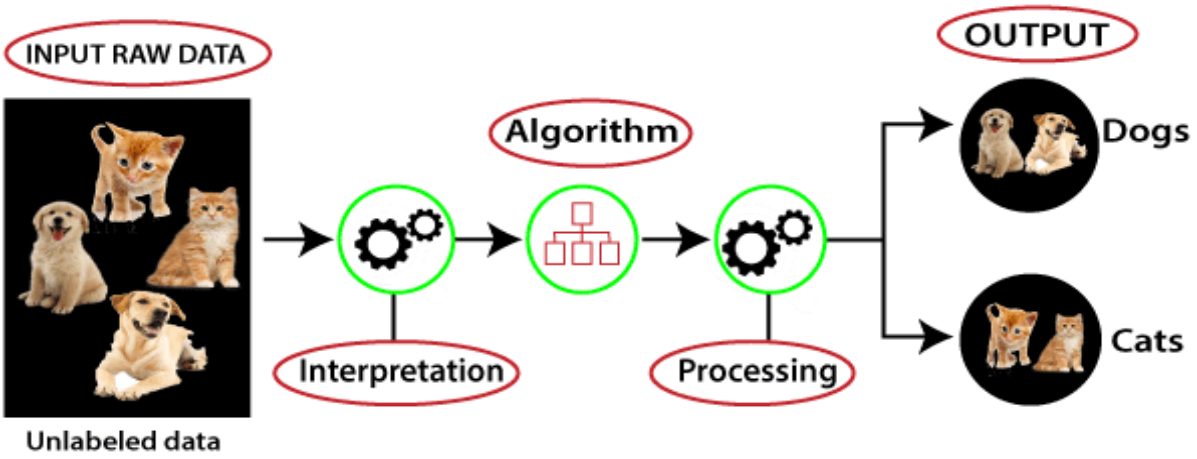
# UNIT-5

## Unsupervised Learning Techniques

**Clustering**

- Unsupervised learning algorithms are used when we don't have labeled data. These models try to identify hidden structures or patterns in the data.
- The main objective is to explore the data and find inherent groupings or relationships.



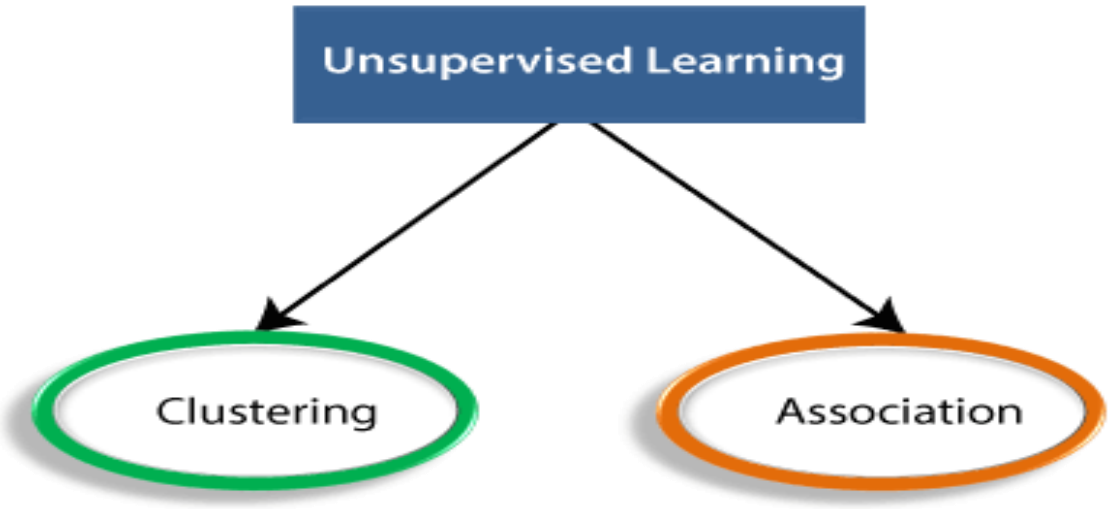

---

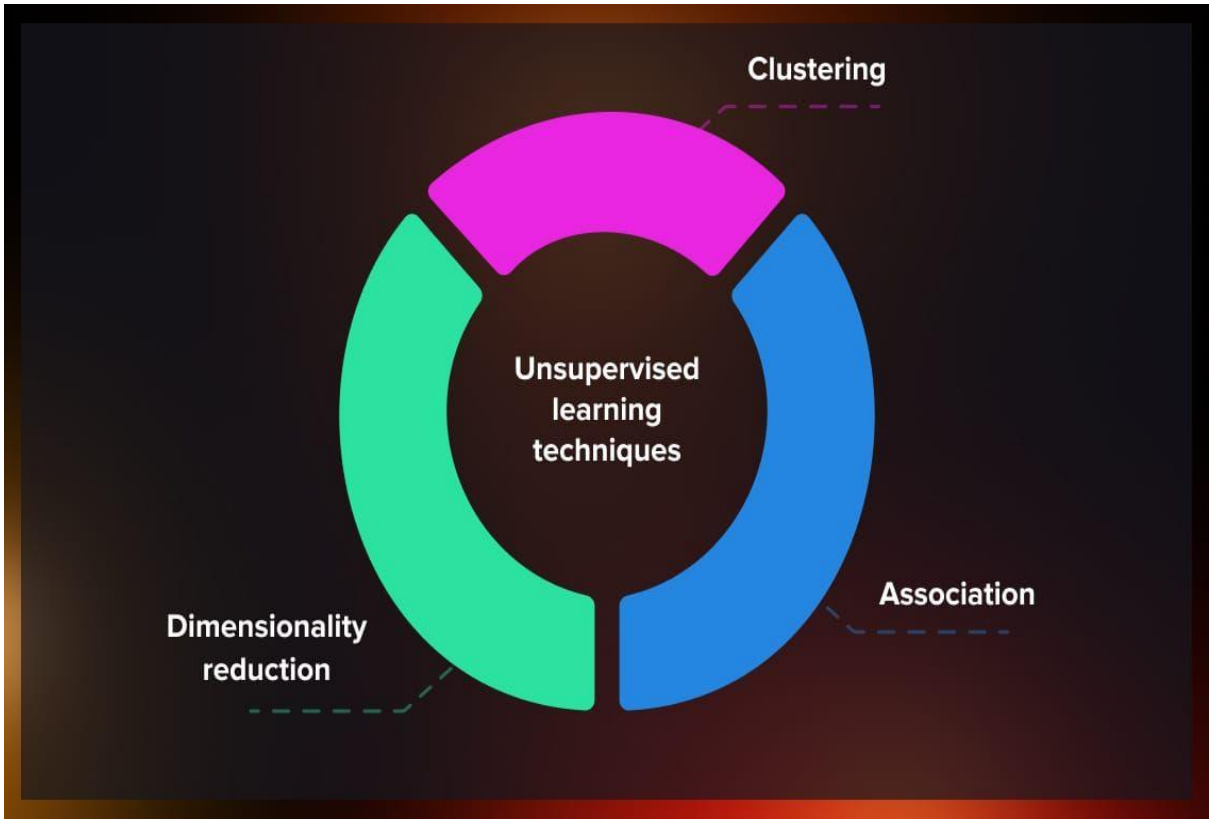
A) Supervised Learning: Like a student learning with a teacher providing answers and feedback.
B) Unsupervised Learning: Like a student discovering patterns and relationships on their own without answers provided.

| Feature | Supervised Learning | Unsupervised Learning |
|---|---|---|
| **Definition** | The model learns from labeled data (input-output pairs). | The model finds hidden patterns in unlabeled data. |
| **Objective** | To predict an outcome or classify data based on labeled examples. | To explore data structure or relationships without labels. |
| **Data Type** | Labeled data (with input and corresponding output). | Unlabeled data (only input data). |
| **Output** | Known target or output variable (classification or regression). | No predefined target; clustering or dimensionality reduction. |
| **Common Algorithms** | - Linear Regression<br>- Decision Trees<br>- Support Vector Machines (SVM)<br>- Neural Networks | - K-Means Clustering<br>- Hierarchical Clustering<br>- Principal Component Analysis (PCA)<br>- Autoencoders |
| **Example Use Cases** | - Spam detection in emails<br>- Disease diagnosis<br>- Credit scoring | - Customer segmentation<br>- Market basket analysis<br>- Anomaly detection |
| **Evaluation Metrics** | Accuracy, Precision, Recall, F1 Score, RMSE (Root Mean Squared Error) | Inertia (within-cluster variance), Silhouette Score, SSE (Sum of Squared Errors) |
| **Training Process** | Learns a function that maps in[illegible] to outputs. | Identifies hidden patterns or groupings in data. |

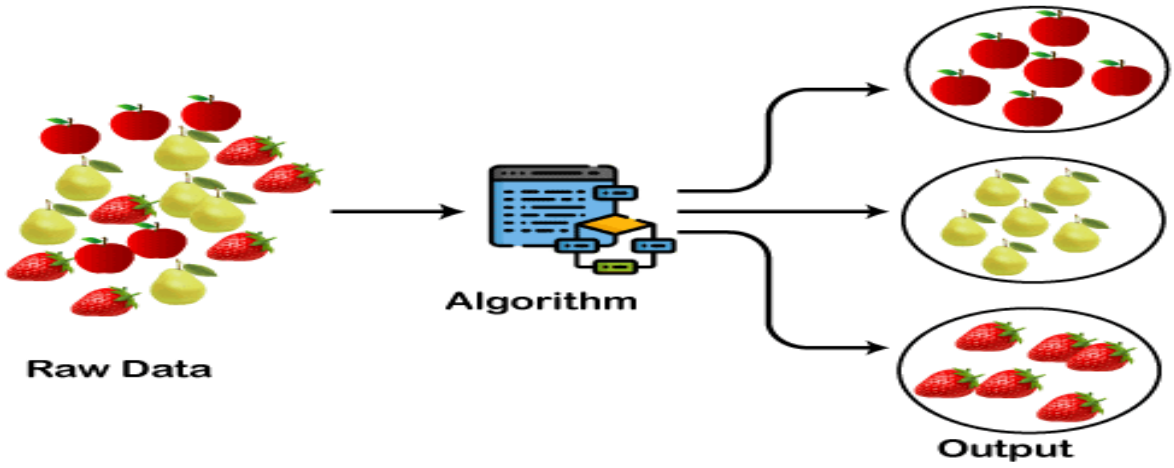
**Types of Unsupervised Learning Algorithm:**

- The unsupervised learning algorithm can be further categorized into two types of problems:






**Clustering:** Clustering is a method of grouping the objects into clusters such that objects with most similarities remains into a group and has less or no similarities with the objects of another group. Cluster analysis finds the commonalities between the data objects and categorizes them as per the presence and absence of those commonalities.




**Applications of Clustering**

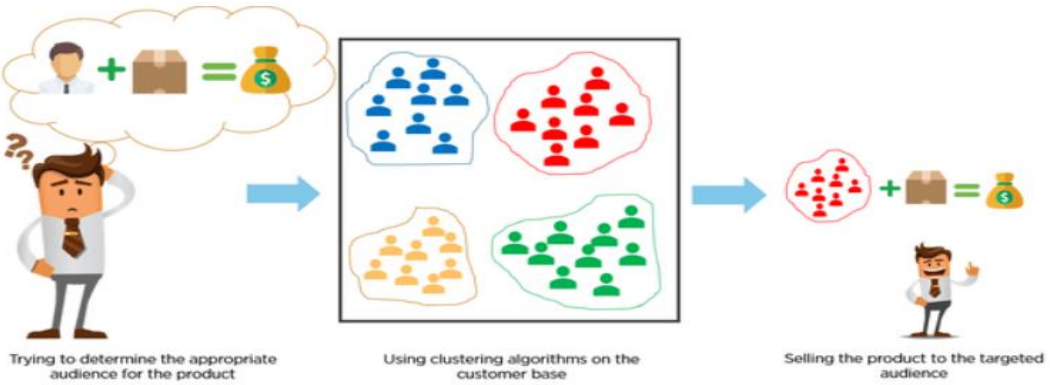
Clustering is used in a variety of fields and industries, such as:

- Market/customer Segmentation: Grouping customers with similar buying behavior for targeted marketing.
- Image Segmentation: Grouping pixels based on color or texture for image analysis.

- Document/Retail Clustering: Grouping similar documents based on content or topics.
- Anomaly Detection: Identifying unusual patterns that do not conform to expected behavior (e.g., fraud detection).
- Biological Data: Grouping genes or proteins that behave similarly.

**A) Customer Segmentation**

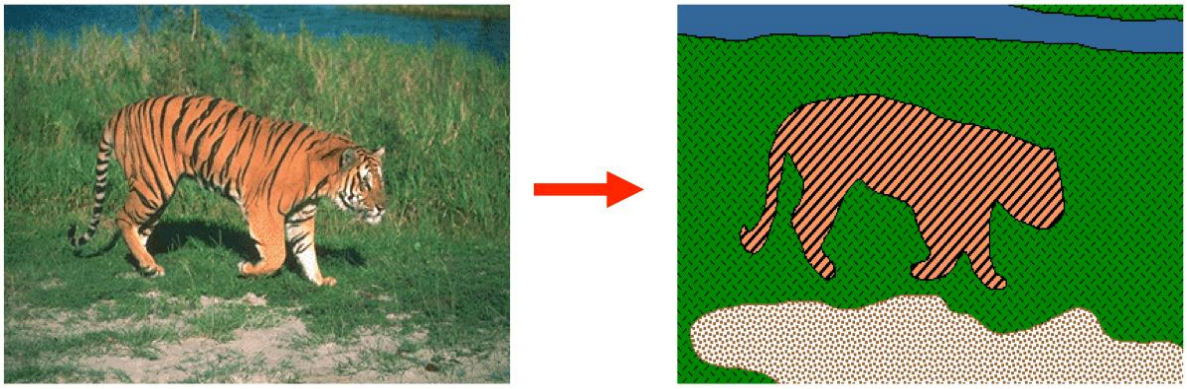
- Customers are categorized by using clustering algorithms according to their purchasing behavior or interests to develop focused marketing campaigns.
- Imagine you have 10M customers, and you want to develop customized or focused marketing campaigns. It is unlikely that you will develop 10M marketing campaigns, so what do we do? We could use clustering to group 10M customers into 25 clusters and then design 25 marketing campaigns instead of 10M.




---

**Image Segmentation**

Image segmentation is the classification of an image into different groups. This type of clustering is useful if you want to isolate objects in an image to analyze each object individually to check what it is.

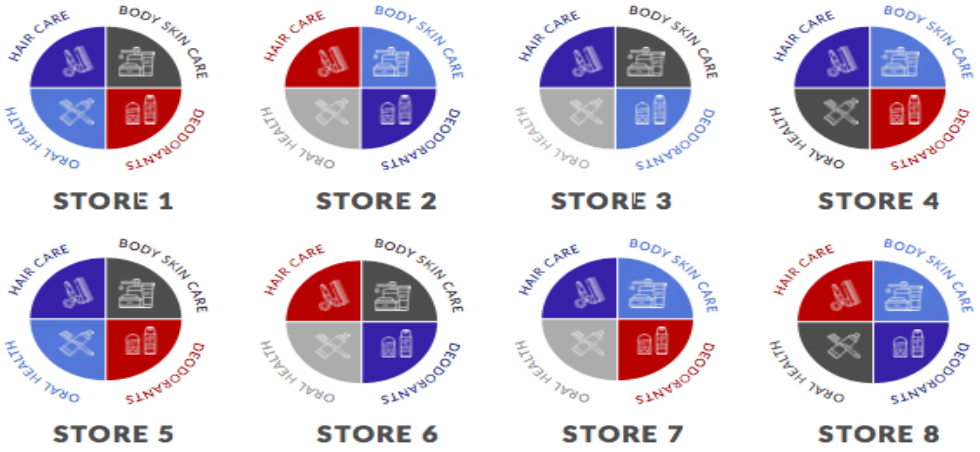


**Retail Clustering**

There are many opportunities for clustering in retail businesses. For example, you can gather data on each store and cluster at store level to generate insights that may tell you which

locations are similar to each other based on attributes like foot traffic, average store sales, number of SKUs, etc.

Another example could be clustering at a category level. In the diagram below, we have eight stores. Different colors represent different clusters. There are four clusters in this example.




**Clustering in Clinical Care / Disease Management**

Healthcare and Clinical Science is again one of those areas that are full of opportunities for clustering that are indeed very impactful in the field. One such example is research published by Komaru & Yoshida et al. 2020, where they collected demographics and laboratory data for 101 patients and then segmented them into 3 clusters.

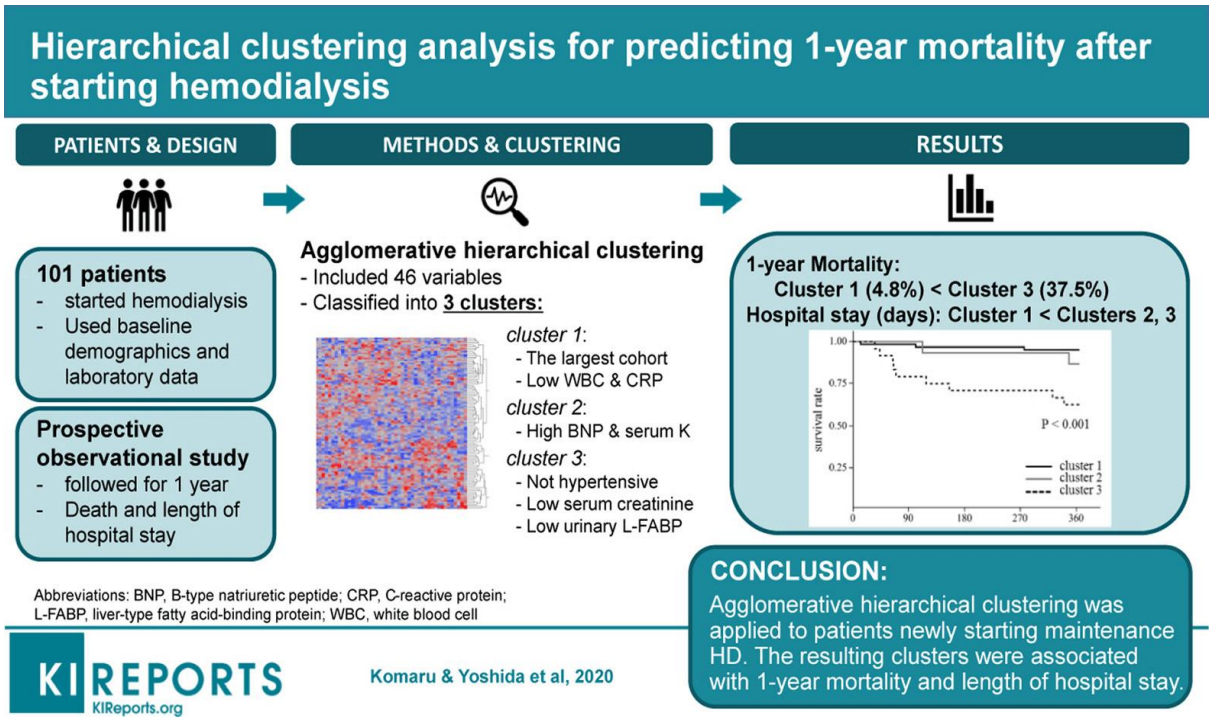
Each cluster was represented by different conditions. For example,

cluster 1 has patients with Low WBC & CRP.

Cluster 2 has patients with High BMP & Serum, and

Cluster 3 has patients with Low Serum.

Each cluster represents a different survival trajectory given the 1-year mortality after hemodialysis.




**Types of Clustering**

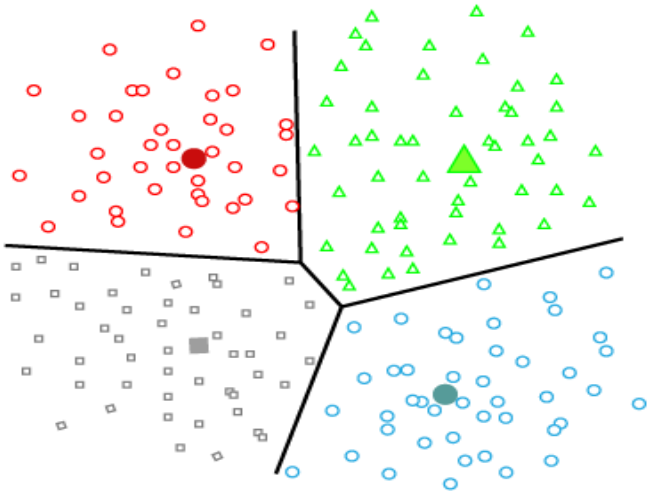
- **Hard Clustering**: In hard clustering, each data point belongs to one and only one cluster. This is the traditional approach, used in algorithms like K-means.
- **Soft Clustering**: In soft clustering, a data point can belong to multiple clusters with different degrees of membership. This is seen in algorithms like Fuzzy C-Means.
- **Partitioning Clustering**
- **Density-Based Clustering**
- **Distribution Model-Based Clustering**
- **Hierarchical Clustering**
- **Fuzzy Clustering**

**Popular Clustering Algorithms (Overview)**

- K-Means Clustering
- Hierarchical Clustering
- DBSCAN (Density-Based Spatial Clustering)
- Mean-Shift Clustering

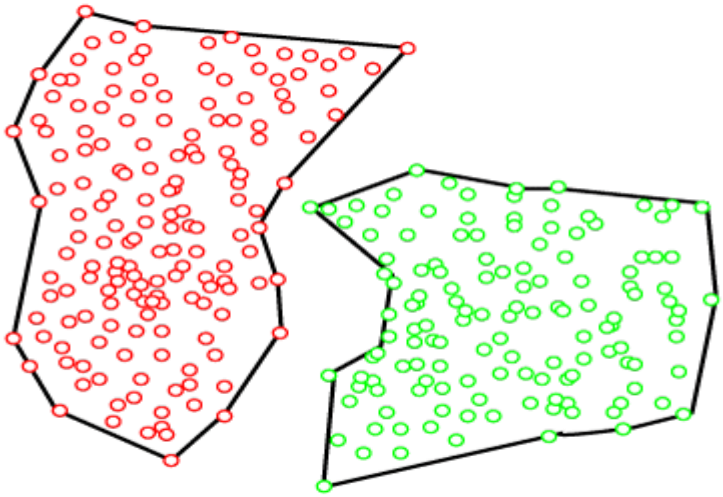
Partitioning Clustering

- It is a type of clustering that divides the data into non-hierarchical groups. It is also known as the **centroid-based method**. The most common example of partitioning clustering is the **K-Means Clustering algorithm**.



**Density-Based Clustering**

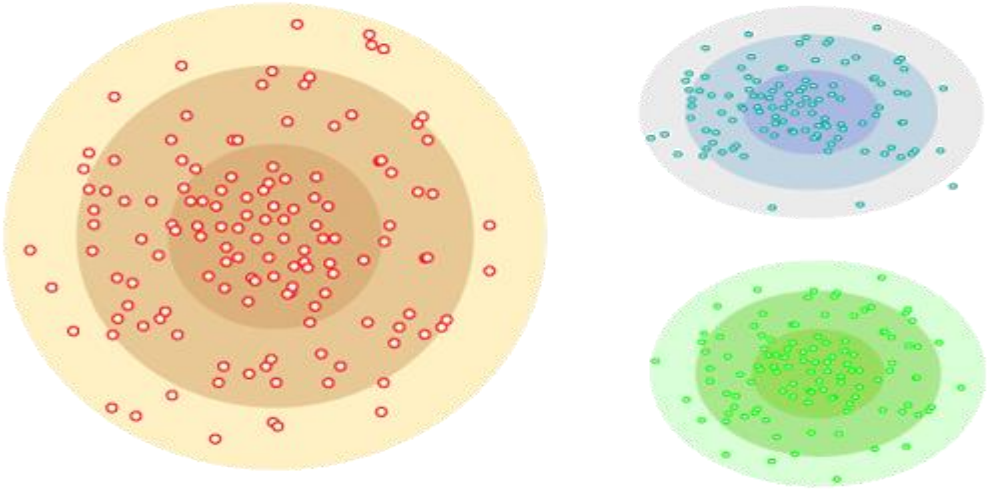
The density-based clustering method connects the highly-dense areas into clusters, and the arbitrarily shaped distributions are formed as long as the dense region can be connected.



**Distribution Model-Based Clustering**

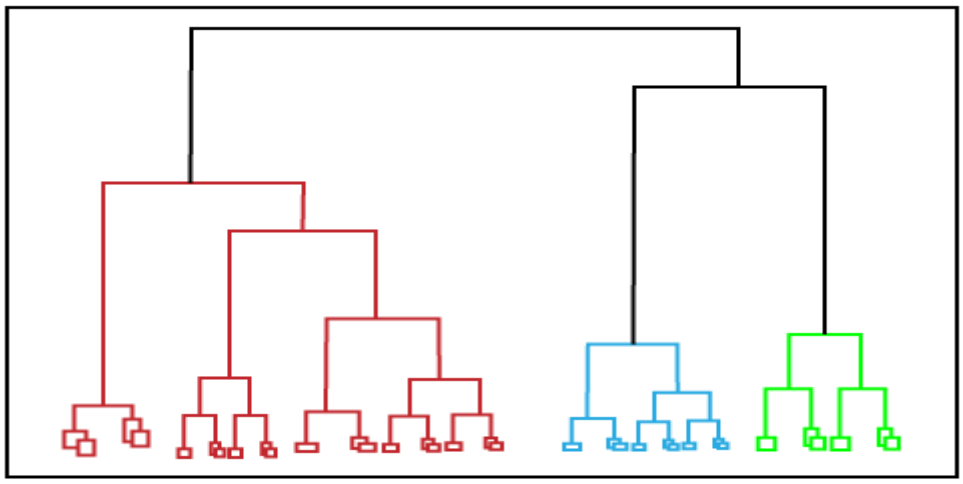
In the distribution model-based clustering method, the data is divided based on the probability of how a dataset belongs to a particular distribution. The grouping is done by assuming some distributions commonly **Gaussian Distribution**.

The example of this type is the **Expectation-Maximization Clustering algorithm** that uses Gaussian Mixture Models (GMM).



**Hierarchical Clustering**

Hierarchical clustering can be used as an alternative for the partitioned clustering as there is no requirement of pre-specifying the number of clusters to be created. In this technique, the dataset is divided into clusters to create a tree-like structure, which is also called a **dendrogram**.



# UNIT-5

## Unsupervised Learning Techniques

### K-Means Clustering

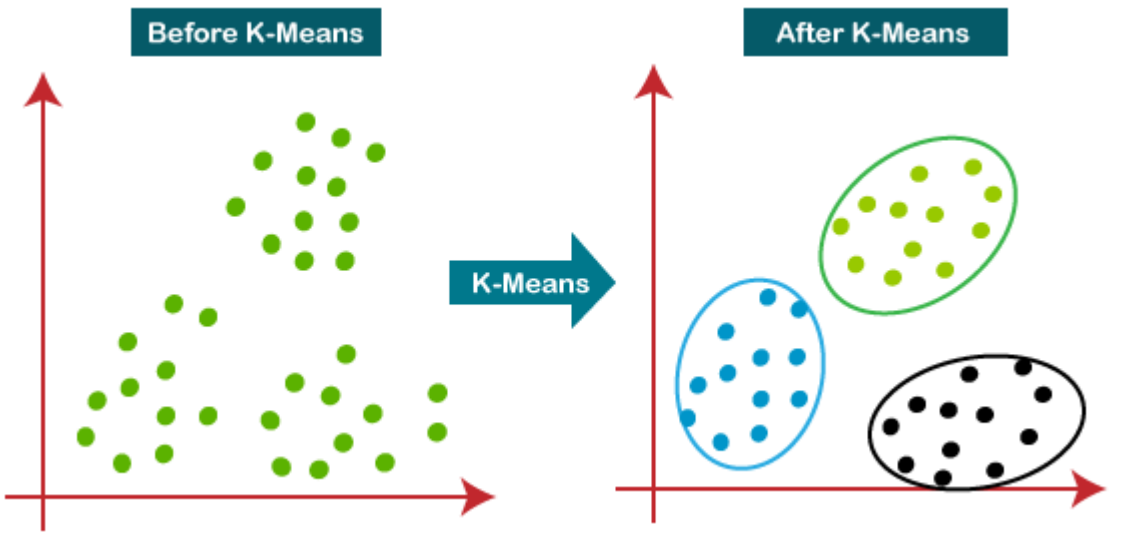
**Cluster analysis**

In unsupervised machine learning, cluster analysis groups unlabeled input data into meaningful output data. This output data can be used to understand the structure of the input data better or to make predictions about unlabeled data. There are many different ways to perform cluster analysis, but the most common method is to use a model that groups input data based on similarity.


---

**K-means Clustering**

Definition: K-Means is one of the most widely used clustering algorithms in unsupervised learning. It aims to partition a dataset into K clusters, where each data point belongs to the cluster with the nearest mean (centroid). The goal of K-Means is to minimize the variance within each cluster.




---

**How does the K-Means Algorithm Work?**

The working of the K-Means algorithm is explained in the below steps:

Step-1: Select the number K to decide the number of clusters.

Step-2: Select random K points or centroids. (It can be other from the input dataset).

Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.

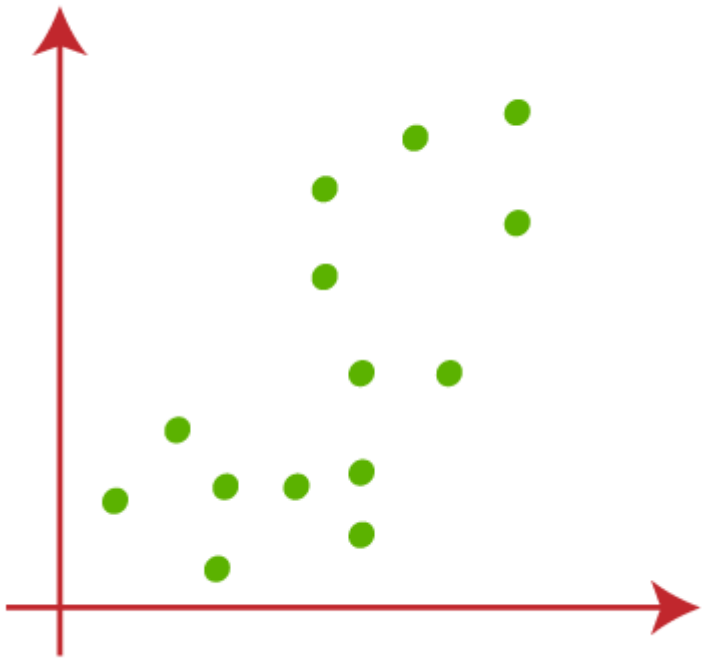
Step-4: Calculate the variance and place a new centroid of each cluster.

Step-5: Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.

Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.

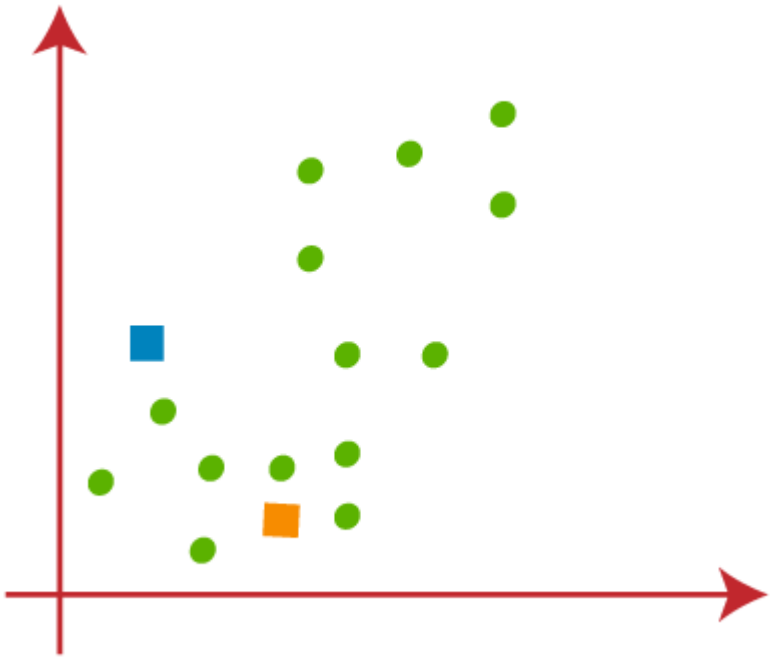
Step-7: The model is ready.

Suppose we have two variables M1 and M2. The x-y axis scatter plot of these two variables is given below:

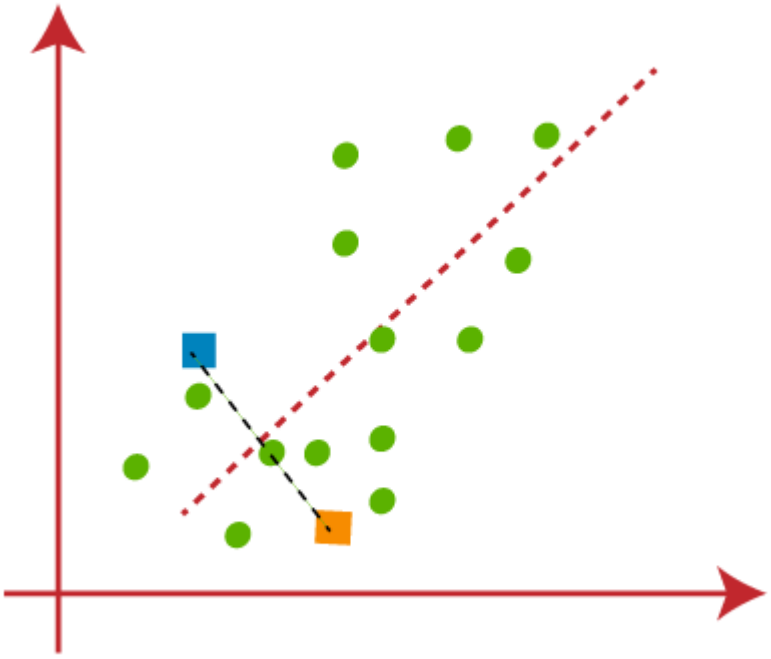


- Let's take number k of clusters, i.e., K=2, to identify the dataset and to put them into different clusters. It means here we will try to group these datasets into two different clusters.
- We need to choose some random k points or centroid to form the cluster. These points can be either the points from the dataset or any other point. So, here we are selecting the below two points as k points, which are not the part of our dataset.

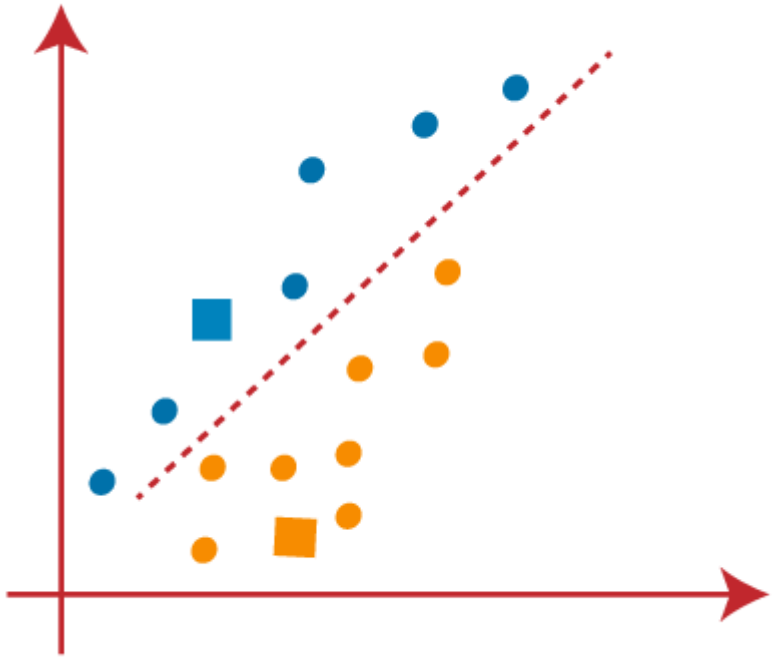
Consider the below image:



- Now we will assign each data point of the scatter plot to its closest K-point or centroid. We will compute it by applying some mathematics that we have studied to calculate the distance between two points. So, we will draw a median between both the centroids. Consider the below image:

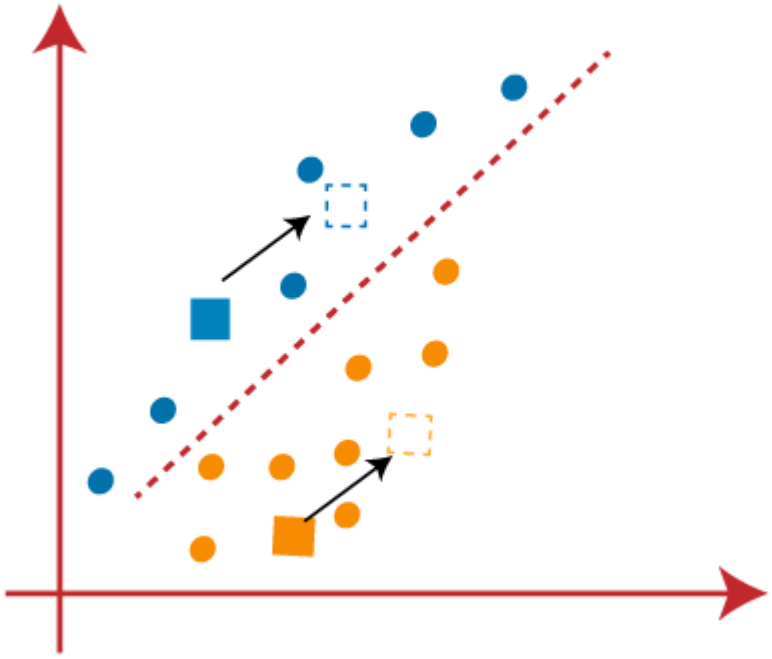


From the above image, it is clear that points left side of the line is near to the K1 or blue centroid, and points to the right of the line are close to the yellow centroid. Let's color them as blue and yellow for clear visualization.

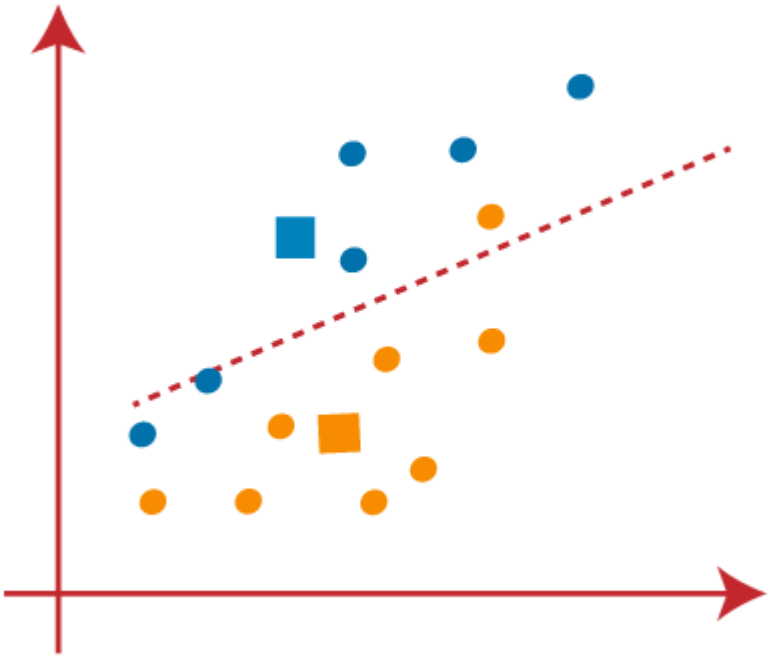


- As we need to find the closest cluster, so we will repeat the process by choosing **a new centroid**. To choose the new centroids, we will compute the center of gravity of

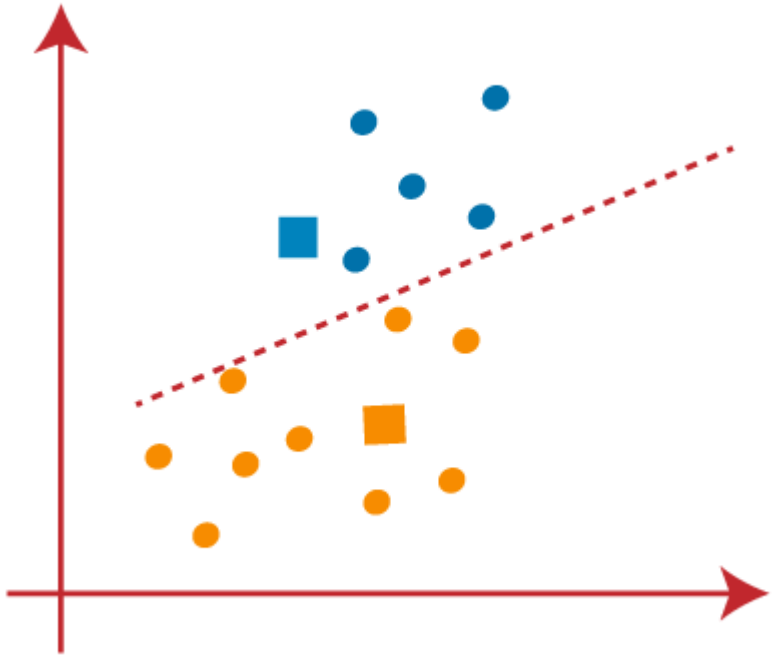
these centroids, and will find new centroids as below:



- Next, we will reassign each datapoint to the new centroid. For this, we will repeat the same process of finding a median line. The median will be like below image:

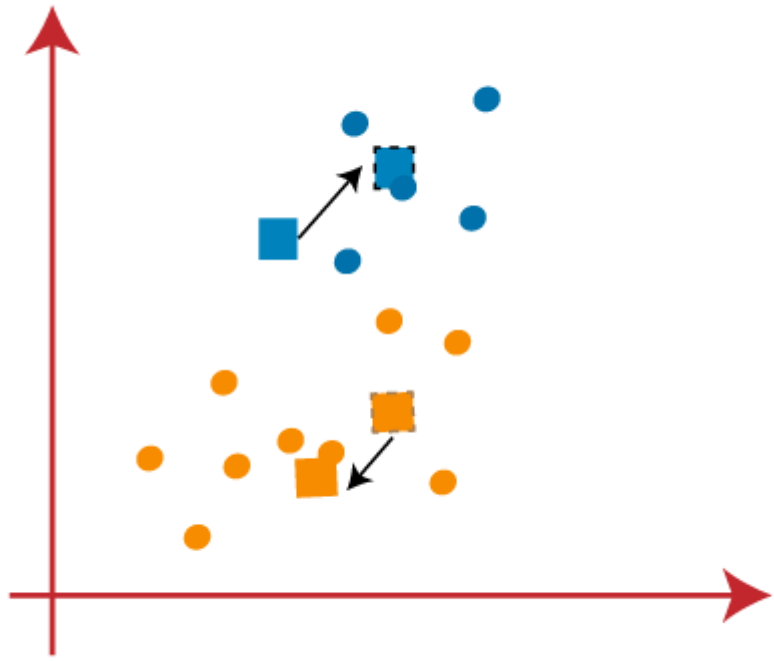


From the above image, we can see, one yellow point is on the left side of the line, and two blue points are right to the line. So, these three points will be assigned to new centroids.

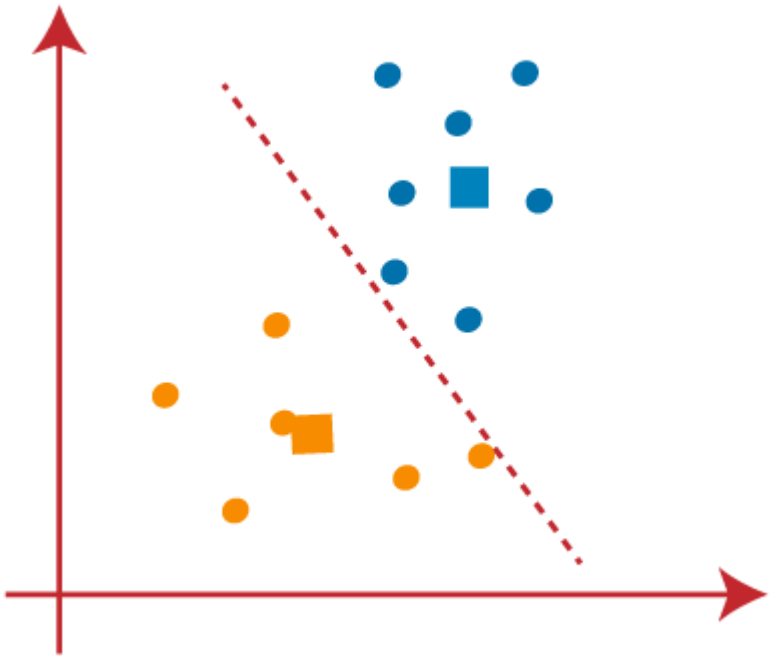


As reassignment has taken place, so we will again go to the step-4, which is finding new centroids or K-points.

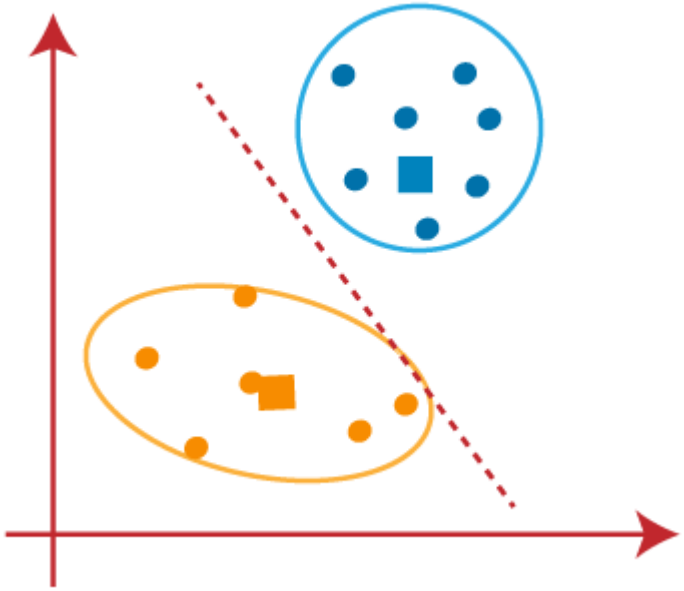
- We will repeat the process by finding the center of gravity of centroids, so the new centroids will be as shown in the below image:



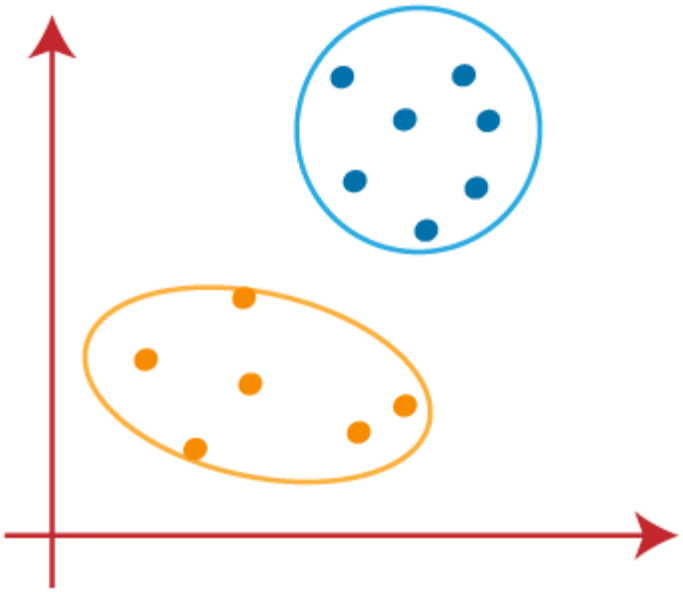
- As we got the new centroids so again will draw the median line and reassign the data points. So, the image will be:



- We can see in the above image; there are no dissimilar data points on either side of the line, which means our model is formed. Consider the below image:



As our model is ready, so we can now remove the assumed centroids, and the two final clusters will be as shown in the below image:



How to choose the value of "K number of clusters" in K-means Clustering?

The performance of the K-means clustering algorithm depends upon highly efficient clusters that it forms. But choosing the optimal number of clusters is a big task. There are some different ways to find the optimal number of clusters, but here we are discussing the most appropriate method to find the number of clusters or value of K. The method is given below:

**Limits of K-Means**

While K-Means is a simple and effective algorithm, it has several limitations:

1. Predefined K:

Limitation: The number of clusters (K) must be specified beforehand. If you don't know the number of clusters in advance, this can be a challenge.

2. Sensitivity to Initial Centroids:

Limitation: K-Means is sensitive to the initial placement of centroids. If the initial centroids are poorly chosen, it can lead to poor clustering results.

3. Sensitivity to Outliers:

Limitation: K-Means is sensitive to outliers since they can significantly influence the centroid position.

4. Convergence to Local Minima:

Limitation: K-Means can converge to local minima, meaning it might not find the best possible cluster configuration. Running the algorithm multiple times with different initializations can help mitigate this.

**Solutions to Limitations:**

- **Elbow Method**: To determine the optimal K by looking for a "knee" or "elbow" in the inertia plot (sum of squared distances of points to their centroids).
- **K-Means++**: A smarter initialization technique that spreads out the initial centroids, improving convergence and results.
- **Use of other algorithms**: When K-Means is unsuitable, alternative clustering algorithms like DBSCAN (density-based) or hierarchical clustering can be used.

# UNIT-5

## Unsupervised Learning Techniques

### Image segmentation using Clustering

**Cluster analysis**

In unsupervised machine learning, cluster analysis groups unlabeled input data into meaningful output data. This output data can be used to understand the structure of the input data better or to make predictions about unlabeled data. There are many different ways to perform cluster analysis, but the most common method is to use a model that groups input data based on similarity.


---

Image segmentation is the classification of an image into different groups. Many kinds of research have been done in the area of image segmentation using clustering. There are different methods and one of the most popular methods is **K-Means clustering algorithm**.


**Figure 1:** (a) is the original image; (b) and (c) are the segmentation results.

Image segmentation is the process of partitioning a digital image into multiple distinct regions containing each pixel(sets of pixels, also known as superpixels) with similar attributes.

The goal of Image segmentation is to change the representation of an image into something that is more meaningful and easier to analyze.

Image segmentation is typically used to locate objects and boundaries(lines, curves, etc.) in images. More precisely, Image Segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain characteristics.

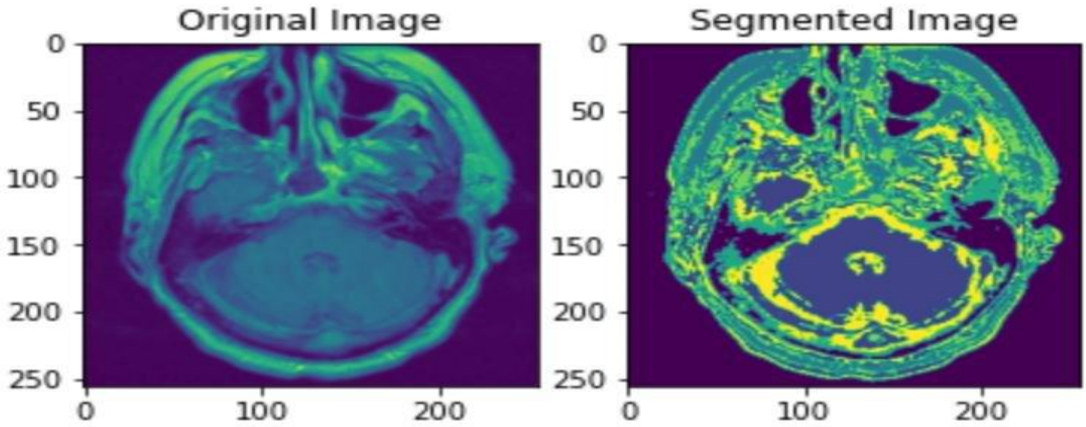
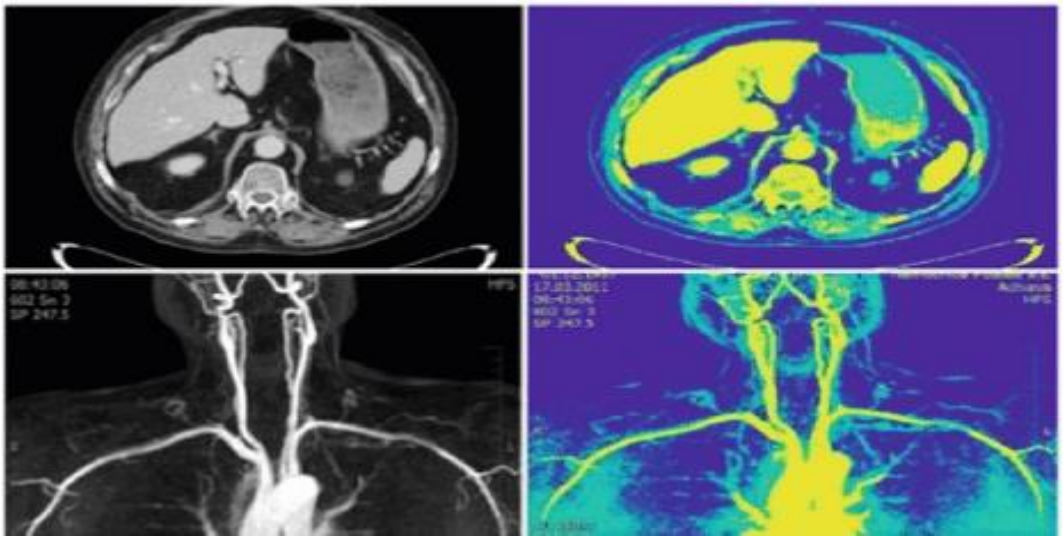



**Image Segmentation using Clustering** is a technique that involves partitioning an image into multiple segments or regions, making it easier to analyze or understand. Clustering groups pixels with similar characteristics (like color, intensity, or texture) together, thereby identifying distinct regions in the image.



**Popular Clustering Techniques for Image Segmentation**

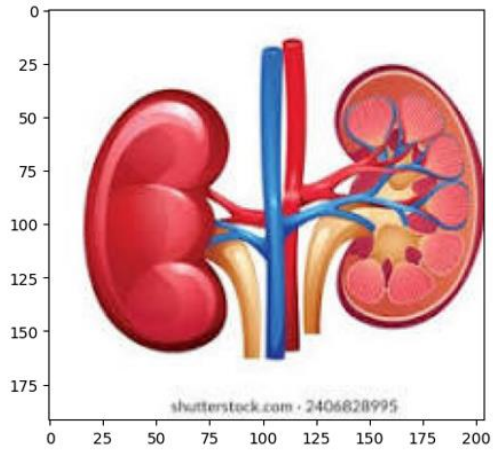
1. K-Means Clustering
2. Mean-Shift Clustering

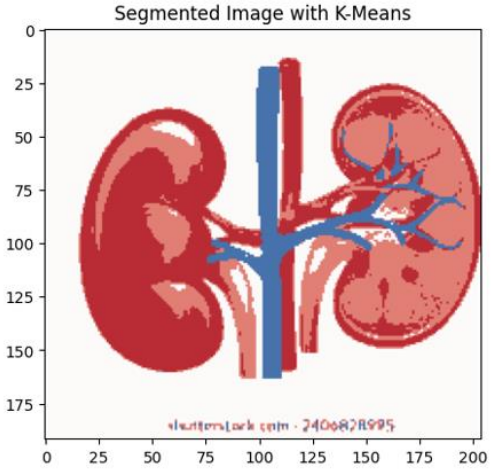
3. DBSCAN (Density-Based Spatial Clustering)
4. Gaussian Mixture Models (GMM)
5. ***K*-Means clustering** algorithm is an unsupervised algorithm and it is used to segment the interest area from the background. It clusters, or partitions the given data into K-clusters or parts based on the K-centroids.
6. The algorithm is used when you have unlabeled data(i.e. data without defined categories or groups). The goal is to find certain groups based on some kind of similarity in the data with the number of groups represented by K.

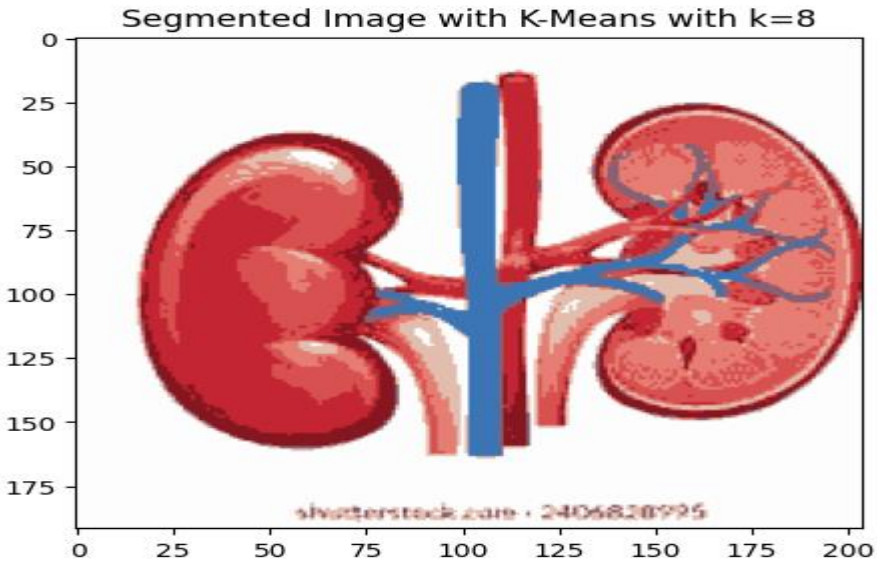

**Image Segmentation Steps:**

1. **Convert the Image Data** to a 2D array where each row represents a pixel and each column represents a feature (e.g., RGB values).
2. **Set the Number of Clusters (k)** based on the desired number of segments.
3. **Run the K-Means Algorithm** to assign each pixel to one of the k clusters.
4. **Replace Pixel Values** with the cluster centroids to create segmented regions.

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Read the image
image = cv2.imread('/content/kid.jpg')
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Reshape the image to a 2D array of pixels
pixel_values = image.reshape((-1, 3))
pixel_values = np.float32(pixel_values)

# Define criteria and apply K-Means
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.2)
k = 4  # Number of clusters
_, labels, centers = cv2.kmeans(pixel_values, k, None, criteria, 10, cv2.KMEANS_RANDOM_CENTERS)

# Convert centers back to uint8 and flatten labels
centers = np.uint8(centers)
segmented_image = centers[labels.flatten()]
segmented_image = segmented_image.reshape(image.shape)

# Display the result
plt.imshow(image)
plt.imshow(segmented_image)
plt.title('Segmented Image with K-Means')
plt.show()
```






As you can see with an increase in the value of K, the image becomes clearer because the K-means algorithm can classify more classes/cluster of colors.

**Applications of Clustering in Image Segmentation**

1. **Medical Imaging**: Identifying regions of interest in MRI or CT scans.
2. **Object Detection**: Separating foreground objects from background.
3. **Satellite Imaging**: Land use and terrain classification.
4. **Photo Editing**: Background replacement or selective editing.
5. **Anomaly Detection**: Identifying unusual patterns in images.

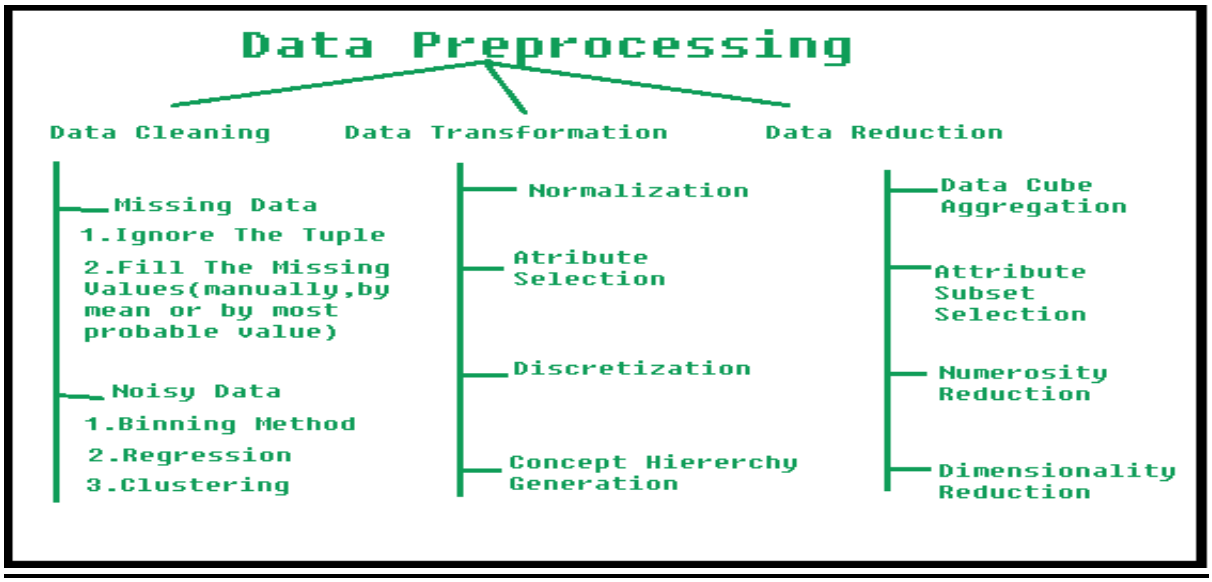
# UNIT-5

## Unsupervised Learning Techniques

### Using clustering for Preprocessing, and semi supervised learning

Clustering for preprocessing is a technique where unsupervised clustering algorithms are applied to data before it undergoes other machine learning or analysis steps. This approach can be especially useful for:

1. Data Reduction: Simplifying a large dataset by grouping similar data points into clusters and representing each cluster with a prototype or centroid.
2. Noise Detection: Identifying outliers or noisy data points that do not fit well within any cluster.
3. Feature Engineering: Adding cluster labels as new features to enhance predictive models.
4. Segmentation: Grouping data into meaningful subsets to better tailor subsequent analysis or modeling tasks.




**Popular Clustering Algorithms for Preprocessing:**

1. **K-Means:**

Use Case: Partitioning data into a fixed number of kkk clusters.

Pros: Fast and easy to interpret.

Cons: Sensitive to outliers and initialization.

2. **Hierarchical Clustering:**

Use Case: Creating a tree-like structure (dendrogram) for grouping data at multiple levels.

Pros: No need to specify the number of clusters beforehand.

Cons: Computationally intensive for large datasets.

3. **DBSCAN (Density-Based Spatial Clustering of Applications with Noise):**

Use Case: Identifying clusters based on data density, useful for noisy datasets.

Pros: Detects outliers and clusters of varying shapes.

Cons: Parameters can be hard to tune.

4. **Gaussian Mixture Models (GMM):**

Use Case: Modeling data as a mixture of Gaussian distributions.

Pros: Provides probabilistic cluster assignments.

Cons: Requires specifying the number of clusters.

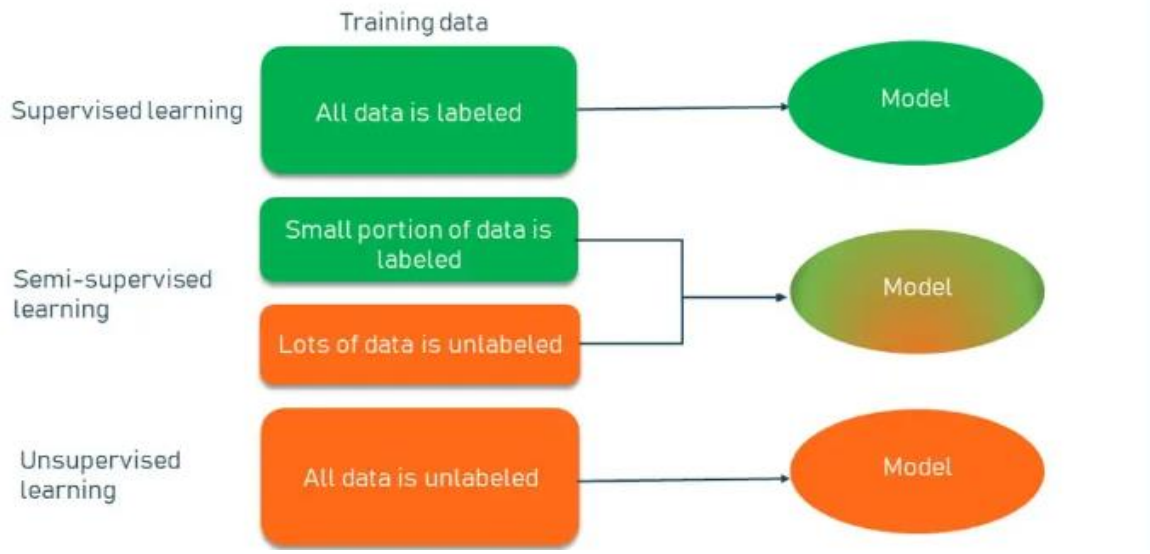
**Steps in Using Clustering for Preprocessing**

1. **Data Preparation**:
    1. Normalize or standardize data if clustering algorithms are distance-based (e.g., K-Means).
2. **Apply Clustering**:
    1. Run a clustering algorithm to divide the data into clusters.
3. **Assign Cluster Labels**:
    1. Each data point gets a cluster label, which can be added as a new feature or used to eliminate unwanted data.
4. **Filter or Transform Data**:
    1. **Filter out**: Exclude noisy clusters or outliers.
    2. **Transform**: Represent each cluster with its centroid or apply dimensionality reduction.

**What is Semi-Supervised Learning?**

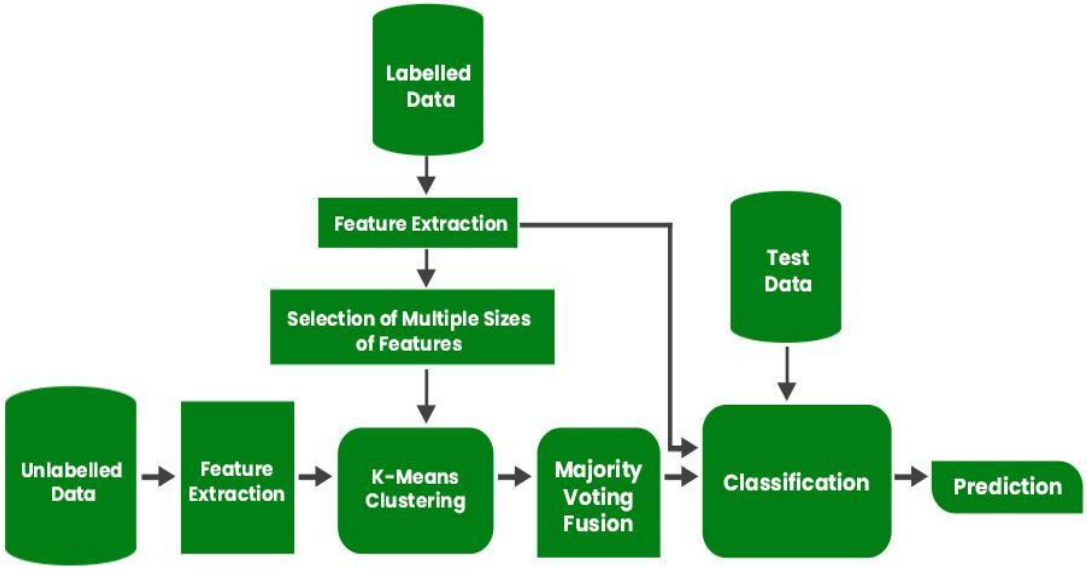
Semi-supervised learning is a type of machine learning that falls in between supervised and unsupervised learning. It is a method that uses a small amount of labeled data and a large amount of unlabeled data to train a model. The goal of semi-supervised learning is to learn a function that can accurately predict the output variable based on the input variables, similar to supervised learning. However, unlike supervised learning, the algorithm is trained on a dataset that contains both labeled and unlabeled data.

Semi-supervised learning is particularly useful when there is a large amount of unlabeled data available, but it's too expensive or difficult to label all of it.




**What is semi-supervised learning with clustering?**

Semi-supervised learning with clustering is a type of machine learning that uses both labeled and unlabeled data to train a model. Clustering is a process of grouping similar data points together based on some measure of similarity, such as distance or density. Semi-supervised learning with clustering uses clustering algorithms to assign labels to unlabeled data, and then uses both labeled and unlabeled data to train a supervised learning model, such as a classifier or a regressor.




**Why use semi-supervised learning with clustering?**

Semi-supervised learning with clustering can offer several benefits over purely supervised or unsupervised learning methods. First, it can leverage the large amount of unlabeled data that is often available in real-world scenarios, and use it to enhance the accuracy and generalization of the model. Second, it can reduce the cost and effort of labeling data manually, which can be tedious, subjective, or error-prone. Third, it can discover hidden patterns or structures in the data that may not be obvious or captured by the existing labels.

**How does semi-supervised learning with clustering work?**

Semi-supervised learning with clustering can be implemented in various ways, depending on the type of data, the clustering algorithm, and the supervised learning model. A common approach is to first apply a clustering algorithm to the unlabeled data, and then combine the labeled and unlabeled data, using the cluster labels as pseudo-labels for the unlabeled data. Next, a supervised learning model is trained on the combined data, with the original labels and pseudo-labels serving as target variables. Finally, the model can be evaluated on a test set, and optionally refined or adjusted.

**What are some examples of semi-supervised learning with clustering in AI?**

Semi-supervised learning with clustering has been applied to various domains and tasks in AI, such as natural language processing for text classification, sentiment analysis, topic modeling, or named entity recognition.

For example, one can cluster unlabeled documents based on their word embeddings and use the cluster labels as pseudo-labels to train a text classifier. This same approach can be used for computer vision tasks like image classification, object detection, face recognition, or semantic segmentation.

Clustering can also be used in bioinformatics for gene expression analysis, protein function prediction, or disease diagnosis. For instance, one can cluster unlabeled gene expression profiles based on their similarity and use the generated labels as pseudo-labels to train a gene classifier.

# UNIT-5

## Unsupervised Learning Techniques

## DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

In data science and machine learning, the ability to uncover hidden patterns and group similar data points is an important skill. Clustering algorithms play a key role in this process.

Clustering is a fundamental machine learning and data science technique that involves grouping similar data points together. It's an unsupervised learning method, meaning it doesn't require labeled data to find patterns.

The primary goal of clustering is to:

- Simplify large datasets into meaningful subgroups
- Identify natural groupings within data
- Reveal hidden patterns and structures

**DBSCAN (Density-Based Spatial Clustering of Applications with Noise)** is a popular unsupervised clustering algorithm used to group points in a dataset based on density. Unlike *k-means*, DBSCAN doesn't require specifying the number of clusters in advance and is capable of identifying clusters of arbitrary shapes, making it useful in a variety of applications.

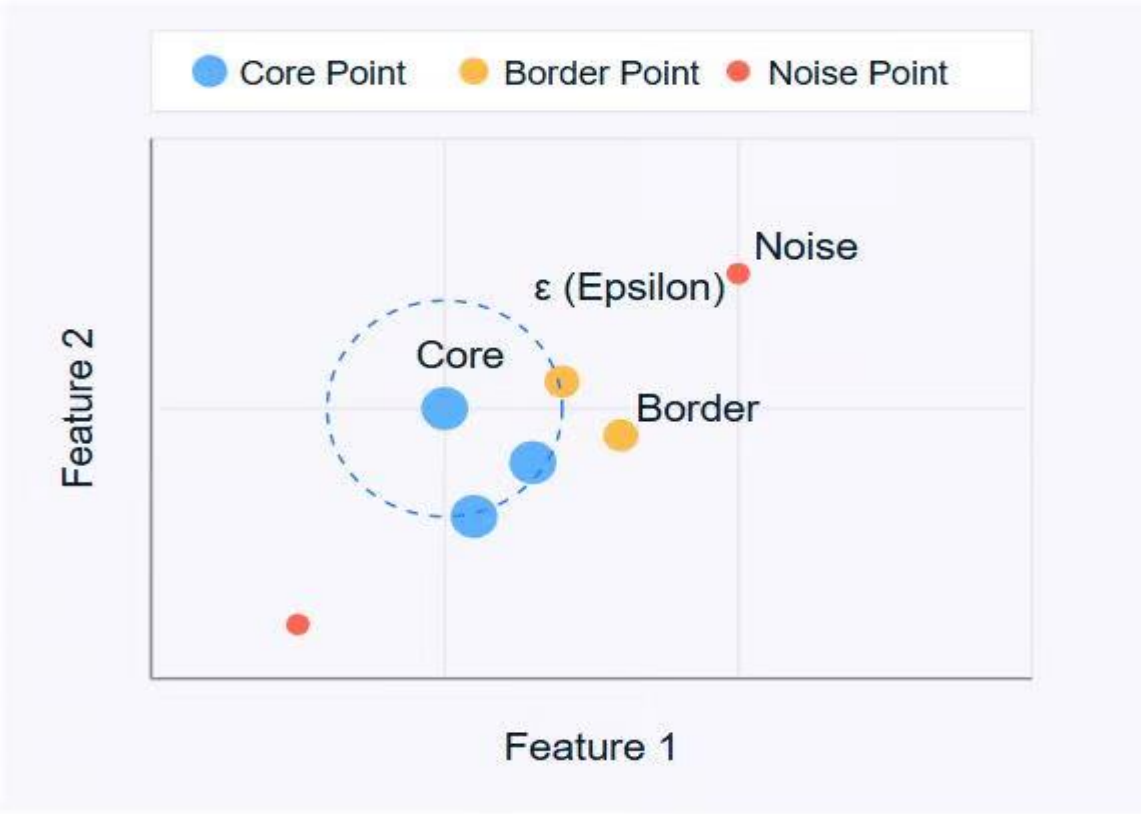
While there are numerous clustering algorithms (you might have heard of **K-means** or **hierarchical clustering**), DBSCAN offers unique advantages. As a density-based method, DBSCAN has several strengths:

1. Flexibility in Cluster Shape
2. No Pre-defined Number of Clusters
3. 
4. Noise Handling
5. Density-Based Insight

**DBSCAN revolves around three key concepts:**

1. **Core Points**: These are points that have at least a minimum number of other points (MinPts) within a specified distance (ε or epsilon).

2. **Border Points**: These are points that are within the ε distance of a core point but don't have MinPts neighbors themselves.
3. **Noise Points**: These are points that are neither core points nor border points. They're not close enough to any cluster to be included.




The diagram above illustrates these concepts. Core points (blue) form the heart of clusters, border points (orange) are on the edge of clusters, and noise points (red) are isolated.

**DBSCAN uses two main parameters:**

- **ε (epsilon)**: The maximum distance between two points for them to be considered as neighbors.
- **MinPts**: The minimum number of points required to form a dense region.

By adjusting these parameters, you can control how the algorithm defines clusters, allowing it to adapt to different types of datasets and clustering requirements.

**How Does DBSCAN Work?**

DBSCAN operates by examining the neighborhood of each point in the dataset. The algorithm follows a step-by-step process to identify clusters based on the density of data points. Let's break down how DBSCAN works:

1. **Parameter Selection**
    - Choose ε (epsilon): The maximum distance between two points for them to be considered as neighbors.
    - Choose MinPts: The minimum number of points required to form a dense region.
2. **Select a Starting Point**
    - The algorithm starts with an arbitrary unvisited point in the dataset.

**3. Examine the Neighborhood**

- It retrieves all points within the ε distance of the starting point.
- If the number of neighboring points is less than MinPts, the point is labeled as noise (for now).
- If there are at least MinPts points within ε distance, the point is marked as a core point, and a new cluster is formed.

**4. Expand the Cluster**

- All the neighbors of the core point are added to the cluster.
- For each of these neighbors:
    - If it's a core point, its neighbors are added to the cluster recursively.
    - If it's not a core point, it's marked as a border point, and the expansion stops.

**4. Repeat the Process**

- The algorithm moves to the next unvisited point in the dataset.
- Steps 3-4 are repeated until all points have been visited.

**5. Finalize Clusters**

- After all points have been processed, the algorithm identifies all clusters.

- Points initially labeled as noise might now be border points if they're within ε distance of a core point.

**6. Handling Noise**

- Any points not belonging to any cluster remain classified as noise.

This process allows DBSCAN to form clusters of arbitrary shapes and identify outliers effectively. The algorithm's ability to find clusters without specifying the number of clusters beforehand is one of its key strengths.

| Feature | DBSCAN | K-Means |
|---|---|---|
| Cluster Shape | Can identify clusters of arbitrary shapes | Assumes clusters are convex and roughly spherical |
| Number of Clusters | Does not require specifying the number of clusters beforehand | Requires specifying the number of clusters (K) in advance |
| Handling Outliers | Identifies outliers as noise points | Forces every point into a cluster, potentially distorting cluster shapes |
| Sensitivity to Parameters | Sensitive to epsilon and MinPts parameters | Sensitive to initial centroid positions and the choice of K |
| Cluster Density | Can find clusters of varying densities | Tends to find clusters of similar spatial extent and density |

**Advantages:**

- No need to specify the number of clusters.
- Works well with non-globular clusters (arbitrary shapes).
- Handles noise and outliers effectively.

**Disadvantages:**

- **Parameter tuning**: Choosing optimal eps and min_samples can be challenging.
- **Performance**: Not efficient for very large datasets with high dimensionality.
- **Uneven densities**: Struggles when clusters have varying densities.

**Use Cases:**

- **Geospatial Data**: Identifying clusters of points on a map.
- **Image Processing**: Segmenting objects in images.
- **Anomaly Detection**: Identifying outliers in datasets.

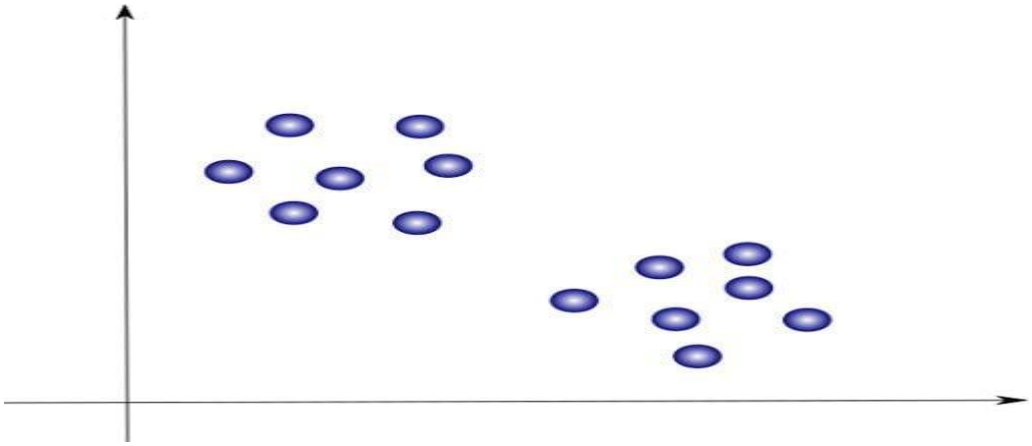
# UNIT-5

## Unsupervised Learning Techniques

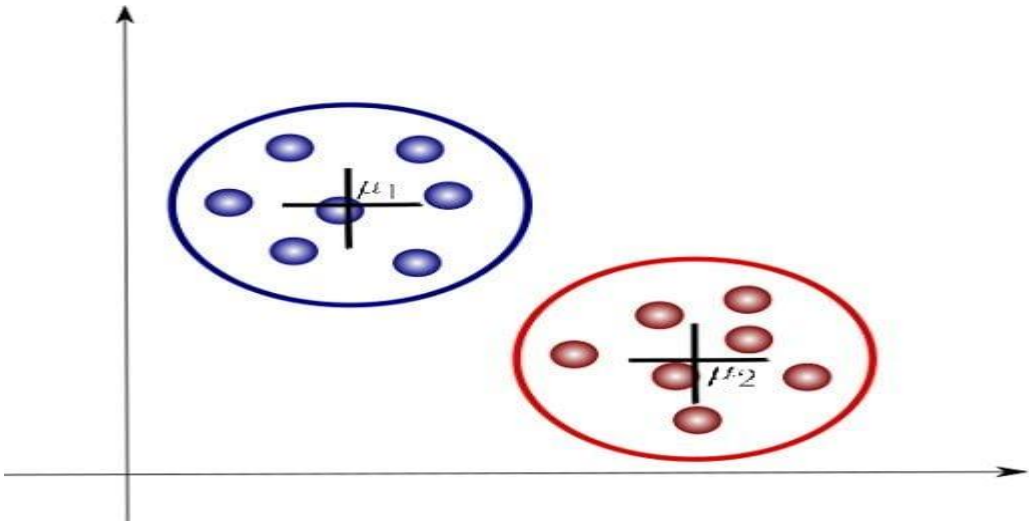
**Gaussian mixture model**

A Gaussian mixture model (GMM) is a machine learning method used to determine the probability each data point belongs to a given cluster. The model is a soft clustering method used in unsupervised learning

Clustering is an unsupervised learning problem where we intend to find clusters of points in our data set that share some common characteristics. Let's suppose we have a data set that looks like this



Our job is to find sets of points that appear close together. In this case, we can clearly identify two clusters of points that we will color blue and red, respectively:

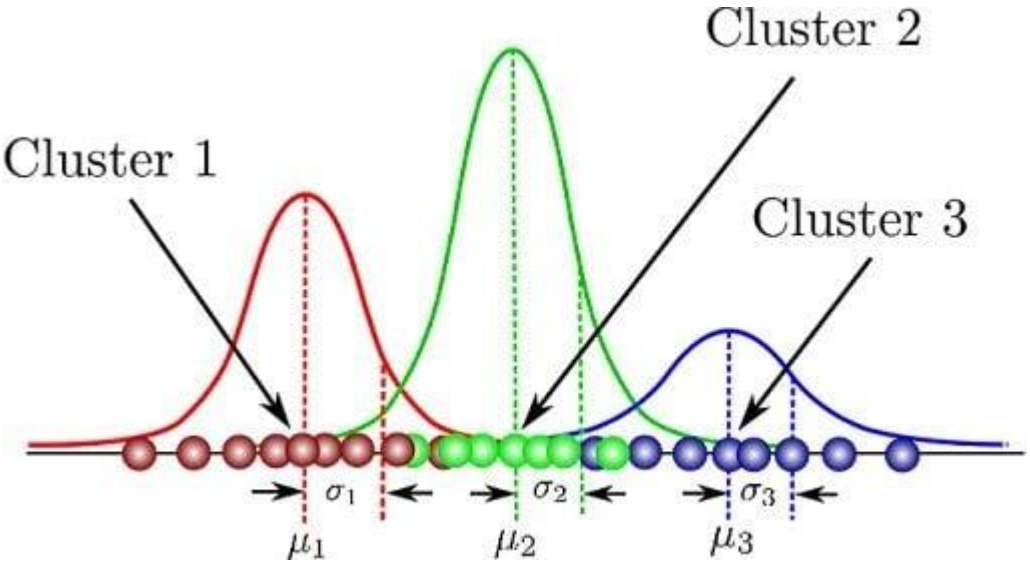



Here, μ1 and μ2 are the centroids of each cluster and are parameters that identify each of these. Let's look at how a Gaussian mixture model can help us analyze these clusters.

**What Is a Gaussian Mixture Model?**

A Gaussian mixture is a function that is composed of several Gaussians, each identified by $k \in \{1,\ldots, K\}$, where $K$ is the number of clusters of our data set. Each Gaussian $k$ in the mixture is comprised of the following parameters:

- A mean μ that defines its center.
- A covariance Σ that defines its width. This would be equivalent to the dimensions of an ellipsoid in a multivariate scenario.
- A mixing probability π that defines how big or small the Gaussian function will be.




Here, we can see that th If we differentiate this equation with respect to the mean and covariance and then equate it to zero, then we will be able to find the optimal values for these parameters, and the solutions will correspond to the maximum likelihood estimation (MLE) for this setting.

ere are three Gaussian functions, hence $K = 3$. Each Gaussian explains the data contained in each of the three clusters available. The mixing coefficients are themselves probabilities and must meet this condition:

$$\sum_{k=1}^{K} \pi_k = 1$$

How do we determine the optimal values for these parameters? To achieve this we must ensure that each Gaussian fits the data points belonging to each cluster. This is exactly what maximum likelihood does.

In general, the Gaussian density function is given by:

$$\mathcal{N}(\mathbf{x}|\mu, \Sigma) = \frac{1}{(2\pi)^{D/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)\right)$$

Where x represents our data points, *D* is the number of dimensions of each data point. μ and Σ are the mean and covariance, respectively. If we have a data set composed of *N* = 1000 three-dimensional points (*D* = 3), then x will be a 1000 × 3 matrix. μ will be a 1 × 3 vector, and Σ will be a 3 × 3 matrix. For later purposes, we will also find it useful to take the log of this equation, which is given by:

$$\ln \mathcal{N}(\mathbf{x}|\mu,\Sigma) = -\frac{D}{2}\ln 2\pi - \frac{1}{2}\ln \Sigma - \frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)$$

**Key Components of GMM**

1. **Components**: Individual Gaussian distributions (subpopulations).
2. **Parameters**:
    1. **Means (**μk) for each Gaussian component.
    2. **Covariances** (Σk) for each component (determines shape and spread).
    3. **Mixing Coefficients** (πk): Probabilities that a data point belongs to a particular component (∑kπk=1).
3. **Number of Components** (K): The number of Gaussian distributions assumed in the model.

**Training Gaussian Mixture Models**

GMMs are typically trained using the **Expectation-Maximization (EM) algorithm**, which iteratively estimates the parameters until convergence.

1. **Initialization**: Start with initial guesses for μk, Σk, and πk.
2. **E-Step (Expectation)**: Calculate the responsibility γik, the probability that data point xi belongs to the k-th Gaussian component:

$$\gamma_{ik} = \frac{\pi_k \mathcal{N}(x_i \mid \mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_i \mid \mu_j, \Sigma_j)}$$

**3. M-Step (Maximization)**: Update the parameters μk, Σk, and πk based on the responsibilities

**4. Repeat** until the parameters converge.

**Advantages :**

- Flexible: Can model complex distributions with multiple peaks.
- Probabilistic: Provides soft assignments (probabilities) for cluster membership.

**Disadvantages :**

    - Sensitive to initialization.
    - Assumes data is generated from Gaussian distributions.
- May overfit if too many components are used.

**Applications of GMM**

1. **Clustering**: Grouping data into clusters where each cluster follows a Gaussian distribution.
2. **Anomaly Detection**: Identifying data points that do not fit any of the Gaussian components.
3. **Image Segmentation**: Separating regions of an image based on pixel intensity distributions.
4. **Density Estimation**: Estimating the underlying probability distribution of the data.
5. **Speech Recognition**: Modeling speech features for classification.

# UNIT-5

## Unsupervised Learning Techniques

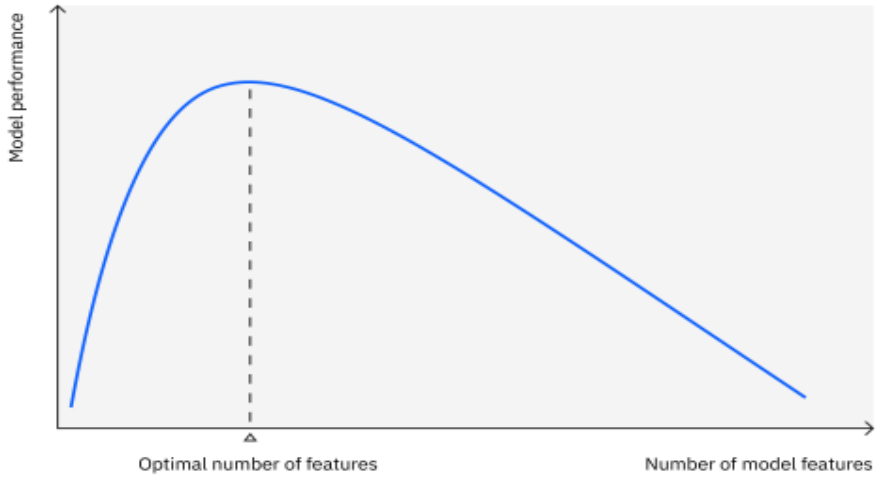
**The Curse of Dimensionality- Main Approaches for Dimensionality Reduction**

The number of input features, variables, or columns present in a given dataset is known as dimensionality, and the process to reduce these features is called dimensionality reduction.

A dataset contains a huge number of input features in various cases, which makes the predictive modeling task more complicated. Because it is very difficult to visualize or make predictions for the training dataset with a high number of features, for such cases, dimensionality reduction techniques are required to use.

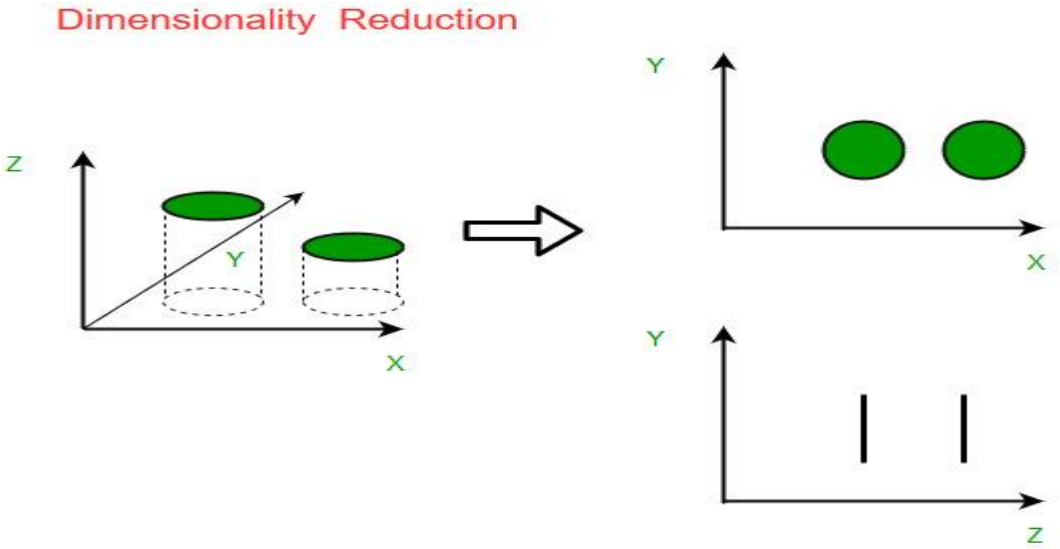
**What is Dimensionality Reduction?**

Dimensionality reduction is a technique used to reduce the number of features in a dataset while retaining as much of the important information as possible. In other words, it is a process of transforming high-dimensional data into a lower-dimensional space that still preserves the essence of the original data.




- In machine learning, high-dimensional data refers to data with a large number of features or variables. The curse of dimensionality is a common problem in machine learning, where the performance of the model deteriorates as the number of features increases. This is because the complexity of the model increases with the number of features, and it becomes more

difficult to find a good solution. In addition, high-dimensional data can also lead to overfitting, where the model fits the training data too closely and does not generalize well to new data.




**Main Approaches for Dimensionality Reduction**

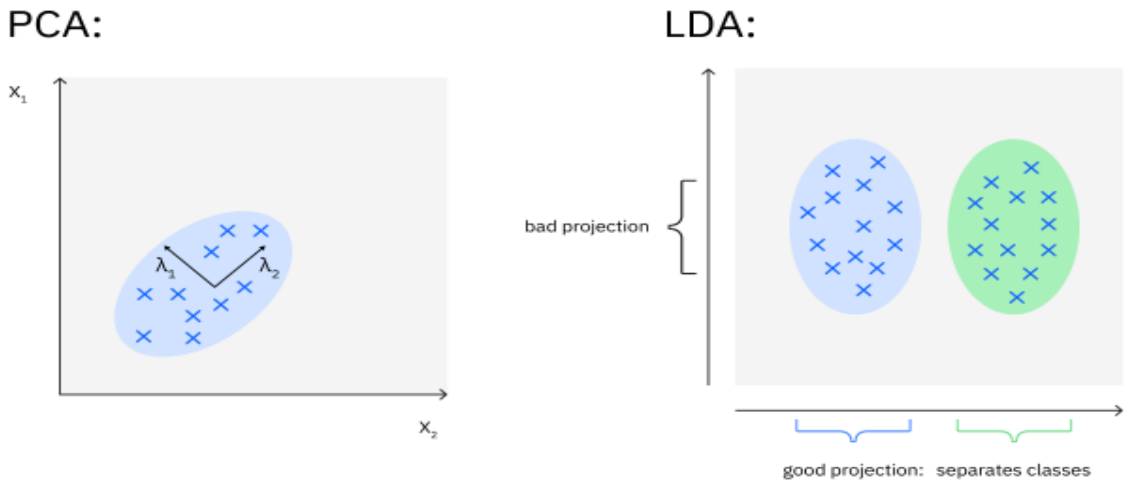
1. Principal Component Analysis (PCA)
2. Linear Discriminant Analysis (LDA)
3. t-Distributed Stochastic Neighbor Embedding (t-SNE)
4. Uniform Manifold Approximation and Projection (UMAP)
5. Autoencoders
6. Feature Selection Techniques

1. **Principal Component Analysis (PCA)**

- **Type**: Linear technique
- **Purpose**: Projects data onto a set of new axes (called principal components) that capture the maximum variance in the data.
- **How it works**:
    - Computes eigenvectors and eigenvalues from the covariance matrix of the data.
    - Principal components are the directions where the data has the most variance.
    - Retain the top k components based on the explained variance.

**2. Linear Discriminant Analysis (LDA)**

- **Type**: Linear technique (supervised)
- **Purpose**: Reduces dimensions while preserving the class separability (good for classification tasks).
- **How it works**:
    - Maximizes the ratio of between-class variance to within-class variance.
    - Projects data onto a new space that best discriminates between classes.




**3. t-Distributed Stochastic Neighbor Embedding (t-SNE)**

- **Type**: Non-linear technique
- **Purpose**: Visualizes high-dimensional data by mapping it to 2D or 3D while preserving local relationships.
- **How it works**:
    - Uses probability distributions to measure similarities between points in high-dimensional and low-dimensional spaces.
    - Minimizes the difference between these distributions.

**4. Uniform Manifold Approximation and Projection (UMAP)**

- **Type**: Non-linear technique
- **Purpose**: Similar to t-SNE but faster and better at preserving both local and global structures.
- **How it works**:

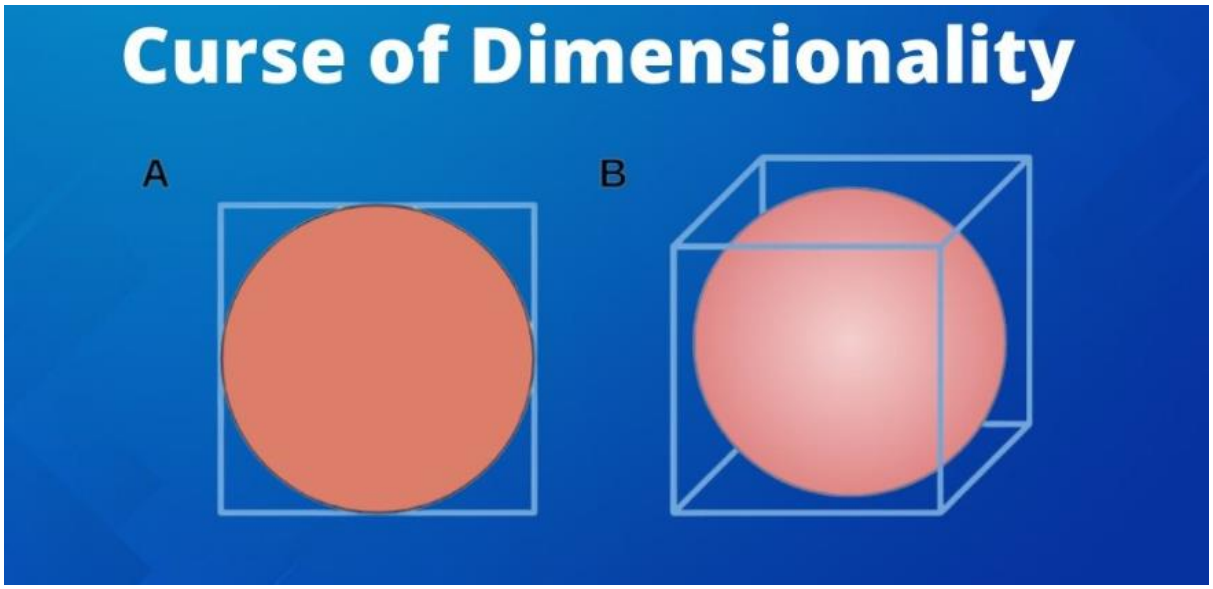
- Builds a graph representing the data's high-dimensional structure.
- Optimizes a low-dimensional embedding based on this graph.

**5. Autoencoders**

- **Type**: Non-linear technique (neural network-based)
- **Purpose**: Learns a compressed, lower-dimensional representation of the input data.
- **How it works**:
  - Consists of an **encoder** (compresses data) and a **decoder** (reconstructs data).
  - Trained to minimize reconstruction error.

**Choosing the Right Approach**

- **For Visualization**: t-SNE, UMAP
- **For Classification**: LDA, PCA (if unsupervised)
- **For Large Datasets**: UMAP, Autoencoders, Feature Selection
- **For Preserving Original Features**: Feature Selection




**Curse of Dimensionality in Machine Learning** arises when working with high-dimensional data, leading to increased computational complexity, overfitting, and spurious correlations.

Techniques like dimensionality reduction, feature selection, and careful model design are essential for mitigating its effects and improving algorithm performance. Navigating this challenge is crucial for unlocking the potential of high-dimensional datasets and ensuring robust machine-learning solutions.

**Curse of Dimensionality** refers to the phenomenon where the efficiency and effectiveness of algorithms deteriorate as the dimensionality of the data increases exponentially

**How to Overcome the Curse of Dimensionality?**

To overcome the curse of dimensionality, you can consider the following strategies:

**1. Dimensionality Reduction Techniques:**

- **Feature Selection:** Identify and select the most relevant features from the original dataset while discarding irrelevant or redundant ones. This reduces the dimensionality of the data, simplifying the model and improving its efficiency.
- **Feature Extraction:** Transform the original high-dimensional data into a lower-dimensional space by creating new features that capture the essential information. Techniques such as Principal Component Analysis (PCA) and t-distributed Stochastic Neighbor Embedding (t-SNE) are commonly used for feature extraction.

**2. Data Preprocessing:**

- **Normalization:** Scale the features to a similar range to prevent certain features from dominating others, especially in distance-based algorithms.
- **Handling Missing Values:** Address missing data appropriately through imputation or deletion to ensure robustness in the model training process.

# UNIT-5

## Unsupervised Learning Techniques

### Principal component Analysis

Principal component analysis (PCA) is a dimensionality reduction and machine learning method used to simplify a large data set into a smaller set while still maintaining significant patterns and trends.

Reducing the number of variables of a data set naturally comes at the expense of accuracy, but the trick in dimensionality reduction is to trade a little accuracy for simplicity. Because smaller data sets are easier to explore and visualize, and thus make analyzing data points much easier and faster for machine learning algorithms without extraneous variables to process.

So, to sum up, the idea of PCA is simple: **reduce the number of variables of a data set.**

### How Do You Do a Principal Component Analysis?

1. Standardize the range of continuous initial variables
2. Compute the covariance matrix to identify correlations
3. Compute the eigenvectors and eigenvalues of the covariance matrix to identify the principal components
4. Create a feature vector to decide which principal components to keep
5. Recast the data along the principal components axes

### Step 1: Standardization

The aim of this step is to standardize the range of the continuous initial variables so that each one of them contributes equally to the analysis.

That is, if there are large differences between the ranges of initial variables, those variables with larger ranges will dominate over those with small ranges (for example, a variable that ranges between 0 and 100 will dominate over a variable that ranges between 0 and 1), which will lead to biased results. So, transforming the data to comparable scales can prevent this problem.

Mathematically, this can be done by subtracting the mean and dividing by the standard deviation for each value of each variable.

$$z = \frac{value - mean}{standard\ deviation}$$

Once the standardization is done, all the variables will be transformed to the same scale.

**Step 2: Covariance Matrix Computation**

The aim of this step is to understand how the variables of the input data set are varying from the mean with respect to each other, or in other words, to see if there is any relationship between them.

The covariance matrix is a $p \times p$ symmetric matrix (where $p$ is the number of dimensions) that has as entries the covariances associated with all possible pairs of the initial variables. For example, for a 3-dimensional data set with 3 variables $x$, $y$, and $z$, the covariance matrix is a 3×3 data matrix of this from:

$$\begin{bmatrix} Cov(x,x) & Cov(x,y) & Cov(x,z) \\ Cov(y,x) & Cov(y,y) & Cov(y,z) \\ Cov(z,x) & Cov(z,y) & Cov(z,z) \end{bmatrix}$$

Since the covariance of a variable with itself is its variance (Cov(a,a)=Var(a)), in the main diagonal (Top left to bottom right) we actually have the variances of each initial variable. And since the covariance is commutative (Cov(a,b)=Cov(b,a)), the entries of the covariance matrix are symmetric with respect to the main diagonal, which means that the upper and the lower triangular portions are equal.

**What do the covariances that we have as entries of the matrix tell us about the correlations between the variables?**

It's actually the sign of the covariance that matters:

- If positive then: the two variables increase or decrease together (correlated)
- If negative then: one increases when the other decreases (Inversely correlated)

Now that we know that the covariance matrix is not more than a table that summarizes the correlations between all the possible pairs of variables, let's move to the next step.

**Step 3: Compute the eigenvectors and eigenvalues of the covariance matrix to identify the principal components**

Eigenvectors and eigenvalues are the linear algebra concepts that we need to compute from the covariance matrix in order to determine the ***principal components*** of the data.

What you first need to know about eigenvectors and eigenvalues is that they always come in pairs, so that every eigenvector has an eigenvalue. Also, their number is equal to the number of dimensions of the data. For example, for a 3-dimensional data set, there are 3 variables, therefore there are 3 eigenvectors with 3 corresponding eigenvalues.

It is eigenvectors and eigenvalues who are behind all the magic of principal components because the eigenvectors of the Covariance matrix are actually *the directions of the axes where*

*there is the most variance* (most information) and that we call Principal Components. And eigenvalues are simply the coefficients attached to eigenvectors, which give the *amount of variance carried in each Principal Component*.

**Principal Component Analysis Example:**

Let's suppose that our data set is 2-dimensional with 2 variables ***x,y*** and that the eigenvectors and eigenvalues of the covariance matrix are as follows:

$$v1 = \begin{bmatrix} 0.6778736 \\ 0.7351785 \end{bmatrix} \qquad \lambda_1 = 1.284028$$

$$v2 = \begin{bmatrix} -0.7351785 \\ 0.6778736 \end{bmatrix} \qquad \lambda_2 = 0.04908323$$

If we rank the eigenvalues in descending order, we get λ1>λ2, which means that the eigenvector that corresponds to the first principal component (PC1) is *v1* and the one that corresponds to the second principal component (PC2) is *v2*.

After having the principal components, to compute the percentage of variance (information) accounted for by each component, we divide the eigenvalue of each component by the sum of eigenvalues. If we apply this on the example above, we find that PC1 and PC2 carry respectively 96 percent and 4 percent of the variance of the data.

**Step 4: Create a Feature Vector**

As we saw in the previous step, computing the eigenvectors and ordering them by their eigenvalues in descending order, allow us to find the principal components in order of significance. In this step, what we do is, to choose whether to keep all these components or discard those of lesser significance (of low eigenvalues), and form with the remaining ones a matrix of vectors that we call *Feature vector*.

So, the feature vector is simply a matrix that has as columns the eigenvectors of the components that we decide to keep. This makes it the first step towards dimensionality reduction, because if we choose to keep only ***p*** eigenvectors (components) out of ***n***, the final data set will have only ***p*** dimensions.

**Principal Component Analysis Example**:

Continuing with the example from the previous step, we can either form a feature vector with both of the eigenvectors *v*1 and *v*2:

$$\begin{bmatrix} 0.6778736 & -0.7351785 \\ 0.7351785 & 0.6778736 \end{bmatrix}$$

Or discard the eigenvector *v*2, which is the one of lesser significance, and form a feature vector with *v*1 only:

$$\begin{bmatrix} 0.6778736 \\ 0.7351785 \end{bmatrix}$$

Discarding the eigenvector *v2* will reduce dimensionality by 1, and will consequently cause a loss of information in the final data set. But given that *v*2 was carrying only 4 percent of the information, the loss will be therefore not important and we will still have 96 percent of the information that is carried by *v*1.

**Step 5: Recast the Data Along the Principal Components Axes**

In the previous steps, apart from standardization, you do not make any changes on the data, you just select the principal components and form the feature vector, but the input data set remains always in terms of the original axes (i.e, in terms of the initial variables).

In this step, which is the last one, the aim is to use the feature vector formed using the eigenvectors of the covariance matrix, to reorient the data from the original axes to the ones represented by the principal components (hence the name Principal Components Analysis). This can be done by multiplying the transpose of the original data set by the transpose of the feature vector.

$$FinalDataSet = FeatureVector^T * StandardizedOriginalDataSet^T$$

1. Calculate the $x_{\text{mean}}$ of the given feature vector $\{x_1, x_2, x_3, \cdots, x_n\}$ as

$x_{\text{mean}} = \frac{1}{n}\sum_{i=1}^{n} x_i$

2. Compute covariance matrix Cov as

$\text{Cov} = \frac{1}{n}\sum_{i=1}^{n} \bar{x}_i . \bar{x}_i^T$

where $\bar{x}_i = x_i - x_{\text{mean}}$

3. Lastly, calculate the eigenvalues and eigenvectors of the corresponding Cov

$\text{Cov} \times e_k = \lambda_k \times e_k for k = 0, 1, \cdots n - 1$

Where $e_k = k^{\text{th}} eigenvector, \lambda_k = k^{\text{th}} eigenvalue$.

4. In order to minimize the dimensions of feature vectors, eigenvectors corresponding to highest eigenvalues are chosen (Shlens, 2014).

$$M_{fv} = \left(e_1, e_2, e_3, \cdots, e_p\right)$$

where $p = number of dimensions of PCA matrix$.

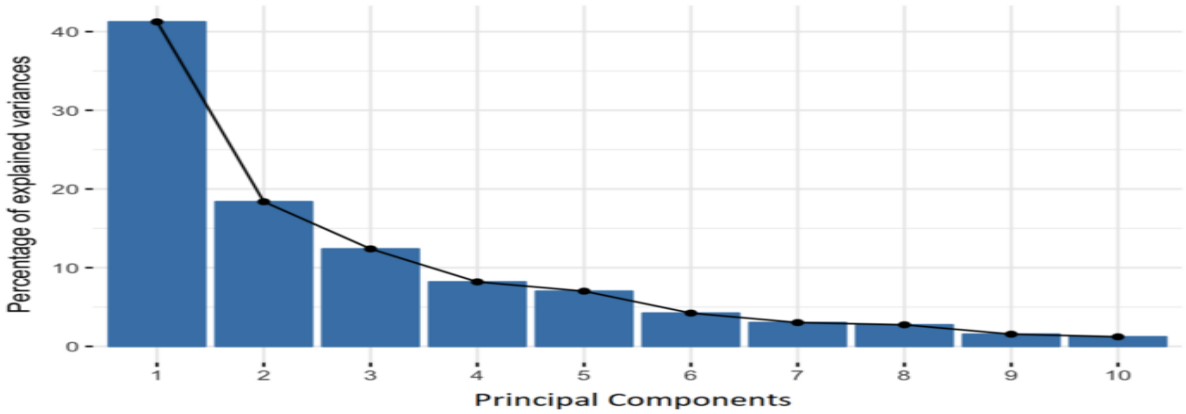
5. The new feature vector is computed as

$$N_{fv} = M'_{fv} \times T$$

where $T = transpose of the mean adjusted original input$.

**What Are Principal Components?**

Principal components are new variables that are constructed as linear combinations or mixtures of the initial variables. These combinations are done in such a way that the new variables (i.e., principal components) are uncorrelated and most of the information within the initial variables is squeezed or compressed into the first components. So, the idea is 10-dimensional data gives you 10 principal components, but PCA tries to put maximum possible information in the first component, then maximum remaining information in the second and so on, until having something like shown in the scree plot below.

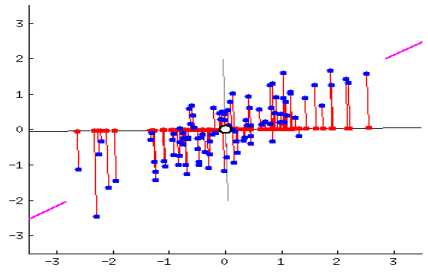



Geometrically speaking, principal components represent the directions of the data that explain a **maximal amount of variance**, that is to say, the lines that capture most information of the data. The relationship between variance and information here, is that, the larger the variance carried by a line, the larger the dispersion of the data points along it, and the larger the dispersion along a line, the more information it has. To put all this simply, just think of principal components as new axes that provide the best angle to see and evaluate the data, so that the differences between the observations are better visible.

**How PCA Constructs the Principal Components**

As there are as many principal components as there are variables in the data, principal components are constructed in such a manner that the first principal component accounts for the **largest possible variance** in the data set.

For example, let's assume that the scatter plot of our data set is as shown below, can we guess the first principal component ? Yes, it's approximately the line that matches the purple marks because it goes through the origin and it's the line in which the projection of the points (red dots) is the most spread out. Or mathematically speaking, it's the line that maximizes the variance (the average of the squared distances from the projected points (red dots) to the origin).



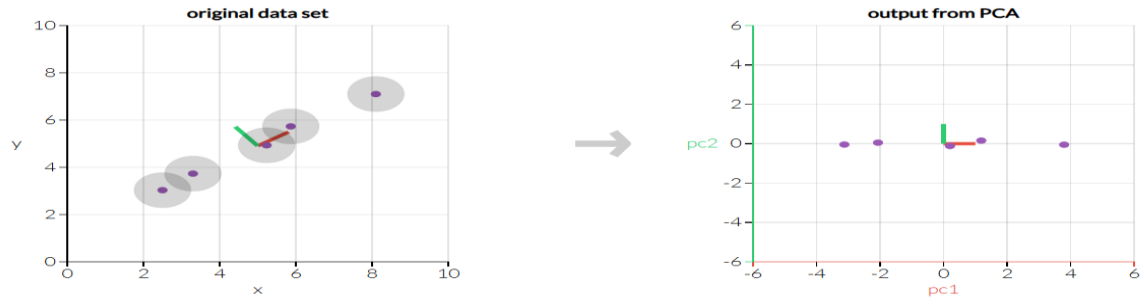
The second principal component is calculated in the same way, with the condition that it is uncorrelated with (i.e., perpendicular to) the first principal component and that it accounts for the next highest variance.

This continues until a total of $p$ principal components have been calculated, equal to the original number of variables.

**2D example**

First, consider a dataset in only two dimensions, like (height, weight). This dataset can be plotted as points in a plane. But if we want to tease out variation, PCA finds a new coordinate system in which every point has a new (x,y) value. The axes don't actually mean anything physical; they're combinations of height and weight called "principal components" that are chosen to give one axes lots of variation.

Drag the points around in the following visualization to see PC coordinate system adjusts.




**3D example**

With three dimensions, PCA is more useful, because it's hard to see through a cloud of data. In the example below, the original data are plotted in 3D, but you can project the data into 2D through a transformation no different than finding a camera angle: rotate the axes to find the

best angle. To see the "official" PCA transformation, click the "Show PCA" button. The PCA transformation ensures that the horizontal axis PC1 has the most variation, the vertical axis PC2 the second-most, and a third axis PC3 the least. Obviously, PC3 is the one we drop