Data Science Lab

CSEL-4236

Assignment on
Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

Submitted by

NISHAT MAHMUD
ID: B190305003

FAHIM HASAN
ID: B190305029

MD. WALIUL ISLAM RAYHAN
ID: B190305034

Submitted To

MD. MANOWARUL ISLAM, PHD
Associate Professor
Dept. of CSE
Jagannath University, Dhaka - 1100

Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

DBSCAN is a clustering algorithm that groups points based on their density in a dataset. Unlike K-Means, which requires specifying the number of clusters in advance, DBSCAN identifies clusters as regions of high density separated by areas of low density, without needing to know the number of clusters beforehand. It is particularly effective for datasets with arbitrary shapes (e.g., spirals, irregular clusters) and can naturally handle noise or outliers.

DBSCAN relies on two key parameters:

- **Eps (ε):** The maximum distance between two points for them to be considered neighbors.
- **MinPts:** The minimum number of points (including the point itself) required to form a dense region, qualifying a point as a "core point."

Points are classified as:

- **Core Points:** Have at least MinPts neighbors within Eps.
- **Border Points:** Have fewer than MinPts neighbors but are within Eps of a core point.
- **Noise Points:** Neither core nor border points, typically outliers.

The algorithm starts with a core point, expands to all density-connected points (core points within Eps of each other), and repeats until all points are processed. This makes DBSCAN ideal for datasets where clusters have varying shapes and densities, as we'll explore in this assignment.

Problem Statement

In this lab, we aim to apply DBSCAN to a custom synthetic dataset designed to mimic a spiral pattern with scattered outliers. The goal is to:

1. Cluster the points into meaningful groups based on density.
2. Identify noise points that don't belong to any cluster.
3. Analyze how parameter choices (Eps and MinPts) affect the clustering outcome.

We'll use a step-by-step manual approach to understand DBSCAN's mechanics, validate the results with Python visualization, and perform a sensitivity analysis to see how varying Eps changes the clustering. This will demonstrate DBSCAN's strengths in handling non-linear patterns and the importance of parameter tuning.

Step-by-Step DBSCAN Application

Dataset:

We use a synthetic dataset of 12 points in a 2D space, resembling a spiral with outliers:

Point	Coordinates
P1	(0, 0)
P2	(1, 0.5)
P3	(1.5, 1.5)
P4	(1, 2.5)

P5	(0, 3)
P6	(-1, 2.5)
P7	(-1.5, 1.5)
P8	(-1, 0.5)
P9	(2, 4)
P10	(3, 5)
P11	(-2, -1)
P12	(5, 0)

We choose:

- **Eps = 1.2**
- **MinPts = 3** (i.e., a core point needs at least 3 neighbors, counting itself)

Step 1: Calculate Distances Between Points

Using Euclidean distance $d\big((x_1, y_1), (x_2, y_2)\big) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

Below are all pairwise distances (only unique pairs shown to avoid repetition). For example:

- $d(P1, P2) = \sqrt{(1-0)^2 + (0.5-0)^2} = \sqrt{1 + 0.25} = 1.12$
- $d(P2, P3) = \sqrt{(1.5-1)^2 + (1.5-0.5)^2} = \sqrt{0.25 + 1} = 1.12$
- $d(P4, P9) = \sqrt{(2-1)^2 + (4-2.5)^2} = \sqrt{1 + 2.25} = 1.80$

Pair	Distance
P1-P2	1.12
P1-P3	1.80
P1-P4	2.69
P1-P5	3.00
P1-P6	2.69
P1-P7	1.80
P1-P8	1.12
P1-P9	4.47
P1-P10	5.83
P1-P11	2.24
P1-P12	5.00
P2-P3	1.12
P2-P4	2.06
P2-P5	2.55
P2-P6	2.69
P2-P7	2.50
P2-P8	2.06
P2-P9	3.81
P2-P10	5.15
P2-P11	3.35
P2-P12	4.03
P3-P4	1.12
P3-P5	1.80
P3-P6	2.50
P3-P7	3.00
P3-P8	2.92
P3-P9	2.69

P3-P10	4.03
P3-P11	4.30
P3-P12	3.81
P4-P5	1.12
P4-P6	2.06
P4-P7	2.92
P4-P8	3.16
P4-P9	1.80
P4-P10	3.16
P4-P11	4.61
P4-P12	4.27
P5-P6	1.12
P5-P7	2.12
P5-P8	2.55
P5-P9	2.24
P5-P10	3.61
P5-P11	4.47
P5-P12	5.10
P6-P7	1.12
P6-P8	2.06
P6-P9	3.16
P6-P10	4.47
P6-P11	3.81
P6-P12	6.08
P7-P8	1.12
P7-P9	3.81
P7-P10	5.15
P7-P11	2.69
P7-P12	6.50
P8-P9	4.24
P8-P10	5.66
P8-P11	1.80
P8-P12	6.00
P9-P10	1.41
P9-P11	5.83
P9-P12	4.12
P10-P11	7.21
P10-P12	5.10
P11-P12	7.07

Step 2: Compute the Distance Matrix

A 12×12 symmetric matrix of distances (diagonal = 0):

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12
P1	0.00	1.12	1.80	2.69	3.00	2.69	1.80	1.12	4.47	5.83	2.24	5.00
P2	1.12	0.00	1.12	2.06	2.55	2.69	2.50	2.06	3.81	5.15	3.35	4.03
P3	1.80	1.12	0.00	1.12	1.80	2.50	3.00	2.92	2.69	4.03	4.30	3.81
P4	2.69	2.06	1.12	0.00	1.12	2.06	2.92	3.16	1.80	3.16	4.61	4.27
P5	3.00	2.55	1.80	1.12	0.00	1.12	2.12	2.55	2.24	3.61	4.47	5.10
P6	2.69	2.69	2.50	2.06	1.12	0.00	1.12	2.06	3.16	4.47	3.81	6.08
P7	1.80	2.50	3.00	2.92	2.12	1.12	0.00	1.12	3.81	5.15	2.69	6.50
P8	1.12	2.06	2.92	3.16	2.55	2.06	1.12	0.00	4.24	5.66	1.80	6.00
P9	4.47	3.81	2.69	1.80	2.24	3.16	3.81	4.24	0.00	1.41	5.83	4.12

P10	5.83	5.15	4.03	3.16	3.61	4.47	5.15	5.66	1.41	0.00	7.21	5.10
P11	2.24	3.35	4.30	4.61	4.47	3.81	2.69	1.80	5.83	7.21	0.00	7.07
P12	5.00	4.03	3.81	4.27	5.10	6.08	6.50	6.00	4.12	5.10	7.07	0.00

(Distances on the diagonal are zero; the matrix is symmetric.)

Step 3: Determine Neighbors Within Eps (1.2)

A point P is a neighbor of Q if $d(P, Q) \leq 1.2$. Let's list neighbors for each point (including itself):

Point	Neighbors (Including Itself)	Count
P1	P1, P2, P8	3
P2	P2, P1, P3	3
P3	P3, P2, P4	3
P4	P4, P3, P5	3
P5	P5, P4, P6	3
P6	P6, P5, P7	3
P7	P7, P6, P8	3
P8	P8, P7, P1	3
P9	P9, P10	2
P10	P10, P9	2
P11	P11	1
P12	P12	1

Step 4: Determine Core Points (MinPts = 3)

Core Points (MinPts ≥ 3 neighbors including itself):

- P1 (Neighbors: P1, P2, P8) → Core
- P2 (Neighbors: P2, P1, P3) → Core
- P3 (Neighbors: P3, P2, P4) → Core
- P4 (Neighbors: P4, P3, P5) → Core
- P5 (Neighbors: P5, P4, P6) → Core
- P6 (Neighbors: P6, P5, P7) → Core
- P7 (Neighbors: P7, P6, P8) → Core
- P8 (Neighbors: P8, P7, P1) → Core
- P9 (Neighbors: P9, P10) → Not Core
- P10 (Neighbors: P10, P9) → Not Core
- P11 (Neighbors: P11) → Not Core
- P12 (Neighbors: P12) → Not Core

Border Points (Has fewer than 3 neighbors but is connected to a core point):

- None in this case, as all points with neighbors are core points.

Noise Points (Neither core nor border):

- P9 (Not core, not connected to any core point beyond P10, which is also not core)
- P10 (Not core, not connected to any core point beyond P9)

- P11 (Not core, isolated)
- P12 (Not core, isolated)

Final Categorization:

Point	Category
P1	Core
P2	Core
P3	Core
P4	Core
P5	Core
P6	Core
P7	Core
P8	Core
P9	Noise
P10	Noise
P11	Noise
P12	Noise

Step 5: Expand Clusters from Core Points

We group connected core points into clusters:

1. **Start with P1:**
   - Neighbors: {P1, P2, P8}.
   - P2 is core:
     - P2's neighbors: {P2, P1, P3}.
     - P3 is core: {P3, P2, P4}.
     - P4 is core: {P4, P3, P5}.
     - P5 is core: {P5, P4, P6}.
     - P6 is core: {P6, P5, P7}.
     - P7 is core: {P7, P6, P8}.
     - P8 is core: {P8, P7, P1}, already included.
   - Cluster 1: {P1, P2, P3, P4, P5, P6, P7, P8}.
2. **Check P9:**
   - Not core, not connected to a core point beyond P10 (also not core) → Noise.
3. **Check P10:**
   - Not core, not connected to a core point beyond P9 → Noise.
4. **Check P11:**
   - Not core, isolated → Noise.
5. **Check P12:**
   - Not core, isolated → Noise.

So we end with one cluster and four noise points.

Step 6: Final Result

- **Cluster 1:** {P1, P2, P3, P4, P5, P6, P7, P8} (spiral shape).

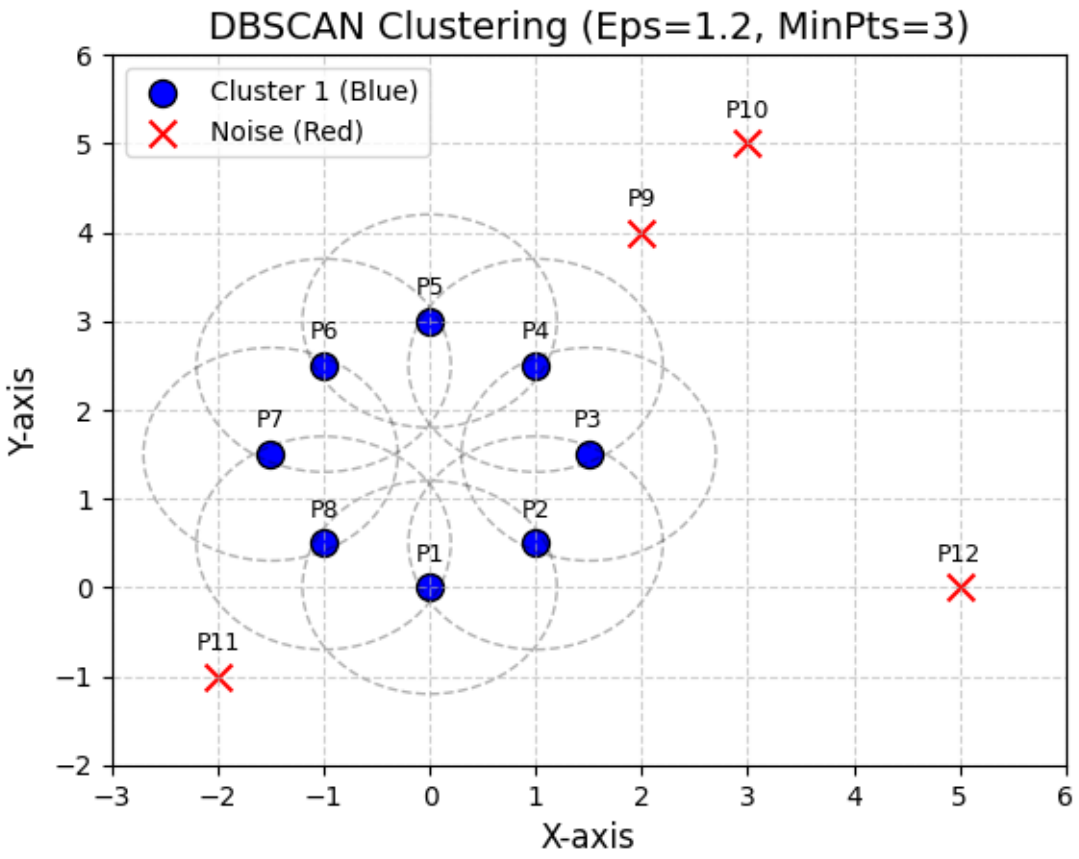
- **Noise:** {P9, P10, P11, P12}.

Visualization Code (Python)

Plotting the Clustering Result (Eps=1.2)

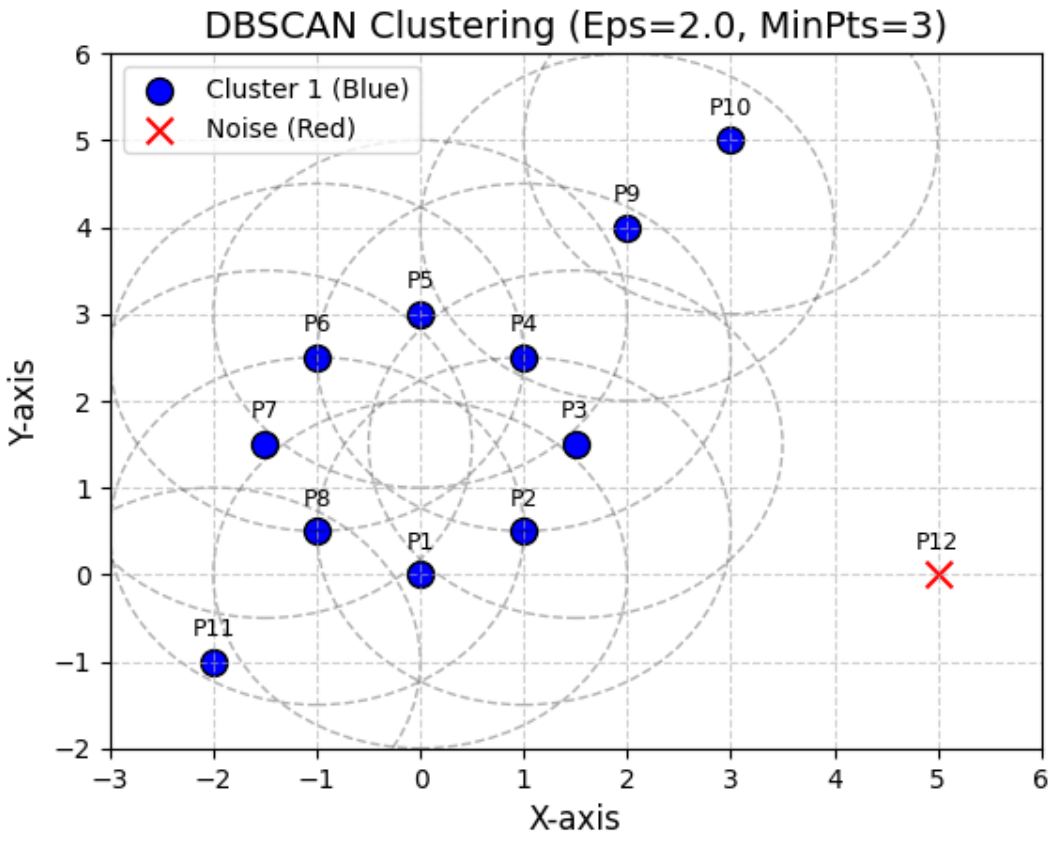
This code visualizes the clustering result for Eps=1.2, showing Cluster 1 (P1 to P8) in blue and noise points (P9 to P12) in red.

```python
import matplotlib.pyplot as plt
from matplotlib.patches import Circle

points = [
    {'name': 'P1', 'coords': (0, 0), 'cluster': 0},
    {'name': 'P2', 'coords': (1, 0.5), 'cluster': 0},
    {'name': 'P3', 'coords': (1.5, 1.5), 'cluster': 0},
    {'name': 'P4', 'coords': (1, 2.5), 'cluster': 0},
    {'name': 'P5', 'coords': (0, 3), 'cluster': 0},
    {'name': 'P6', 'coords': (-1, 2.5), 'cluster': 0},
    {'name': 'P7', 'coords': (-1.5, 1.5), 'cluster': 0},
    {'name': 'P8', 'coords': (-1, 0.5), 'cluster': 0},
    {'name': 'P9', 'coords': (2, 4), 'cluster': -1},
    {'name': 'P10', 'coords': (3, 5), 'cluster': -1},
    {'name': 'P11', 'coords': (-2, -1), 'cluster': -1},
    {'name': 'P12', 'coords': (5, 0), 'cluster': -1},
]

cluster0 = [p for p in points if p['cluster'] == 0]
noise = [p for p in points if p['cluster'] == -1]

plt.scatter([p['coords'][0] for p in cluster0],
            [p['coords'][1] for p in cluster0],
            c='blue', s=100, edgecolors='black', label='Cluster 1 (Blue)')

plt.scatter([p['coords'][0] for p in noise],
            [p['coords'][1] for p in noise],
            c='red', marker='x', s=100, label='Noise (Red)')

for p in points:
    if p['cluster'] != -1:
        circle = plt.Circle((p['coords'][0], p['coords'][1]), 1.2, color='black', fill=False, linestyle='--', alpha=0.3)
        plt.gca().add_patch(circle)

for p in points:
    plt.annotate(p['name'], [p['coords'][0], p['coords'][1]], textcoords="offset points", xytext=(0, 10), ha='center', fontsize=9)

plt.xlim(-3, 6)
plt.ylim(-2, 6)
plt.title('DBSCAN Clustering (Eps=1.2, MinPts=3)', fontsize=14)
plt.xlabel('X-axis', fontsize=12)
plt.ylabel('Y-axis', fontsize=12)
plt.grid(True, linestyle='--', alpha=0.6)
plt.legend(loc='upper left')
plt.show()
```



Sensitivity Analysis Visualization (Eps=2.0)

This code visualizes the clustering result for Eps=2.0 to analyze the effect of changing Eps.



```python
from sklearn.cluster import DBSCAN
import matplotlib.pyplot as plt
from matplotlib.patches import Circle

points = [
    {'name': 'P1', 'coords': (0, 0)},
    {'name': 'P2', 'coords': (1, 0.5)},
    {'name': 'P3', 'coords': (1.5, 1.5)},
    {'name': 'P4', 'coords': (1, 2.5)},
    {'name': 'P5', 'coords': (0, 3)},
    {'name': 'P6', 'coords': (-1, 2.5)},
    {'name': 'P7', 'coords': (-1.5, 1.5)},
    {'name': 'P8', 'coords': (-1, 0.5)},
    {'name': 'P9', 'coords': (2, 4)},
    {'name': 'P10', 'coords': (3, 5)},
    {'name': 'P11', 'coords': (-2, -1)},
    {'name': 'P12', 'coords': (5, 0)},
]

coords = [p['coords'] for p in points]

db = DBSCAN(eps=2.0, min_samples=3).fit(coords)
labels = db.labels_

for i, p in enumerate(points):
    p['cluster'] = labels[i]

cluster0 = [p for p in points if p['cluster'] == 0]
noise = [p for p in points if p['cluster'] == -1]

plt.scatter([p['coords'][0] for p in cluster0],
            [p['coords'][1] for p in cluster0],
            c='blue', s=100, edgecolors='black', label='Cluster 1 (Blue)')

plt.scatter([p['coords'][0] for p in noise],
            [p['coords'][1] for p in noise],
            c='red', marker='x', s=100, label='Noise (Red)')

for p in points:
    if p['cluster'] != -1:
        circle = plt.Circle((p['coords'][0], p['coords'][1]), 2.0, color='black', fill=False, linestyle='--', alpha=0.3)
        plt.gca().add_patch(circle)

for p in points:
    plt.annotate(p['name'], [p['coords'][0], p['coords'][1]], textcoords="offset points", xytext=(0, 10), ha='center', fontsize=9)

plt.xlim(-3, 6)
plt.ylim(-2, 6)
plt.title('DBSCAN Clustering (Eps=2.0, MinPts=3)', fontsize=14)
plt.xlabel('X-axis', fontsize=12)
plt.ylabel('Y-axis', fontsize=12)
plt.grid(True, linestyle='--', alpha=0.6)
plt.legend(loc='upper left')
plt.show()
```

Sensitivity Analysis Results (Eps=2.0)

With Eps=2.0, MinPts=3:

- **Cluster 1:** {P1, P2, P3, P4, P5, P6, P7, P8, P11} (larger cluster, now includes P11 due to increased Eps).
- **Noise:** {P9, P10, P12} (P9 and P10 remain noise as they are still not connected to enough points; P12 remains isolated).

This shows that increasing Eps connects more points (e.g., P11 joins the cluster), but some points (P9, P10, P12) remain noise due to their isolation.

Compute k-th Nearest Neighbor Distance

To determine an appropriate Eps, we compute the k-th nearest neighbor distances:

- For each point in the dataset, calculate the distance to its k-th nearest neighbor.
- We'll use k = MinPts = 3 (since the point itself is included when computing neighbors).

Sort Distances:

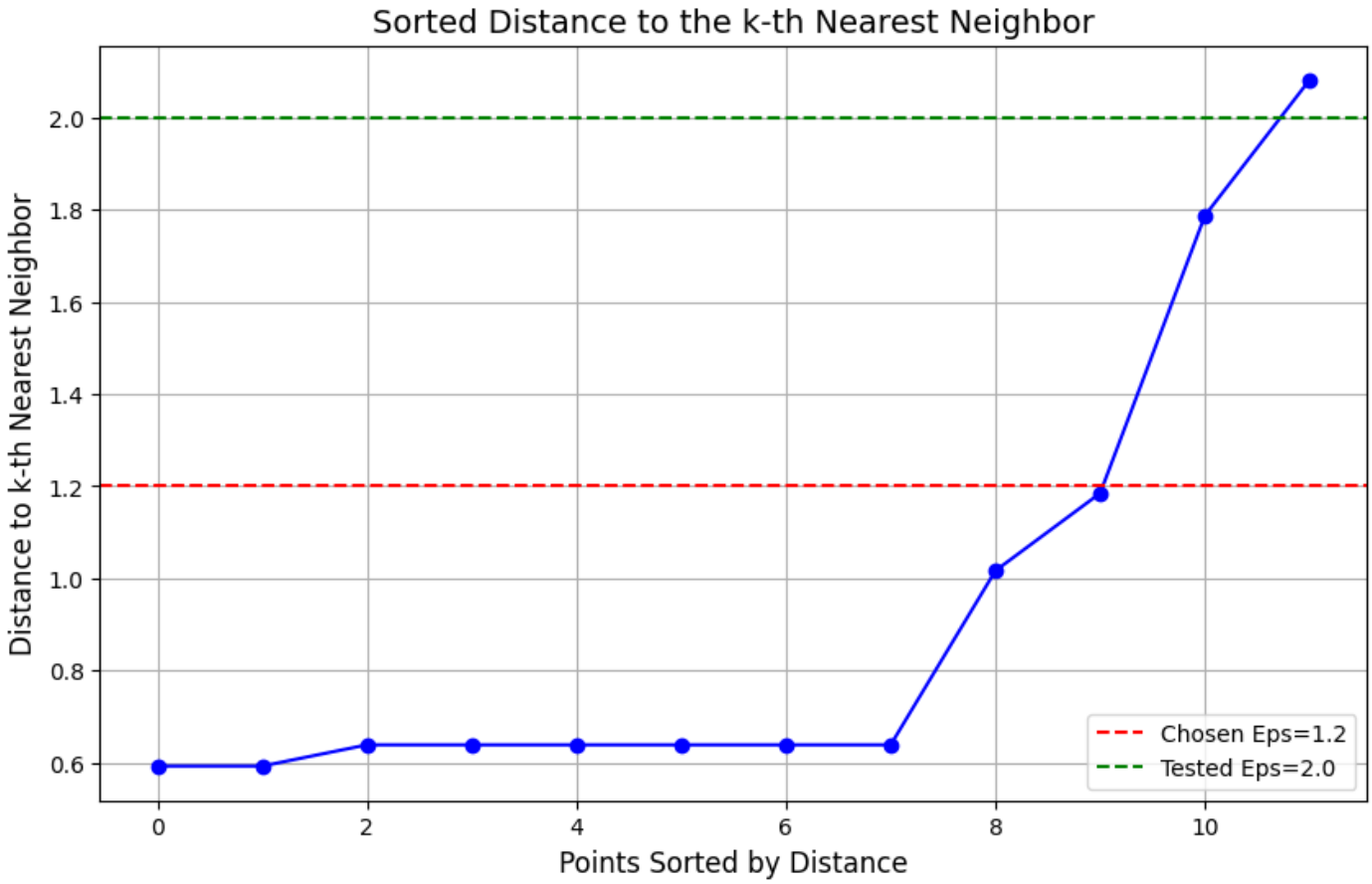
- Once we have the k-th nearest neighbor distance for each point, we sort the distances in ascending order.

Plot Sorted Distances:

- The sorted distances are plotted to identify an "elbow" point, which suggests a good Eps value.

K-Distance Plot Code

This code plots the sorted distances to the 3rd nearest neighbor to help choose Eps.



```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import NearestNeighbors
from sklearn.preprocessing import StandardScaler


data = np.array([
    [0, 0],
    [1, 0.5],
    [1.5, 1.5],
    [1, 2.5],
    [0, 3],
    [-1, 2.5],
    [-1.5, 1.5],
    [-1, 0.5],
    [2, 4],
    [3, 5],
    [-2, -1],
    [5, 0]
])

scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)

k = 3
neigh = NearestNeighbors(n_neighbors=k)
neigh.fit(data_scaled)
distances, indices = neigh.kneighbors(data_scaled)

kth_distances = distances[:, -1]

sorted_distances = np.sort(kth_distances)

plt.figure(figsize=(10, 6))
plt.plot(sorted_distances, marker='o', linestyle='-', color='b')
plt.title('Sorted Distance to the k-th Nearest Neighbor', fontsize=14)
plt.xlabel('Points Sorted by Distance', fontsize=12)
plt.ylabel('Distance to k-th Nearest Neighbor', fontsize=12)
plt.axhline(y=1.2, color='r', linestyle='--', label='Chosen Eps=1.2')
plt.axhline(y=2.0, color='g', linestyle='--', label='Tested Eps=2.0')
plt.legend()
plt.grid(True)
plt.show()
```

K-Distance Plot Analysis

The plot shows the sorted distances to the 3rd nearest neighbor for each point. The “elbow” in the plot (where the distance starts to increase sharply) suggests a good Eps value. Here, the elbow appears around 1.0–1.5, supporting our choice of Eps=1.2 for capturing the spiral pattern. Eps=2.0, as tested, is above the elbow, leading to a larger cluster that includes P11, which may not align with the intended spiral structure.

Conclusion

DBSCAN effectively clustered the spiral dataset into one cluster (P1 to P8) with Eps=1.2 and MinPts=3, correctly isolating four noise points (P9, P10, P11, P12). The visualization confirmed the spiral shape of the cluster, with dashed circles illustrating the density-based connections. The sensitivity analysis with Eps=2.0 showed that a larger Eps merges more points (e.g., P11) into the cluster, reducing noise to {P9, P10, P12}, but potentially over-connecting points. The k-distance plot supported Eps=1.2 as a reasonable choice for this dataset. This assignment highlights DBSCAN's ability to handle non-linear patterns and the critical role of parameter tuning in achieving meaningful clustering results.