Department of
Computer Science and Engineering

Title: Implementation of DBSCAN Clustering Algorithm

Data Mining Lab

CSE 436

Green University of Bangladesh

1 Objective(s)

- To understand the concept of Density-Based Spatial Clustering of Applications with Noise (DBSCAN)
- To implement the DBSCAN algorithm using Python on a given dataset
- To analyze how DBSCAN identifies clusters and detects outliers

2 Problem analysis

2.1 Overview of DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a density-based clustering algorithm that groups together points that are closely packed (points with many nearby neighbors), marking as outliers points that lie alone in low-density regions.

2.2 Key Concepts

2.2.1 Density-Based Clustering

- Clusters are dense regions in the data space, separated by regions of lower density
- A cluster is defined as a maximal set of density-connected points

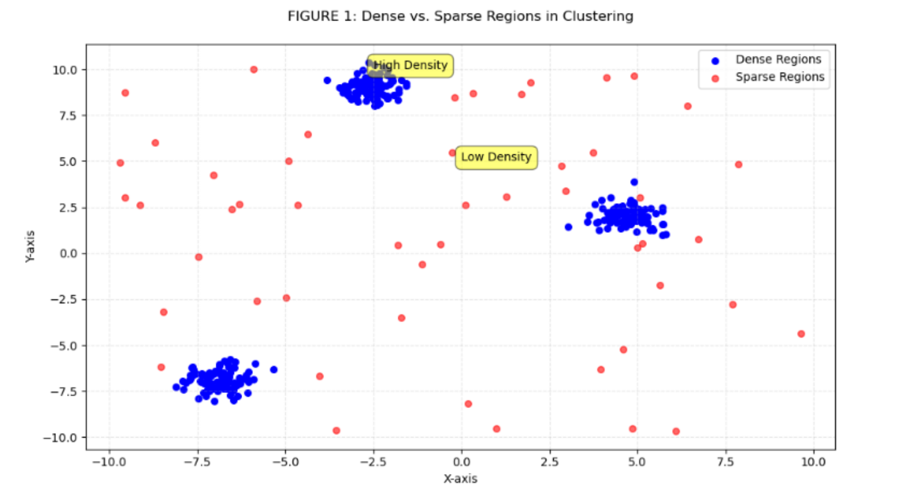


Figure 1: Dense vs. Sparse Regions in Clustering: Blue dots represent dense regions (clusters), while red dots represent sparse regions. High-density clusters are clearly separable, and low-density areas are scattered noise points.

2.2.2 Important Parameters

- ϵ (eps): The radius of the neighborhood around a point
- MinPts (min_samples): The minimum number of points required to form a dense region

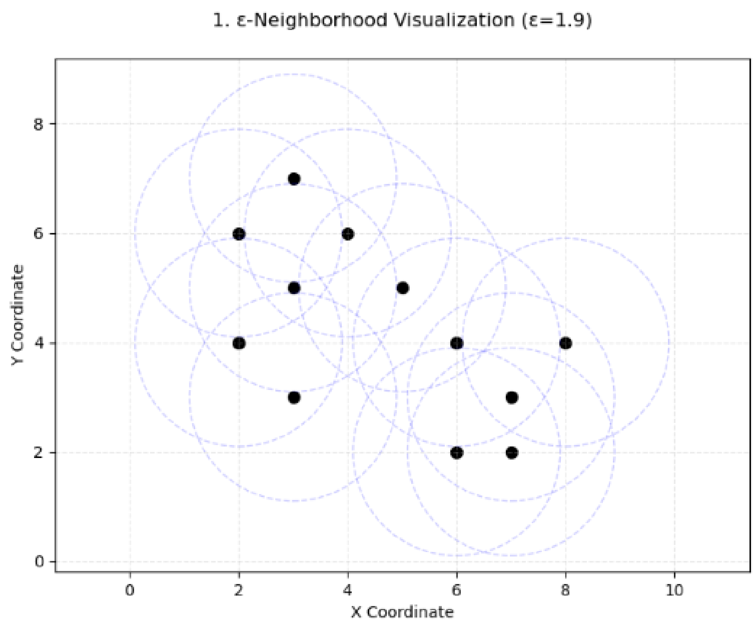


Figure 2: ϵ-Neighborhood Visualization for $\epsilon = 1.9$: Each point is surrounded by a circle of radius ϵ, showing which points fall into each other's neighborhoods—an essential concept in DBSCAN clustering.

2.2.3 Types of Points

- **Core Point**: A point that has at least MinPts within its ϵ-neighborhood
- **Border Point**: A point that has fewer than MinPts within its ϵ-neighborhood but is reachable from a core point
- **Noise Point**: A point that is neither a core point nor a border point

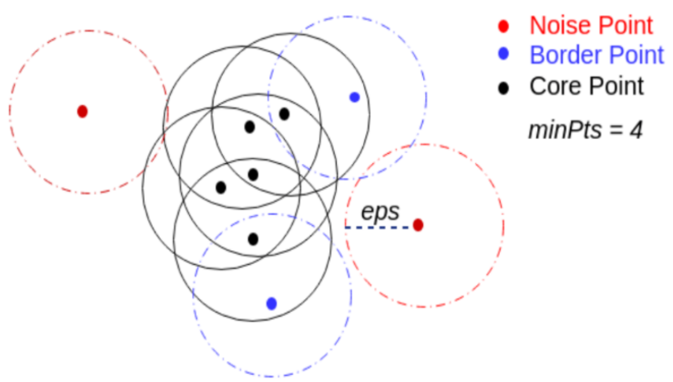


Figure 3: DBSCAN concepts showing core, border and noise points.

2.3 Advantages and Disadvantages

2.3.1 Advantages:

- Does not require specifying the number of clusters
- Can find clusters of arbitrary shape
- Robust to outliers

2.3.2 Disadvantages:

- Sensitive to parameters ϵ and MinPts
- Struggles with clusters of varying densities

2.3.3 Reachability and Connectivity

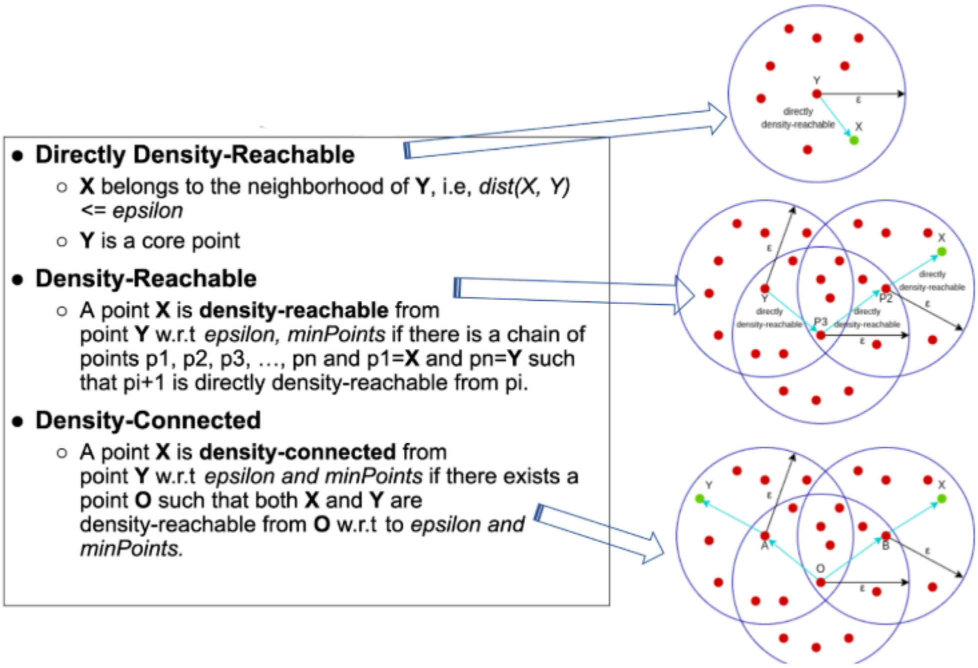


Figure 4: Core Concepts in DBSCAN: Visual explanation of Directly Density-Reachable, Density-Reachable, and Density-Connected points based on ϵ and `minPts`. These concepts define how DBSCAN forms clusters from core points and expands through reachable points.

3 Algorithm

Algorithm 1: DBSCAN Algorithm

Input: D: dataset, eps: radius, MinPts: minimum points
Output: Set of clusters

START
Step 1: Initialize all points as unvisited
Step 2: **for** *each point p in D* **do**
    if *p is visited* **then**
        continue to next point
    end
    Mark p as visited
    N = getNeighbors(p, eps)
    if *sizeOf(N) < MinPts* **then**
        Mark p as NOISE
    end
    else
        C = new Cluster()
        expandCluster(p, N, C, eps, MinPts)
        Add C to set of clusters
    end
end
END

4 Implementation in Python

4.1 Input Dataset

The dataset `points_data.csv` contains 2D points with coordinates (X, Y), represented in the table below:

Table 1: 'points_data.csv' Data Samples

Point	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12
X	3	4	5	6	7	6	7	8	3	2	3	2
Y	7	6	5	4	3	2	2	4	3	6	5	4

4.2 Implementation

The following implementation uses the scikit-learn library to perform DBSCAN clustering on a 2D dataset of points.

```python
# Author: Ataullha
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.cluster import DBSCAN

# Load the dataset
df = pd.read_csv("points_data.csv")
X = df[['X', 'Y']].values  # Extract features

# Apply DBSCAN
eps_value = 1.9  # epsilon/ radius
min_samples_value = 4  # MinPts
clustering = DBSCAN(eps=eps_value, min_samples=min_samples_value)
labels = clustering.fit_predict(X)  # Get cluster labels
df['Cluster'] = labels  # Add labels to DataFrame

print(df)

# Plot the clusters
plt.figure(figsize=(8, 6))
# noise indicated as -1, clusters indicated as 0, 1, ... and so on
unique_labels = set(labels)  # e.g., {-1, 0, 1, ...} (noise + n clusters)
colors = plt.cm.get_cmap("tab10", len(unique_labels))  # Get n colors

for label in unique_labels:
    if label == -1:  # Noise points
        cluster_points = X[labels == label]
        plt.scatter(cluster_points[:, 0], cluster_points[:, 1],
                    label='Noise', color='black', edgecolor='k')
    else:  # Regular clusters
        cluster_points = X[labels == label]
        plt.scatter(cluster_points[:, 0], cluster_points[:, 1],
                    label=f'Cluster {label}', edgecolor='k')

plt.title("DBSCAN Clustering")
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
plt.legend()
plt.show()
```

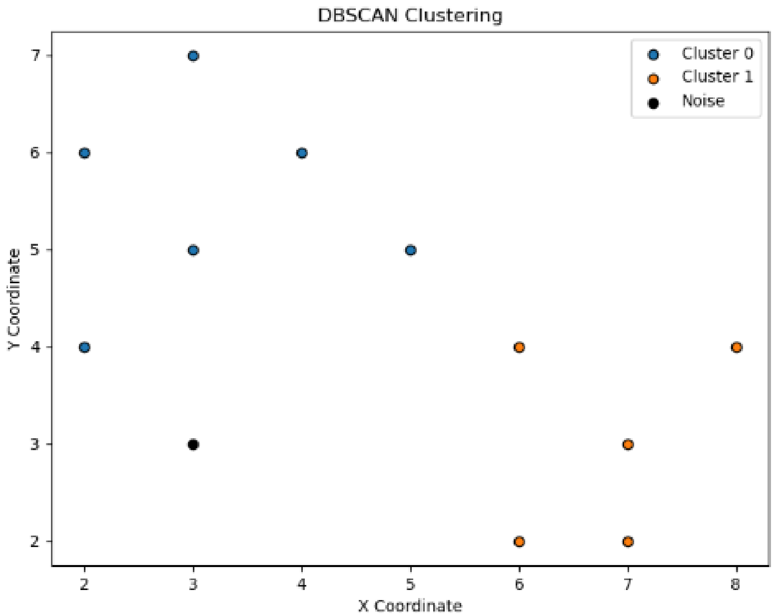
Listing 1: DBSCAN Clustering in Python.

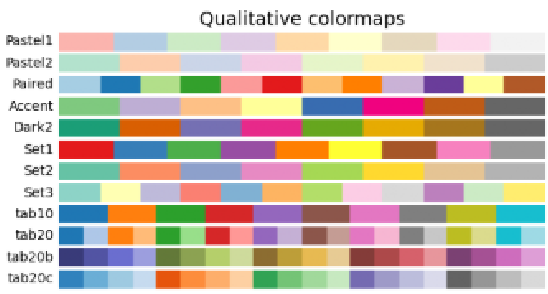
4.3 Output

The output shows the cluster assignments for each point, with -1 indicating noise points:

Table 2: Sample Output

Index	Point	X	Y	Cluster
0	P1	3	7	0
1	P2	4	6	0
2	P3	5	5	0
3	P4	6	4	1
4	P5	7	3	1
5	P6	6	2	1
6	P7	7	2	1
7	P8	8	4	1
8	P9	3	3	-1
9	P10	2	6	0
10	P11	3	5	0
11	P12	2	4	0





Example of "tab10" colormap (Source: Matplotlib Docs)

Figure 5: Visualization of DBSCAN clustering results (First Figure) and color-map explanation (Second Figure).

5 Analysis

5.1 Effect of Parameters

- ϵ (eps):
  - Too small: Many small clusters and noise points
  - Too large: Fewer, larger clusters, may merge separate clusters
- MinPts:
  - Too small: Noise points may be classified as clusters
  - Too large: Sparse clusters may be classified as noise

5.2 Comparison with K-Means

Table 3: Comparison between DBSCAN and K-Means

Feature	**DBSCAN**	**K-Means**
Cluster Shape	Arbitrary	Spherical
Outliers	Detects noise	Sensitive to outliers
Parameters	ϵ, MinPts	Number of clusters (k)

6 Discussion & Conclusion

In this experiment, we implemented the DBSCAN clustering algorithm using Python. The algorithm successfully identified clusters of arbitrary shape and detected noise points in the dataset. The key advantage of DBSCAN is its ability to find clusters without requiring the number of clusters to be specified beforehand. However, the quality of clustering is highly dependent on the choice of ϵ and MinPts parameters.

7 Lab Task

- Apply DBSCAN on a customer segmentation dataset
- Compare results with K-Means clustering
- Vary ϵ from 0.5 to 2.5 (steps of 0.5) and MinPts from 3 to 10
- Observe and document changes in clustering results

8 Lab Exercise (Submit as a report)

Write a report documenting your findings from the lab tasks, including:

- The effect of changing ϵ and MinPts parameters
- Comparison between DBSCAN and K-Means results
- Challenges faced during implementation

9 Policy

Copying from internet, classmate, seniors, or from any other source is strongly prohibited. 100% marks will be *deducted* if any such copying is detected.