### **1. Algorithm Overview (20%)**

DBSCAN is a density-based clustering algorithm that identifies clusters as areas of high point density separated by areas of low point density. It categorizes points into three types:

**Core points**: Points that have at least min\_samples points (including themselves) within a distance of eps.

**Border points**: Points that are within the eps neighborhood of a core point but have fewer than min\_samples neighbors themselves.

**Noise points (outliers)**: Points that are neither core nor border points.

**Clustering Process**:

* DBSCAN starts with an arbitrary point.
* If it’s a core point, it begins forming a cluster by collecting all points within its eps radius that are density-reachable (i.e., they can be reached from core points through a chain of neighboring core points).
* This process continues recursively until the cluster cannot be expanded further.
* Points not assigned to any cluster are labeled as noise.

**Key Parameters**

* eps (epsilon):
  + Defines the maximum distance between two points for them to be considered neighbors. It determines the radius of the neighborhood around a point.
* min\_samples:
  + Specifies the minimum number of points required (including the point itself) to form a dense region. This helps differentiate between core points (dense areas) and noise.

Together, these parameters control the sensitivity of DBSCAN to data density and play a critical role in cluster formation.

**Strengths**

* Detects arbitrarily shaped clusters: Unlike K-means, DBSCAN can find non-spherical and complex cluster shapes.
* Handles noise/outliers well: It naturally labels low-density points as noise without forcing them into clusters.
* No need to specify the number of clusters: Unlike K-means, DBSCAN determines the number of clusters based on data distribution.

**Limitations**

* Parameter sensitivity: Choosing appropriate values for eps and min\_samples is crucial and can be non-trivial, especially in high-dimensional spaces.
* Varying density problems: DBSCAN struggles when clusters have significantly different densities—same eps value might be too small for some clusters and too large for others.
* Scalability: The algorithm has a time complexity of approximately O(n²) in the worst case (though this can be improved with spatial indexing), which may be inefficient for large datasets.

### **2. Algorithm Comparison (40%)**

### **Compare Performance: When Does DBSCAN Outperform K-Means and Hierarchical Clustering?**

### The Silhouette Score is a measure of how well each data point fits within its assigned cluster. A higher silhouette score indicates better clustering performance.

### **DBSCAN Performance:**

### Silhouette Score for DBSCAN - make\_moons: 0.3860

### Silhouette Score for DBSCAN - make\_blobs: 0.4850

### **K-Means Performance:**

### Silhouette Score for K-Means - make\_moons: 0.4955

### Silhouette Score for K-Means - make\_blobs: 0.7866

### **Hierarchical Clustering Performance:**

### Silhouette Score for Hierarchical - make\_moons: 0.4487

### Silhouette Score for Hierarchical - make\_blobs: 0.7833

### **When DBSCAN Outperforms:**

### **Non-linearly Separable Data**: DBSCAN is designed to handle datasets where clusters are not necessarily spherical or convex (like the make\_moons dataset). For non-linearly separable datasets (e.g., make\_moons), DBSCAN can identify arbitrarily shaped clusters, which K-Means and hierarchical clustering may struggle with. In such cases, DBSCAN performs better than K-Means or Hierarchical Clustering because it can find clusters that are not just circular or elliptical in shape, while K-Means assumes spherical clusters and hierarchical clustering's performance can degrade with complex shapes.

### **Handling Noise**: DBSCAN performs well when there is a clear distinction between clusters and noise. For example, in datasets with noise points (outliers), DBSCAN can label them as noise (using -1), while K-Means and Hierarchical clustering might incorrectly group them with other clusters. The make\_moons dataset, with its inherent noise, shows DBSCAN’s ability to handle noise points more effectively.

### **When DBSCAN Struggles**:

### **Varying Density of Clusters**: DBSCAN struggles when clusters have varying densities. The make\_blobs dataset is a good example of this—DBSCAN’s performance here is slightly lower than K-Means because it assumes that clusters are equally dense. For highly dense regions and sparse regions in a dataset (like in make\_blobs), DBSCAN might fail to identify appropriate clusters or classify points as noise unnecessarily.

### **Clusters with Varying Densities:** If the dataset contains clusters of differing densities, DBSCAN might assign points from low-density clusters to the noise category (outliers) or merge different clusters together if their density is similar. This is evident from the lower Silhouette Score for make\_blobs (0.4850) compared to K-Means (0.7866), which shows DBSCAN’s difficulty in handling well-separated, evenly dense blobs.

### **High Dimensionality**: DBSCAN struggles with high-dimensional data because the concept of "neighborhood" (distance between points) becomes less meaningful as the number of dimensions increases. As dimensionality increases, the distance between all points becomes similar, making it hard to identify dense regions.

### **Parameter Sensitivity:** The eps and min\_samples parameters in DBSCAN need to be carefully chosen. If eps is too small, most points will be considered noise; if it's too large, DBSCAN may merge distinct clusters together. The \*\*Silhouette Score for make\_moons\*\* (0.3860) could be improved if the eps` parameter were better tuned.

### **Why**: DBSCAN relies on the eps and min\_samples parameters to define the density threshold for clusters. If these parameters are not appropriately tuned for the varying densities, DBSCAN can miss out on meaningful clusters.

### **Trade-offs: What Factors Influence the Choice Between These Clustering Methods?**

### The choice between DBSCAN, K-Means, and Hierarchical Clustering depends on several factors:

### **Shape of Clusters:**

### DBSCAN is preferred when the clusters are of arbitrary shape (e.g., crescent shapes like make\_moons).

### K-Means is better for spherical or convex clusters. It tends to struggle with complex, non-convex shapes.

### Hierarchical Clustering works well with a variety of shapes but might not be as efficient in capturing complex patterns in large datasets.

### **Presence of Noise/Outliers:**

### DBSCAN handles noise well by labeling low-density points as outliers (noise), making it ideal for datasets with outliers or sparse data regions.

### K-Means and Hierarchical Clustering generally assume all points belong to a cluster and struggle with outliers. K-Means may pull outliers into nearby clusters, whereas Hierarchical Clustering might create additional small clusters for noise.

### **Number of Clusters**:

### K-Means requires specifying the number of clusters in advance (k), which can be a disadvantage if the optimal k is not known. It may require running multiple tests (e.g., using the elbow method) to find an appropriate k.

### DBSCAN does not require specifying the number of clusters, as clusters emerge based on density and distance. However, the right choice of eps and min\_samples is critical.

### Hierarchical Clustering also does not require a predefined number of clusters and allows for easy visualization of the hierarchical relationships through a dendrogram. However, the number of clusters can be subjective and depends on the desired level of granularity.

### **3. Table Update (20%)**

**Compare and contrast characteristics for all three algorithms:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **k-Means** | **Hierarchical Clustering** | **DBSCAN** |
| **Definition** | Partitioning algorithm that assigns points to k clusters based on centroids | Builds a hierarchy of clusters using distance metrics | Density-based algorithm that clusters based on density and distance |
| **Approach** | Iteratively minimizes variance within k clusters | Agglomerative (bottom-up) or divisive (top-down) | Identifies dense regions of points and groups them together, while labeling sparse points as noise |
| **Number of Clusters** | Requires predefined k | Can be determined from dendrogram but subjective | No need to predefine the number of clusters; determined by density and parameters (eps, min\_samples) |
| **Cluster Shape** | Prefers spherical clusters | Works well with various shapes but can be unstable | Can identify arbitrarily shaped clusters, based on density |
| **Initialization** | Randomly selects k initial centroids | No initialization needed | No initialization required; clusters form naturally based on density |
| **Result** | Hard assignments—each point belongs to a single cluster | Hierarchical structure (tree/dendrogram) | Hard assignments (points assigned to clusters or labeled as noise) |
| **Interpretability** | Moderate—cluster assignments but no hierarchy | High—dendrogram can be analyzed | Moderate—can clearly identify noise and clusters, but no hierarchical structure |
| **Strengths** | Simple, fast and efficient on large datasets | Can capture hierarchical relationships | 1. Can detect arbitrarily shaped clusters 2. Handles noise and outliers effectively 3. No need to predefine the number of clusters |
| **Limitations** | Sensitive to initial centroids and k choice | Computationally expensive for large datasets | 1. Sensitive to eps and min\_samples parameters 2. Struggles with clusters of varying densities 3. Can be computationally expensive with large datasets |

### **4. Code Documentation & Submission Quality (20%)**

https://github.com/Sheona-Hans/BINF-5507-Materials/tree/main/Assignment/Assignment3