Module 5

Q. Spectral Clustering in Machine Learning

=>

1. Introduction

1. **Spectral Clustering** is an advanced **unsupervised learning algorithm** that groups data points based on **connectivity** rather than just distance.
2. Unlike algorithms like **K-Means**, which use **compactness (distance)** between data points, spectral clustering focuses on **how well the data points are connected**.
3. It can cluster two data points together even if they are far apart, as long as there is a **path of connected points** between them.
4. This makes it especially useful for identifying **non-linear or complex-shaped clusters**.
5. Example – Two circular or spiral-shaped clusters that are connected internally but distant in terms of Euclidean distance can be separated using spectral clustering.

2. Concept and Working

1. Spectral Clustering is based on the idea of a **graph representation of data**.
2. Each **data point is treated as a node**, and the **similarity between points** is represented as an **edge** between nodes.
3. The algorithm uses **eigenvalues and eigenvectors** of this similarity matrix to map data into a **lower-dimensional space** for easier clustering.
4. Once the data is projected, a simple algorithm like **K-Means** is applied on the transformed data to form clusters.
5. Hence, spectral clustering combines the power of **graph theory and linear algebra** for efficient clustering.

3. Connectivity-Based Clustering

1. Traditional clustering depends on **distance**, which fails for non-convex clusters.
2. Spectral clustering, however, depends on **connectivity**, meaning it groups points that are **linked directly or indirectly**.
3. This allows points to belong to the same cluster even if they are not close in terms of geometric distance.
4. Example – In a "two moons" dataset, each moon can be correctly identified as one cluster because points within a moon are connected, even though the distance between moons is small.

Steps in Spectral Clustering

Step 1 – Building the Similarity Graph

1. The first step is to build a **similarity graph** represented as an **adjacency matrix (A)**.
2. This matrix shows how each data point is connected to others.
3. The similarity graph can be built in several ways:

(a) Epsilon-Neighborhood Graph

1. A parameter **ε (epsilon)** is selected beforehand.
2. Each point is connected to all points lying within its **epsilon-radius**.
3. If all distances are similar in scale, **edge weights are not stored**, as they add no extra information.
4. The resulting graph is **undirected and unweighted**.
5. Example – In a dataset where ε = 2, all points within distance 2 of each other are considered connected.

(b) K-Nearest Neighbors (KNN) Graph

1. A parameter **k** is chosen in advance.
2. An edge is drawn from point **u to v** if **v** is among the **k-nearest neighbors** of **u**.
3. This results in a **directed and weighted graph**, because **v** being near **u** doesn't always mean **u** is near **v**.
4. To make it **undirected**, two common approaches are used:

- (i) Draw an edge if **either** u is among v's k-nearest neighbors **or** vice versa.
- (ii) Draw an edge **only if both** u and v are among each other's k-nearest neighbors.

5. Example – If k = 3, each point connects to its 3 closest points, forming small local neighborhoods.

(c) Fully Connected Graph

1. In this graph, **each point is connected to every other point**.
2. The edges are **weighted** by the **distance** between two points.
3. This type of graph captures the **global structure** of the data.
4. Example – In image clustering, every pixel is connected to every other pixel, but with smaller weights for distant pixels.

Step 2 – Constructing the Laplacian Matrix

1. Once the similarity matrix is built, the **degree matrix (D)** is computed.
2. The **degree matrix** has diagonal entries representing the total connections of each node.
3. The **Graph Laplacian** is then calculated as:
   (L = D - A), where A is the adjacency matrix.
4. The Laplacian captures the connectivity structure of the entire dataset.

Step 3 – Eigenvalue Decomposition

1. Next, the **eigenvalues and eigenvectors** of the Laplacian matrix are calculated.
2. The **smallest k eigenvectors** are selected to represent the data in a **lower-dimensional space**.
3. These eigenvectors capture the **main connectivity patterns** in the graph.
4. Each row of this transformed matrix represents a data point in the new feature space.

Step 4 – Applying Clustering

1. The transformed data points are then clustered using a simple algorithm like **K-Means**.
2. K-Means now operates in the new space, where clusters are linearly separable.
3. Finally, each cluster in the new space corresponds to a meaningful group in the original data.

10. Example

1. Consider 200 data points forming two **crescent-shaped (moon)** clusters.
2. K-Means fails to separate them because of their curved boundaries.
3. Spectral clustering creates a **connectivity graph** and finds two clusters based on graph structure.
4. Thus, it successfully separates both moon-shaped groups using connectivity rather than distance.

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

=>

1. Introduction

1. DBSCAN is a **density-based clustering algorithm** used to group data points that are close to each other based on a distance measure.
2. It stands for **Density-Based Spatial Clustering of Applications with Noise**.
3. Unlike **K-Means**, DBSCAN does not require you to specify the number of clusters beforehand.
4. It can find clusters of **arbitrary shapes** and can **handle noise (outliers)** efficiently.
5. It is widely used in applications like **geographical data analysis, image segmentation, and anomaly detection**.

2. Key Idea

1. The main idea of DBSCAN is to find areas of **high data density** separated by areas of **low density**.
2. A cluster is formed when there are enough points close to one another based on two parameters:
    a. **Epsilon (ε)** – The radius of the neighborhood around a data point.
    b. **MinPts** – The minimum number of points required to form a dense region.
3. Points that do not belong to any cluster are considered **noise or outliers**.

3. Types of Points in DBSCAN

1. **Core Points:** A point that has at least **MinPts** points (including itself) within a distance of **ε**.
    a. Example: If ε = 2 and MinPts = 4, a point with 4 or more neighbors within 2 units is a core point.
2. **Border Points:** A point that is **not a core point** but lies within the neighborhood (ε) of a core point.
3. **Noise Points:** A point that is **neither a core point nor a border point**, meaning it lies in a low-density region.

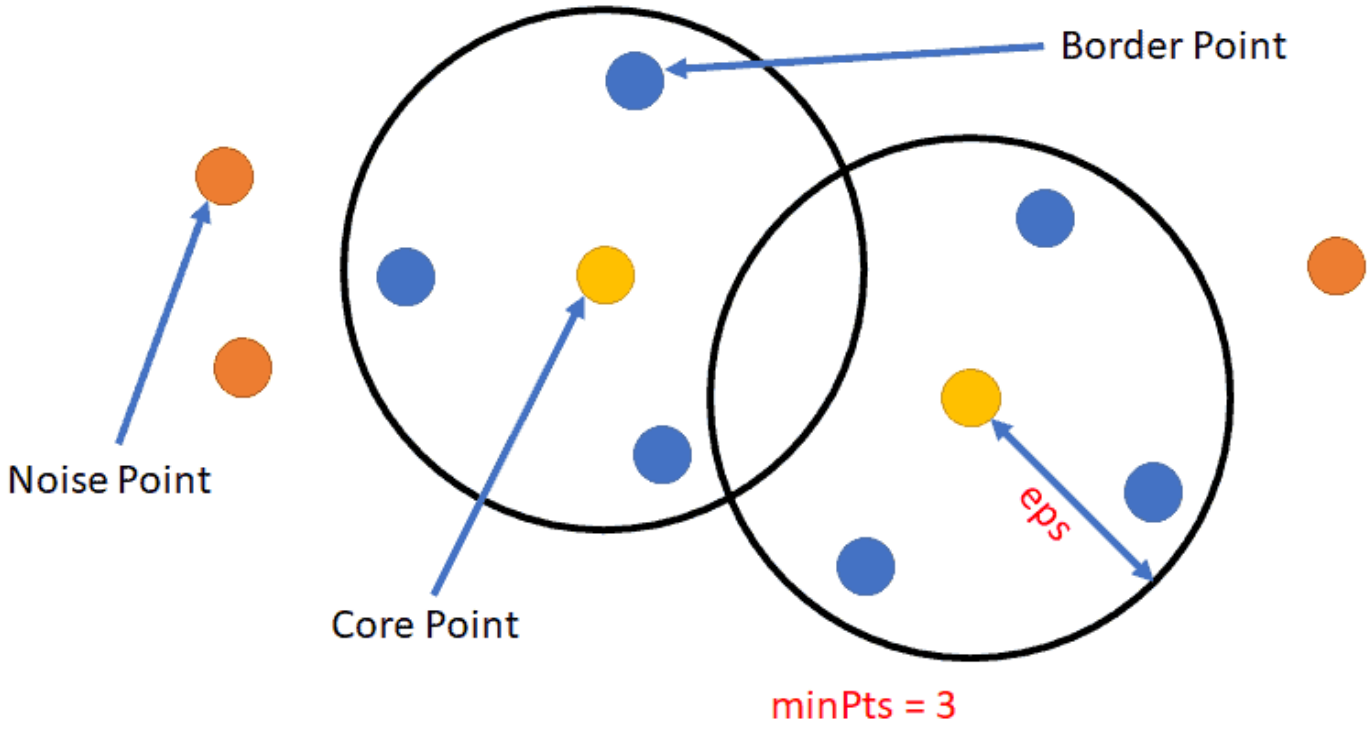
4. Steps in DBSCAN Algorithm

1. **Step 1:** Select an unvisited data point from the dataset.
2. **Step 2:** Find all points within the **ε-neighborhood** of that point.
3. **Step 3:**
    a. If the number of neighbors ≥ **MinPts**, mark the point as a **core point** and form a new cluster.
    b. Else, mark the point as **noise** temporarily.

4. **Step 4:** Expand the cluster by adding all points that are **density-reachable** (directly or indirectly connected to the core point).
5. **Step 5:** Repeat the process for all remaining unvisited points until all points are labeled as **core, border, or noise**.

5. Example

1. Suppose you have a dataset of geographical locations of houses.
2. Let **ε = 1 km** and **MinPts = 5**.
3. DBSCAN will identify clusters of houses that are closely packed (within 1 km of each other) and mark isolated houses as **outliers**.
4. This helps in identifying **densely populated regions** and **isolated properties**.



Q. Minimum Spanning Tree (MST)

=>

1. A **Spanning Tree** is a subgraph of a connected, undirected graph that includes all the vertices and has no cycles.
2. It connects all the vertices with the **minimum possible number of edges** — if there are (V) vertices, then the spanning tree will have exactly (V - 1) edges.
3. A **Minimum Spanning Tree (MST)** is a spanning tree that has the **least total edge weight** among all possible spanning trees.
4. MST helps in **reducing complexity** and **minimizing connection cost** while keeping all points connected.
5. MST is widely used in **network design, clustering, image segmentation**, and **transportation systems**.

2. Properties of a Spanning Tree

1. The number of **vertices (V)** in the spanning tree equals the number of vertices in the graph.
2. The number of **edges (E)** in the spanning tree is always **V - 1**.
3. It must be **connected** — there should be only one component.
4. It must be **acyclic** — no closed loops are allowed.
5. The **total cost or weight** of the tree is the sum of the edge weights included.
6. There can be **multiple MSTs** possible for a single graph if several edges have the same weights.

3. Minimum Spanning Tree for Clustering

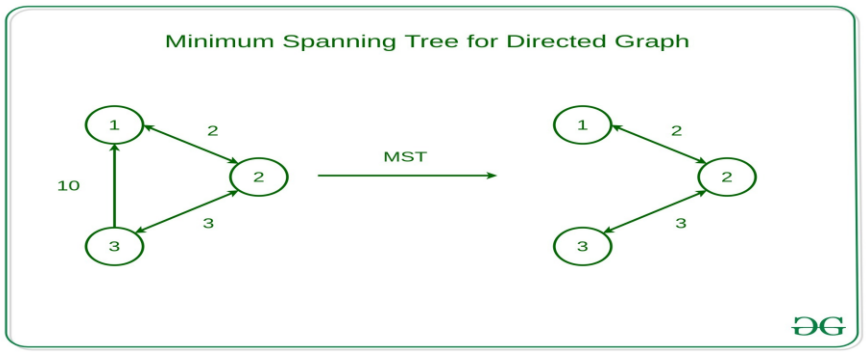
1. MST can be used for **clustering** by treating each data point as a node in a graph.
2. The **distance or dissimilarity** between data points is used as the **edge weight**.
3. MST is first constructed to connect all points using the smallest distances possible.
4. Once the MST is formed, the **longest edges** (which connect distant points or clusters) can be **removed**.
5. The remaining connected components represent **clusters** of similar points.
6. This technique helps to identify **natural groupings** in data without specifying the number of clusters in advance.

Example:

- Suppose you have 6 points representing cities on a map.
- MST connects these cities with the **minimum total road length**.

- If you remove the longest road between two distant groups, you effectively form **two clusters** of nearby cities.

4. Working Example



Step 1: Graph Construction

1. Consider three vertices: 1, 2, and 3.
2. The edges and their weights are:
    a. Edge (1 → 2) with weight 2
    b. Edge (2 → 3) with weight 3
    c. Edge (3 → 1) with weight 10

Step 2: Finding MST

1. To form an MST, we select edges with the **lowest weights** that do **not form a cycle**.
2. The edges selected are:
    a. (1 → 2) with weight 2
    b. (2 → 3) with weight 3
3. The total weight = 2 + 3 = **5**, which is the **minimum possible**.
4. The edge (3 → 1) with weight 10 is **ignored** because it forms a cycle and increases cost.

Step 3: Result

1. The MST connects all nodes (1, 2, 3) with the **least total weight (5)**.
2. If used for clustering, the weak connection (weight 10) could be cut, forming **two distinct clusters** based on connectivity.
3. The diagram shows the **original graph** and the **MST** on the right side with minimal edges.

5. Algorithms to Find MST

A. Kruskal's Algorithm

1. It is a **greedy algorithm** that builds the MST by adding edges in increasing order of weight.
2. Steps:
    a. Sort all edges by their weights.
    b. Add the smallest edge that does not form a cycle.
    c. Repeat until there are (V - 1) edges in the tree.
3. It is efficiently implemented using **Disjoint Set Union (DSU)** to track connected components.
4. Example: Used in **network design** or **data clustering** where cost minimization is required.

B. Prim's Algorithm

1. Another **greedy approach**, but it grows the MST one vertex at a time.
2. Steps:
    a. Start from any vertex.
    b. At each step, add the smallest edge connecting the current MST to a new vertex.
    c. Continue until all vertices are included.
3. Best implemented using a **priority queue** to select minimum-weight edges quickly.
4. Example: Used in **routing problems**, **image segmentation**, and **clustering** based on similarity.

C. Boruvka's Algorithm

1. One of the oldest MST algorithms.
2. Starts by treating each vertex as an individual tree.
3. In each iteration, the **cheapest edge** connecting each tree to another is added.
4. Trees merge gradually until one MST remains.
5. Useful for **parallel computing** and **large sparse graphs**.