. **Machine Learning**

S.N.	Content
1	Introduction, Supervised learning algorithm: types of learning, application, Supervised learning: Linear Regression Model, Naive Bayes classifier Decision Tree, K nearest neighbor, Logistic Regression, Support Vector Machine, Random forest algorithm.
2	Unsupervised learning algorithm: Grouping unlabelled items using k-means clustering, Hierarchical Clustering, Probabilistic clustering, Association rule mining, Gaussian mixture model.
3	Feature extraction - Principal component analysis, Singular value decomposition. Feature selection – feature ranking and subset selection, introduction to deep learning, filter, wrapper and embedded methods, evaluating Machine Learning algorithms.
4	Semi supervised learning, Reinforcement learning: Markov decision process (MDP), Bellman equations, policy evaluation using Monte Carlo, Policy iteration and Value iteration, Q-Learning, State-Action-Reward-State-Action (SARSA), Model-based Reinforcement Learning.
5	Recommended system, Collaborative filtering, Content-based filtering Artificial neural network, Perceptron, Multilayer network, Backpropagation, Introduction to Deep learning

Course Code	Course Name	Course Outcome	Details
6CS4-02	**Machine Learning**	**CO1**	Have a good understanding of the fundamental issues and challenges of machine learning: data, model selection, model complexity, etc.**(K2)**
		CO2	Recognize and implement various ways of selecting suitable model parameters for different machine learning techniques**.(K3)**
		CO3	Compare and evaluate machine learning models based on mathematical analysis**.(K4)**
		CO4	Design and implement various machine learning algorithms in a range of real-world applications**.(K3)**

Text Book/References :

- Machine Learning. Tom Mitchell. First Edition, McGraw- Hill, 2013.
- Jason Bell, ―Machine learning – Hands on for Developers and Technical Professionals‖, First Edition, Wiley, 2014
- Ethem Alpaydin, ”Introduction to Machine Learning”, MIT Press, Prentice Hall of India, 3rd Edition2014.
- Mehryar Mohri, Afshin Rostamizadeh, Ameet Talwalkar ” Foundations of Machine Learning”, MIT
- Press,2012.
- Kevin P. Murphy ”Machine Learning: A Probabilistic Perspective”, The MIT Press, 2012
- Bishop-Pattern-Recognition-and-Machine-Learning-2006.

Introduction to Machine Learning

Machine Learning (ML) is the process of programming computers to improve performance using **example data or past experience**.

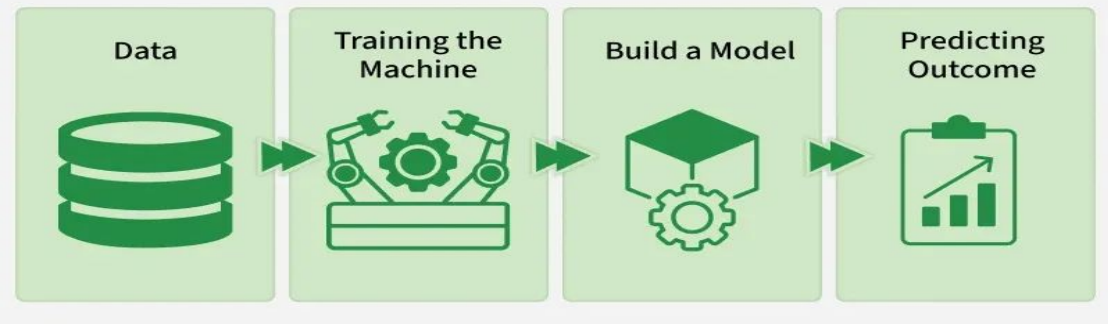
An ML system uses a **model** that is defined by a set of **parameters**.

Learning refers to the process of **optimizing these parameters** using training data.

The goal is to **optimize a performance criterion** (e.g., accuracy, error, cost).

ML models can be:

- **Predictive** – to make future predictions.
- **Descriptive** – to discover patterns and gain insights from data.
- **Both predictive and descriptive**.



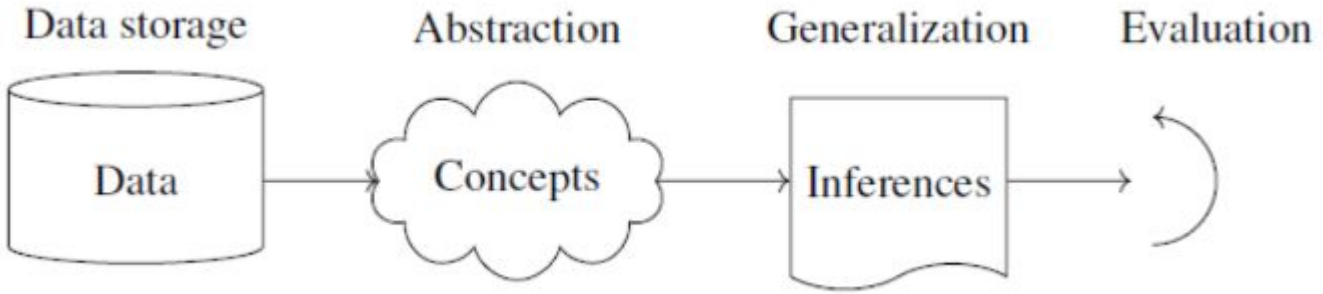
Definition of Learning : A computer program is said to **learn from experience (E)**

- With respect to a **task (T)**
- And a **performance measure (P)**
- If its performance on **T**, measured by **P**, **improves with experience E**

Example: Handwriting Recognition Learning Problem

- **Task (T):** Recognizing and classifying handwritten words from images
- **Performance Measure (P):** Percentage of words correctly classified
- **Training Experience (E):** A labeled dataset of handwritten word images

Fig1: Basic components of learning process



1. Data Storage

- Data storage is a **fundamental component** of the learning process.
- It enables storing and retrieving **large volumes of data** for reasoning and learning.

- **Humans:**
  - Store data in the **brain**
  - Retrieve information using **electrochemical signals**

- **Computers:**
  - Store data using **hard disks, flash memory, RAM**, etc.
  - Retrieve data via **cables, buses, and electronic signals**

2. Abstraction

- Abstraction is the process of **extracting knowledge** from stored data.
- It involves forming **general concepts** that represent the data as a whole.
- Knowledge creation includes:
  - Applying **existing models**
  - Developing **new models**
- **Training** refers to fitting a model to a dataset.
- After training, data is transformed into an **abstract representation** that summarizes the original information.

3. Generalization

- Generalization enables the use of learned knowledge for **future tasks**.
- These tasks are **similar but not identical** to previously seen data.
- The goal is to identify **relevant patterns and properties** that apply to new situations.
- Effective generalization allows a model to **perform well on unseen data**.

4. Evaluation

- Evaluation is the **final component** of the learning process.
- It measures the **usefulness and effectiveness** of the learned knowledge.
- Evaluation provides **feedback** on model performance.
- Common evaluation metrics include **accuracy, error rate, precision, recall**, etc.
- The feedback is used to **refine and improve** the overall learning process.
- Helps in validating whether learning objectives have been successfully achieved.

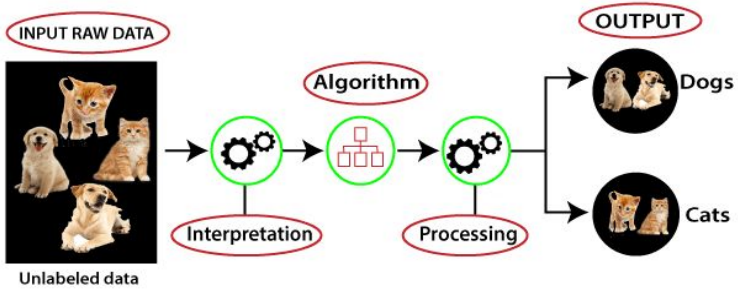
Types of Machine Learning Algorithms

1. Supervised Learning

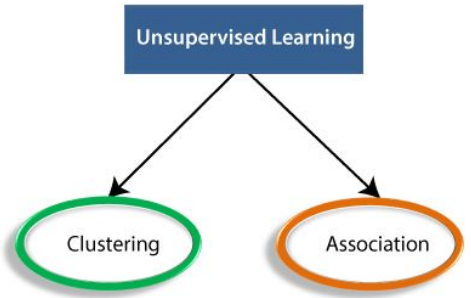
- Training data consists of **input–output pairs (labeled data)**.
- The algorithm learns a **function that maps inputs to outputs**.
- Also known as **learning from exemplars**.
- Goal: **Generalize** from training data to correctly predict unseen inputs.
- Each training example includes:
  - **Input object** (usually a feature vector)
  - **Correct output (target label/value)**
- Used for:
  - **Classification** (discrete output)
  - **Regression** (continuous output)
- Many supervised learning algorithms exist, each with **specific strengths and weaknesses**.
- **No single algorithm** works best for all supervised learning problems.

2. Unsupervised Learning

- Training data contains **no labeled outputs**.
- The algorithm identifies **patterns, similarities, or structures** in the data.
- Focuses on **discovering hidden relationships** among inputs.
- No predefined output values → **no direct performance evaluation**.
- Common statistical approach: **Density estimation**.
- Most common method: **Clustering**
  - Groups similar data points together
  - Used for **exploratory data analysis**
- Helps uncover **hidden patterns or groupings** in datasets.



Clustering



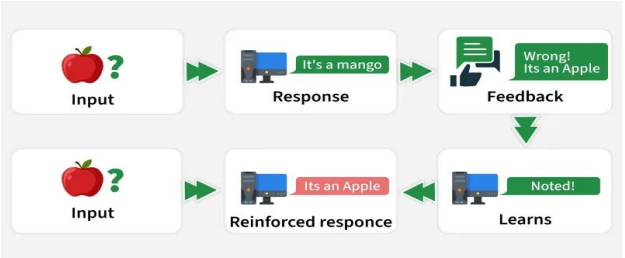
- It groups data objects into **clusters**
- Objects within the **same cluster are highly similar**
- Objects in **different clusters have little or no similarity**
- Similarity is measured using distance or similarity metrics
- Cluster analysis identifies **common patterns** in data
- Data objects are categorized based on the **presence or absence of common features**
- Common clustering algorithms include: K-Means, Hierarchical Clustering, DBSCAN

Association (Association Rule Mining)

- Association is an **unsupervised learning method**
- Used to discover **relationships between variables** in large datasets
- Identifies items that **frequently occur together**
- Rules are of the form: **X → Y**
- Example:Customers who buy **Bread (X)** also tend to buy **Butter or Jam (Y)**

3. Reinforcement Learning: Lies **between supervised and unsupervised learning**.

- The algorithm receives **feedback in the form of rewards or penalties**.
- It is told **whether an action is good or bad**, but not how to fix it.
- Learns by **trial and error** through interaction with the environment.
- Often called **learning with a critic**.
- Objective: **Maximize cumulative reward** over time.
- Actions may affect:
  - Immediate rewards
  - Future states and long-term rewards. Widely used in **robotics, game playing, and control systems**.



Regression Analysis

- A **statistical method** used to model the relationship between:
  - **Dependent (target) variable** and
  - **One or more independent (predictor) variables**
- Explains how the **dependent variable changes** when an independent variable changes, **while others are held constant**.
- Used to **predict continuous/real-valued outputs** such as:
  - Temperature, age, salary, price, sales, etc.

Key Terminologies in Regression Analysis

- **Dependent Variable (Target):**
  - The main variable to be **predicted or analyzed**
- **Independent Variable (Predictor):**
  - Variables that **influence or predict** the dependent variable
- **Outliers:**
  - Extremely high or low values compared to other observations
  - Can **negatively affect model accuracy** and should be handled carefully

Multicollinearity:

- Occurs when independent variables are **highly correlated with each other**
- Makes it difficult to identify the most influential predictor
- Should be avoided in regression models

Underfitting and Overfitting:

- **Overfitting:** Model performs well on training data but poorly on test data
- **Underfitting:** Model performs poorly on both training and test data

Why Do We Use Regression Analysis?

- Helps in **predicting future continuous values** accurately
- Widely used in **machine learning and data science** applications
- Common real-world applications include:
  - Weather forecasting
  - Sales and revenue prediction
  - Market trend analysis

Benefits of Regression Analysis

- Estimates the **relationship between target and predictors**
- Identifies **trends and patterns** in data
- Determines:
  - Most important factors
  - Least important factors
  - Impact of each predictor on the target variable

Regression algorithms example

- linear regression, kernel ridge regression (KRR), support vector regression (SVR), and Lasso

Linear Regression

- Linear regression is a **statistical regression technique** used for **predictive analysis**.
- It is one of the **simplest and most widely used** regression algorithms.
- Used to model the **relationship between continuous variables**.
- Commonly applied to **regression problems in machine learning**.

Working Principle

- Establishes a **linear relationship** between:
  - **Independent variable (X-axis)**
  - **Dependent variable (Y-axis)**
- If there is **one input variable**, it is called **Simple Linear Regression**.
- If there are **multiple input variables**, it is called **Multiple Linear Regression**.

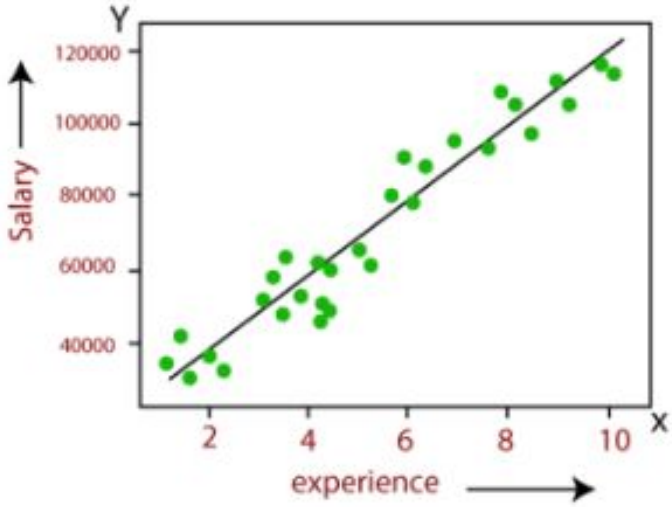
Mathematical Model:

$$Y=aX+b$$

- **Y** → Dependent variable (target)
- **X** → Independent variable (predictor)
- **a** → Slope (rate of change of Y with respect to X)
- **b** → Intercept (value of Y when X = 0)

Example

- Predicting **employee salary** based on **years of experience**
- Experience → Independent variable
- Salary → Dependent variable

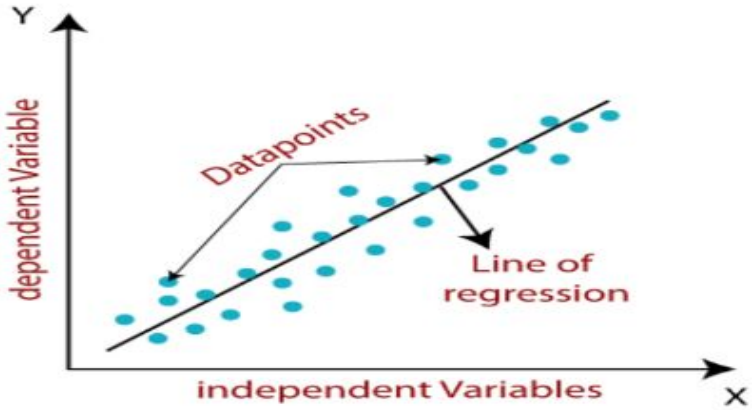


Applications of Linear Regression

- Trend analysis and **sales forecasting**
- **Salary prediction**
- **Real estate price estimation**
- **Estimated Time of Arrival (ETA)** prediction in traffic systems

Linear Regression

- Linear regression is one of the **simplest and most widely used** machine learning algorithms.
- It is a **statistical method** mainly used for **predictive analysis**.
- Used to predict **continuous / real-valued variables** such as:
  - Sales, salary, age, product price, etc.
- It models a **linear relationship** between:
  - **Dependent variable (Y)** and
  - **One or more independent variables (X)**
- The model explains **how the dependent variable changes** with changes in the independent variable(s).
- The relationship is represented by a **sloped straight line**.



Mathematical Representation of Linear Regression

- Linear regression can be expressed as:

$$y = a_0 + a_1 x + \varepsilon$$

Where:

- **y** → Dependent variable (target variable)
- **x** → Independent variable (predictor variable)
- **a0** → Intercept of the line
  - Provides an **additional degree of freedom**
  - Represents the value of y when x=0
- **a1**→ Linear regression coefficient (slope)
  - Acts as a **scale factor** for the input variable
  - Shows how much y changes with respect to x
- **ε**→ Random error term
  - Represents noise or unexplained variation in the data

Training Data

- The values of **x** and **y** are taken from the **training dataset**.
- These data points are used to **learn the best values of a0 and a1** for the linear regression model.

Types of Linear Regression

- **Simple Linear Regression**
  - Uses **one independent variable** to predict a numerical dependent variable.
  - Example: Predicting salary based on years of experience.

- **Multiple Linear Regression**
  - Uses **more than one independent variable** to predict a numerical dependent variable.
  - Example: Predicting house price based on area, number of bedrooms, and location.

Linear Regression Line

- The **regression line** shows the relationship between **dependent** and **independent variables**.
- **Types of relationship:**
  - **Positive Linear Relationship:** Both X and Y increase.
  - **Negative Linear Relationship:** X increases while Y decreases.

Finding the Best Fit Line

- Goal: Minimize the **error** between predicted and actual values.
- Different **coefficients (a0, a1)** produce different lines.
- **Best fit line:** Line with the **least error**.

Cost Function

- Determines the **best coefficients** for the regression line.
- Measures **how well the model predicts output** from input.
- Known as the **Mean Squared Error (MSE)** for Linear Regression:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - (a_1 x_i + a_0)\right)^2$$

Where:

- N = Total number of observations
- yi = Actual value
- $a_1 x_i + a_0$ = Predicted value

Residuals

- **Residual:** Distance between actual and predicted values.
- **High residuals** → Observed points far from regression line → Higher cost function.
- **Low residuals** → Observed points close to regression line → Lower cost function.

Simple Linear Regression (SLR)

- A regression algorithm that models the relationship between:
  - **One independent variable (X)** and
  - **One dependent variable (Y)**
- The relationship is **linear**, represented by a **sloped straight line**.
- The **dependent variable must be continuous/real-valued**.
- The **independent variable** may be **continuous or categorical**.

Objectives of Simple Linear Regression

- **Model the relationship** between two variables
  - Example: Income vs Expenditure, Experience vs Salary
- **Predict or forecast future values**
  - Example: Weather prediction, company revenue estimation

Simple Linear Regression Model: $$y = a_0 + a_1 x + \varepsilon$$

a0 → Intercept of the regression line (value of y when x=0)
a1 → Slope of the regression line

- Indicates whether the line is increasing or decreasing

ε → Error term (negligible for a good model)

Multiple Linear Regression (MLR) Equation:

- **Multiple Linear Regression models the linear relationship between a single continuous dependent variable and more than one independent variable.**

$$y = a_0 + a_1 x_1 + a_2 x_2 + a_3 x_3 + \cdots + a_n x_n + \varepsilon$$

Where:

- **y** → Dependent variable (target variable)
- **x1,x2,…,xn** → Independent variables (predictor variables)
- **a0** → Intercept of the regression model
- **a1,a2,…,an** → Regression coefficients (weights of predictors)
- **ε**→ Error term (random noise)

- An extension of **Simple Linear Regression**.
- Used when the dependent variable is influenced by **more than one independent variable**.

Models the linear relationship between:

- **One continuous dependent variable (Y)** and
- **Multiple independent variables (X_1, X_2, …, X□)**

- Each coefficient ai represents the **effect of the corresponding predictor xi** on the dependent variable.
- The model estimates how **multiple variables together influence** the target value.
- Used when the output depends on **more than one factor** (e.g., house price, CO_2 emission, sales prediction).

Example of MLR

- Predicting **CO_2 emissions** based on:
  - Engine size
  - Number of cylinders

Key Points of Multiple Linear Regression

- The **dependent variable (Y)** must be **continuous/real-valued**.
- Independent variables may be **continuous or categorical**.
- Each predictor must have a **linear relationship** with the dependent variable.
- The regression model fits a **hyperplane** in a **multidimensional feature space**.

Logistic Regression

Logistic Regression is a **supervised learning algorithm** used to solve **classification problems**, especially when the **dependent (output) variable is categorical**.

Key Characteristics

- Used for **binary or discrete outcomes** such as:
  - 0 / 1
  - Yes / No
  - True / False
  - Spam / Not Spam

- It is a **predictive analysis algorithm**
- Works on the **concept of probability**
- Although called “regression,” it is **different from linear regression** in its usage and output
- Logistic regression uses the **sigmoid (logistic) function** to map predicted values to probabilities.

Sigmoid Function Formula

$$f(x) = \frac{1}{1 + e^{-x}}$$

Where:

- f(x): Output value between **0 and 1**
- x: Input to the function (linear combination of features)
- e: Base of the natural logarithm

- A **threshold value** (commonly **0.5**) is used for classification:

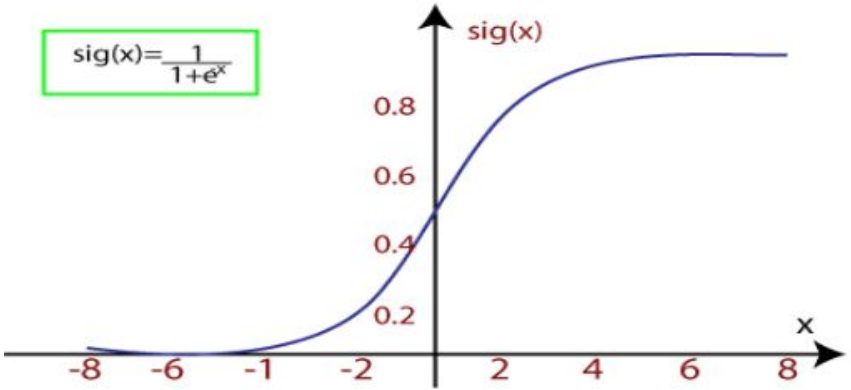
  - If $f(x) \geq 0.5 \rightarrow$ Output = **1**
  - If $f(x) < 0.5 \rightarrow$ Output = **0**

This converts probability values into **class labels**.

Produces an **S-shaped curve**

As input increases, output approaches **1**

As input decreases, output approaches **0**



Types of Logistic Regression

1. **Binary Logistic Regression**
    - Two classes
    - Examples:
        - Pass / Fail
        - 0 / 1
2. **Multinomial Logistic Regression**
    - More than two classes (no order)
    - Examples:
        - Cat / Dog / Lion
3. **Ordinal Logistic Regression**
    - Classes with an inherent order
    - Examples:
        - Low / Medium / High

Probabilistic Models

- Probabilistic models form the **third major family** of machine learning algorithms.
- Unlike distance-based (e.g., k-NN) or logical models, they use **probability theory** for classification.
- Features and target variables are treated as **random variables**.
- These models explicitly **represent and handle uncertainty** in data.

Types of Probabilistic Models

1. Predictive Probabilistic Models

- Based on **conditional probability distribution** P(Y∣X)P(Y \mid X)P(Y∣X).
- Predict the target variable Y given the input features X.
- Focus on **direct prediction** rather than data generation.

2. Generative Probabilistic Models

- Estimate the **joint probability distribution** P(X,Y).
- From the joint distribution, **conditional and marginal probabilities** can be derived.
- Capable of **generating new data samples along with labels**.
- Learn the **relationship between variables** and use it to infer unseen data.

Key Characteristics

- Model **uncertainty and variability** in data.
- Suitable when data contains **noise or incomplete information**.
- Can infer new instances once the probability relationships are learned.

Example

- **Naïve Bayes** is a widely used **probabilistic classifier**.

Bayes Rule

- Bayes rule is used to compute **conditional probability**.
- It describes the probability of an event **A**, given that **B** has already occurred.

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)}$$

Naïve Bayes Algorithm

- Naïve Bayes is a **probabilistic classification algorithm**.
- It is based on the concept of **conditional probability**.
- The algorithm evaluates **evidence (features)** to determine the **likelihood of each class**.
- Each data instance is assigned the class with the **highest posterior probability**.
- Assumes that features are **conditionally independent** given the class (naïve assumption).
- It is named as "Naive" because it assumes the presence of one feature does not affect other features. The "Bayes" part of the name refers to its basis in Bayes' Theorem.

Key Idea of Conditional Probability

- Conditional probability measures the chance of an event **occurring given another event has already happened**.
- In Naïve Bayes, this means estimating the probability of a **class label given observed features**.

Working Principle

- Compute the probability of each class using Bayes rule.
- Compare probabilities for all classes.
- Assign the label corresponding to the **maximum probability**.

Naïve Bayes: Step-by-Step

Step 1: Identify the **class labels** (e.g., Spam, Ham).

Step 2: Calculate **prior probabilities** for each class.

Step 3: Count feature occurrences for each class from the training data.

Step 4: Compute **likelihood probabilities** P(feature∣class)P(\text{feature} \mid \text{class})P(feature∣class) using Bayes rule (with smoothing if needed).

Step 5: Calculate the **posterior probability** for each class:

$$P(\text{class} \mid \text{features}) \propto P(\text{class}) \prod P(\text{feature} \mid \text{class})$$

Step 6: Assign the input to the **class with the highest posterior probability**.

Example of the Naïve Bayes Classifier: Dataset Attributes

- **Outlook**: Sunny, Overcast, Rainy
- **Temperature**: Hot, Mild, Cool
- **Humidity**: High, Normal
- **Windy**: True, False
- **Class (Play)**: Yes / No **(Weather / Play Tennis Dataset)**

The weather data, with counts and probabilities

outlook			temperature			humidity			windy		play	
	yes	no		yes	no		yes	no		yes	no	yes	no
sunny	2	3	hot	2	2	high	3	4	false	6	2	9	5
overcast	4	0	mild	4	2	normal	6	1	true	3	3		
rainy	3	2	cool	3	1								
sunny	2/9	3/5	hot	2/9	2/5	high	3/9	4/5	false	6/9	2/5	9/14	5/14
overcast	4/9	0/5	mild	4/9	2/5	normal	6/9	1/5	true	3/9	3/5		
rainy	3/9	2/5	cool	3/9	1/5								

A new day

outlook	temperature	humidity	windy	play
sunny	cool	high	true	?

Step 1: Class Distribution (Prior Probabilities)

Total instances = **14**

- Play = **Yes** → 9
- Play = **No** → 5

- P(Yes)=9/14, P(No)=5/ 14

Step 2: Conditional Probabilities (from Dataset)

For Play = Yes

- P(Sunny | Yes) = 2 / 9, P(Cool | Yes) = 3 / 9 , P(High | Yes) = 3 / 9, P(True | Yes) = 3 / 9

For Play = No

- P(Sunny | No) = 3 / 5, P(Cool | No) = 1 / 5 , P(High | No) = 4 / 5, P(True | No) = 3 / 5

Step 3: New Day to Predict

Class = **?**

Outlook	Temperature	Humidity	Windy
Sunny	Cool	High	True

Step 4: Likelihood Calculation

Likelihood of Yes

$$P(\text{Yes}) \times P(\text{Sunny}|\text{Yes}) \times P(\text{Cool}|\text{Yes}) \times P(\text{High}|\text{Yes}) \times P(\text{True}|\text{Yes})$$

$$= \frac{9}{14} \times \frac{2}{9} \times \frac{3}{9} \times \frac{3}{9} \times \frac{3}{9}$$

$$= 0.0053$$

Likelihood of No

$$P(\text{No}) \times P(\text{Sunny}|\text{No}) \times P(\text{Cool}|\text{No}) \times P(\text{High}|\text{No}) \times P(\text{True}|\text{No})$$

$$= \frac{5}{14} \times \frac{3}{5} \times \frac{1}{5} \times \frac{4}{5} \times \frac{3}{5}$$

$$= 0.0206$$

Step 5: Final Prediction

- Likelihood(Yes) = **0.0053**
- Likelihood(No) = **0.0206**

Therefore, the prediction is No

(The class with the **higher likelihood** is chosen)

1. Gaussian Naïve Bayes

- Used when features are **continuous (real-valued)**
- Assumes feature values follow a **Gaussian (Normal) distribution**
- Gaussian distribution is **bell-shaped** and **symmetric about the mean**
- Defined by **mean (μ)** and **standard deviation (σ).** (Ex: Suitable for numerical data such as: Height, weight, temperature , Medical and sensor data)

2. Multinomial Naïve Bayes

- Used when features represent **frequency or counts**
- Commonly applied in **text classification**
- Feature values indicate **how many times a word appears** in a document
- Works well with **bag-of-words** and **TF-IDF** models. (Ex: Spam detection, Document classification, Sentiment analysis)

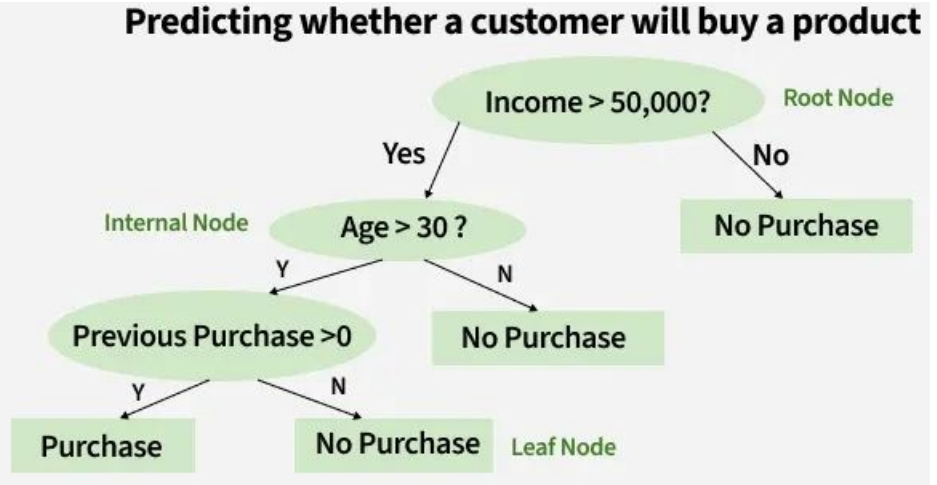
3. Bernoulli Naïve Bayes

- Used for **binary-valued features**
- Feature values indicate **presence (1) or absence (0)** of a term
- Frequency of terms is **not considered**
- More suitable for **short text documents,** (Ex: Email spam filtering,Text and document classification)

Decision Tree

- A Decision Tree is a **supervised learning algorithm**
- Used for both **classification** and **regression** tasks
- Has a **hierarchical tree structure**
- Works like a **flowchart** to make decisions step by step

Structure of a Decision Tree



- **Root Node**
  - Topmost node
  - Represents the entire dataset
- **Internal Nodes**
  - Represent **attribute (feature) tests**
  - Decide how the data is split
- **Branches**
  - Represent **outcomes of attribute tests**
  - Correspond to different feature values
- **Leaf Nodes**
  - Represent **final decisions or predictions**
  - Each leaf node corresponds to a **class label** (or value in regression)

Working of Decision Tree

- The dataset is split based on **feature values**
- Goal is to create **pure subsets**
- In an ideal split:
  - All data points in a subset belong to the **same class**
- Splitting continues until:
  - Maximum purity is achieved, or
  - Stopping criteria are met

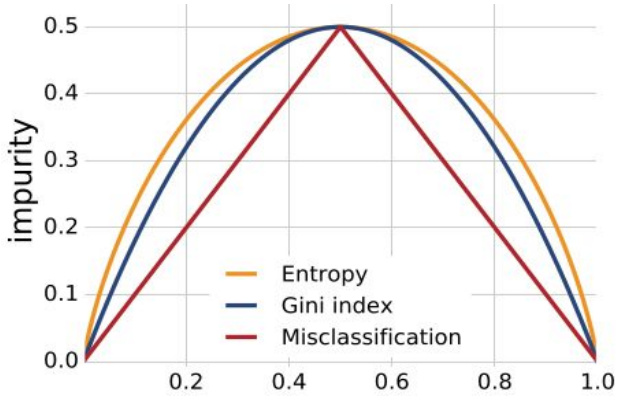
Attribute Selection Measures

To decide the best feature for splitting, decision trees use **attribute selection measures**.
The two most popular measures are: 1.**Information Gain** **2. Gini Index**

1. Information Gain

Best split = largest information gain

- Measures **reduction in uncertainty (entropy)** after splitting
- Tells how useful a feature is in classifying data
- A good feature: Creates **clearer and purer groups**
- The feature with the **highest Information Gain** is selected
- Commonly used in **ID3 and C4.5 algorithms**



Let **S** be a set of training instances

Let **A** be an attribute

$$\text{Gain}(S, A) = \text{Entropy}(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} \cdot \text{Entropy}(S_v)$$

Let **Values(A)** be the set of all possible values of attribute **A**

Let $\mathbf{S_v}$ be the subset of **S** for which attribute **A** has value **v**

$$\text{Entropy}(S) = -\sum_{i=1}^{n} p_i \log_2(p_i)$$

Where: pi is the probability of class **i**
n is the number of classes

Entropy(S): Measures the impurity or uncertainty in the entire dataset **S**
Entropy(S_v) : Measures the impurity of subset $\mathbf{S_v}$ after splitting on attribute **A**
$|S_v|$ / |S|: Represents the proportion (weight) of subset $\mathbf{S_v}$ in the dataset
The summation term represents the **expected entropy after splitting** on attribute **A**
Information Gain is the **reduction in entropy** after the split

-Entropy = 0 → no uncertainty
-Max entropy → completely mixed classes

2. Gini Index

$$Gini = 1 - \sum_{i=1}^{n} p_i^2$$

pi = proportion of class *i* in the node

- Measures **impurity** in a dataset
- Indicates how often a randomly chosen element would be **incorrectly classified**
- Lower Gini Index means:
  - Better purity
  - Better split
- Attribute with the **lowest Gini Index** is preferred

Gini = 0 → perfectly pure (only one class)

- if we have a group of people where all bought the product (100% "Yes") the Gini Index is 0 indicate perfect purity. But if the group has an equal mix of "Yes" and "No" the Gini Index would be 0.5 show high impurity or uncertainty.

Example

- Node has 80% Purchase, 20% No Purchase

$$\text{Gini} = 1 - (0.8^2 + 0.2^2) = 0.32$$

✅ Lower = better

Types of Decision Trees

There are **two main types of Decision Trees**:

1. Classification Trees

- Used when the **output (decision variable) is categorical**
- Outcome values are **discrete**
- Examples: Yes / No, Fit / Unfit, Pass / Fail
- The decision tree predicts a **class label**
- The example discussed earlier (Fit / Unfit) is a **classification tree**

2. Regression Trees

Best split = lowest weighted MSE after splitting

- Used when the **output (decision variable) is continuous**
- Outcome values are **numerical**
- Examples: 123, Salary prediction, Temperature prediction
- The decision tree predicts a **real-valued number**

- **Mean Squared Error (MSE)**

For regression, we don't have classes — we predict numbers.

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

- $\bar{y}$ = mean value in the node
- Measures how spread out target values are

ID3 Decision Tree Algorithm

- **ID3** stands for **Iterative Dichotomiser 3**
- It is a **greedy algorithm**
- Used to build **classification decision trees**
- It selects attributes based on **Information Gain**
- The attribute with the **highest Information Gain** is chosen for splitting

- Decision Trees can be used for both classification and regression problems. The ID3 algorithm is a widely used method for constructing classification trees by selecting attributes that maximize Information Gain.

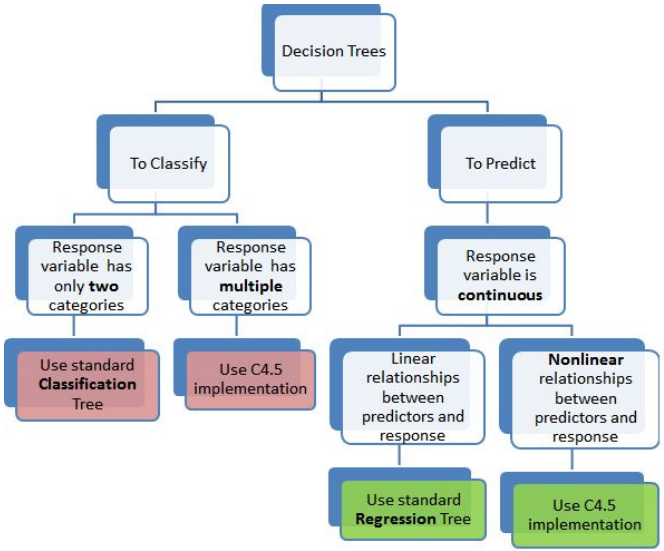
How a Decision Tree Is Trained:

Step 1: Initialize the Root Node Use the **entire training dataset**

- Compute the impurity of the node:
  - **Gini impurity** or **Entropy** for classification
  - **Mean Squared Error (MSE)** for regression

Step 2: Evaluate All Possible Splits For each feature in the dataset:

- Consider all possible split thresholds
- Partition the data into child nodes
- Compute the impurity of each child node
- Calculate the **weighted impurity** of the split



Step 3: Select the Optimal Split

- Choose the split that **maximizes impurity reduction**
  - (or **minimizes MSE** for regression trees)

Step 4: Grow the Tree Recursively

- Apply the same splitting process to each child node
- Each node operates only on the subset of data it receives

Step 5: Apply Stopping Criteria Stop splitting when **any** of the following conditions is met:

- The node is **pure** (contains only one class)
- The **maximum tree depth** is reached
- The number of samples in the node is below a minimum threshold
- The reduction in impurity is smaller than a predefined value

Step 6: Assign Leaf Node Predictions

- **Classification:** assign the **majority class** of samples in the node
- **Regression:** assign the **mean target value** of samples in the node

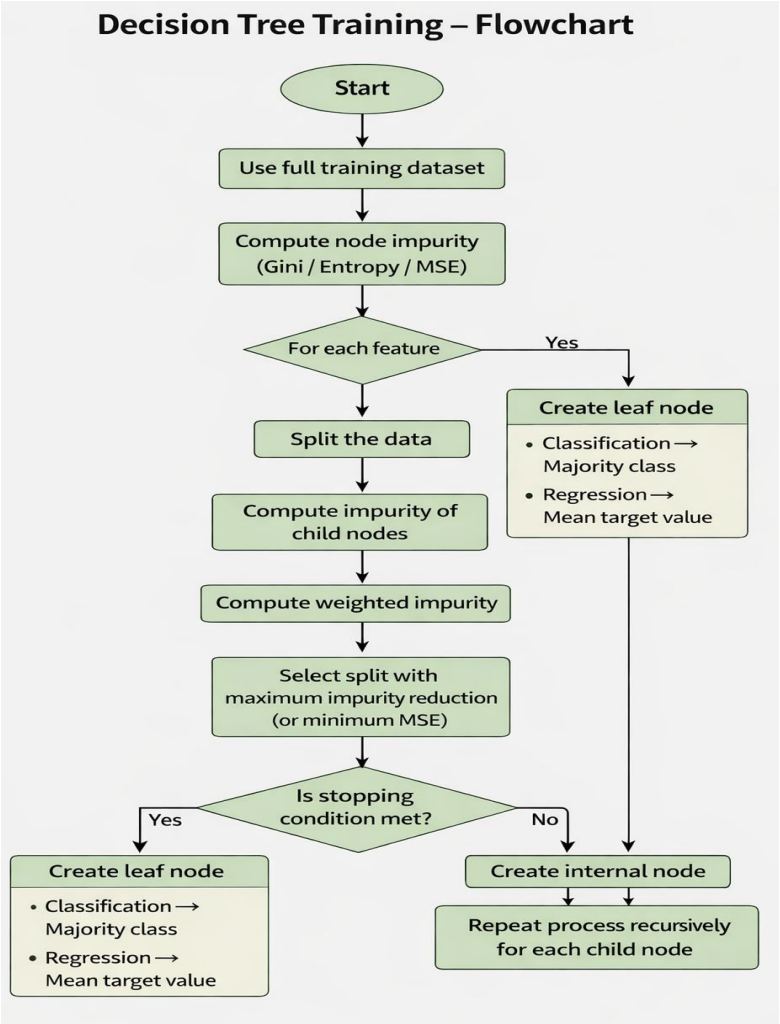
Step 5: Apply Stopping Criteria

Stop splitting when **any** of the following conditions is met:

- The node is **pure** (contains only one class)
- The **maximum tree depth** is reached
- The number of samples in the node is below a minimum threshold
- The reduction in impurity is smaller than a predefined value

Step 6: Assign Leaf Node Predictions

- **Classification:** assign the **majority class** of samples in the node
- **Regression:** assign the **mean target value** of samples in the node



Fully Worked Numerical Example

(Classification Tree using Gini)

Dataset:

Customer	Income (₹00s)	N
1	30	No
2	40	No
3	50	Yes
4	60	Yes
5	70	Yes

Target: Purchase (Yes / No)

1 Step 1: Compute Gini at Root Node:

$$\text{Gini}_{root} = 1 - \left(\frac{3}{5}\right)^2 - \left(\frac{2}{5}\right)^2 = 0.48$$

Class counts: Yes = 3

No = 2

$p(\text{Yes}) = \frac{3}{5}$, $p(\text{No}) = \frac{2}{5}$

$$\text{Gini}_{root} = 1 - 1\left(-\frac{3}{5}\right)^2 = \frac{1}{25} - \frac{9}{25} - \frac{4}{25} = \frac{12}{25}$$

$= 0.48$ $= 0.48$

2 Step 2: Try Possible Splits:

Split 1: Income ≤ 45 **Weighted Gini:** 0.00 **Gini Reduction:** 0.48

Left node: 30, 40

Total = 5

$$\text{Gini}_{left} = 1 - 1 - \frac{3}{5} = 1 - \frac{9}{25} - \frac{4}{25} = \frac{12}{25} = 0.48$$

Split	Income	Right node	Purchase
<= 45	< 30	50, 60	< 0
<= 55	< 2.67	60, 70	< 0

✓ **Split 1** (Income ≤ 55)

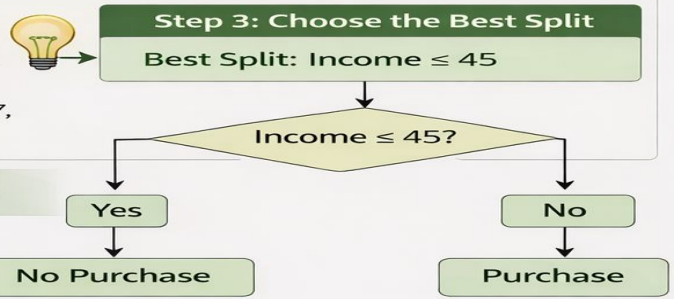
✗ **Split 2** (Weighted Gini = 0.00, Gini Reduction = 0.213)

4 Step 3: Choose the Best Split

Best Split: Income ≤ 45

✓ Split 1 (Weighted Gini = 0.00, Gini Reduction = 0.213)

✗ Split 2 (Weighted Gini = 0.267, Gini Reduction = 0.213)



4 Step 4: Create Leaf Nodes:

Final Decision Tree:

The Play Golf dataset contains 14 instances with 4 input attributes (Outlook, Temperature, Humidity, Wind) and 1 target attribute (Play Golf).

Day	Outlook	Temperature	Humidity	Wind	Play Golf
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

4.1 Calculate Overall Entropy

For the entire dataset:

- Total instances: 14
- Yes (positive): 9
- No (negative): 5

$H(S) = -(9/14) \times \log_2(9/14) - (5/14) \times \log_2(5/14)$

$H(S) = -(0.643) \times (-0.638) - (0.357) \times (-1.485)$

$H(S) = 0.410 + 0.530$

$H(S) = 0.940$

4.2 Information Gain for Outlook

Outlook has three values: Sunny, Overcast, and Rain.

Outlook = Sunny (5 instances)

- Yes: 2, No: 3
- $H(Sunny) = -(2/5) \times \log_2(2/5) - (3/5) \times \log_2(3/5) = 0.971$

Outlook = Overcast (4 instances)

- Yes: 4, No: 0
- $H(Overcast) = -(4/4) \times \log_2(4/4) - (0/4) \times \log_2(0/4) = 0.000$

Outlook = Rain (5 instances)

- Yes: 3, No: 2
- $H(Rain) = -(3/5) \times \log_2(3/5) - (2/5) \times \log_2(2/5) = 0.971$

Weighted Average Entropy:

$H(S|Outlook) = (5/14) \times 0.971 + (4/14) \times 0.000 + (5/14) \times 0.971$

$H(S|Outlook) = 0.347 + 0.000 + 0.347 = 0.694$

$IG(S, Outlook) = 0.940 - 0.694 = 0.246$

4.3 Information Gain for Temperature

Temperature has three values: Hot, Mild, and Cool.

Temperature = Hot (4 instances)

- Yes: 2, No: 2
- $H(Hot) = -(2/4) \times \log_2(2/4) - (2/4) \times \log_2(2/4) = 1.000$

Temperature = Mild (6 instances)

- Yes: 4, No: 2
- $H(Mild) = -(4/6) \times \log_2(4/6) - (2/6) \times \log_2(2/6) = 0.918$

Temperature = Cool (4 instances)

- Yes: 3, No: 1
- $H(Cool) = -(3/4) \times \log_2(3/4) - (1/4) \times \log_2(1/4) = 0.811$

Weighted Average Entropy:

$H(S|Temperature) = (4/14) \times 1.000 + (6/14) \times 0.918 + (4/14) \times 0.811$

$H(S|Temperature) = 0.286 + 0.393 + 0.232 = 0.911$

$IG(S, Temperature) = 0.940 - 0.911 = 0.029$

4.4 Information Gain for Humidity

Humidity has two values: High and Normal.

Humidity = High (7 instances)

- Yes: 3, No: 4
- $H(High) = -(3/7) \times \log_2(3/7) - (4/7) \times \log_2(4/7) = 0.985$

Humidity = Normal (7 instances)

- Yes: 6, No: 1
- $H(Normal) = -(6/7) \times \log_2(6/7) - (1/7) \times \log_2(1/7) = 0.592$

Weighted Average Entropy:

$H(S|Humidity) = (7/14) \times 0.985 + (7/14) \times 0.592$

$H(S|Humidity) = 0.493 + 0.296 = 0.789$

$IG(S, Humidity) = 0.940 - 0.789 = 0.151$

4.5 Information Gain for Wind

Wind has two values: Weak and Strong.

Wind = Weak (8 instances)

- Yes: 6, No: 2
- $H(Weak) = -(6/8) \times \log_2(6/8) - (2/8) \times \log_2(2/8) = 0.811$

Wind = Strong (6 instances)

- Yes: 3, No: 3
- $H(Strong) = -(3/6) \times \log_2(3/6) - (3/6) \times \log_2(3/6) = 1.000$

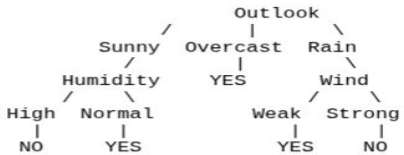
Weighted Average Entropy:

$H(S|Wind) = (8/14) \times 0.811 + (6/14) \times 1.000$

$H(S|Wind) = 0.463 + 0.429 = 0.892$

IG(S, Wind) = 0.940 - 0.892 = 0.048

Attribute	Information Gain
Outlook	**0.246** ✅ (Highest)
Humidity	0.151
Wind	0.048
Temperature	0.029



7. Decision Rules

The decision tree generates the following classification rules:

1. **IF Outlook = Overcast THEN Play Golf = YES**
2. **IF Outlook = Sunny AND Humidity = High THEN Play Golf = NO**
3. **IF Outlook = Sunny AND Humidity = Normal THEN Play Golf = YES**
4. **IF Outlook = Rain AND Wind = Weak THEN Play Golf = YES**
5. **IF Outlook = Rain AND Wind = Strong THEN Play Golf = NO**

5.1 Root Node Selection

Since Outlook has the highest information gain (0.246), it becomes the root node of the decision tree.

5.2 Branch: Outlook = Overcast

All instances where Outlook = Overcast have Play Golf = Yes (4 out of 4). This branch is pure and becomes a leaf node with prediction YES.

5.3 Branch: Outlook = Sunny

For the Sunny subset (5 instances: 2 Yes, 3 No), we need to split further. Calculate information gain for remaining attributes:

- IG(Sunny, Humidity) = 0.971 - 0.000 = 0.971
- IG(Sunny, Temperature) = 0.971 - 0.951 = 0.020
- IG(Sunny, Wind) = 0.971 - 0.951 = 0.020

Humidity has the highest information gain (0.971), so we split on Humidity:

- If Humidity = High → Play Golf = NO (3 instances, all No)
- If Humidity = Normal → Play Golf = YES (2 instances, all Yes)

5.4 Branch: Outlook = Rain

For the Rain subset (5 instances: 3 Yes, 2 No), we need to split further. Calculate information gain for remaining attributes:

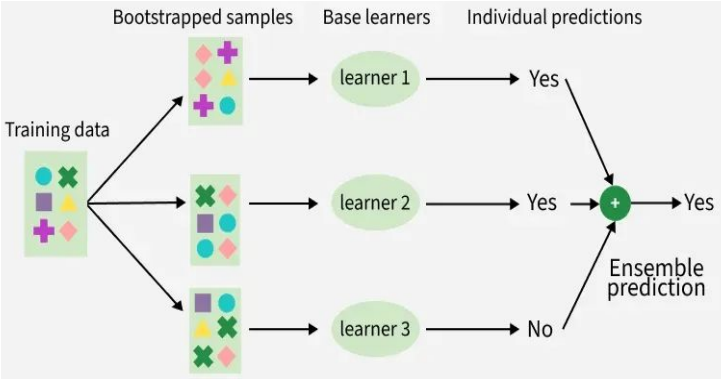
- IG(Rain, Wind) = 0.971 - 0.000 = 0.971
- IG(Rain, Humidity) = 0.971 - 0.951 = 0.020
- IG(Rain, Temperature) = 0.971 - 0.951 = 0.020

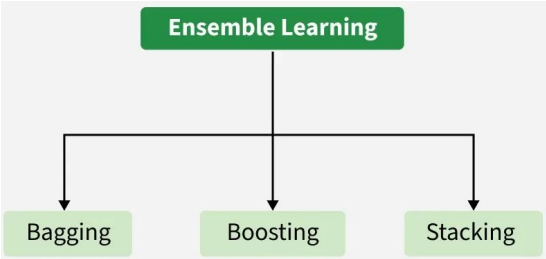
Wind has the highest information gain (0.971), so we split on Wind:

- If Wind = Weak → Play Golf = YES (3 instances, all Yes)
- If Wind = Strong → Play Golf = NO (2 instances, all No)

Ensemble Learning

- Ensemble Learning is a machine learning technique where multiple models (weak learners) are combined to produce a stronger and more accurate model.





Method	Training Style	Main Focus	Error Reduction
Bagging	Parallel	Reduce Variance	Handles Overfitting
Boosting	Sequential	Reduce Bias	Improves Weak Models
Stacking	Hierarchical	Optimal Combination	Improves Overall Performance

Use **Bagging** when model overfits (high variance)

Use **Boosting** when model underfits (high bias)

Use **Stacking** when combining different strong models

Ensemble Learning combines multiple weak models to build a powerful predictive system.

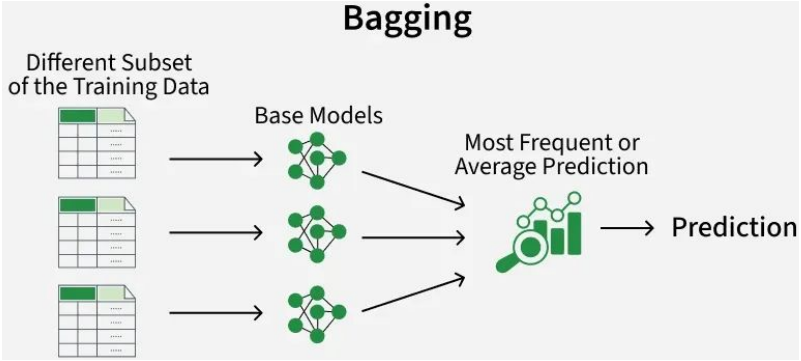
- Bagging → Parallel → Reduce Variance
- Boosting → Sequential → Reduce Bias
- Stacking → Meta-Learning → Best Combination

Ensemble methods often outperform single machine learning models in real-world problems.

Bagging (Bootstrap Aggregating)

Multiple models are trained **independently** on different random subsets of the dataset.

- Sampling is done **with replacement**
- Models are trained in **parallel**
- Predictions are combined using:
    - Majority Voting (Classification)
    - Averaging (Regression)



Reduce **variance** and prevent **overfitting**

🧩 Steps of Bagging

1. Bootstrap Sampling
2. Train base learners independently
3. Aggregate predictions
4. Use OOB samples for evaluation
5. Produce final prediction

📌 Example

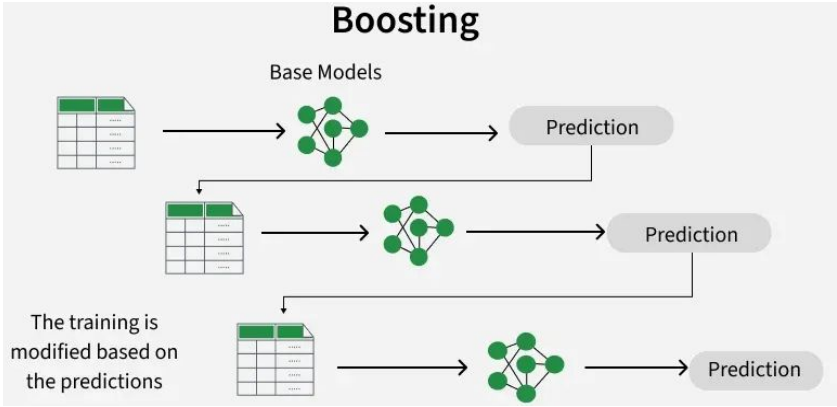
Random Forest

Boosting

Models are trained **sequentially**, where each new model focuses on correcting previous errors.

- Weak learners trained one after another
- Misclassified samples receive higher weight
- Final output is a weighted combination

Reduce **bias** and improve accuracy



🧩 Steps of Boosting

1. Initialize equal weights
2. Train first weak learner
3. Increase weight of misclassified samples
4. Train next learner
5. Combine all models

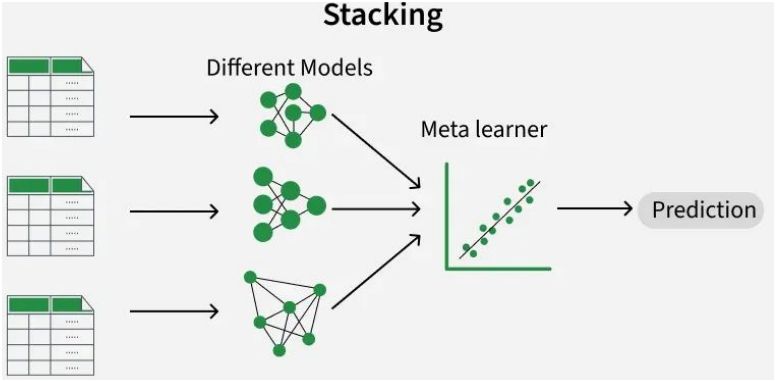
📌 Example

AdaBoost, Gradient Boosting, XGBoost

AdaBoost

Training:
For all $\{x^t, r^t\}_{t=1}^N \in \mathcal{X}$, initialize $p_1^t = 1/N$
For all base-learners $j = 1, \ldots, L$
Randomly draw $\mathcal{X}_j$ from $\mathcal{X}$ with probabilities p_j^t
Train d_j using $\mathcal{X}_j$
For each (x^t, r^t), calculate $y_j^t \leftarrow d_j(x^t)$
Calculate error rate: $\epsilon_j \leftarrow \sum_t p_j^t \cdot 1(y_j^t \neq r^t)$
If $\epsilon_j > 1/2$, then $L \leftarrow j - 1$; stop
$\beta_j \leftarrow \epsilon_j / (1 - \epsilon_j)$
For each (x^t, r^t), decrease probabilities if correct:
If $y_j^t = r^t$, then $p_{j+1}^t \leftarrow \beta_j p_j^t$ Else $p_{j+1}^t \leftarrow p_j^t$
Normalize probabilities:
$Z_j \leftarrow \sum_t p_{j+1}^t$; $p_{j+1}^t \leftarrow p_{j+1}^t / Z_j$
Testing:
Given x, calculate $d_j(x), j = 1, \ldots, L$
Calculate class outputs, $i = 1, \ldots, K$:
$y_i = \sum_{j=1}^L \left(\log \frac{1}{\beta_j}\right) d_{ji}(x)$

Stacking (Stacked Generalization)



Different types of models are trained and their outputs are used as inputs for a **meta-model**.

- Base models make predictions
- Meta-model learns how to best combine them

Achieve optimal model combination

Ensemble Techniques

Random Forest (Bagging):

- Multiple decision trees
- Uses bootstrapped samples
- Reduces overfitting

🌿 **Random Subspace:** Uses random feature subsets, Increases diversity
🚀 **AdaBoost (Boosting) :** Focuses on difficult samples, Uses weighted voting
⚡ **Gradient Boosting (GBM):** Sequential tree building, Minimizes loss function
🔥 **XGBoost:** Optimized gradient boosting, Regularization + pruning, High performance
🐱 **CatBoost:** Handles categorical features efficiently, Reduces overfitting automatically,

Random Forest

Random Forest is a powerful **ensemble learning algorithm** that uses **multiple decision trees** to make better and more accurate predictions.

Instead of relying on a single decision tree, Random Forest combines predictions from many trees using:

- ✅ **Majority Voting** (for Classification)
- ✅ **Averaging** (for Regression)

This improves accuracy and reduces errors.

- **Random** → Uses random samples of data and random features
- **Forest** → Collection of many decision trees

It belongs to the **Bagging (Bootstrap Aggregating)** family of ensemble methods.

Reduces Overfitting
✔ Reduces Variance
✔ Handles Noise Better
✔ Improves Generalization

Using randomness in both **data sampling** and **feature selection** makes trees diverse, and diversity improves ensemble performance.

Random Forest is an ensemble learning technique that:

- Builds multiple decision trees
- Uses random data samples and random features
- Combines predictions using voting or averaging
- Reduces variance and overfitting
- Provides strong, reliable predictions

1. **Random Data Sampling**
2. **Random Feature Selection**
3. **Build Many Decision Trees**
4. **Each Tree Makes Individual Predictions**
5. **Combine Results (Voting/Averaging)**
6. **Final Prediction**

Predict whether a student will Pass or Fail based on three features:

- **Study Hours** - Number of hours spent studying per week
- **Attendance** - Percentage of classes attended
- **Previous Marks** - Score from previous examination

Assume we have historical data of 100 students with their features and outcomes (Pass/Fail).

Step 1: Create Many Decision Trees (Bootstrap Sampling)

- Random Forest doesn't build just one decision tree - it builds many trees (typically hundreds or thousands). Each tree is trained on a different random sample of the data.

Bootstrap Sampling Process: Bootstrap sampling is a technique where we randomly select samples from the dataset with replacement. This means:

- The same student can appear multiple times in a sample
- Some students may not appear in a particular sample at all
- Each tree sees a slightly different version of the training data

📌 Example: Building 5 Trees

For this example, suppose we decide to build 5 decision trees:

- **Tree 1:** Trained on a random sample of 100 students (with replacement)

- **Tree 2:** Trained on a different random sample of 100 students
- **Tree 3:** Trained on yet another random sample of 100 students
- **Tree 4:** Trained on a different random sample of 100 students
- **Tree 5:** Trained on another random sample of 100 students

Important: *Even though all trees are trained on 100 students each, the students in each sample are different due to random sampling with replacement. This diversity is crucial for Random Forest's success.*

Step 2: Pick Random Features at Each Split

Fe While building each tree, Random Forest introduces another layer of randomness: at every node split, the algorithm randomly selects only a subset of features to consider.

Unlike traditional decision trees that consider all features at each split, Random Forest:

- Randomly selects a subset of features at each node
- Does NOT use all features at once
- The typical number of features to consider is √n (square root of total features)

📌 **Example: Feature Selection:** In our problem, we have 3 features: 1. Study Hours 2. Attendance 3.Previous Marks

At any given split point, the tree might randomly choose only 2 out of 3 features. For example:

- **Option 1:** (Study Hours, Attendance)
- **Option 2:** (Attendance, Previous Marks)
- **Option 3:** (Study Hours, Previous Marks) **Why This Matters:** *This random feature selection makes each tree different from the others, reducing correlation between trees and improving the overall ensemble performance.*

📌 Example: New Student Prediction:

A new student arrives with the following characteristics:

- **Study Hours:** 5 hours per week , **Attendance:** 80%, **Previous Marks:** 60

Each of our 5 trees examines this student and makes a prediction:

Tree	Prediction
Tree 1	Pass
Tree 2	Pass
Tree 3	Fail
Tree 4	Pass
Tree 5	Fail

Step 4: Combine the Predictions

The final step is to combine the predictions from all trees. The method of combination depends on whether we're doing classification or regression.

📌 For Classification: Majority Voting

From our example:

In classification problems, Random Forest uses majority voting. The class that receives the most votes from the individual trees becomes the final prediction.

- **Fail:** 2 votes (Trees 3, 5)

Final Output: Pass

In regression problems, Random Forest takes the average of all tree predictions.

Example: Predicting Salary

Suppose instead of Pass/Fail, we were predicting a student's expected salary after graduation. Each tree might predict:

Tree	Salary Prediction
Tree 1	$50,000
Tree 2	$55,000
Tree 3	$52,000
Tree 4	$48,000
Tree 5	$53,000

Calculation:

```
Average = (50,000 + 55,000 + 52,000 + 48,000 + 53,000) ÷ 5
Average = 258,000 ÷ 5
Average = 51,600
```

Final Predicted Salary: $51,600

Key parameters to optimize:

- **n_estimators:** Number of trees in the forest
- **max_features:** Number of features to consider at each split
- **max_depth:** Maximum depth of each tree
- **min_samples_split:** Minimum samples required to split a node
- **min_samples_leaf:** Minimum samples required at a leaf node

Example of random forest with majority voting

Below is a clear step-by-step outline of the Random Forest algorithm.

Step 1: Build Multiple Decision Trees

The Random Forest algorithm generates many decision trees.
Each tree is constructed as follows:

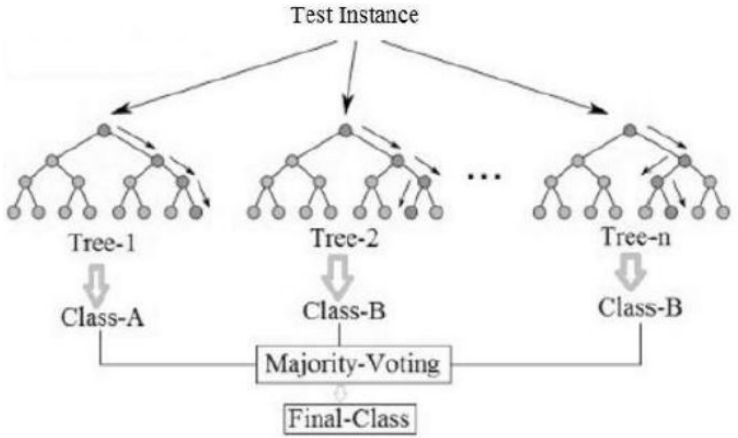
a) Bootstrap Sampling

- Suppose the training dataset contains **N examples**.
- Randomly select **N examples with replacement** from the original dataset.
- This new sample becomes the training set for building one decision tree.
- Because sampling is done with replacement:
    - Some examples may appear multiple times.
    - Some examples may not appear at all.

b) Random Feature Selection at Each Node

- Assume there are **M total input variables (features)**.
- At each node of the tree:
    - Randomly select **m features** out of the M features.
    - Choose the **best split** only among these m selected features.
- The value of **m remains constant** for all trees in the forest.

This ensures diversity among trees and reduces correlation between them.



c) Grow the Tree Fully

- Each tree is grown to its **maximum possible depth**.
- No pruning is performed.
- Trees are allowed to fully develop.

Step 2: Make Predictions (Voting Mechanism)

To classify a new input:

1. Send the input vector down **each tree** in the forest.
2. Each tree produces its own classification.
3. Every tree “votes” for a class.
4. The forest selects the class that receives the **majority of votes**.

For Regression Problems

- Each tree outputs a numeric prediction.
- The final prediction is the **average** of all tree outputs.

K-Nearest Neighbor (KNN) Algorithm

- K-Nearest Neighbors (KNN) is a supervised machine learning algorithm used for classification and regression problems.

- It predicts the output for a new data point by analyzing the K closest training samples in the feature space
- KNN works on the idea that similar data points exist close to each other.
- No explicit training phase is required; all computation happens during prediction.

Steps of KNN Algorithm

1. Choose the number of neighbors **K**.
2. Calculate the **distance** between the test data point and all training data points.
3. Select the **K nearest neighbors** based on the smallest distances.
4. **For Classification**:
    - Assign the class label that occurs most frequently among the K neighbors (majority voting).
5. **For Regression**:
    - Compute the **average (mean)** of the target values of the K neighbors.

Choice of K :

- Small K (e.g., K = 1): Sensitive to noise, High variance
- Large K: More stable predictions, May smooth out important patterns
- Optimal K is often chosen using cross-validation.

- Non-parametric: Does not assume any underlying data distribution.
- Instance-based (Lazy Learning): Stores all training data and makes decisions at prediction time.
- Unlike eager learners, KNN does not build a predictive model during the training phase.
- Instead, it stores the entire training dataset in memory.

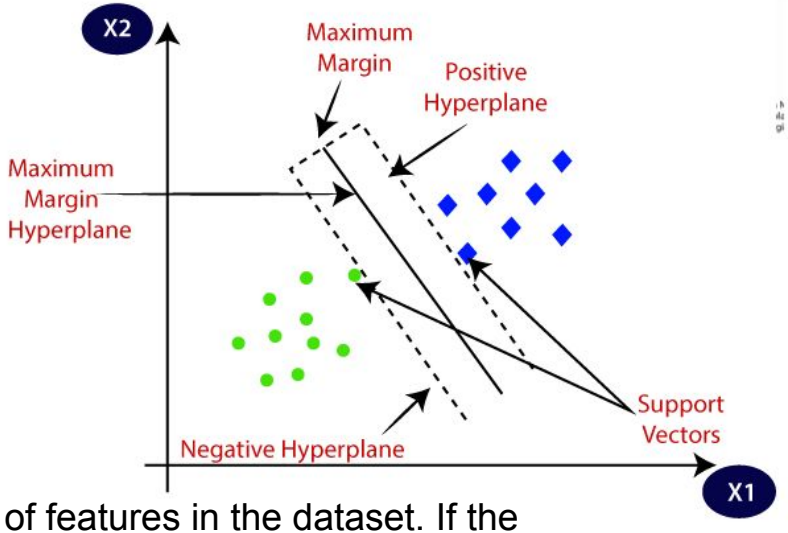
Choice of k in K-Nearest Neighbors (KNN)

- In the K-Nearest Neighbors (KNN) algorithm, k is a user-defined parameter.
- It represents the number of nearest data points (neighbors) considered while making a prediction.
- The algorithm uses these k neighbors to:
- -Decide the majority class in classification
- -Compute the average value in regression

Support Vector Machine

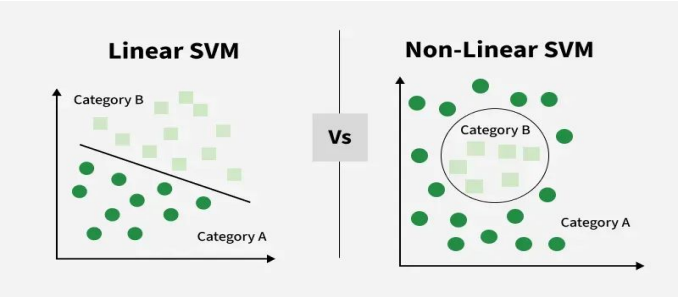
- Support Vector Machine (SVM) is a popular Supervised Learning algorithm.
- It is used for both Classification and Regression problems.
- SVM is primarily used for Classification in Machine Learning.
- The main goal of SVM is to create the best decision boundary.
- This boundary separates data points of different classes in an n-dimensional space.
- The decision boundary helps in correctly classifying new data points in the future.
- The best decision boundary created by SVM is called a Hyperplane.
  In:
  2D space → hyperplane is a line
  3D space → hyperplane is a plane
  Higher dimensions → hyperplane is a subspace
- SVM selects certain extreme data points that are closest to the hyperplane.
- These data points are called Support Vectors. Support vectors directly influence the position and orientation of the hyperplane.

- The algorithm relies on support vectors to construct the hyperplane.
- Since these vectors “support” or define the boundary, the algorithm is named Support Vector Machine (SVM).
- there are two different categories that are classified using a decision boundary or hyperplane: Figure
- In an n-dimensional space, there can be multiple lines or decision boundaries that separate the classes.
- SVM aims to find the best possible decision boundary to classify the data points accurately.
- This best decision boundary is called the Hyperplane



- The dimension of the hyperplane depends on the number of features in the dataset. If the dataset has:
  - 2 features → the hyperplane is a straight line
  - 3 features → the hyperplane is a 2-dimensional plane
  - More than 3 features → the hyperplane exists in higher-dimensional space

- SVM always selects the hyperplane with the maximum margin.
- Margin is defined as the maximum distance between the nearest data points of different classes.
- A larger margin leads to:
  1. Better class separation
  2. Improved generalization on unseen data
- The data points or vectors that are closest to the hyperplane are called Support Vectors.
- These vectors directly affect the position and orientation of the hyperplane.
- Removing or changing a support vector changes the hyperplane.

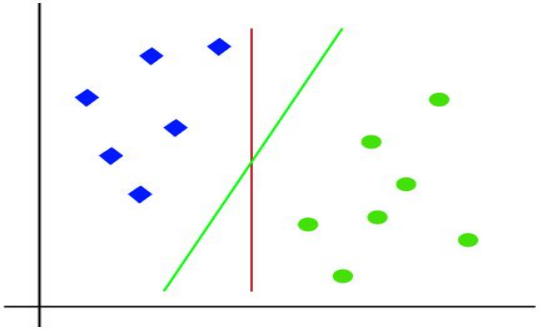


SVM can be broadly classified into two types:

1. Linear SVM

- Linear SVM is used for linearly separable data.
- If a dataset can be classified into two classes using a single straight line, it is called linearly separable data.
- The classifier used for such data is called a Linear SVM classifier.

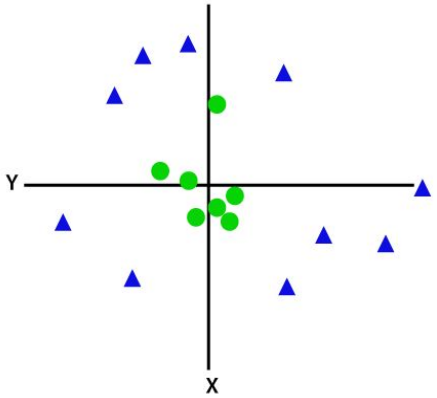
- Consider a dataset with two class labels: Green and Blue.
- The dataset contains two features, namely x_1 and x_2.
- Each data point is represented as a pair (x_1, x_2).
- The objective is to classify each data point into Green or Blue class.



- Since the data exists in a 2-dimensional space, a straight line can be used to separate the two classes. Such data is called linearly separable data.
- However, there can be multiple straight lines that separate the two classes.
- Among all possible separating lines, SVM selects the best line or decision boundary.
- This best decision boundary is called the Hyperplane.
- SVM identifies the closest data points from both classes to the hyperplane.
- These closest data points are called Support Vectors.
- The distance between the support vectors and the hyperplane is known as the Margin.

- The main goal of SVM is to maximize the margin.
- The hyperplane that has the maximum margin is called the Optimal Hyperplane.
- A larger margin results in better classification and improved generalization.

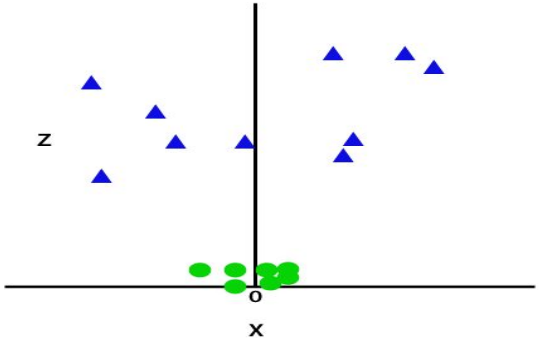
2. Non-Linear SVM

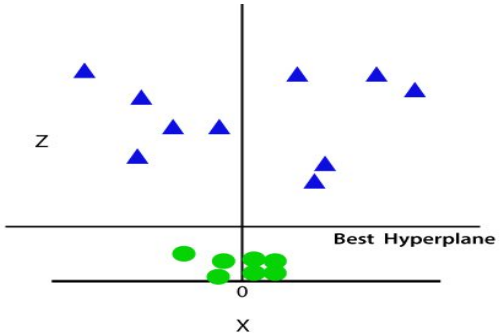


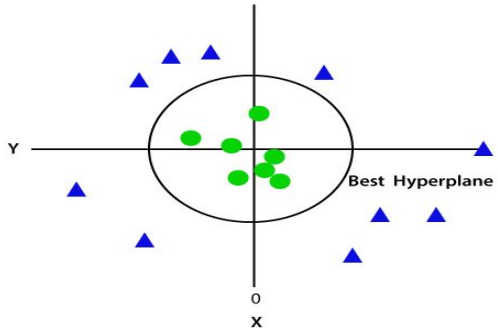
- Non-Linear SVM is used for non-linearly separable data.
- If a dataset cannot be classified using a straight line, it is called non-linear data.
- The classifier used for such data is called a Non-Linear SVM classifier.
- For linearly arranged data, classes can be separated using a single straight line.
- For non-linear data, it is not possible to draw a single straight line to separate the classes.
- Hence, a different approach is required to classify such data.

❖ To separate non-linear data, SVM adds an extra dimension.
❖ Initially, data is represented using two dimensions (x and y).
❖ For non-linear separation, a third dimension (z) is introduced.

- The third dimension is calculated using the equation: $z = x^2 + y^2$
- By adding this new dimension, the non-linear data becomes linearly separable in higher-dimensional space.
- After transformation, the data points are represented in 3-D space.
- SVM separates the data using a plane (hyperplane) in this higher dimension.
- The separating plane appears parallel to the x-axis in 3-D space.
- When the data is projected back to 2-D space with z = 1:
  The separating boundary becomes a circle (circumference).
- Hence, for non-linear data, SVM can produce circular or curved decision boundaries.







SVM Kernels

- A kernel transforms the input data space into a required form.
- SVM uses a technique called the Kernel Trick.
- The kernel trick:
  - ❖ Takes low-dimensional input space
  - ❖ Transforms it into a higher-dimensional space
- In simple words, kernels convert non-separable problems into separable problems by adding more dimensions.
- Kernels make SVM: More powerful, More flexible, More accurate
- SVM uses different kernel functions depending on the nature of data.
- One of the commonly used kernels is the Linear Kernel.

$$K(x, x_i) = \sum (x \cdot x_i)$$

- Linear Kernel is used when the data is linearly separable.
- It works as a dot product between two observations.

- The kernel computes the product of two vectors, say x and x_i.
- This product is calculated as the sum of the multiplication of corresponding input values.
- Hence, the linear kernel measures the similarity between two vectors.

Kernel Type	Formula	Best Used When	Decision Boundary	Key Features
Linear	$K(x, x_i) = x \cdot x_i$	Data is linearly separable	Straight line / plane	Fast, simple, low computation
Polynomial	$K(x, x_i) = (x \cdot x_i + c)^d$	Data has polynomial relationship	Curved	Degree d controls complexity
RBF (Gaussian)	$K(x, x_i) = \exp\left(-\frac{\|x - x_i\|^2}{2\sigma^2}\right)$	Highly non-linear data	Circular / complex	Most powerful, flexible, widely used
Sigmoid	$K(x, x_i) = \tanh(\alpha(x \cdot x_i) + c)$	Neural-network-like behavior	Non-linear	Similar to NN activation function

SVM Example: Consider a binary classification problem with two classes:

- Class labels: +1 and −1

Training dataset consists of:

- Feature vectors: X
- Corresponding labels: Y

- The equation of a linear hyperplane is: $w^T x + b = 0$

  Where:

  - $w \rightarrow$ **Weight vector** (normal/perpendicular to the hyperplane)
  - $b \rightarrow$ **Bias term** (offset from the origin)
  - $x \rightarrow$ Input feature vector
- The distance of a data point x_i from the hyperplane is given by:

$$d_i = \frac{|w^T x_i + b|}{\|w\|}$$

Where:

- $\|w\| \rightarrow$ **Euclidean norm** of vector w
- This distance determines how far the point lies from the decision boundary

Linear SVM Classifier (Prediction Rule)

- The predicted class label $\hat{y}$ is:

$$\hat{y} = \begin{cases} +1, & \text{if } w^T x + b \geq 0 \\ -1, & \text{if } w^T x + b < 0 \end{cases}$$

- Classification depends on **which side of the hyperplane** the point lies.

Optimization Problem for Hard Margin SVM

For linearly separable data, SVM aims to:

- Maximize the margin
- Correctly classify all data points

Objective Function

$$\min_{w,b} \frac{1}{2} ||w||^2$$

- Minimizing $||w||^2$ results in **maximum margin**.

Constraints

$$y_i(w^T x_i + b) \geq 1 \quad \text{for } i = 1, 2, \ldots, m$$

Where:

- y_i → Class label (+1 or −1)
- x_i → Feature vector of the i^{th} data point
- m → Number of training samples
- This constraint ensures:
  - Correct classification
  - Data points lie **outside the margin**

Soft Margin SVM (For Non-Separable Data)

- When data contains **noise or outliers**, perfect separation is not possible.
- SVM introduces **slack variables** ζi\zeta_iζi.

Modified Objective Function

$$\min_{w,b} \left(\frac{1}{2} \|w\|^2 + C \sum_{i=1}^{m} \zeta_i \right)$$

Where:

- C → **Regularization parameter**
- ζ_i → Slack variables (margin violation)

Constraints for Soft Margin

$$y_i(w^T x_i + b) \geq 1 - \zeta_i$$

$$\zeta_i \geq 0 \quad \text{for } i = 1, 2, \ldots, m$$

- Larger C → Less misclassification, smaller margin
- Smaller C → More tolerance, larger margin

SVM Decision Boundary (Dual Form)

- After solving the **dual optimization problem**, the decision function becomes:

$$w = \sum_{i=1}^{m} \alpha_i t_i K(x_i, x) + b$$

Where:

- α_i → Lagrange multipliers
- t_i → Class labels
- $K(x_i, x)$ → Kernel function
- Only **support vectors** have non-zero α_i

Bias Term Calculation

- For support vectors:

$$t_i(w^T x_i - b) = 1$$

- Hence, bias term:

$$b = w^T x_i - t_i$$

- Bias is calculated using **any support vector**.

Advantages of Support Vector Machine (SVM)

- High-Dimensional Performance

SVM performs well in high-dimensional spaces, making it suitable for image classification and gene expression analysis.

- Non-Linear Capability

By using kernel functions such as RBF and Polynomial, SVM effectively handles non-linear relationships.

- Outlier Resilience

The soft margin mechanism allows SVM to tolerate outliers, improving robustness in spam detection and anomaly detection.

- Binary and Multiclass Support

SVM is effective for both binary classification and multiclass classification, widely used in text classification tasks.

- Memory Efficient

SVM relies only on support vectors, making it more memory efficient compared to many other machine learning algorithms.

Disadvantages of Support Vector Machine (SVM)

- Slow Training

Training SVM can be slow for large datasets, which affects performance in data mining applications.

- Parameter Tuning Difficulty

Choosing the appropriate kernel function and tuning parameters such as C requires careful experimentation.

- Noise Sensitivity

SVM may perform poorly on noisy datasets or when classes significantly overlap.

- Limited Interpretability

The decision boundary in higher dimensions is complex, making SVM less interpretable than simpler models.

- Feature Scaling Sensitivity

Proper feature scaling is necessary; otherwise, SVM performance can degrade significantly.