UNIT - III

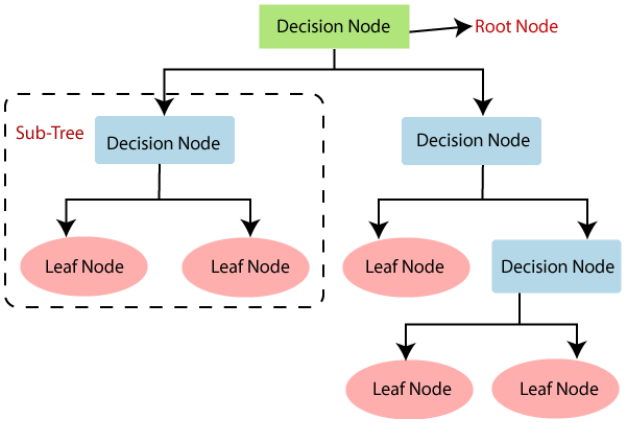
Learning with Trees – Decision Trees – Constructing Decision Trees – Classification and Regression Trees – Ensemble Learning – Boosting – Bagging – Different ways to Combine Classifiers – Basic Statistics – Gaussian Mixture Models – Nearest Neighbor Methods – Unsupervised Learning – K means Algorithms

LEARNING WITH TREES

DECISION TREES

- Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems.
- It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.
- In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node.
- Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.
- The decisions or the tests are performed on the basis of features of the given dataset.
- It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.
- It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.

Below diagram explains the general structure of a decision tree:



Example:

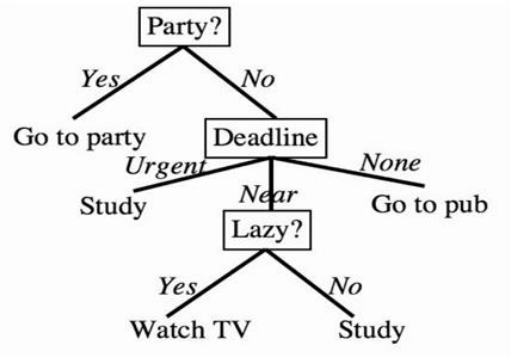


FIGURE 12.1 A simple decision tree to decide how you will spend the evening.

- One of the reasons that decision trees are popular is that we can turn them into a set of logical disjunctions (if ... then rules) that then go into program code very simply.

Ex:

- if there is a party then go to it
- if there is not a party and you have an urgent deadline then study

CONSTRUCTING DECISION TREE

Types of Decision Tree Algorithms:

- **ID3:** This algorithm measures how mixed up the data is at a node using something called entropy. It then chooses the feature that helps to clarify the data the most.
- **C4.5:** This is an improved version of ID3 that can handle missing data and continuous attributes.
- **CART:** This algorithm uses a different measure called Gini impurity to decide how to split the data. It can be used for both classification (sorting data into categories) and regression (predicting continuous values) tasks.

ID3 (Iterative Dichotomiser 3)

The **ID3 (Iterative Dichotomiser 3)** algorithm is a decision tree learning algorithm used in machine learning and data mining for classification tasks. It was developed by **Ross Quinlan** in the 1980s and is the predecessor of more advanced decision tree algorithms like **C4.5** and **CART**.

How the ID3 Algorithm Works?

ID3 builds a decision tree by selecting attributes that maximize information gain (or minimize entropy). The process follows these steps:

1. **Calculate Entropy**
   Entropy measures the impurity or disorder in a dataset. It is calculated using the formula:

$$H(S) = -\sum p_i \log_2(p_i)$$

   where p_i is the probability of class i.

2. **Compute Information Gain**
   Information Gain measures the reduction in entropy achieved by splitting the dataset based on an attribute. It is calculated as:

$$IG(S, A) = H(S) - \sum \frac{|S_v|}{|S|} H(S_v)$$

   where:

   - S is the original dataset
   - S_v is a subset of S for which attribute A has value v
   - $H(S)$ is the entropy of S
   - $H(S_v)$ is the entropy of each subset after the split

3. **Select the Best Attribute**
   The attribute with the **highest Information Gain** is chosen as the root node.
4. **Split the Data**
   The dataset is split based on the selected attribute, and the process is repeated recursively for each subset until one of the stopping conditions is met:
   - All instances in a subset belong to the same class.
   - There are no remaining attributes to split on.
   - The dataset is empty.
5. **Assign a Leaf Node**
   If further splitting is not possible, the most common class label in the subset is assigned to the leaf node.

Advantages of ID3

Simple and easy to implement
Provides a human-readable decision tree
Works well with categorical data

Disadvantages of ID3

Overfits on noisy or small datasets
Cannot handle continuous numerical values directly (must be discretized)
Prefers attributes with many values (can be biased toward high-cardinality attributes)

Example of ID3

Step 1: Calculate Entropy of the Entire Dataset

We first calculate the **entropy** of the dataset before any splits.

Weather	Play Outside?
Sunny	Yes
Rainy	No
Overcast	Yes
Rainy	No
Sunny	Yes

- **Yes = 3 times**
- **No = 2 times**

The entropy formula is:

$$H(S) = -p_{\text{yes}} \log_2(p_{\text{yes}}) - p_{\text{no}} \log_2(p_{\text{no}})$$

Let's compute it.

The entropy of the dataset is **0.971** (rounded).

Step 2: Compute Information Gain for "Weather"

The **Weather** attribute has three unique values:

- **Sunny** → (2 instances: ["Yes", "Yes"])
- **Rainy** → (2 instances: ["No", "No"])

- **Overcast** → (1 instance: ["Yes"])

Step 2.1: Compute Entropy for Each Subset

1. **Sunny (2 instances: ["Yes", "Yes"])**

$$H(Sunny) = -(2/2)\log_2(2/2) = 0$$

2. **Rainy (2 instances: ["No", "No"])**

$$H(Rainy) = -(2/2)\log_2(2/2) = 0$$

3. **Overcast (1 instance: ["Yes"])**

$$H(Overcast) = -(1/1)\log_2(1/1) = 0$$

Since all subsets have entropy **0**, the weighted entropy is:

$$H_{\text{split}}(\text{Weather}) = \left(\frac{2}{5} \times 0\right) + \left(\frac{2}{5} \times 0\right) + \left(\frac{1}{5} \times 0\right) = 0$$

Step 2.2: Compute Information Gain

$$IG(\text{Weather}) = H(S) - H_{\text{split}}(\text{Weather})$$

$$IG(\text{Weather}) = 0.971 - 0 = 0.971$$

✅ Since **Weather perfectly classifies the data (IG = 0.971)**, it is chosen as the **root node of the decision tree**.

Step 3: Decision Tree

Since the entropy after splitting is **0**, we **stop here**. The final tree is:

```
      Weather
    /   |   \
Sunny  Overcast  Rainy
 Yes     Yes      No
```

C4.5 ALGORITHM (IMPROVEMENT OF ID3)

The **C4.5 algorithm** is an improved version of the **ID3 decision tree algorithm** developed by **Ross Quinlan**. It overcomes some limitations of ID3 and is widely used in **classification problems**.

How C4.5 Works

1. **Start with the entire dataset** and calculate the entropy.
2. **Select the best attribute** to split the data using the **Gain Ratio** (instead of just Information Gain).
3. **Create branches** based on attribute values.
4. **Handle missing values** and **continuous data** (ID3 cannot handle these well).
5. **Recursively repeat** the process for each subset until all data is classified.
6. **Use pruning** to remove unnecessary branches and prevent overfitting.

Key Improvements Over ID3

Feature	ID3	C4.5
Splitting Criterion	Information Gain (IG)	Gain Ratio (GR)
Handles Continuous Data	❌ No	☑ Yes
Handles Missing Values	❌ No	☑ Yes
Pruning (to prevent overfitting)	❌ No	☑ Yes
Handles Multiple Classes	☑ Yes	☑ Yes

1. Gain Ratio (Better than Information Gain)

- ID3 uses **Information Gain**, which can favor attributes with many values.
- **C4.5 solves this issue** by introducing **Gain Ratio**, which normalizes Information Gain.
- **Formula for Gain Ratio:**

Formula for Gain Ratio:

$$GR(A) = \frac{IG(A)}{SplitInfo(A)}$$

Where:

- $IG(A)$ = Information Gain of attribute A.
- $SplitInfo(A)$ = Measures how evenly data is split.

This avoids bias toward attributes with many unique values.

2. Handling Continuous Data

- C4.5 **splits numerical attributes** into two groups:
  "Less than threshold" and **"Greater than threshold"**
- It finds the best threshold dynamically instead of requiring pre-defined categories.

3. Handling Missing Values

- Instead of removing rows with missing values, **C4.5 estimates probabilities** based on available data.
- If an attribute is missing, it distributes the instance across possible values **proportionally**.

4. Pruning (Reduces Overfitting)

- **C4.5 performs "post-pruning"** by removing less significant branches.
- This results in a **simpler tree** that generalizes better to new data.

Example: Deciding Whether to Play Outside

Imagine a dataset where we decide whether to **play outside** based on **weather conditions**.

Weather	Temperature	Wind	Play Outside?
Sunny	Hot	Weak	No
Sunny	Hot	Strong	No
Overcast	Hot	Weak	Yes
Rainy	Mild	Weak	Yes
Rainy	Cool	Weak	Yes
Rainy	Cool	Strong	No
Overcast	Cool	Strong	Yes
Sunny	Mild	Weak	No
Sunny	Cool	Weak	Yes
Rainy	Mild	Strong	No
Sunny	Mild	Strong	No
Overcast	Mild	Strong	Yes
Overcast	Hot	Weak	Yes
Rainy	Mild	Weak	Yes

Step 1: Calculate Entropy of the Dataset

Entropy measures uncertainty in the dataset. It is calculated as:

$$H(S) = -p_{Yes} \log_2 p_{Yes} - p_{No} \log_2 p_{No}$$

From the dataset:

- **Yes** = 8
- **No** = 6
  Total = **14**

$$H(S) = -\left(\frac{8}{14} \log_2 \frac{8}{14} + \frac{6}{14} \log_2 \frac{6}{14}\right)$$

Entropy = 0.9852

Step 2: Compute Gain Ratio for Each Attribute

Information Gain of "Weather"

Weather has 3 values: **Sunny, Overcast, Rainy.**

Weather	Total	Yes	No	Entropy
Sunny	5	1	4	0.7219
Overcast	4	4	0	0.0000
Rainy	5	3	2	0.9709

Weighted Entropy for Weather:

$$H(Weather) = \left(\frac{5}{14} \times 0.7219\right) + \left(\frac{4}{14} \times 0.0000\right) + \left(\frac{5}{14} \times 0.9709\right)$$

$$H(Weather) = 0.6041$$

$$\text{Information Gain} = H(S) - H(Weather) = 0.9852 - 0.6041 = 0.3806$$

Split Information for Weather:

$$\text{SplitInfo} = -\left(\frac{5}{14} \log_2 \frac{5}{14} + \frac{4}{14} \log_2 \frac{4}{14} + \frac{5}{14} \log_2 \frac{5}{14}\right)$$

$$\text{SplitInfo} = 1.5761$$

Gain Ratio for Weather:

$$\frac{\text{Information Gain}}{\text{SplitInfo}} = \frac{0.3806}{1.5761} = 0.2413$$

Information Gain of "Temperature"

Temperature has 3 values: **Hot, Mild, Cool.**

Temperature	Total	Yes	No	Entropy
Hot	4	2	2	1.0000
Mild	6	3	3	1.0000
Cool	4	3	1	0.8113

Weighted Entropy for Temperature:

$$H(Temperature) = \left(\frac{4}{14} \times 1.0000\right) + \left(\frac{6}{14} \times 1.0000\right) + \left(\frac{4}{14} \times 0.8113\right)$$

$$H(Temperature) = 0.9461$$

$$\text{Information Gain} = 0.9852 - 0.9461 = 0.0391$$

Gain Ratio for Temperature:

$$\frac{0.0391}{1.5551} = 0.0251$$

Information Gain of "Wind"

Wind has 2 values: **Weak, Strong.**

Wind	Total	Yes	No	Entropy
Weak	8	6	2	0.8113
Strong	6	2	4	0.9183

Weighted Entropy for Wind:

$$H(Wind) = \left(\frac{8}{14} \times 0.8113\right) + \left(\frac{6}{14} \times 0.9183\right)$$

$$H(Wind) = 0.8571$$

$$\text{Information Gain} = 0.9852 - 0.8571 = 0.1281$$

Gain Ratio for Wind:

$$\frac{0.1281}{0.9852} = 0.1300$$

Step 3: Select the Best Attribute and Split the Data

The Gain Ratios are:

- **Weather = 0.2413**
- **Temperature = 0.0251**
- **Wind = 0.1300**

Since **Weather has the highest Gain Ratio**, we choose **Weather** as the root node.

Step 4: Repeat the Process for Each Subset

Subtree for "Weather = Overcast"

- All values are "Yes" → **Leaf Node: "Yes"**

Subtree for "Weather = Rainy"

- Subset:

```yaml
Rainy, Mild, Weak   → Yes
Rainy, Cool, Weak   → Yes
Rainy, Cool, Strong → No
Rainy, Mild, Strong → No
Rainy, Mild, Weak   → Yes
```

- Entropy = **0.9709**
- Best attribute: "**Wind**"
  - **Weak → Yes**
  - **Strong → No**

Subtree for "Weather = Sunny"

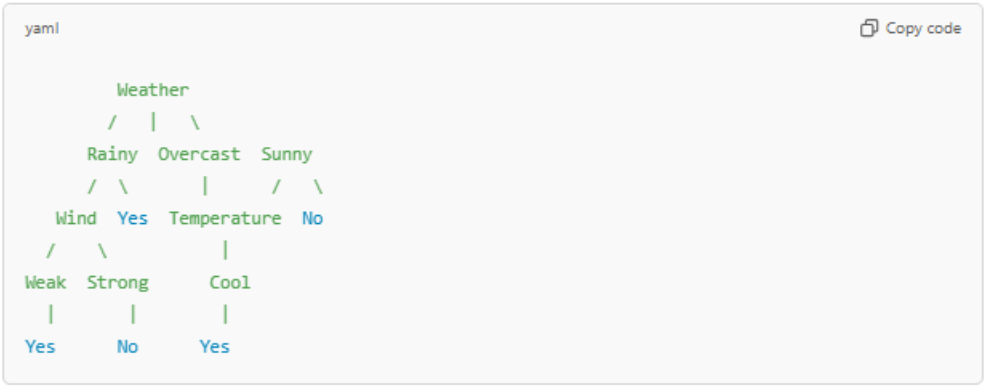
- Subset:

```yaml
Sunny, Hot, Weak   → No
Sunny, Hot, Strong → No
Sunny, Mild, Weak  → No
Sunny, Cool, Weak  → Yes
Sunny, Mild, Strong → No
```

- Entropy = **0.7219**
- Best attribute: "**Temperature**"
  - **Cool → Yes**
  - Else → **No**

Step 5: Final Decision Tree

yaml Copy code



Handling Missing Values

C4.5 handles missing values. Instead of replacing the missing value with a single most common value, it **distributes the missing instance proportionally** across the possible values.

Example:

Weather	Temperature	Play Tennis?
Sunny	**Hot**	No
Rainy	**? (Missing)**	Yes
Rainy	**Mild**	No
Overcast	**Hot**	Yes
Rainy	**Hot**	Yes

Step 1: Find Other "Rainy" Rows:

We look at all rows where **Weather = Rainy** and check their **Temperature** values:

Weather	Temperature
Rainy	**Mild**
Rainy	**Hot**
Rainy	**? (Missing)**

We see that:

- **Hot appears 60% of the time**

- **Mild appears 40% of the time**

Step 2: Distribute the Missing Value

Instead of assuming **Hot or Mild**, **C4.5 splits the missing row:**

- **60% of the row is treated as "Hot"**
- **40% of the row is treated as "Mild"**

Now, the dataset **conceptually** looks like:

Weather	Temperature	Play Tennis?	Weight
Sunny	Hot	No	1
Rainy	Hot (60%)	Yes	0.6
Rainy	Mild (40%)	Yes	0.4
Rainy	Mild	No	1
Overcast	Hot	Yes	1
Rainy	Hot	Yes	1

Step 3: Compute Entropy (Using Weighted Contributions)

Since the missing row is **split between "Hot" and "Mild"**, entropy and information gain are calculated by **weighting the contributions accordingly**.

How to handle continuous data?

Handling Continuous (Numerical) Data

Unlike ID3, which requires categorical attributes, **C4.5 can split numerical data dynamically** by finding the best threshold.

How It Works:

1. **Sort the numerical values** in ascending order.
2. **Find the best split point** by testing different thresholds.
3. **Convert the numerical attribute into two categories:**
    - **≤ threshold**
    - **> threshold**

Example:

Temperature	Play Outside?
15°C	No
18°C	No
22°C	Yes
24°C	Yes
30°C	Yes

Step 1: Find possible split points between values:

- Between 18°C & 22°C → **Threshold = 20°C**
- Between 22°C & 24°C → **Threshold = 23°C**

Step 2: Compute Information Gain for each threshold and pick the best one.

- Suppose **20°C gives the highest Gain Ratio**, C4.5 splits the data:
  - **≤ 20°C → "No"**
  - **> 20°C → "Yes"**

Now, "Temperature" is treated like a categorical variable without predefining ranges!

CLASSIFICATION AND REGRESSION TREES (CART)

The **CART algorithm (Classification and Regression Trees)** is a decision tree learning technique used for classification and regression tasks. It was introduced by **Breiman et al. (1984)** and is widely used in machine learning for predictive modelling.

How CART Works

CART constructs binary decision trees by recursively splitting the dataset into two subsets based on feature values. The algorithm selects the best split at each step using **Gini impurity (for classification)** or **mean squared error (for regression)**.

Steps in the CART Algorithm

1. Start with the Entire Dataset (Root Node)

- The root node contains all the training samples.
- The goal is to find the best feature and value to split the dataset into two groups.

2. Choose the Best Split

- For **classification** problems, the split is chosen based on **Gini Index** (default in CART).
- For **regression** problems, the split is chosen based on **Mean Squared Error (MSE)**.

Splitting Criteria:

- **Gini Index (for classification)**

$$G = 1 - \sum p_i^2$$

pi is the probability of each class. A lower Gini Index means purer nodes.

- **Mean Squared Error (MSE) (for regression)**

$$MSE = \frac{1}{n} \sum (y_i - \bar{y})^2$$

Measures the variance within the group

3. Recursively Split the Dataset

- The dataset is split into two subsets at each step.
- The process continues until a **stopping condition** is met.

4. Pruning (Optional):

- Reduce tree complexity by pruning unnecessary nodes to prevent overfitting.

5. Define Stopping Criteria

- The tree stops growing if:
  - A node contains only **one class** (classification).
  - The number of samples in a node is less than a threshold.
  - The **maximum depth** is reached.
  - The **information gain** is too small.

6. Assign Leaf Node Values

- For classification, assign the most common class in that node.
- For regression, assign the average target value.

1. CART for Classification (Using Gini Index)

The **Gini Index** measures the impurity of a node. The formula for Gini Index is:

$$Gini = 1 - \sum(p_i^2)$$

where p_i is the probability of class i.

Example Dataset

ID	Feature: Age	Label: Play Tennis (Yes=1, No=0)
1	25	Yes (1)
2	30	Yes (1)
3	35	No (0)
4	40	No (0)
5	45	No (0)

Step 1: Calculate Gini Index for Root Node

There are 2 "Yes" and 3 "No":

$$Gini = 1 - ((2/5)^2 + (3/5)^2) = 1 - (0.16 + 0.36) = 1 - 0.52 = 0.48$$

Step 2: Find the Best Split

Let's split the dataset at **Age = 30**:

- **Left Node (Age ≤ 30)**: { (25, Yes), (30, Yes) } → 2 Yes
- **Right Node (Age > 30)**: { (35, No), (40, No), (45, No) } → 3 No

Gini for Left Node

$$Gini = 1 - ((2/2)^2 + (0/2)^2) = 1 - (1 + 0) = 0$$

Gini for Right Node

$$Gini = 1 - ((0/3)^2 + (3/3)^2) = 1 - (0 + 1) = 0$$

Weighted Gini

$$Gini_{split} = \frac{2}{5}(0) + \frac{3}{5}(0) = 0$$

Since the split at **Age = 30** results in **Gini = 0**, this is the best split.

Final Decision Tree

```
      Age <= 30?
     /      \
   Yes      No
```

2. CART for Regression (Using Mean Squared Error)

For regression, CART splits the data based on **Mean Squared Error (MSE)**:

$$MSE = \frac{1}{n} \sum (y_i - \hat{y})^2$$

where y_i are actual values and $\hat{y}$ is the mean of the subset.

Example Dataset

ID	Feature: Age	Target: Salary (in $1000s)
1	25	50
2	30	55
3	35	60
4	40	70
5	45	80

Step 1: Calculate MSE for Root Node

Mean Salary:

$$\bar{y} = \frac{50 + 55 + 60 + 70 + 80}{5} = 63$$

$$MSE = \frac{(50-63)^2 + (55-63)^2 + (60-63)^2 + (70-63)^2 + (80-63)^2}{5}$$

$$= \frac{169 + 64 + 9 + 49 + 289}{5} = \frac{580}{5} = 116$$

Step 2: Find the Best Split

Let's split at **Age = 35**:

- **Left Node (Age ≤ 35)**: { (25, 50), (30, 55), (35, 60) }

- **Right Node (Age > 35)**: { (40, 70), (45, 80) }

Final Decision Tree

MSE for Left Node

Mean = $\frac{50+55+60}{3} = 55$

$$MSE = \frac{(50-55)^2 + (55-55)^2 + (60-55)^2}{3} = \frac{25+0+25}{3} = 16.67$$

MSE for Right Node

Mean = $\frac{70+80}{2} = 75$

$$MSE = \frac{(70-75)^2 + (80-75)^2}{2} = \frac{25+25}{2} = 25$$

Weighted MSE

$$MSE_{split} = \frac{3}{5}(16.67) + \frac{2}{5}(25) = 10 + 10 = 20$$

Since **MSE is lower after splitting**, we split at **Age = 35**.

Age ≤ 35?
/ \
55 75

Advantages and Disadvantages of the **CART (Classification and Regression Trees) algorithm**:

Aspect	Advantages	Disadvantages
Interpretability	Easy to understand and interpret (visualizable as a tree).	Large trees become complex and hard to interpret.
Handling Data	Works well with numerical and categorical data.	Sensitive to small changes in data (can lead to different trees).
Feature Selection	Automatically selects the most important features.	Can be biased towards features with more levels (e.g., continuous data).
Non-Linearity	Captures non-linear relationships well.	Struggles with smooth functions (can create a step-like boundary).
Overfitting	Can be controlled using **pruning** or **max depth constraints**.	Without pruning, it tends to overfit the training data.
Preprocessing	Requires little to no data preprocessing (no scaling needed).	Splits can be biased if data is imbalanced.
Computational Cost	Faster than many complex models (e.g., neural networks).	Can become computationally expensive for very deep trees.

Aspect	Advantages	Disadvantages
Missing Values	Can handle missing values naturally.	Does not inherently perform well on missing data without imputation.

ENSEMBLE LEARNING

Ensemble learning is a technique in machine learning where multiple models (often called **weak learners** or **base models**) are combined to create a stronger, more accurate model. The main idea is that multiple models working together can reduce errors and improve predictions compared to a single model.

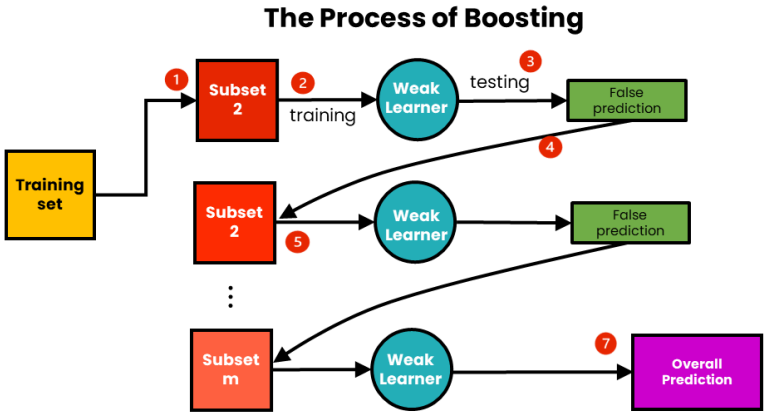
Types of Ensemble Learning

1. Boosting

- Models are trained sequentially, where each new model **corrects the mistakes** of the previous ones.
- Helps **reduce bias** and improve weak models.
  Example: **AdaBoost, Gradient Boosting (XGBoost, LightGBM, CatBoost)**.

Steps in Boosting:

1. Train a model on the dataset.
2. Identify the incorrectly predicted samples and assign **higher weights**.
3. Train a new model that focuses on correcting these mistakes.
4. Repeat the process for several iterations.



Example: AdaBoost (Adaptive Boosting)

- Assigns more weight to misclassified samples and improves the next weak learner.

AdaBoost (Adaptive Boosting) Algorithm

AdaBoost (Adaptive Boosting) is an **ensemble learning** method that combines multiple **weak learners** (usually decision stumps) to create a strong classifier. It **adjusts the weights** of misclassified samples to focus more on difficult cases in each iteration.

How AdaBoost Works

1. **Initialize Weights**:
   - Assign equal weights to all training samples.

   Each sample i gets an initial weight:

   $$w_i = \frac{1}{N}$$

   where N is the number of samples.

2. **Train a Weak Learner**:
   - A weak model (e.g., Decision Stump) is trained on the dataset.

   A weak model $h_t(x)$ is trained on the weighted dataset.

3. **Calculate Error**:
   - The error is measured as the total weight of misclassified samples.

   $$e_t = \sum_{i=1}^{N} w_i \cdot I(h_t(x_i) \neq y_i)$$

   where:

   - $I(h_t(x_i) \neq y_i)$ is 1 if the model is incorrect, else 0.
   - e_t is the total weight of misclassified samples.

4. **Update Model Weight**:
   - The model is given a weight based on its accuracy.
   - More accurate models get higher weights.

   $$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - e_t}{e_t} \right)$$

   - If error is low, α_t is high (stronger model).
   - If error is high, α_t is low.

5. **Update Sample Weights**:
    - Misclassified samples get **higher weights** so the next model focuses more on them.

$$w_i = w_i \times e^{\alpha_t}$$

- Increase weights for misclassified samples.
- Normalize so all weights sum to 1.

6. **Repeat Steps 2-5**:
    - Train multiple weak models iteratively.
    - Combine them using a weighted majority vote (for classification) or weighted sum (for regression).

1. Classification: Weighted Majority Vote

Each weak classifier gets a weight based on its accuracy. More accurate classifiers get **higher influence** on the final decision.

Formula for Final Prediction

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

where:

- $H(x)$ = final prediction
- α_t = weight of the weak classifier h_t (computed as $\alpha_t = \frac{1}{2}\ln\left(\frac{1-e_t}{e_t}\right)$)
- $h_t(x)$ = prediction of weak classifier t
- T = total number of weak classifiers

2. Regression: Weighted Sum of Predictions

For regression problems, AdaBoost uses a **weighted sum** of weak learners.

Formula for Final Prediction

$$H(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$$

where:

- $H(x)$ = final regression prediction
- α_t = weight of the weak model (inversely related to mean squared error)
- $h_t(x)$ = prediction from weak model t

AdaBoost Algorithm:

- **Initialize Weights:** Assign equal weights to all training samples.
- **Train a Weak Learner:** Train a simple model (like a **Decision Stump**, a one-level decision tree).
- **Calculate Error:** Check how many samples are misclassified. If a sample is misclassified, increase its weight so the next model focuses on it more.
- **Update Weights:** Increase the importance (weight) of misclassified samples. Reduce the weight of correctly classified samples.
- **Repeat Steps 2-4:** Train multiple weak learners, each correcting the mistakes of the previous one.
- **Final Prediction:** Combine all weak learners using a **weighted majority vote** (for classification) or **weighted sum** (for regression).

Example: AdaBoost (Adaptive Boosting) is an **ensemble learning algorithm** that combines multiple weak classifiers to form a strong classifier.
It **iteratively trains weak models** and **adjusts sample weights** to focus on difficult cases.

Step 1: Dataset

Consider a small **binary classification dataset**:

Sample	Feature x	Class y
1	1.0	+1
2	2.0	-1
3	3.0	+1
4	4.0	-1

Each sample belongs to either **Class +1** or **Class -1**.

Step 2: Initialize Sample Weights

Initially, all samples are given equal importance. Since we have **4 samples**, their initial weight is:

$$w_i = \frac{1}{N} = \frac{1}{4} = 0.25$$

Sample	Weight wi
1	0.25
2	0.25
3	0.25
4	0.25

Step 3: Train the First Weak Classifier

We use a **Decision Stump** (one-level decision tree).
A decision stump chooses a **single feature threshold** to split the data.

Let's say our first weak classifier predicts:

- If **x<2.5, predict +1**
- If **x≥2.5, predict -1**

Sample	Feature x	True y	Prediction h1(x)	Correct?
1	1.0	+1	+1	☑
2	2.0	-1	+1	❌
3	3.0	+1	-1	❌
4	4.0	-1	-1	☑

Misclassified samples: **x=2.0 and x=3.0**

Step 4: Compute Weighted Error $\epsilon 1$

The error of the weak classifier is the **sum of weights of misclassified samples**:

$$\epsilon_1 = w_2 + w_3 = 0.25 + 0.25 = 0.5$$

Step 5: Compute Classifier Weight $\alpha 1$

The weight of the weak classifier is given by:

$$\alpha_1 = \frac{1}{2} \ln \left(\frac{1 - \epsilon_1}{\epsilon_1} \right)$$

Substituting $\epsilon_1 = 0.5$:

$$\alpha_1 = \frac{1}{2} \ln \left(\frac{1 - 0.5}{0.5} \right) = \frac{1}{2} \ln(1) = 0$$

Since $\alpha_1 = 0$, this classifier **has no influence**. In practice, a better weak learner is chosen.

Step 6: Update Sample Weights

We update weights using:

$$w_i^{(t+1)} = w_i^{(t)} \times e^{\pm \alpha_t}$$

- **Increase weight for misclassified samples** → Make them more important.
- **Decrease weight for correctly classified samples** → Make them less important.

Since $\alpha_1 = 0$, weights **stay the same**. In a real case, a better weak classifier is chosen.

Step 7: Repeat for More Weak Learners

After **several iterations**, we get multiple weak classifiers h1,h2,h3,... with different weights αt.

Each classifier focuses more on **previously misclassified samples**.

Step 8: Final Prediction

The final prediction is made by **combining all weak classifiers**:

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

This means:

- Classifiers with **higher weights** (α_t) contribute more.
- The **sign function** determines the final class.

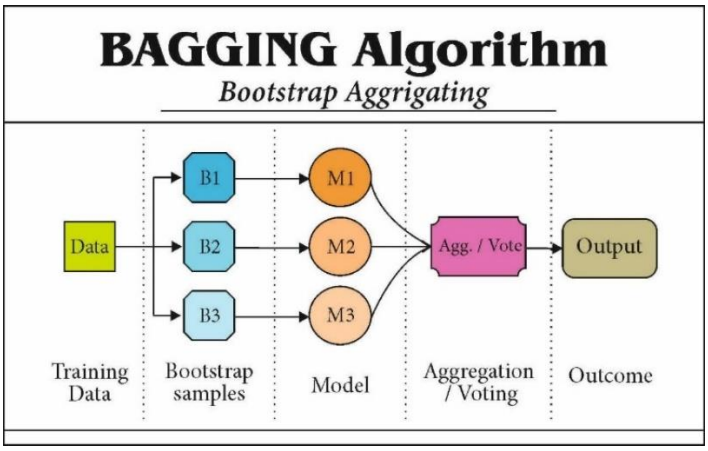
2. Bagging (Bootstrap Aggregating)

- Multiple models (usually the same algorithm) are trained on **random subsets** of data.
- Predictions are averaged (for regression) or voted (for classification).
- Helps **reduce variance** and prevents overfitting.
  Example: **Random Forest** (combines multiple decision trees).

Steps in Bagging:

1. Create multiple training datasets using **random sampling with replacement**.
2. Train a separate model on each dataset.

3. Combine the predictions using **majority voting** (for classification) or **averaging** (for regression).



Example: Random Forest (Bagging on Decision Trees)

- Instead of a single Decision Tree, Random Forest builds **multiple trees** and combines their predictions.
- The random forest algorithm is a machine learning technique that uses multiple decision trees to make predictions. It can be used for classification and regression tasks

How the Random Forest algorithm works:

1. Create multiple datasets → Randomly pick data with replacement (some data may be repeated).
2. Train multiple decision trees → Each tree learns from a different dataset.
3. Make predictions → Each tree makes its own prediction.
4. Combine the results →
    - For classification → Take the majority vote (most common prediction).
    - For regression → Take the average of all predictions.
5. More trees = better accuracy & less overfitting.
6. Every tree in the forest makes its own predictions without relying on others.
7. Each tree is built using random samples and features to reduce mistakes.
8. Sufficient data ensures the trees are different and learn unique patterns and variety.
9. Combining the predictions from different trees leads to a more accurate final result.

The Basic Random Forest Training Algorithm

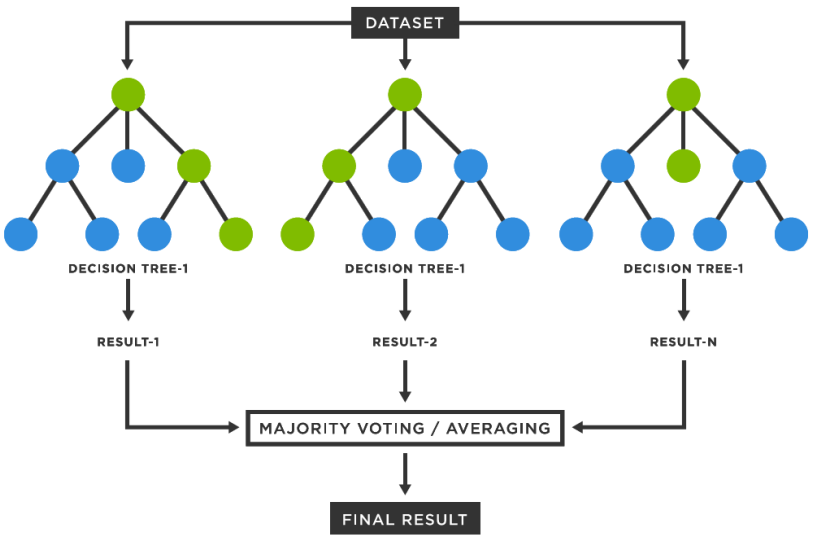
- For each of N trees:
  - create a new bootstrap sample of the training set
  - use this bootstrap sample to train a decision tree
  - at each node of the decision tree, randomly select m features, and compute the information gain (or Gini impurity) only on that set of features, selecting the optimal one
  - repeat until the tree is complete

Advantages of Random Forest

- Random Forest provides very accurate predictions even with large datasets.
- Random Forest can handle missing data well without compromising with accuracy.
- It doesn't require normalization or standardization on dataset.
- When we combine multiple decision trees it reduces the risk of overfitting of the model.

Limitations of Random Forest

- It can be computationally expensive especially with a large number of trees.
- It's harder to interpret the model compared to simpler models like decision trees.



DIFFERENT WAYS TO COMBINE CLASSIFIERS

Combining multiple classifiers can improve machine learning model performance by leveraging the strengths of different algorithms. There are various ways to combine classifiers:

- **Voting:** It is a method to combine predictions from multiple models in ensemble learning. There are two types of voting
    1. **Majority Voting (Hard voting)**: The most common approach, where each classifier casts a vote for a class, and the class with the most votes is chosen as the final prediction.
    2. **Averaging (Soft voting):** Models give probabilities and the class with the highest average probabilities is chosen.

- **Stacking:** Train several models, then we use another model (meta model) to combine their predictions for better result.
- **Bagging (Bootstrap Aggregating)**

  Trains multiple instances of the same classifier on different subsets of data. Reduces variance and prevents overfitting. Example: Random Forest (uses bagging with decision trees).
- **Boosting**

  Sequentially trains classifiers, where each new model focuses on the mistakes of the previous ones. Reduces bias and increases accuracy. Examples:

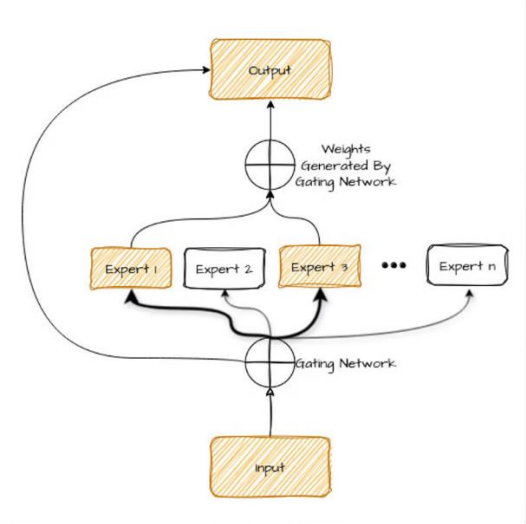
    - **AdaBoost**: Assigns higher weights to misclassified instances.
    - **Gradient Boosting**: Uses gradient descent to minimize loss.
    - **XGBoost**: Optimized version of gradient boosting.

MIXTURE OF EXPERTS (MOE) ALGORITHM IN MACHINE LEARNING

The **Mixture of Experts (MoE)** is an **ensemble learning technique** that **divides a complex problem into subproblems** and assigns specialized models (called *experts*) to solve each subproblem. A gating network learns to **combine the outputs** of these experts to make a final prediction.

MoE is inspired by **divide-and-conquer strategies** in problem-solving. Instead of training a single model to handle all cases, MoE allows different models to specialize in different regions of the input space.

It is widely used in **deep learning** and **large-scale AI models**, such as **Google's Switch Transormers**, which use MoE to efficiently allocate computational resources.



Input: This is the problem or data you want to handle.

Experts: These are smaller models, each trained to be really good at a specific part of the overall problem. Think of them like the different specialists on your team.

Gating network: This is like a manager who decides which expert is best suited for each part of the problem. It looks at the input and figures out who should work on what.

Output: This is the final answer or solution that the model produces after the experts have done their work.

Let's define the mathematical structure of MoE:

- Suppose we have **N experts**, each denoted as $E_i(x)$, where $i = 1, 2, ..., N$.
- The **gating network** assigns a weight $g_i(x)$ to each expert based on the input x.
- The final output is a **weighted sum** of expert predictions:

$$y = \sum_{i=1}^{N} g_i(x) E_i(x)$$

where:

- $g_i(x)$ is the gating function that assigns a probability to each expert.
- $E_i(x)$ is the output of expert i.
- The gating function ensures $\sum g_i(x) = 1$ (typically using a softmax function):

Advantages of MoE

Scalability – MoE can handle large-scale problems by distributing tasks across specialized models.
Improved Accuracy – Experts specialize in different areas, leading to better generalization.
Parallel Computation – Experts can run independently**,** making MoE efficient for distributed computing.
Reduced Overfitting – Specialization prevents overfitting to general patterns.

Disadvantages of MoE

Complexity – Requires careful tuning of experts and the gating function.
Training Instability – If the gating network overfits, it may favor only a single expert.
Computational Cost – Large MoE models require more memory and computation.

BASIC STATISTICS

Mean:

- The "mean" is the average value of a dataset.
- It is calculated by adding up all the values in the dataset and dividing by the number of observations.
- The mean is a useful measure of central tendency because it is sensitive to outliers, meaning that extreme values can significantly affect the value of the mean.

Median:

- The "median" is the middle value in a dataset.
- It is calculated by arranging the values in the dataset in order and finding the value that lies in the middle.
- If there are an even number of values in the dataset, the median is the average of the two middle values.
- The median is a useful measure of central tendency because it is not affected by outliers, meaning that extreme values do not significantly affect the value of the median.

Mode:

- The "mode" is the most common value in a dataset.
- It is calculated by finding the value that occurs most frequently in the dataset.
- If there are multiple values that occur with the same frequency, the dataset is said to be bimodal, trimodal, or multimodal.
- The mode is a useful measure of central tendency because it can identify the most common value in a dataset.
- However, it is not a good measure of central tendency for datasets with a wide range of values or datasets with no repeating values.

Variance:

Variance, in statistics, is a measure of how spread out or dispersed data points are from their average (mean), calculated by averaging the squared differences from the mean.

$$\text{Variance } (s^2) = \sum \frac{(x_i - \bar{x})^2}{N-1}$$

Where,

- x_i is the i^{th} observation,
- $\bar{x}$ is the mean, and
- N is the number of observations

Covariance:

Covariance is a measure of relationship between two variables that is scale dependent, i.e. how much will a variable change when another variable changes.

$$\text{Covariance } (x, y) = \sum \frac{(x_i - \bar{x})(y_i - \bar{y})}{N-1}$$

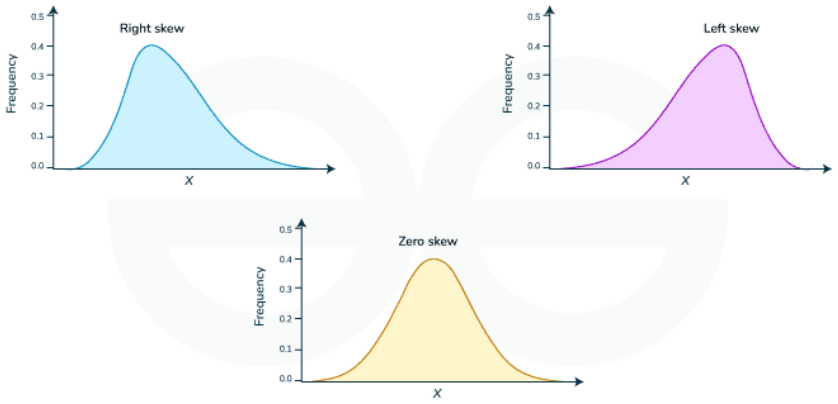
Where,

- x_i is the i^{th} observation in variable x,
- $\bar{x}$ is the mean for variable x,
- y_i is the i^{th} observation in variable y,
- $\bar{y}$ is the mean for variable y, and
- N is the number of observations

Standard Deviation: The square root of the variance is known as the standard deviation.

Interquartile Range: The range between the first and third quartiles, measuring data spread around the median.

Skewness: Indicates data asymmetry.



Positive Skewness (Right Skew): In a positively skewed distribution, the tail on the right side (the larger values) is longer than the tail on the left side (the smaller values).
In the case of a positively skewed dataset,

$$Mean > Median > Mode$$

Negative Skewness (Left Skew): In a negatively skewed distribution, the tail on the left side (the smaller values) is longer than the tail on the right side (the larger values). In the case of a negatively skewed dataset,

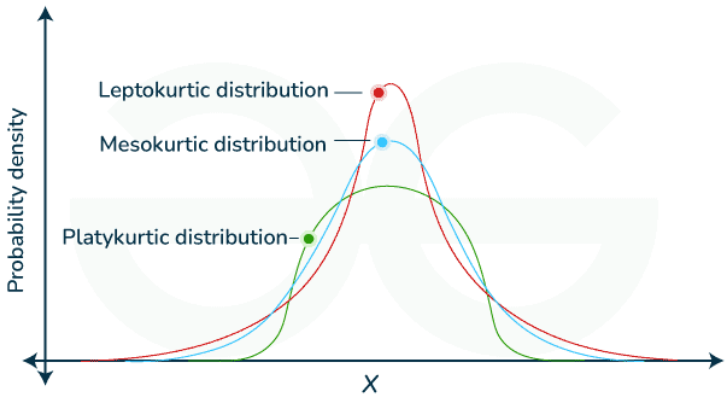
$$Mean < Median < Mode$$

Zero Skewness (Symmetrical Distribution): Zero skewness indicates a perfectly symmetrical distribution, where the mean, median, and mode are equal.

Kurtosis: It is also a characteristic of the frequency distribution. It gives an idea about the shape of a frequency distribution. Basically, the measure of kurtosis is the extent to which a frequency distribution is peaked in comparison with a normal curve.

Types of Kurtoses: The following figure describes the classification of kurtosis:

- **Leptokurtic:** Leptokurtic is a curve having a high peak than the normal distribution. In this curve, there is too much concentration of items near the central value.
- **Mesokurtic:** Mesokurtic is a curve having a normal peak than the normal curve. In this curve, there is equal distribution of items around the central value.
- **Platykurtic:** Platykurtic is a curve having a low peak than the normal curve is called platykurtic. In this curve, there is less concentration of items around the central value.



Mahalanobis Distance: The Mahalanobis distance is a statistical measurement that determines how far a point is from a distribution. It's used in many fields, including computer science, chemometrics, and cluster analysis.

It is a powerful technique that considers the correlations between variables in a dataset, making it a valuable tool in various applications such as outlier detection, clustering, and classification.

$$D^2 = (x-\mu)^T\Sigma^{-1}(x-\mu)$$

Where D^2 is the squared Mahalanobis Distance, x is the point in question, μ is the mean vector of the distribution, Σ is the covariance matrix of the distribution, and ^{T} denotes the transpose of a matrix.

The Gaussian / Normal Distribution: Normal distribution, also known as the Gaussian distribution, is a continuous probability distribution that is symmetric about the mean, depicting that data near the mean are more frequent in occurrence than data far from the mean.

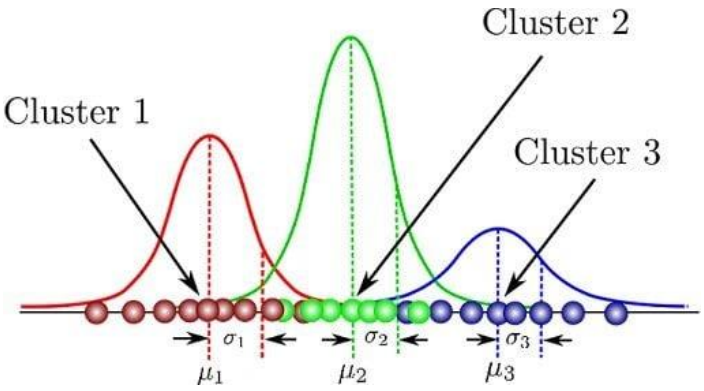
$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

$\mu =$ mean of x
$\sigma =$ standard deviation of x
$\pi \approx 3.14159 \ldots$
$e \approx 2.71828 \ldots$

GAUSSIAN MIXTURE MODELS

A Gaussian mixture model is a soft clustering technique used in unsupervised learning to determine the probability that a given data point belongs to a cluster. It's composed of several Gaussians, each identified by $k \in \{1,\ldots, K\}$, where K is the number of clusters in a data set.

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.



- A mean μ that defines its center.
- A covariance Σ that defines its width. Define the shape and spread of each component.
- A mixing probability π (weights) that defines Probability of selecting each component.

Model Training

- Training a GMM involves setting the parameters using available data.
- The Expectation-Maximization (EM) technique is often employed, alternating between the Expectation (E) and Maximization (M) steps until convergence.

Expectation-Maximization:

- During the E step, the model calculates the probability of each data point belonging to each Gaussian component.
- The M step then adjusts the model's parameters based on these probabilities.

Key Ideas Behind GMM

1. **Mixture of Gaussians**
   - Instead of assuming all points belong to just one cluster (like in k-means), GMM assumes data is a mix of several Gaussian distributions.
   - Each distribution represents one hidden group (e.g., different flavors of candy).
2. **Soft Clustering (Probabilities Instead of Hard Labels)**
   - Instead of saying, "This point is in **Cluster A**," GMM says, "This point is **70% likely** to be in **Cluster A** and **30% likely** to be in **Cluster B**."
3. **Expectation-Maximization (EM) Algorithm**
   - Since we don't know which Gaussian a point belongs to, we start with a guess.
   - We then refine this guess using the **E-step** (Expectation) and **M-step** (Maximization) until the clusters make sense.

Example: Imagine a Class of Students

Let's say we measure the **heights of students** in a school. If we plot the heights, we might see **three peaks** in the data.

- One peak for **elementary students** (shorter kids).
- Another peak for **middle school students** (medium height).
- A final peak for **high school students** (taller kids).

GMM assumes that each peak represents a **Gaussian distribution**, and the overall height distribution is just a mix of these three groups.

If we give a new student's height, GMM can tell us the **probability** that the student belongs to each group.

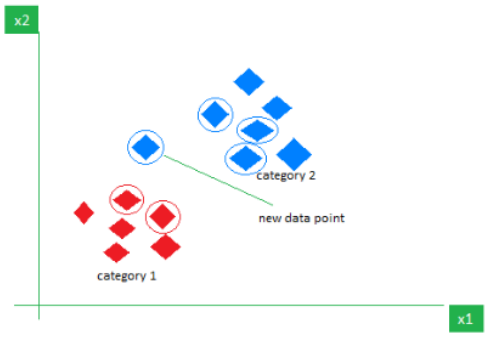
NEAREST NEIGHBOR METHODS

K-Nearest Neighbors Algorithm

K-Nearest Neighbors (KNN) is a simple way to classify things by looking at what's nearby. The K-Nearest Neighbors (KNN) algorithm is a supervised machine learning method employed to tackle classification and regression problems.

K-Nearest Neighbors is also called as a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification it performs an action on the dataset.

As an example, consider the following table of data points containing two features:



The new point is classified as **Category 2** because most of its closest neighbors are blue squares. KNN assigns the category based on the majority of nearby points.
The image shows how KNN predicts the category of a **new data point** based on its closest neighbours.

- The **red diamonds** represent **Category 1** and the **blue squares** represent **Category 2**.

- The **new data point** checks its closest neighbours (circled points).
- Since the majority of its closest neighbours are blue squares (Category 2) KNN predicts the new data point belongs to Category 2.

How algorithm works:

Step 1: Selecting the optimal value of K

• K represents the number of nearest neighbors that needs to be considered while making prediction.

Step 2: Calculating distance

• To measure the similarity between target and training data points, Euclidean distance is used. Distance is calculated between each of the data points in the dataset and target point.

Step 3: Finding Nearest Neighbors

• The k data points with the smallest distances to the target point are the nearest neighbors.

Step 4: Voting for Classification or Taking Average for Regression

• In the classification problem, the class labels of K-nearest neighbors are determined by performing majority voting. The class with the most occurrences among the neighbors becomes the predicted class for the target data point.

• In the regression problem, the class label is calculated by taking average of the target values of K nearest neighbors. The calculated average value becomes the predicted output for the target data point.

Example:

Given Query:
X= (Maths=6, CS=8) → Find the class?

Step 1: Select K neighbors

Maths	**CS**	**Result**
4	3	Fail
6	7	Pass
6	8	Pass
5	5	Fail
8	8	Pass

Given K=3.

Euclidean Formula:

$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

Where:

- x_1, y_1 = Actual values from the table
- x_2, y_2 = Observed value i.e. (6, 8)

Step 2: Calculate the Euclidean distance for K number of neighbors:

1. $\sqrt{(6-4)^2 + (8-3)^2} = \sqrt{4+25} = \sqrt{29} \approx 5.38$
2. $\sqrt{(6-6)^2 + (8-7)^2} = \sqrt{0+1} = \sqrt{1} = 1$ (✔ Pass)
3. $\sqrt{(6-6)^2 + (8-8)^2} = \sqrt{0+0} = 0$ (✔ Pass)
4. $\sqrt{(6-5)^2 + (8-5)^2} = \sqrt{1+9} = \sqrt{10} \approx 3.16$
5. $\sqrt{(6-8)^2 + (8-8)^2} = \sqrt{4+0} = \sqrt{4} = 2$ (✔ Pass)

Step 3:

As per the result, K=3 and we need to consider the 3 smallest values (smallest distances) from the new data point to the actual data points.

- Majority of the data points are **Pass**.

Thus, we assign the new data point into the **Pass** category.

Therefore: Maths=6, CS=8⇒Result is Pass

Advantages and Disadvantages of the KNN Algorithm

Advantages:

- **Easy to implement:** The KNN algorithm is easy to implement because its complexity is relatively low as compared to other machine learning algorithms.

- **No training required:** KNN stores all data in memory and doesn't require any training so when new data points are added it automatically adjusts and uses the new data for future predictions.
- **Few Hyperparameters:** The only parameters which are required in the training of a KNN algorithm are the value of k and the choice of the distance metric which we would like to choose from our evaluation metric.
- **Flexible**: It works for **Classification** problem like is this email spam or not? and also work for **Regression task** like predicting house prices based on nearby similar houses.

Disadvantages:

- **Doesn't scale well:** KNN is considered as a "lazy" algorithm as it is very slow especially with large datasets
- **Curse of Dimensionality:** When the number of features increases KNN struggles to classify data accurately a problem known as curse of dimensionality.
- **Prone to Overfitting:** As the algorithm is affected due to the curse of dimensionality it is prone to the problem of overfitting as well.

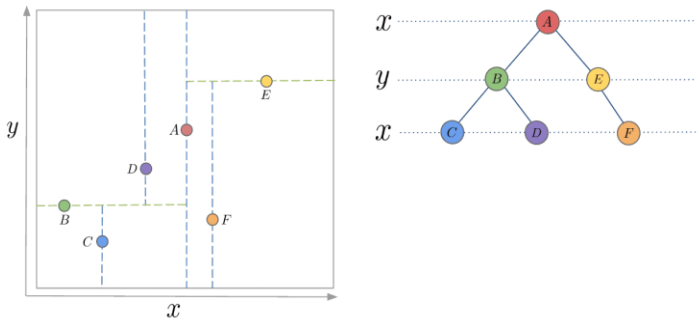
K-dimensional tree

A **k-d tree** is a special kind of **binary search tree** that helps organize points in multiple dimensions (like 2D or 3D space).

Imagine you have a list of locations on a map (like stores or houses), and you want to quickly find the one closest to you. Instead of checking every single location one by one, a **k-d tree** organizes them in a way that makes searching much faster.

How does it work?

1. It starts by dividing the space based on one coordinate (like splitting a map along a vertical line).
2. Then, it keeps dividing the smaller sections using other coordinates (like splitting horizontally next).
3. This process continues, making it easier to search for nearby points.



The **purpose** of a **k-d tree** is to efficiently organize and search points in **multiple dimensions** (2D, 3D, or higher).

1. **Fast Nearest Neighbor Search**
    - Example: Finding the closest gas station or restaurant to your location.
2. **Range Search**
    - Example: Finding all delivery addresses within a certain distance from a warehouse.
3. **Efficient Spatial Partitioning**
    - Example: Used in 3D graphics and gaming to speed up rendering by organizing objects in space.
4. **Machine Learning (KNN Algorithm)**
    - Helps speed up the **k-Nearest Neighbors (KNN)** classifier by reducing search time.
5. **Robotics & Pathfinding**
    - Used in motion planning for robots to navigate around obstacles efficiently.

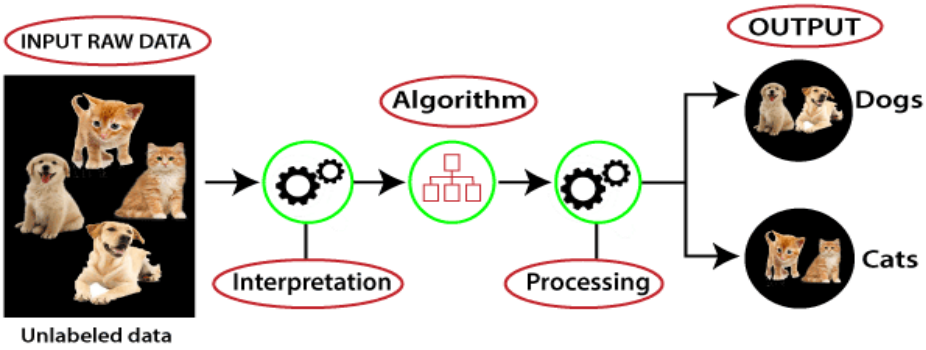
Why use a k-d tree?

- Faster searches than checking every point one by one (especially in large datasets).
- Organizes multi-dimensional data in a structured way.

UNSUPERVISED LEARNING

Unsupervised learning is a type of machine learning that works with data that has no labels or categories. The main goal is to find patterns and relationships in the data without any guidance.In this approach, the machine analyzes unorganized information and groups it based on similarities, patterns, or differences. Unlike supervised learning, there is **no teacher** or training involved. The machine must uncover hidden structures in the data on its own.

Example



Imagine you have a machine learning model trained on a large dataset of unlabeled images, containing both dogs and cats. The model has never seen an image of a dog or cat before, and it has no pre-existing labels or categories for these animals. Your task is to use unsupervised learning to identify the dogs and cats in a new, unseen image. suppose it is given an image having both dogs and cats which it has never seen. Thus, the machine has no idea about the features of dogs and cats so we can't categorize it as 'dogs and cats '. But it can categorize them according to their similarities, patterns, and differences, i.e., we can easily categorize the above

picture into two parts. The first may contain all pics having dogs in them and the second part may contain all pics having cats in them. Here you didn't learn anything before, which means no training data or examples.

It allows the model to work on its own to discover patterns and information that was previously undetected. It mainly deals with unlabeled 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.
- **Association**: An association rule is an unsupervised learning method which is used for finding the relationships between variables in the large database. It determines the set of items that occurs together in the dataset. Association rule makes marketing strategy more effective. Such as people who buy X item (suppose a bread) are also tend to purchase Y (Butter/Jam) item. A typical example of Association rule is Market Basket Analysis.

Advantages of Unsupervised Learning

- Unsupervised learning is used for more complex tasks as compared to supervised learning because, in unsupervised learning, we don't have labeled input data.
- Unsupervised learning is preferable as it is easy to get unlabeled data in comparison to labeled data.

Disadvantages of Unsupervised Learning

- Unsupervised learning is intrinsically more difficult than supervised learning as it does not have corresponding output.
- The result of the unsupervised learning algorithm might be less accurate as input data is not labeled, and algorithms do not know the exact output in advance.

K MEANS ALGORITHM

- K-means clustering is a popular unsupervised machine learning algorithm used for partitioning a dataset into a pre-defined number of clusters. The goal is to group similar data points together and discover underlying patterns or structures within the data.
- The first property of clusters states that the points within a cluster should be similar to each other. So, our aim here is to minimize the distance between the points within a cluster.
- There is an algorithm that tries to minimize the distance of the points in a cluster with their centroid – the k-means clustering technique.

- K-means is a centroid-based algorithm or a distance-based algorithm, where we calculate the distances to assign a point to a cluster. In K-Means, each cluster is associated with a centroid.
- The main objective of the K-Means algorithm is to minimize the sum of distances between the points and their respective cluster centroid.
- Optimization plays a crucial role in the k-means clustering algorithm. The goal of the optimization process is to find the best set of centroids that minimizes the sum of squared distances between each data point and its closest centroid.

How K-Means Clustering Works?

- **Initialization**: Start by randomly selecting K points from the dataset. These points will act as the initial cluster centroids.
- **Assignment:** For each data point in the dataset, calculate the distance between that point and each of the K centroids. Assign the data point to the cluster whose centroid is closest to it. This step effectively forms K clusters.
- **Update centroids:** Once all data points have been assigned to clusters, recalculate the centroids of the clusters by taking the mean of all data points assigned to each cluster.
- **Repeat:** Repeat steps 2 and 3 until convergence. Convergence occurs when the centroids no longer change significantly or when a specified number of iterations is reached.
- **Final Result:** Once convergence is achieved, the algorithm outputs the final cluster centroids and the assignment of each data point to a cluster.

Mathematical Representation

The objective of K-Means is to minimize the sum of squared differences between each point and its assigned cluster centroid:

$$J = \sum_{i=1}^{K} \sum_{x_j \in C_i} ||x_j - \mu_i||^2$$

Where:

- K = Number of clusters
- x_j = Data points
- μ_i = Centroid of cluster C_i

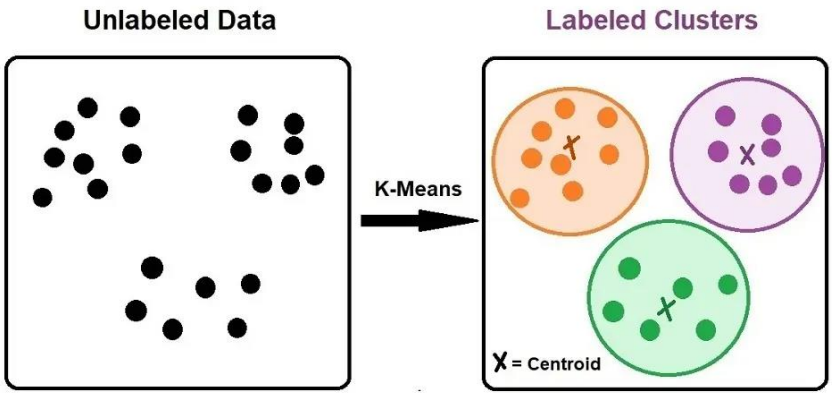
Choosing the Right K (Elbow Method)

- Plot the **Within-Cluster Sum of Squares (WCSS)** for different values of K.
- Look for an "elbow point," where the WCSS decrease slows down.

Objective of k means Clustering

The main objective of k-means clustering is to partition your data into a specific number (k) of groups, where data points within each group are similar and dissimilar to points in other groups. It achieves this by minimizing the distance between data points and their assigned cluster's center, called the centroid.

- **Grouping similar data points:** K-means aims to identify patterns in your data by grouping data points that share similar characteristics together. This allows you to discover underlying structures within the data.
- **Minimizing within-cluster distance:** The algorithm strives to make sure data points within a cluster are as close as possible to each other, as measured by a distance metric (usually Euclidean distance). This ensures tight-knit clusters with high cohesiveness.
- **Maximizing between-cluster distance:** Conversely, k-means also tries to maximize the separation between clusters. Ideally, data points from different clusters should be far apart, making the clusters distinct from each other.



Advantages of K-means

1. Simple and easy to implement: The k-means algorithm is easy to understand and implement, making it a popular choice for clustering tasks.
2. Fast and efficient: K-means is computationally efficient and can handle large datasets with high dimensionality.
3. Scalability: K-means can handle large datasets with many data points and can be easily scaled to handle even larger datasets.
4. Flexibility: K-means can be easily adapted to different applications and can be used with varying metrics of distance and initialization methods.

Disadvantages of K-Means

1. Sensitivity to initial centroids: K-means is sensitive to the initial selection of centroids and can converge to a suboptimal solution.
2. Requires specifying the number of clusters: The number of clusters k needs to be specified before running the algorithm, which can be challenging in some applications.
3. Sensitive to outliers: K-means is sensitive to outliers, which can have a significant impact on the resulting clusters.

Example:

No.	Height	Weight	Cluster
1	185	72	K1
2	170	56	K2
3	168	60	?
4	179	68	?

(*Note: Keep point 1 and 2 as centroids and label them as K1 & K2*)

Step 1:
Decide the centroid. So let's consider that point ① & ② are the centroids of the cluster K1 & K2.
K1 = (185, 72)
K2 = (170, 56)

Step 2:

Find the shortest distance between point ③ & ① and then point ③ & ②. First, take ③ & ①:

$$d = \sqrt{(168 - 185)^2 + (60 - 72)^2} = 20.80$$

Then calculate for ③ & ②:

$$d = \sqrt{(168 - 170)^2 + (60 - 56)^2} = 4.48$$ *(Minimum distance)*

Hence, point ③ belongs to K2 cluster.

Step 3:

Calculate the new centroid. As there is no new point added in K1, K1 remains the same.

In K2, a new point is added. Calculate the centroid of K2 by taking the mean:

$$K2 = \left(\frac{170 + 168}{2}, \frac{60 + 56}{2}\right)$$

$$K2 = (169, 58)$$

New centroid of K2 and update it.

Step 4:

Find the shortest distance between ④ & ① and ④ & updated K2.

For ④ & ①:

$d = \sqrt{(179 - 185)^2 + (68 - 72)^2} = 6.32$ *(minimum distance)*

For ④ & updated K2:

$d = \sqrt{(179 - 169)^2 + (68 - 58)^2} = 14.14$

As 6.32 is minimum distance, point ④ belongs to K1 cluster.

Step 5 (Repeat Step 3):

Repeat Step 3 by calculating the new centroid of K1 by doing its mean:

$$K1 = \left(\frac{185 + 179}{2}, \frac{72 + 68}{2}\right)$$

$$K1 = (182, 70)$$

This becomes the new centroid of K1 and update it.

Step 6:

Total clusters are K = 2.

K1 = {1, 4}
K2 = {2, 3}

No.	Height	Weight	Cluster
1	185	72	K1
2	170	56	K2
3	168	60	K2
4	179	68	K1