fraud-detection

February 21, 2024

[1]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

[2]:
```python
data = pd.read_csv('/content/creditcard.csv')
data.head()
```

[2]:
```
   Time        V1        V2        V3        V4        V5        V6        V7  \
0   0.0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599
1   0.0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
2   1.0 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
3   1.0 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
4   2.0 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941

         V8        V9  …       V21       V22       V23       V24       V25  \
0  0.098698  0.363787  … -0.018307  0.277838 -0.110474  0.066928  0.128539
1  0.085102 -0.255425  … -0.225775 -0.638672  0.101288 -0.339846  0.167170
2  0.247676 -1.514654  …  0.247998  0.771679  0.909412 -0.689281 -0.327642
3  0.377436 -1.387024  … -0.108300  0.005274 -0.190321 -1.175575  0.647376
4 -0.270533  0.817739  … -0.009431  0.798278 -0.137458  0.141267 -0.206010

        V26       V27       V28  Amount  Class
0 -0.189115  0.133558 -0.021053  149.62    0.0
1  0.125895 -0.008983  0.014724    2.69    0.0
2 -0.139097 -0.055353 -0.059752  378.66    0.0
3 -0.221929  0.062723  0.061458  123.50    0.0
4  0.502292  0.219422  0.215153   69.99    0.0

[5 rows x 31 columns]
```

[3]:
```python
data.describe()
```

[3]:
```
                Time             V1             V2             V3  \
count  182329.000000  182329.000000  182329.000000  182329.000000
mean    64969.194676      -0.140528       0.022185       0.417493
std     30828.520647       1.873983       1.614571       1.416903
```

```
min         0.000000     -56.407510     -72.715728     -33.680984
25%     42410.000000      -0.975751      -0.560485      -0.172141
50%     63339.000000      -0.139500       0.094318       0.570588
75%     81849.000000       1.196395       0.791998       1.267890
max    125353.000000       2.439207      22.057729       9.382558

                  V4             V5             V6             V7  \
count  182329.000000  182329.000000  182329.000000  182329.000000
mean        0.096426      -0.150145       0.053535      -0.068736
std         1.384183       1.355202       1.304002       1.212992
min        -5.683171     -42.147898     -26.160506     -43.557242
25%        -0.768971      -0.809987      -0.699692      -0.582837
50%         0.099818      -0.203398      -0.209359      -0.023584
75%         0.914961       0.412695       0.447010       0.477721
max        16.875344      34.801666      22.529298      36.677268

                  V8             V9  …            V21            V22  \
count  182329.000000  182328.000000  …  182328.000000  182328.000000
mean        0.025907       0.014692  …      -0.021476      -0.067798
std         1.223914       1.144658  …       0.742347       0.678239
min       -73.216718     -13.434066  …     -34.830382     -10.933144
25%        -0.170715      -0.661010  …      -0.228923      -0.542896
50%         0.050375      -0.076790  …      -0.049453      -0.053276
75%         0.344873       0.634247  …       0.138317       0.389549
max        20.007208      15.594995  …      27.202839      10.503090

                 V23            V24            V25            V26  \
count  182328.000000  182328.000000  182328.000000  182328.000000
mean       -0.019683       0.007206       0.078579       0.010185
std         0.594938       0.600877       0.476265       0.489616
min       -44.807735      -2.836627     -10.295397      -2.604551
25%        -0.169356      -0.335944      -0.216393      -0.330644
50%        -0.032859       0.056785       0.122284      -0.059203
75%         0.106009       0.418491       0.393032       0.268662
max        19.002942       4.022866       7.519589       3.517346

                 V27            V28         Amount          Class
count  182328.000000  182328.000000  182328.00000  182328.000000
mean        0.002032       0.002194      88.52699       0.002002
std         0.392343       0.307251     247.55138       0.044698
min       -22.565679     -11.710896       0.00000       0.000000
25%        -0.066026      -0.034975       5.76000       0.000000
50%         0.007456       0.020019      22.43000       0.000000
75%         0.089507       0.078215      78.12000       0.000000
max        12.152401      33.847808   19656.53000       1.000000

[8 rows x 31 columns]
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 182329 entries, 0 to 182328
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   Time    182329 non-null  float64
 1   V1      182329 non-null  float64
 2   V2      182329 non-null  float64
 3   V3      182329 non-null  float64
 4   V4      182329 non-null  float64
 5   V5      182329 non-null  float64
 6   V6      182329 non-null  float64
 7   V7      182329 non-null  float64
 8   V8      182329 non-null  float64
 9   V9      182328 non-null  float64
 10  V10     182328 non-null  float64
 11  V11     182328 non-null  float64
 12  V12     182328 non-null  float64
 13  V13     182328 non-null  float64
 14  V14     182328 non-null  float64
 15  V15     182328 non-null  float64
 16  V16     182328 non-null  float64
 17  V17     182328 non-null  float64
 18  V18     182328 non-null  float64
 19  V19     182328 non-null  float64
 20  V20     182328 non-null  float64
 21  V21     182328 non-null  float64
 22  V22     182328 non-null  float64
 23  V23     182328 non-null  float64
 24  V24     182328 non-null  float64
 25  V25     182328 non-null  float64
 26  V26     182328 non-null  float64
 27  V27     182328 non-null  float64
 28  V28     182328 non-null  float64
 29  Amount  182328 non-null  float64
 30  Class   182328 non-null  float64
dtypes: float64(31)
memory usage: 43.1 MB
```

[5]:
```
data.isna().sum()
```

[5]:
```
Time      0
V1        0
V2        0
V3        0
```

```
V4        0
V5        0
V6        0
V7        0
V8        0
V9        1
V10       1
V11       1
V12       1
V13       1
V14       1
V15       1
V16       1
V17       1
V18       1
V19       1
V20       1
V21       1
V22       1
V23       1
V24       1
V25       1
V26       1
V27       1
V28       1
Amount    1
Class     1
dtype: int64
```

Data Preprocessing:

[6]:
```
from sklearn.impute import SimpleImputer

# Impute missing values with the mean for numerical columns
imputer = SimpleImputer(strategy="mean")

# Fill missing values in numerical columns
data = pd.DataFrame(imputer.fit_transform(data), columns=data.columns)

# Check if any missing values remain
print(data.isnull().sum())
```

```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
```

```
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

[7]:

```
from sklearn.model_selection import train_test_split

# Separate features (X) and target variable (y)
X = data.drop(columns=["Class"])  # Features
y = data["Class"]  # Target variable

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

Feature Selection:

[9]:

```
from sklearn.preprocessing import StandardScaler

# Create a StandardScaler object
scaler = StandardScaler()

# Fit the scaler to the training data and transform it
X_train_scaled = scaler.fit_transform(X_train)

# Transform the testing data using the fitted scaler
```

```
X_test_scaled = scaler.transform(X_test)
```

Data Splitting:

[10]:
```
from sklearn.model_selection import train_test_split

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

Model Selection:

[11]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score, roc_auc_score
```

[12]:
```
# Define models
models = {
    'Logistic Regression': LogisticRegression(max_iter=1000),
    'Random Forest': RandomForestClassifier(),
    'Gradient Boosting': GradientBoostingClassifier(),
    'Neural Network': MLPClassifier(max_iter=1000)  # adjusting max_iter based␣
↪on convergence
}
```

[13]:
```
# Define a threshold value to classify as 1 (fraudulent)
threshold_value = 0.5

# Convert continuous target values to binary labels
y_train = (y_train > threshold_value).astype(int)
```

Model Evaluation:

[14]:
```
# Train and evaluate models
for name, model in models.items():
    print(f"\033[1mTraining {name}:\033[0m")
    model.fit(X_train, y_train)

    # Predict on the test set
    y_pred = model.predict(X_test)

    # Evaluate the model
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
```

```
    auc_roc = roc_auc_score(y_test, y_pred)

    # Print evaluation metrics
    print(f"\033[1;3mAccuracy:\033[0m {accuracy:.4f}, \n\033[1;3mPrecision:
↪\033[0m {precision:.4f}, \n\033[1;3mRecall:\033[0m {recall:.4f}, \n\033[1;
↪3mF1-score:\033[0m {f1:.4f}, \n\033[1;3mAUC-ROC:\033[0m {auc_roc:.4f}\n")
```

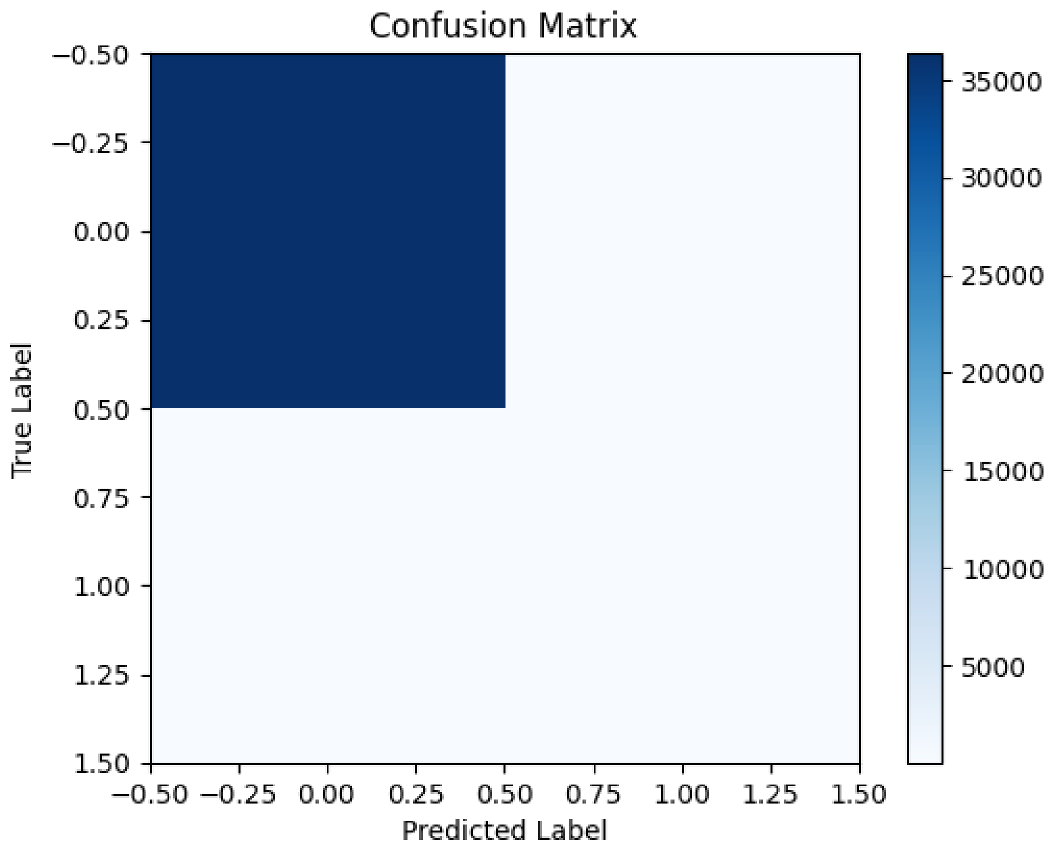
```
Training Logistic Regression:
Accuracy: 0.9990,
Precision: 0.8085,
Recall: 0.5758,
F1-score: 0.6726,
AUC-ROC: 0.7878

Training Random Forest:
Accuracy: 0.9995,
Precision: 0.9259,
Recall: 0.7576,
F1-score: 0.8333,
AUC-ROC: 0.8787

Training Gradient Boosting:
Accuracy: 0.9989,
Precision: 0.8000,
Recall: 0.5455,
F1-score: 0.6486,
AUC-ROC: 0.7726

Training Neural Network:
Accuracy: 0.9979,
Precision: 0.4479,
Recall: 0.6515,
F1-score: 0.5309,
AUC-ROC: 0.8250
```

Data Visualization:

[15]:
```
from sklearn.metrics import confusion_matrix, roc_curve,␣
↪precision_recall_curve, auc
```

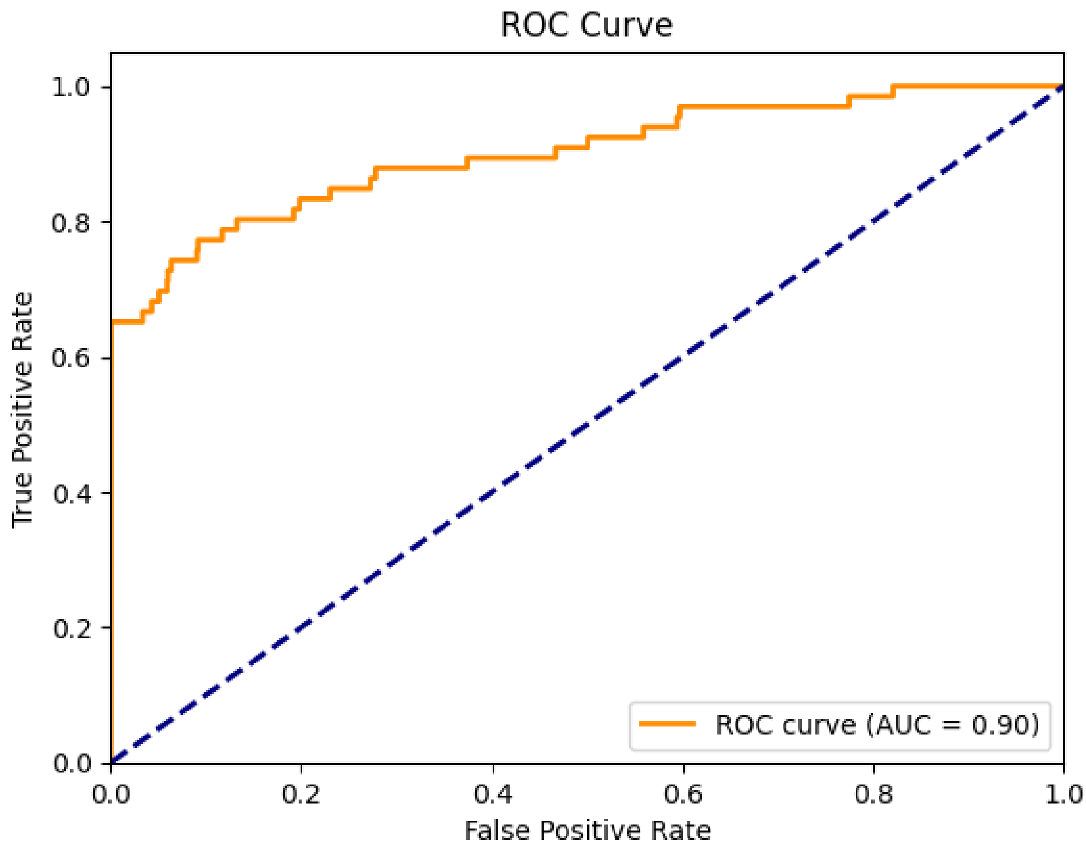
[16]:
```
# Plot confusion matrix
y_pred = model.predict(X_test)
conf_matrix = confusion_matrix(y_test, y_pred)
plt.imshow(conf_matrix, cmap='Blues', interpolation='nearest')
plt.title('Confusion Matrix')
plt.colorbar()
plt.xlabel('Predicted Label')
```

```python
plt.ylabel('True Label')
plt.show()
```

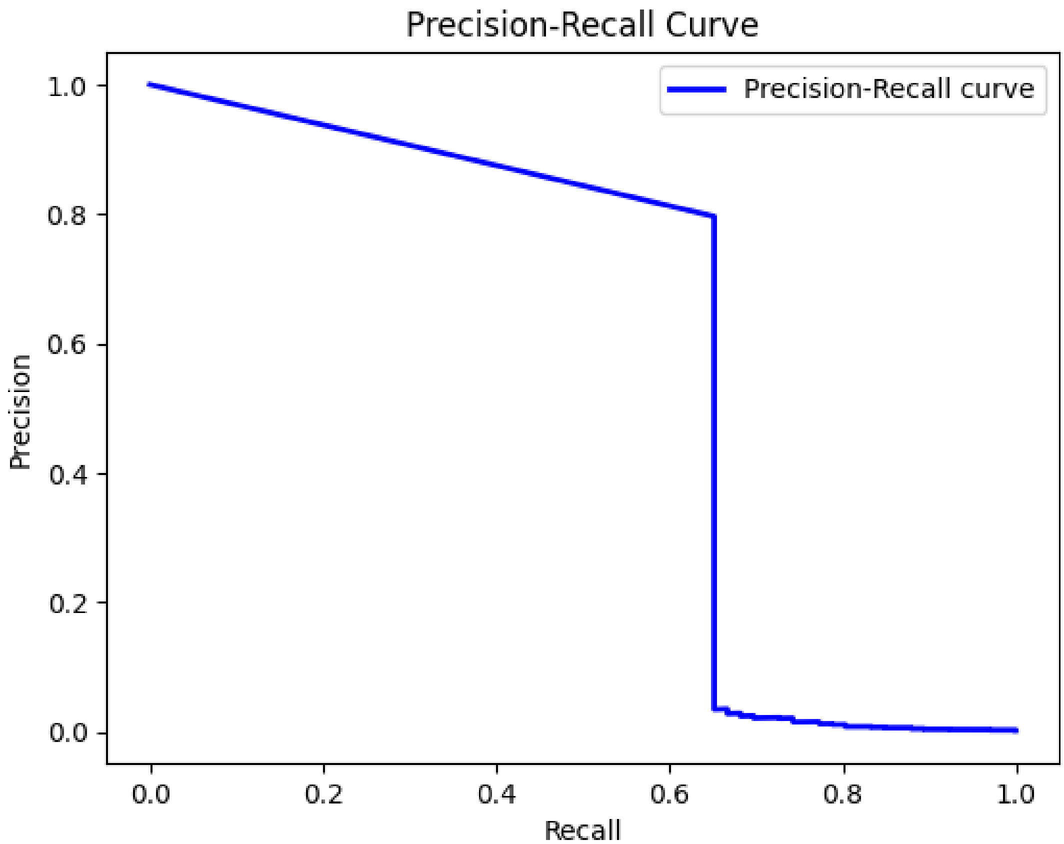


[17]:

```python
# Plot ROC curve
y_pred_proba = model.predict_proba(X_test)[:, 1]
fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.
↪2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()
```



[18]:
```
# Plot precision-recall curve
precision, recall, _ = precision_recall_curve(y_test, y_pred_proba)
plt.plot(recall, precision, color='blue', lw=2, label='Precision-Recall curve')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc='upper right')
plt.show()
```



Precision-Recall Curve
Precision-Recall curve
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0