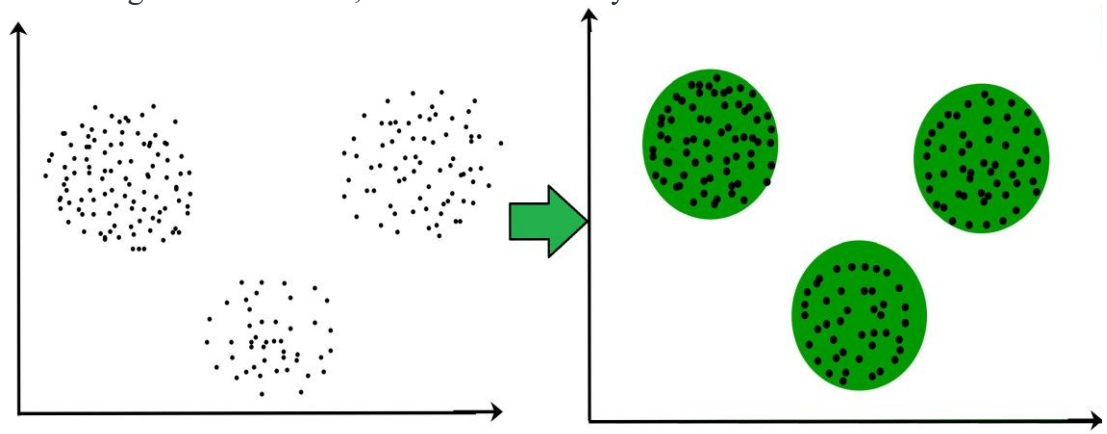
T. Y. B. Tech (Computer Engineering)

Academic Year -2025-2026 (Semester –V)

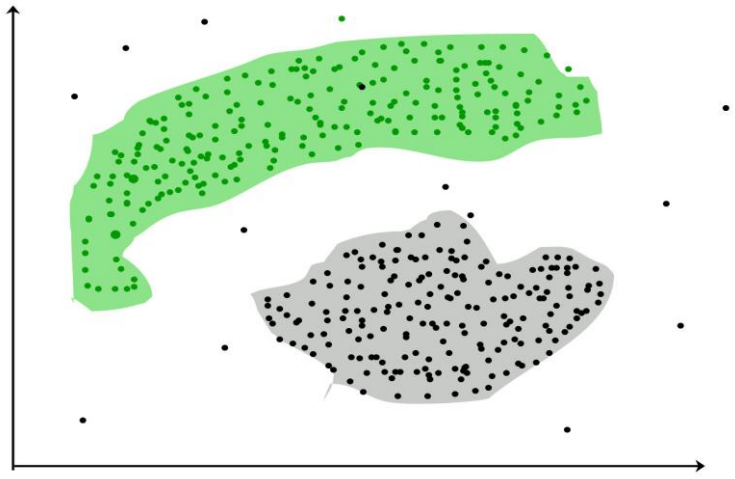
[CS3201T]:- Machine Learning

UNIT-V	**Unsupervised Learning: Clustering Techniques**	**08 Hours**
Clustering Techniques: K-Means, K-Medoids, Hierarchical (average linkage, Ward's method), Density-based (DBSCAN), Spectral Clustering, MST Clustering, BIRCH, CURE; Evaluation Metrics: Intrinsic (Silhouette Score, Davies–Bouldin Index), Extrinsic (Homogeneity, Completeness, Adjusted Rand Index); Elbow method; Outlier Detection, Applications. **Case Study**: Customer Segmentation and Fraud Detection in Retail		

Introduction to Clustering: It is basically a type of *unsupervised learning method*. An unsupervised learning method is a method in which we draw references from datasets consisting of input data without labeled responses. Generally, it is used as a process to find meaningful structure, explanatory underlying processes, generative features, and groupings inherent in a set of examples.
Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group and dissimilar to the data points in other groups. It is basically a collection of objects on the basis of similarity and dissimilarity between them.
For example The data points in the graph below clustered together can be classified into one single group. We can distinguish the clusters, and we can identify that there are 3 clusters in the below picture.



It is not necessary for clusters to be spherical as depicted below:

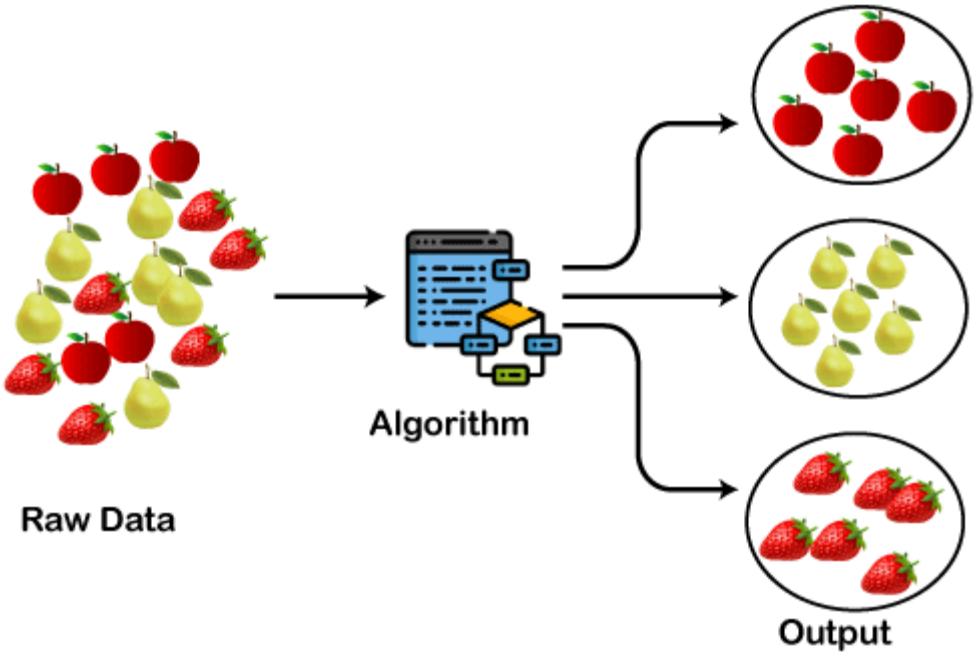


In short
Clustering or cluster analysis is a machine learning technique, which groups the unlabelled dataset. It can be defined as ***"A way of grouping the data points into different clusters, consisting of similar data points. The objects with the possible similarities remain in a group that has less or no similarities with another group."***
It does it by finding some similar patterns in the unlabelled dataset such as shape, size, color, behavior, etc., and divides them as per the presence and absence of those similar patterns.
It is an unsupervised learning method, hence no supervision is provided to the algorithm, and it deals with the unlabeled dataset.

After applying this clustering technique, each cluster or group is provided with a cluster-ID. ML system can use this id to simplify the processing of large and complex datasets.
The clustering technique is commonly used for **statistical data analysis.**
Example: Let's understand the clustering technique with the real-world example of Mall: When we visit any shopping mall, we can observe that the things with similar usage are grouped together. Such as the t-shirts are grouped in one section, and trousers are at other sections, similarly, at vegetable sections, apples, bananas, Mangoes, etc., are grouped in separate sections, so that we can easily find out the things. The clustering technique also works in the same way. Other examples of clustering are grouping documents according to the topic.
The clustering technique can be widely used in various tasks. Some most common uses of this technique are:

- Market Segmentation
- Statistical data analysis
- Social network analysis
- Image segmentation
- Anomaly detection, etc.

Apart from these general usages, it is used by the **Amazon** in its recommendation system to provide the recommendations as per the past search of products. **Netflix** also uses this technique to recommend the movies and web-series to its users as per the watch history.
The below diagram explains the working of the clustering algorithm. We can see the different fruits are divided into several groups with similar properties.



Clustering refers to grouping similar data points together, based on their attributes or features.
For example, if we have the income and expenditure for a set of people, we can divide them into the following groups:

- First – Earn high, spend high
- Second – Earn high, spend low
- Third – Earn low, spend low
- Fourth – Earn low, spend high

Each of these groups would hold a population with similar features and can be useful in pitching the relevant scheme/product to the group. Think of credit cards, car/property loans, and so on. In simple words:
The idea behind clustering is grouping data points together, such that each individual cluster holds the most similar points.

Why Clustering?
Clustering is very much important as it determines the intrinsic grouping among the unlabelled data present. There are no criteria for good clustering. It depends on the user, and what criteria they may use which satisfy their need. For instance, we could be interested in finding representatives for homogeneous groups (data reduction), finding “natural clusters” and describing their unknown properties (“natural” data types), in finding useful and suitable groupings (“useful” data classes) or in finding unusual data objects (outlier detection). This

algorithm must make some assumptions that constitute the similarity of points and each assumption make different and equally valid clusters.

Types of Clustering

Several approaches to clustering exist. Each approach is best suited to a particular data distribution. Below is a short discussion of four common approaches, focusing on centroid-based clustering using k-means.

1. Centroid-based Clustering

Centroid-based clustering organizes the data into non-hierarchical clusters, in contrast to hierarchical clustering defined below. k-means is the most widely-used centroid-based clustering algorithm. Centroid-based algorithms are efficient but sensitive to initial conditions and outliers. This course focuses on k-means because it is an efficient, effective, and simple clustering algorithm.
Example: K-Means clustering, K-Mode clustering

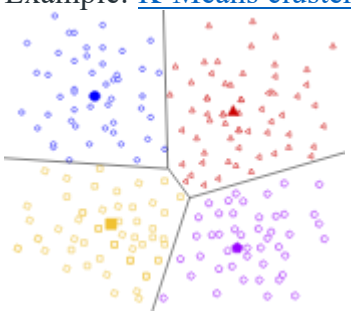


Figure 1: Example of centroid-based clustering.

2. Density-based Clustering

Density-based clustering connects areas of high example density into clusters. This allows for arbitrary-shaped distributions as long as dense areas can be connected. These algorithms have difficulty with data of varying densities and high dimensions. Further, by design, these algorithms do not assign outliers to clusters.
Example: DBSCAN(Density-Based Spatial Clustering of Applications with Noise)

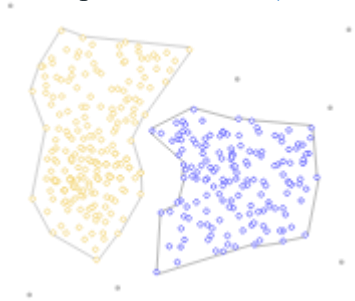


Figure 2: Example of density-based clustering.

3. Distribution-based Clustering

This clustering approach assumes data is composed of distributions, such as **Gaussian distributions**. In Figure 3, the distribution-based algorithm clusters data into three Gaussian distributions. As distance from the distribution's center increases, the probability that a point belongs to the distribution decreases. The bands show that decrease in probability. When you do not know the type of distribution in your data, you should use a different algorithm.
Example: Gaussian Mixture Models (GMM)

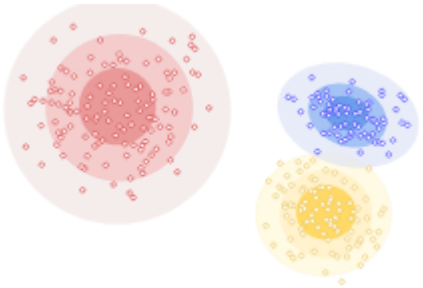


Figure 3: Example of distribution-based clustering.

4. Hierarchical Clustering/ connectivity-based clustering algorithms

Hierarchical clustering creates a tree of clusters. Hierarchical clustering, not surprisingly, is well suited to hierarchical data, such as taxonomies. See *Comparison of 61 Sequenced Escherichia coli Genomes* by Oksana Lukjancenko, Trudy Wassenaar & Dave Ussery for an example. In addition, another advantage is that any number of clusters can be chosen by cutting the tree at the right level.

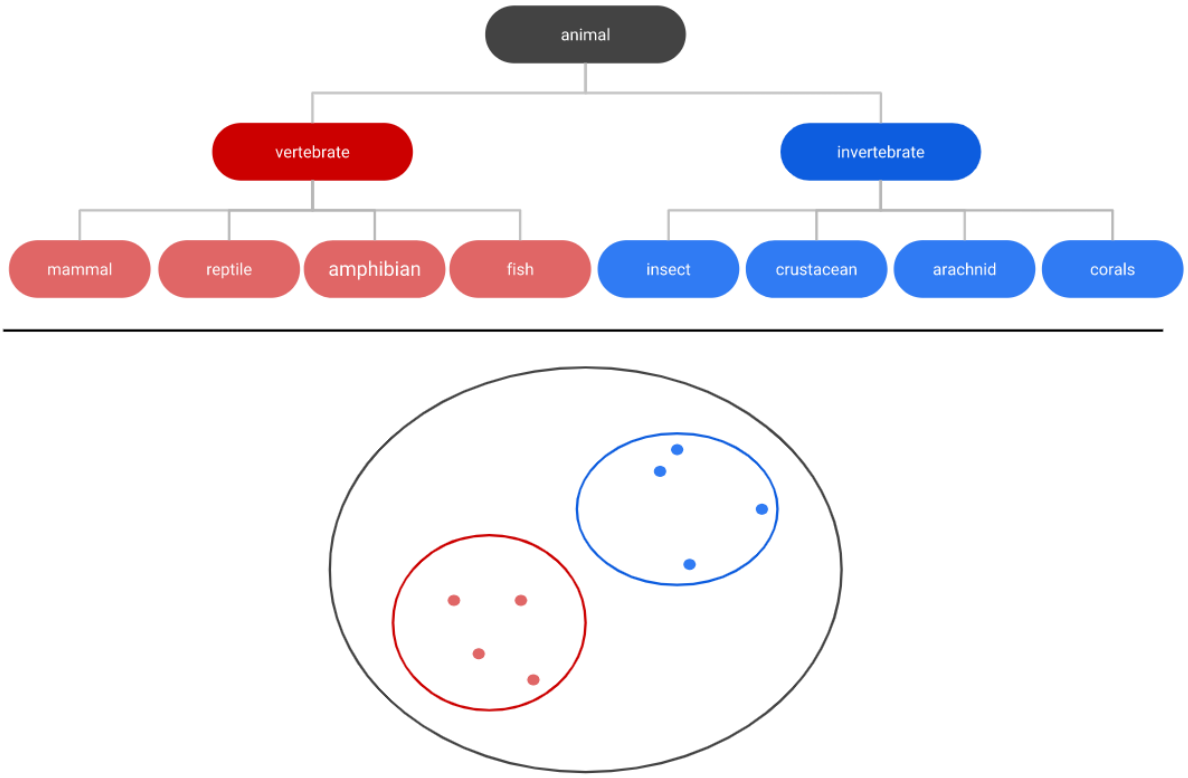


Figure 4: Example of a hierarchical tree clustering animals.

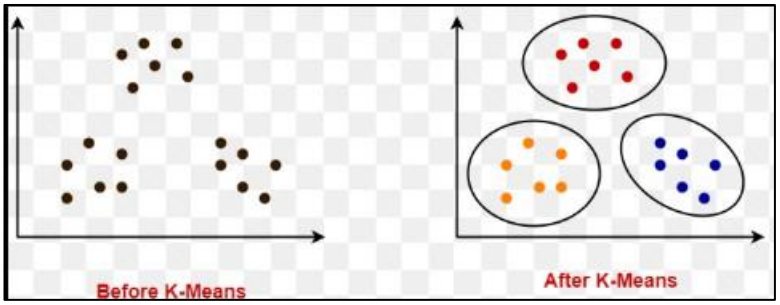
It is divided into two category

- **Agglomerative** (bottom-up *approach*)
- **Divisive** (top-down *approach*)

Clustering Techniques:

K-Means:

K-Means Clustering is an unsupervised machine learning algorithm that helps group data points into clusters based on their inherent similarity. Unlike supervised learning, where we train models using labeled data, K-Means is used when we have data that is not labeled and the goal is to uncover hidden patterns or structures. For example, an online store can use K-Means to segment customers into groups like "Budget Shoppers," "Frequent Buyers," and "Big Spenders" based on their purchase history.



Working of K-Means Clustering

Suppose we are given a data set of items with certain features and values for these features like a vector. The task is to categorize those items into groups. To achieve this we will use the K-means algorithm. "k*k*" represents the number of groups or clusters we want to classify our items into.

The algorithm will categorize the items into "k*k*" groups or clusters of similarity. To calculate that similarity we will use the Euclidean distance as a measurement. The algorithm works as follows:

1. **Initialization:** We begin by randomly selecting k cluster centroids.
2. **Assignment Step:** Each data point is assigned to the nearest centroid, forming clusters.
3. **Update Step:** After the assignment, we recalculate the centroid of each cluster by averaging the points within it.
4. **Repeat:** This process repeats until the centroids no longer change or the maximum number of iterations is reached.

The goal is to partition the dataset into k*k* clusters such that data points within each cluster are more similar to each other than to those in other clusters.

Why Use K-Means Clustering?

K-Means is popular in a wide variety of applications due to its simplicity, efficiency and effectiveness. Here's why it is widely used:

1. **Data Segmentation:** One of the most common uses of K-Means is segmenting data into distinct groups. For example, businesses use K-Means to group customers based on behavior, such as purchasing patterns or website interaction.

2. **Image Compression**: K-Means can be used to reduce the complexity of images by grouping similar pixels into clusters, effectively compressing the image. This is useful for image storage and processing.
3. **Anomaly Detection:** K-Means can be applied to detect anomalies or outliers by identifying data points that do not belong to any of the clusters.
4. **Document Clustering:** In natural language processing (NLP), K-Means is used to group similar documents or articles together. It's often used in applications like recommendation systems or news categorization.
5. **Organizing Large Datasets:** When dealing with large datasets, K-Means can help in organizing the data into smaller, more manageable chunks based on similarities, improving the efficiency of data analysis.

Challenges with K-Means Clustering

K-Means algorithm has the following limitations:

- **Choosing the Right Number of Clusters** (k*k*): One of the biggest challenges is deciding how many clusters to use.
- **Sensitive to Initial Centroids:** The final clusters can vary depending on the initial random placement of centroids.
- **Non-Spherical Clusters:** K-Means assumes that the clusters are spherical and equally sized. This can be a problem when the actual clusters in the data are of different shapes or densities.
- **Outliers**: K-Means is sensitive to outliers, which can distort the centroid and, ultimately, the clusters.

K-Medoids:

K-Medoids (also called Partitioning Around Medoid) algorithm was proposed in 1987 by Kaufman and Rousseeuw. A medoid can be defined as a point in the cluster, whose dissimilarities with all the other points in the cluster are minimum. The dissimilarity of the medoid(Ci) and object(Pi) is calculated by using E = |Pi - Ci|

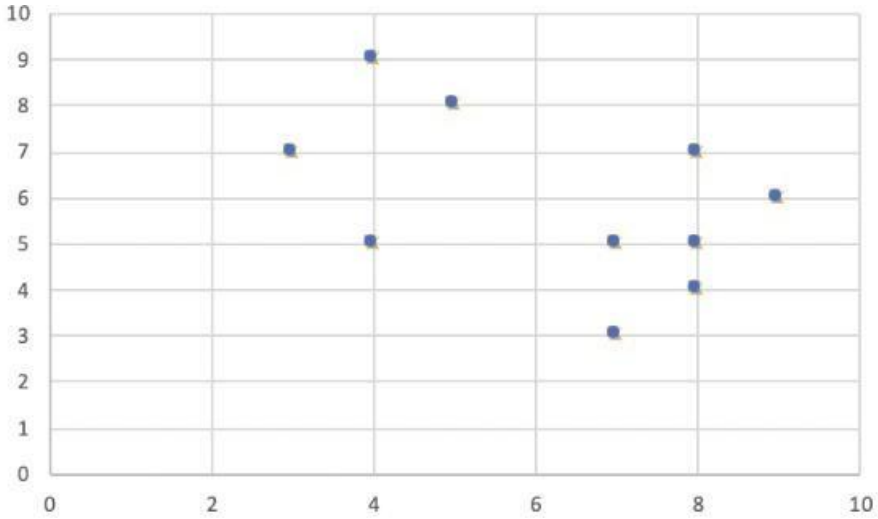
Algorithm:

1. Initialize: select *k* random points out of the *n* data points as the medoids.
2. Associate each data point to the closest medoid by using any common distance metric methods.
3. While the cost decreases: For each medoid m, for each data o point which is not a medoid:
    - Swap m and o, associate each data point to the closest medoid, and recompute the cost.
    - If the total cost is more than that in the previous step, undo the swap.

Let's consider the following example: If a graph is drawn using the above data points, we obtain the following:

	X	Y
0	8	7
1	3	7
2	4	9
3	9	6
4	8	5
5	5	8
6	7	3
7	8	4
8	7	5
9	4	5

Step 1: Let the randomly selected 2 medoids, so select k = 2, and let **C1 -(4, 5)** and **C2 -(8, 5)** are the two medoids.



Step 2: Calculating cost. The dissimilarity of each non-medoid point with the medoids is calculated and tabulated:

	X	Y	Dissimilarity from C1	Dissimilarity from C2
0	8	7	6	2
1	3	7	3	7
2	4	9	4	8
3	9	6	6	2
4	8	5	-	-
5	5	8	4	6
6	7	3	5	3
7	8	4	5	1
8	7	5	3	1
9	4	5	-	-

Here we have used Manhattan distance formula to calculate the distance matrices between medoid and non-medoid points. That formula tell that **Distance = |X1-X2| + |Y1-Y2|.**

Each point is assigned to the cluster of that medoid whose dissimilarity is less. Points 1, 2, and 5 go to cluster C1 and 0, 3, 6, 7, 8 go to cluster C2. The Cost = (3 + 4 + 4) + (3 + 1 + 1 + 2 + 2) = 20

Step 3: randomly select one non-medoid point and recalculate the cost. Let the randomly selected point be (8, 4). The dissimilarity of each non-medoid point with the medoids - C1 (4, 5) and C2 (8, 4) is calculated and tabulated.

	X	Y	Dissimilarity from C1	Dissimilarity from C2
0	8	7	6	3
1	3	7	3	8
2	4	9	4	9
3	9	6	6	3
4	8	5	4	1
5	5	8	4	7
6	7	3	5	2
7	8	4	-	-
8	7	5	3	2
9	4	5	-	-

Each point is assigned to that cluster whose dissimilarity is less. So, points 1, 2, and 5 go to cluster C1 and 0, 3, 6, 7, 8 go to cluster C2. The New cost = (3 + 4 + 4) + (2 + 2 + 1 + 3 + 3) = 22 Swap Cost = New Cost - Previous Cost = 22 - 20 and **2 >0** As the swap cost is not less than zero, we undo the swap. Hence (4, 5) and (8, 5) are the final medoids. The clustering would be in the following way The **time complexity** is $O(k*(n-k)^2)$ $O(k*(n-k)^2)$

Advantages:

1. It is simple to understand and easy to implement.
2. K-Medoid Algorithm is fast and converges in a fixed number of steps.
3. PAM is less sensitive to outliers than other partitioning algorithms.

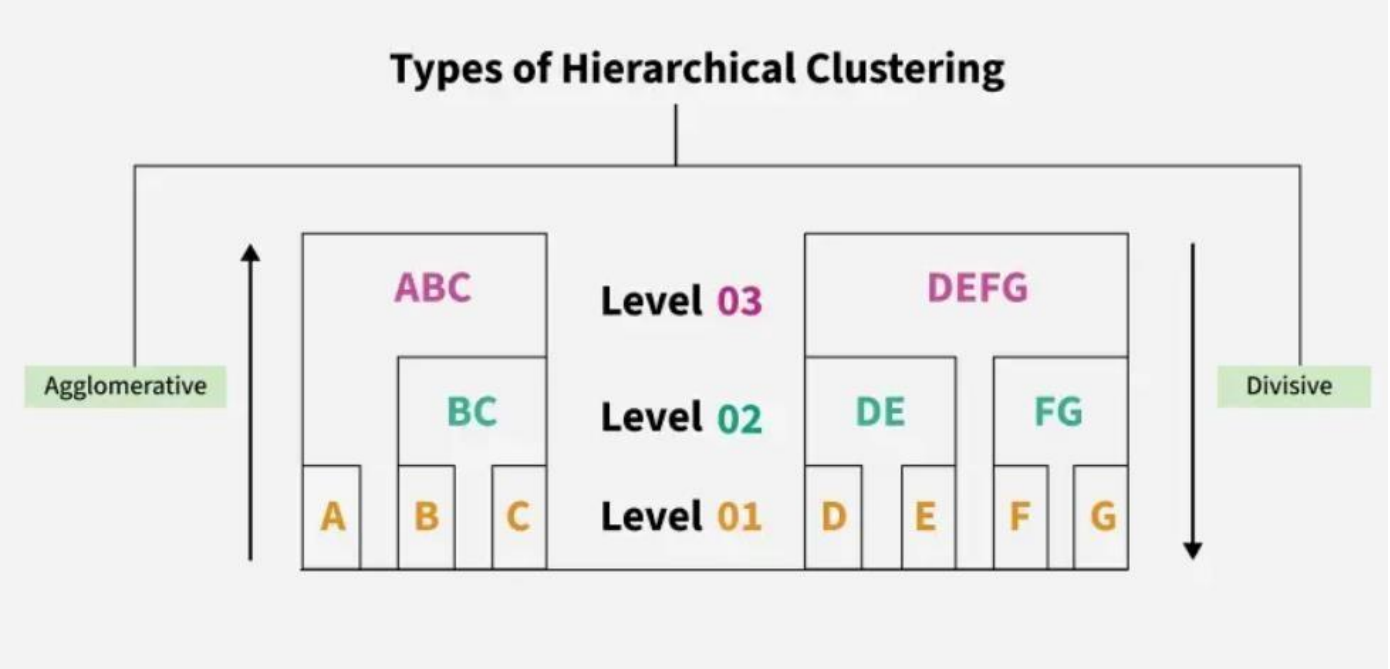
Disadvantages:

1. The main disadvantage of K-Medoid algorithms is that it is not suitable for clustering non- spherical (arbitrarily shaped) groups of objects. This is because it relies on minimizing the distances between the non-medoid objects and the medoid (the cluster center) - briefly, it uses compactness as clustering criteria instead of connectivity.
2. It may obtain different results for different runs on the same dataset because the first k medoids are chosen randomly.

Hierarchical:

Hierarchical clustering is an unsupervised learning technique used to group similar data points into clusters by building a hierarchy (tree-like structure). Unlike flat clustering like k-means hierarchical clustering does not require specifying the number of clusters in advance.

The algorithm builds clusters step by step either by progressively merging smaller clusters or by splitting a large cluster into smaller ones. The process is often visualized using a dendrogram, which helps to understand data similarity.



Imagine we have four fruits with different weights: an apple (100g), a banana (120g), a cherry (50g) and a grape (30g). Hierarchical clustering starts by treating each fruit as its own group.

- Start with each fruit as its own cluster.
- Merge the closest items: grape (30g) and cherry (50g) are grouped first.
- Next, apple (100g) and banana (120g) are grouped.
- Finally, these two clusters merge into one.

Finally all the fruits are merged into one large group, showing how hierarchical clustering progressively combines the most similar data points.

Types of Hierarchical Clustering

Now we understand the basics of hierarchical clustering. There are two main types of hierarchical clustering.

1. Agglomerative Clustering
2. Divisive clustering

1. Hierarchical Agglomerative Clustering

It is also known as the bottom-up approach or hierarchical agglomerative clustering (HAC). Bottom-up algorithms treat each data as a singleton cluster at the outset and then successively agglomerate pairs of clusters until all clusters have been merged into a single cluster that contains all data.

Workflow for Hierarchical Agglomerative clustering

1. **Start with individual points:** Each data point is its own cluster. For example if we have 5 data points we start with 5 clusters each containing just one data point.
2. **Calculate distances between clusters**: Calculate the distance between every pair of clusters. Initially since each cluster has one point this is the distance between the two data points.
3. **Merge the closest clusters**: Identify the two clusters with the smallest distance and merge them into a single cluster.
4. **Update distance matrix**: After merging we now have one less cluster. Recalculate the distances between the new cluster and the remaining clusters.
5. **Repeat steps 3 and 4**: Keep merging the closest clusters and updating the distance matrix until we have only one cluster left.
6. **Create a dendrogram**: As the process continues we can visualize the merging of clusters using a tree-like diagram called a dendrogram. It shows the hierarchy of how clusters are merged.

2. Hierarchical Divisive clustering

Divisive clustering is also known as a top-down approach. Top-down clustering requires a method for splitting a cluster that contains the whole data and proceeds by splitting clusters recursively until individual data have been split into singleton clusters.

Workflow for Hierarchical Divisive clustering :

1. **Start with all data points in one cluster:** Treat the entire dataset as a single large cluster.
2. **Split the cluster**: Divide the cluster into two smaller clusters. The division is typically done by finding the two most dissimilar points in the cluster and using them to separate the data into two parts.
3. **Repeat the process**: For each of the new clusters, repeat the splitting process: Choose the cluster with the most dissimilar points and split it again into two smaller clusters.
4. **Stop when each data point is in its own cluster**: Continue this process until every data point is its own cluster or the stopping condition (such as a predefined number of clusters) is met.

Types of Linkages in Hierarchical Clustering

Hierarchical clustering is used to group similar data points and organise data in a tree-like structure. Key part of this process is **linkage** which **calculates the distance between clusters before they are merged or divided**. Different types of linkage is used measure this distance differently. In this article, we'll look at different linkage methods and see how they affect the cluster formation.

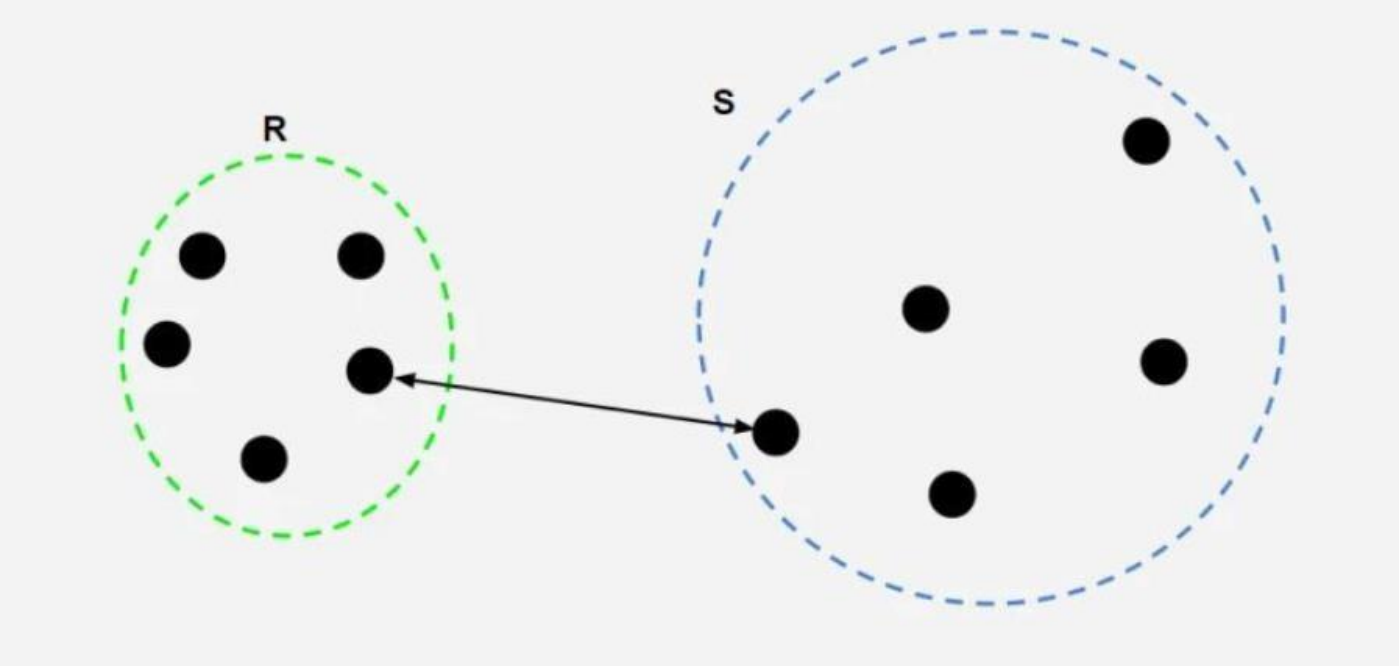
1. Single Linkage

For two clusters **R** and **S** the **single linkage returns the minimum distance between two points**. This method creates long, chain-like clusters because it is **sensitive to outliers** and can **connect clusters based on a very small number of close points**.

$L(R,S)=\min(D(i,j)), i \epsilon R, j \epsilon S$ $L(R,S)=min(D(i,j)),\ i \epsilon R, j \epsilon S$

where

D(i, j): Distance function between points **i** and **j**.



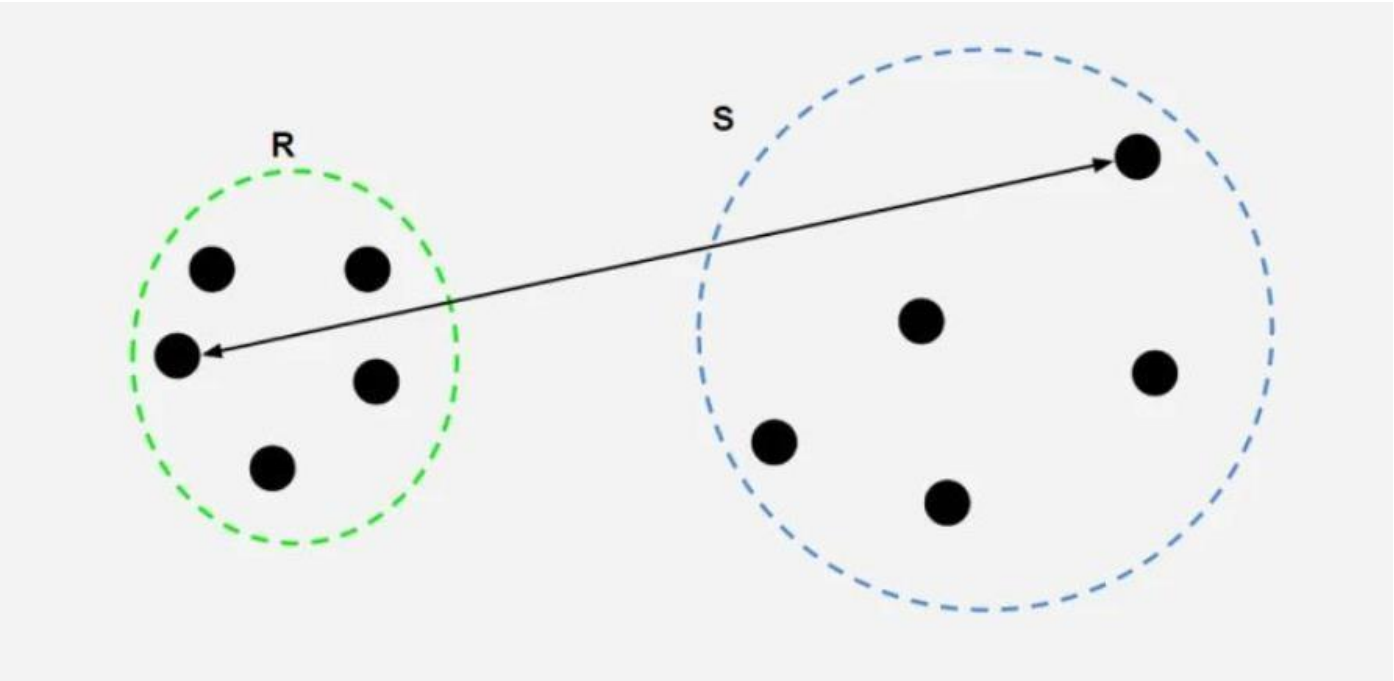
2. Complete Linkage

For two clusters **R** and **S** the **complete linkage returns the maximum distance between two points**. It tends to create compact and spherical clusters because it is **more sensitive to outliers and tries to make sure that the clusters are not too far**.

$L(R,S)=\max(D(i,j)), i \in R,\ j \in S$ $L(R,S)=\max(D(i,j)),\ i \in R,\ j \in S$

where

- **D(i, j)**: Distance function between points **i** and **j**.



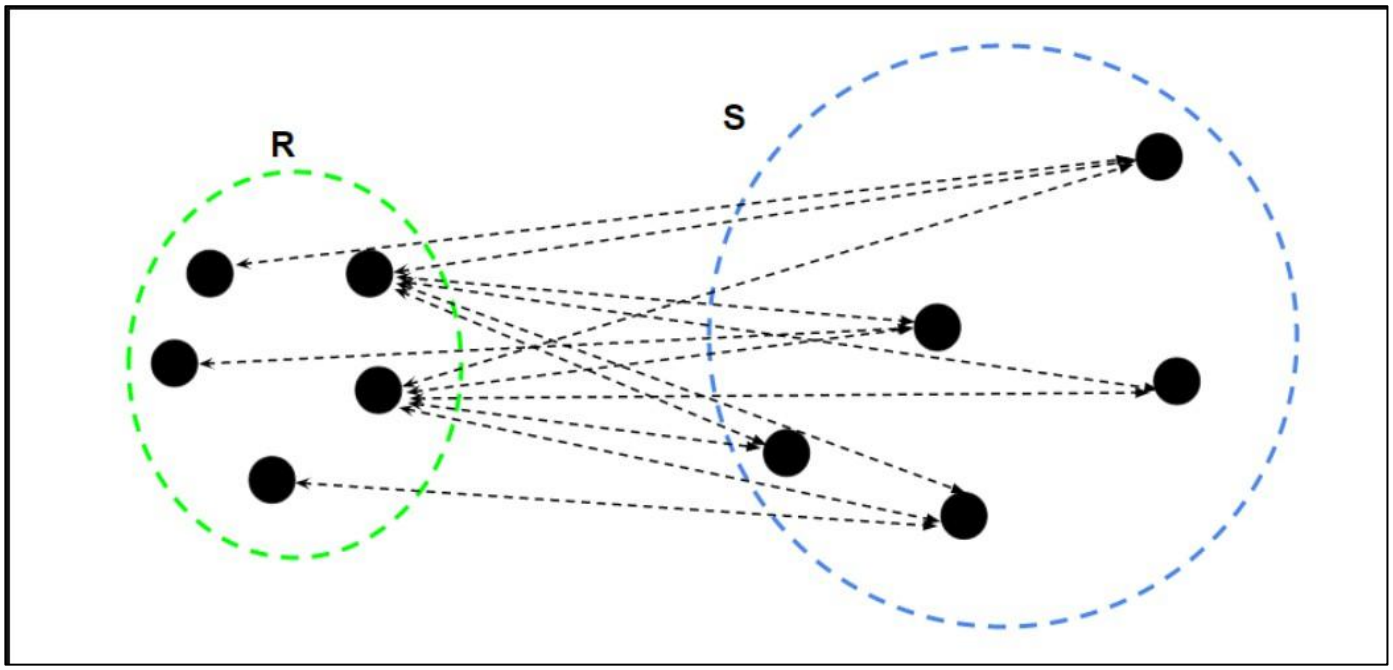
3. Average Linkage

It returns the **average distance between all pairs of points from two clusters**. This method maintain a **balance between single and complete linkage** by considering all pairs of points not just the closest or farthest point. It usually **results in clusters that are moderately compact**.

$$L(R,S) = \frac{1}{n_R \times n_S} \sum_{i=1}^{n_R} \sum_{j=1}^{n_S} D(i,j), i \in R, j \in S$$

where

- n_R : Number of data-points in R
- n_S : Number of data-points in S



Steps in Hierarchical Clustering using Average Linkage

1. At the beginning, each data point is treated as its own individual cluster.
2. The distances between all clusters are calculated using the **average linkage method**, which considers the average distance between all pairs of points in two clusters.
3. The two clusters with the **smallest average distance** are then merged into a single cluster.
4. After merging, the distances between the new cluster and the remaining clusters are updated, and the process is repeated.
5. This procedure continues until all points are combined into one single cluster. Finally, a **dendrogram** is used to decide the appropriate number of clusters.

Advantages of Average Linkage

- It produces **balanced and stable clusters**, which are not stretched out or elongated like those formed by single linkage.
- It is **less sensitive to outliers** compared to both single linkage and complete linkage methods.

- It is capable of capturing both **compactness** (how close the points within a cluster are) and **separation** (how far clusters are from each other).

Disadvantages of Average Linkage

- It is **computationally expensive** for large datasets, since it requires calculating many pairwise distances.
- It may sometimes merge clusters that are **not truly compact**, which can affect accuracy.
- The results are still **sensitive to the choice of distance metric**, such as Euclidean or Manhattan distance.

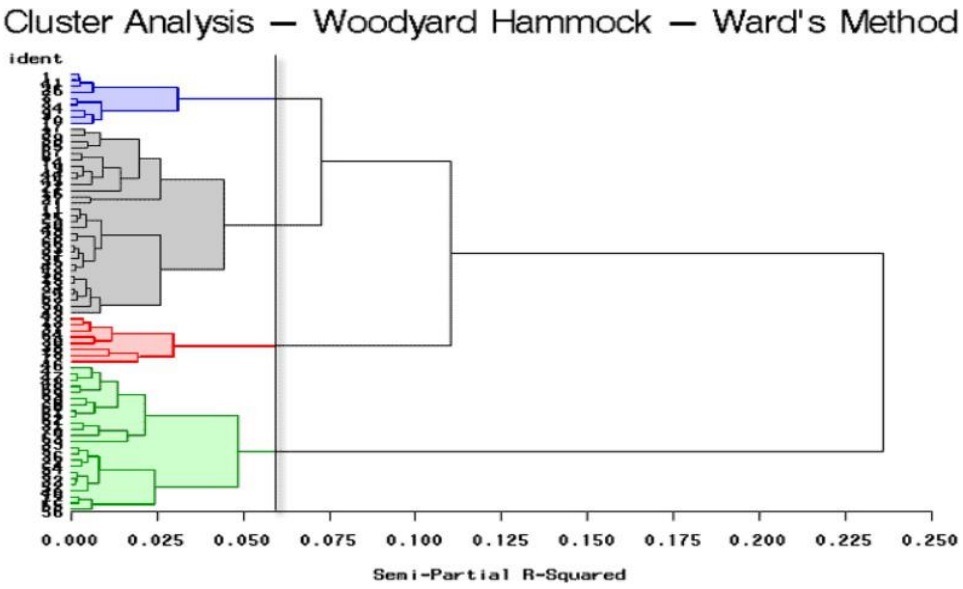
Applications of Average Linkage

- It is widely used in **gene expression analysis** in bioinformatics, where relationships between genes are studied.
- It can be applied in **customer segmentation**, where groups of customers with similar behaviours are identified.
- It is also used in **document clustering** for text mining, where documents with similar content are grouped together.

Ward's Method (Minimum variance method):

What is Ward's Method?

Ward's method (a.k.a. Minimum variance method or Ward's Minimum Variance Clustering Method) is an alternative to single-link clustering. Popular in fields like linguistics, it's liked because it usually creates compact, even-sized clusters. Like most other clustering methods, Ward's method is computationally intensive. However, Ward's has significantly fewer computations than other methods. The drawback is this usually results in less than optimal clusters. That said, the resulting clusters are usually good enough for most purposes.



Like other clustering

methods, Ward's method starts with n clusters, each containing a single object. These n clusters are combined to make one cluster containing all objects. At each step, the process makes a new cluster that minimizes variance, measured by an index called E (also called the sum of squares index).

At each step, the following calculations are made to find E:

Find the mean of each cluster.

1. Calculate the distance between each object in a particular cluster, and that cluster's mean.
2. Square the differences from Step 2.
3. Sum (add up) the squared values from Step 3.
4. Add up all the sums of squares from Step 4.

In order to select a new cluster at each step, every possible combination of clusters must be considered. This entire cumbersome procedure makes it practically impossible to perform by hand, making a computer a necessity for most data sets containing more than a handful of data points.

Density-based(DBSCAN):

DBSCAN is a density-based clustering algorithm that groups data points that are closely packed together and marks outliers as noise based on their density in the feature space. It identifies clusters as dense regions in the data space separated by areas of lower density. Unlike K-Means or hierarchical clustering which assumes clusters are compact and spherical, DBSCAN perform well in handling real- world data irregularities such as:

- **Arbitrary-Shaped Clusters**: Clusters can take any shape not just circular or convex.
- **Noise and Outliers**: It effectively identifies and handles noise points without assigning them to any cluster.

The figure above shows a data set with clustering algorithms: K-Means and Hierarchical handling compact, spherical clusters with varying noise tolerance while DBSCAN manages arbitrary-shaped clusters and noise handling.

Key Parameters in DBSCAN

1. eps: This defines the radius of the neighborhood around a data point. If the distance between two points is less than or equal to eps they are considered neighbors. A common method to determine eps is by analyzing the k-distance graph. Choosing the right eps is important:

- If eps is too small most points will be classified as noise.
- If eps is too large clusters may merge and the algorithm may fail to distinguish between them.

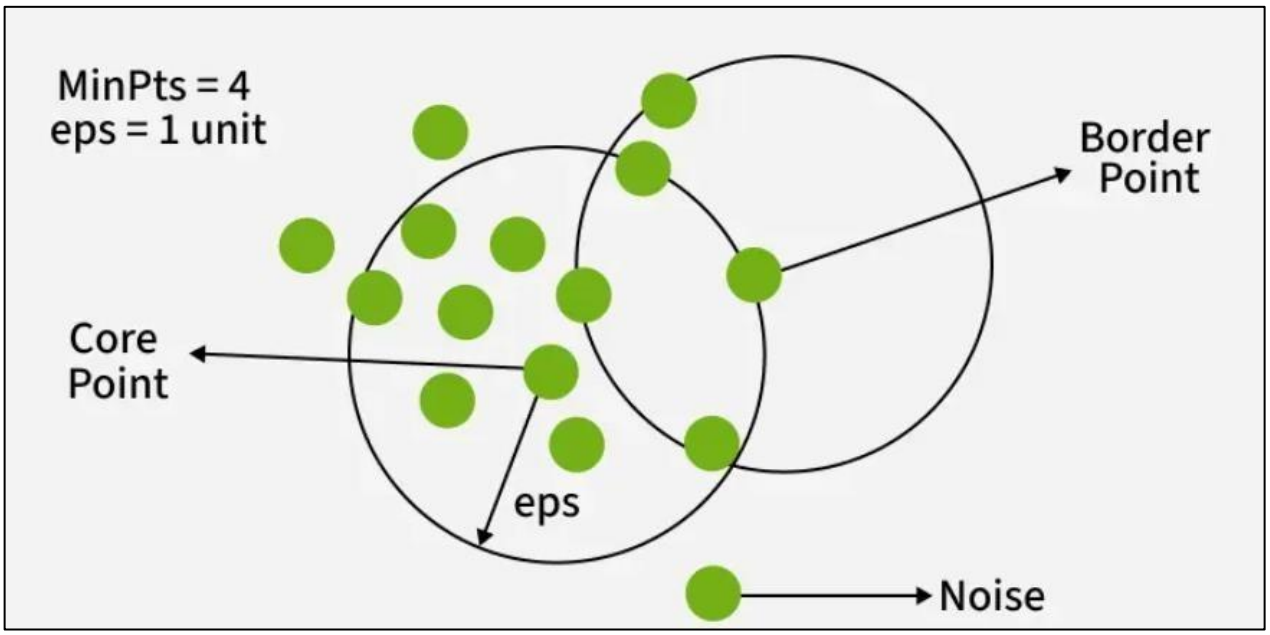
2. MinPts: This is the minimum number of points required within the **eps** radius to form a dense region. A general rule of thumb is to set MinPts >= D+1 where **D** is the number of dimensions in the dataset. *For most cases a minimum value of **MinPts = 3** is recommended.*

How Does DBSCAN Work?

DBSCAN works by categorizing data points into three types:

1. Core points which have a sufficient number of neighbors within a specified radius (eplison)
2. Border points which are near core points but lack enough neighbors to be core points themselves
3. Noise points which do not belong to any cluster.

By iteratively expanding clusters from core points and connecting density-reachable points, DBSCAN forms clusters without relying on rigid assumptions about their shape or size.



Steps in the DBSCAN Algorithm

1. **Identify Core Points**: For each point in the dataset count the number of points within its eps neighborhood. If the count meets or exceeds MinPts mark the point as a core point.
2. **Form Clusters**: For each core point that is not already assigned to a cluster create a new cluster. Recursively find all density-connected points i.e points within the eps radius of the core point and add them to the cluster.
3. **Density Connectivity**: Two points a and b are density-connected if there exists a chain of points where each point is within the eps radius of the next and at least one point in the chain is a core point. This chaining process ensures that all points in a cluster are connected through a series of dense regions.
4. **Label Noise Points**: After processing all points any point that does not belong to a cluster is labeled as noise.

When to use DBSCAN?

Now that we've seen how DBSCAN works and compared it to K-Means, let's see when DBSCAN is the right choice for our clustering needs. The unique properties of DBSCAN make it particularly well-suited for certain types of data and problem domains.

Complex cluster shapes

Building on our previous comparison, DBSCAN truly shines when dealing with non-globular cluster shapes. If your data forms arbitrary patterns like the half-moons we explored earlier, DBSCAN is likely to outperform traditional algorithms like K-Means.
For example, in geographical analysis, natural formations like river systems or urban sprawl often form irregular shapes that DBSCAN can effectively identify.

Unknown number of clusters

One of DBSCAN's key advantages is its ability to determine the number of clusters automatically. This is particularly useful in exploratory data analysis where you might not have prior knowledge about the underlying structure of your data.
Consider a market segmentation problem: you might not know in advance how many distinct customer groups exist. DBSCAN can help uncover these segments without requiring you to guess the number of clusters.

Datasets with noise

DBSCAN's approach to handling noise points makes it robust to outliers. This is crucial in many real- world datasets where measurement errors or anomalies are common.
For instance, in anomaly detection systems for network security, DBSCAN can effectively separate normal network traffic patterns from potential security threats.

Varying cluster densities

Unlike K-Means, which assume clusters of similar density, DBSCAN can identify clusters of varying densities. This is particularly useful in scenarios where some groups in your data are more tightly packed than others.

An example could be analyzing galaxy distributions in astronomy, where different regions of space have varying densities of celestial objects.
While DBSCAN is powerful, it's important to be aware of its limitations:

1. **Parameter Sensitivity**: As we discussed earlier, choosing appropriate values for ε and MinPts is crucial. Poor choices can lead to suboptimal clustering results.
2. **High-Dimensional Data**: DBSCAN's performance can degrade with high-dimensional data due to the "curse of dimensionality."
3. **Varying Densities**: While DBSCAN can handle clusters of different densities, extremely varying densities in the same dataset can still pose challenges.
4. **Scalability**: For very large datasets, DBSCAN might be computationally expensive compared to algorithms like K-Means.

Spectral Clustering:

In the clustering algorithm that we have studied before we used compactness(distance) between the data points as a characteristic to cluster our data points. However, we can also use connectivity between the data point as a feature to cluster our data points. Using connectivity we can cluster two data points into the same clusters even if the distance between the two data points is larger.

Spectral Clustering

Spectral Clustering is a variant of the clustering algorithm that uses the connectivity between the data points to form the clustering. It uses eigenvalues and eigenvectors of the data matrix to forecast the data into lower dimensions space to cluster the data points. It is based on the idea of a graph representation of data where the data point are represented as nodes and the similarity between the data points are represented by an edge.

Steps performed for spectral Clustering

Building the Similarity Graph Of The Data: This step builds the Similarity Graph in the form of an adjacency matrix which is represented by A. The adjacency matrix can be built in the following manners:

- **Epsilon-neighbourhood Graph:** A parameter epsilon is fixed beforehand. Then, each point is connected to all the points which lie in its epsilon-radius. If all the distances between any two points are similar in scale then typically the weights of the edges ie the distance between the two points are not stored since they do not provide any additional information. Thus, in this case, the graph built is an undirected and unweighted graph.
- **K-Nearest Neighbours** A parameter k is fixed beforehand. Then, for two vertices u and v, an edge is directed from u to v only if v is among the k-nearest neighbours of u. Note that this leads to the formation of a weighted and directed graph because it is not always the case that for each u having v as one of the k-nearest neighbours, it will be the same case for v having u among its k- nearest neighbours. To make this graph undirected, one of the following approaches is followed:-
    1. Direct an edge from u to v and from v to u if either v is among the k-nearest neighbours of u **OR** u is among the k-nearest neighbours of v.

2. Direct an edge from u to v and from v to u if v is among the k-nearest neighbours of u **AND** u is among the k-nearest neighbours of v.

3. **Fully-Connected Graph:** To build this graph, each point is connected with an undirected edge-weighted by the distance between the two points to every other point. Since this approach is used to model the local neighbourhood relationships thus typically the Gaussian similarity metric is used to calculate the distance.

Projecting the data onto a lower Dimensional Space: This step is done to account for the possibility that members of the same cluster may be far away in the given dimensional space. Thus the dimensional space is reduced so that those points are closer in the reduced dimensional space and thus can be clustered together by a traditional clustering algorithm. It is done by computing the **Graph Laplacian Matrix**.

Properties:

1. **Assumption-Less:** This clustering technique, unlike other traditional techniques do not assume the data to follow some property. Thus this makes this technique to answer a more-generic class of clustering problems.

2. **Ease of implementation and Speed:** This algorithm is easier to implement than other clustering algorithms and is also very fast as it mainly consists of mathematical computations.

3. **Not-Scalable:** Since it involves the building of matrices and computation of eigenvalues and eigenvectors it is time-consuming for dense datasets.

4. **Dimensionality Reduction**: The algorithm uses eigenvalue decomposition to reduce the dimensionality of the data, making it easier to visualize and analyze.

5. **Cluster Shape**: This technique can handle non-linear cluster shapes, making it suitable for a wide range of applications.

6. **Noise Sensitivity**: It is sensitive to noise and outliers, which may affect the quality of the resulting clusters.

7. **Number of Clusters:** The algorithm requires the user to specify the number of clusters beforehand, which can be challenging in some cases.

8. **Memory Requirements:** The algorithm requires significant memory to store the similarity matrix, which can be a limitation for large datasets.

Advantages of Spectral Clustering:

1. Scalability: Spectral clustering can handle large datasets and high-dimensional data, as it reduces the dimensionality of the data before clustering.

2. Flexibility: Spectral clustering can be applied to non-linearly separable data, as it does not rely on traditional distance-based clustering methods.

3. Robustness: Spectral clustering can be more robust to noise and outliers in the data, as it considers the global structure of the data, rather than just local distances between data points.

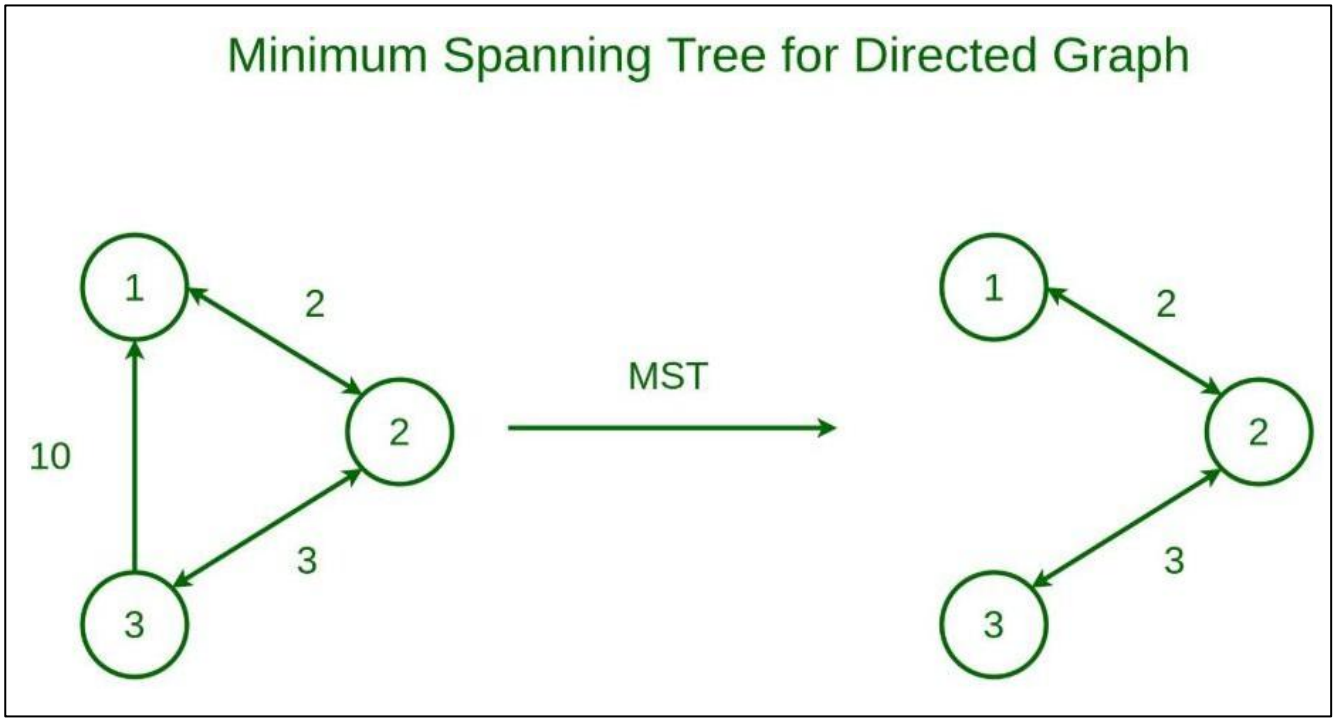
Disadvantages of Spectral Clustering:

1. Complexity: Spectral clustering can be computationally expensive, especially for large datasets, as it requires the calculation of eigenvectors and eigenvalues.
2. Model selection: Choosing the right number of clusters and the right similarity matrix can be challenging and may require expert knowledge or trial and error.

MST Clustering:

A **spanning tree** is defined as a tree-like subgraph of a connected, undirected graph that includes all the vertices of the graph. Or, to say in Layman's words, it is a subset of the edges of the graph that forms a tree (**acyclic**) where every node of the graph is a part of the tree.

The minimum spanning tree has all the properties of a spanning tree with an added constraint of having the minimum possible weights among all possible spanning trees. Like a spanning tree, there can also be many possible MSTs for a graph.



Properties of a Spanning Tree

The spanning tree holds the **below-mentioned principles**:

- The number of vertices (**V**) in the graph and the spanning tree is the same.
- There is a fixed number of edges in the spanning tree which is equal to one less than the total number of vertices (**E** = **V-1**).
- The spanning tree should not be **disconnected**, as in there should only be a single source of component, not more than that.
- The spanning tree should be **acyclic,** which means there would not be any cycle in the tree.
- The total cost (or weight) of the spanning tree is defined as the sum of the edge weights of all the edges of the spanning tree.

- There can be many possible spanning trees for a graph.

Minimum Spanning Tree

*A **minimum spanning tree (MST)** is defined as a **spanning tree** that has the minimum weight among all the possible spanning trees.*

The minimum spanning tree has all the properties of a spanning tree with an added constraint of having the minimum possible weights among all possible spanning trees. Like a spanning tree, there can also be many possible MSTs for a graph.

Algorithms to find Minimum Spanning Tree

➢ Kruskal's Minimum Spanning Tree Algorithm

This is one of the popular algorithms for finding the minimum spanning tree from a connected, undirected graph. This is a greedy algorithm. The algorithm workflow is as below:

- First, it sorts all the edges of the graph by their weights,
- Then starts the iterations of finding the spanning tree.
- At each iteration, the algorithm adds the next lowest-weight edge one by one, such that the edges picked until now does not form a cycle.

➢ Prim's Minimum Spanning Tree Algorithm:

This is also a greedy algorithm. This algorithm has the following workflow:

- It starts by selecting an arbitrary vertex and then adding it to the MST.
- Then, it repeatedly checks for the minimum edge weight that connects one vertex of MST to another vertex that is not yet in the MST.
- This process is continued until all the vertices are included in the MST.

➢ Boruvka's Minimum Spanning Tree Algorithm:

This is also a graph traversal algorithm used to find the minimum spanning tree of a connected, undirected graph. This is one of the oldest algorithms. The algorithm works by iteratively building the minimum spanning tree, starting with each vertex in the graph as its own tree. In each iteration, the algorithm finds the cheapest edge that connects a tree to another tree, and adds that edge to the minimum spanning tree. This is almost similar to the Prim's algorithm for finding the minimum spanning tree. The algorithm has the following workflow:

- Initialize a forest of trees, with each vertex in the graph as its own tree.
- For each tree in the forest:
    - Find the cheapest edge that connects it to another tree. Add these edges to the minimum spanning tree.
    - Update the forest by merging the trees connected by the added edges.
- Repeat the above steps until the forest contains only one tree, which is the minimum spanning tree.

Applications of Minimum Spanning Trees:

- **Network design**: Spanning trees can be used in network design to find the minimum number of connections required to connect all nodes. Minimum spanning trees, in particular, can help minimize the cost of the connections by selecting the cheapest edges.

- **Image processing**: Spanning trees can be used in image processing to identify regions of similar intensity or color, which can be useful for segmentation and classification tasks.
- **Biology**: Spanning trees and minimum spanning trees can be used in biology to construct phylogenetic trees to represent evolutionary relationships among species or genes.
- **Social network analysis**: Spanning trees and minimum spanning trees can be used in social network analysis to identify important connections and relationships among individuals or groups.

BIRCH:

Clustering algorithms like K-means clustering do not perform clustering very efficiently and it is difficult to process large datasets with a limited amount of resources (like memory or a slower CPU). So, regular clustering algorithms do not scale well in terms of running time and quality as the size of the dataset increases. This is where BIRCH clustering comes in. **Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH)** is a clustering algorithm that can cluster large datasets by first generating a small and compact summary of the large dataset that retains as much information as possible. This smaller summary is then clustered instead of clustering the larger dataset. BIRCH is often used to complement other clustering algorithms by creating a summary of the dataset that the other clustering algorithm can now use. However, BIRCH has one major drawback - it can only process metric attributes. A **metric attribute** is any attribute whose values can be represented in Euclidean space i.e., no categorical attributes should be present. Before we implement BIRCH, we must understand two important terms: **Clustering Feature (CF) and CF - Tree Clustering Feature (CF):** BIRCH summarizes large datasets into smaller, dense regions called Clustering Feature (CF) entries. Formally, a Clustering Feature entry is defined as an ordered triple, *(N, LS, SS)* where 'N' is the number of data points in the cluster, 'LS' is the linear sum of the data points and 'SS' is the squared sum of the data points in the cluster. It is possible for a CF entry to be composed of other CF entries. **CF Tree:** The CF tree is the actual compact representation that we have been speaking of so far. A CF tree is a tree where each leaf node contains a sub-cluster. Every entry in a CF tree contains a pointer to a child node and a CF entry made up of the sum of CF entries in the child nodes. There is a maximum number of entries in each leaf node. This maximum number is called the ***threshold***. We will learn more about what this threshold value is. **Parameters of BIRCH Algorithm :**

- ***threshold*** : threshold is the maximum number of data points a sub-cluster in the leaf node of the CF tree can hold.
- ***branching_factor*** : This parameter specifies the maximum number of CF sub-clusters in each node (internal node).

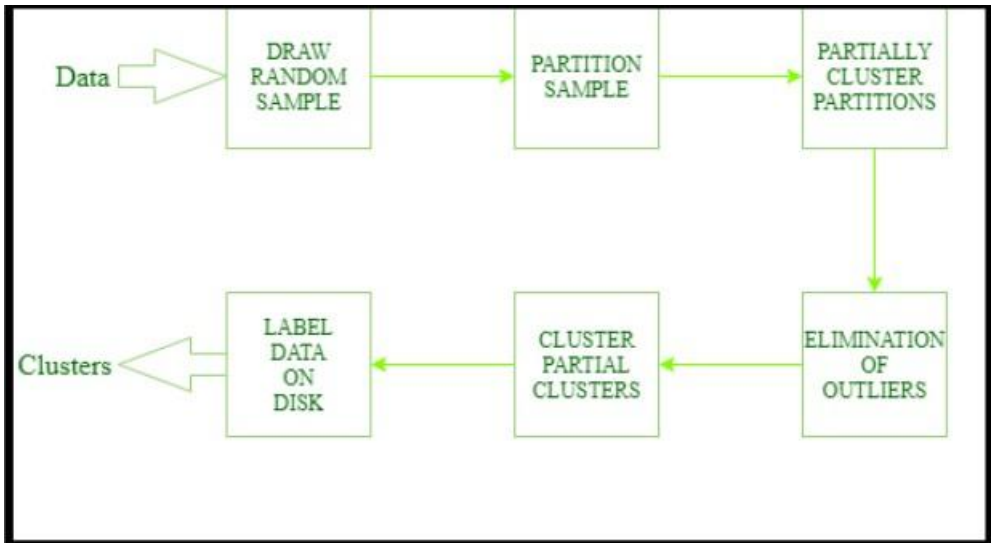
- ***n_clusters*** : The number of clusters to be returned after the entire BIRCH algorithm is complete i.e., number of clusters after the final clustering step. If set to None, the final clustering step is not performed and intermediate clusters are returned.

CURE:

CURE(Clustering Using Representatives)

- It is a hierarchical based clustering technique, that adopts a middle ground between the centroid based and the all-point extremes. Hierarchical clustering is a type of clustering, that starts with a single point cluster, and moves to merge with another cluster, until the desired number of clusters are formed.
- It is used for identifying the spherical and non-spherical clusters.
- It is useful for discovering groups and identifying interesting distributions in the underlying data.
- Instead of using one point centroid, as in most of data mining algorithms, CURE uses a set of well-defined representative points, for efficiently handling the clusters and eliminating the outliers.

Six steps in CURE algorithm:



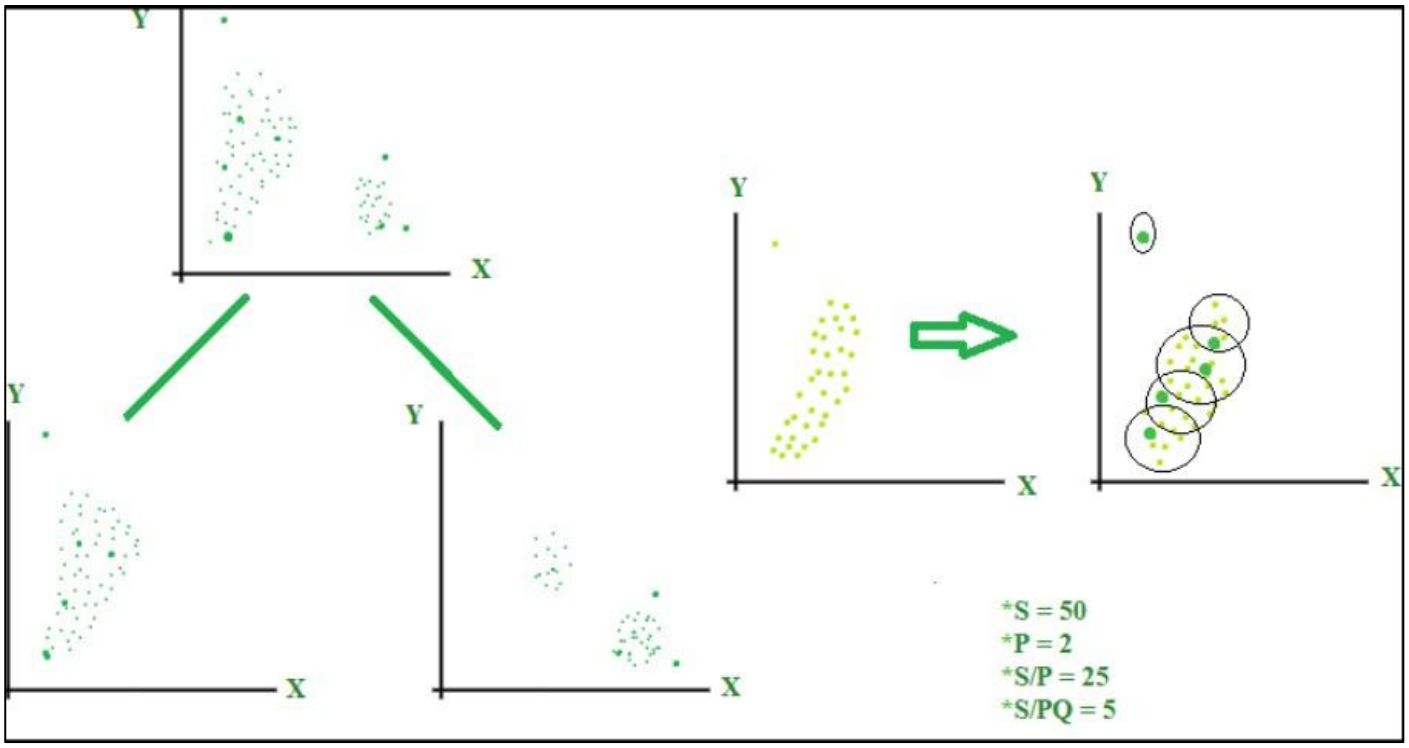
Idea: Random sample, say 's' is drawn out of a given data. This random sample is partitioned, say 'p' partitions with size s/p. The partitioned sample is partially clustered, into say 's/pq' clusters. Outliers are discarded/eliminated from this partially clustered partition. The partially clustered partitions need to be clustered again. Label the data in the disk.

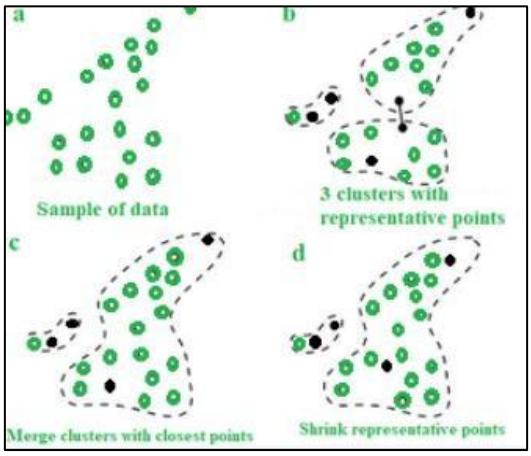
- Procedure :

1. Select target sample number 'gfg'.
2. Choose 'gfg' well scattered points in a cluster.
3. These scattered points are shrunk towards centroid.
4. These points are used as representatives of clusters and used in 'Dmin' cluster merging approach. In Dmin(distance minimum) cluster merging approach, the minimum distance

from the scattered point inside the sample 'gfg' and the points outside 'gfg sample, is calculated. The point having the least distance to the scattered point inside the sample, when compared to other points, is considered and merged into the sample.

5. After every such merging, new sample points will be selected to represent the new cluster.
6. Cluster merging will stop until target, say 'k' is reached.





EvaluationMetrics: Intrinsic (Silhouette Score, Davies–Bouldin Index)

In Supervised Learning, the labels are known and evaluation can be done by calculating the degree of correctness by comparing the predicted values against the labels. However, in Unsupervised Learning,

the labels are not known, which makes it hard to evaluate the degree of correctness as there is **no ground truth**.

That being said, it is still consistent that a *good* clustering algorithm has clusters that have **small within-cluster variance** (data points in a cluster are similar to each other) and **large between-cluster variance** (clusters are dissimilar to other clusters).

There are two types of evaluation metrics for clustering,

- **Extrinsic Measures**: These measures require ground truth labels, which may not be available in practice
- **Intrinsic Measures**: These measures do not require ground truth labels (applicable to all unsupervised learning results)

1. Silhouette Score

The **Silhouette Score** is a way to measure how good the clusters are in a dataset. It helps us understand how well the data points have been grouped. The score ranges from -1 to 1.

- A score close to 1 means a point fits really well in its group (cluster) and is far from other groups.
- A score close to 0 means the point is on the border between two clusters.
- A score close to -1 means the point might be in the wrong cluster.

Silhouette Score (S) for a data point i is calculated as:

$$S(i) = \frac{b(i) - a(i)}{max(a(i), b(i))}$$

where,

- a(i) is the average distance from i to other data points in the same cluster.
- b(i) is the smallest average distance from i to data points in a different cluster.

2. Davies-Bouldin Index

The **Davies-Bouldin Index (DBI)** helps us measure how good the clustering is in a dataset. It looks at how tight each cluster is (compactness), and how far apart the clusters are (separation).

- Lower DBI = better, clearer clusters
- Higher DBI = messy, overlapping clusters A

lower score is better, because it means:

- Points in the same cluster are close to each other.
- Different clusters are far apart from one another.

Davies-Bouldin Index (DB) is calculated as:

$$DB = \frac{1}{k}\sum_{i=1}^{k} \max_{j \neq i} \left(\frac{R_{ii}+R_{jj}}{R_{ij}}\right)$$

where,

- k is the total number of clusters.
- *Rii* is the compactness of cluster i.
- *Rjj* is the compactness of cluster j.
- *Rij* is the dissimilarity (distance) between cluster i and cluster j.

How to calculate Davies-Bouldin Index?

1. Calculate the average distance between points in each cluster and the centroid of the cluster.
2. Calculate the distance between the centroids of each cluster and the nearest cluster.
3. Calculate the ratio of the average distance between points in a cluster and the centroid of the cluster to the distance between the centroids of the cluster and the nearest cluster.
4. Repeat steps 1-3 for each cluster.
5. Calculate the average of the ratios for all clusters.

Lower vs. Higher DB Index Values

The Davies-Bouldin Index (DBI) is a metric for evaluating the validity of a clustering solution. It is a relative clustering validity index, meaning that it compares the clustering results to a hypothetical "ideal" clustering. A lower DBI value indicates a better clustering solution.

- Higher DB index values correspond to poorer clustering solutions. This is because a higher DBI value indicates that the clusters are not well-separated and/or that the clusters are not compact.
- However, a lower DB index value is desirable. It indicates that the clusters are well-separated and compact, which is often a good indication of a successful clustering solution.

Extrinsic (Homogeneity, Completeness, Adjusted Rand Index):

Homogeneity

Homogeneity is an **extrinsic evaluation metric** for clustering. If a cluster contains only data points of **one true class**, then the cluster is **homogeneous**. If a cluster mixes data points from different classes, it loses homogeneity. **Homogeneity** measures whether each cluster contains only members of a single class. A clustering result satisfies homogeneity if all of its clusters contain only data points which are members of a single class.

When we perform clustering, we don't use class labels. But to evaluate how good the clustering is, we compare it with the **actual labels (ground truth)**. Homogeneity helps check whether the clustering kept data points of the same class **together**.

$$Homogeneity = 1 - \frac{H(C|K)}{H(C)}$$

Where:

- CCC = set of **true classes**
- KKK = set of **clusters**
- H(C)H(C)H(C) = entropy of the class distribution (measures uncertainty of classes)
- H(C|K)H(C|K)H(C|K) = conditional entropy (uncertainty of classes given clusters)

Completeness:

Completeness is an **extrinsic clustering evaluation metric**. If all points of a class are grouped into **one single cluster**, then the clustering has **high completeness**. If the points of a class are scattered across different clusters, completeness decreases.

Completeness measures whether all members of a given class are assigned to the same cluster. A clustering result satisfies completeness if all the data points that are members of a given class are elements of the same cluster.

Completeness is defined using **conditional entropy** of clusters given the true classes:

$$Completeness = 1 - \frac{H(K|C)}{H(K)}$$

Where:

- CCC = set of **true classes**
- KKK = set of **clusters**
- H(K)H(K)H(K) = entropy of the cluster distribution (uncertainty of clusters)
- H(K|C)H(K|C)H(K|C) = conditional entropy of clusters given classes

If clustering perfectly preserves classes in one cluster, H(K|C)=0H(K|C) = 0H(K|C)=0, so **Completeness = 1**. If clustering spreads class members across clusters, Completeness goes towards 0.

AdjustedRandIndex:

Measures the similarity between two data clusters using the Adjusted Rand Index (ARI) metric in clustering machine learning models.

Purpose

The Adjusted Rand Index (ARI) metric is intended to measure the similarity between two data clusters. This metric is specifically used for clustering machine learning models to quantify how well the model is clustering and producing data groups. It involves comparing the model's produced clusters against the actual (true) clusters found in the dataset.

Test Mechanism

The Adjusted Rand Index (ARI) is calculated using the adjusted_rand_score method from the sklearn.metrics module in Python. The test requires inputs including the model itself and the model's training and test datasets. The model's computed clusters and the true clusters are compared, and the similarities are measured to compute the ARI.

Signs of High Risk

- If the ARI is close to zero, it signifies that the model's cluster assignments are random and do not match the actual dataset clusters, indicating a high risk.
- An ARI of less than zero indicates that the model's clustering performance is worse than random.

Strengths

- ARI is normalized and provides a consistent metric between -1 and +1, irrespective of raw cluster sizes or dataset size variations.
- It does not require a ground truth for computation, making it ideal for unsupervised learning model evaluations.
- It penalizes for false positives and false negatives, providing a robust measure of clustering quality.

Limitations

- In real-world situations, true clustering is often unknown, which can hinder the practical application of the ARI.
- The ARI requires all individual data instances to be independent, which may not always hold true.
- It may be difficult to interpret the implications of an ARI score without context or a benchmark, as it is heavily dependent on the characteristics of the dataset used.

Elbow method:

In K-means clustering, the algorithm partitions data into k clusters by minimizing the distances between points and their cluster centroids. However, deciding the ideal k is not straightforward. The Elbow Method helps by plotting the Within-Cluster Sum of Squares (WCSS) against increasing k values and looking for a point where the improvement slows down, this point is called the "elbow."

Working of Elbow Point

The Elbow Method works in the following steps:

1. We iterate over a range of k values, typically from 1 to n (where n is a hyperparameter you choose).

2. For each k, we calculate a distance measure called WCSS (Within-Cluster Sum of Squares). This tells us how spread out the data points are within each cluster.

WCSS measures how well the data points are clustered around their respective centroids. It is defined as the sum of the squared distances between each point and its cluster centroid:

$$\text{WCSS} = \sum_{i=1}^{k} \sum_{j=1}^{n_i} \text{distance}\left(x_j^{(i)}, c_i\right)^2$$

where:

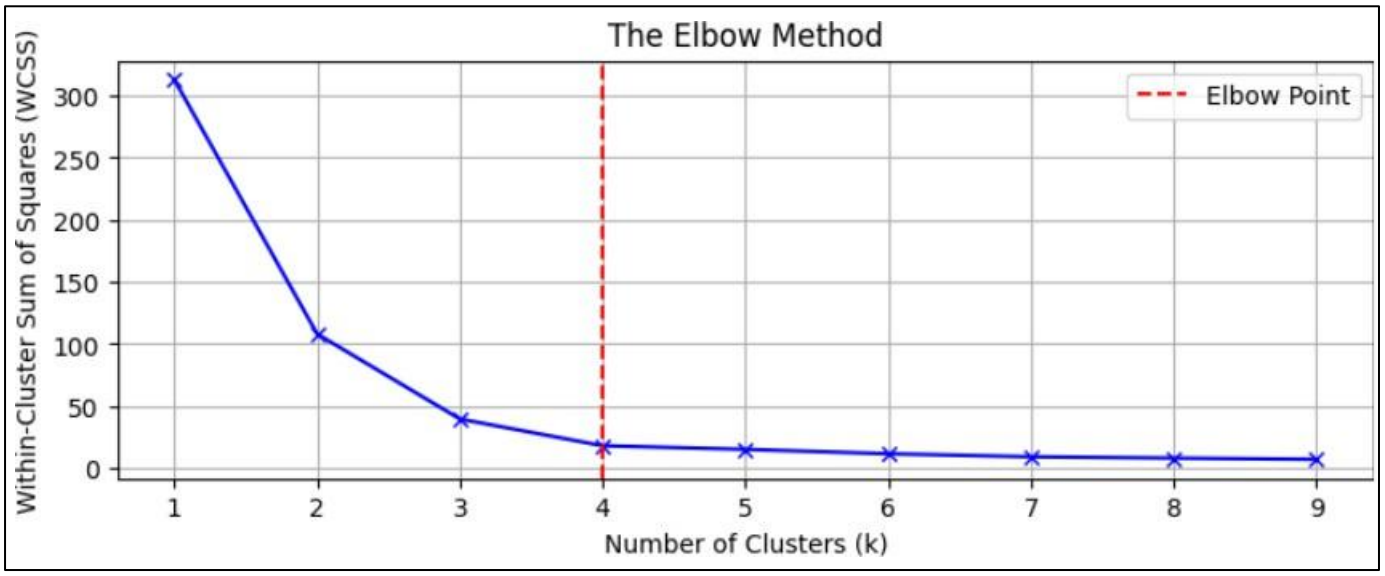
- distance(xj(i),ci)distance(*xj*(*i*),*ci*)represents the distance between the jth*jth* data point xj(i)*xj*(*i*) in cluster i and the centroid ci*ci*of that cluster.

3. We try different k values (number of clusters). For each k, we run KMeans and calculate the WCSS.

4. We plot a graph with k on the X-axis and WCSS on the Y-axis.

5. As we increase k, the WCSS typically decreases because we're creating more clusters, which tend to capture more data variations. However, there comes a point where adding more clusters results in only a marginal decrease in WCSS. This is where we observe an "elbow" shape in the graph.

- **Before the elbow**: Increasing k significantly reduces WCSS, indicating that new clusters effectively capture more of the data's variability.
- **After the elbow**: Adding more clusters results in a minimal reduction in WCSS, suggesting that these extra clusters may not be necessary and could lead to overfitting.



Elbow Point

The goal is to identify the point where the rate of decrease in WCSS sharply changes, indicating that adding more clusters (beyond this point) yields diminishing returns. This "elbow" point suggests the optimal number of clusters.

Elbow Method Drawbacks

The elbow method, while a useful tool for determining the optimal number of clusters in K-means clustering, has some drawbacks:

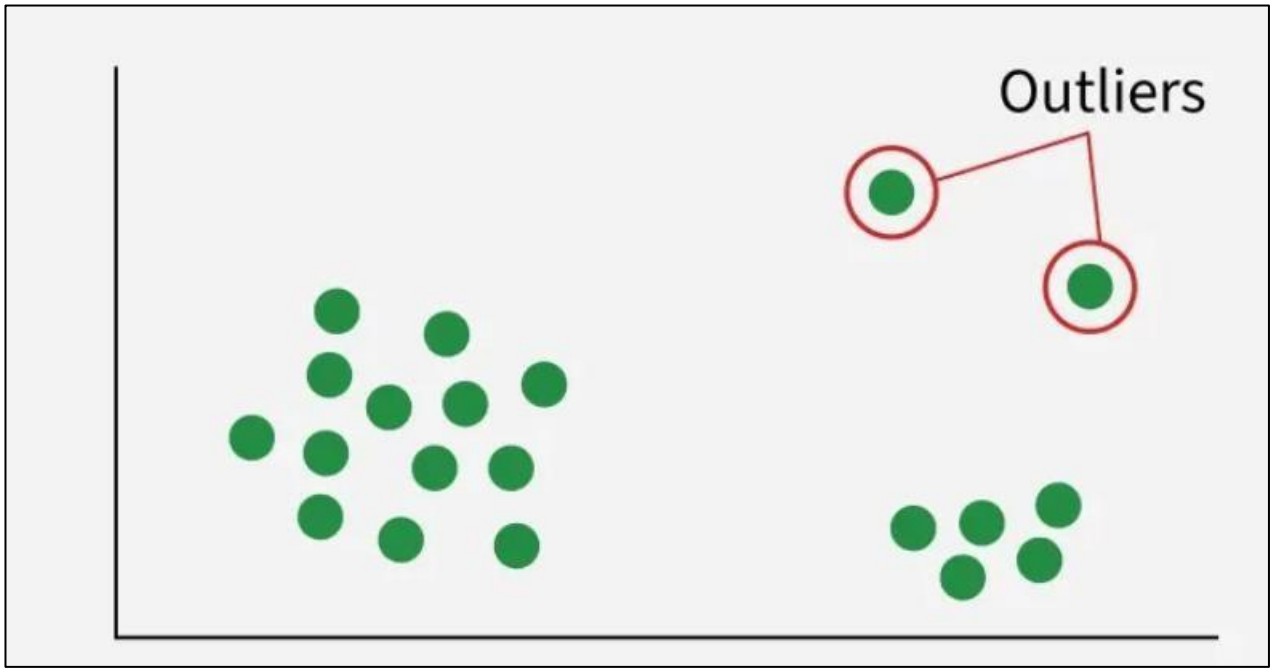
- **Subjectivity**: The choice of the “elbow point” can be subjective and might vary between individuals analyzing the same data.
- **Non-Gaussian Data**: It assumes that clusters are spherical and equally sized, which may not hold for complex datasets with irregularly shaped or differently sized clusters.
- **Sensitivity to Initialization**: K-means itself is sensitive to initial cluster centroids, which can affect the WCSS values and, consequently, the choice of the optimal K.

- **Inefficient for Large Datasets**: For large datasets, calculating WCSS for a range of K values can be computationally expensive and time-consuming.
- **Unsuitable for All Distributions**: The elbow method is not suitable for all data distributions, especially when clusters have varying densities or are non-convex.
- **Limited to K-means**: It specifically applies to K-means clustering and may not be suitable for other clustering algorithms with different objectives.

Despite these drawbacks, the elbow method remains a valuable starting point for selecting the number of clusters, and it often provides useful insights into the data's underlying structure. However, it's essential to complement it with other validation techniques when working with more complex datasets or different clustering algorithms.

Outlier Detection:

In machine learning, outliers are data points that deviate significantly from the general distribution of the dataset. They may occur due to errors in data collection, natural variation or rare events. While sometimes they contain useful insights like in fraud detection but in many cases they negatively affect model accuracy and skew results making outlier detection a crucial preprocessing step.



- **Impact on ML models**: Can bias parameter estimation, increase variance and reduce model accuracy.
- **Sources**: Data entry errors, measurement noise, genuine rare events.
- **Handling methods:** Removal, transformation or robust algorithms less sensitive to outliers.
- **Domain dependence**: What is considered an outlier in one domain (e.g., finance) may be normal in another.

Types of Outliers

Outliers can be categorized as:

1. Global Outliers (Point Anomalies):

- Individual data points that lie far from the rest.
- **Example**: A wine record with alcohol level 20% when most are <15%.

2. Contextual Outliers:

- Outliers relative to a specific context or condition.
- **Example**: 30°C might be normal in summer but an outlier in winter.

3. Collective Outliers:

- A group of related data points behaving anomalously together.
- **Example**: A sudden spike in residual sugar and acidity in wine samples together.

Outliers Detection Methods:

1. Z-Score Method

he Z-Score method is a statistical technique that detects outliers based on how far a data point is from the mean, measured in terms of standard deviations. It assumes the data follows a normal distribution. A point with a very high or low Z-score (typically $|Z| > 3$) is flagged as an outlier because it lies in the extreme tails of the distribution.

$$Z = \frac{x-\mu}{\sigma}$$

Where,

- Z: The Z-score (standard score): It tells us how many standard deviations a data point is away from the mean.
- x: The actual data value (the observation we are testing).
- μ: The mean of the dataset (average of all data points).
- σ: The standard deviation of the dataset (a measure of spread/variability).

How it works: Compares distance of a point from the mean in units of standard deviation.

- **Intuition**: Outliers are "too far" from the center of the bell curve.
- **Pros**: Simple, fast, works well with normally distributed data.
- **Cons**: Not reliable for skewed or non-normal distributions.

2. IQR Method (Interquartile Range)

The IQR method is a robust statistical approach that identifies outliers by examining the spread of the middle 50% of the data. It calculates the Interquartile Range (IQR), which is the difference between the

75th percentile (Q3) and 25th percentile (Q1). Any value that falls below Q1 − 1.5 × IQR or above Q3 + 1.5 × IQR is considered an outlier.

$$IQR = Q3 - Q1$$

Outliers Thresholds:

- Lower bound=Q1−1.5×IQRLower bound=$Q1-1.5\times IQR$
- Upper bound=Q3+1.5×IQRUpper bound=$Q3+1.5\times IQR$

Intuition: Values too far below or above the "box" in a boxplot are flagged.

- **Pros**: Robust to non-normal data, less influenced by extreme values.
- **Cons**: Doesn't adapt well to very skewed distributions.

3. Isolation Forest

Isolation Forest is a model-based anomaly detection algorithm that isolates outliers instead of profiling normal data. It builds multiple random decision trees by repeatedly splitting the data. Since outliers are few and different, they are easier to isolate and require fewer splits.

How it works:

- Randomly select features and split values.
- Construct isolation trees.
- Compute average path length for each point.
- Shorter path = more likely outlier.

Pros: Works well in high dimensions, efficient.

Cons: Requires choosing contamination (expected outlier fraction).

4. Local Outlier Factor (LOF)

The Local Outlier Factor (LOF) method is a density-based anomaly detection technique that compares the local density of a data point to that of its neighbors. If a point has significantly lower density than its neighbors, it is flagged as an outlier.

How it works:

- For each point, find k-nearest neighbors.
- Estimate local density based on neighbor distances.
- Compare the density of the point with its neighbors.
- If the point's density ≪ neighbors → outlier.

Pros: Works well with clusters of varying density.

Cons: Sensitive to choice of k (neighbors).

Applications:

Market Segmentation:

Businesses use clustering to group customers based on demographics, purchase history, and behavior. This enables targeted marketing campaigns and personalized product recommendations, improving customer engagement and sales.

Anomaly Detection:

Clustering helps identify unusual patterns or outliers in datasets, which can indicate fraudulent activities in financial transactions, network intrusions in cybersecurity, or defects in manufacturing processes.

Image Segmentation:

In computer vision, clustering is used to divide images into meaningful regions or objects, which is crucial for tasks like object recognition, medical image analysis, and autonomous driving.

Document Clustering and Text Mining:

Clustering documents based on their content allows for efficient organization, summarization, and retrieval of information. It also aids in categorizing news articles, analyzing sentiment, and identifying key themes in large text corpuses.

Recommendation Systems:

By clustering users with similar preferences or items with similar characteristics, recommendation systems can suggest relevant movies, products, or content to users, enhancing user experience.

Genomic and Proteomic Analysis:

In bioinformatics, clustering helps group genes or proteins with similar expression patterns or functions, facilitating the discovery of biological pathways and disease mechanisms.

Geospatial Analysis:

Clustering can be applied to geographical data to identify regions with similar characteristics, such as areas with high crime rates or specific environmental conditions, aiding urban planning and resource allocation.

Data Preprocessing and Dimensionality Reduction:

Clustering can be used as a preliminary step in data analysis to reduce the complexity of large datasets by grouping similar data points, making subsequent analysis more manageable.