

MACHINE LEARNING

UNSUPERVISED

CHAP 9

Unsupervised ML

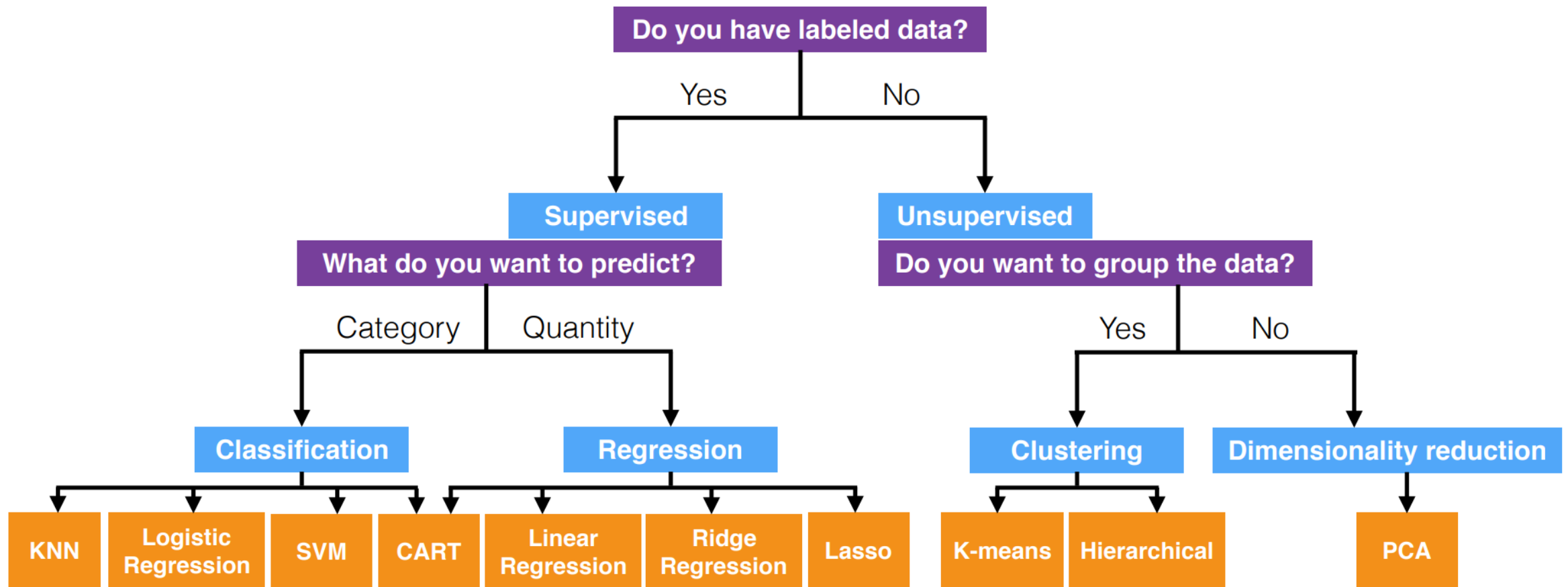
Jean Tabet

j.tabet@ul.edu.lb

→ OUTLINE

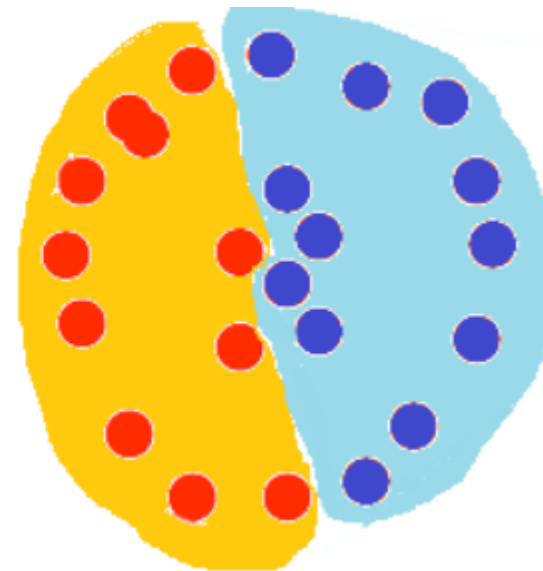
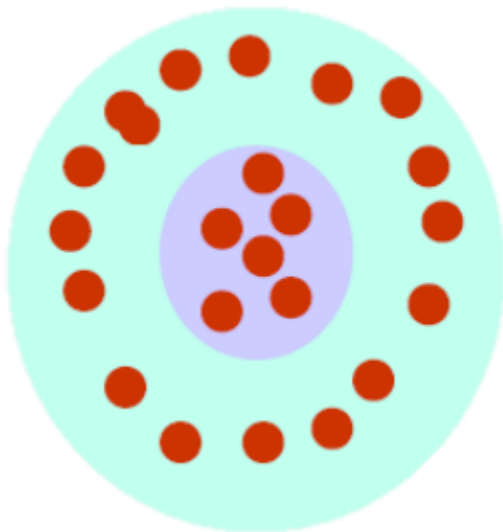
- Introduction to clustering
- Clustering methods
 - K-means clustering
 - Hierarchical clustering
 - DBSCAN

➔ INTRODUCTION TO CLUSTERING



➔ INTRODUCTION TO CLUSTERING

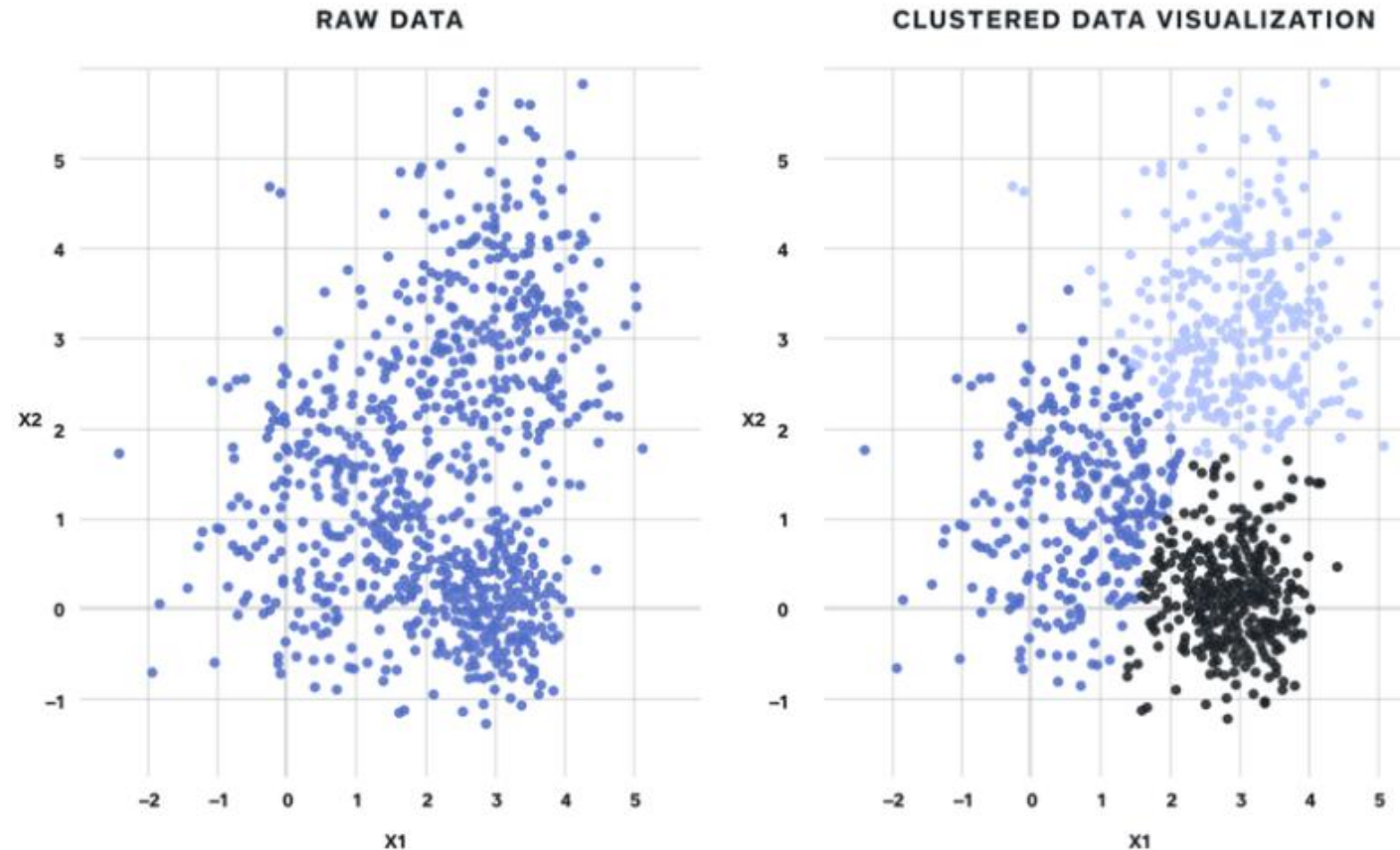
- Unsupervised ML is a set of statistical tools for data that only has features/input X available, but no response y .
- The organization of unlabeled data into similarity groups called clusters.
- A cluster is a collection of data items which are “similar” between them, and “dissimilar” to data items in other clusters.
- Goal: Discover interesting patterns/properties of the data



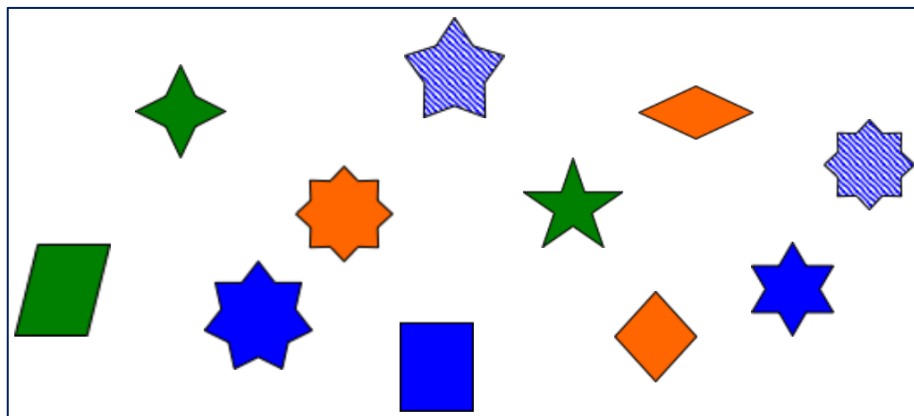
→ CLUSTERING

A set of methods for finding subgroups within the dataset:

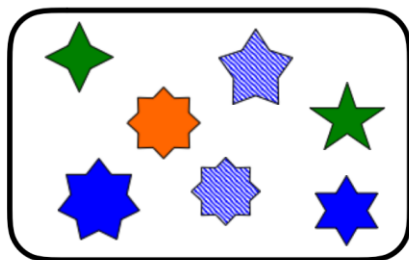
- Observations should share common characteristics within the same group, but differ across groups.
- Groupings are determined from attributes of the data itself — differs from classification.



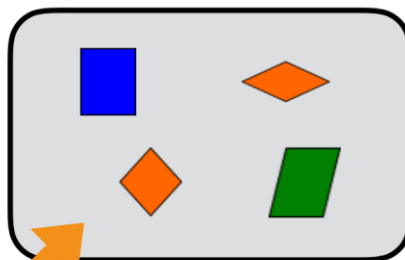
CLUSTERING VS. CLASSIFICATION



Classification



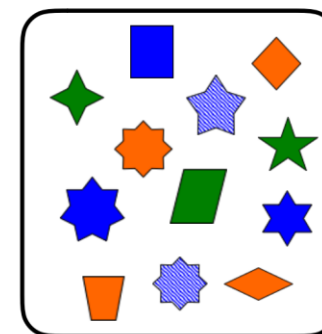
Class A



Class B



Clustering



Dataset

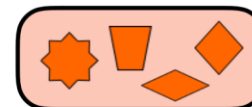
Cluster A



Cluster B



Cluster C



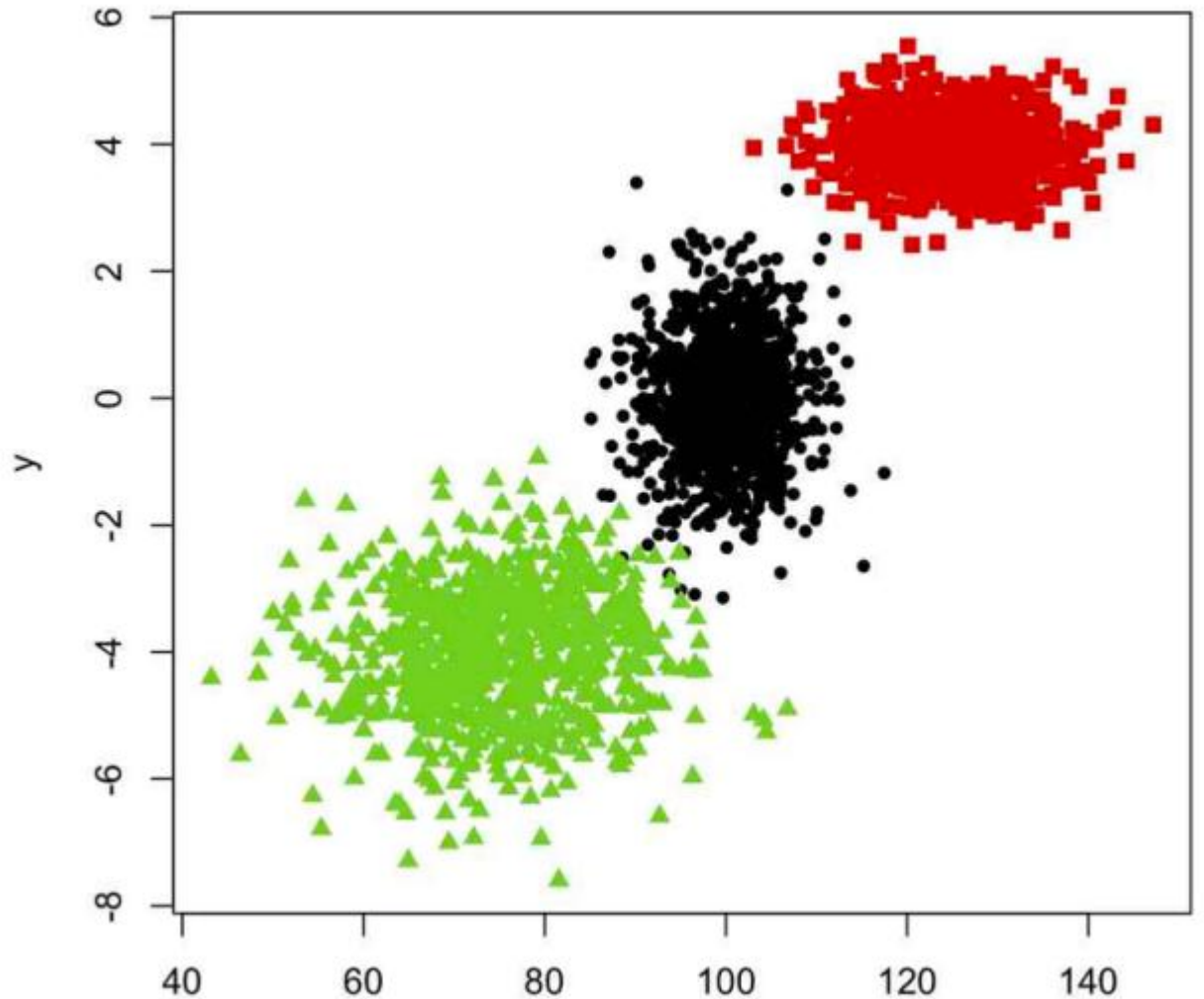
Cluster D



→ K-MEANS CLUSTERING

- A good clustering is one for which the within-cluster variation is as small as possible.
- Denote each cluster by C_k , and let $W(C_k)$ be a measure of the within cluster variation.
- K-means aims to solve:

$$\text{minimize}_{C_1, \dots, C_K} \left\{ \sum_{k=1}^K W(C_k) \right\}$$



→ K-MEANS CLUSTERING

- The most common choice to measure within-cluster variation is squared Euclidean distance:

$$W(C_k) = \frac{1}{|C_k|} \sum_{i, i' \in C_k} \sum_{j=1}^p (x_{ij} - x_{i'j})^2$$

- Which means overall we solve:

$$\text{minimize}_{C_1, \dots, C_K} \left\{ \sum_{k=1}^K \frac{1}{|C_k|} \sum_{i, i' \in C_k} \sum_{j=1}^p (x_{ij} - x_{i'j})^2 \right\}$$

- In practice, use an iterative algorithm that finds a local minimum to this optimization.

→ K-MEANS CLUSTERING

- K-means (MacQueen, 1967) is a **partitional clustering** algorithm
- Let the set of data points D be $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$,
where $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{ir})$ is a **vector** in $X \subseteq R^r$, and r is the number of dimensions.
- The k -means algorithm partitions the given data into k clusters:
 - Each cluster has a cluster **center**, called **centroid**.
 - k is specified by the user

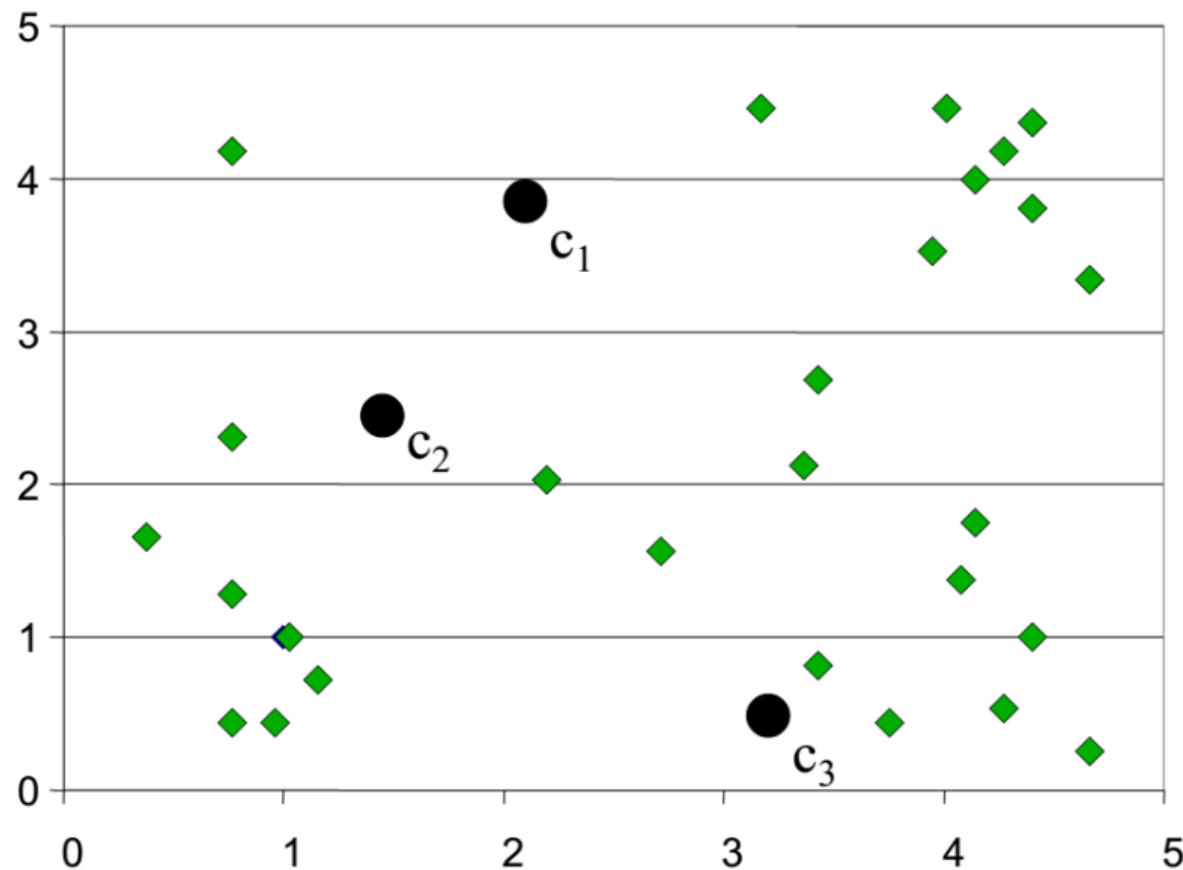
➔ K-MEANS ALGORITHM

- Given k , the *k-means* algorithm works as follows:
 1. Choose k (random) data points (**seeds**) to be the initial **centroids**, cluster centers
 2. Assign each data point to the closest **centroid**
 3. Re-compute the **centroids** using the current cluster memberships
 4. If a convergence criterion is not met, repeat steps 2 and 3

➔ K-MEANS CLUSTERING EXAMPLE:

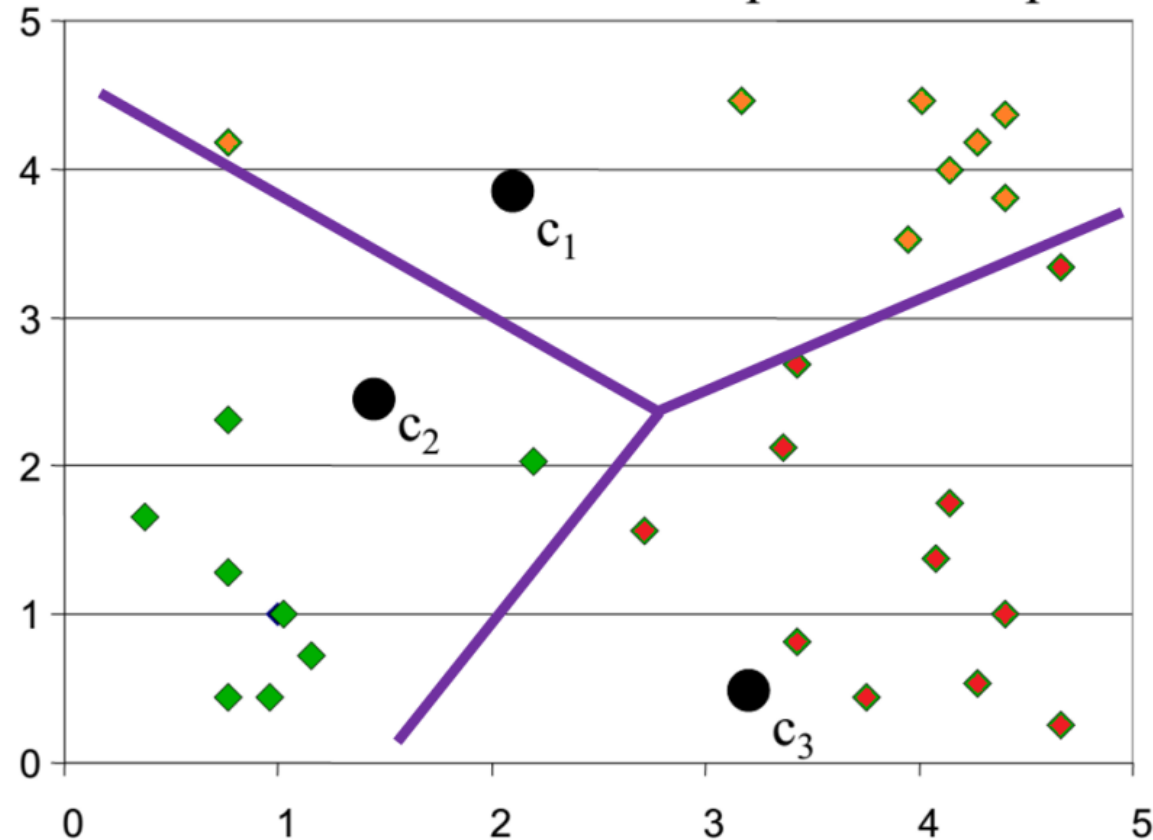
step 1

Randomly initialize the cluster centers



step 2

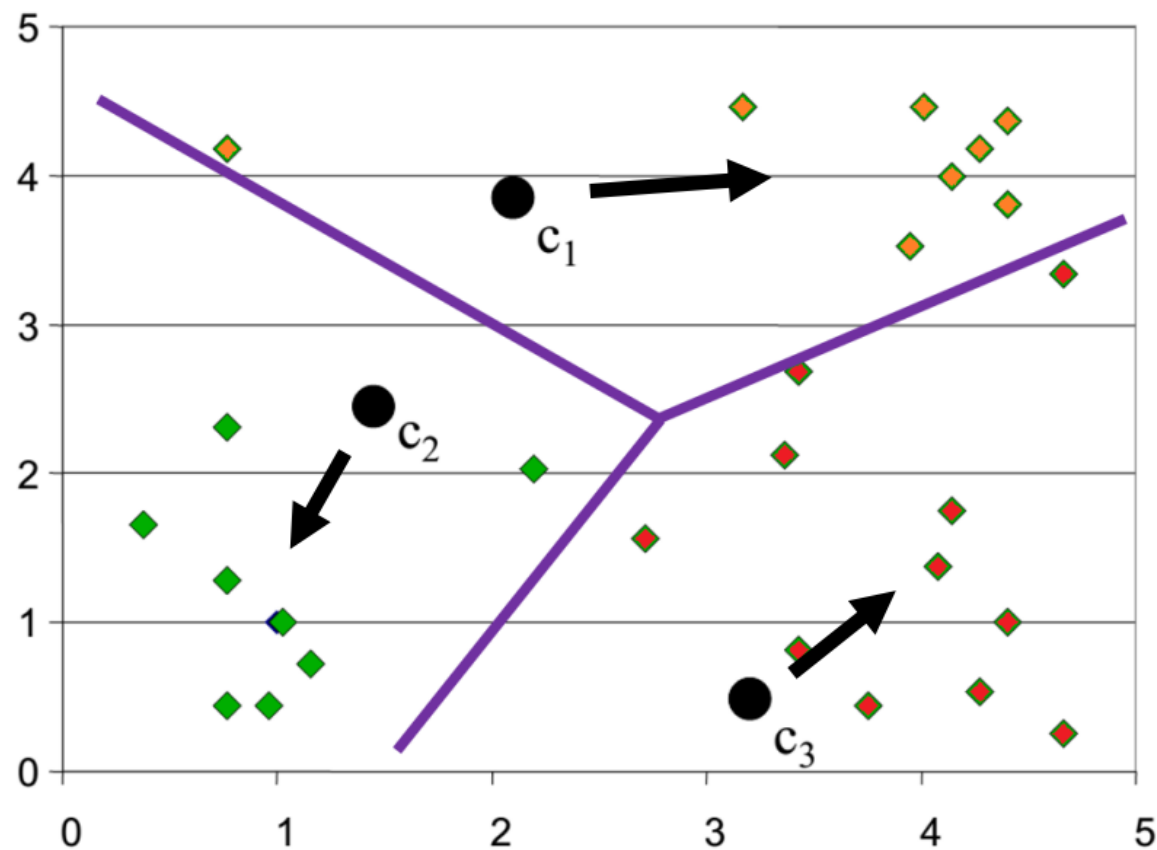
Determine cluster membership for each input



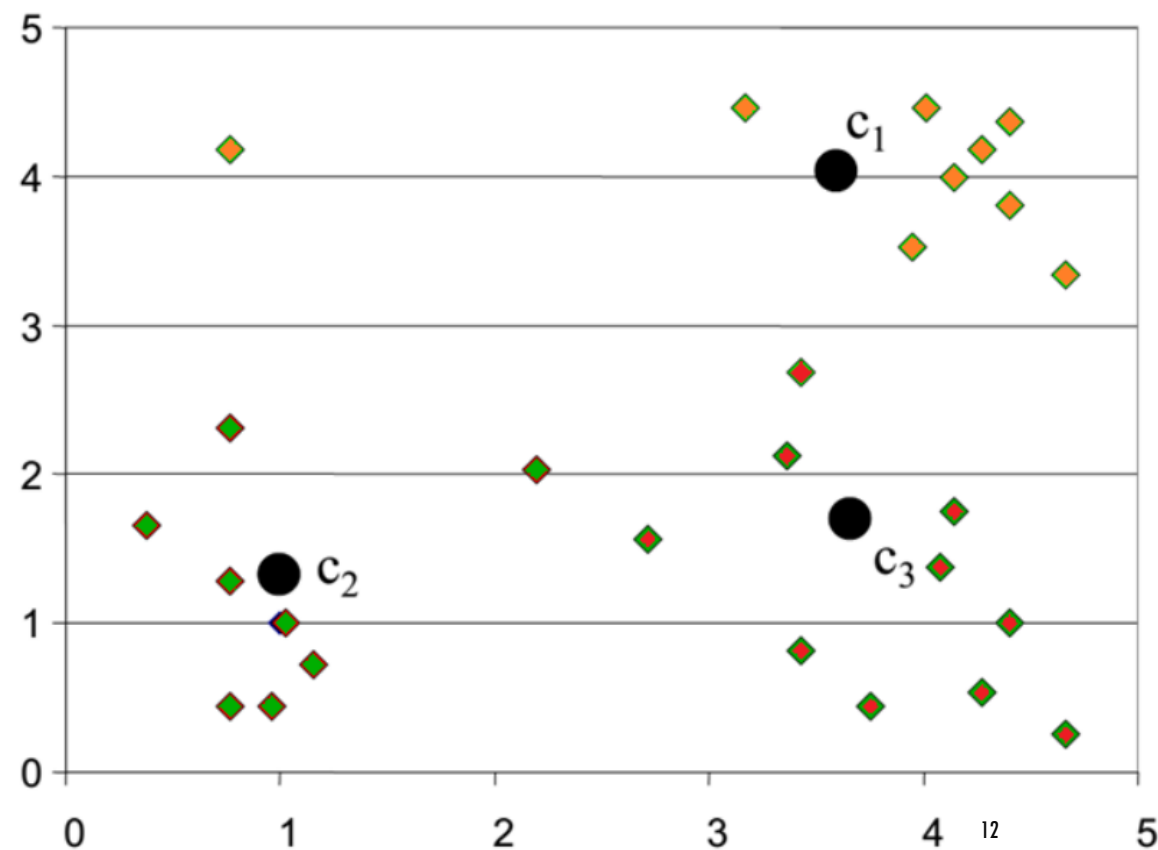
→ K-MEANS CLUSTERING EXAMPLE:

step 3

Re-estimate cluster centers (adapt synaptic weights)

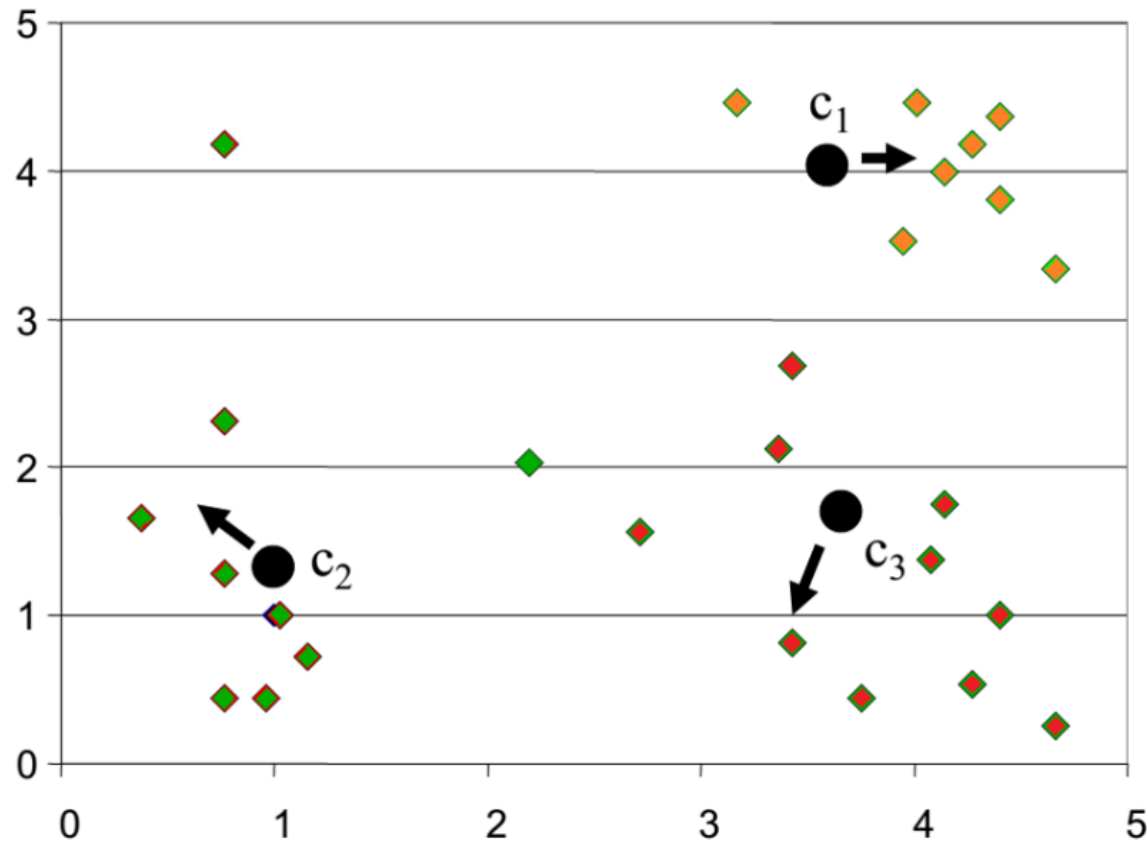


Result of first iteration

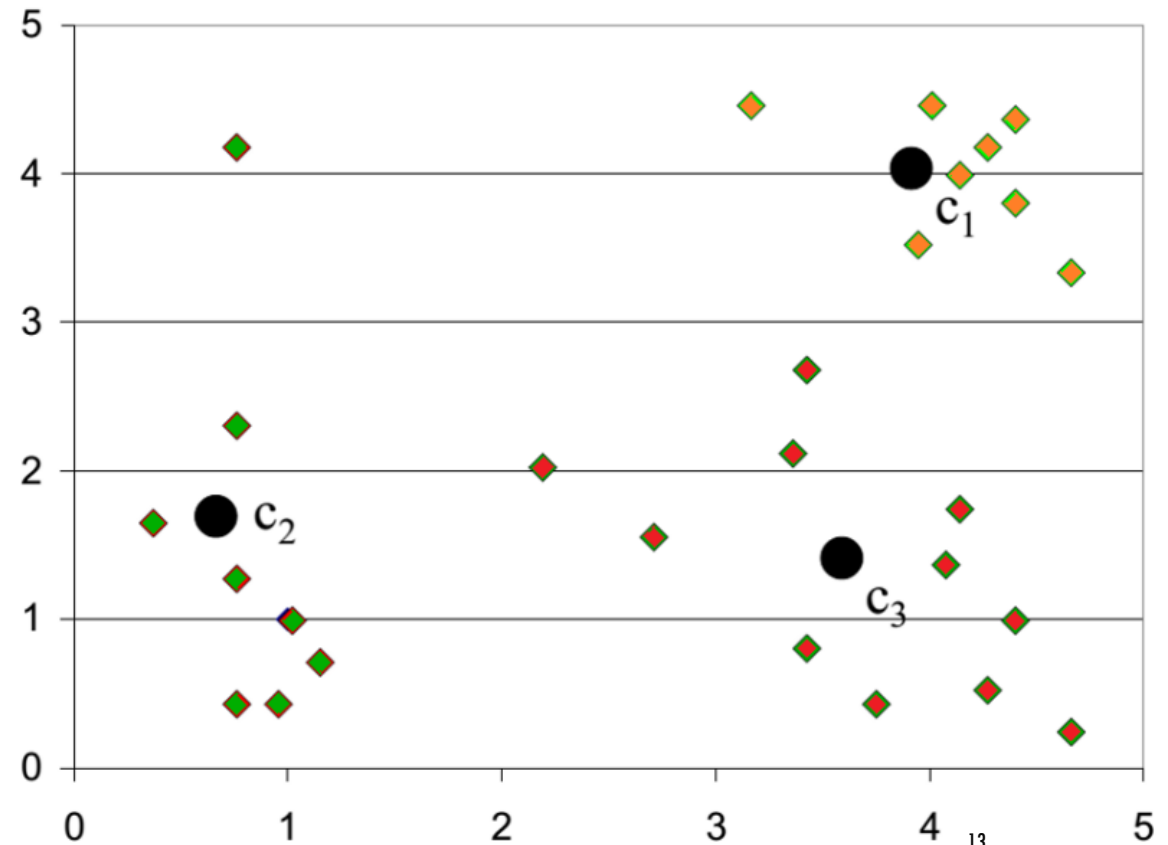


→ K-MEANS CLUSTERING EXAMPLE:

Second iteration



Result of second iteration



➔ STRENGTHS OF K-MEANS

- Strengths:
 - Simple: easy to understand and to implement
 - Efficient: Time complexity: $O(tkn)$,
where n is the number of data points,
 k is the number of clusters, and
 t is the number of iterations.
 - Since both k and t are small. k -means is considered a linear algorithm.
- K-means is the most popular clustering algorithm.
- Note that: it terminates at a **local optimum** if SSE is used.
The **global optimum** is hard to find due to complexity.

➔ WEAKNESSES OF K-MEANS

- The algorithm is only applicable if the mean is defined.
 - For categorical data, k-mode
 - the centroid is represented by the most frequent values.
- The user needs to specify k.
- The algorithm is sensitive to outliers.
 - Outliers are data points that are very far away from other data points.
 - Outliers could be errors in the data recording or some special data points with very different values.

➔ K-MEANS IMPLEMENTATION USING SKLEARN

```
from sklearn.cluster import KMeans

# Create the K-means model
# n_clusters = 5 because we generated 5 groups;
# n_init = 1 means: try only one initialization

kmeans = KMeans(n_clusters=5, init='k-means++', n_init=1, random_state=0)

# Train / fit the model on the data

kmeans.fit(data)

# Display the coordinates of the cluster centers
print("Cluster centers:")
print(kmeans.cluster_centers_)
```

→ HIERARCHICAL CLUSTERING

- Cluster based on distances between observations.
- Represented as a tree hierarchy (dendrogram) rather than a partition of data.
- Does not require committing to a choice of K .

➔ BUILDING A DENDROGRAM

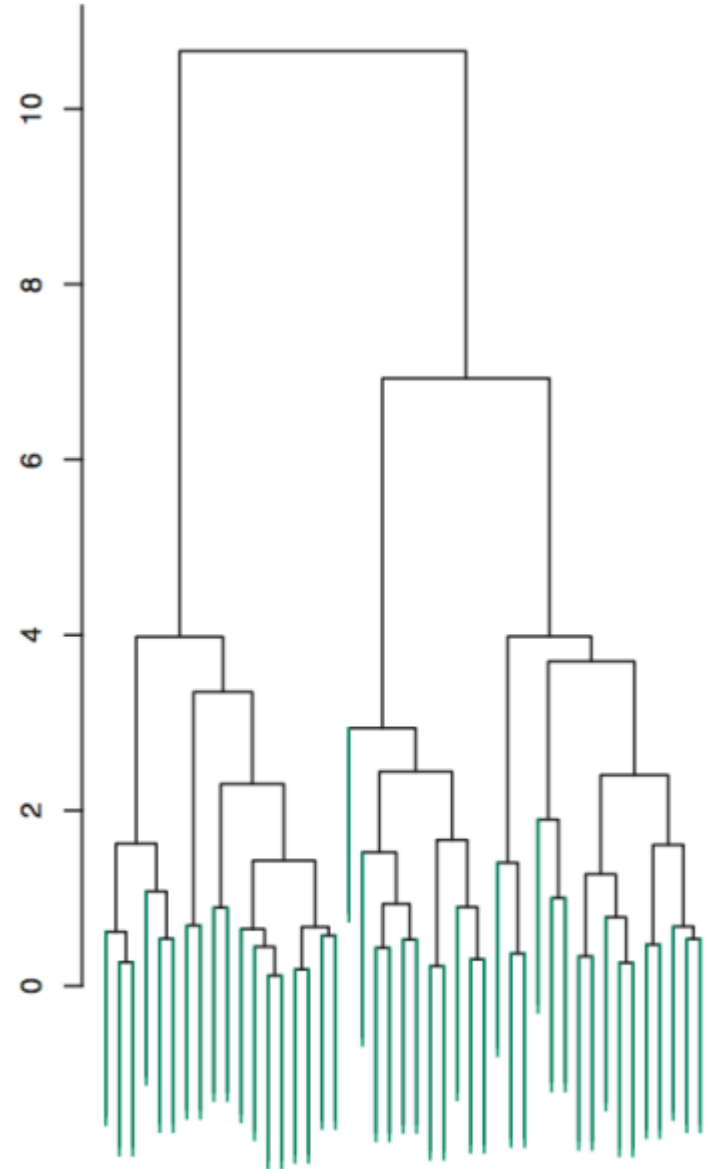
- A dendrogram is most commonly built using a bottom-up or agglomerative algorithm.
- We start at the leaves and group observations until we reach the root containing the entire dataset.
- Like in k-means, we need a measure of similarity. Again, the most common is Euclidean distance.

→ HIERARCHICAL CLUSTERING ALGORITHM

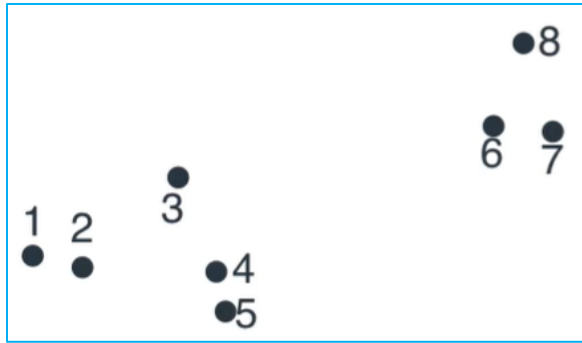
1. Initialize each observation to its own cluster.
2. For $i = n, n-1, \dots, 2$:
 - a. Examine all pairwise inter-cluster similarities among the i clusters and identify the pair of clusters that are most similar. Fuse these two clusters. The dissimilarity between these two clusters indicates the height in the dendrogram at which the fusion occurs.
 - b. Compute the new pairwise inter-cluster similarities among the $i-1$ remaining clusters.

→ DENDROGRAMS

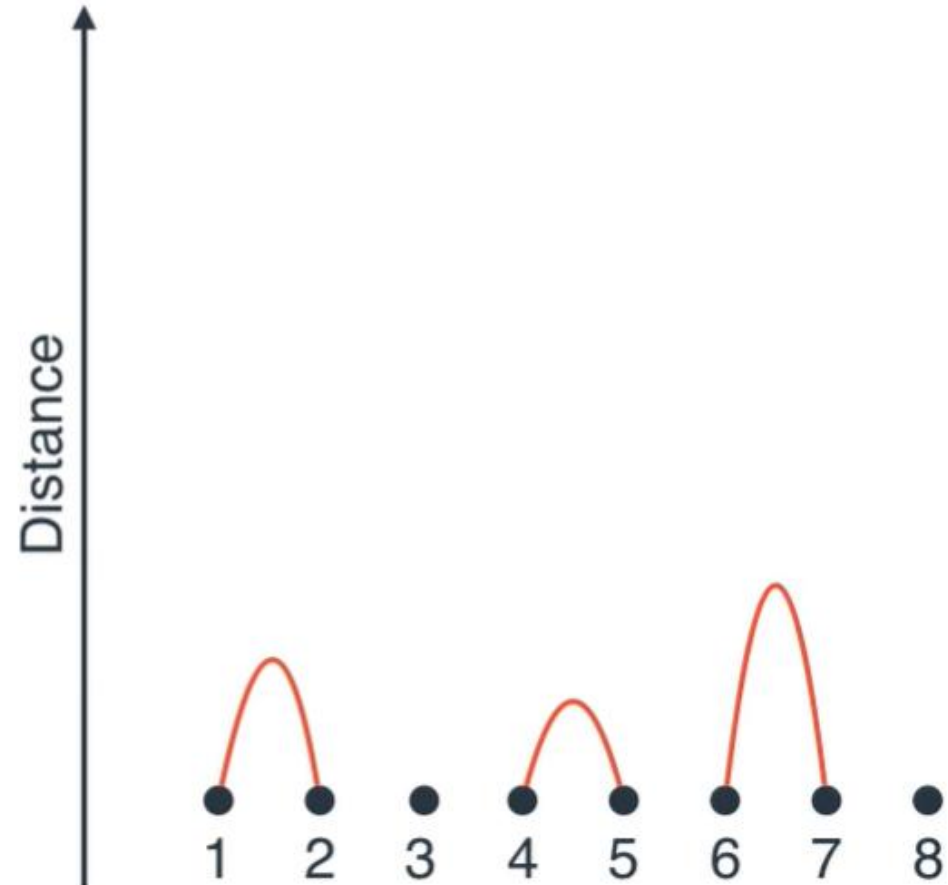
- Each leaf in a dendrogram is a sample/observation.
- As we move up the dendrogram, observations that are similar to each other begin to fuse into branches.
- Branches then fuse into bigger branches.
- Observations that fuse later (near the top of the tree, or root) are more different than observations that fuse earlier.



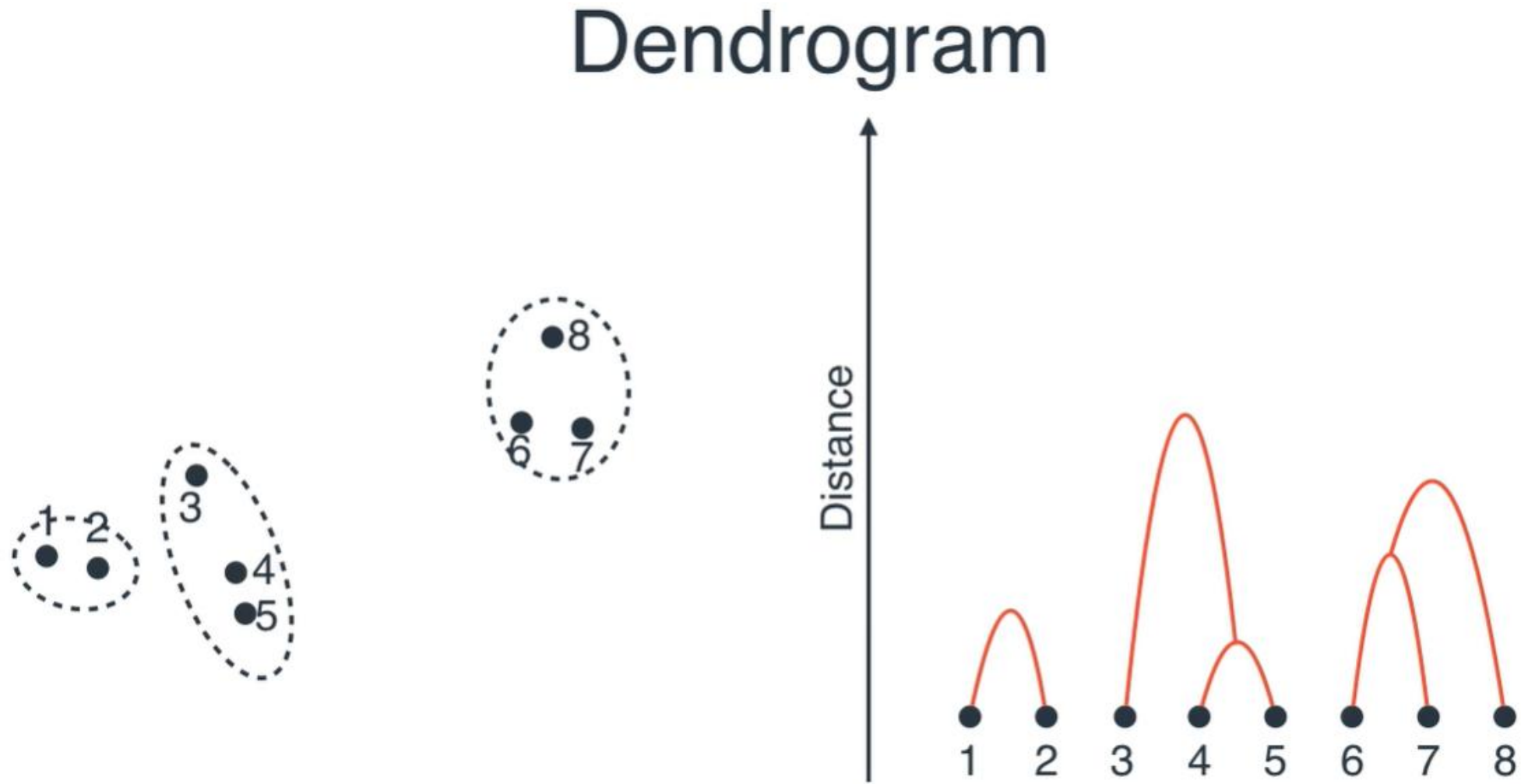
→ DENDROGRAMS



Dendrogram

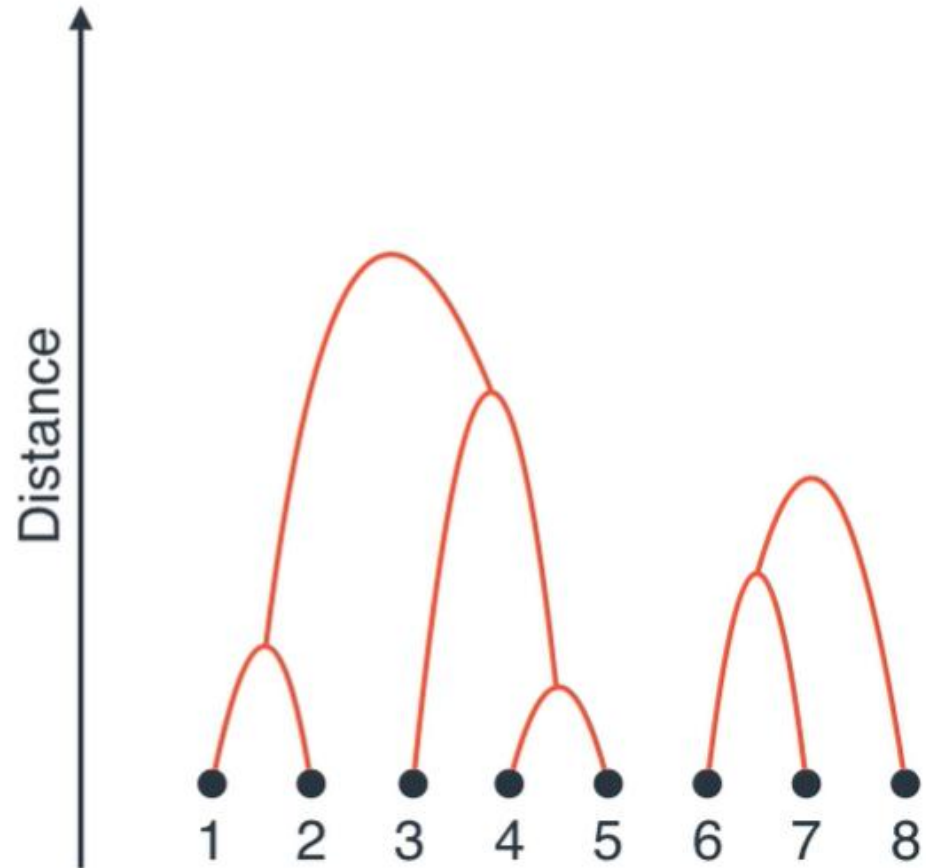
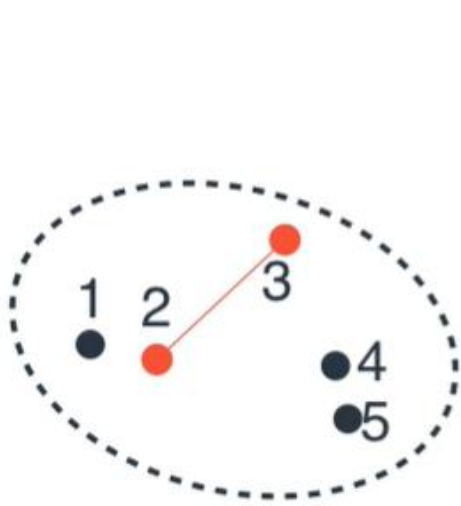


→ DENDROGRAMS



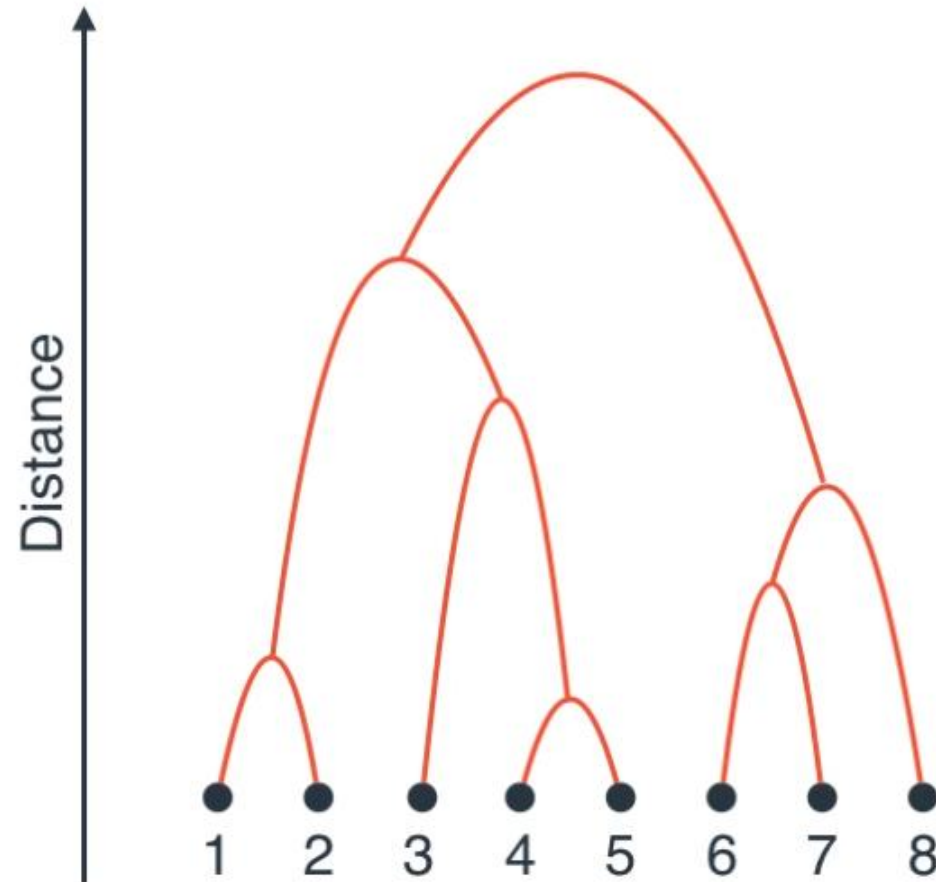
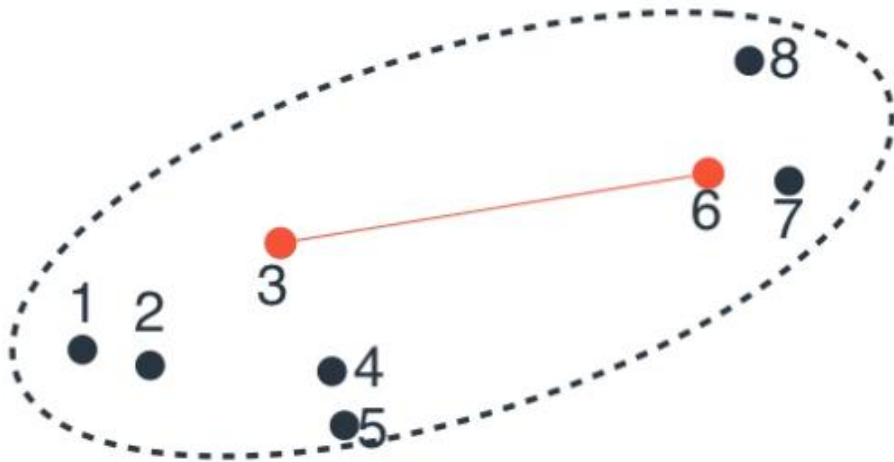
→ DENDROGRAMS

Dendrogram



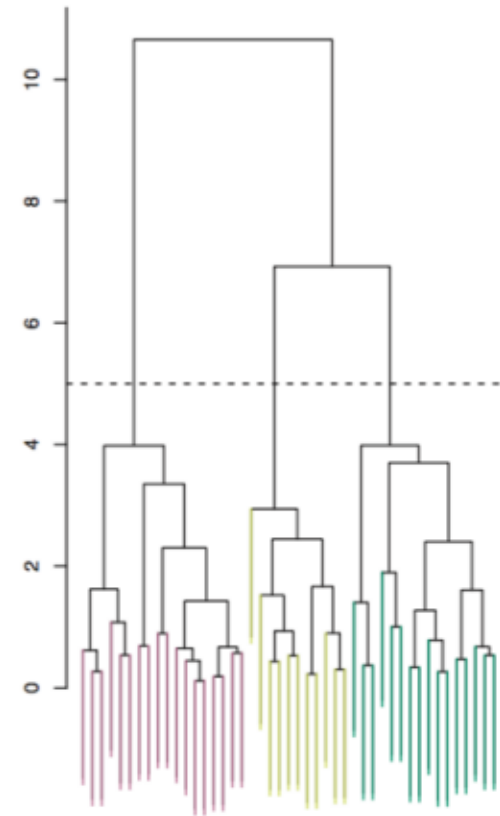
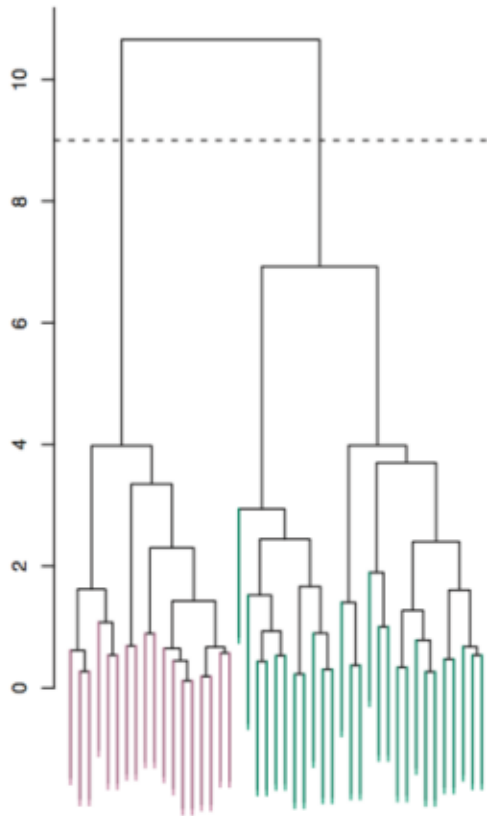
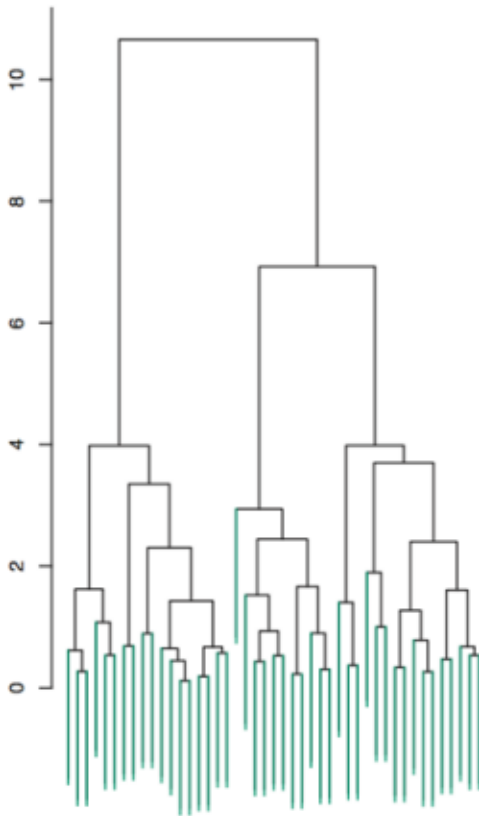
→ DENDROGRAMS

Dendrogram



➔ OBTAINING CLUSTERS

- Clusters are created by making a horizontal cut across the dendrogram. Clusters are the separate trees below the cut.

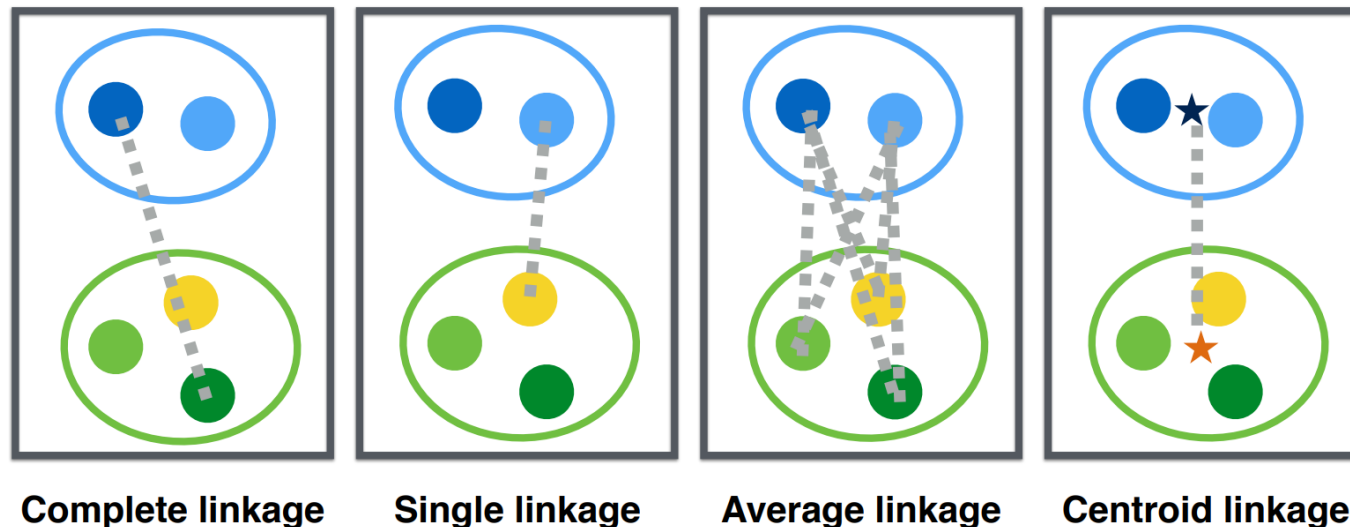


➔ DISTANCE BETWEEN GROUPS

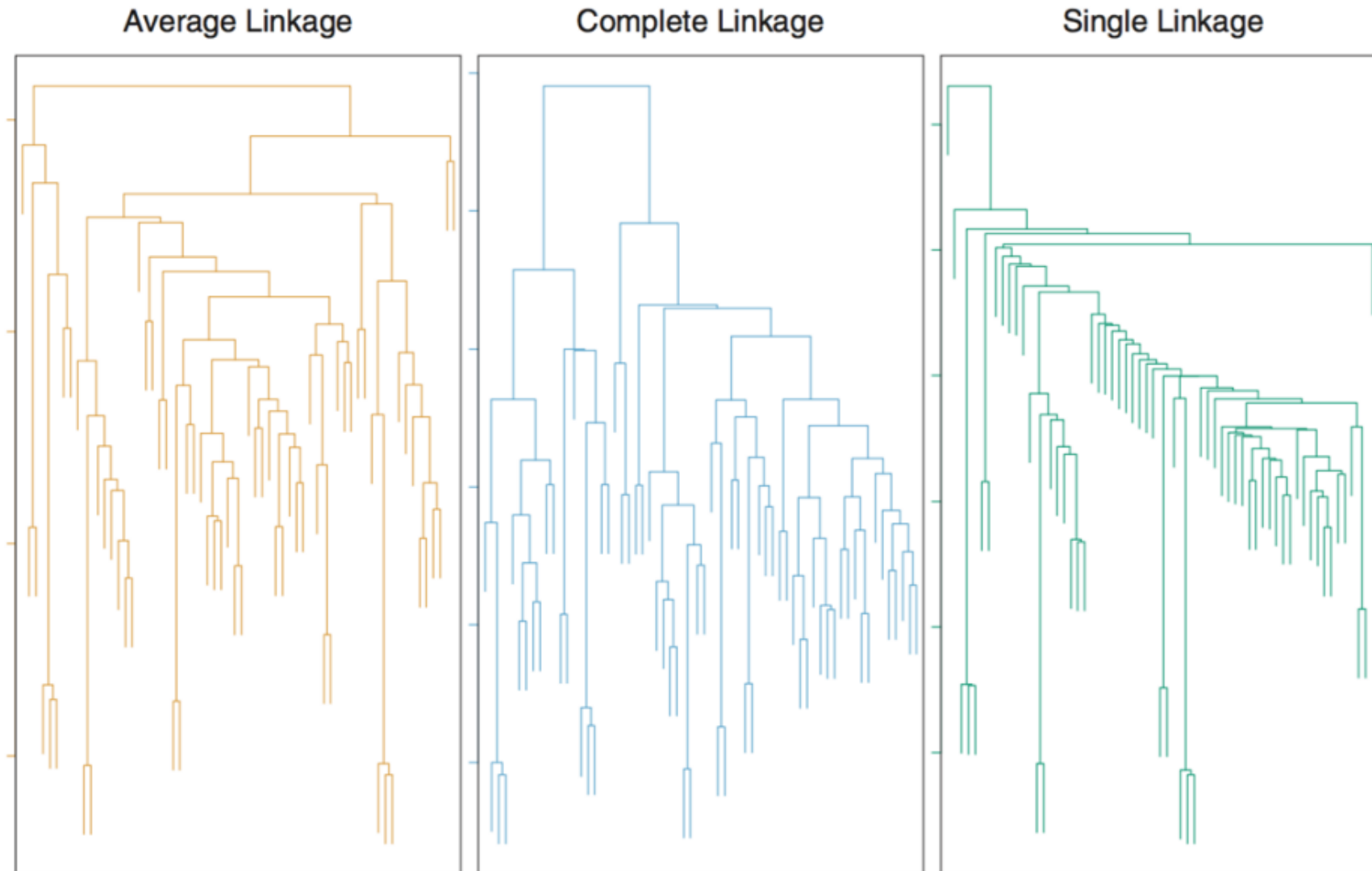
It's easy to compute Euclidean distance between two observations.

What is the distance or similarity between two groups or clusters of observations?

Linkage: defines the dissimilarity between two groups of observations. Most common types are complete, average, single, and centroid.



➔ DIFFERENT LINKAGE, DIFFERENT DENDROGRAM



→ HIERARCHICAL CLUSTERING PROS AND CONS

Pros:

- Don't have to choose a value of K (number of clusters) before running algorithm

Cons:

- Do have to pick where to cut the dendrogram to obtain clusters
- Sensitive to similarity measure and type of linkage used

➔ HIERARCHICAL CLUSTERING PROS AND CONS

```
from sklearn.cluster import AgglomerativeClustering
```

```
# Create the hierarchical clustering model
```

```
# n_clusters = 5 because we generated 5 groups
```

```
# linkage = 'ward' merges clusters while minimizing the increase in variance
```

```
hierarchical = AgglomerativeClustering(n_clusters=5, linkage='ward')
```

```
# Train the model and directly get the predicted cluster labels
```

```
predicted_labels = hierarchical.fit_predict(data)
```

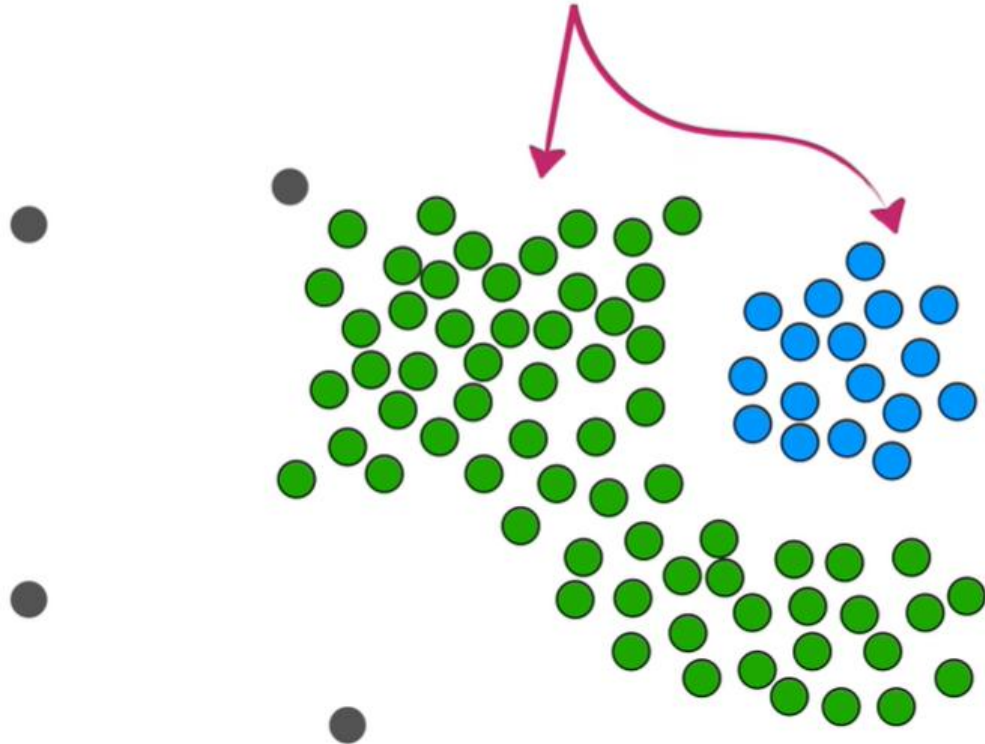
```
print("Hierarchical clustering completed.")
```

```
print("Number of clusters found:", len(np.unique(predicted_labels)))
```

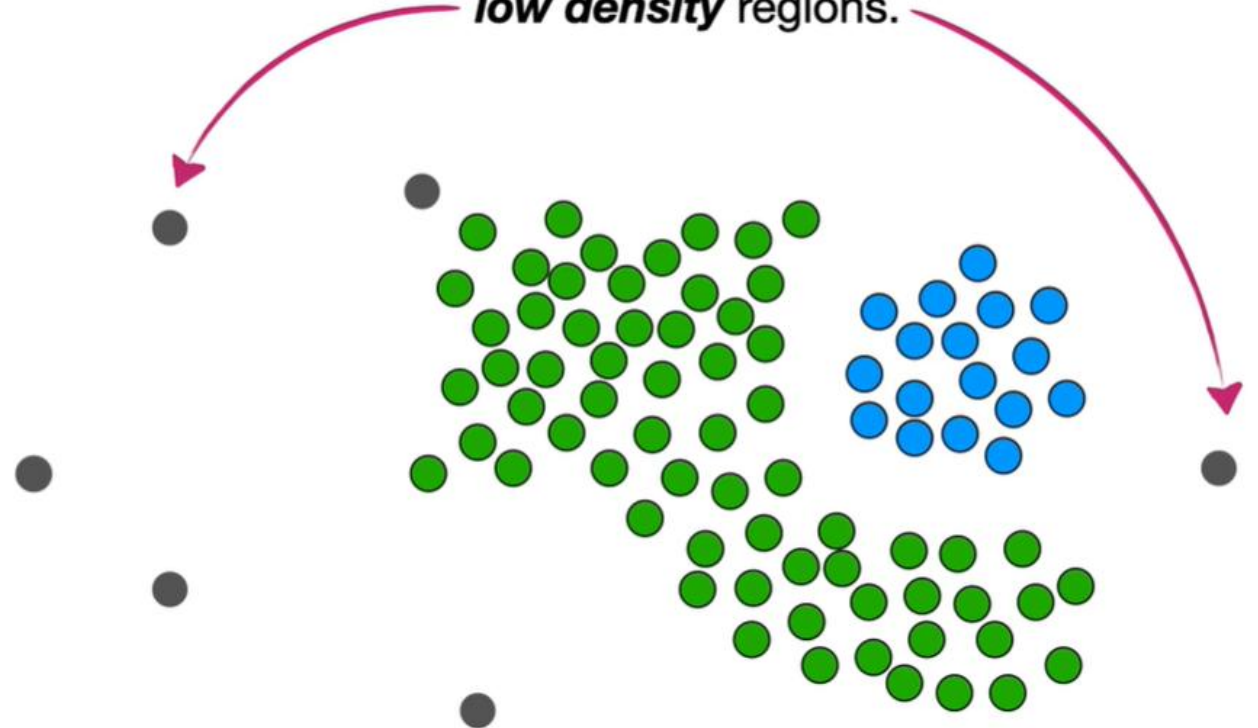
→ DBSCAN CLUSTERING

- DBSCAN (Density-Based Spatial Clustering of Applications with Noise) identifies clusters by densities of the points.

Clusters are in **high density** regions...

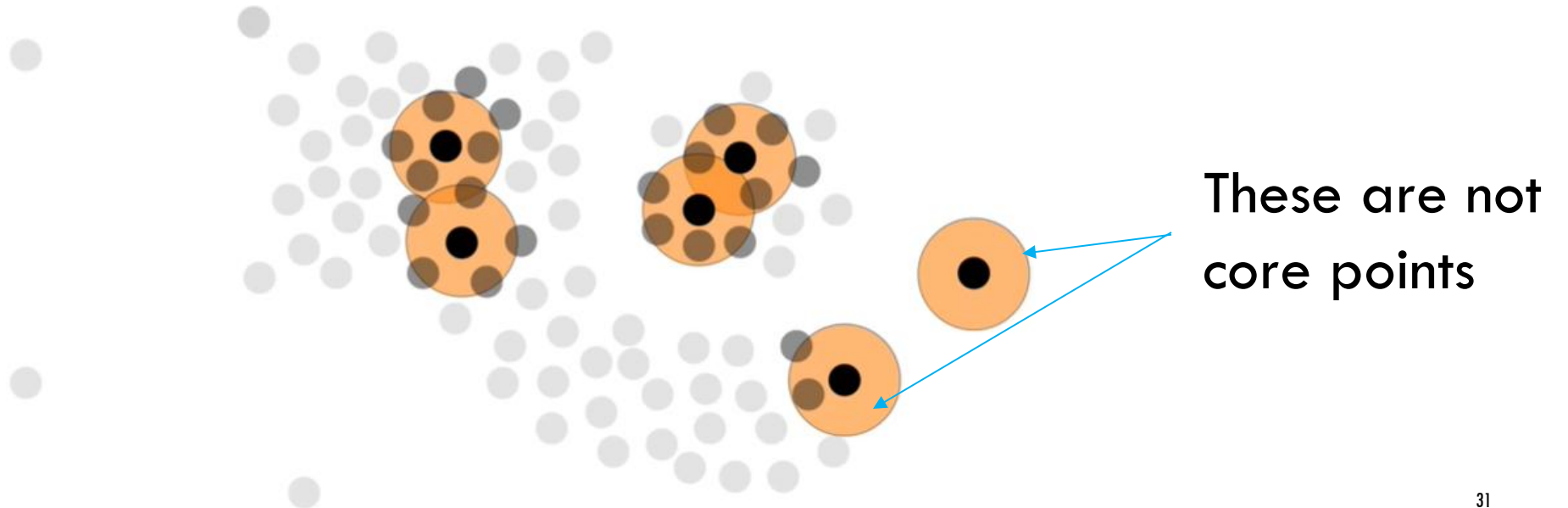


...and outliers tend to be in **low density** regions.



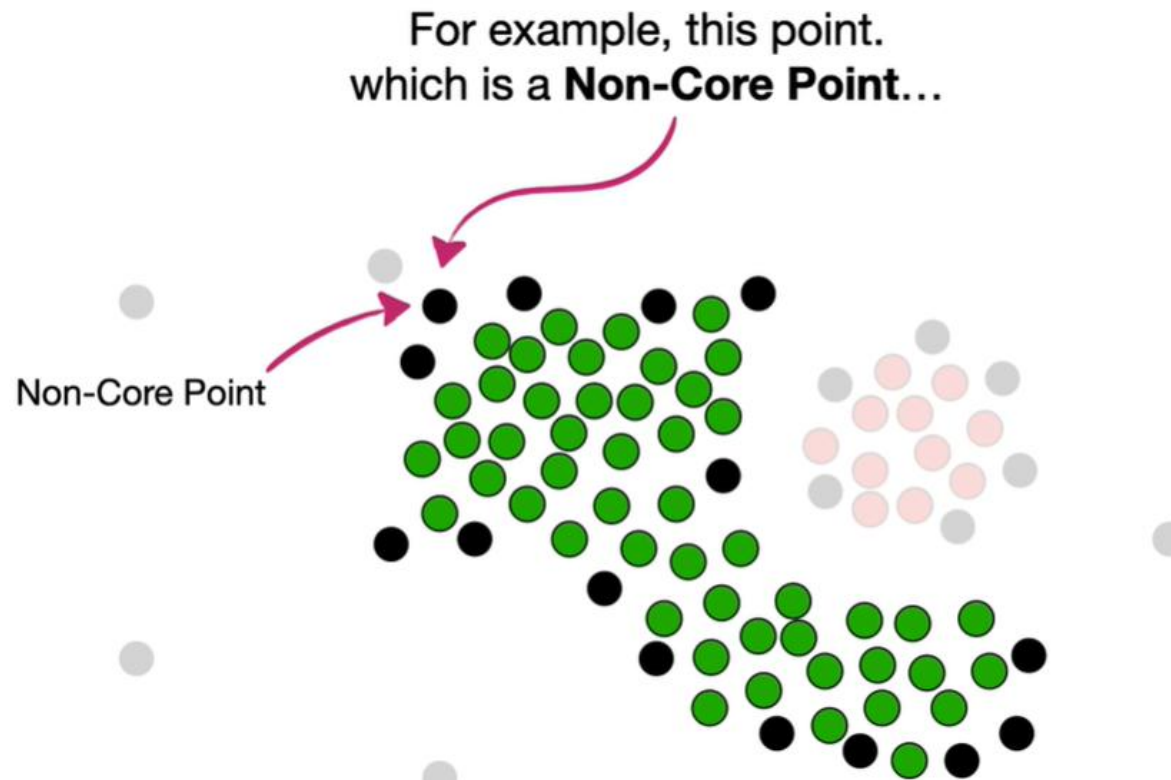
➔ DBSCAN CLUSTERING

- If we start with a core point that is close to at least a certain number of other points (orange circle).
- The radius of the orange circle is user defined in DBSCAN.
- The next core points that are close to the first core points are added to the same cluster and the used to extend it further.



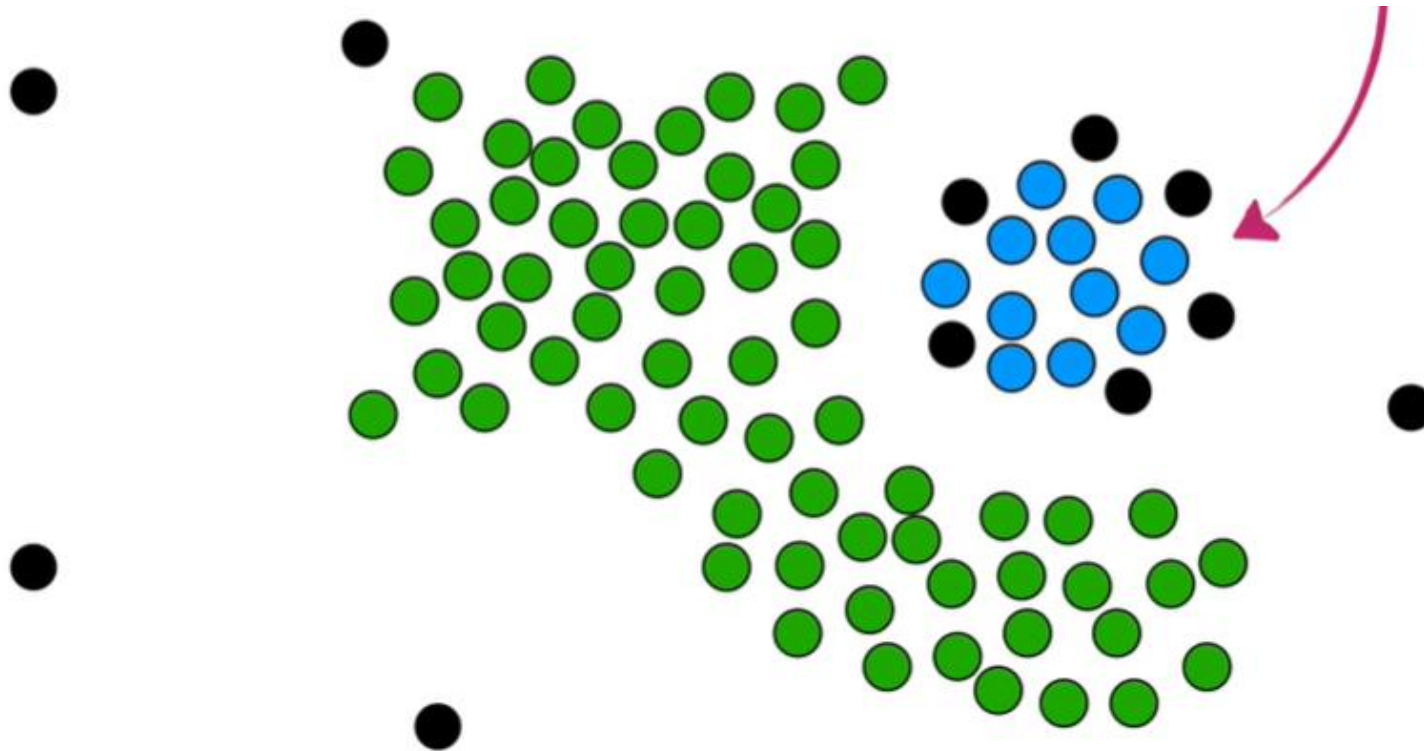
➔ DBSCAN CLUSTERING

- After we finish adding core-points to a certain cluster, we add all of the non-core points that are close to core points in the same cluster.
- Non-core points can only join a cluster. They can not extend it further.



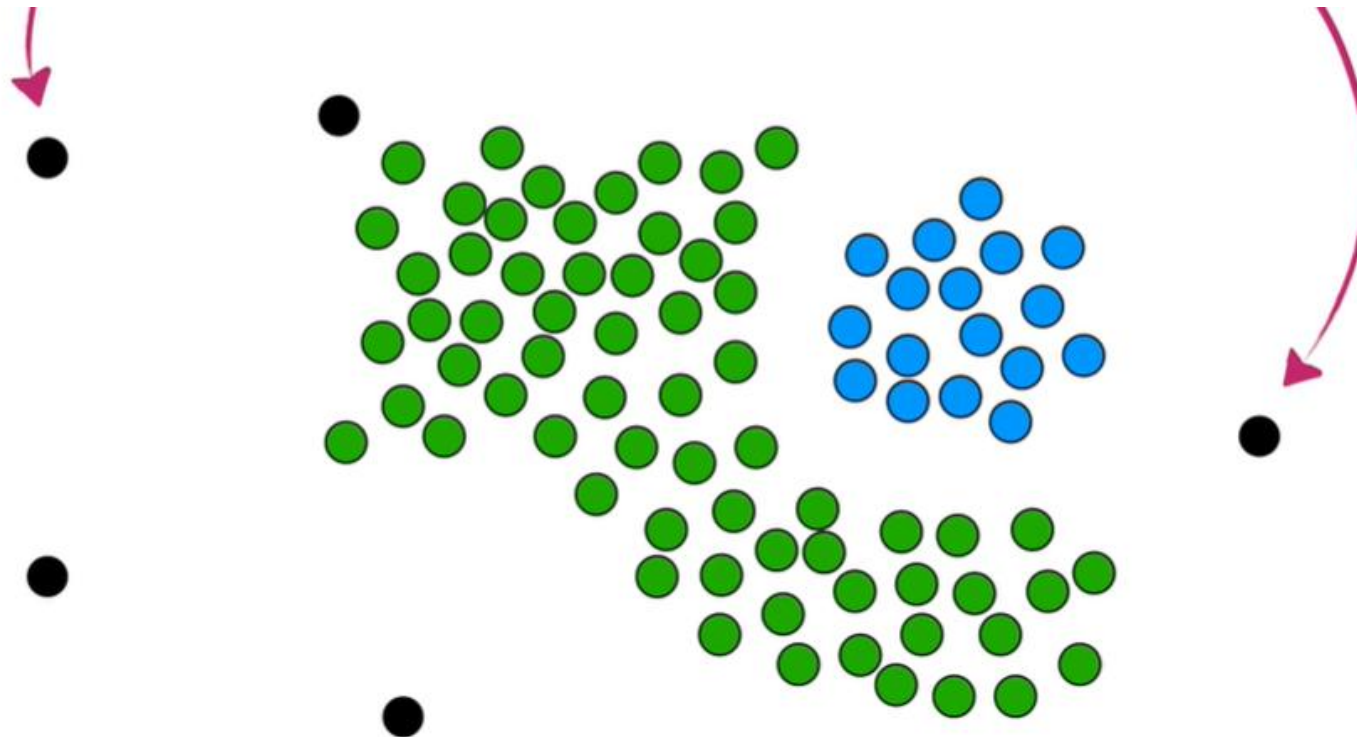
➔ DBSCAN CLUSTERING

- None of these core points are close to the first cluster they form a new, second cluster because they are close to each other.
- The non-core points that are close to the second cluster are added to it.



→ DBSCAN CLUSTERING

- Any remaining non-core points that are not close to core points in either cluster are not added to clusters.
- They are called outliers.



➔ DBSCAN CLUSTERING

Pros:

- No need to choose K
- Detects noise/outliers automatically
- Finds any cluster shape
- No centroid/spherical assumption
- Good for well-separated dense data

Cons:

- Sensitive to eps and min_samples
- Struggles with varying density
- Not good in high dimensions
- Slower than K-means on large data
- No predict() for new data

➔ DBSCAN CLUSTERING

```
from sklearn.cluster import DBSCAN
```

```
# Create the DBSCAN model
```

```
# eps controls the neighborhood size
```

```
# min_samples controls the minimum number of points needed to create a dense region
```

```
dbscan = DBSCAN(eps=0.4, min_samples=5)
```

```
# Fit the model and get the predicted labels
```

```
dbscan_labels = dbscan.fit_predict(data_scaled)
```

```
print(dbscan_labels[:20])
```