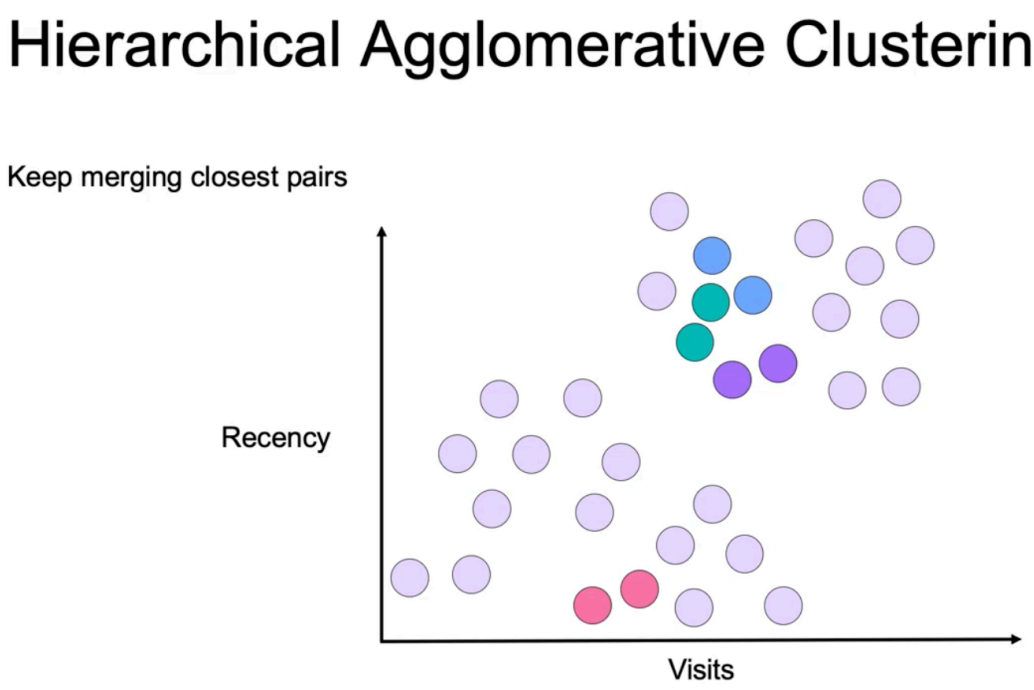
Module 3

Hierarchical Agglomerative Clustering

- The algorithm starts by identifying the closest pair of points based on a defined distance metric.
- Clusters can be formed by merging points or existing clusters, depending on their proximity.

Distance Metrics and Linkage Criteria

- The distance between points and clusters can be calculated using various methods, such as average distance or minimum distance.
- The choice of linkage criterion affects how clusters are formed and merged throughout the process.



Hierarchical Agglomerative Clustering: Hierarchical Linkage Types

Understanding Cluster Merging

- The algorithm merges clusters based on average distances, stopping when all distances exceed a defined threshold.

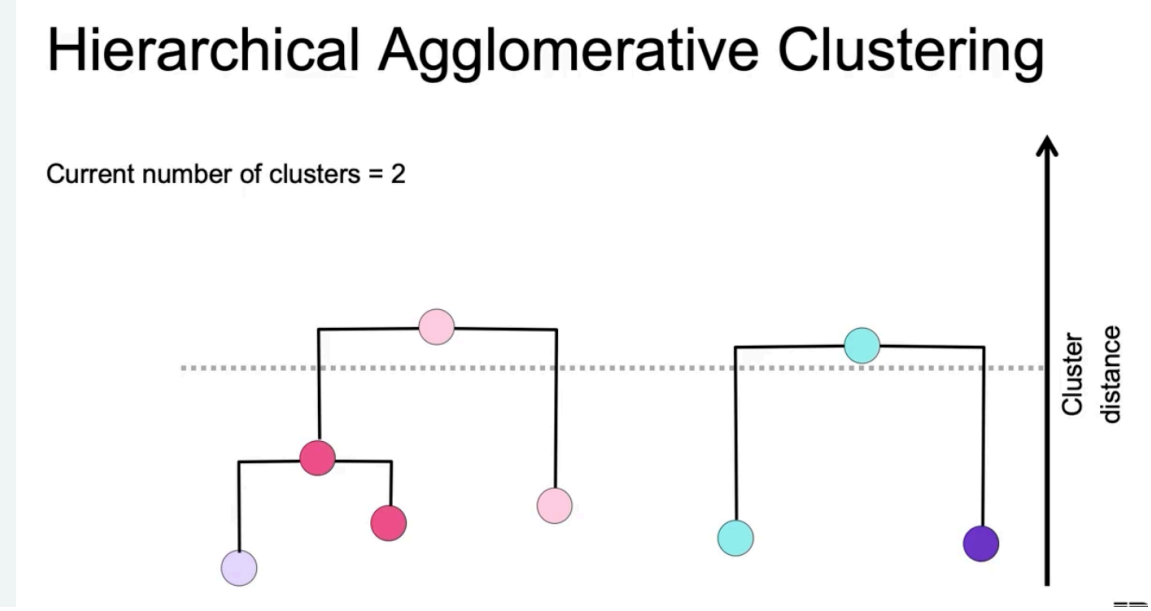
- As clusters merge, their average distances change, influencing the decision to continue merging.

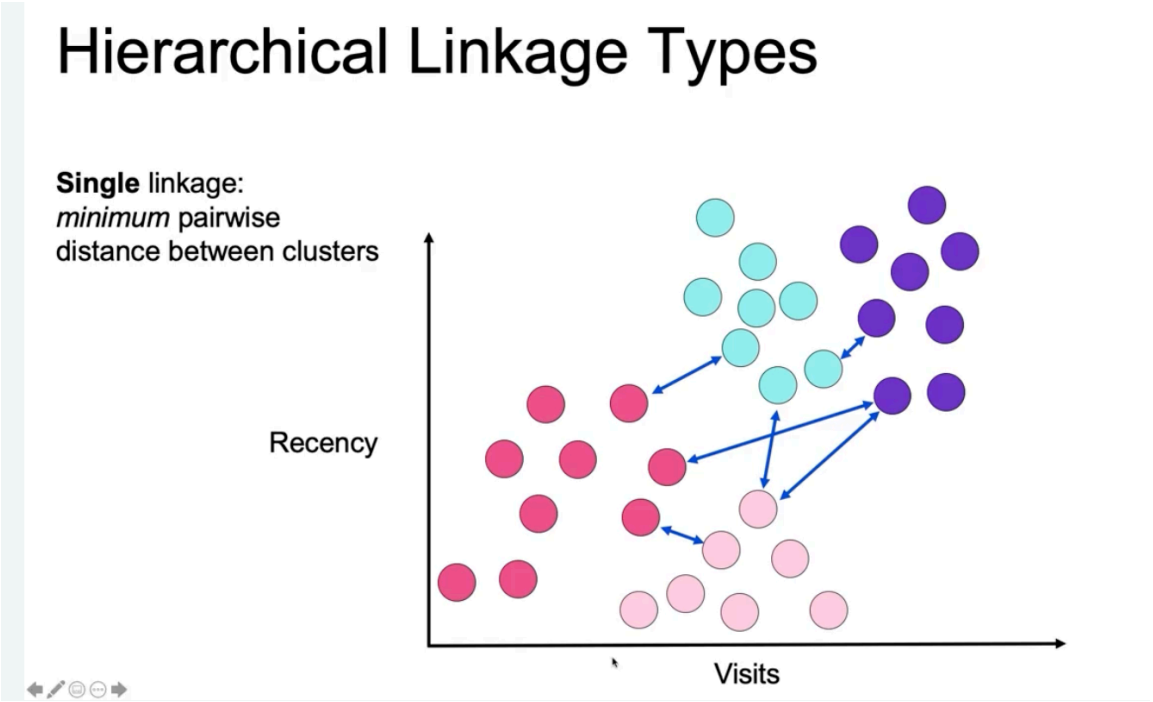
Linkage Types for Measuring Distances

- **Single Linkage**: Measures the minimum pairwise distance between clusters, ensuring clear separation but can be affected by outliers.
- **Complete Linkage**: Uses the maximum distance between points in clusters, better at handling noise but may disrupt larger clusters.

Average and Ward Linkage

- **Average Linkage**: Takes the average distance of points in clusters, balancing pros and cons of single and complete linkage.
- **Ward Linkage**: Minimizes inertia (sum of squared distances to centroids), similar to k-means, aiming for optimal cluster formation.





Applying Hierarchical Agglomerative Clustering

Hierarchical Agglomerative Clustering Overview

- The process begins by importing the AgglomerativeClustering class and creating an instance with specified hyperparameters.
- Users can set the number of clusters (e.g., three) and choose the distance metric, such as Euclidean distance.

Stopping Conditions and Linkage Types

- Stopping conditions can be based on a predetermined number of clusters or a threshold for minimum average cluster distances.
- Different linkage types are discussed, including single linkage, complete linkage, average linkage, and Ward's linkage, which optimizes inertia reduction.

DBSCAN

- DBSCAN identifies clusters of data rather than simply partitioning them, making it effective in datasets with noise.
- It operates on the principle that points in a cluster should be within a certain distance from each other.

Key Inputs for DBSCAN

- The algorithm requires defining a distance metric and an Epsilon value, which determines how close points must be to be considered part of the same cluster.
- N_clu (minimum samples) is another important input, indicating the minimum number of points required for a point to be classified as a core point.

Classification of Points

- Points can be classified as core points (having enough neighbors), density reachable points (connected to core points), or noise (not part of any cluster).
- Clusters are formed by connected core points and their density-reachable points, while noise points are excluded from clusters.

Understanding DBSCAN

- DBSCAN starts with a random point and defines a radius (epsilon) to identify neighboring points. If enough points are found within this radius, a cluster is formed.
- Points can be classified as core points (having enough neighbors), density-reachable points (not core but close to core points), or outliers (not part of any cluster).

Strengths and Weaknesses

- Strengths: DBSCAN does not require the number of clusters to be specified in advance, can handle noise, and is effective for arbitrary shapes.
- Weaknesses: It requires careful tuning of parameters, can struggle with varying cluster densities, and may be challenging in higher-dimensional spaces.

Implementation in Python

- The algorithm is implemented using the `DBSCAN` class from the `sklearn.cluster` library, where parameters like epsilon and minimum samples are set to define clusters.

Overall, the lecture provides a comprehensive overview of how DBSCAN works, its advantages and limitations, and a brief guide on its implementation.

Mean Shift

Mean Shift Clustering Algorithm

- The algorithm partitions points based on the densest point within a defined window, unlike k-means which uses the mean of points.
- It calculates a weighted mean around each point, giving more weight to points closer to the original point within the window.

Steps to Implement Mean Shift

- Choose a starting point and a window size, then calculate the weighted mean within that window.
- Shift the window to center around the new mean and repeat until convergence, grouping points that reach the same mode.

Strengths and Weaknesses

- Strengths: Model-free, does not require pre-defining the number of clusters, robust to outliers.
- Weaknesses: Results depend heavily on window size, selection of bandwidth can be challenging, and it can be slow for large datasets.

Python Implementation

- Import the mean shift class from sklearn, create an instance with a specified bandwidth, and fit it to the data to predict clusters.