Lab 03

Working with Standard Scaler and SMOTE

1.Use Standard Scaler to standardize the features of a Credit card fraud dataset. Include code, description and screenshots of outputs.

- Standard scalar transforms the distribution of each feature to have a mean of zero and a standard deviation of one.
- It ensures that all features are on the same scale, preventing any single feature from dominating the learning process due to its larger magnitude.

1. Import all required libraries.

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
```

2. Load the data set and read it to a data frame.
3. Display basic information about the dataset.

```
data=pd.read_csv("credit_card.csv")

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

   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
1  0.125895 -0.008983  0.014724    2.69      0
2 -0.139097 -0.055353 -0.059752  378.66      0
3 -0.221929  0.062723  0.061458  123.50      0
4  0.502292  0.219422  0.215153   69.99      0

[5 rows x 31 columns]
```

4. Select features to standardize and initialize standard scaler.

```
feature=data.drop(columns=['Time','Class'])
```

```
[27]:
scaler=StandardScaler()
```

Time is not a feature that reflects fraud pattern and Class is our target value. So, there is no need of standardization for these values.

5. Apply scaler to the features and store the result in a new data frame.

```
scaled_feature=scaler.fit_transform(feature)
```

```
[31]:
df=pd.DataFrame(scaled_feature,columns=feature.columns)
```

6. Combine Time and Class with scaled features and display the results.

```
df['Time'] = data['Time']
df['Class']=data['Class']

[35]:
print("\nStandardized Feature:\n",df.head())

Standardized Feature:
          V1        V2        V3        V4        V5        V6        V7  \
0 -0.694242 -0.044075  1.672773  0.973366 -0.245117  0.347068  0.193679
1  0.608496  0.161176  0.109797  0.316523  0.043483 -0.061820 -0.063700
2 -0.693500 -0.811578  1.169468  0.268231 -0.364572  1.351454  0.639776
3 -0.493325 -0.112169  1.182516 -0.609727 -0.007469  0.936150  0.192071
4 -0.591330  0.531541  1.021412  0.284655 -0.295015  0.071999  0.479302

         V8        V9       V10  ...       V22       V23       V24       V25  \
0  0.082637  0.331128  0.083386  ...  0.382854 -0.176911  0.110507  0.246585
1  0.071253 -0.232494 -0.153350  ... -0.880077  0.162201 -0.561131  0.320694
2  0.207373 -1.378675  0.190700  ...  1.063358  1.456320 -1.138092 -0.628537
3  0.316018 -1.262503 -0.050468  ...  0.007267 -0.304777 -1.941027  1.241904
4 -0.226510  0.744326  0.691625  ...  1.100011 -0.220123  0.233250 -0.395202

        V26       V27       V28    Amount  Time  Class
0 -0.392170  0.330892 -0.063781  0.244964   0.0      0
1  0.261069 -0.022256  0.044608 -0.342475   0.0      0
2 -0.288447 -0.137137 -0.181021  1.160686   1.0      0
3 -0.460217  0.155396  0.186189  0.140534   1.0      0
4  1.041611  0.543620  0.651816 -0.073403   2.0      0

[5 rows x 31 columns]
```

2. Apply SMOTE to address the class imbalance problem. Include code, description and screenshots of outputs.

Synthetic minority oversampling technique is one of the most commonly used oversampling methods to solve the imbalance problem.

1. Import required libraries and read the dataset.

```
import pandas as pd
from collections import Counter
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
```

```
data=pd.read_csv('credit_card.csv')
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
```

```
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

2. Separate features and targets. Then check the class distribution.

```
x=data.drop('Class',axis=1)
y=data['Class']

[27]:
print("Class distribution before SMOTE:", Counter(y))
Class distribution before SMOTE: Counter({0: 284315, 1: 492})
```

The output is showing class imabalance. Inorder to avoid this imbalance, we can use SMOTE.

3. Split dataset into training and test sets.

```
x_train,x_test, y_train, y_test = train_test_split(x,y,test_size=0.2,random_state=42,stratify=y)
```

4. Apply SMOTE to the training dataset. Then check class distribution again.

```
smote=SMOTE(random_state=42)
x_train_smote, y_train_smote = smote.fit_resample(x_train, y_train)

[25]:
print("Class distribution after SMOTE:", Counter(y_train_smote))
Class distribution after SMOTE: Counter({0: 227451, 1: 227451})
```

SMOTE is applied only on the training set to prevent information leakage into the test set.

5. Display the resampled training set.

```python
x_train_smote.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 454902 entries, 0 to 454901
Data columns (total 30 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    454902 non-null  float64
 1   V1      454902 non-null  float64
 2   V2      454902 non-null  float64
 3   V3      454902 non-null  float64
 4   V4      454902 non-null  float64
 5   V5      454902 non-null  float64
 6   V6      454902 non-null  float64
 7   V7      454902 non-null  float64
 8   V8      454902 non-null  float64
 9   V9      454902 non-null  float64
 10  V10     454902 non-null  float64
 11  V11     454902 non-null  float64
 12  V12     454902 non-null  float64
 13  V13     454902 non-null  float64
 14  V14     454902 non-null  float64
 15  V15     454902 non-null  float64
 16  V16     454902 non-null  float64
 17  V17     454902 non-null  float64
 18  V18     454902 non-null  float64
 19  V19     454902 non-null  float64
 20  V20     454902 non-null  float64
 21  V21     454902 non-null  float64
 22  V22     454902 non-null  float64
 23  V23     454902 non-null  float64
 24  V24     454902 non-null  float64
 25  V25     454902 non-null  float64
 26  V26     454902 non-null  float64
 27  V27     454902 non-null  float64
 28  V28     454902 non-null  float64
 29  Amount  454902 non-null  float64
dtypes: float64(30)
memory usage: 104.1 MB
```

3. Apply Decision Tree Classifier Model and Evaluate the Performance with Confusion matrix, accuracy, precision, Recall and F1 Score.

1. Import required libraries.

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from imblearn.over_sampling import SMOTE
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score
```

2. Load and analyze the dataset.

```
data=pd.read_csv("creditcard.csv")
```

```
data.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.18
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.12
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.13
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.50

5 rows × 31 columns

3. Separate the data as features and targets. Since Time column is not relevant, we can drop that column. Now, check the class distribution.

```
x = data.drop(columns=['Class', 'Time'])
y = data['Class']
```

```
print("Class distribution before SMOTE:", y.value_counts())
```

```
Class distribution before SMOTE: Class
0    284315
1       492
Name: count, dtype: int64
```

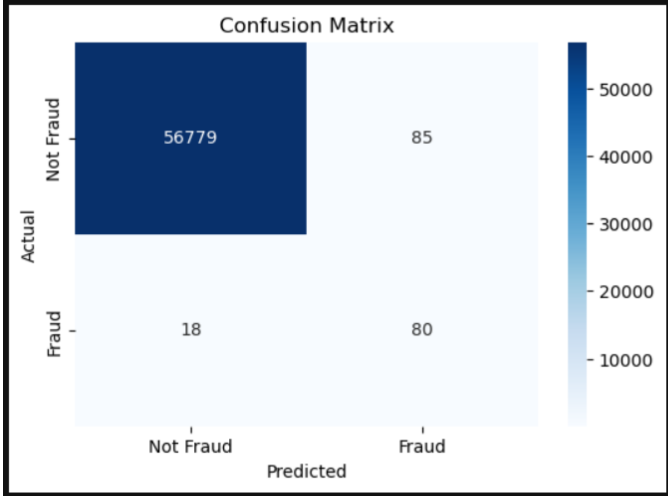
4. Split the data into training and test sets.

```
x_train,x_test, y_train, y_test = train_test_split(x, y,test_size=0.2, random_state=42, stratify=y)
```

5.Apply SMOTE to training set, to make it balanced.

```
smote = SMOTE(random_state=42)
x_train_smote, y_train_smote = smote.fit_resample(x_train, y_train)
```

```python
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Fraud', 'Fraud'], yticklabels=['Not Fraud', 'Fraud'])
plt.title("Confusion Matrix")
plt.ylabel("Actual")
plt.xlabel("Predicted")
plt.show()
```



6.Check the class distribution after applying SMOTE.

```python
print("Class distribution after SMOTE:", pd.Series(y_train_smote).value_counts())
```

```
Class distribution after SMOTE: Class
0    227451
1    227451
Name: count, dtype: int64
```

7.Initialize the decision tree classifier and train the model.

```python
model = DecisionTreeClassifier(random_state=42)
model.fit(x_train_smote, y_train_smote)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

8.Predict value on test set

```python
y_pred = model.predict(x_test)
```

9. Evaluate model’s performance.

```python
conf_matrix = confusion_matrix(y_test, y_pred)
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)
```

10. Display the results.

```python
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
```

```
Accuracy: 0.9982
Precision: 0.4848
Recall: 0.8163
F1 Score: 0.6084
```

```python
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Fraud', 'Fraud'], yticklabels=['Not Fraud', 'Fraud'])
plt.title("Confusion Matrix")
plt.ylabel("Actual")
plt.xlabel("Predicted")
plt.show()
```