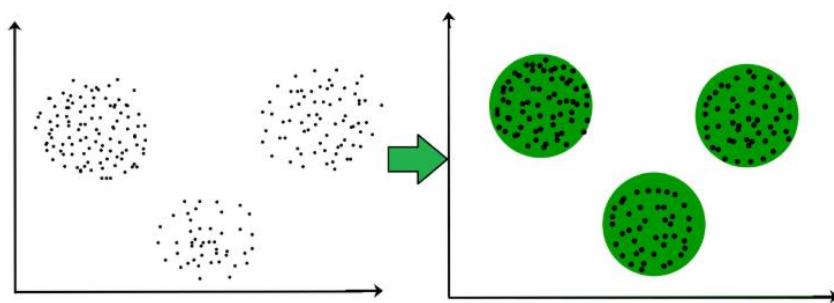


Unit 3

Unsupervised Learning

❖ Clustering:

Clustering is a machine learning technique used to group similar data points into clusters. It is an **unsupervised learning** method, meaning it works without predefined labels or categories, instead analyzing the inherent structure of the data. Clustering aims to maximize the similarity of data points within a cluster and minimize the similarity between different clusters.



Key Features of Clustering:

1. **Similarity Measurement:** Clustering algorithms rely on metrics like distance (e.g., Euclidean, Manhattan) or density to evaluate how similar or dissimilar data points are.
2. **Unsupervised:** There is no labeled data; the algorithm identifies patterns or groups based solely on the input features.
3. **Applications:** Clustering is widely used in customer segmentation, image processing, bioinformatics, anomaly detection, and more.

Types of Clustering Algorithms

1. **Partitioning Methods:** Divide the data into a fixed number of clusters.
 - **Example:** K-Means, K-Medoids
2. **Hierarchical Methods:** Create a hierarchy of clusters, often represented as a dendrogram.
 - **Example:** Agglomerative Clustering, Divisive Clustering
3. **Density-Based Methods:** Identify clusters based on regions of high data density.
 - **Example:** DBSCAN, OPTICS
4. **Model-Based Methods:** Assume data is generated by a mixture of underlying probability distributions.
 - **Example:** Gaussian Mixture Models (GMM)

5. **Grid-Based Methods:** Divide the data space into grids and group points within dense grids.
 - **Example:** STING

Example of Clustering in Action

Imagine you have customer data, including their age, income, and purchasing habits. Using clustering:

- Customers might be grouped into "high spenders," "budget-conscious buyers," and "occasional shoppers."
- This segmentation can help businesses tailor marketing strategies to specific groups.

Clustering is a versatile tool for exploring and understanding the structure of data without needing explicit labels.

Applications of Clustering in different fields:

1. **Marketing:** It can be used to characterize & discover customer segments for marketing purposes.
2. **Biology:** It can be used for classification among different species of plants and animals.
3. **Libraries:** It is used in clustering different books on the basis of topics and information.
4. **Insurance:** It is used to acknowledge the customers, their policies and identifying the frauds.
5. **City Planning:** It is used to make groups of houses and to study their values based on their geographical locations and other factors present.
6. **Earthquake studies:** By learning the earthquake-affected areas we can determine the dangerous zones.
7. **Image Processing:** Clustering can be used to group similar images together, classify images based on content, and identify patterns in image data.
8. **Genetics:** Clustering is used to group genes that have similar expression patterns and identify gene networks that work together in biological processes.
9. **Finance:** Clustering is used to identify market segments based on customer behavior, identify patterns in stock market data, and analyze risk in investment portfolios.

10. **Customer Service:** Clustering is used to group customer inquiries and complaints into categories, identify common issues, and develop targeted solutions.
11. **Manufacturing:** Clustering is used to group similar products together, optimize production processes, and identify defects in manufacturing processes.
12. **Medical diagnosis:** Clustering is used to group patients with similar symptoms or diseases, which helps in making accurate diagnoses and identifying effective treatments.
13. **Fraud detection:** Clustering is used to identify suspicious patterns or anomalies in financial transactions, which can help in detecting fraud or other financial crimes.
14. **Traffic analysis:** Clustering is used to group similar patterns of traffic data, such as peak hours, routes, and speeds, which can help in improving transportation planning and infrastructure.
15. **Social network analysis:** Clustering is used to identify communities or groups within social networks, which can help in understanding social behavior, influence, and trends.
16. **Cybersecurity:** Clustering is used to group similar patterns of network traffic or system behavior, which can help in detecting and preventing cyberattacks.
17. **Climate analysis:** Clustering is used to group similar patterns of climate data, such as temperature, precipitation, and wind, which can help in understanding climate change and its impact on the environment.
18. **Sports analysis:** Clustering is used to group similar patterns of player or team performance data, which can help in analyzing player or team strengths and weaknesses and making strategic decisions.
19. **Crime analysis:** Clustering is used to group similar patterns of crime data, such as location, time, and type, which can help in identifying crime hotspots, predicting future crime trends, and improving crime prevention strategies.

Program:

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

# Example data (2D points)
data = np.array([
    [1, 2], [1, 4], [1, 0],
    [10, 2], [10, 4], [10, 0]
])

# Number of clusters
k = 2

# Initialize and fit K-Means
kmeans = KMeans(n_clusters=k, random_state=0)
kmeans.fit(data)

# Get cluster centers and labels
centers = kmeans.cluster_centers_
labels = kmeans.labels_

# Plotting the results
for i in range(k):
    cluster_points = data[labels == i]
    plt.scatter(cluster_points[:, 0], cluster_points[:, 1], label=f'Cluster {i+1}')

plt.scatter(centers[:, 0], centers[:, 1], c='red', marker='x', label='Centroids')
plt.legend()
plt.title('K-Means Clustering Example')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.show()
```

Explanation of the Code

1. **Dataset:** A simple array of 2D points is created.
2. **KMeans Model:** The number of clusters (kkk) is set to 2.
3. **Fit the Model:** The model identifies the clusters in the dataset.
4. **Results:**
 - o `labels`: Indicates the cluster assignment for each data point.
 - o `centers`: Locations of the cluster centroids.
5. **Visualization:** Data points are plotted along with their cluster centroids.

what is Centroids:

A **centroid** is the central point of a cluster in a dataset. In the context of clustering algorithms like **K-Means**, the centroid represents the "mean" position of all the data points that belong to

that cluster. It acts as a reference point for the cluster and is used to define its structure and boundaries.

Key Features of Centroids:

1. Calculation:

- For a cluster with n points, the centroid is the arithmetic mean of all the points in that cluster.
- In a 2D dataset, the coordinates of the centroid are:

$$x_c = \frac{1}{n} \sum_{i=1}^n x_i, \quad y_c = \frac{1}{n} \sum_{i=1}^n y_i$$

- Similarly, in higher dimensions, the centroid is calculated by averaging the coordinates in each dimension.

2. Role in Clustering:

- In the **K-Means** algorithm, centroids are iteratively updated to minimize the distance between data points and their corresponding centroid.
- The centroid determines the "center" of a cluster, ensuring that the cluster accurately represents its data points.

3. Properties:

- It doesn't need to correspond to an actual data point in the dataset.
- Centroids can shift during the clustering process as clusters are refined.

Example

If we have the following points in a 2D space for one cluster:

$$(1, 2), (3, 4), (5, 6)$$

The centroid is calculated as:

$$x_c = \frac{1+3+5}{3} = 3, \quad y_c = \frac{2+4+6}{3} = 4$$

Thus, the centroid is located at (3, 4).

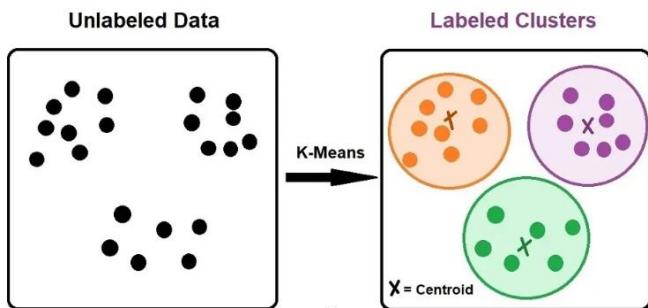
`data = np.array([[1, 2], [1, 4], [1, 0], [10, 2], [10, 4], [10, 0]])` this value in centroid finde using this formula

The centroids of the two clusters are:

- **Centroid 1:** (1.0, 2.0) (1.0, 2.0) (1.0, 2.0)
- **Centroid 2:** (10.0, 2.0) (10.0, 2.0) (10.0, 2.0)

These centroids are calculated as the mean of the points within each respective cluster.

❖ Data using k-means clustering



K-means is an unsupervised learning method for clustering data points. The algorithm iteratively divides data points into K clusters by minimizing the variance in each cluster.

K-means clustering is a popular unsupervised machine learning algorithm used for grouping data into clusters. It identifies groups of data points (clusters) such that points in the same cluster are more similar to each other than to points in other clusters.

Steps in K-means Clustering

1. **Choose the Number of Clusters (k):** Decide how many clusters you want to create.
2. **Initialize Centroids:** Randomly select k initial cluster centroids.
3. **Assign Points to Clusters:**
 - o Calculate the distance between each data point and each centroid.
 - o Assign each point to the nearest centroid.
4. **Update Centroids:** Recalculate the centroids as the mean of all points in each cluster.
5. **Repeat:**
 - o Continue assigning points and updating centroids until:
 - Centroids no longer change significantly (convergence).
 - A maximum number of iterations is reached.

Applications of K-means

- Customer segmentation in marketing.
- Image compression.
- Anomaly detection.
- Document clustering.
- Bioinformatics.

Program 1:

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

# Sample data
X = np.array([
    [1, 2], [1.5, 1.8], [5, 8],
    [8, 8], [1, 0.6], [9, 11],
    [8, 2], [10, 2], [9, 3]
])

# Applying K-means
kmeans = KMeans(n_clusters=3, random_state=0)
kmeans.fit(X)

# Cluster centroids and labels
centroids = kmeans.cluster_centers_
labels = kmeans.labels_

# Visualize the clusters
colors = ['r', 'g', 'b']
for i in range(len(X)):
    plt.scatter(X[i][0], X[i][1], c=colors[labels[i]])

plt.scatter(centroids[:, 0], centroids[:, 1], marker='*', s=200, c='yellow') # Centroids
plt.title('K-means Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

Program 2:

```
import sys
import matplotlib
matplotlib.use('Agg') # Non-interactive backend for saving plots

import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Data points
x = [4, 5, 10, 4, 3, 11, 14, 6, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]

data = list(zip(x, y))

# Apply K-means clustering
kmeans = KMeans(n_clusters=2)
kmeans.fit(data)
```

```

# Plot the data points with their cluster labels
plt.scatter(x, y, c=kmeans.labels_, cmap='viridis')
plt.scatter(*zip(*kmeans.cluster_centers_), color='red', marker='*', s=200,
label='Centroids')
plt.title("K-means Clustering")
plt.xlabel("X")
plt.ylabel("Y")
plt.legend()

# Save the plot to a file
plt.savefig('kmeans_output.png') # Save the figure as an image file
print("Plot saved as 'kmeans_output.png'")

```

❖ compressing image using vector quantization

cmd in install:

pip install pillow numpy scikit-learn matplotlib

Vector quantization (VQ) is a powerful technique used for compressing images by mapping pixel intensities or feature values into a smaller, discrete set of representative values. The steps to compress an image using vector quantization are as follows:

1. Divide the Image into Blocks

- Split the image into smaller blocks of fixed size (e.g., 4×44 or 8×88 pixels).
- Treat each block as a vector in a multi-dimensional space (e.g., for an 8×88 block, each vector has 64 dimensions).

2. Create a Codebook

- Perform clustering on the training data (a collection of blocks) using a clustering algorithm like **k-means**.
- The centroids of the clusters form the *codebook*, which represents the most common patterns or features in the image.

3. Quantize the Blocks

- For each block in the image:
 1. Find the nearest vector (centroid) in the codebook.
 2. Replace the block with the index of its closest centroid.
- This step effectively reduces the amount of information needed, as instead of storing the original pixel values, you only store the index of the centroid.

4. Store the Codebook and Compressed Data

- Save the codebook separately; it is required for reconstruction.
- The compressed image is represented as a sequence of indices pointing to the entries in the codebook.

5. Reconstruct the Image

- During decompression:
 - Replace each index in the compressed data with its corresponding vector from the codebook.
 - Reshape the vectors back into blocks and reassemble them to form the image.

Advantages of Vector Quantization:

- High compression ratio.
- Works well for images with repetitive patterns or textures.

Limitations:

- Lossy compression: The reconstructed image may not be identical to the original.
- Artifacts can occur if the codebook size is too small or if the blocks are not representative of the image.

compressing image using vector quantization

Vector Quantization (VQ) is a technique for compressing images by dividing an image into smaller regions and representing them with representative values or "codewords." Below is an easy-to-understand Python program to compress an image using VQ.

This program uses `NumPy` for mathematical computations and `Pillow` for image handling.

cmd in install:

```
pip install pillow numpy scikit-learn matplotlib
```

Program:

```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

def vector_quantization(image_path, n_colors):
    """
    Compress an image using vector quantization with KMeans clustering.

    :param image_path: Path to the image file.
    :param n_colors: Number of colors for quantization.
    """

    # Load the image
    image = Image.open(image_path)
    image = image.convert("RGB")
    image_array = np.array(image)
```

```

# Reshape image to a 2D array of pixels
h, w, c = image_array.shape
pixels = image_array.reshape(-1, 3)

# Apply KMeans to cluster pixels
kmeans = KMeans(n_clusters=n_colors, random_state=0)
kmeans.fit(pixels)
compressed_pixels = kmeans.cluster_centers_[kmeans.labels_]

# Reshape compressed pixels to the original image shape
compressed_image = compressed_pixels.reshape(h, w, c).astype('uint8')

# Save and display the result
compressed_img = Image.fromarray(compressed_image)
compressed_img.save("compressed_image.png")

# Display the original and compressed images
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.title("Original Image")
plt.imshow(image)
plt.axis('off')

plt.subplot(1, 2, 2)
plt.title(f"Compressed Image ({n_colors} colors)")
plt.imshow(compressed_img)
plt.axis('off')

plt.show()

# Usage
image_path = "NLS-MainArt.800.jpg" # Replace with the path to your image
n_colors = 16 # Number of colors for quantization
vector_quantization(image_path, n_colors)

```

How it Works

1. **Image Loading:** The image is loaded and converted to an RGB format.
2. **Reshape:** The image is reshaped into a 2D array of pixels (each pixel is represented by its RGB values).
3. **KMeans Clustering:** Pixels are clustered into `n_colors` groups using the KMeans algorithm.
4. **Reconstruction:** Each pixel is replaced by the centroid of its cluster, reducing the color palette.
5. **Save and Display:** The compressed image is saved and displayed alongside the original.

❖ Mean shift clustering model

Meanshift is falling under the category of a clustering algorithm in contrast of Unsupervised learning that assigns the data points to the clusters iteratively by shifting points towards the mode (mode is the highest density of data points in the region, in the context of the Meanshift).

As such, it is also known as the **Mode-seeking algorithm**. Mean-shift algorithm has applications in the field of image processing and computer vision.

Note:

Given a set of data points, the algorithm iteratively assigns each data point towards the closest cluster centroid and direction to the closest cluster centroid is determined by where most of the points nearby are at. So each iteration each data point will move closer to where the most points are at, which is or will lead to the cluster center. When the algorithm stops, each point is assigned to a cluster.

Unlike the popular K-Means cluster algorithm, mean-shift does not require specifying the number of clusters in advance. The number of clusters is determined by the algorithm with respect to the data.

Mean-shift clustering is a non-parametric, density-based clustering algorithm that can be used to identify clusters in a dataset. It is particularly useful for datasets where the clusters have arbitrary shapes and are not well-separated by linear boundaries.

The basic idea behind mean-shift clustering is to shift each data point towards the mode (i.e., the highest density) of the distribution of points within a certain radius.

The algorithm iteratively performs these shifts until the points converge to a local maximum of the density function. These local maxima represent the clusters in the data.

The process of mean-shift clustering algorithm can be summarized as follows:

Initialize the data points as cluster centroids.

Repeat the following steps until convergence or a maximum number of iterations is reached:

For each data point, calculate the mean of all points within a certain radius (i.e., the “kernel”) centered at the data point.

Shift the data point to the mean.

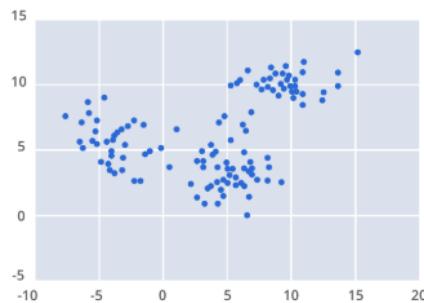
Identify the cluster centroids as the points that have not moved after convergence.

Return the final cluster centroids and the assignments of data points to clusters. One of the main advantages of mean-shift clustering is that it does not require the number of clusters to be specified beforehand. It also does not make any assumptions about the distribution of the data, and can handle arbitrary shapes and sizes of clusters. However, it can be sensitive to the choice of kernel and the radius of the kernel.

Mean-Shift clustering can be applied to various types of data, including image and video processing, object tracking and bioinformatics.

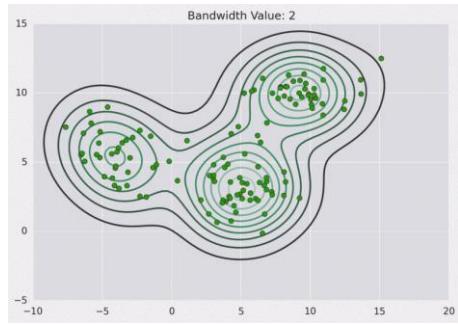
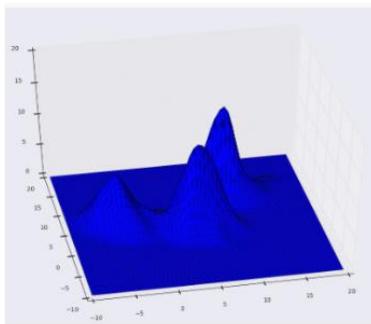
Kernel Density Estimation –

The first step when applying mean shift clustering algorithms is representing your data in a mathematical manner this means representing your data as points such as the set below.



Mean-shift builds upon the concept of kernel density estimation, in short KDE. Imagine that the above data was sampled from a probability distribution. KDE is a method to estimate the underlying distribution also called the probability density function for a set of data. It works by placing a kernel on each point in the data set. A kernel is a fancy mathematical word for a weighting function generally used in convolution. There are many different types of kernels, but the most popular one is the Gaussian kernel. Adding up all of the individual kernels generates a probability surface example density function. Depending on the kernel bandwidth parameter used, the resultant density function will vary. Below is the KDE surface for our points above using a Gaussian kernel with a kernel bandwidth of 2.

Surface plot:



CMD in install :

```
pip install numpy pandas scikit-learn matplotlib
```

Program:

```
import numpy as np
import pandas as pd
from sklearn.cluster import MeanShift
from sklearn.datasets import make_blobs
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Generate synthetic data
clusters = [[2, 2, 2], [7, 7, 7], [5, 13, 13]]
X, _ = make_blobs(n_samples=150, centers=clusters, cluster_std=0.60)

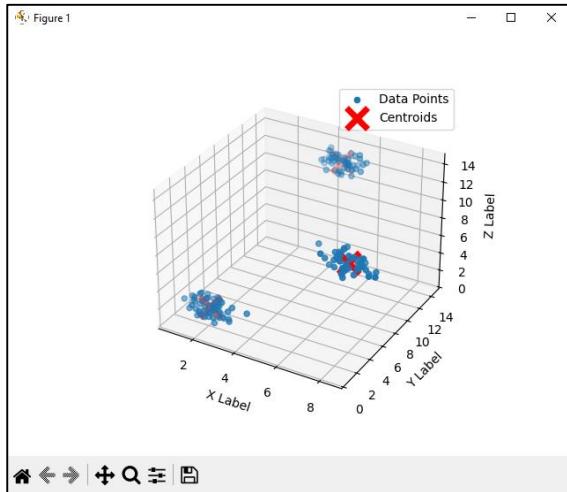
# Train the Mean Shift model
ms = MeanShift()
ms.fit(X)
cluster_centers = ms.cluster_centers_

# Plot data points and centroids in a 3D graph
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X[:, 0], X[:, 1], X[:, 2], marker='o', label='Data Points')

# Plot the cluster centers
ax.scatter(
    cluster_centers[:, 0],
    cluster_centers[:, 1],
    cluster_centers[:, 2],
    marker='x', color='red', s=300, linewidth=5, label='Centroids', zorder=10
)

# Add labels and legend
ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')
plt.legend()
plt.show()
```

output:



To illustrate, suppose we are given a data set $\{u_i\}$ of points in d-dimensional space, sampled from some larger population, and that we have chosen a kernel K having bandwidth parameter h . Together, these data and kernel function returns the following kernel density estimator for the full population's density function.

$$f_K(\mathbf{u}) = \frac{1}{nh^d} \sum_{i=1}^n K\left(\frac{\mathbf{u} - \mathbf{u}_i}{h}\right)$$

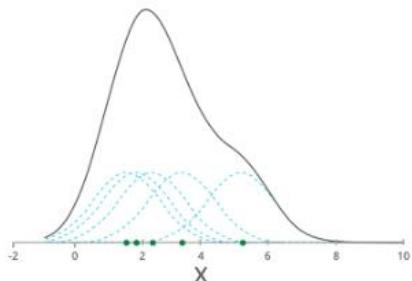
The kernel function here is required to satisfy the following two conditions:

1. $\int K(\mathbf{u})d\mathbf{u} = 1$
2. $K(\mathbf{u}) = K(|\mathbf{u}|)$ for all values of \mathbf{u}

Two popular kernel functions that satisfy these conditions are given by-

1. Flat/Uniform $K(\mathbf{u}) = \frac{1}{2} \begin{cases} 1 & -1 \leq |\mathbf{u}| \leq 1 \\ 0 & \text{else} \end{cases}$
2. Gaussian $= K(\mathbf{u}) = \frac{1}{(2\pi)^{d/2}} e^{-\frac{1}{2}|\mathbf{u}|^2}$

Below we plot an example in one dimension using the Gaussian kernel to estimate the density of some population along the x-axis. We can see that each sample point adds a small Gaussian to our estimate, centered about it and equations above may look a bit intimidating, but the graphic here should clarify that the concept is pretty straightforward.



Example of kernel density estimation using a gaussian kernel for each data point: Adding up small Gaussians about each example returns our net estimate for the total density, the black curve.

❖ Agglomerative clustering:

Agglomerative Clustering is a type of hierarchical clustering algorithm that builds a hierarchy of clusters in a "bottom-up" manner. It starts with each data point as its own cluster and iteratively merges the closest clusters until a single cluster remains or a stopping criterion is met.

How Agglomerative Clustering Works

1. **Initialization:**
 - Treat each data point as a separate cluster (singleton clusters).
 - At the start, if there are n data points, there will be n clusters.
2. **Proximity Calculation:**
 - Compute the distance (or similarity) between all pairs of clusters.
 - A distance metric (e.g., Euclidean, Manhattan) and a linkage criterion (e.g., single, complete, average) are used to define "closeness."
3. **Merge Clusters:**
 - Identify the two closest clusters and merge them into a single cluster.
 - Update the distance matrix to reflect the new set of clusters.
4. **Repeat:**
 - Steps 2 and 3 are repeated until:
 - A predefined number of clusters is reached, or
 - All points are merged into a single cluster.
5. **Hierarchy Representation:**
 - The merging process can be represented as a **dendrogram**, a tree-like diagram that shows the sequence of merges and the distance at which clusters were combined.

Linkage Criteria

The choice of linkage determines how the distance between clusters is computed:

1. **Single Linkage:**
 - Distance between the closest points of two clusters.
 - Can lead to "chaining," where clusters form elongated shapes.
2. **Complete Linkage:**
 - Distance between the farthest points of two clusters.
 - Tends to create compact, spherical clusters.

3. **Average Linkage:**
 - o Average distance between all points in one cluster and all points in another.
 - o Balances between compactness and chaining.
4. **Ward's Linkage:**
 - o Minimizes the total variance within clusters.
 - o Often used when the goal is to minimize cluster dispersion.

Advantages

1. **Hierarchy Representation:**
 - o Results in a dendrogram, which provides a clear picture of the clustering structure.
2. **No Need for a Fixed Number of Clusters:**
 - o Can determine the number of clusters from the dendrogram by cutting at an appropriate level.
3. **Flexibility:**
 - o Supports various distance metrics and linkage criteria.

Disadvantages

1. **Scalability:**
 - o Computationally expensive for large datasets ($O(n^3)$ in the worst case).
2. **Sensitivity to Noise:**
 - o Sensitive to outliers, which can distort clustering results.
3. **Lack of Robustness:**
 - o Small changes in data can result in different dendrogram structures.

Program:

Here's a Python implementation using `scikit-learn`:

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import AgglomerativeClustering

from scipy.cluster.hierarchy import dendrogram, linkage

# Generate synthetic data

from sklearn.datasets import make_blobs

X, _ = make_blobs(n_samples=100, centers=3, random_state=42,
cluster_std=1.5)

# Perform Agglomerative Clustering
```

```

model = AgglomerativeClustering(n_clusters=3, linkage='ward')

labels = model.fit_predict(X)

# Plot clusters

plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', marker='o')

plt.title("Agglomerative Clustering")

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

plt.show()

# Dendrogram

Z = linkage(X, method='ward') # Perform hierarchical clustering

plt.figure(figsize=(10, 5))

dendrogram(Z)

plt.title("Dendrogram")

plt.xlabel("Data Points")

plt.ylabel("Distance")

plt.show()

```

PROGRAM 2

```

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import AgglomerativeClustering

from scipy.cluster.hierarchy import dendrogram, linkage

# Generate synthetic data

from sklearn.datasets import make_blobs

```

```

X, _ = make_blobs(n_samples=100, centers=3, random_state=42,
cluster_std=1.5)

# Perform Agglomerative Clustering

model = AgglomerativeClustering(n_clusters=3, linkage='ward')

labels = model.fit_predict(X)

# Plot clusters

plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', marker='o')

plt.title("Agglomerative Clustering")

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

plt.show()

# Dendrogram

Z = linkage(X, method='ward') # Perform hierarchical clustering

plt.figure(figsize=(10, 5))

dendrogram(Z)

plt.title("Dendrogram")

plt.xlabel("Data Points")

plt.ylabel("Distance")

plt.show()

```

Applications

1. **Biology:**
 - Grouping genes or proteins with similar functionalities.
2. **Document Clustering:**
 - Organizing documents with similar content.
3. **Image Segmentation:**
 - Dividing an image into regions based on color or texture.
4. **Market Segmentation:**

- Clustering customers based on purchasing behavior.

Agglomerative clustering is versatile and interpretable but computationally expensive for large datasets