dbscan

July 2, 2024

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

1.0.1 Imports

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

1.0.2 Implementation

[2]:
```
def get_dist_matrix(data):
    n = len(data)
    dist_matrix = np.zeros((n, n))

    for i in range(n):
        for j in range(i + 1, n):
            dist = np.linalg.norm(data[i] - data[j])
            dist_matrix[i][j], dist_matrix[j][i] = dist, dist

    return dist_matrix
```

[3]:
```
def get_core_points(dist_matrix, epsilon, min_points):
    return [i for i in range(len(dist_matrix)) if np.sum(dist_matrix[i] <=␣
 ↪epsilon) >= min_points]
```

[22]:
```
def expand_cluster(core_point, dist_matrix, epsilon, min_points, cluster_id,␣
 ↪belongs_to_cluster, core_points):
    belongs_to_cluster[core_point] = cluster_id
    neighbors = np.where(dist_matrix[core_point] <= epsilon)[0]

    if len(neighbors) < min_points:
        return

    for neighbor in neighbors:
        if belongs_to_cluster[neighbor] == -1:
            belongs_to_cluster[neighbor] = cluster_id
            if neighbor in core_points:
```

```
                expand_cluster(neighbor, dist_matrix, epsilon, min_points,␣
↪cluster_id, belongs_to_cluster, core_points)
```

[23]:
```
def dbscan(data, epsilon, min_points):
    # Compute distance matrix and use that to get the core points
    dist_matrix = get_dist_matrix(data)
    core_points = get_core_points(dist_matrix, epsilon, min_points)

    # Assign data points to clusters
    cluster_id = 0
    belongs_to_cluster = np.full(len(data), -1)
    for point in core_points:
        if belongs_to_cluster[point] == -1:
            cluster_id += 1
            expand_cluster(point, dist_matrix, epsilon, min_points, cluster_id,␣
↪belongs_to_cluster, core_points)

    return belongs_to_cluster
```

1.0.3 Generate Test Dataset

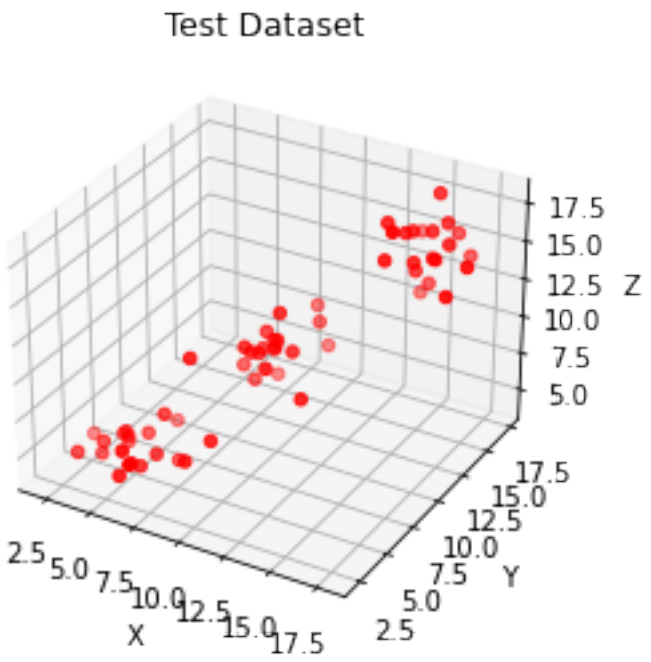
[48]:
```
data = np.array([np.random.normal(5, 1.5, 3) for i in range(20)])
data = np.concatenate((data, [np.random.normal(10, 1.5, 3) for i in range(20)]))
data = np.concatenate((data, [np.random.normal(15, 1.5, 3) for i in range(20)]))
```

[79]:
```
fig = plt.figure()
ax = fig.add_subplot(111, projection = "3d")

ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.set_zlabel("Z")
ax.set_title("Test Dataset")

ax.scatter(data[:, 0], data[:, 1], data[:, 2], c = "red", marker = "o")
```

[79]:
```
<mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x1f662dd2640>
```



1.0.4 Test

[80]:
```
epsilon = 2
min_points = 4

belongs_to_cluster = dbscan(data, epsilon, min_points)
```

[81]:
```
belongs_to_cluster
```

[81]:
```
array([ 1,  1,  1,  1,  1, -1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1, -1,
        1,  1,  1,  2,  2,  2,  2,  2,  2,  2, -1,  2,  2,  2, -1, -1, -1,
       -1,  2, -1,  2,  2, -1, -1,  3,  3,  3,  3,  3,  3,  3,  3, -1,  3,
        3,  3,  3,  3,  3,  3,  3,  3,  3])
```

[82]:
```
clusters = {}
for i, cluster_id, in enumerate(belongs_to_cluster):
    if cluster_id not in clusters:
        clusters[cluster_id] = []
    clusters[cluster_id].append(data[i])
```

[83]:
```
fig = plt.figure()
ax = fig.add_subplot(111, projection = "3d")

ax.set_xlabel("X")
ax.set_ylabel("Y")
```

```
ax.set_zlabel("Z")
ax.set_title("Test Dataset w/ Clustering")

cluster_1 = np.array(clusters[1])
ax.scatter(cluster_1[:, 0], cluster_1[:, 1], cluster_1[:, 2], c = "green",␣
 ↪marker = "o", label = "Cluster 1")

cluster_2 = np.array(clusters[2])
ax.scatter(cluster_2[:, 0], cluster_2[:, 1], cluster_2[:, 2], c = "orange",␣
 ↪marker = "o", label = "Cluster 2")

cluster_3 = np.array(clusters[3])
ax.scatter(cluster_3[:, 0], cluster_3[:, 1], cluster_3[:, 2], c = "blue",␣
 ↪marker = "o", label = "Cluster 3")

noise = np.array(clusters[-1])
ax.scatter(noise[:, 0], noise[:, 1], noise[:, 2], c = "red", marker = "o",␣
 ↪label = "Noise")

ax.legend()
```

[83]: <matplotlib.legend.Legend at 0x1f662e06910>