wxrmffvlk

January 28, 2025

[2]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler, RobustScaler
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,␣
 ↪GradientBoostingClassifier, AdaBoostClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
import xgboost as xgb
from sklearn.naive_bayes import GaussianNB
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from sklearn.metrics.cluster import silhouette_score
import matplotlib.pyplot as plt
import seaborn as sns
```

[3]:
```
#loading the dataset
data = pd.read_csv(r"C:\Users\91703\Downloads\Online fraud detection.csv")
```

[4]:
```
data.head()
```

[4]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
0	1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M1979787155	0.0	0.0	0	0
1	1	PAYMENT	1864.28	C1666544295	21249.0	19384.72	M2044282225	0.0	0.0	0	0
2	1	TRANSFER	181.00	C1305486145	181.0	0.00	C553264065	0.0	0.0	1	0
3	1	CASH_OUT	181.00	C840083671	181.0	0.00	C38997010	21182.0	0.0	1	0
4	1	PAYMENT	11668.14	C2048537720	41554.0	29885.86					

```
4  M1230701703             0.0             0.0        0               0
```

[5]:
```
data.dtypes
```

[5]:
```
step                int64
type               object
amount            float64
nameOrig           object
oldbalanceOrg     float64
newbalanceOrig    float64
nameDest           object
oldbalanceDest    float64
newbalanceDest    float64
isFraud             int64
isFlaggedFraud      int64
dtype: object
```

[6]:
```
#Normalizing numerical columns
numeric_cols = ['amount', 'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest',␣
 ↪'newbalanceDest']
scaler = StandardScaler()
data[numeric_cols] = scaler.fit_transform(data[numeric_cols])
```

[7]:
```
data.head()
```

[7]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	\
0	1	PAYMENT	-0.561738	C1231006815	-0.236855	-0.243832	
1	1	PAYMENT	-0.591840	C1666544295	-0.286956	-0.290673	
2	1	TRANSFER	-0.598194	C1305486145	-0.294045	-0.297117	
3	1	CASH_OUT	-0.598194	C840083671	-0.294045	-0.297117	
4	1	PAYMENT	-0.554837	C2048537720	-0.280123	-0.287183	

	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
0	M1979787155	-0.425883	-0.461062	0	0
1	M2044282225	-0.425883	-0.461062	0	0
2	C553264065	-0.425883	-0.461062	1	0
3	C38997010	-0.416661	-0.461062	1	0
4	M1230701703	-0.425883	-0.461062	0	0

[8]:
```
#doing the Label Encoder to transform the below columns
le = LabelEncoder()
data['type'] = le.fit_transform(data['type'])
data['nameOrig'] = le.fit_transform(data['nameOrig'])
data['nameDest'] = le.fit_transform(data['nameDest'])
```

[9]:
```
#Splitting data into training and testing sets:
X = data.drop('isFraud', axis=1) # features
```

```
y = data['isFraud'] # target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[13]:
```
#Define XGBoost classifier #
xgb_model = xgb.XGBClassifier(
max_depth=6,              # Maximum tree depth
learning_rate=0.3,        # Learning rate (step size)
n_estimators=150,         # Number of trees
gamma=0,                  # Minimum loss reduction
subsample=0.8,            # Fraction of samples for each tree
colsample_bytree=0.8,     # Fraction of features for each tree
reg_alpha=0.1,            # L1 regularization term
reg_lambda=0.1            # L2 regularization term
)
#Train XGBoost model
xgb_model.fit(X_train, y_train)
```

[13]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.8, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=0, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.3, 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=150, n_jobs=None,
              num_parallel_tree=None, random_state=None, …)
```

[14]:
```
#Make predictions on test set
y_pred_xgb = xgb_model.predict(X_test)
print(y_pred_xgb)
```

```
[0 0 0 … 0 0 0]
```

[15]:
```
print("XGBoost Model Performance:")
print("Accuracy:", accuracy_score(y_test, y_pred_xgb))
print("Classification Report:", classification_report(y_test, y_pred_xgb))
print("Confusion Matrix:", confusion_matrix(y_test, y_pred_xgb))
```

```
XGBoost Model Performance:
Accuracy: 0.9998283384593377
Classification Report:               precision    recall  f1-score   support

           0       1.00      1.00      1.00    209491
           1       0.98      0.85      0.91       224
```

```
    accuracy                           1.00    209715
   macro avg       0.99      0.93      0.96    209715
weighted avg       1.00      1.00      1.00    209715

Confusion Matrix: [[209488      3]
 [    33    191]]
```

[10]:
```
# ADABOOST CLASSIFIER #
adaboost_model = AdaBoostClassifier(
n_estimators=100,
learning_rate=0.5,
random_state=42
)
```

[16]:
```
#Train AdaBoost model
adaboost_model.fit(X_train, y_train)
```

[16]:
```
AdaBoostClassifier(learning_rate=0.5, n_estimators=100, random_state=42)
```

[15]:
```
import warnings
warnings.filterwarnings('ignore')
```

[17]:
```
#Make predictions on test set
y_pred_adaboost = adaboost_model.predict(X_test)
print(y_pred_adaboost)
```

```
[0 0 0 … 0 0 0]
```

[18]:
```
#Evaluate model performance
print("AdaBoost Model Performance:")
print("Accuracy:", accuracy_score(y_test, y_pred_adaboost))
print("Classification Report:", classification_report(y_test, y_pred_adaboost))
print("Confusion Matrix:", confusion_matrix(y_test, y_pred_adaboost))
```

```
AdaBoost Model Performance:
Accuracy: 0.9995183940109196
Classification Report:                precision    recall  f1-score   support

           0       1.00      1.00      1.00    209491
           1       0.99      0.55      0.71       224

    accuracy                           1.00    209715
   macro avg       1.00      0.78      0.86    209715
weighted avg       1.00      1.00      1.00    209715

Confusion Matrix: [[209490      1]
 [   100    124]]
```

[21]:
```
#Scale data using StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[22]:
```
#K-Nearest Neighbors (KNN) classification #
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train_scaled, y_train)
```

[22]:
```
KNeighborsClassifier()
```

[23]:
```
#Make predictions on test set
y_pred_knn = knn_model.predict(X_test_scaled)
print(y_pred_knn)
```

```
[0 0 0 … 0 0 0]
```

[24]:
```
#Evaluate model performance
print("KNN Model Performance:")
print("Accuracy:", accuracy_score(y_test, y_pred_knn))
print("Classification Report:", classification_report(y_test, y_pred_knn))
print("Confusion Matrix:", confusion_matrix(y_test, y_pred_knn))
```

```
KNN Model Performance:
Accuracy: 0.9994277948644589
Classification Report:                precision    recall  f1-score   support

           0       1.00      1.00      1.00    209491
           1       0.99      0.47      0.64       224

    accuracy                           1.00    209715
   macro avg       0.99      0.73      0.82    209715
weighted avg       1.00      1.00      1.00    209715

Confusion Matrix: [[209490      1]
 [   119    105]]
```

[ ]: