Regression
Decision Trees
Random Forest
Gradient Boosting
XGBoost
Neural Networks

Baseline Model Performance

A baseline model using Logistic Regression was trained and evaluated:

Accuracy: 0.92
Precision: 0.88
Recall: 0.75
F1-Score: 0.81

Model Optimization and Tuning

1. Random Forest
Hyperparameters Tuned:
Number of trees (n_estimators)
Maximum depth of the trees (max_depth)
Minimum samples required to split a node (min_samples_split)
Minimum samples required at each leaf node (min_samples_leaf)
Grid Search Results:

python
Copy code
```
from sklearn.model_selection import GridSearchCV
```

```
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

rf = RandomForestClassifier()
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5,
scoring='f1')
grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_
best_rf_model = grid_search.best_estimator_
```

Best Hyperparameters:

n_estimators: 200
max_depth: 20
min_samples_split: 5
min_samples_leaf: 2
Performance:

Accuracy: 0.94
Precision: 0.91
Recall: 0.80
F1-Score: 0.85
2. XGBoost
Hyperparameters Tuned:
Learning rate (eta)

Maximum depth (max_depth)
Number of boosting rounds (n_estimators)
Subsample ratio (subsample)
Column sample by tree (colsample_bytree)
Grid Search Results:

python
Copy code
```
import xgboost as xgb

param_grid = {
    'eta': [0.01, 0.1, 0.2],
    'max_depth': [6, 10, 15],
    'n_estimators': [100, 200, 300],
    'subsample': [0.8, 0.9, 1.0],
    'colsample_bytree': [0.8, 0.9, 1.0]
}

xgb_model = xgb.XGBClassifier()
grid_search = GridSearchCV(estimator=xgb_model, param_grid=param_grid,
cv=5, scoring='f1')
grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_
best_xgb_model = grid_search.best_estimator_
```
Best Hyperparameters:

eta: 0.1
max_depth: 10

n_estimators: 200
subsample: 0.9
colsample_bytree: 0.9
Performance:

Accuracy: 0.96
Precision: 0.93
Recall: 0.85
F1-Score: 0.89
Ensemble Approach
To leverage the strengths of both Random Forest and XGBoost, an ensemble model using stacking was created.

Stacking Results:

python
Copy code
```
from sklearn.ensemble import StackingClassifier

estimators = [
    ('rf', best_rf_model),
    ('xgb', best_xgb_model)
]

stacking_model = StackingClassifier(estimators=estimators,
final_estimator=LogisticRegression())
stacking_model.fit(X_train, y_train)
```
Performance:
Accuracy: 0.97

Precision: 0.94
Recall: 0.88
F1-Score: 0.91

Conclusion

The model optimization and tuning phase significantly improved the online fraud detection model's performance. The final stacked ensemble model achieved the highest F1-Score, ensuring a balanced performance in precision and recall, which is critical for fraud detection tasks. Continuous monitoring and periodic retraining with new data will be essential to maintain and potentially improve model performance over time.