**INSAID**

**Internship Task (Data Science & Machine Learning)**

**­­Name: Siddhant Saraf**

**1. Data cleaning including missing values, outliers, and multi-collinearity.**

import pandas as pd

import numpy as np

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import StandardScaler

from sklearn.ensemble import IsolationForest

from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

# Load the dataset

data = pd.read\_csv("financial\_transactions.csv")

# Handling missing values

# Assuming missing values are represented as NaN

# For numeric columns, impute missing values with median

numeric\_cols = ['amount', 'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest', 'newbalanceDest']

imputer = SimpleImputer(strategy='median')

data[numeric\_cols] = imputer.fit\_transform(data[numeric\_cols])

# Handling outliers using Isolation Forest

outlier\_detector = IsolationForest(contamination=0.05)  # Adjust contamination based on dataset

data['outlier'] = outlier\_detector.fit\_predict(data[numeric\_cols])

# Remove outliers

data = data[data['outlier'] == 1]

# Drop the outlier flag column

data.drop(columns=['outlier'], inplace=True)

# Multicollinearity check using VIF

def calculate\_vif(X):

    vif\_data = pd.DataFrame()

    vif\_data["Feature"] = X.columns

    vif\_data["VIF"] = [variance\_inflation\_factor(X.values, i) for i in range(len(X.columns))]

    return vif\_data

# Separate X and y

X = data.drop(['isFraud', 'isFlaggedFraud', 'nameOrig', 'nameDest'], axis=1)

y = data['isFraud']

# Calculate VIF

vif = calculate\_vif(X)

print(vif)

**2. Describe your fraud detection model in elaboration.**

Our fraud detection model leverages a Random Forest classifier due to its robustness, accuracy, and ability to handle class imbalance effectively. We began by preprocessing the data, which included handling missing values, outliers, and multicollinearity. The dataset, which consists of transactional data with features like transaction type, amount, and account balances, was first split into training and testing sets. To address the issue of class imbalance, we employed the Synthetic Minority Over-sampling Technique (SMOTE) to ensure the model could accurately learn to detect fraudulent transactions.

The Random Forest classifier was then trained on the resampled training data. This model is particularly suitable for our use case as it constructs multiple decision trees during training and outputs the mode of the classes (classification) of the individual trees, reducing the risk of overfitting and increasing generalizability.

We evaluated the model's performance using various metrics, including accuracy, precision, recall, F1-score, and the Area Under the Receiver Operating Characteristic Curve (AUC-ROC). These metrics provided a comprehensive understanding of the model's ability to correctly identify fraudulent transactions while minimizing false positives.

**3. How did you select variables to be included in the model?**

I will use feature importance from the Random Forest model and correlation analysis to select relevant variables. Features contributing significantly to the prediction of fraud will be included.

**4. Demonstrate the performance of the model by using best set of tools.**

I will use metrics such as accuracy, precision, recall, F1-score, and AUC-ROC to evaluate model performance.

from sklearn.model\_selection import train\_test\_split

from imblearn.over\_sampling import SMOTE

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification\_report, roc\_auc\_score, confusion\_matrix, roc\_curve

# Split the data

X = data.drop(['isFraud', 'isFlaggedFraud', 'nameOrig', 'nameDest'], axis=1)

y = data['isFraud']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Handle class imbalance using SMOTE

smote = SMOTE(random\_state=42)

X\_train\_res, y\_train\_res = smote.fit\_resample(X\_train, y\_train)

# Train Random Forest classifier

model = RandomForestClassifier(random\_state=42)

model.fit(X\_train\_res, y\_train\_res)

# Predictions

y\_pred = model.predict(X\_test)

y\_prob = model.predict\_proba(X\_test)[:, 1]

# Performance metrics

print(classification\_report(y\_test, y\_pred))

print(f"AUC-ROC: {roc\_auc\_score(y\_test, y\_prob)}")

# Confusion Matrix

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues')

plt.title('Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.show()

# ROC Curve

fpr, tpr, thresholds = roc\_curve(y\_test, y\_prob)

plt.figure(figsize=(8, 6))

plt.plot(fpr, tpr, label=f'AUC-ROC: {roc\_auc\_score(y\_test, y\_prob):.2f}')

plt.plot([0, 1], [0, 1], 'k--')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve')

plt.legend()

plt.show()

**5. What are the key factors that predict fraudulent customer?**

Key factors are identified using feature importance from the Random Forest model:

# Feature importance

feature\_importance = model.feature\_importances\_

features = X.columns

important\_features = pd.DataFrame({'Feature': features, 'Importance': feature\_importance})

important\_features.sort\_values(by='Importance', ascending=False, inplace=True)

important\_features.head(10)

**6. Do these factors make sense? If yes, How? If not, How not?**

The key factors identified by the Random Forest model for predicting fraudulent transactions generally make sense when we consider the nature of financial fraud. These factors include transaction amount, the type of transaction, and the balances before and after transactions for both the origin and destination accounts.

Firstly, *transaction amount* is a critical factor. Fraudulent transactions often involve unusually large amounts compared to regular transactions. Fraudsters aim to maximize their gains before detection, leading to a spike in the transaction amount, making it a significant indicator of potential fraud.

Secondly, *type of transaction* plays a crucial role. Certain types of transactions, such as TRANSFER and CASH\_OUT, are more prone to fraud. Transfers and cash withdrawals are common methods for fraudsters to move and extract money quickly from compromised accounts. In contrast, transaction types like PAYMENT might be less associated with fraud, as they often involve paying bills or regular expenses.

Thirdly, *balances before and after the transaction* (both for the origin and destination accounts) provide vital clues. Significant changes in these balances can indicate fraudulent activity. For instance, if a large amount is withdrawn, resulting in a sudden drop in the account balance (newbalanceOrig), it may signal that the account is being drained. Similarly, if the destination account sees a sudden large increase in its balance (newbalanceDest), it could be receiving fraudulently transferred funds. In addition, discrepancies between the expected new balance and the actual new balance after a transaction can be red flags for fraud detection.

**7. What kind of prevention should be adopted while company update its infrastructure?**

When updating its infrastructure, the company should adopt several preventive measures to enhance security and minimize the risk of fraud. First, implementing a robust real-time monitoring system is crucial. This system should leverage advanced machine learning models to detect suspicious transactions as they occur, allowing for immediate intervention. Multi-factor authentication (MFA) should be enforced, especially for high-value transactions, adding an extra layer of security by requiring multiple forms of verification from users. The company should also set up transaction limits to flag or halt unusually large transactions for further review.

**8. Assuming these actions have been implemented, how would you determine if they work?**

To determine if the implemented actions are effective in combating fraudulent activities, we would need to conduct thorough monitoring, analysis, and evaluation. Firstly, we would continuously monitor the performance of our fraud detection system in real-time. This includes tracking metrics such as the number of fraudulent transactions detected, false positive rates, and the overall accuracy of the system. Additionally, we would collect feedback from users and stakeholders to gauge their satisfaction with the new security measures and any observed improvements in their confidence in the system's ability to detect fraud. Furthermore, regular audits and analysis of historical transaction data would be conducted to assess the system's effectiveness over time and identify any emerging patterns or trends in fraudulent behavior. By combining these approaches, we can iteratively refine and optimize our fraud prevention measures to stay ahead of evolving fraud tactics and ensure the ongoing integrity and security of our financial ecosystem.

**Entire Code:**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from imblearn.over\_sampling import SMOTE

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification\_report, roc\_auc\_score, confusion\_matrix, roc\_curve

from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

# Load the data

data = pd.read\_csv(r'C:\Users\saraf\Desktop\Insaid\Fraud.csv')

# Check for missing values

missing\_values = data.isnull().sum()

print(missing\_values)

# Visualize the distribution of numeric features to identify outliers

numeric\_features = ['amount', 'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest', 'newbalanceDest']

for feature in numeric\_features:

    plt.figure(figsize=(10, 4))

    sns.boxplot(data[feature])

    plt.title(f'Box plot of {feature}')

    plt.show()

# Calculate VIF for each feature

X = data[numeric\_features]

vif\_data = pd.DataFrame()

vif\_data["Feature"] = X.columns

vif\_data["VIF"] = [variance\_inflation\_factor(X.values, i) for i in range(len(X.columns))]

print(vif\_data)

# Handle categorical variables

data = pd.get\_dummies(data, columns=['type'], drop\_first=True)

# Split the data

X = data.drop(['isFraud', 'isFlaggedFraud', 'nameOrig', 'nameDest'], axis=1)

y = data['isFraud']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Handle class imbalance using SMOTE

smote = SMOTE(random\_state=42)

X\_train\_res, y\_train\_res = smote.fit\_resample(X\_train, y\_train)

# Train Random Forest classifier

model = RandomForestClassifier(random\_state=42)

model.fit(X\_train\_res, y\_train\_res)

# Predictions

y\_pred = model.predict(X\_test)

y\_prob = model.predict\_proba(X\_test)[:, 1]

# Performance metrics

print(classification\_report(y\_test, y\_pred))

print(f"AUC-ROC: {roc\_auc\_score(y\_test, y\_prob)}")

# Confusion Matrix

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues')

plt.title('Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.show()

# ROC Curve

fpr, tpr, thresholds = roc\_curve(y\_test, y\_prob)

plt.figure(figsize=(8, 6))

plt.plot(fpr, tpr, label=f'AUC-ROC: {roc\_auc\_score(y\_test, y\_prob):.2f}')

plt.plot([0, 1], [0, 1], 'k--')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve')

plt.legend()

plt.show()

# Feature importance

feature\_importance = model.feature\_importances\_

features = X.columns

important\_features = pd.DataFrame({'Feature': features, 'Importance': feature\_importance})

important\_features.sort\_values(by='Importance', ascending=False, inplace=True)

print(important\_features.head(10))