LAB 6: DBSCAN Clustering Algorithm

DBSCAN is a density based clustering method. It stands for Density-based spatial clustering of applications with noise clustering method. The main idea behind DBSCAN is that a point belongs to a cluster if it is close to many points from that cluster.

The DBSCAN algorithm uses two parameters:

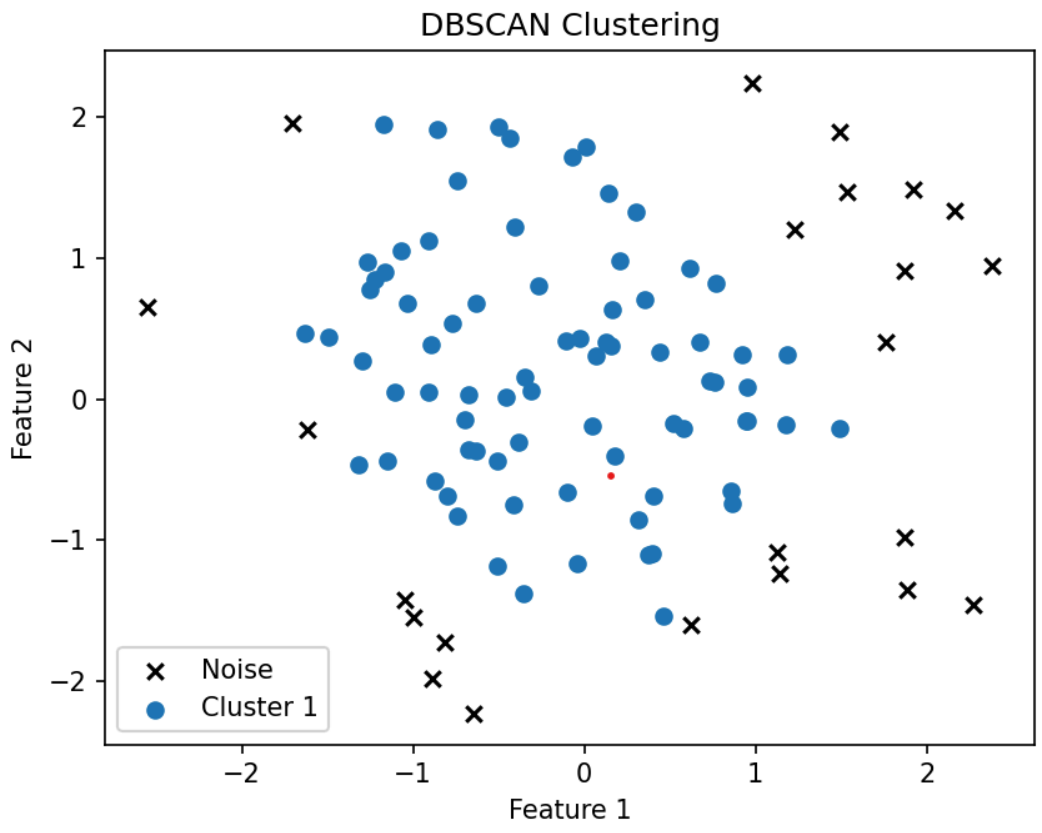
Epsilon (ε): The distance that specifies the neighborhoods. Two points are considered to be neighbors if the distance between them are less than or equal to ε.
minPts: The minimum number of points (a threshold) clustered together for a region to be considered dense.

Based on these two parameters, points are classified as core point, border point, or outlier:

Core Point: Data point that has at least min Pts number of points within epsilon (e) distance.
Border Point: Data point that has at least one core point within epsilon (e) distance and lower than minPts number of points within epsilon (e) distance from it.
Noise or Outlier Point: Data point that has no core points within epsilon (e) distance.

Program:
```
class DBSCAN:
    def __init__(self, eps, min_samples):
        self.eps = eps
        self.min_samples = min_samples
    def _euclidean_distance(self, p1, p2):
        return ((p1 - p2) ** 2).sum() ** 0.5
    def _region_query(self, X, point_idx):
        neighbors = []
        for i, point in enumerate(X):
            if self._euclidean_distance(X[point_idx], point) <= self.eps:
                neighbors.append(i)
        return neighbors
    def _expand_cluster(self, X, labels, point_idx, cluster_label):
        seeds = set(self._region_query(X, point_idx))
        if len(seeds) < self.min_samples:
            labels[point_idx] = -1  # mark as noise
            return False
        else:
            labels[point_idx] = cluster_label
            for seed_idx in seeds:
                labels[seed_idx] = cluster_label
            while seeds:
                current_point_idx = seeds.pop()
                current_neighborhood = set(self._region_query(X, current_point_idx))
                if len(current_neighborhood) >= self.min_samples:
                    for idx in current_neighborhood:
                        if labels[idx] == 0:
                            seeds.add(idx)
                            labels[idx] = cluster_label
        return True

    def fit_predict(self, X):
        cluster_label = 0
        n_points = X.shape[0]
        labels = [0] * n_points  # 0 represents unvisited
        for point_idx in range(n_points):
            if labels[point_idx] != 0:
                continue
            if self._expand_cluster(X, labels, point_idx, cluster_label + 1):
                cluster_label += 1
        return labels
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(0)
X = np.random.randn(100, 2)
dbscan = DBSCAN(eps=0.5, min_samples=5)
labels = dbscan.fit_predict(X)
unique_labels = np.unique(labels)
for label in unique_labels:
    if label == -1:
        plt.scatter(X[labels == label][:, 0], X[labels == label][:, 1], c='k', marker='x',
label='Noise')
    else:
        plt.scatter(X[labels == label][:, 0], X[labels == label][:, 1], label=f'Cluster {label}')
plt.title('DBSCAN Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```


DBSCAN Clustering
Feature 1
Feature 2
Noise
Cluster 1