# EXPERIMENT NO. 13

| |
|---|
| **Student Name and Roll Number:** Piyush Gambhir – 21CSU349 |
| **Semester /Section:** Semester-V – AIML-V-B (AL-3) |
| **Link to Code:** NCU-Lab-Manual-And-End-Semester-Projects/NCU-CSL347-AAIES-Lab_Manual at main · Piyush-Gambhir/NCU-Lab-Manual-And-End-Semester-Projects (github.com) |
| **Date:** 25.11.2023 |
| **Faculty Signature:** |
| **Grade:** |

**Objective(s):**

- Understand and study Support Vector Machines for classification problems.
- Study about how Anomaly Detection can be performed using supervised learning.

**Outcome:**
Students will be familiarized with Support Vector Machine for classification.

**Problem Statement:**

Python program to implement Credit Card Fraud detection using Support Vector Machine classification.

URL for the dataset:

https://www.kaggle.com/datasets/dhanushnarayananr/credit-card-fraud

**Background Study:**

SVMs are machine learning algorithms used for classification and regression. In anomaly detection, SVMs serve as one-class classifiers, identifying anomalies as data points deviating from the learned normal representation. They create a decision boundary to separate normal data from anomalies, making them effective in detecting novel or rare instances outside the normal data distribution. SVM-based anomaly detection is useful when labeled anomaly data is scarce, making it applicable in domains like fraud and intrusion detection.

**Question Bank:**

1. What is a Support Vector Machine?
   A Support Vector Machine (SVM) is a supervised machine learning algorithm used for classification and regression tasks. It works by finding a hyperplane that best separates different classes or predicts a target value while maximizing the margin (distance) between the hyperplane and the closest data points (support vectors).

2. How SVM is used to solve problems?
   SVM is used to solve problems by identifying the optimal hyperplane that separates different classes or predicts values for regression tasks. It involves selecting the hyperplane that maximizes the margin between support vectors while minimizing classification errors. SVM can also employ kernel functions to transform data into higher-dimensional spaces, allowing it to handle non-linear separations.

3. List out the advantages and disadvantages of SVM on Anomaly detection?

Advantages of SVM for Anomaly Detection

- Effective in High-Dimensional Spaces: SVM works well in high-dimensional feature spaces, which is beneficial for capturing complex relationships in anomaly data.
- Ability to Handle Unbalanced Data: SVM can be adjusted to handle unbalanced datasets, which is common in anomaly detection scenarios.
- Robust to Outliers: SVM's focus on support vectors makes it less sensitive to outliers, which is important in anomaly detection where outliers represent anomalies.

Disadvantages of SVM for Anomaly Detection:

- Sensitivity to Hyperparameters: SVM's performance is influenced by hyperparameters like the choice of kernel and regularization parameters, which may require careful tuning.
- Computationally Intensive: Training SVMs can be computationally intensive, especially for large datasets or when using non-linear kernels.
- Difficulty with Large Datasets: SVM's efficiency diminishes with larger datasets due to the need to store and compute kernel matrices.
- Limited Interpretability: SVMs provide decision boundaries without inherent feature importance scores, making it less interpretable compared to some other algorithms.
- Keep in mind that the effectiveness of SVM for anomaly detection depends on the specific characteristics of the data and the problem at hand.

# Student Work Area

Algorithm/Flowchart/Code/Sample Outputs

# Experiment 13

### Problem Statement:

Python program to implement Credit Card Fraud detection using Support Vector Machine classification.

URL for the dataset: https://www.kaggle.com/datasets/dhanushnarayananr/credit-card-fraud

### Install Dependencies:

In [ ]:
```
! pip install tabulate
```

```
Requirement already satisfied: tabulate in c:\users\mainp\appdata\local\programs\python\python
311\lib\site-packages (0.9.0)
```

### Code:

In [ ]:
```
# importing required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as snss

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, accuracy_score, precisi
```

In [ ]:
```
# importing the dataset
dataset = pd.read_csv('card_transdata_dataset.csv')
print(dataset.head().to_markdown())
```

```
|    |   distance_from_home |   distance_from_last_transaction |   ratio_to_median_purchase_pr
ice |   repeat_retailer |   used_chip |   used_pin_number |   online_order |   fraud |
|---:|---------------------:|---------------------------------:|------------------------------
---:|------------------:|------------:|------------------:|---------------:|--------:|
|  0 |             57.8779  |                         0.31114  |                         1.945
94  |                 1 |           1 |                 0 |              0 |       0 |
|  1 |             10.8299  |                         0.175592 |                         1.294
22  |                 1 |           0 |                 0 |              0 |       0 |
|  2 |              5.09108 |                         0.805153 |                         0.427
715 |                 1 |           0 |                 0 |              1 |       0 |
|  3 |              2.24756 |                         5.60004  |                         0.362
663 |                 1 |           1 |                 0 |              1 |       0 |
|  4 |             44.1909  |                         0.566486 |                         2.222
77  |                 1 |           1 |                 0 |              1 |       0 |
```

In [ ]:

In [ ]:
```
# checking for null values
print("\nChecking for null values:")
print(dataset.isnull().sum())

# dropping null values

dataset = dataset.dropna()
print(dataset.isnull().sum)
```

```
Checking for null values:
distance_from_home                0
distance_from_last_transaction    0
ratio_to_median_purchase_price    0
repeat_retailer                   0
used_chip                         0
used_pin_number                   0
online_order                      0
fraud                             0
dtype: int64
<bound method NDFrame._add_numeric_operations.<locals>.sum of         distance_from_home  dist
ance_from_last_transaction  \
0                    False                           False
1                    False                           False
2                    False                           False
3                    False                           False
4                    False                           False
...                    ...                             ...
999995               False                           False
999996               False                           False
999997               False                           False
999998               False                           False
999999               False                           False

        ratio_to_median_purchase_price  repeat_retailer  used_chip  \
0                                False            False      False
1                                False            False      False
2                                False            False      False
3                                False            False      False
4                                False            False      False
...                                ...              ...        ...
999995                           False            False      False
999996                           False            False      False
999997                           False            False      False
999998                           False            False      False
999999                           False            False      False

        used_pin_number  online_order  fraud
0                 False         False  False
1                 False         False  False
2                 False         False  False
3                 False         False  False
4                 False         False  False
...                 ...           ...    ...
999995            False         False  False
999996            False         False  False
999997            False         False  False
999998            False         False  False
999999            False         False  False

[1000000 rows x 8 columns]>
```

In [ ]:

```
# checking for duplicate values
print("\nChecking for duplicate values:")
print(dataset.duplicated().sum())
```

```
Checking for duplicate values:
0
```

In [ ]:

```
# checking for outliers
print("\nChecking for outliers:")
print(dataset.describe())
```

```
Checking for outliers:
       distance_from_home  distance_from_last_transaction  \
count      1000000.000000                  1000000.000000
mean            26.628792                        5.036519
std             65.390784                       25.843093
min              0.004874                        0.000118
25%              3.878008                        0.296671
50%              9.967760                        0.998650
75%             25.743985                        3.355748
max          10632.723672                    11851.104565

       ratio_to_median_purchase_price  repeat_retailer       used_chip  \
count                  1000000.000000   1000000.000000  1000000.000000
mean                         1.824182         0.881536        0.350399
std                          2.799589         0.323157        0.477095
min                          0.004399         0.000000        0.000000
25%                          0.475673         1.000000        0.000000
50%                          0.997717         1.000000        0.000000
75%                          2.096370         1.000000        1.000000
max                        267.802942         1.000000        1.000000

       used_pin_number    online_order           fraud
count   1000000.000000  1000000.000000  1000000.000000
mean          0.100608        0.650552        0.087403
std           0.300809        0.476796        0.282425
min           0.000000        0.000000        0.000000
25%           0.000000        0.000000        0.000000
50%           0.000000        1.000000        0.000000
75%           0.000000        1.000000        0.000000
max           1.000000        1.000000        1.000000
```

In [ ]:
```
# checking the number of fraud and non-fraud transactions
print("\nNumber of fraud and non-fraud transactions:")
print(dataset['fraud'].value_counts())
```

```
Number of fraud and non-fraud transactions:
fraud
0.0    912597
1.0     87403
Name: count, dtype: int64
```

In [ ]:
```
# visualizing the number of fraud and non-fraud transactions
fraud_data = dataset[dataset['fraud'] == 1]
non_fraud_data = dataset[dataset['fraud'] == 0]

# calculating the minimum number of fraud or non-fraud transactions
min_instances = min(len(fraud_data), len(non_fraud_data))
```

In [ ]:
```
# creating a subset of the data with equal number of fraud and non-fraud transactions
fraud_subset = fraud_data.sample(n=20000, random_state=42)
non_fraud_subset = non_fraud_data.sample(n=20000, random_state=42)
```

In [ ]:
```
# concatenating the fraud and non-fraud subsets
small_dataset = pd.concat([fraud_subset, non_fraud_subset])
```

In [ ]:
```
# scaling the dataset
sc = StandardScaler()
sc.fit(small_dataset.drop('fraud', axis=1))
```

Out[ ]:
```
▾ StandardScaler
StandardScaler()
```

In [ ]:
```
# splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(small_dataset.drop(
    'fraud', axis=1), small_dataset['fraud'], test_size=0.2, random_state=42)
```

In [ ]:
```
# training the model using svm and changing the kernel to poly
classifier = SVC(kernel='linear', random_state=0)
# classifier = SVC(kernel='linear', random_state=0)
classifier.fit(X_train, y_train)
```

Out[ ]:
```
SVC

SVC(kernel='linear', random_state=0)
```

In [ ]:
```
# predicting the test set results
y_pred = classifier.predict(X_test)
```

In [ ]:
```
# calculating the metrics
print("\nConfusion Matrix:")
cm = confusion_matrix(y_test, y_pred)
print(cm)

print("\nConfusion Matrix Display:")
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()

print("\nAccuracy Score:")
print(accuracy_score(y_test, y_pred))

print("\nPrecision Score:")
print(precision_score(y_test, y_pred))

print("\nRecall Score:")
print(recall_score(y_test, y_pred))

print("\nF1 Score:")
print(f1_score(y_test, y_pred))

print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Confusion Matrix:
[[3694  292]
 [ 134 3880]]

Confusion Matrix Display:

Accuracy Score:
0.94675

Precision Score:
0.9300095877277086

Recall Score:
0.966616841056303

F1 Score:
0.9479599315905204

Classification Report:
              precision    recall  f1-score   support

         0.0       0.96      0.93      0.95      3986
         1.0       0.93      0.97      0.95      4014

    accuracy                           0.95      8000
   macro avg       0.95      0.95      0.95      8000
weighted avg       0.95      0.95      0.95      8000
```