ads-phase-4

October 25, 2023

Naan Mudhalvan

Data Science - Credit Card Fraud Detection (PHASE 4)

[3]:
```python
import pandas as pd
import numpy as np
# read dataset
df = pd.read_csv('/content/creditcard.csv')
print(df)
```

```
        Time        V1        V2        V3        V4        V5        V6  \
0          0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388
1          0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361
2          1 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499
3          1 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203
4          2 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921
…        …       …       …       …       …       …       …
11954  20631  1.504204 -0.411728  0.200090 -0.778753 -0.442232 -0.119677
11955  20636  1.134994  0.096340  0.277921  0.319692  0.742800  1.611803
11956  20638 -6.305012  3.944886 -4.707362  1.539602 -3.934785 -1.730565
11957  20638  1.161960 -0.398297  1.123732 -0.474237 -1.226667 -0.519325
11958  20642  1.291096 -0.226628  0.708386 -0.719236 -0.659099 -0.273757

             V7        V8        V9  …       V21       V22       V23  \
0      0.239599  0.098698  0.363787  … -0.018307  0.277838 -0.110474
1     -0.078803  0.085102 -0.255425  … -0.225775 -0.638672  0.101288
2      0.791461  0.247676 -1.514654  …  0.247998  0.771679  0.909412
3      0.237609  0.377436 -1.387024  … -0.108300  0.005274 -0.190321
4      0.592941 -0.270533  0.817739  … -0.009431  0.798278 -0.137458
…          …       …       …  …       …       …       …
11954 -0.782660 -0.165178  0.691819  … -0.136231 -0.217274 -0.143260
11955 -0.458649  0.390012  1.424541  … -0.395605 -0.743542  0.222256
11956 -2.104936  3.843447  0.863458  …  0.073140 -0.039935 -0.108896
11957 -0.804179  0.070134  3.262926  … -0.121191  0.097255  0.050903
11958 -0.612042 -0.111488  3.032258  …       NaN       NaN       NaN

            V24       V25       V26       V27       V28  Amount  Class
0      0.066928  0.128539 -0.189115  0.133558 -0.021053  149.62    0.0
1     -0.339846  0.167170  0.125895 -0.008983  0.014724    2.69    0.0
```

```
2     -0.689281 -0.327642 -0.139097 -0.055353 -0.059752  378.66    0.0
3     -1.175575  0.647376 -0.221929  0.062723  0.061458  123.50    0.0
4      0.141267 -0.206010  0.502292  0.219422  0.215153   69.99    0.0
…         …         …         …         …         …        …     …
11954 -1.057332  0.529188 -0.235062 -0.012089  0.000905    9.00    0.0
11955 -1.859104 -0.109777  0.279049  0.012398 -0.009090    0.99    0.0
11956  0.691434 -0.261979 -0.447540  0.212900 -0.031021   89.99    0.0
11957  0.330479  0.315692 -0.712765  0.073836  0.028055   11.85    0.0
11958       NaN       NaN       NaN       NaN       NaN     NaN    NaN

[11959 rows x 31 columns]
```

[4]:
```python
df['Hour'] = df['Time'] // 3600  # Convert seconds to hours
df['Day'] = df['Time'] // (3600 * 24)  # Convert seconds to days
df['Weekday'] = pd.to_datetime(df['Time']).dt.dayofweek  # Extract weekday (0 =␣
↪Monday, 6 = Sunday)

#Amount-Based Features
bins = [0, 50, 100, 500, 1000, np.inf]
labels = ['Very Low', 'Low', 'Medium', 'High', 'Very High']
df['Amount_Category'] = pd.cut(df['Amount'], bins=bins, labels=labels)

# Transaction Frequency
df['Transactions_Last_Hour'] = df.groupby('Hour')['Hour'].transform('count')

# Statistical Aggregations
for i in range(1, 29):
    df[f'V{i}_Mean'] = df.groupby('Class')[f'V{i}'].transform('mean')
    df[f'V{i}_StdDev'] = df.groupby('Class')[f'V{i}'].transform('std')

# Interaction Features
df['V1_V2_Multiplication'] = df['V1'] * df['V2']

X = df.drop('Class', axis=1)  # Features
y = df['Class']  # Target variable

from sklearn.impute import SimpleImputer
# Create an imputer object
imputer = SimpleImputer(strategy='mean')

# Select only numeric columns
X_numeric = X.select_dtypes(include=[np.number])

# Impute missing values in features
X_imputed = imputer.fit_transform(X_numeric)
```

[8]:

```python
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X_imputed, y, test_size=0.
 ↪2, random_state=42)

# Create a Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the classifier
rf_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = rf_classifier.predict(X_test)

print(y.isnull().sum())
# Remove rows with NaN values in the target variable
y = y.dropna()
X_imputed = X_imputed[y.index]  # Update X_imputed to match the updated y

from sklearn.impute import SimpleImputer

# Create an imputer object for target variable y
y_imputer = SimpleImputer(strategy='mean')
y_imputed = y_imputer.fit_transform(y.values.reshape(-1, 1))

# Replace y with imputed values
y = pd.Series(y_imputed.ravel(), index=y.index)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

print("Accuracy: {:.2f}%".format(accuracy * 100))
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(classification_rep)
```

```
0
Accuracy: 100.00%
Confusion Matrix:
[[2378    0]
```

```
 [   0   14]]
Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      2378
         1.0       1.00      1.00      1.00        14

    accuracy                           1.00      2392
   macro avg       1.00      1.00      1.00      2392
weighted avg       1.00      1.00      1.00      2392
```

[9]:
```python
from sklearn.metrics import accuracy_score
#Accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: {:.2f}%".format(accuracy * 100))
```

```
Accuracy: 100.00%
```

[10]:
```python
from sklearn.metrics import precision_score, recall_score, f1_score
#Precision, Recall, and F1-Score
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Precision: {:.2f}".format(precision))
print("Recall: {:.2f}".format(recall))
print("F1 Score: {:.2f}".format(f1))
```

```
Precision: 1.00
Recall: 1.00
F1 Score: 1.00
```

[11]:
```python
from sklearn.metrics import confusion_matrix
#Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

```
Confusion Matrix:
[[2378    0]
 [   0   14]]
```

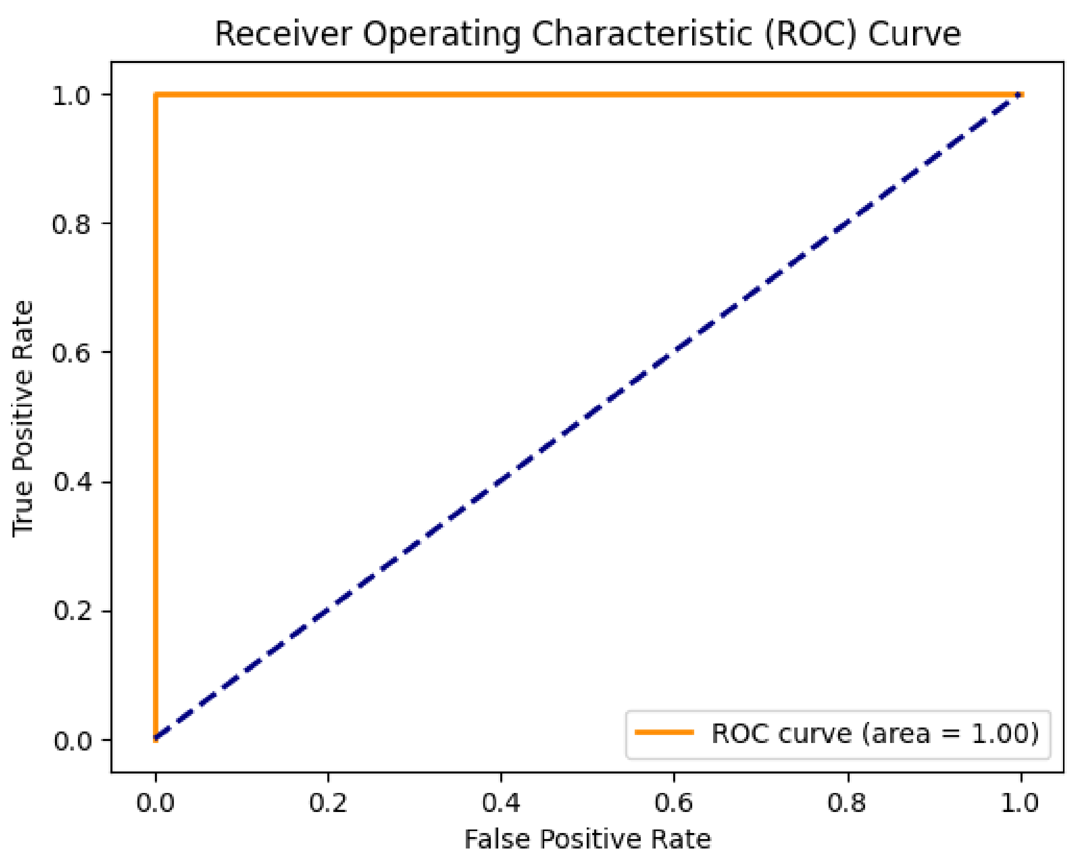
[12]:
```python
from sklearn.metrics import roc_auc_score, roc_curve
import matplotlib.pyplot as plt

#Receiver Operating Characteristic (ROC) Curve and Area Under the Curve␣
 ↪(AUC-ROC):
```

```
# Calculate ROC-AUC score
roc_auc = roc_auc_score(y_test, rf_classifier.predict_proba(X_test)[:, 1])

# Plot ROC curve
fpr, tpr, _ = roc_curve(y_test, rf_classifier.predict_proba(X_test)[:, 1])
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = {:.2f})'.
 ↪format(roc_auc))
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()

print("ROC-AUC Score: {:.2f}".format(roc_auc))
```




```
ROC-AUC Score: 1.00
```

[15]:
```python
from sklearn.model_selection import cross_val_score

# Cross-Validation
# Perform 10-fold cross-validation

cv_scores = cross_val_score(rf_classifier, X_imputed, y, cv=10,␣
↪scoring='accuracy')
print("Cross-Validation Scores:", cv_scores)
print("Mean Accuracy: {:.2f}".format(cv_scores.mean()))
```

```
Cross-Validation Scores: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
Mean Accuracy: 1.00
```