Lab Number: 11

Title
Programmatically perform Density Based Clustering

Objective
To implement the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm using Python to identify core points, border points, and noise points in the customer dataset.

IDE/Tools Used
VS Code

Programming Language Used
Python

Theory
Density-based clustering: Density-based clustering is an unsupervised machine learning method that groups data points based on the idea that clusters are dense regions of data separated by sparse regions. This approach can discover clusters of arbitrary shapes and is robust to outliers, which are labeled as noise. A common algorithm used is DBSCAN.

DBSCAN: DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a density-based clustering algorithm that groups together nearby points that are closely packed, while marking outliers as noise. It can identify clusters of arbitrary shapes, unlike algorithms like K-Means which assume spherical clusters. DBSCAN is effective at finding clusters of varying shapes and densities and at identifying outliers, but can be sensitive to its two main parameters: the radius (epsilon) and the minimum number of points required to form a dense region (MinPts).

Key Parameters:
- **ε (eps):** Maximum distance between two samples to consider them neighbors
- **min_samples:** Minimum number of points to form a dense region

Types of Points in DBSCAN:
- **Core Point:** Has ≥ min_samples neighbors within eps
- **Border Point:** Has fewer neighbors but belongs to a core point's neighborhood
- **Noise Point/Outlier:** Does not belong to any cluster

Advantages of DBSCAN

- Detects clusters of arbitrary shapes
- Identifies noise/outliers
- No need to specify number of clusters beforehand

Limitations of DBSCAN

- Choosing good eps and min_samples can be difficult
- Struggles with varying densities

Scaler: A scaler is a preprocessing tool that transforms numerical features so they fall within a specific range or distribution. Scaling is often required for machine learning algorithms, especially those that rely on distances or gradients, to prevent variables with larger numeric ranges from dominating the model.

StandardScaler: It is a specific type of scaler that transforms each feature by subtracting the mean and dividing by the standard deviation. This results in values that follow a standard normal distribution with:

- mean = 0
- standard deviation = 1

Standard scaling ensures that all features contribute proportionally in distance-based algorithms such as K-Means, DBSCAN, K-NN, and PCA.

Implementation

1. Dataset

The dataset used is the same customer_data.csv from previous labs and contains:

Attribute	Description
Age	Age of customer in years
AnnualIncome	Annual income (in thousands)
SpendingScore	Score assigned based on spending behavior

This dataset is suitable for DBSCAN because some points are dense clusters while others are sparse, allowing DBSCAN to identify noise/outliers. It is saved in the same location as the python file in csv format.

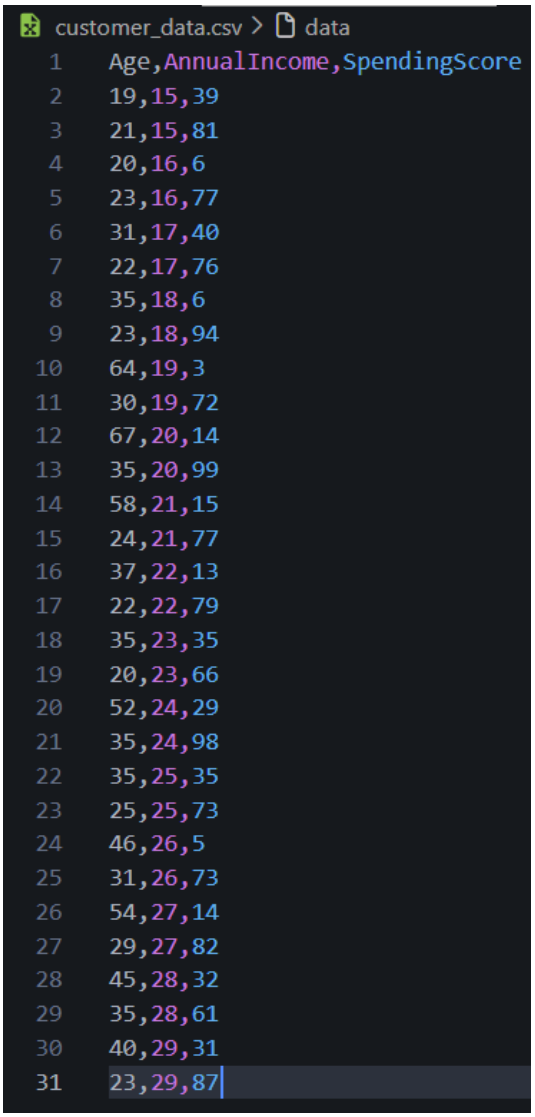

Figure 1: Visualization of the dataset used

2. Code

The python code to implement the DBSCAN is given below:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import DBSCAN
import matplotlib.pyplot as plt

# Loading the dataset
df = pd.read_csv("./customer_data.csv");

# Select the columns from the csv
X = df[["Age", "AnnualIncome", "SpendingScore"]]

# Standardize the data
scaler = StandardScaler();
X_scaled = scaler.fit_transform(X)

# Applying the DBSCAN Algo
algo = DBSCAN(eps=0.7, min_samples=3)
labels = algo.fit_predict(X_scaled)

# Adding the labels for cluster to the dataframe
df["Clusters"] = labels

# Displaying the points, and the cluster they are in
for cluster_id in sorted(df['Clusters'].unique()):
    print(f"\nCluster {cluster_id}:")
    print(df[df['Clusters'] == cluster_id])

# Visualizing using the scatter plot
plt.figure(figsize=(10, 7))
scatter = plt.scatter(df['AnnualIncome'], df['SpendingScore'],
c=df['Clusters'], s=70)

# Add text labels to each point
for i in range(len(df)):
    plt.text(df['AnnualIncome'][i] + 0.1,
             df['SpendingScore'][i] + 0.5,
             str(df['Clusters'][i]),
             fontsize=9)

plt.xlabel("Annual Income")
plt.ylabel("Spending Score")
plt.title("DBSCAN Clustering (with cluster labels)")
plt.show()
```

Output of the code:

```
sayuj@Sayujya MINGW64 ~/OneDrive/Desktop/BIM054/BIM_7th/IT 274_Data Warehousing and Data Mining/Labs/Lab 11 DBSCAN
$ python -u "c:\Users\sayuj\OneDrive\Desktop\BIM054\BIM_7th\IT 274_Data Warehousing and Data Mining\Labs\Lab 11 DBSCAN\dbscan.py"
Cluster -1:
    Age  AnnualIncome  SpendingScore  Clusters
0    19            15             39        -1
2    20            16              6        -1
4    31            17             40        -1
6    35            18              6        -1
8    64            19              3        -1
10   67            20             14        -1
11   35            20             99        -1
12   58            21             15        -1
14   37            22             13        -1
16   35            23             35        -1
18   52            24             29        -1
19   35            24             98        -1
20   35            25             35        -1
22   46            26              5        -1
24   54            27             14        -1
26   45            28             32        -1
28   40            29             31        -1

Cluster 0:
   Age  AnnualIncome  SpendingScore  Clusters
1   21            15             81         0
3   23            16             77         0
5   22            17             76         0
7   23            18             94         0

Cluster 1:
    Age  AnnualIncome  SpendingScore  Clusters
9    30            19             72         1
13   24            21             77         1
15   22            22             79         1
17   20            23             66         1
21   25            25             73         1
23   31            26             73         1
25   29            27             82         1
27   35            28             61         1
29   23            29             87         1
```

Figure 2: Output in the console

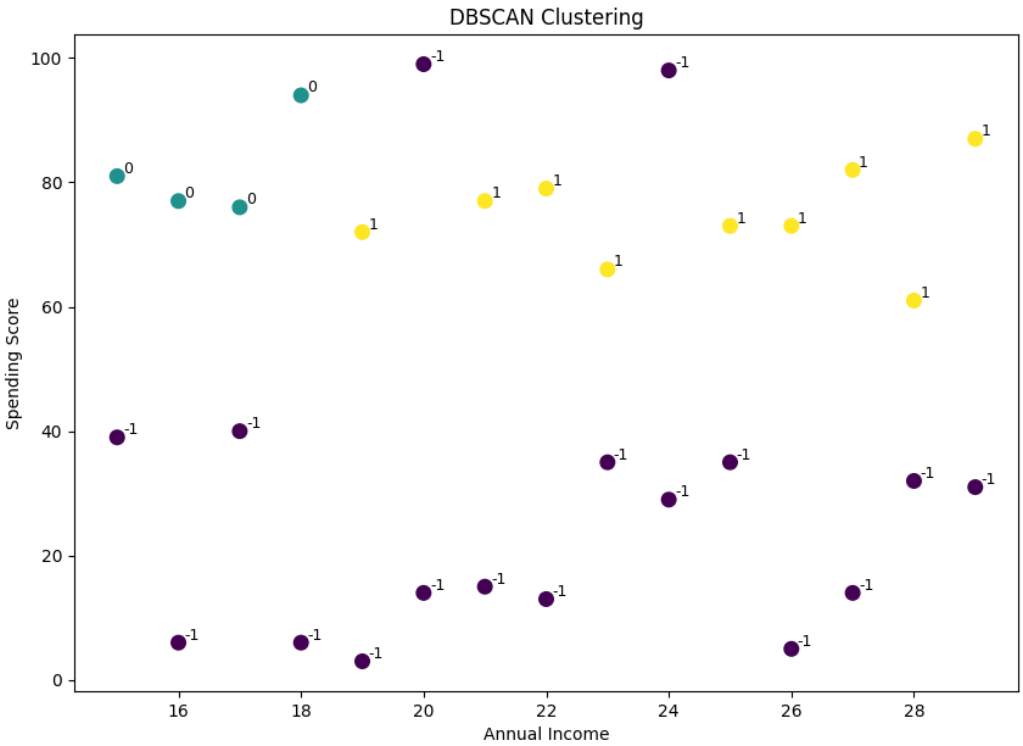

Figure 3: Scatter plot showing the clusters

Discussion

The DBSCAN clustering algorithm was applied to the customer dataset using standardized features (Age, AnnualIncome, SpendingScore). The output produced three groups: Cluster 0, Cluster 1, and Cluster –1, where Cluster –1 represents noise or outliers according to DBSCAN's density-based logic.

Cluster 0 consists of four customers who have relatively low age, low annual income, and high spending scores. These individuals represent high-spending customers despite lower income levels, forming a dense region that DBSCAN identifies as a valid cluster.

Cluster 1 contains customers with slightly higher income but consistently high spending scores, forming a second dense region. These points are similar in terms of spending behavior and lie within a moderate density neighborhood.

Cluster –1 includes the majority of points (17 instances) that DBSCAN considers noise, meaning these customers do not lie within a dense neighborhood of either cluster. This typically happens when the dataset contains scattered points or when the chosen eps/min_samples values result in strict density thresholds. The noise points in this output show very mixed spending behavior and incomes, explaining why DBSCAN did not cluster them.

Including a scatter plot with cluster labels further made the cluster structure visually interpretable. Points from Cluster 0 and Cluster 1 appear in well-formed dense regions, while outliers are distributed irregularly across the plot.

Further, StandardScaler was applied before performing DBSCAN because the algorithm is distance-based. DBSCAN uses Euclidean distance internally to determine whether points fall inside an ε-neighborhood. Without scaling, variables with larger numerical ranges, such as Annual Income or Spending Score, would dominate the distance calculation and distort cluster formation. By standardizing all features to have mean 0 and standard deviation 1, each variable contributes equally, resulting in more meaningful and balanced clusters. Therefore, scaling is a critical preprocessing step for DBSCAN on multi-feature numerical datasets.

Conclusion

In this lab, DBSCAN was applied to the standardized customer dataset, resulting in two meaningful clusters and a large group of outliers. The clusters captured customers with similar income spending behavior, demonstrating DBSCAN's ability to detect dense regions of similar points. Meanwhile, the algorithm labeled scattered or inconsistent data points as noise, reflecting its strength in handling irregular datasets.