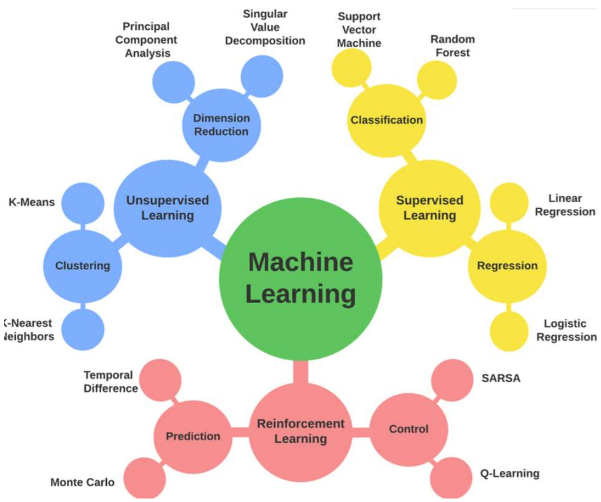
Các phương pháp học máy
Machine learning methods

4 TC: 2 LT – 2 TH

Giảng viên: **Tạ Hoàng Thắng**

tahoangthang@gmail.com

0975399307



PAM

Definitions

PAM (Partitioning Around Medoids or k-medoids) is a clustering algorithm used in data mining and machine learning, particularly for partitioning a dataset into groups or clusters.

- It is a **robust version of the k-means algorithm** and is commonly used for clustering when the **data contains noise or outliers**.
- Instead of using the mean (as in k-means), PAM **uses actual data points as the center of clusters**, which are called medoids.

https://en.wikipedia.org/wiki/K-medoids

https://www.cs.umb.edu/cs738/pam1.pdf

PAM

How it works?

PAM uses a **greedy search** which may not find **the optimal solution**, but it is faster than exhaustive search.

- (BUILD) Initialize: greedily select **k** of the **n** data points as the medoids to minimize the cost.
- Associate each data point to the closest medoid **based-on the dissimilarity values (Manhattan Distance)**.
- (SWAP) While the cost of the configuration decreases:
  - For each medoid *m*, and for each non-medoid data point *o*:
    - Consider the swap of *m* and *o*, and compute the cost change
    - If the cost change is the current best, remember this *m* and *o* combination
  - Perform the best swap of **m_best** and **o_best** if it decreases the cost function. Otherwise, the algorithm terminates.

PAM

Examples

Given 2 points (4, 5) and (7, 8), calculate the Manhattan distance between them.

$$c = \sum_{Ci} \sum_{Pi \in Ci} |Pi - Ci|$$

2. Manhattan Distance:

The formula for Manhattan distance d is:

$$d = |x_2 - x_1| + |y_2 - y_1|$$

For points $(4, 5)$ and $(7, 8)$:

$$d = |7 - 4| + |8 - 5| = 3 + 3 = 6$$

PAM

Examples

Given 2 points (4, 5, 8, 1) and (7, 8, 9, 14), calculate the Manhattan distance between them?

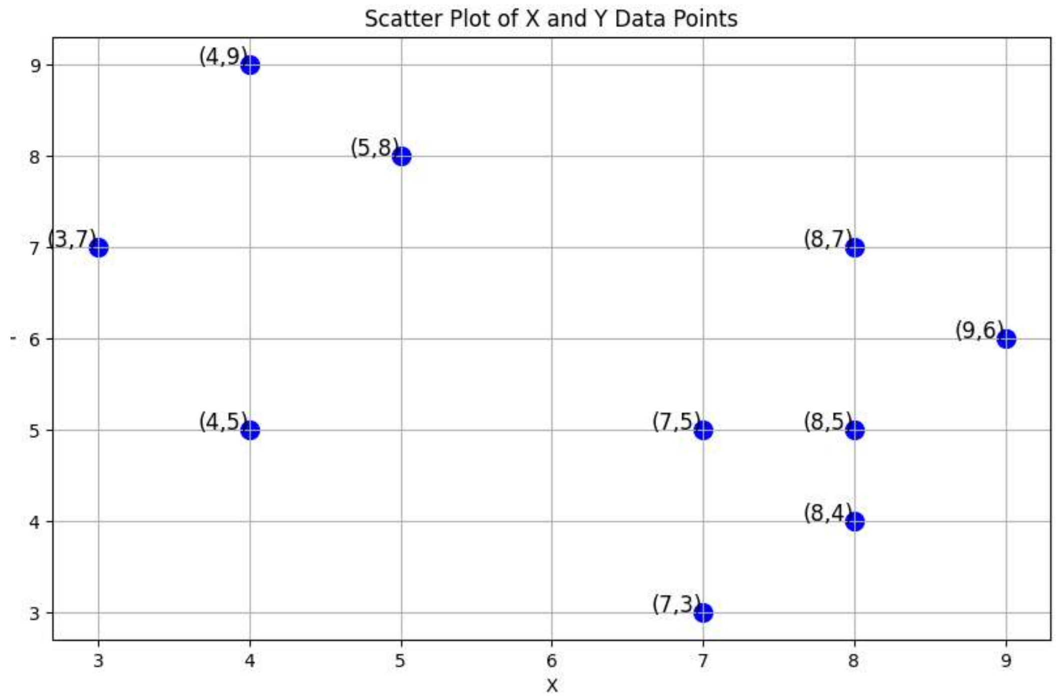
PAM

Examples

https://www.geeksforgeeks.org/ml-k-medoids-clustering-with-example/

Given this table data:

	X	Y
0	8	7
1	3	7
2	4	9
3	9	6
4	8	5
5	5	8
6	7	3
7	8	4
8	7	5
9	4	5



PAM

Examples

Step 1: Let the randomly selected 2 medoids, so select k = 2, and let **C1 (4, 5)** and **C2 (8, 5)** are the two medoids.

Step 2: Calculating cost. The dissimilarity of each non-medoid point with the medoids is calculated and tabulated:

	X	Y	Dissimilarity from C1	Dissimilarity from C2
0	8	7	6	2
1	3	7	3	7
2	4	9	4	8
3	9	6	6	2
4	**8**	**5**	-	-
5	5	8	4	6
6	7	3	5	3
7	8	4	5	1
8	7	5	3	1
9	**4**	**5**	-	-

PAM

Examples

Step 2: Calculating cost. The dissimilarity of each non-medoid point with the medoids is calculated and tabulated:

Each point is assigned to the cluster of that medoid whose dissimilarity is less. Points 1, 2, and 5 go to cluster C1 and 0, 3, 6, 7, 8 go to cluster C2. **The Cost = (3 + 4 + 4) + (3 + 1 + 1 + 2 + 2) = 20**

	X	Y	Dissimilarity from C1	Dissimilarity from C2
0	8	7	6	2
1	3	7	3	7
2	4	9	4	8
3	9	6	6	2
4	8	5	-	-
5	5	8	4	6
6	7	3	5	3
7	8	4	5	1
8	7	5	3	1
9	4	5	-	-

PAM

Examples

Step 3: randomly select one non-medoid point and recalculate the cost. Let the randomly selected point be (8, 4). The dissimilarity of each non-medoid point with the medoids C1 (4, 5) and C2 (8, 4) is calculated and tabulated.

	X	Y	Dissimilarity from C1	Dissimilarity from C2
0	8	7	6	3
1	3	7	3	8
2	4	9	4	9
3	9	6	6	3
4	8	5	4	1
5	5	8	4	7
6	7	3	5	2
7	8	4	-	-
8	7	5	3	2
9	4	5	-	-

PAM

Examples

Step 3: The medoids C1 (4, 5) and C2 (8, 4) is calculated and tabulated. So, points 1, 2, and 5 go to cluster C1 and 0, 3, 6, 7, 8 go to cluster C2. The New cost = (3 + 4 + 4) + (2 + 2 + 1 + 3 + 3) = 22 Swap Cost = New Cost – Previous Cost = 22 – 20 and **2 >0.** **As the swap cost is not less than zero, we undo the swap**. Hence (4, 5) and (8, 5) are the final medoids.

	X	Y	Dissimilarity from C1	Dissimilarity from C2
0	8	7	6	3
1	3	7	3	8
2	4	9	4	9
3	9	6	6	3
4	8	5	4	1
5	5	8	4	7
6	7	3	5	2
7	**8**	**4**	-	-
8	7	5	3	2
9	**4**	**5**	-	-

PAM

Examples

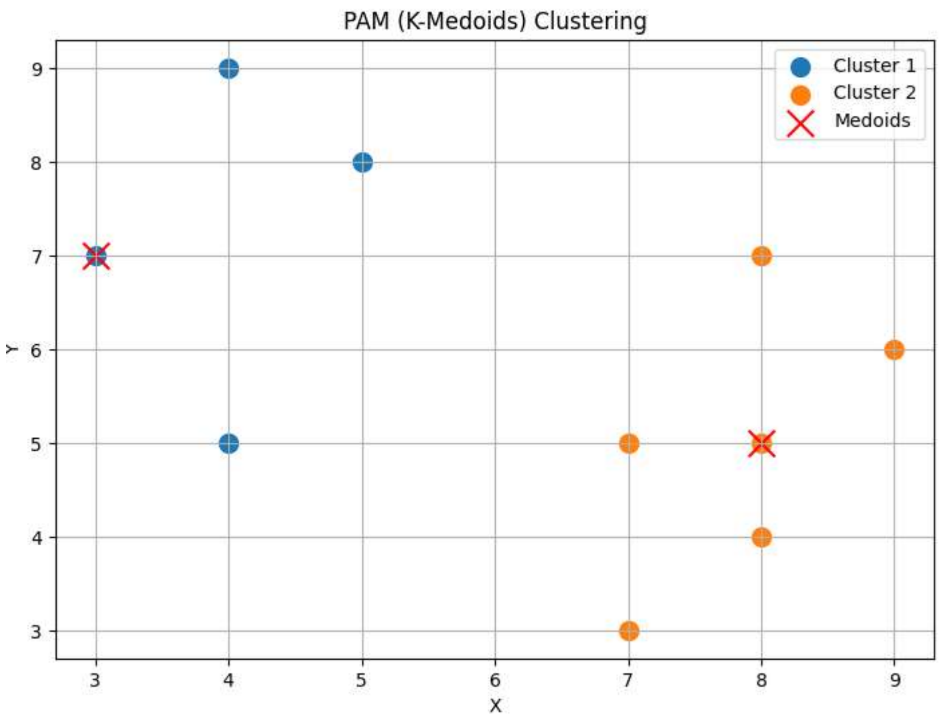
pip install scikit-learn-extra

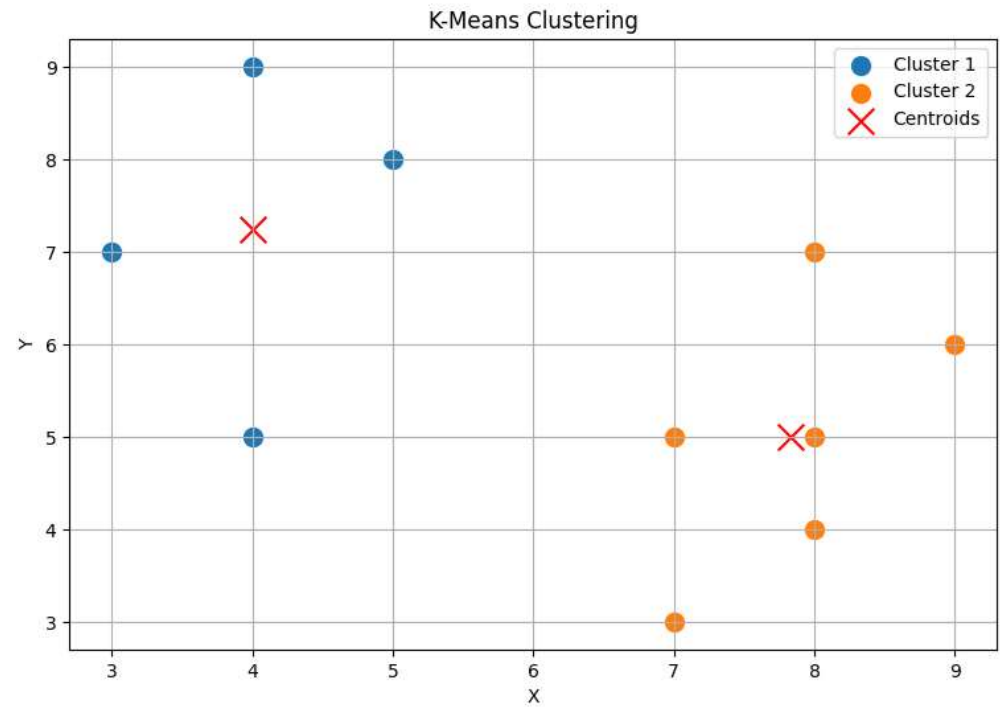
```python
# Install scikit-learn-extra if not installed
# !pip install scikit-learn-extra

import numpy as np
import matplotlib.pyplot as plt
from sklearn_extra.cluster import KMedoids

# Data from the image
X = [8, 3, 4, 9, 8, 5, 7, 8, 7, 4]
Y = [7, 7, 9, 6, 5, 8, 3, 4, 5, 5]

# Combine the X and Y into a single array of data points
data = np.array(list(zip(X, Y)))

# Apply PAM (K-Medoids) with k=2 clusters
kmedoids = KMedoids(n_clusters=2, random_state=0)
kmedoids.fit(data)

# Get cluster labels
labels = kmedoids.labels_

# Plot the data points and clusters
plt.figure(figsize=(6,6))
for i, label in enumerate(np.unique(labels)):
    cluster_data = data[labels == label]
    plt.scatter(cluster_data[:, 0], cluster_data[:, 1],
                label=f'Cluster {i+1}', s=100)

# Mark the medoids
medoids = kmedoids.cluster_centers_
plt.scatter(medoids[:, 0], medoids[:, 1], marker='x',
            color='red', s=200, label='Medoids')

# Add labels, title, and legend
plt.xlabel('X')
plt.ylabel('Y')
plt.title('PAM (K-Medoids) Clustering')
plt.legend()

# Show the plot
plt.grid(True)
plt.show()
```

PAM

Examples

pip install scikit-learn-extra





PAM

Examples

Calculate clusters for X = np.array([7, 3, 4, 9, 8, 5, 7, 8, 1, 2])

Y = np.array([7, 7, 9, 6, 5, 8, 1, 4, 5, 5]) using PAM?

PAM vs. K-means

Medoids vs. Centroids:

- In PAM, the cluster center is always an actual data point (medoid), while k-means uses the average (centroid).

Robustness:

- PAM is more robust to noise and outliers because it uses medoids, which are less influenced by extreme values compared to centroids.

Computational Cost:

- PAM **can be slower than k-means due to the repeated computation of distances for medoid swaps**, especially on large datasets.

Hierarchical clustering

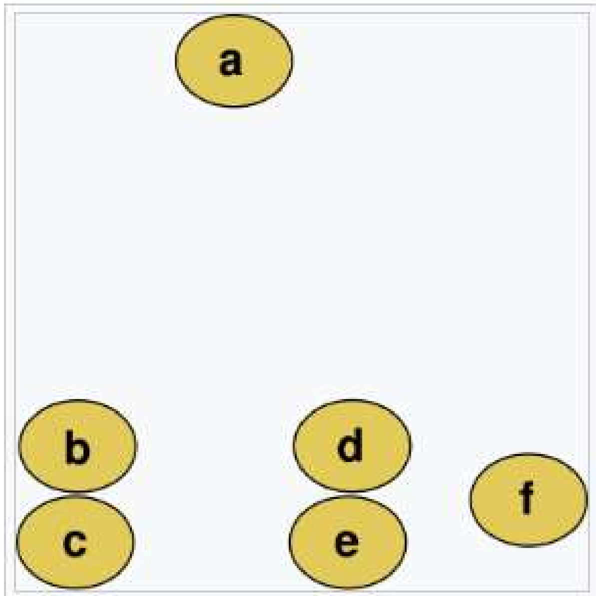
Definitions:

Wikipedia: In data mining and statistics, **hierarchical clustering (also called hierarchical cluster analysis or HCA)** is a method of cluster analysis that seeks to build a hierarchy of clusters, fall into two categories:

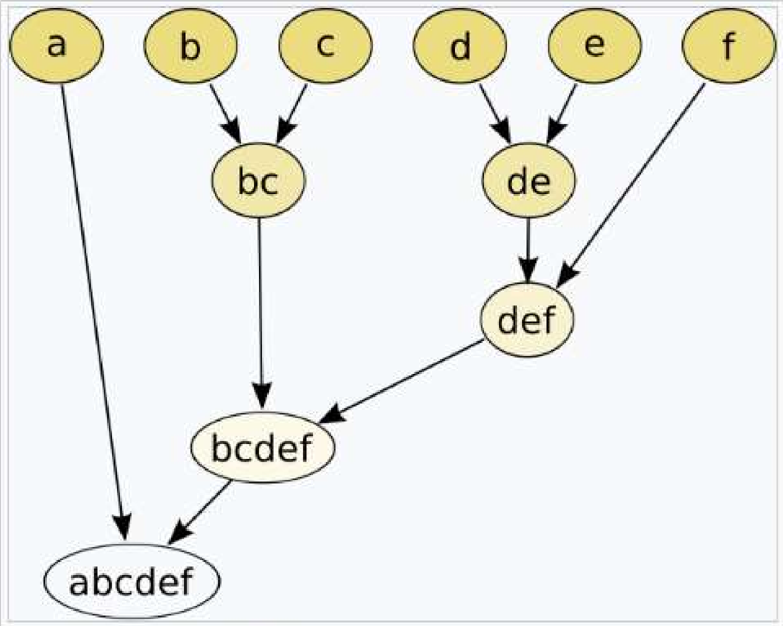
- **Agglomerative:** This is a "bottom-up" approach: Each observation starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy.
- **Divisive:** This is a "top-down" approach: All observations start in one cluster, and splits are performed recursively as one moves down the hierarchy.

Hierarchical clustering

Agglomerative clustering example



Raw data



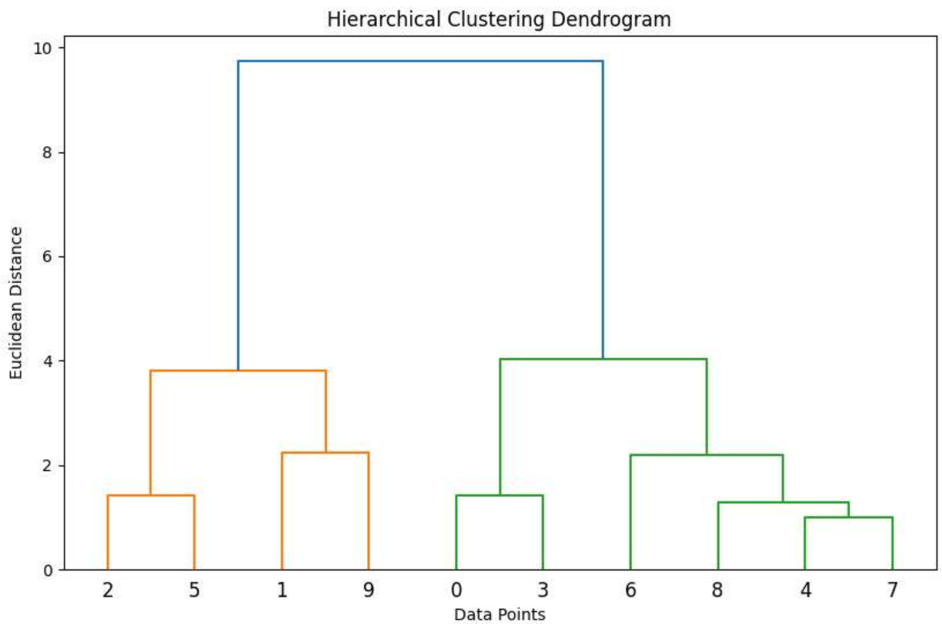
Traditional representation

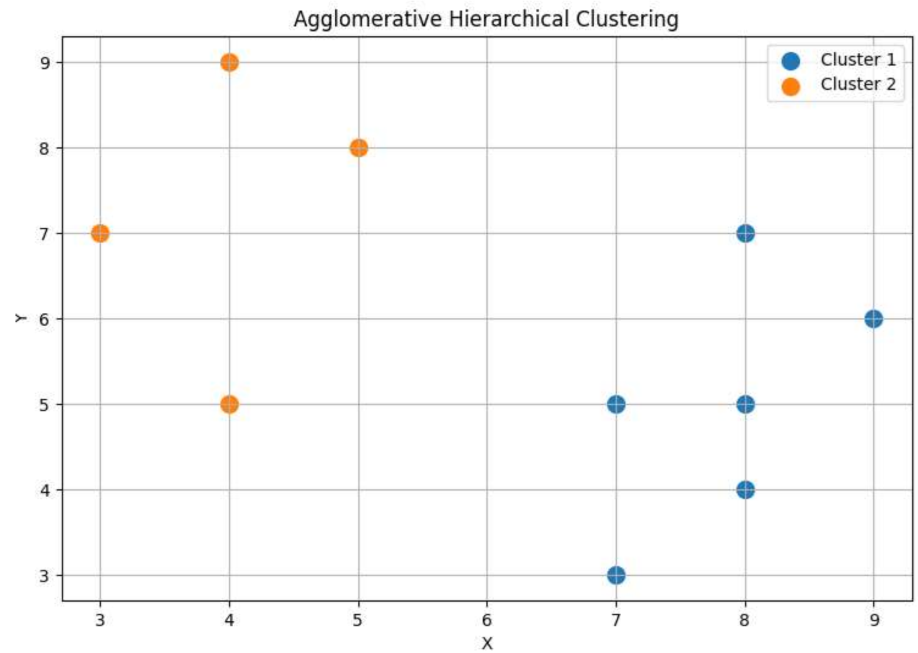
Hierarchical clustering

Agglomerative clustering example

X = [8, 3, 4, 9, 8, 5, 7, 8, 7, 4]

Y = [7, 7, 9, 6, 5, 8, 3, 4, 5, 5]





Hierarchical clustering

Agglomerative clustering example

```python
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.cluster import AgglomerativeClustering

# Data from the image
X = [8, 3, 4, 9, 8, 5, 7, 8, 7, 4]
Y = [7, 7, 9, 6, 5, 8, 3, 4, 5, 5]

# Combine the X and Y into a single array of data points
data = np.array(list(zip(X, Y)))

# Perform hierarchical clustering using different linkage methods
linked = linkage(data, method='ward')

# Plot the dendrogram
plt.figure(figsize=(8, 6))
dendrogram(linked,
           orientation='top',
           distance_sort='ascending',
           show_leaf_counts=True)
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Data Points')
plt.ylabel('Euclidean Distance')
plt.show()
```

```python
# Perform Agglomerative Clustering (cut the dendrogram at 2 clusters)
cluster = AgglomerativeClustering(n_clusters=2,
            metric='euclidean', linkage='ward')
labels = cluster.fit_predict(data)

# Plot the clusters
plt.figure(figsize=(6,6))
for i, label in enumerate(np.unique(labels)):
    cluster_data = data[labels == label]
    plt.scatter(cluster_data[:, 0], cluster_data[:, 1],
                label=f'Cluster {i+1}', s=100)

# Add labels, title, and legend
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Agglomerative Hierarchical Clustering')
plt.legend()
plt.grid(True)
plt.show()
```

DBSCAN

Definitions

Wikipedia: **Density-based spatial clustering of applications with noise (DBSCAN)** is a data clustering algorithm proposed by Martin Ester, Hans-Peter Kriegel, Jörg Sander and Xiaowei Xu in 1996.

It is a density-based clustering non-parametric algorithm:

- given a set of points in some space, it groups together points that are closely packed (points with many nearby neighbors),
- marks as outliers points that lie alone in low-density regions (those whose nearest neighbors are too far away).

https://www.geeksforgeeks.org/dbscan-clustering-in-ml-density-based-clustering/

https://www.youtube.com/watch?v=-p354tQsKrs

DBSCAN

Key Concepts

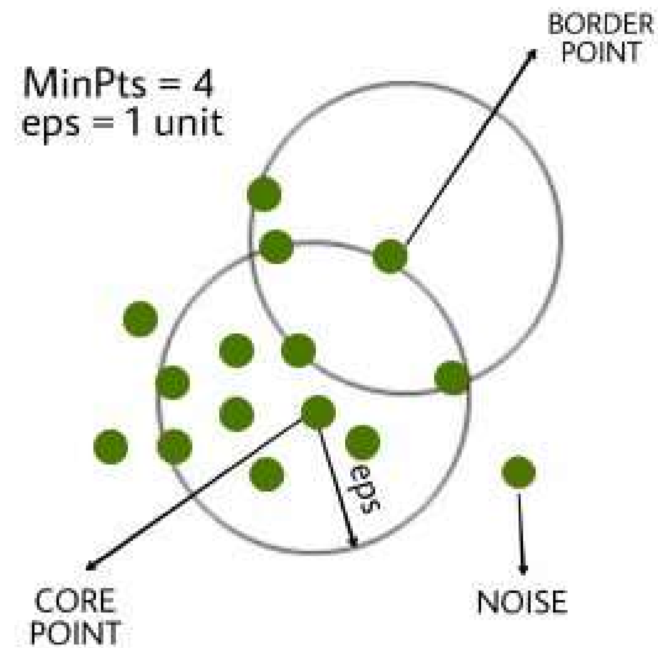
Epsilon (ε): This parameter defines the **radius** of the neighborhood around a point.

MinPts: This parameter specifies the **minimum number of points required to form a dense region**.

Core Points: A point is considered a core point if it has at least MinPts neighbors (including itself) within the ε radius.

Border Points: A point that is not a core point but lies within the ε neighborhood of a core point.

Noise Points: Any point that is neither a core point nor a border point is considered noise.



DBSCAN

Examples

X = [8, 3, 4, 9, 8, 5, 7, 8, 7, 4], Y = [7, 7, 9, 6, 5, 8, 3, 4, 5, 5]

Steps:

- **Set Parameters**: Choose values for ε (epsilon) and MinPts. For this example, let's set ε = 2 and MinPts = 3.
- **Calculate Distances:** Compute the distance between each pair of points. We will use Euclidean distance.
- **Identify Core Points:** A point is a core point if it has at least MinPts (3) neighbors within ε (2).
- **Form Clusters:** Use the core points to form clusters and identify border and noise points.

DBSCAN

Examples

X = [8, 3, 4, 9, 8, 5, 7, 8, 7, 4], Y = [7, 7, 9, 6, 5, 8, 3, 4, 5, 5]

Step 1: Set Parameters

- Epsilon (ε) = 2
- MinPts = 3

DBSCAN

Examples

X = [8, 3, 4, 9, 8, 5, 7, 8, 7, 4], Y = [7, 7, 9, 6, 5, 8, 3, 4, 5, 5]

Step 2: Calculate Distances

Point	(X, Y)
P1	(8, 7)
P2	(3, 7)
P3	(4, 9)
P4	(9, 6)
P5	(8, 5)
P6	(5, 8)
P7	(7, 3)
P8	(8, 4)
P9	(7, 5)
P10	(4, 5)

$$d(P_i, P_j) = \sqrt{(X_i - X_j)^2 + (Y_i - Y_j)^2}$$

Let's calculate a few distances for demonstration:

- $d(P1, P2) = \sqrt{(8-3)^2 + (7-7)^2} = \sqrt{5^2} = 5$
- $d(P1, P3) = \sqrt{(8-4)^2 + (7-9)^2} = \sqrt{4^2 + (-2)^2} = \sqrt{16+4} = \sqrt{20} \approx 4.47$
- $d(P1, P4) = \sqrt{(8-9)^2 + (7-6)^2} = \sqrt{1^2 + 1^2} = \sqrt{2} \approx 1.41$
- $d(P1, P5) = \sqrt{(8-8)^2 + (7-5)^2} = \sqrt{0 + 2^2} = 2$
- $d(P1, P6) = \sqrt{(8-5)^2 + (7-8)^2} = \sqrt{3^2 + (-1)^2} = \sqrt{9+1} = \sqrt{10} \approx 3.16$
- $d(P1, P7) = \sqrt{(8-7)^2 + (7-3)^2} = \sqrt{1^2 + 4^2} = \sqrt{1+16} = \sqrt{17} \approx 4.12$

DBSCAN

Examples

X = [8, 3, 4, 9, 8, 5, 7, 8, 7, 4], Y = [7, 7, 9, 6, 5, 8, 3, 4, 5, 5]

Step 3: Identify Core Points

- **P1 (8, 7):** Neighbors: P4 (1.41), P5 (2) → 2 neighbors (not a core point)
- **P2 (3, 7):** Neighbors: P1 (5), P3 (6.16), P4 (6.08), P5 (5.66), P6 (2.24), P7 (4.12), P8 (5), P9 (4.12), P10 (0) → 1 neighbor (not a core point)
- **P3 (4, 9):** Neighbors: P1 (4.47), P2 (6.16), P3 (0), P4 (5), P5 (5.10), P6 (1), P7 (6.08), P8 (5.1), P9 (4.47), P10 (4) → 1 neighbor (not a core point)
- **P4 (9, 6):** Neighbors: P1 (1.41), P2 (6.08), P3 (5), P5 (1), P6 (4.12), P7 (2.24), P8 (4.12), P9 (3.16), P10 (5.10) → 3 neighbors (core point)
- **P5 (8, 5):** Neighbors: P1 (2), P2 (5.66), P3 (5.1), P4 (1), P6 (3.16), P7 (2.24), P8 (1.41), P9 (2.24), P10 (4) → 4 neighbors (core point)
- **P6 (5, 8):** Neighbors: P1 (3.16), P2 (2.24), P3 (1), P4 (4.12), P5 (3.16), P7 (5.66), P8 (4.47), P9 (3.16), P10 (3) → 3 neighbors (core point)

DBSCAN

Examples

X = [8, 3, 4, 9, 8, 5, 7, 8, 7, 4], Y = [7, 7, 9, 6, 5, 8, 3, 4, 5, 5]

Step 3: Identify Core Points

- **P6 (5, 8):** Neighbors: P1 (3.16), P2 (2.24), P3 (1), P4 (4.12), P5 (3.16), P7 (5.66), P8 (4.47), P9 (3.16), P10 (3) → 3 neighbors (core point)
- **P7 (7, 3):** Neighbors: P1 (4.12), P2 (4.12), P3 (6.08), P4 (2.24), P5 (2.24), P6 (5.66), P8 (1), P9 (1), P10 (4) → 2 neighbors (not a core point)
- **P8 (8, 4):** Neighbors: P1 (2), P2 (5), P3 (5.1), P4 (4.12), P5 (1.41), P6 (4.47), P7 (1), P9 (1), P10 (2) → 4 neighbors (core point)
- **P9 (7, 5):** Neighbors: P1 (3.16), P2 (4.12), P3 (4.47), P4 (3.16), P5 (2.24), P6 (3.16), P7 (1), P8 (1) → 5 neighbors (core point)
- **P10 (4, 5):** Neighbors: P1 (4), P2 (0), P3 (4.47), P4 (5.10), P5 (4), P6 (3), P7 (4) → 1 neighbor (not a core point)

DBSCAN

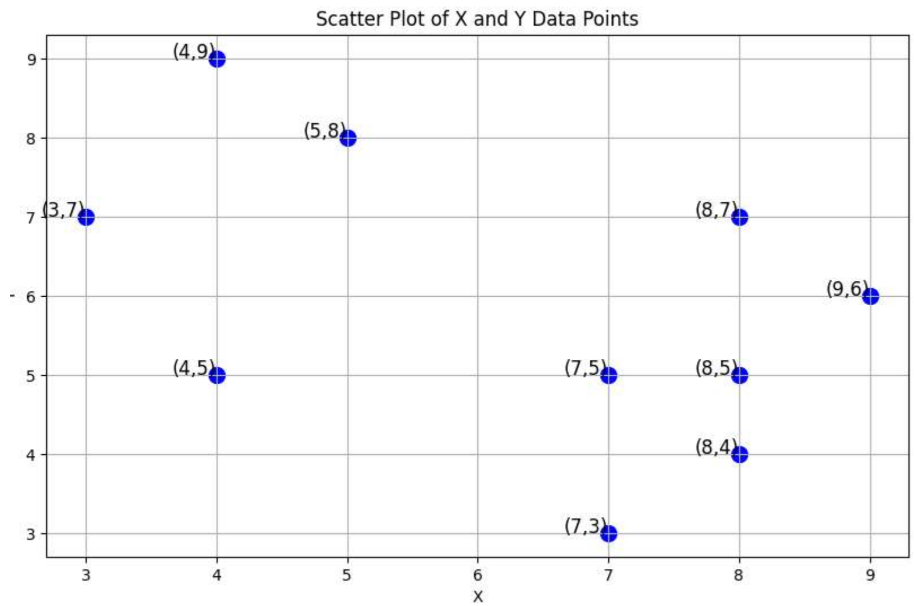
Examples

X = [8, 3, 4, 9, 8, 5, 7, 8, 7, 4], Y = [7, 7, 9, 6, 5, 8, 3, 4, 5, 5]

Step 4: Form Clusters

Core Points: **P1, P7, P5, P8, P9**

- Form clusters based on the core points
  - Starting from P1 (core point), we find its neighbors (P7, P5, P8, P9) that are core points, leading to:
  - Cluster Members: P1, P7, P5, P8, P9 (as P1 is connected to core points)



Point	(X, Y)
P1	(8, 7)
P2	(3, 7)
P3	(4, 9)
P4	(9, 6)
P5	(8, 5)
P6	(5, 8)
P7	(7, 3)
P8	(8, 4)
P9	(7, 5)
P10	(4, 5)

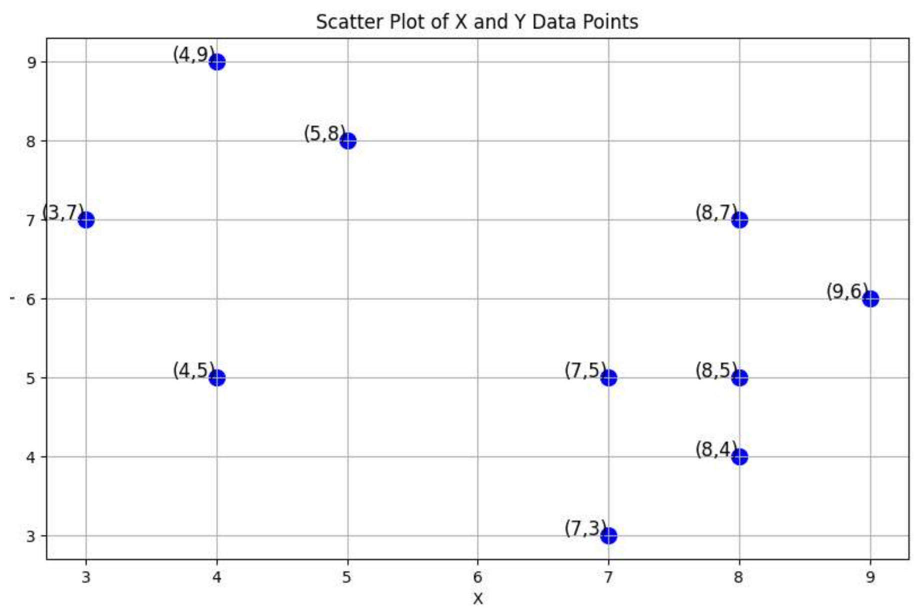
DBSCAN

Examples

X = [8, 3, 4, 9, 8, 5, 7, 8, 7, 4],
Y = [7, 7, 9, 6, 5, 8, 3, 4, 5, 5]

Final Clusters and Noise

- Cluster 0: {**P1, P7, P5, P8, P9**}
- Others: Noise Points



Point	(X, Y)
P1	(8, 7)
P2	(3, 7)
P3	(4, 9)
P4	(9, 6)
P5	(8, 5)
P6	(5, 8)
P7	(7, 3)
P8	(8, 4)
P9	(7, 5)
P10	(4, 5)

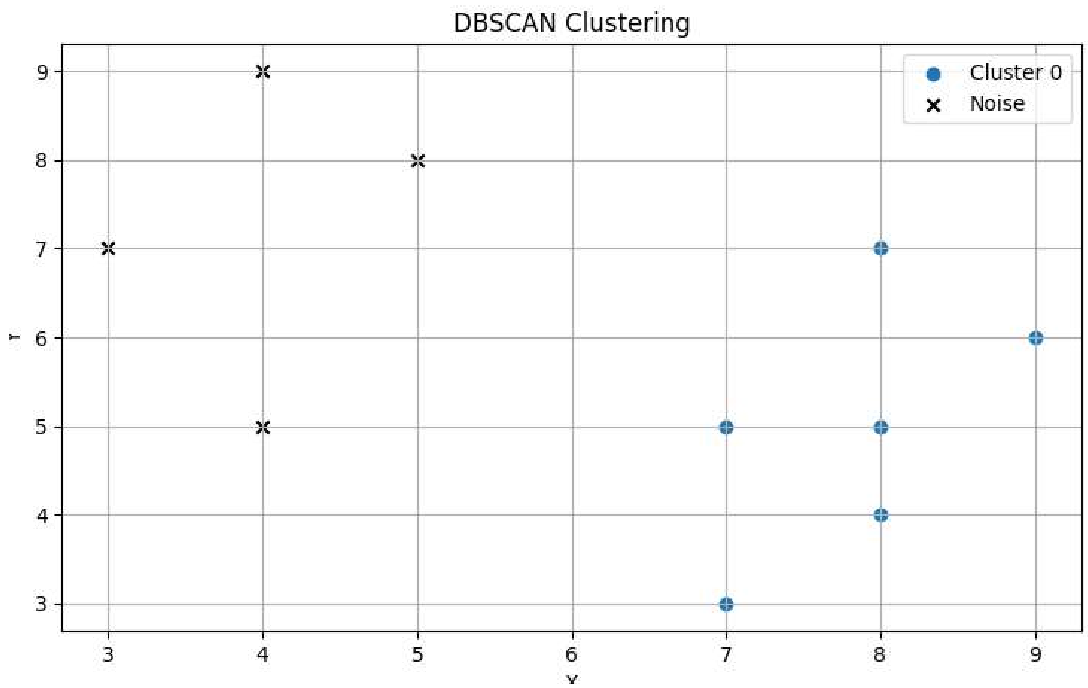
DBSCAN

Examples

X = [8, 3, 4, 9, 8, 5, 7, 8, 7, 4],
Y = [7, 7, 9, 6, 5, 8, 3, 4, 5, 5]

Final Clusters and Noise

- Cluster 0: {**P1, P7, P5, P8, P9**}
- Others: Noise Points



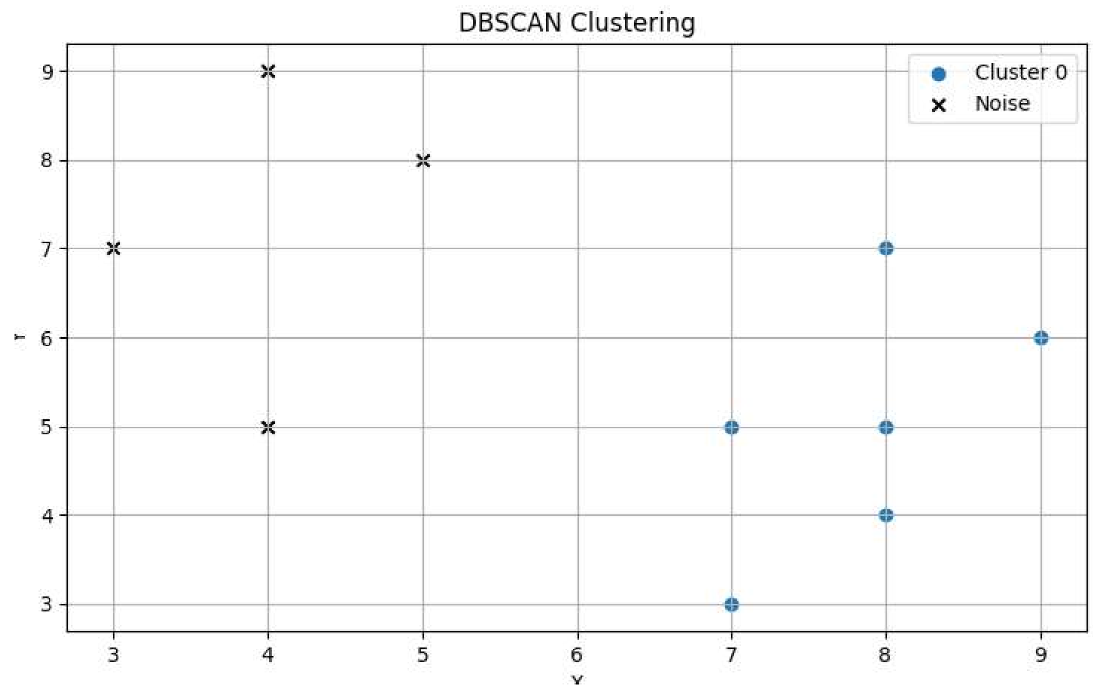
DBSCAN

Examples

X = [8, 3, 4, 9, 8, 5, 7, 8, 7, 4],
Y = [7, 7, 9, 6, 5, 8, 3, 4, 5, 5]

Final Clusters and Noise

- Cluster 0: {**P1, P7, P5, P8, P9**}
- Others: Noise Points



DBSCAN

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN

# Step 1: Define the dataset
X = np.array([8, 3, 4, 9, 8, 5, 7, 8, 7, 4])
Y = np.array([7, 7, 9, 6, 5, 8, 3, 4, 5, 5])

# Combine X and Y into a single array of coordinates
data = np.column_stack((X, Y))

# Step 2: Set parameters for DBSCAN
epsilon = 2   # ε
min_samples = 3   # MinPts

# Step 3: Apply DBSCAN
dbscan = DBSCAN(eps=epsilon, min_samples=min_samples)
labels = dbscan.fit_predict(data)

# Step 4: Output clusters and noise
unique_labels = set(labels)
clusters = {label: [] for label in unique_labels if label != -1}
```


```python
# Step 3: Apply DBSCAN
dbscan = DBSCAN(eps=epsilon, min_samples=min_samples)
labels = dbscan.fit_predict(data)

# Step 4: Output clusters and noise
unique_labels = set(labels)
clusters = {label: [] for label in unique_labels if label != -1}
```


```python
for point, label in zip(data, labels):
    if label != -1:  # Ignore noise points
        clusters[label].append(point)

# Display results
print("Clusters:")
for cluster_id, points in clusters.items():
    print(f"Cluster {cluster_id}: {points}")

print(f"Noise points: {data[labels == -1]}")
```

```python
# Step 5: Visualization
plt.figure(figsize=(8, 6))
for cluster_id, points in clusters.items():
    points = np.array(points)
    plt.scatter(points[:, 0], points[:, 1],
        label=f'Cluster {cluster_id}')

# Highlight noise points
noise_points = data[labels == -1]
if noise_points.size > 0:
    plt.scatter(noise_points[:, 0], noise_points[:, 1],
        color='black', label='Noise', marker='x')

plt.title('DBSCAN Clustering')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.grid(True)
plt.show()
```

DBSCAN

Exercises

https://www.youtube.com/watch?v=-p354tQsKrs