# Loading Dataset

import pandas as pd

import numpy as np

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

from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier, LocalOutlierFactor

from sklearn.ensemble import IsolationForest

from xgboost import XGBClassifier

from sklearn.metrics import classification\_report, confusion\_matrix, roc\_auc\_score, accuracy\_score, f1\_score

# Load the first dataset

df = pd.read\_csv('/content/drive/MyDrive/AAA\_datasets/thesis/creditcard\_2023.csv')

# Drop the 'id' column as it's not needed

df = df.drop('id', axis=1)

# Shuffle and split the dataset into training and testing sets

X = df.drop('Class', axis=1)

y = df['Class']

# Split the dataset into training and testing sets (80% train, 20% test)

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

# Scale the 'Amount' column

scaler = StandardScaler()

X\_train['Amount'] = scaler.fit\_transform(X\_train[['Amount']])

X\_test['Amount'] = scaler.transform(X\_test[['Amount']]) # Use the same scaler for the test set

# Train supervised models

xgb = XGBClassifier(use\_label\_encoder=False, eval\_metric='logloss')

xgb.fit(X\_train, y\_train)

knn = KNeighborsClassifier(n\_neighbors=3)

knn.fit(X\_train, y\_train)

# Train unsupervised models

iso\_forest = IsolationForest(contamination=0.01, random\_state=42, n\_estimators=100)

iso\_forest.fit(X\_train)

# Predict on training set for each model

xgb\_probs = xgb.predict\_proba(X\_train)[:, 1] # Probability of class 1 (fraud)

xgb\_preds = np.where(xgb\_probs >= 0.9, 1, 0)

knn\_preds = knn.predict(X\_train)

iso\_preds = iso\_forest.predict(X\_train)

iso\_preds = np.where(iso\_preds == -1, 1, 0)

lof\_test = LocalOutlierFactor(n\_neighbors=20, contamination=0.01)

lof\_preds\_test = lof\_test.fit\_predict(X\_train)

lof\_preds = np.where(lof\_preds\_test == -1, 1, 0)

# Evaluate each model on the training set

def evaluate\_model(y\_true, y\_pred, model\_name):

print(f"{model\_name} Evaluation:")

print(confusion\_matrix(y\_true, y\_pred))

print(classification\_report(y\_true, y\_pred, target\_names=["Legit", "Fraud"]))

print("Accuracy:", accuracy\_score(y\_true, y\_pred))

print("ROC AUC Score:", roc\_auc\_score(y\_true, y\_pred))

print("\n")

# Evaluate each model

evaluate\_model(y\_train, xgb\_preds, "XGBoost")

evaluate\_model(y\_train, knn\_preds, "KNN")

evaluate\_model(y\_train, iso\_preds, "Isolation Forest")

evaluate\_model(y\_train, lof\_preds, "Local Outlier Factor")

# Combine predictions using majority vote for hybrid model

final\_preds = []

for i in range(len(X\_train)):

vote\_score = (

(1 if xgb\_preds[i] == 1 else 0) \* 1.5 +

(1 if knn\_preds[i] == 1 else 0) \* 1.5 +

(1 if iso\_preds[i] == 1 else 0) \* 1 +

(1 if lof\_preds[i] == 1 else 0) \* 1

)

final\_preds.append(1 if vote\_score >= 2.5 else 0)

# Evaluate results on the hybrid model

print("Hybrid Model Evaluation on Training Set:")

evaluate\_model(y\_train, final\_preds, "Hybrid Model")

# Compare fraud rates on training dataset

print("Fraud rate predicted by each model on training dataset:")

print("XGBoost:", sum(xgb\_preds), "frauds")

print("KNN:", sum(knn\_preds), "frauds")

print("Isolation Forest:", sum(iso\_preds), "frauds")

print("LOF:", sum(lof\_preds), "frauds")

print(f"Final hybrid fraud detection rate (train set): {sum(final\_preds) / len(final\_preds):.4f}")

# Macro and Weighted F1-Score Calculations for training dataset

print("Macro F1 (Train):", f1\_score(y\_train, final\_preds, average='macro'))

print("Weighted F1 (Train):", f1\_score(y\_train, final\_preds, average='weighted'))

# Predict using trained models on the test dataset

xgb\_probs\_test = xgb.predict\_proba(X\_test)[:, 1]

xgb\_preds\_test = np.where(xgb\_probs\_test >= 0.9, 1, 0)

knn\_preds\_test = knn.predict(X\_test)

iso\_preds\_test = np.where(iso\_forest.predict(X\_test) == -1, 1, 0)

lof\_preds\_test = np.where(lof\_test.fit\_predict(X\_test) == -1, 1, 0) # Use the same LOF model

# Evaluate each model on the test dataset

print("Evaluation on Test Dataset:")

evaluate\_model(y\_test, xgb\_preds\_test, "XGBoost")

evaluate\_model(y\_test, knn\_preds\_test, "KNN")

evaluate\_model(y\_test, iso\_preds\_test, "Isolation Forest")

evaluate\_model(y\_test, lof\_preds\_test, "Local Outlier Factor")

# Voting ensemble for the test dataset

final\_preds\_test = []

for i in range(len(X\_test)):

vote\_score = (

(1 if xgb\_preds\_test[i] == 1 else 0) \* 1.5 +

(1 if knn\_preds\_test[i] == 1 else 0) \* 1.5 +

(1 if iso\_preds\_test[i] == 1 else 0) \* 1 +

(1 if lof\_preds\_test[i] == 1 else 0) \* 1

)

final\_preds\_test.append(1 if vote\_score >= 2.5 else 0)

# Evaluate on test dataset for hybrid model

print("Hybrid Model Evaluation on Test Dataset:")

evaluate\_model(y\_test, final\_preds\_test, "Hybrid Model")

# Compare fraud rates on test dataset

print("Fraud rate predicted by each model on test dataset:")

print("XGBoost:", sum(xgb\_preds\_test), "frauds")

print("KNN:", sum(knn\_preds\_test), "frauds")

print("Isolation Forest:", sum(iso\_preds\_test), "frauds")

print("LOF:", sum(lof\_preds\_test), "frauds")

print(f"Final hybrid fraud detection rate (test set): {sum(final\_preds\_test) / len(final\_preds\_test):.4f}")

# Macro and Weighted F1-Score Calculations for test dataset

print("Macro F1 (Test):", f1\_score(y\_test, final\_preds\_test, average='macro'))

print("Weighted F1 (Test):", f1\_score(y\_test, final\_preds\_test, average='weighted'))

# Fraud Ratio Comparison Between Predicted and Actual for test dataset

print("Actual fraud ratio (Test):", y\_test.mean())

print("Predicted fraud ratio (Hybrid Test):", sum(final\_preds\_test) / len(final\_preds\_test))

DATASET:

<https://www.kaggle.com/datasets/nelgiriyewithana/credit-card-fraud-detection-dataset-2023/data>