KALINGA UNIVERSITY NAYA RAIPUR

Faculty Of CS & IT

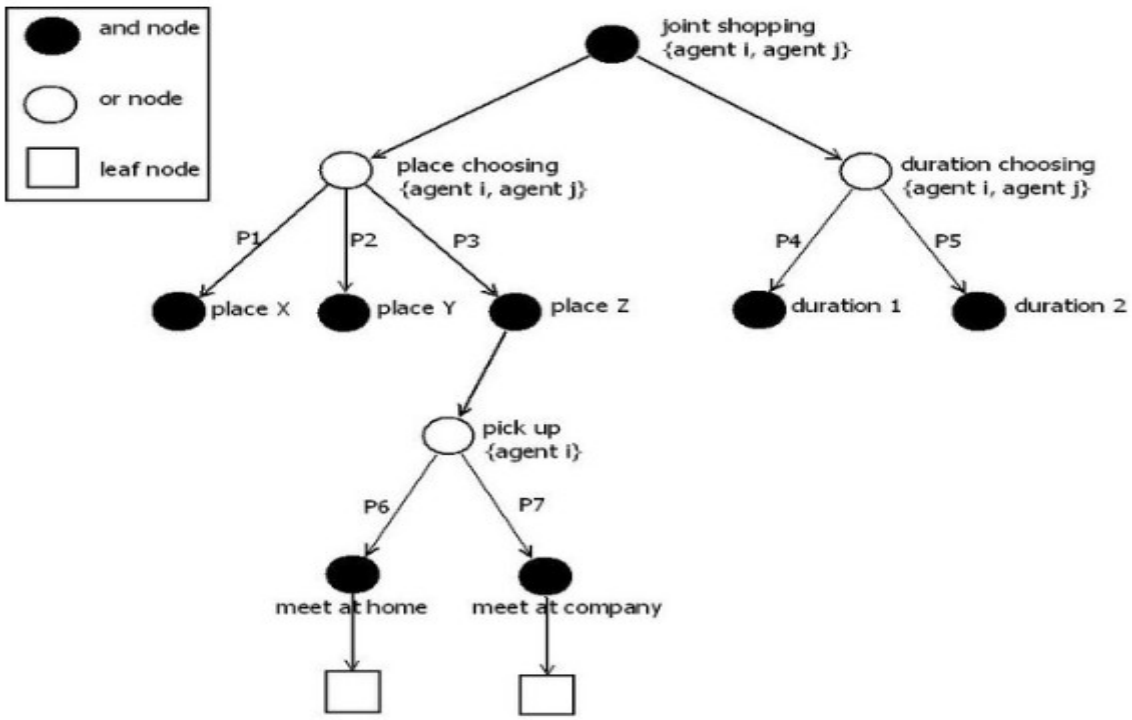
Programme: -BCAAIML
Course Name: - Machine Learning Techniques
Course Code: -BCAAIML501 **Sem : 5th**

Unit-III

Tree and Probabilistic Models

Tree and Probabilistic Models are essential tools in machine learning used for both classification and regression tasks. **Tree-based models**, such as **Decision Trees**, work by recursively splitting data into subsets based on feature values, creating a tree-like structure where each internal node represents a decision rule, and each leaf node represents an output. Advanced tree models like **Classification and Regression Trees (CART)** improve predictive accuracy.

Ensemble methods such as **Bagging** and **Boosting** combine multiple trees to reduce variance and bias, leading to better generalization. On the other hand, **probabilistic models** rely on statistical principles to make predictions by estimating the likelihood of outcomes. They use **basic statistics** and techniques like **Gaussian Mixture Models (GMMs)** to model data distributions, and are fundamental in understanding uncertainty and learning from data. Probabilistic models are particularly powerful when combined with tree-based techniques for hybrid learning systems.



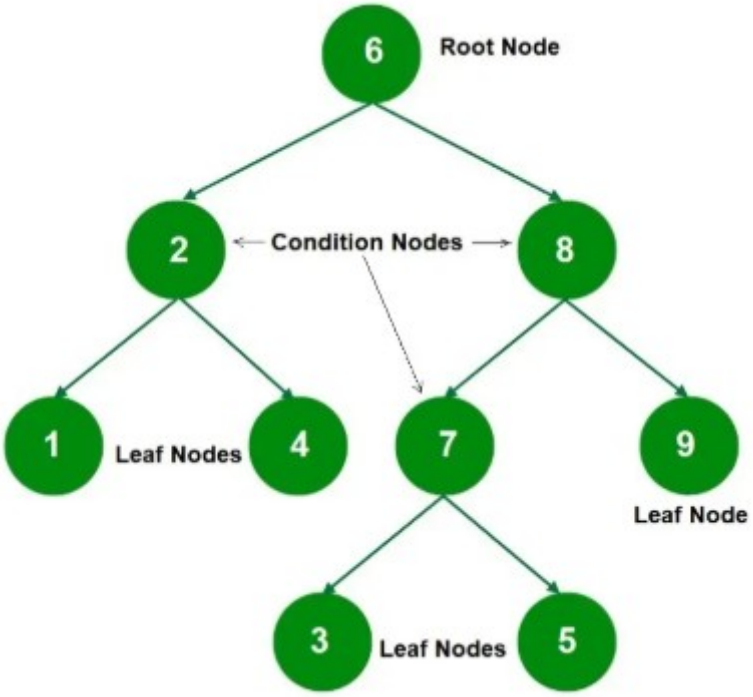
Learning with Trees

Learning with Trees refers to the use of decision tree-based models to learn patterns and relationships in data for tasks like classification and regression.

A **decision tree** breaks down a dataset into smaller and smaller subsets while at the same time an associated tree structure is incrementally developed. Each internal node of the tree represents a test on a feature, each branch represents the outcome of the test, and each leaf node represents a class label or a predicted value.

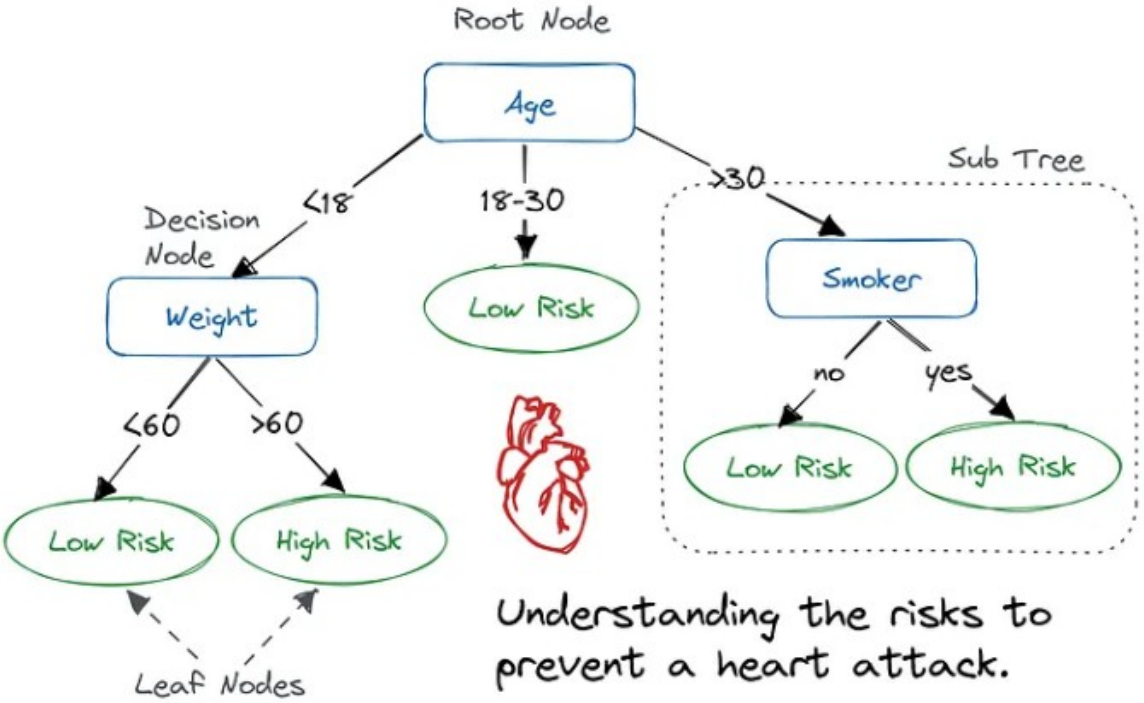
The tree is constructed using algorithms like ID3, C4.5, or CART, which select features based on criteria such as **information gain** or **Gini index**. Tree models are simple to understand, interpret, and visualize, and they do not require much data preprocessing.

They are particularly useful when the relationship between features and output is non-linear or when data is noisy. However, they are prone to overfitting, which can be mitigated using pruning or ensemble methods like random forests and boosting.



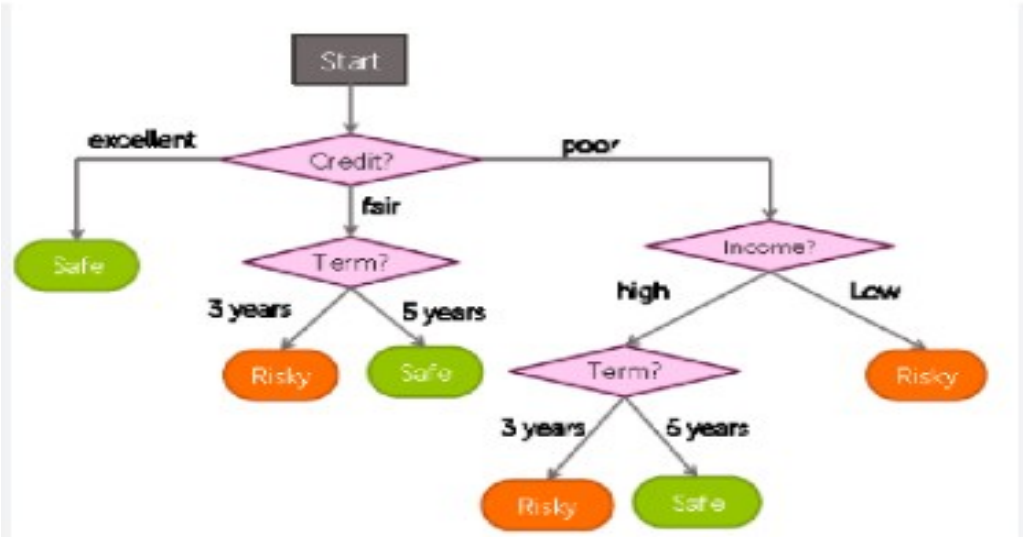
Decision Trees

Decision Trees are a popular supervised learning method used for classification and regression tasks. They model decisions using a tree-like structure where each internal node represents a test on a feature, each branch corresponds to an outcome of the test, and each leaf node holds a predicted output.



The tree is built by recursively splitting the data based on features that provide the best separation of classes or values, using measures like **Information Gain**, **Gini Index**, or **Gain Ratio**. Decision trees are easy to understand, interpret, and visualize. However, they can easily overfit the training data if not properly pruned or regularized. Despite this, they serve as a foundation for more advanced ensemble methods like **Random Forests** and **Boosted Trees**, making them a core component in many machine learning systems.

Constructing Decision Trees



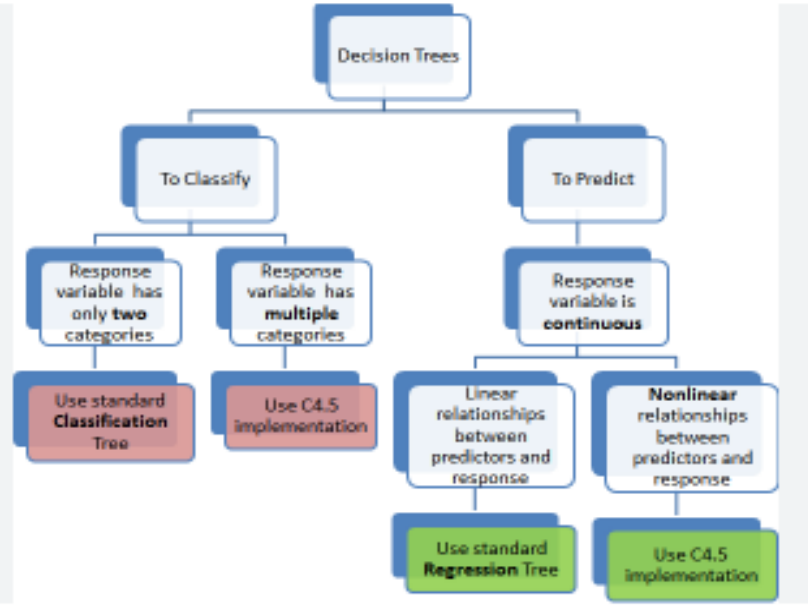
Constructing Decision Trees involves building a tree-based model by selecting features that best split the data at each node.

The construction process begins at the root node and proceeds by evaluating all possible features and selecting the one that offers the highest **Information Gain**, **Gain Ratio**, or lowest **Gini Index**, depending on the chosen algorithm (e.g., ID3, C4.5, CART). Once the best feature is selected, the dataset is divided based on its possible values, creating branches.

This process is repeated recursively for each child node until a stopping criterion is met — such as all samples in a node belonging to the same class, reaching a maximum tree depth, or the number of samples falling below a threshold.

To avoid **overfitting**, **pruning techniques** can be applied after the full tree is grown, by removing branches that have little contribution to predictive accuracy. The result is a model that is both interpretable and effective in handling complex datasets

Classification and Regression Trees.



CART is a decision tree algorithm used for both classification (categorical output) and regression (numerical output). It builds binary trees using the feature that best splits the data into two distinct groups.

1. Purpose:

- **Classification Trees:** Used when the target variable is categorical.
- **Regression Trees:** Used when the target variable is continuous.

2. Splitting Criterion:

- **Classification:** Uses **Gini Index** to measure the impurity of a node.

- **Regression:** Uses **Mean Squared Error (MSE)** or **Mean Absolute Error (MAE)** to determine splits.

3. Binary Splits Only:

- CART creates **binary trees**, meaning each internal node has exactly two branches, unlike some other algorithms like ID3 or C4.5 which may have multi-way splits.

4. Tree Construction Process:

- Start from the root node with all training data.
- Evaluate all possible splits and choose the one that results in the best separation.
- Recursively split the data at each child node.
- Continue until a stopping condition is met (e.g., minimum node size, maximum depth).

5. Pruning:

- **Post-pruning** is applied to reduce overfitting by removing branches that do not provide significant predictive power.
- CART uses **cost-complexity pruning** to balance tree depth and accuracy.

6. Advantages:

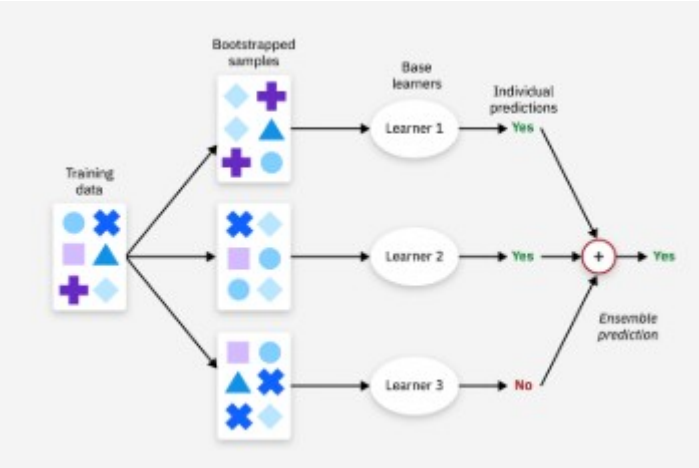
- Handles both classification and regression tasks.
- Easy to understand and interpret.
- Can handle both numerical and categorical data.

7. Limitations:

- Can easily overfit if the tree is too deep.
- Sensitive to small variations in data.

Ensemble Learning –

Ensemble Learning is a machine learning technique that combines multiple individual models, often called **base learners**, to create a more powerful and accurate predictive model.

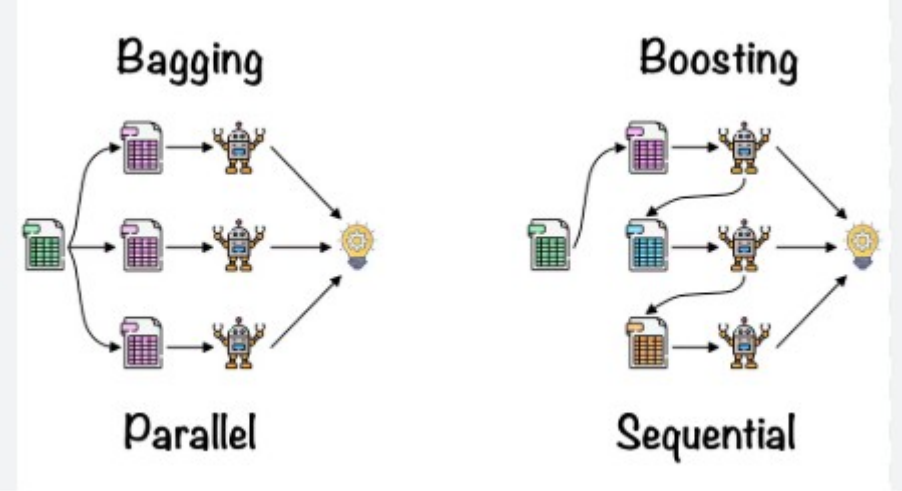


The idea is that by aggregating the predictions of several models, the overall performance improves in terms of accuracy, stability, and generalization.

Boosting

Boosting builds models **sequentially**, where each new model tries to correct the errors made by the previous ones. Instead of equal weights, boosting gives more importance to data points that were misclassified. This method reduces **bias** and can significantly improve accuracy.

Examples include **AdaBoost**, **Gradient Boosting**, and **XGBoost**. However, boosting can be more prone to **overfitting** and is generally more sensitive to noisy data.



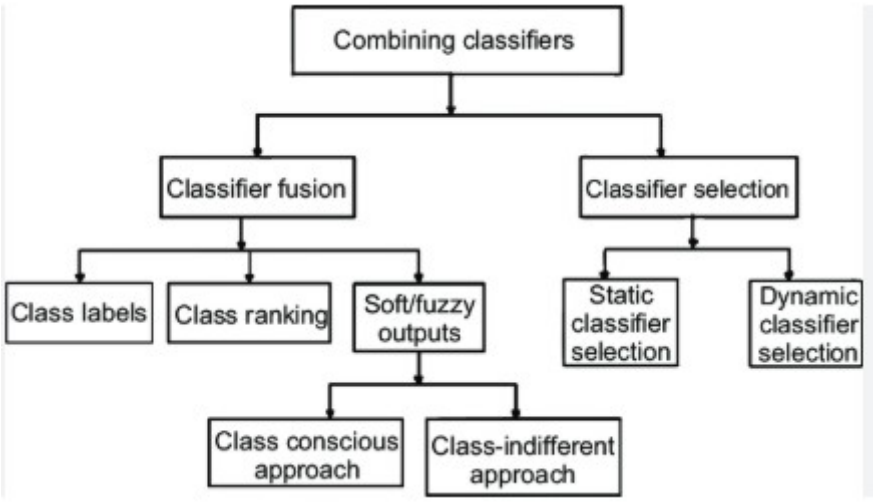
Bagging

Bagging works by training multiple models independently on different random subsets of the training data created through bootstrapping (sampling with replacement).

The final prediction is made by **majority voting** (for classification) or **averaging** (for regression). This technique reduces **variance** and helps prevent overfitting.

A well-known example of bagging is the **Random Forest** algorithm, which builds multiple decision trees and combines their results.

Different ways to Combine Classifiers



Combining classifiers is a key technique in **ensemble learning** to improve prediction performance by leveraging the strengths of multiple models. The goal is to make more accurate decisions than any single classifier alone. Common methods include:

1. **Majority Voting:** Each classifier casts a vote, and the class with the most votes is selected. It is simple and effective in classification tasks.
2. **Averaging (for regression):** The predictions from multiple models are averaged to give the final output, reducing variance.
3. **Weighted Voting/Averaging:** Each classifier is assigned a weight based on its accuracy or confidence, and the final prediction is made by weighted sum.
4. **Stacking (Stacked Generalization):** Involves training a meta-classifier that learns how to best combine the predictions of several base classifiers.
5. **Boosting and Bagging:** These methods implicitly combine classifiers—boosting focuses on correcting previous errors, while bagging averages over random subsets of data.
6. **Bayesian Model Averaging:** Uses probabilities to weigh models based on their likelihood given the data.
7. **Blending:** Similar to stacking but uses a hold-out validation set instead of cross-validation to train the meta-model.

These techniques help in **reducing overfitting, improving accuracy**, and making models more **robust** and **generalizable**.

Probability and Learnings

Probability and Learning are closely linked in machine learning, as probability provides a mathematical framework for modeling uncertainty, reasoning under incomplete information, and making predictions.

In learning systems, probability helps estimate how likely a certain outcome is given the data. It allows models to not only make decisions but also quantify **confidence levels** in those decisions. Probabilistic models, such as **Bayesian classifiers**, **Hidden Markov Models (HMMs)**, and **Gaussian Mixture Models (GMMs)**, use probability distributions to represent data and guide learning.

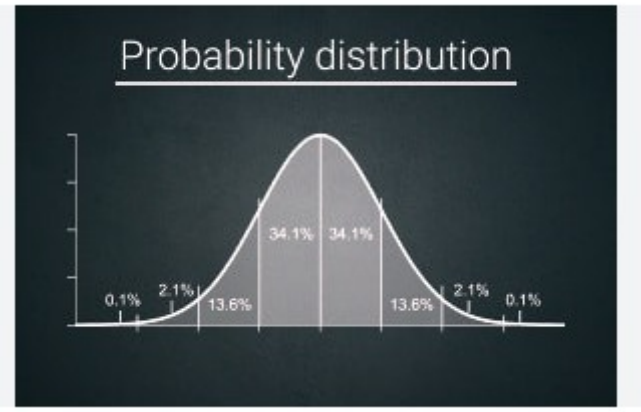
These models are particularly powerful when dealing with noisy, uncertain, or incomplete data. By using probability, learning algorithms can update beliefs about data patterns as more information becomes available, which forms the core of **Bayesian learning**.

Thus, probability enhances the flexibility, adaptability, and reliability of learning systems in complex real-world applications.

Data into Probabilities

Converting Data into Probabilities is a key step in probabilistic machine learning, where raw data is transformed into probability distributions to model uncertainty and make predictions.

This process involves estimating how likely certain outcomes are based on observed data. Techniques such as **frequency-based estimation**, **maximum likelihood estimation (MLE)**, and **Bayesian inference** are commonly used.



For example, in classification tasks, the frequency of each class in the training data can be used to compute the **prior probability** of that class. For more complex models, **conditional probabilities** are calculated to assess the likelihood of an outcome given certain features (e.g., P(class | features)).

Probabilistic models like **Naive Bayes**, **Gaussian Mixture Models**, and **logistic regression** rely on these estimates to make decisions. By converting data into probabilities, models can reason more effectively under uncertainty and provide more informative and interpretable results.

Basic Statistics

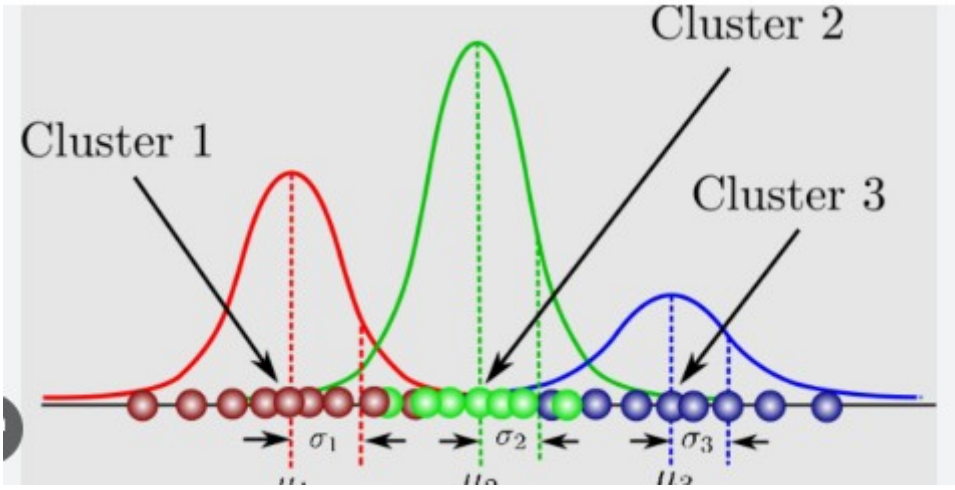
Basic statistics provide foundational tools for analyzing and interpreting data, crucial in all areas of machine learning.

Key Concepts:

1. **Mean (Average):** The sum of all values divided by the number of values. It represents the central tendency.
2. **Median:** The middle value when data is sorted. Less sensitive to outliers.
3. **Mode:** The most frequently occurring value in a dataset.
4. **Variance:** Measures how much data values vary from the mean.
5. **Standard Deviation:** The square root of variance; shows how spread out the data is.
6. **Probability Distributions:** Describe how values are distributed (e.g., Normal distribution, Binomial distribution).
7. **Covariance:** Indicates how two variables vary together.
8. **Correlation:** Measures the strength and direction of the relationship between two variables.
9. **Skewness and Kurtosis:** Measure the asymmetry and peakness of the data distribution..

Gaussian Mixture Models

A **Gaussian Mixture Model** is a **probabilistic model** that assumes data is generated from a mixture of several **Gaussian (normal) distributions**, each representing a cluster.



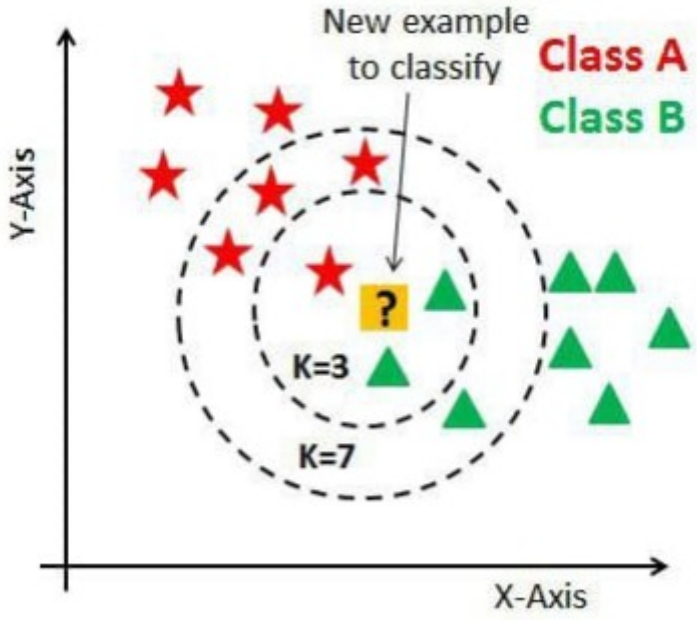
Key Points:

1. **Components:** GMM consists of multiple Gaussian distributions, each with its own mean and variance.
2. **Soft Clustering:** Unlike K-means, which uses hard assignments, GMM provides **probabilities** of belonging to each cluster.
3. **Parameters:** Each Gaussian is defined by:
    - Mean (μ): Center of the distribution
    - Covariance (Σ): Shape/spread of the distribution

- Weight (π): Mixing proportion, representing how much each component contributes

4. **Expectation-Maximization (EM) Algorithm:** GMMs are typically trained using the EM algorithm:
   - **E-step:** Estimate the probability that each point belongs to each Gaussian.
   - **M-step:** Update the parameters (means, covariances, and weights) to maximize the likelihood.
5. **Advantages:**
   - Can model complex, overlapping data
   - More flexible than K-means (which assumes spherical clusters)
6. **Applications:**
   - Clustering
   - Anomaly detection
   - Image segmentation
   - Speech and handwriting recognition
7. **Limitations:**
   - Can be sensitive to initialization
   - Computationally more intensive
   - Assumes data is normally distributed within each cluster

Nearest Neighbor Methods



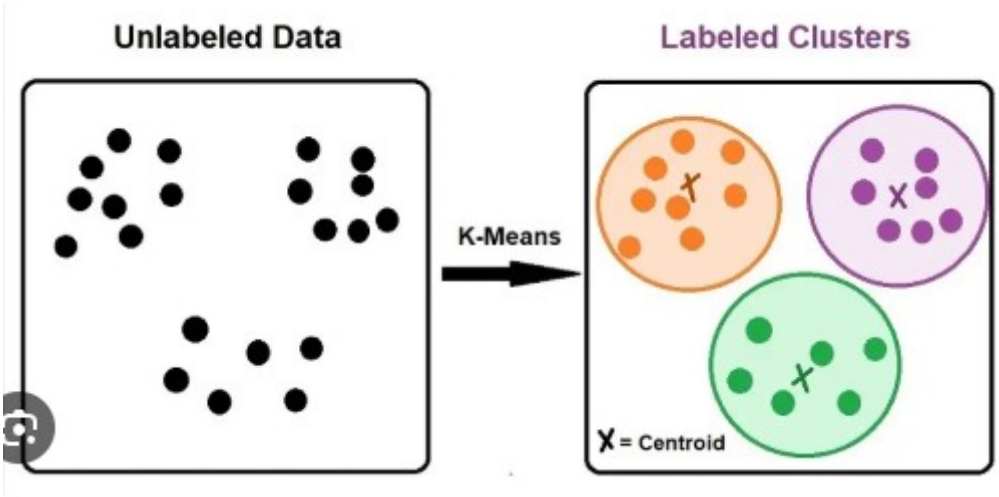
Nearest Neighbor Methods are simple yet powerful **instance-based learning algorithms** used for classification and regression.

The most common version is the **k-Nearest Neighbors (k-NN)** algorithm, which predicts the output for a new data point by finding the **k closest data points** (neighbors) in the training set based on a distance metric, typically **Euclidean distance**. For **classification**, the algorithm assigns the most common class among the neighbors, while for **regression**, it calculates the average of their values.

Nearest neighbor methods are **non-parametric**, meaning they make no assumption about the underlying data distribution. They are intuitive and effective, especially for low-dimensional data. However, they can be computationally expensive during prediction, as they require storing the entire dataset and comparing against all points.

Their performance also heavily depends on the choice of **k** and the **distance metric**, and they may suffer from the **curse of dimensionality** in high-dimensional spaces.

Unsupervised Learning



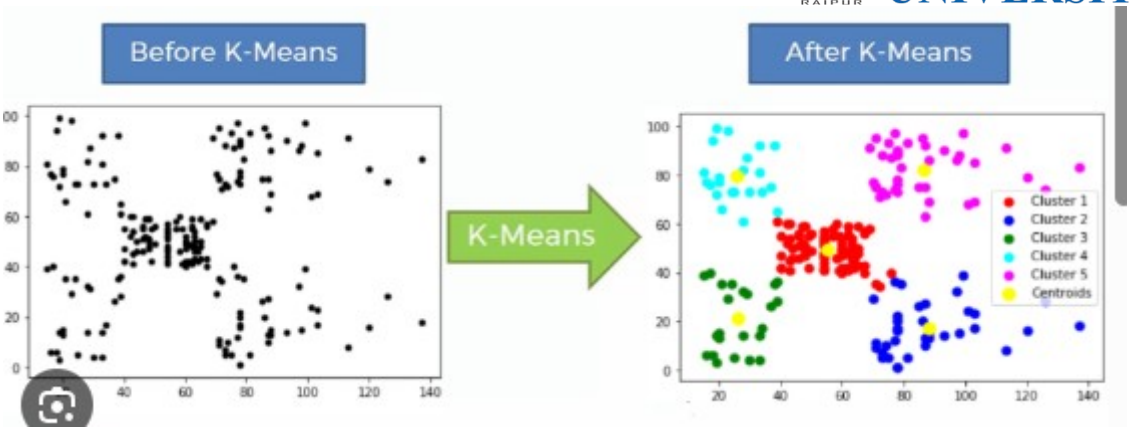
Unsupervised Learning is a type of machine learning where the model is trained on data without labeled outputs.

The goal is to find **hidden patterns or groupings** in the data. One of the most widely used algorithms in this category is the **K-Means Clustering Algorithm**.

Unsupervised Learning is a type of machine learning where the model is trained on data without labeled outputs. The goal is to find **hidden patterns or groupings** in the data. One of the most widely used algorithms in this category is the **K-Means Clustering Algorithm**.

K-Means Algorithm

K-Means is a **centroid-based clustering algorithm** that partitions the dataset into **K distinct non-overlapping clusters**. Each data point is assigned to the cluster whose center (mean) is nearest, and the objective is to minimize the **intra-cluster variance** (within-cluster sum of squares).



🔹 *Key Steps in K-Means:*

1. **Initialization:**
   - Choose the number of clusters **K**.
   - Randomly initialize K centroids (either randomly selected points or via smart methods like K-Means++).
2. **Assignment Step:**
   - Assign each data point to the **nearest centroid** using a distance metric (typically **Euclidean distance**).
3. **Update Step:**
   - Recalculate the **centroids** by taking the **mean of all points** assigned to each cluster.
4. **Repeat:**
   - Steps 2 and 3 are repeated until convergence (i.e., when cluster assignments no longer change or a maximum number of iterations is reached).

🔹 *Objective Function:*

Minimize the **sum of squared distances** between each point and its assigned cluster centroid:

$J=\sum i=1k\sum x\in Ci\|x-\mu i\|2J = \sum_{i=1}^{k} \sum_{x \in C_i} \| x - \mu_i \|^2$

Where:

- CiC_i is the set of points in cluster ii
- $\mu i\mu_i$ is the centroid of cluster ii

🔹 *Important Concepts:*

- **Centroids:** Mean point of all data points in a cluster.
- **Inertia:** Sum of squared distances from each point to its assigned centroid (used as a measure of cluster compactness).
- **Convergence:** When centroids no longer move significantly between iterations.

🔹 *Advantages:*

- Simple and fast, even on large datasets.

- Efficient for low-dimensional, well-separated data.
- Easy to implement and interpret.

🔹 *Limitations:*

- Needs pre-specification of K (number of clusters).
- Sensitive to initial centroid positions (may converge to local minima).
- Works poorly with **non-spherical clusters** or **clusters of varying sizes and densities**.
- Affected by **outliers** and **noisy data**.

🔹 *Improvements and Variants:*

- **K-Means++:** Improves initialization of centroids to reduce the chances of poor convergence.
- **Mini-batch K-Means:** Faster for large datasets by using a subset of the data in each iteration.
- **Elbow Method & Silhouette Score:** Techniques used to choose the optimal number of clusters.

🔹 *Applications:*

- Customer segmentation in marketing
- Image compression and segmentation
- Document and text clustering
- Pattern recognition and anomaly detection

Vector Quantization

Vector Quantization is a technique used primarily in **data compression**, **pattern recognition**, and **clustering**. It works by mapping input data vectors to a finite set of **code vectors** (also called centroids or prototypes), which represent clusters in the data.

🔹 *Key Concepts:*

1. **Codebook:** A set of representative vectors (centroids) created during the training process.
2. **Encoding:** Each input vector is assigned to the nearest code vector, reducing dimensionality and redundancy.
3. **Quantization Error:** The difference between the input vector and its assigned code vector. The goal is to minimize this error across all inputs.
4. **Applications:** Image compression, speech recognition, and signal processing.

🔹 *Process:*

- Initialization of code vectors.
- Assignment of data points to the nearest code vector (like in K-Means).
- Updating code vectors as the average of assigned inputs.
- Repeat until convergence.

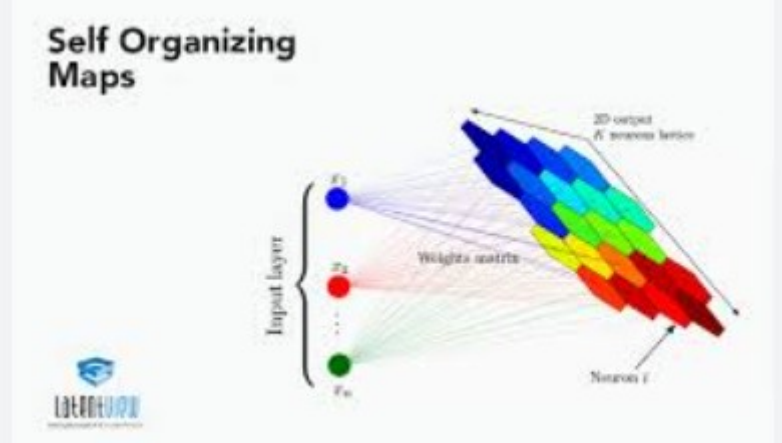
🔹 *Advantages:*

- Reduces storage and transmission costs.
- Fast lookup once the codebook is built.

🔹 *Limitations:*

- Sensitive to initial codebook values.
- Less effective with non-spherical or complex data distributions.

Self Organizing Feature Map



A **Self-Organizing Feature Map**, proposed by **Teuvo Kohonen**, is a type of **artificial neural network** used for **unsupervised learning**, especially for visualizing high-dimensional data in a lower-dimensional (usually 2D) space.

🔹 *Key Characteristics:*

1. **Topology Preservation:** Similar input vectors are mapped to nearby neurons on the grid.
2. **Grid Structure:** Typically a 2D array of neurons, each with a weight vector of the same dimension as the input data.
3. **Competitive Learning:** Only the neuron with the closest weight (called the **Best Matching Unit** or BMU) and its neighbors are updated during learning.
4. **Neighborhood Function:** Determines how much neighboring neurons are updated, usually decreasing over time.

🔹 *Training Steps:*

1. Initialize weight vectors randomly.
2. For each input vector:
    - Find the BMU.
    - Update the BMU and its neighbors' weights to move closer to the input vector.
3. Gradually reduce learning rate and neighborhood radius over time.
4. Repeat for multiple epochs.

🔹 *Advantages:*

- Effective for **visualizing complex data**.
- Preserves the structure and topology of the original dataset.
- Useful in **clustering**, **dimensionality reduction**, and **feature extraction**.

🔹 *Applications:*

- Pattern recognition
- Data visualization (e.g., customer behavior, genome data)
- Speech and image processing
- Robotics and control systems

🔹 *Limitations:*

- Requires careful tuning of parameters (learning rate, neighborhood size).
- Slower training compared to simpler clustering methods.
- May not perform well on very large or noisy datasets without preprocessing.