Machine Learning Handbook

Concepts | Intuition | Mathematics
Real-World Applications

A comprehensive guide to understanding therinciples and techniques of Machine Learning

From foundational ideas to advanced algorithen—designed for students, developers, and data enthusiast, This handboak focuses on clarity, depth, and conceptual und understanding—without code, but rich in mathematicanduu, industry relevance, irgance, and structured learning

Vishnu Kumar D S

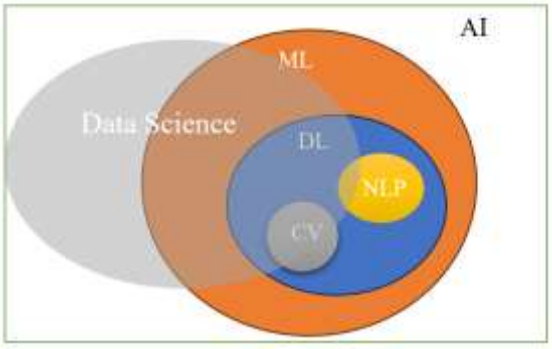
A theory-first resource for mastering Machine Learning fundamentals

Table of Contents

Everyone's talking about Machine Learning — but what exactly is it?

You will Know About it in This Document?

Before going to Machine learning we will understand Difference between ML, DL, AI, Data Science.



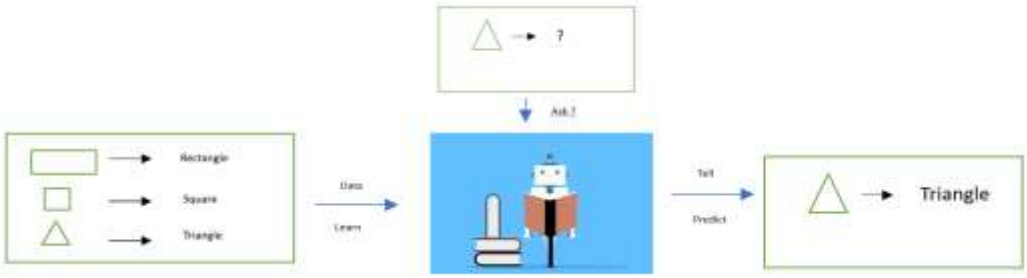
Element	What It Is (One Line)
AI	Making machines think and act smart.
ML	Teaching machines to learn from data.
DL	ML using deep neural networks.
Data Science	Finding insights from data.
NLP	Understanding and generating human language.
CV	Teaching machines to see and interpret images.

Machine Learning:

Machine Learning is a way to teach Computer to learn from Experience as like Human do.

Instead of giving a list of instructions to the computer we will give the data and examples, so then Computer look into the data and it will understand data and finds the pattern, learn how to make decisions on its own.

Ex: Machine Learning is like teaching a child. Show it enough examples, correct its mistakes, and soon it will start figuring things out on its own.



Why we use Machine Learning?

- It helps computers learn on their own
- It works well with big data
- It finds hidden patterns
- It powers smart technology

Where is Machine Learning used?

Machine Learning is used **almost everywhere** today. Google Search-learn from your data and recommend you, **Voice Assistants** – Siri,Alexa,Google Assistant Understand and respond using machine learning, It is used in all fields Now.

***Classification of Machine* Learning**

- Supervised learning
    1. Regression
    2. Classification
- Unsupervised learning
    1. Clustering
    2. Dimensional Reduction
    3. Anomaly Detection
- Reinforcement learning
    1. Value-Based – Learns the value of actions (e.g., Q-learning).
    2. Policy-Based – Learns a policy for actions directly.
    3. Model-Based – Builds a model of the environment.
- Semi-Supervised learning

1. Supervised learning:

Supervised learning uses labeled data to train the model, meaning the input comes with the correct output.

Example: Email spam detection (email → spam or not spam).

2. Unsupervised learning:

Unsupervised learning works with data that has no labels, and the model tries to find hidden patterns or groupings.

Example: Customer segmentation, grouping similar customers.

3. Reinforcement learning:

Reinforcement learning is based on a system where an agent learns by interacting with an environment and receiving rewards or penalties.

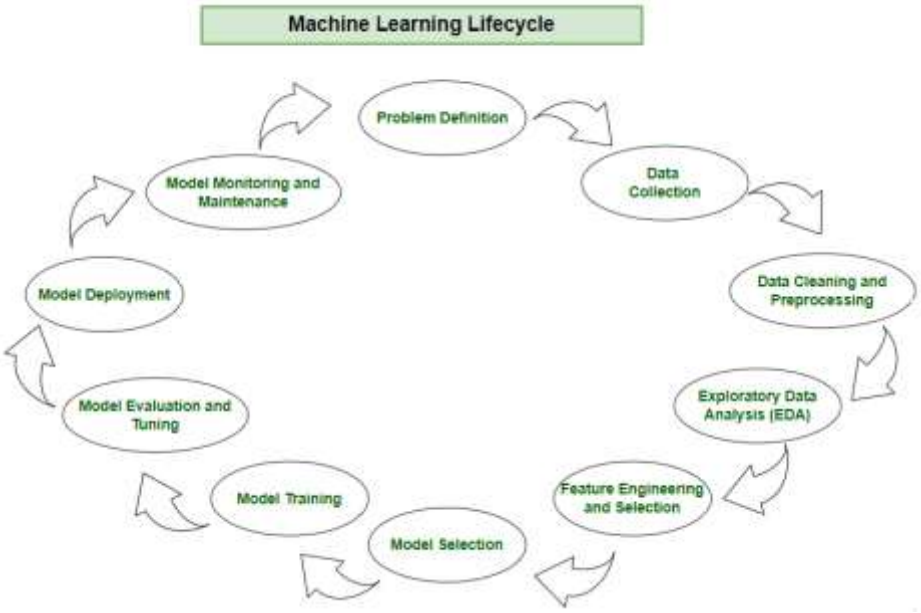
Example: A robot learning to walk or a game AI improving by playing.

4. Semi-Supervised Learning:

Semi-supervised learning Empty is the combination of both supervised and unsupervised, it uses a small amount of labeled data and a large amount of unlabeled data to improve learning accuracy.

Example: Training a model with a few labeled medical scans and many unlabeled ones.

Machine Learning Life Cycle



Problem Statements: It defends the business objective that ML model aim to solve

- Increasing sales
- Reducing churn
- Targeting the right audience
- Recommending movies from user data

Data Collection: Collecting data for the relevant problem statement from different sources like databases and websites or from Survey, forms.

Data Cleaning and Processing: So, after collecting the mess raw data We need to organize that raw data into a meaningful order like in rows and columns as tables. Filling or removing missing Fielding or remove, missing or duplicate.

Exploratory data analysis (EDA): Here we will understand the data and find the missing values like nan and replace them by using some techniques like mean and median, mode. Handling outlier or removing them and checking the data set whether it is balanced or not and making the data set balance using resampling technique.

Feature engineering and feature selection: After understanding the data we will find out the relationship between the data columns of each feature. If the data set is Imbalance, we will balance those data set using some techniques like SMOTE. And selecting the features which are very important and the features which are irrelevant to data set or we can drop them.

Model Selection: After selecting the important features, we will select different machine learning algorithms

Model Training: We will train the data on the Machine learning algorithm. Before training we will split the data set into two parts one is training and another one is testing. So as training will be 70% data and testing will be 30% (Or Training - 80% and Testing – 20%)

Model evaluation and tuning: Here we will test the machine learning model which we trained before on the testing data then we will evaluate using performance metrics, we will perform hyperparameter tuning and find the best algorithm or model which fits our data and give best accurate results

Model deployment: After performing all the above steps now it's time to release our machine learning model to production environment for the users. We can create a pickle file of the model that gives the best accuracy, and deploy it by creating an end-to-end pipeline. We can also use Flask, a popular Python framework, for web deployment to serve the model to end users.

Model Monitoring and Automation: Model monitoring is a critical part of the machine learning lifecycle that ensures your deployed models continue to perform well over time. Once a model is live, it interacts with real-world data that may differ significantly from the training data. Without monitoring, a model could silently degrade in performance, leading to poor business outcomes.

Here we will learn one by one of machine learning algorithms:

Supervised learning: Supervised learning works on labeled data, where both input features and corresponding target outputs are provided. The goal is to learn a mapping function from inputs to outputs, which can be used to make predictions on unseen data.
1. Regression: Predict continuous/numeric values (e.g., price, temperature).
2. Classification: Predict **categorical labels** or class memberships (e.g., spam or not spam).

REGRESSION ALGORITHMS	**CLASSIFICATION ALGORITHMS**	**ALGORITHMS Works on both REGRESSION & CLASSIFICATION**
Linear regression	Logistic Regression	Decision tree
Polynomial regression	Naive Bayes	SVM
Ridge Regression (L2 Regularization)		KNN
Lasso Regression (L1 Regularization)		Random Forest(Bagging)
Elastic Net Regression		Ada Boost (Boosting)
Multivariate Regression		Xgboost (Boosting) Gradient boosting (Boosting)

Unsupervised learning: Unsupervised learning works on unlabeled data, meaning the model is not given any output labels. The goal is to find patterns, structures, or groupings in the data.
1. Clustering
2. Dimensionality Reduction

CLUSTERING ALGORITHMS	DIMENSIONALITY REDUCTION ALGORITHMS
K means Clustering	PCA
DBSCAN Clustering	T-SNE
Hierarchical Clustering	UMAP

SUPERVISED LEARNING

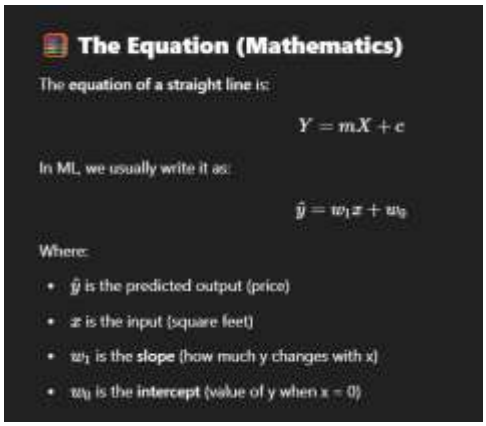
REGRESSION ALGORITHMS

1.Linear Regression: It is a type of supervised machine learning algorithm that computes the linear relationship between a dependent variable and one or more independent variable.
Goal: Finding the best fit line or linear line which helps for prediction

Simple Intuition: Imagine you're a data analyst at a real estate company. You want to predict the price of a house based on its square footage**.**

Square Feet (X)	Price (Y)
1000	₹50L
1500	₹75L
2000	₹100L

Now, you want a model that can **predict price for 1800 sq ft**.
Linear Regression fits a **straight line** that best explains this relationship.



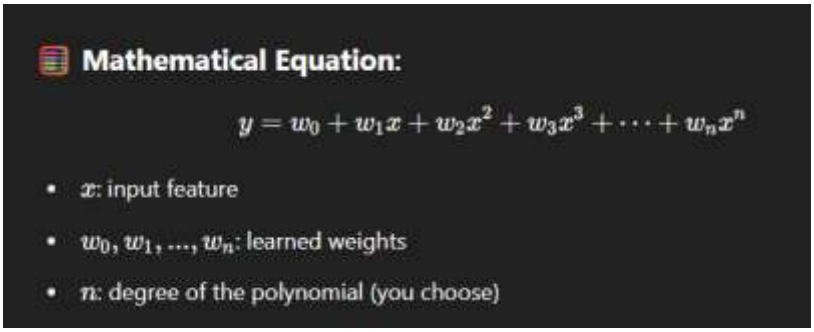
📌 **Goal of Linear Regression:**

Find the best values of w1 and w0 so that the **line fits the data as closely as possible**.

⚙️ How Does It Learn? (Behind the Scenes): We use **Loss Function + Optimization**.

2. Polynomial Regression: Polynomial Regression is a type of **supervised learning algorithm** that models the relationship between the independent variable x and the dependent variable y as an **nth-degree polynomial**. (It is a generalized form of **Linear Regression** where the data relationship is **non-linear** but still can be modeled with a curve.)
Goal: To **fit a curve (not a straight line)** through the data that best predicts the target values.

Simple Intuition: Imagine you're predicting the **price of a car** based on its **age**. Sometimes car prices drop quickly in the first few years and then level off. Relationship is not linear – cured so, a Straight fit line won't work.



Car Age (X)	Car Price (Y)
1 year	₹8L
2 years	₹6L
3 years	₹4L
4 years	₹3.8L

⚙️ How Does It Learn?

(Behind the Scenes): Same idea as Linear Regression — using **Mean Squared Error (MSE)** and **Gradient Descent** (or closed-form solution).
But before that, we **transform the input features** to polynomial form (like x^2, x^3, etc.).

3. Multivariate Regression: Multivariate Regression is an extension of linear regression where the model **uses more than one input variable** to predict a continuous output.

It's still linear, but instead of just one x, we have **multiple features** like x1, x2, ..., xn.
Goal: To find the best-fit **hyperplane** in high-dimensional space that predicts the target variable.

Simple Intuition: Imagine you're predicting **house price**, but now based on:

- Square footage
- Number of bedrooms
- Location score
- Age of the house

These are multiple **independent variables** used together.

Mathematical Equation:

$$y = w_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n$$

- y: target (e.g., house price)
- $x_1, x_2, \ldots, x_n$: features (e.g., size, rooms)
- w_i: weights learned

How It Learns:

- Uses **MSE** as loss function
- Optimized using **Gradient Descent** or **Normal Equation**

Square Feet	Bedrooms	Age	Price (₹)
1000	2	5	50L
1500	3	3	75L
2000	4	2	100L

4.Ridge Regression (L2 Regularization): Ridge Regression is a **linear regression** technique that adds an **L2 penalty** (squared magnitude of coefficients) to the **loss function**.

It discourages large weights by adding a **penalty term**, making the model more generalizable and reducing overfitting.

🎯 Goal:

Minimize prediction error + keep the weights small.

🧠 Simple Intuition:

Suppose you have **many features** (columns), and some are **correlated** or **noisy**.

- Normal linear regression might **overfit**.
- Ridge shrinks the coefficients without making any of them zero — it **controls complexity**.

🧮 Mathematical Formulation:

Loss Function:

$$\text{Cost} = \sum (y_i - \hat{y}_i)^2 + \lambda \sum w_j^2$$

Where:

- λ: regularization strength (you tune this)
- w_j: model coefficients (weights)

⚙️ How It Helps:

- Prevents overfitting by **shrinking weights**
- Keeps all features, just **reduces their influence**

✅ Summary Table:

Aspect	Details
Type	Supervised Learning
Target	Continuous Output
Penalization	$\lambda \sum w^2$ (L2 norm)
Coefficients	Shrunk toward zero, but not exactly zero
Best For	Multicollinear data, large feature sets

5.Lasso Regression (L1 Regularization): Lasso Regression adds an **L1 penalty** (absolute value of coefficients) to the loss function.

It's like Ridge, but this version can **eliminate irrelevant features** by shrinking some weights to **exactly zero**.

🎯 **Goal:**

Minimize prediction error + feature selection.

🧠 **Simple Intuition:**

Let's say you're working with **hundreds of features**, but only **a few actually matter**.

- Lasso **automatically selects important features**
- Unimportant features get zero weight (they are ignored)

🧮 **Mathematical Formulation:**

Loss Function:

$$\text{Cost} = \sum (y_i - \hat{y}_i)^2 + \lambda \sum |w_j|$$

Where:

- λ: regularization strength
- w_j: weights

⚙️ **How It Helps:**

- Reduces overfitting
- Does **automatic feature selection** (sparse model)

✅ **Summary Table:**

Aspect	Details
Type	Supervised Learning
Target	Continuous Output
Penalization	lambda\sum
Coefficients	Many become exactly zero
Best For	High-dimensional, sparse data

6. Elastic Net Regression: It is a regularized linear regression technique that combines both L1(Lasso) and L2(Ridge)

It brings the best of both worlds:

- Feature selection from Lasso
- Coefficient shrinkage from Ridge

Goal: Prevent overfitting and improve model generalization by:

- Keeping only important features
- Controlling large coefficient values

Simple Intuition:

Let's say you have a dataset with:

- Many features, some of which are correlated
- You want to select useful ones, but also not throw away correlated ones
- Lasso will randomly pick one and drop the rest (not ideal)
- Ridge keeps all, but doesn't remove noise
- Elastic Net handles both scenarios well by balancing them.

Mathematical Equation:

Elastic Net minimizes the following:

$$\text{Loss} = \sum(y_i - \hat{y}_i)^2 + \lambda_1 \sum |w_j| + \lambda_2 \sum w_j^2$$

Where:

- λ_1: controls **L1** penalty (like Lasso)
- λ_2: controls **L2** penalty (like Ridge)

Or with a mixing ratio $\alpha \in [0, 1]$:

$$\text{Loss} = \sum(y_i - \hat{y}_i)^2 + \lambda \left[\alpha \sum |w_j| + (1 - \alpha) \sum w_j^2\right]$$

- $\alpha = 1$: behaves like **Lasso**
- $\alpha = 0$: behaves like **Ridge**
- $\alpha = 0.5$: mix of both

When to Use Which?

Type	Use When...
Ridge	You have **many small/medium features**, and all are somewhat useful
Lasso	You want to **select features** or remove irrelevant ones
ElasticNet	You want **both** Ridge + Lasso behavior (L1 + L2)

CLASSIFICATION ALGORITHMS

1. Logistic Regression: Most of them think its Regression Algorithm But, Logistic Regression is actually a classification algorithm that predicts the probability of a data point belonging to a particular class using a sigmoid (S-shaped) function.
used for **binary and multi-class classification**,

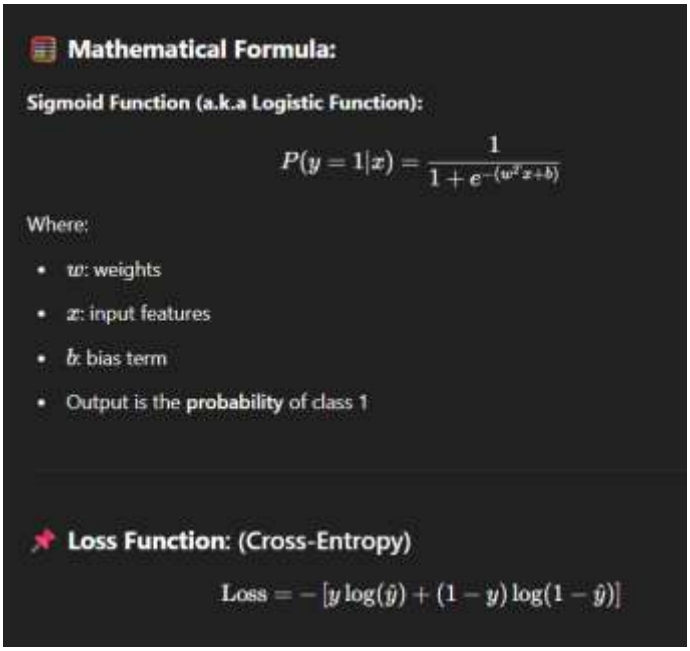
Why the Name "Regression"?

Because it is **based on linear regression concepts** but instead of predicting a continuous output, it uses a **sigmoid function** to output **probabilities**, which are then converted into class labels (e.g., 0 or 1).

Simple Intuition:

Imagine you're predicting whether a person will buy a product based on age and income.

- Linear regression might predict a value like 0.8 or 1.2
- But we need a **yes/no** answer → logistic regression takes that value and **squeezes it between 0 and 1**
- If result > 0.5 → Class 1
- If result ≤ 0.5 → Class 0



Aspect	Details
Type	Supervised Learning
Problem Type	Classification
Output	Probability (0 to 1) → Class Label

Aspect	Details
Function Used	Sigmoid / Logistic Function
Loss Function	Cross Entropy / Log Loss
Works Best When	Target is binary or multi-class

2.Naive Bayes: Naive Bayes is a **probabilistic classification algorithm** based on **Bayes' Theorem**, with the "naive" assumption that **all features are independent** of each other.

😳 Why Called "Naive"?

Because it **assumes** that all input features (variables) are **independent given the class label**, which is rarely true in real-world data — but the algorithm still performs surprisingly well!

🧠 Simple Intuition:

Imagine you're classifying emails as **Spam** or **Not Spam**.

- You look at the presence of words like "free", "money", "offer", etc.
- Even if some words are related, Naive Bayes treats them as **independent** and multiplies their individual probabilities to compute the overall likelihood.

📐 Mathematical Formula:

Bayes' Theorem:

$$P(Y|X) = \frac{P(X|Y) \cdot P(Y)}{P(X)}$$

Where:

- $P(Y|X)$: Posterior probability (target given features)
- $P(X|Y)$: Likelihood (features given target)
- $P(Y)$: Prior probability of the class
- $P(X)$: Probability of the input (used for normalization)

🔧 In Naive Bayes:

$$P(Y|x_1, x_2, \ldots, x_n) \propto P(Y) \cdot P(x_1|Y) \cdot P(x_2|Y) \cdot \ldots \cdot P(x_n|Y)$$

Each feature x_i is considered **independent** given class Y.

Types of Naive Bayes:

Type	Used For
Gaussian NB	Continuous data (assumes normal distribution)
Multinomial NB	Discrete counts (e.g., word frequencies)
Bernoulli NB	Binary/Boolean features (e.g., word present or not)

REGRESSION & CLASSIFICATION ALGORITHMS

1. Decision Tree: A Decision Tree is a **tree-like model** used to make decisions by **splitting the data** into smaller subsets based on feature values, leading to a prediction at the **leaf nodes**.

Simple Intuition: Imagine you're trying to decide whether to play tennis.
You ask:

- Is it **sunny**?
    - Yes → Check **humidity**?
        - High → Don't play
        - Normal → Play
    - No → Check if it's **rainy** or **overcast**, and so on...

You're building a **flowchart of decisions**, which is exactly how a decision tree works!

How It Works:

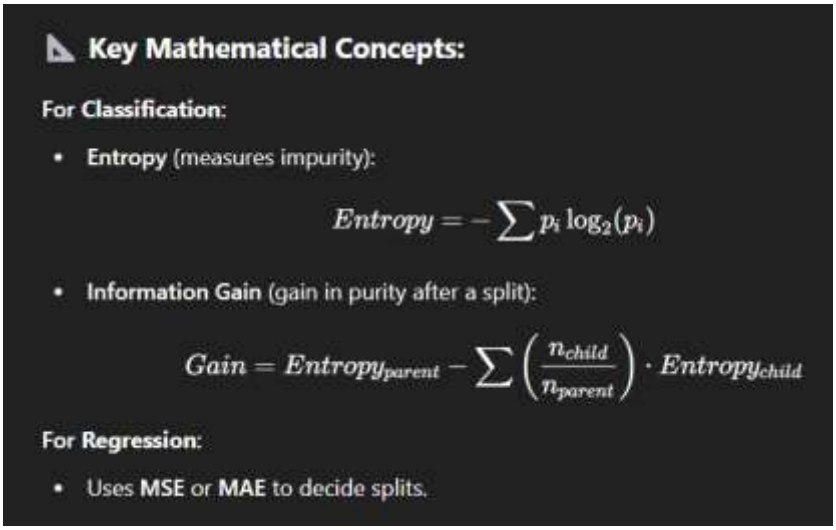
1. **Start** at the root node (whole dataset)
2. **Choose the best feature** to split based on:
    - **Information Gain** (for classification)
    - **Gini Impurity** (for classification)
    - **MSE (Mean Squared Error)** (for regression)
3. **Split** the data into branches
4. **Repeat** the process recursively until:
    - A stopping condition is met (max depth, pure node, etc.)

📌 **Advantages:**

- Easy to understand and visualize
- Works on both numeric and categorical data
- No need for feature scaling
- Can capture non-linear relationships

⚠️ **Disadvantages:**

- **Overfitting** on noisy data (fixable using pruning or ensemble methods)
- Sensitive to small changes in data



How Decision Trees Work for Both Classification & Regression

If Target is Categorical → Classification Tree

✅ What it does:

- Predicts class labels (like “yes” or “no”, “spam” or “not spam”).

🔧 How it chooses splits:

- Uses impurity metrics to find the best feature to split:
  - Gini Impurity
  - Entropy (Information Gain)

📦 Example:

Predict whether a customer will churn:

- Features: age, contract type, monthly charges
- Target: churn = yes or no

If Target is Continuous → Regression Tree

✅ What it does:

- Predicts a numeric/continuous value (like price, temperature, etc.)

🔧 How it chooses splits:

- Uses error reduction metrics:
  - Mean Squared Error (MSE)
  - Mean Absolute Error (MAE)

📦 Example:

Predict house price:

- Features: size, location, number of rooms

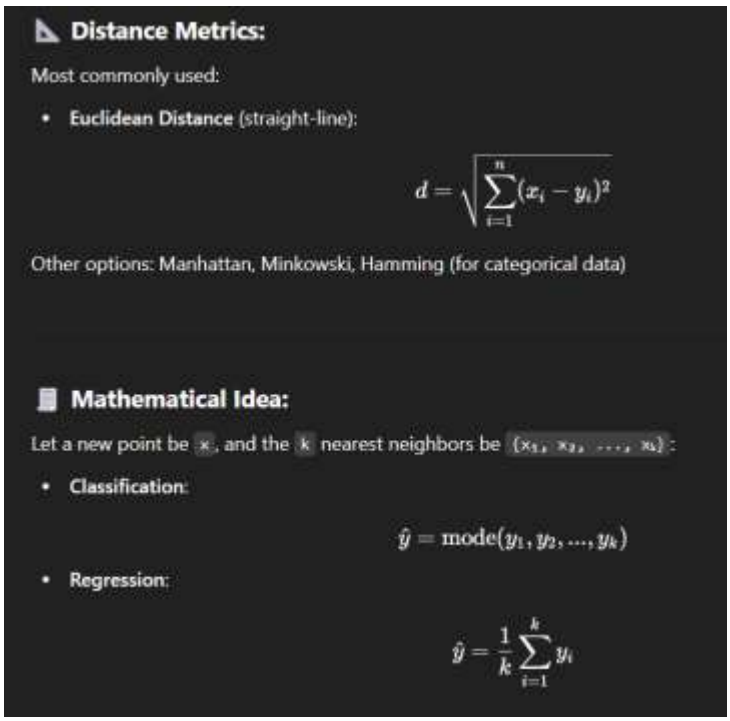
- Target: price in ₹

2. KNN: K-Nearest Neighbors is a **lazy**, **instance-based learning algorithm**:

- It **stores** all training data.
- To predict, it **looks at the K closest points** (neighbors) to a query point.
- It **aggregates** their labels:
  - **Classification**: majority vote
  - **Regression**: average of values

🔧 **How It Works:**

1. **Choose a value for k** (number of neighbors to consider).
2. **Measure the distance** between the new input and all training examples.
3. **Pick the k closest data points**.
4. For:
   - **Classification** → Majority vote of neighbors' class labels.
   - **Regression** → Average the values of neighbors.



Simple Example (2D Classification)

Imagine this dataset:

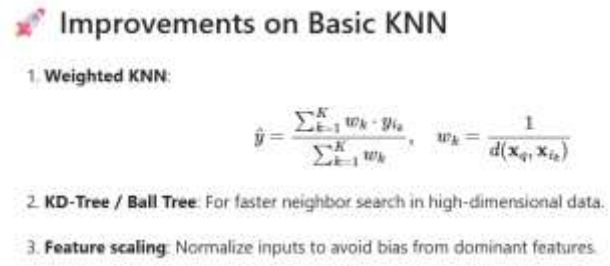
Blue points → Class 0
Red points → Class 1

Now, a new point lands in the middle. KNN will look at its **3 nearest neighbors**:

- If 2 of them are red and 1 is blue → it predicts **Class 1**
- If you increase K, the decision can flip.

📉 *Bias-Variance Tradeoff*

K Value	Bias	Variance	Description
Small K (e.g., 1)	Low	High	Very sensitive to noise (overfitting)
Large K	High	Low	Smoother boundaries (underfitting)



3.Support Vector Machines (SVM): SVM is a machine learning method that draws the **best boundary** between two groups in data. It tries to **keep this boundary as far away as possible** from the closest points on both sides.

Kernels (For non-linear data)

- Kernels let SVM work with **curved boundaries**.
- They "trick" the model into thinking the data is in a higher dimension.

Common Kernels:

Type	Use when...
Linear	Data is already separable
RBF	Complex, unknown patterns
Polynomial	Data has curved relationships
Sigmoid	Rarely used, like neural nets

🔧 Important Parameters (SVC)

- **C** = Controls margin flexibility.
- **Kernel** = Type of transformation.
- **Gamma** = Controls how far a single point can affect the boundary.
- **Degree** = Used in polynomial kernels.

📏 SVM for Regression (SVR)

- **Used for:** Predicting numbers (not classes).
- Fits a line or curve within a margin (called **epsilon-tube**).

- **Epsilon** = How much error we allow without penalty.
- **C** = Balances smoothness vs. closeness to the data.

✅ **SVC (Support Vector Classifier)**:
Used when the goal is to **classify things** (like cat vs. dog, spam vs. not-spam).
- It draws a line (or curve) that **separates classes** with the **maximum gap** between them.
- If perfect separation isn't possible, it allows some mistakes but still tries to keep the gap wide (this is the **soft margin** idea).

Easy Example: "Is this email spam or not?" — SVC finds the best rule (line) to decide that.

📈 **SVR (Support Vector Regressor)**:
Used when the goal is to **predict numbers** (like house prices, temperatures).
- It tries to draw a **line that fits the data**, but allows a small **tolerance (ε)** where errors are okay (called the ε-tube).
- Only points **outside** this tube affect the model.

Easy Example: "What will be the house price next year?" — SVR draws a line that predicts it, ignoring small errors.

Method	What It Does	Use Case
SVM	A base method that finds the best boundary or fit	Classification or regression
SVC	Classifies things by separating them with a wide margin	Spam detection, image recognition
SVR	Predicts numbers by fitting a line within an error margin	Price prediction, time series

Ensemble Techniques

Ensemble Techniques in Machine Learning: Ensemble techniques combine predictions from multiple models to improve accuracy, reduce variance/bias, and create more robust models.

1.Random Forest (Bagging): A **Random Forest** is an **ensemble learning method** that builds **multiple decision trees** and combines their outputs to improve performance and reduce overfitting.
- For **classification**, it takes the **majority vote**.
- For **regression**, it takes the **average** of predictions.

How Random Forest Works (Both Classifier & Regressor)?
Training Phase:
1. Draw **random bootstrap samples** from training data (sampling with replacement).
2. For each sample, grow a **Decision Tree**:
    - At each node, a **random subset of features** is considered (not all features).
    - Tree is grown to full depth or until stopping conditions.
3. Repeat for **N trees**.

Prediction Phase:

- **Classification**: Each tree votes for a class → final prediction = majority vote.
- **Regression**: Each tree gives a number → final prediction = average of all outputs.

3. Random Forest Classifier

Outcome:

- Predicted class labels
- Class probabilities (by vote frequency)

Example Output:

```
model.predict(X)       # e.g., ['cat', 'dog', 'dog']
model.predict_proba(X) # e.g., [[0.1, 0.9], [0.8, 0.2], [0.3, 0.7]]
```

---

4. Random Forest Regression — Math Intuition & Formulas

Let's say we want to predict a continuous value $\hat{y}$ for input $\mathbf{x}$.

You grow T decision trees:

$$\hat{y}_t = f_t(\mathbf{x}) \quad \text{for } t = 1 \text{ to } T$$

Final Prediction:

$$\hat{y} = \frac{1}{T} \sum_{t=1}^{T} f_t(\mathbf{x})$$

This is just the **average** of all predictions.

1.Ada Boost (Boosting): AdaBoost is a Boosting technique that combines multiple weak learners (typically decision stumps) into a strong model by focusing more on difficult-to-predict instances.

Intuition Behind AdaBoost

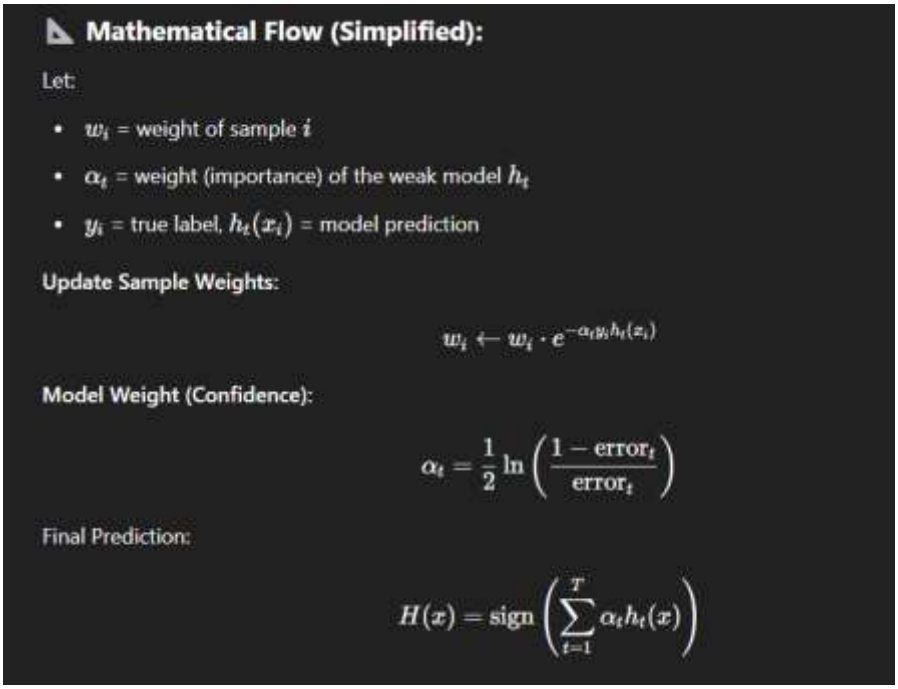
Imagine you're a teacher:

- In the first round, you teach a topic and test your students.
- The students who got it wrong? You focus more on them in the next lesson.
- Keep repeating until almost everyone understands.

That's what AdaBoost does — it gives more weight to the "harder" examples.

How AdaBoost Works

1. Start with equal weights for all training examples.
2. Train a weak model (e.g., a decision stump).
3. Evaluate performance:
    - Increase the weights for misclassified examples.
    - Decrease the weights for correctly classified ones.
4. Train the next weak learner on the updated weights.
5. Combine all weak learners using a weighted vote.



Feature	AdaBoost Details
Type	Boosting Ensemble
Base Estimator	Decision Stump (1-level decision tree)
Learner Style	Sequential (each model corrects the last)
Handles Classification	✅ Yes
Handles Regression	✅ Yes (modified version: AdaBoost.R2)
Bias/Variance	Reduces **bias**
Sensitive to Noisy Data	⚠️ Yes (outliers can mislead it)
Scikit-learn Class	AdaBoostClassifier, AdaBoostRegressor

2.Gradient Boosting: Gradient Boosting is a powerful Boosting technique that builds models sequentially, where each new model tries to correct the errors of the previous one — using gradients (derivatives of a loss function).

Goal: To minimize prediction error by learning from previous mistakes using gradient descent.

🧠 Simple Intuition:

Imagine you're painting a picture layer by layer:

- First layer: rough sketch.
- Next layers: correct details, shadows, and refine lines.
- Final result: a polished artwork.

In Gradient Boosting, each new model (tree) adds a small correction to the errors made by the previous models.

How It Works – Step by Step:

1. Start with a simple prediction (e.g., mean of target values).
2. Calculate residuals (errors between actual and predicted values).
3. Fit a new model to these residuals.
4. Update predictions by adding a fraction of the new model's output.
5. Repeat steps 2–4 for a set number of iterations or until errors are small.

Math Behind It (Simplified):

Let:

- y_i be the true value
- $\hat{y}_i^{(t)}$ be the prediction at iteration t
- $L(y, \hat{y})$ be a loss function (e.g., squared loss for regression)

Step 1: Initialize model with constant prediction

$$\hat{y}_i^{(0)} = \text{mean}(y_i)$$

Step 2: For each iteration $t = 1, 2, ..., T$:

- Compute **pseudo-residuals** (negative gradient):

$$r_i^{(t)} = -\left[\frac{\partial L(y_i, \hat{y}_i^{(t-1)})}{\partial \hat{y}_i}\right]$$

- Fit a weak model $h_t(x)$ to residuals
- Update prediction:

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + \eta \cdot h_t(x_i)$$

where η is the **learning rate** (controls step size)

Parameter	Meaning
n_estimators	Number of boosting rounds (trees)
learning_rate	Shrinks contribution of each tree
max_depth	Depth of individual trees
subsample	Sample fraction of data for each tree
loss	Loss function (MSE, log-loss, etc.)

3.XGBoost (Extreme Gradient Boosting): XGBoost is an optimized and regularized version of Gradient Boosting. It is designed to be extremely fast, accurate, and scalable — especially for structured/tabular data.

🧠 Intuition

XGBoost builds trees one at a time, like traditional Gradient Boosting.
But it adds several improvements:

- Regularization to avoid overfitting
- Efficient handling of missing values
- Parallelization of tree building
- Optimized computation using advanced data structures (DMatrix)

🧠 Mathematical Foundation (Simplified)

XGBoost minimizes a **regularized objective function**:

$$\mathcal{L}(\phi) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

Where:

- l is the loss function (e.g., MSE, log-loss)
- f_k is a tree in the ensemble
- $\Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2$
  (regularization term: number of leaves T, weights w)

Each tree adds to the previous predictions:

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$$

XGBoost uses **second-order Taylor approximation** (gradient + hessian) for fast and accurate optimization.

Parameter	Description
n_estimators	Number of trees
learning_rate	Step size shrinkage
max_depth	Maximum depth of a tree
gamma	Minimum loss reduction to make a split
lambda	L2 regularization term
alpha	L1 regularization term

Parameter	Description
subsample	% of rows to sample per tree
colsample_bytree	% of features to use per tree

4.LightGBM (Light Gradient Boosting Machine): LightGBM is a high-performance, gradient boosting framework developed by Microsoft. It's designed for speed and efficiency, especially on large datasets with lots of features.

How It Works (Simplified)

- **Step 1**: Convert continuous values to discrete bins (e.g., 255 bins).
- **Step 2**: Use **gradient and hessian** (like XGBoost) to split nodes.
- **Step 3**: Grow the tree **leaf-wise** by choosing the split that gives **maximum gain**.
- **Step 4**: Repeat until a stopping criterion (like max leaves or depth) is met.

🔢 Objective Function (same as XGBoost)

LightGBM minimizes a **regularized loss function**:

$$\mathcal{L} = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \Omega(f)$$

Where:

- l is the loss function (e.g., MSE, binary log loss)
- $\Omega(f)$ is a complexity penalty (like regularization)

Parameter	Purpose
num_leaves	Number of leaves in one tree
max_depth	Maximum depth of tree
learning_rate	Step size shrinkage
n_estimators	Number of boosting rounds
feature_fraction	Fraction of features used per iteration
bagging_fraction	Row sampling per iteration
lambda_l1 / lambda_l2	L1 and L2 regularization terms
min_data_in_leaf	Minimum samples per leaf to avoid overfitting

5.CatBoost: CatBoost is a **gradient boosting decision tree (GBDT)** algorithm — like XGBoost or LightGBM — but it adds **unique mathematical ideas** to improve performance on **categorical data** and reduce **overfitting**.

CatBoost is unique because it avoids overfitting using ordered boosting, handles categorical features natively, and builds symmetric trees that are efficient and accurate.

How it Works: CatBoost works by building decision trees sequentially using gradient boosting, while handling categorical features using target-based encoding with ordered statistics. It prevents overfitting by using ordered boosting, which learns only from past data in each permutation.

Mathematically:

Let's say we want to model a function $F(x)$ that predicts the target y.

We start with an initial prediction:

$$F_0(x) = \text{average}(y)$$

Then for **each iteration** m, we:

1. Compute the gradient (i.e., the error/residual)

$$r_i = -\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}$$

2. Train a new tree $h_m(x)$ to predict these gradients.
3. Update the model:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$$

Where:

- γ_m is the learning rate
- $h_m(x)$ is the m-th decision tree

Type	Algorithm	Goal	When to Use	Used In
Bagging	Random Forest	Reduce overfitting (variance)	When model overfits	Tabular data, medical, banking
Bagging	Bagged Trees	Stable and accurate model	For robust predictions	Spam detection, churn prediction
Boosting	AdaBoost	Improve weak models	Simple models need improvement	Face & fraud detection
Boosting	Gradient Boosting	Minimize errors step-by-step	Need better accuracy	Sales prediction, risk scoring
Boosting	XGBoost	Fast + accurate boosting	Large structured data	Competitions, fintech
Boosting	LightGBM	Faster + memory efficient	Big data, high speed needed	Ranking, large datasets
Boosting	CatBoost	Handles categories directly	Many categorical features	Ecommerce, recommendations

UNSUPERVISED LEARNING

CLUSTERING ALGORITHMS

1. K-Means Clustering: **K-Means** is a **clustering algorithm** used in **unsupervised machine learning** to group similar data points into **K number of clusters**.

🧠 **Simple Intuition:**

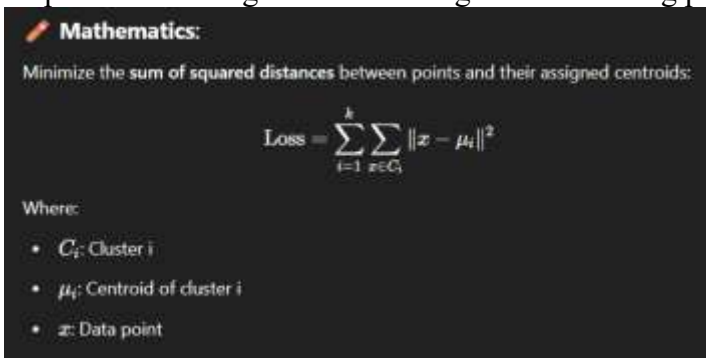
Imagine you want to group customers by purchasing habits. K-Means will divide them into K groups where customers in the same group behave similarly.

⚙️ **How It Works (Steps):**

1. **Choose K** (number of clusters).
2. **Initialize K centroids** randomly.
3. Assign each data point to the **nearest centroid**.
4. Recalculate centroids as the **mean of assigned points**.
5. Repeat steps 3–4 until **centroids stop moving** (convergence).

How do we select K value?
Wcss = With in the Cluster Sum of Squares.

Eucledian Distance: it is used in K-Means to measure how close a point is to a cluster's centroid.
Random Intialization: it provides starting centroids to begin the clustering process.



When to Use:

- You know (or can estimate) the number of clusters (K)
- Data is continuous and well-separated
- You want **hard clustering** (one point = one cluster)

❌ **Limitations:**

- Need to specify K
- Doesn’t work well with irregular shapes or outliers
- Sensitive to initial centroid placement

2.Hierarchical Clustering: Hierarchical Clustering is an unsupervised machine learning algorithm used to group similar data points into clusters by building a tree-like structure (called a dendrogram) that shows how data points are related.

Types:

1. **Agglomerative (Bottom-Up) →**
   Start with each point as its own cluster and merge the closest ones step by step.

2. **Divisive (Top-Down) →**
   Start with all points in one cluster and split them recursively.

🛠️ **How It Works (Agglomerative – Most Common):**

1. Start with each data point as its own cluster
2. Find the two closest clusters and merge them
3. Repeat step 2 until all data points are in one big cluster
4. Visualize using a dendrogram
5. Cut the dendrogram at a certain height to decide how many clusters you want

📈 Dendrogram Example: - Used to Decide How many numbers of clusters to be used.

A tree diagram showing how clusters are combined. Cutting it horizontally gives your final clusters. And threshold value mainly helps to decide how many clusters to be make.

Distance Measures (Linkage Criteria):

- **Single Linkage** → Min distance between points
- **Complete Linkage** → Max distance
- **Average Linkage** → Avg distance
- **Ward's Method** → Minimizes variance between clusters

When to Use:

- You don't know the number of clusters (K)
- You want to see nested relationships
- Dataset is not too large (can be slow for big data)

2.DBSCAN Clustering: DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is an unsupervised machine learning algorithm used to group together points that are densely packed, while ignoring outliers (noise).

- It can discover clusters of different shapes and size from a large amount of data which is containing noise and outliers.

Term	Meaning
ε (epsilon)	Radius to search for neighboring points
MinPts	Minimum number of points required to form a dense region
Core Point	Has at least MinPts within ε radius
Border Point	Has fewer than MinPts within ε, but is close to a core point
Noise	Not a core or border point (isolated/outlier)

⚙️ **How DBSCAN Works:**

1. Pick an unvisited point.
2. If it has enough neighbors (MinPts within ε), it's a **core point**, start a cluster.
3. Expand the cluster by including neighbors.
4. Repeat until all points are visited.

Why Use DBSCAN?

- Can find **arbitrarily shaped clusters**
- **Ignores noise/outliers**
- Doesn't require number of clusters (K) in advance

3. Mean Shift Clustering: Mean Shift is an unsupervised clustering algorithm that locates the centers of dense regions (clusters) by shifting data points iteratively toward the highest density of data points (i.e., the mode of the distribution).

- It **does not require the number of clusters (K)** to be specified in advance.
- It can find clusters of **arbitrary shape** and is useful for **mode-seeking**.

Term	Meaning
Bandwidth (h)	Radius used to define the neighborhood (window) around a point (also called kernel size)
Kernel	A weighting function (e.g., Gaussian) that determines the influence of nearby points
Mode	The peak/density maximum found by shifting the point to the mean of nearby points
Convergence	When shifts become small enough or no more updates are needed

⚙️ **How Mean Shift Works:**

1. **Initialize**: Start at each data point.
2. **For each point**:
    - Define a window around it using **bandwidth**.
    - Compute the **mean of all points** within this window.
    - Shift the center to the **mean** of those points.
3. Repeat the shift step until convergence (i.e., the window center no longer moves).
4. Points that converge to the **same mode** belong to the **same cluster**.

⭐ **Why Use Mean Shift?**

- ✅ Can find **arbitrarily shaped** clusters
- ✅ Automatically discovers the **number of clusters**
- ✅ Ignores outliers that do not converge with any dense region
- ✅ Works well with **non-parametric density estimation**

📉 **Limitations:**

- 🛑 **Computationally expensive** for large datasets (especially in high dimensions)
- 🛑 **Bandwidth parameter** is sensitive — too small = too many clusters, too big = merge clusters

4. OPTICS: Ordering Points To Identify the Clustering Structure

It is an **unsupervised, density-based clustering algorithm** like DBSCAN, but better at:

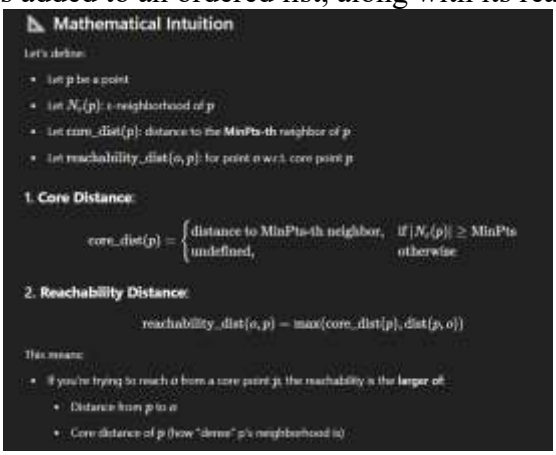
- Detecting **clusters of varying density**
- Avoiding hard cutoffs caused by ε in DBSCAN

How OPTICS Works (Step-by-step)

Term	Meaning
ε (epsilon)	Maximum neighborhood radius (but not a hard cutoff like DBSCAN)
MinPts	Minimum number of neighbors to form a dense region
Core Distance	Distance from a point to its MinPts-th nearest neighbor
Reachability Distance	Distance to a point from a core point, adjusted by core distance

🧠 **Main Steps:**

1. **For each point**:
   - Compute **core distance**: the distance to the MinPts-th closest point.
2. **Start from a point with smallest reachability**:
   - Create a priority queue of neighbors based on **reachability distance**.
3. **Expand neighbors recursively**:
   - Only if core distance exists (i.e., it's a core point)
4. **Record the ordering**:
   - Each point is added to an ordered list, along with its reachability distance.



3.Silhouette (Clustering): The Silhouette Score is a metric to evaluate the quality of a clustering — it measures how well a data point fits within its own cluster versus how well it would fit in the next closest cluster.

It works for any clustering algorithm (KMeans, DBSCAN, OPTICS, etc.), and is particularly good for:

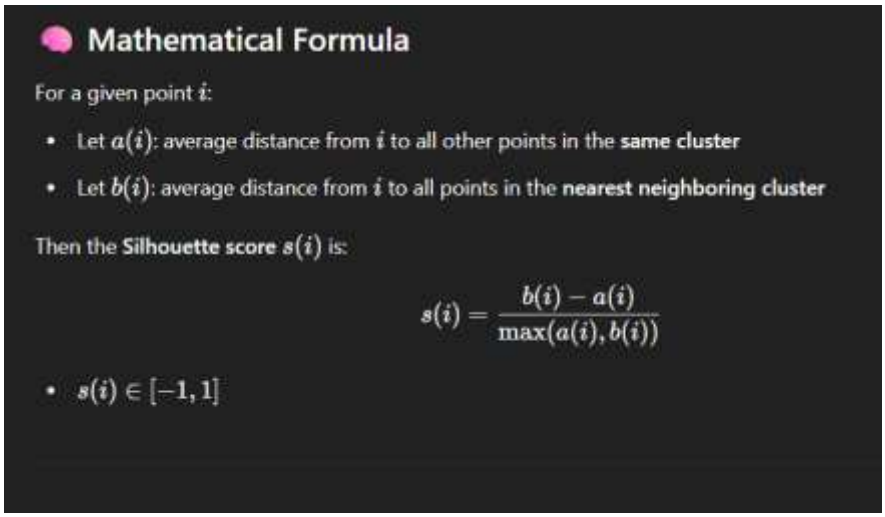
- Choosing the optimal number of clusters (K)
- Comparing different clustering results

Intuition:

"Silhouette compares intra-cluster tightness with inter-cluster separation for every point."

If a point is:

- Well-matched to its own cluster and far away from others → score close to 1
- Between two clusters → score near 0
- Misclassified or in the wrong cluster → score close to -1



Interpret the Score:

Score Range	Interpretation
≈1	Perfect match to cluster; far from others
≈0	On or near decision boundary
< 0	Probably assigned to wrong cluster

DIMENSIONALITY REDUCTION

1. PCA (Principal Component Analysis) is an **unsupervised linear transformation** technique used to:

- Reduce the number of features (dimensions)
- Retain **maximum variance** (most important information)
- Remove multicollinearity and noise

Why Use PCA?

Goal	Benefit
Reduce dimensions	Less memory & faster computation
Remove redundancy	Handles correlated features
Improve visualization	2D or 3D plots of high-D data
Preprocessing	Before ML models to reduce overfitting

Math Behind PCA (Step-by-Step)

Let's say we have a data matrix X of shape $n \times d$

1. Standardize the Data (zero mean)

$$X_{\text{norm}} = X - \bar{X}$$

2. Compute Covariance Matrix:

$$\Sigma = \frac{1}{n-1} X^T X$$

3. Find Eigenvectors and Eigenvalues:

$$\Sigma v = \lambda v$$

- v: eigenvector → direction (principal component)
- λ: eigenvalue → amount of variance explained

4. Sort Eigenvectors:

- Keep the top **k** eigenvectors with largest eigenvalues

5. Transform Data:

$$X_{\text{reduced}} = X \cdot W$$

- Where W is a matrix of top **k** eigenvectors

PCA OUTPUT:

Output	Meaning
Principal Components	**New axes/directions**
Explained Variance Ratio	**% of variance each component explains**
Transformed Data	**Original data projected into lower dimensions**

When to Use PCA?

✅ Before clustering (KMeans, DBSCAN)
✅ Before modeling to reduce overfitting
✅ For feature extraction or compression
✅ For visualizing high-dimensional data in 2D or 3D

⚠️ When Not to Use PCA

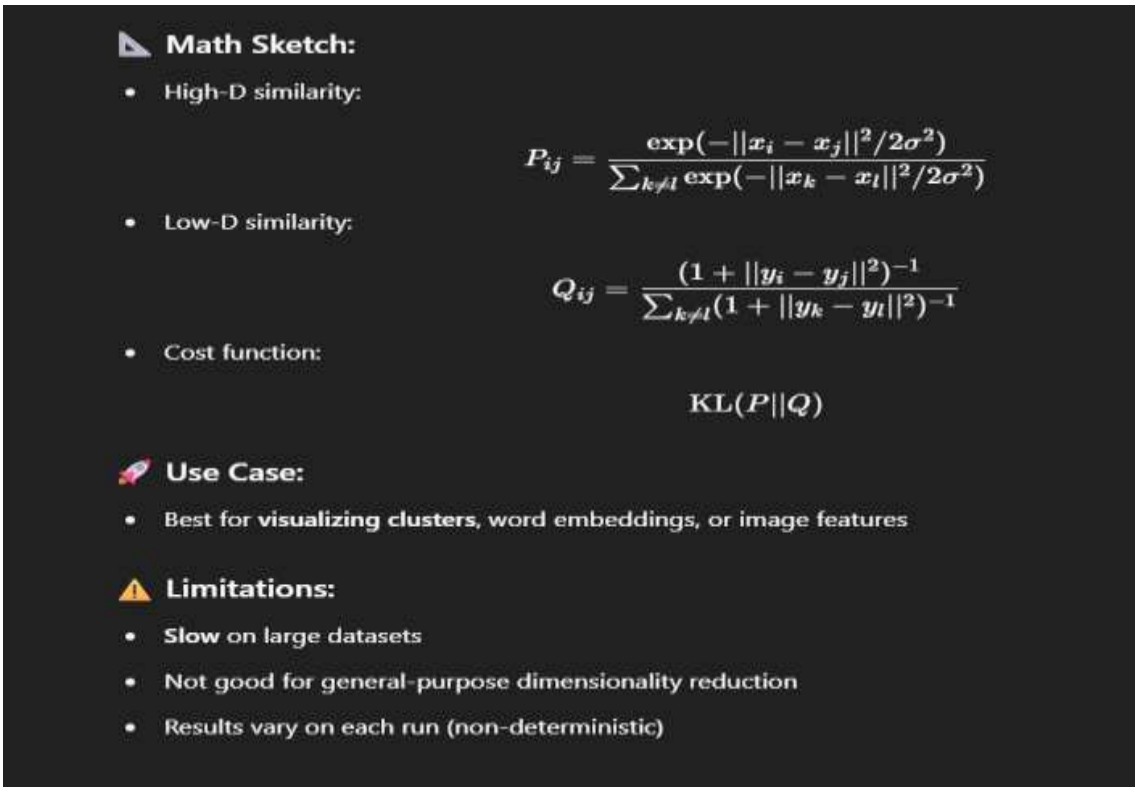
- When interpretability of original features is critical
- On data with **nonlinear structure** (use **Kernel PCA**, **t-SNE**, or **UMAP**)

2.T-SNE: t-SNE (t-distributed Stochastic Neighbor Embedding) is a machine learning technique primarily used for visualizing high-dimensional data by reducing it to a lower-dimensional space, typically 2D or 3D, for easier analysis and interpretation.

🧠 **Intuition:** If two points are close in high-D space, their similarity (probability) should stay high in 2D or 3D.

⚙️ How It Works:

1. Compute pairwise similarities in high-D using a Gaussian distribution.
2. Compute similarities in low-D using a Student t-distribution.
3. Minimize Kullback–Leibler (KL) divergence between the two.



3. UMAP: UMAP is a dimensionality reduction algorithm used to project high-dimensional data into 2D or 3D space while preserving the structure and relationships in the data.

Key Purpose: To simplify complex data for visualization, clustering, or faster model training.

How UMAP Works (Simple Terms):

1. Builds a graph of data in high-dimensional space (based on neighborhood relationships).
2. Optimizes a low-dimensional layout that maintains those relationships as closely as possible.

Mathematics Behind UMAP (High-Level Overview):

1. **Graph Construction** in High-Dimensional Space:
   - Each point finds its **nearest neighbors**.
   - For each point, a **probability distribution** is created based on the distance to its neighbors using a **fuzzy topological structure**.
2. **Low-Dimensional Embedding**:
   - UMAP builds another graph in low dimensions (2D/3D).
   - It **optimizes the layout** by trying to **minimize the difference** (cross-entropy) between the high-dimensional and low-dimensional graphs.
3. **Objective Function (Simplified)**:
   UMAP minimizes the **cross-entropy loss** between the two graphs:

$$\text{Loss} = -\sum_{(i,j)} [p_{ij} \log(q_{ij}) + (1 - p_{ij}) \log(1 - q_{ij})]$$

   - p_{ij}: probability of connection in high-dimensional space
   - q_{ij}: probability of connection in low-dimensional space

Feature	UMAP	t-SNE	PCA
Preserves global + local structure	✅ Yes	❌ Mostly local only	✅ Global only
Speed	⚡ Fast	🐢 Slower	⚡ Very fast
Dimensionality	Any (2D, 3D, >3)	Usually 2D or 3D	Any
Scalability	✅ High	❌ Less scalable	✅ High

ANOMALY DETECTION

1. Isolation Forest:

1. Isolation Forest

- **Idea**: Anomalies are **easier to isolate** with fewer random splits
- **How it works**:
  - Build random trees by randomly selecting a feature & split value
  - Anomalies require **fewer splits** to isolate
- **Anomaly Score**:

$$s(x, n) = 2^{-\frac{E(h(x))}{c(n)}}$$

  - $E(h(x))$: average path length to isolate point x
  - $c(n)$: average path length of a Binary Search Tree with n samples
- **Interpretation**:
  - Score close to **1** → likely anomaly
  - Score < **0.5** → likely normal

✅ Great for: **large datasets**, tabular data

2. Local Outlier Factor (LOF):

2. Local Outlier Factor (LOF)

- **Idea**: Anomalies have **lower local density** than neighbors
- **Key Concepts**:
  - **k-distance**: distance to k-th nearest neighbor
  - **Local Reachability Density (LRD)**: inverse of avg. distance to neighbors
  - **LOF Score**:

$$LOF_k(p) = \frac{\sum_{o \in N_k(p)} \frac{LRD_k(o)}{LRD_k(p)}}{|N_k(p)|}$$

  - LOF > 1: possible anomaly

✅ Great for: **local, density-based anomalies**

Algorithm	Core Idea	Key Concepts	Strengths	Weaknesses
Isolation Forest	Isolate anomalies by randomly splitting features	Isolation Score, Tree-based splits	Fast, Scalable, Good for high-dimensional data	Assumes anomalies are few and isolated
Local Outlier Factor (LOF)	Compares local density of a point to its neighbors	k-distance, Local Reachability Density (LRD), LOF Score	Good for local density anomalies	Struggles with high dimensions
One-Class SVM	Fits a boundary around normal data points	Decision function, Kernel trick	Theoretically strong	Slow on large datasets
Autoencoders	Reconstruct input data with neural networks	Reconstruction Error	Learns nonlinear patterns, flexible	Needs tuning & large data
Elliptic Envelope	Assumes Gaussian distribution and finds the envelope	Manhattan Distance	Simple, fast	Assumes normal distribution
KNN Anomaly Detection	Distance to k-nearest neighbors	Average/median distance to k-NNs	Simple, interpretable	Computationally expensive

Out-of-Bag (OOB) Data in Machine Learning?

Out-of-Bag Data refers to the subset of training data that was not used to train a particular model (tree) during bootstrap sampling in ensemble methods like Random Forest.

Out-of-Bag (OOB) Data Works – Step-by-Step

🧩 Step-by-Step Explanation:

1. Bootstrap Sampling:
   - Random Forest builds multiple decision trees.
   - For each tree, it randomly samples (with replacement) from the original dataset.
   - Some data points get picked multiple times; others aren't picked at all — those are the Out-of-Bag (OOB) samples for that tree.
2. Training the Tree:
   - The tree is trained only on the bootstrap sample (e.g., 63% of data).
3. OOB Evaluation:
   - After the tree is trained, we use the OOB samples to test it.
   - Since these data points were not seen during training, they give a true unbiased error estimate.
4. OOB Predictions:
   - Each OOB sample is passed through all trees that did not see it during training.
   - For each sample, majority vote (classification) or average (regression) from those trees is used to make a final prediction.

5. OOB Score:
    - The predictions are compared to the actual values.
    - Accuracy (or RMSE, etc.) is computed over just the OOB data, giving an OOB error estimate or OOB score.

Why OOB Data is Useful:

It acts like a **validation set**, without needing a separate dataset.

You can use OOB samples to:

- Estimate **model performance (OOB score)**
- Reduce the need for **cross-validation**

Why It Works Well:

- It simulates cross-validation.
- It's built-in and automatic in many libraries (e.g.,RandomForestClassifier(oob_score=True) in sklearn).
- Makes efficient use of data — no need to split the dataset manually.

Step	Description
Bootstrap Sampling	Train tree on random 63% sample (with replacement)
OOB Data	Remaining 37% unused data for that tree
Prediction	Test OOB samples using trees that didn't see them
OOB Score	Accuracy or error based on OOB predictions

TIME SERIES

Non time series: It is nothing but, in our dataset we will not be having any column which represent the time. Eg: House price prediction.

Time Series: It is nothing but, in our data, set we have a column related to Time. Eg: Sales Data.

Time Series: A Time Series is a sequence of data points collected or recorded at specific time intervals, typically ordered chronologically.

Examples of Time Series Data:

- Stock prices recorded every minute
- Daily temperature readings
- Monthly sales reports
- Heart rate monitor data per second

Task	Description
Forecasting	Predict future values (e.g., next month's sales)
Anomaly Detection	Detect unusual patterns (e.g., fraud, failures)
Classification	Classify sequences (e.g., normal vs. abnormal ECG)
Clustering	Group similar time patterns

Components of time series:

1.Trend - Where is the data heading overall?
2.Season - What patterns repeat regularly?
3.Cycle - Is there a wave pattern, but with no fixed timing?
4.Noise - Can we ignore this randomness?

Component	Meaning	Example
1. Trend	Long-term increase or decrease in the data over time	Housing prices increasing over years
2. Seasonality	Regular, periodic fluctuations in data	Sales spike every December (holiday season)
3. Cyclic	Long-term wave-like patterns, **not fixed** like seasonality	Economic recession or boom cycles
4. Noise (Irregular)	Random or unpredictable variations	Sudden data spikes due to errors or events

Concept	Definition	Use Case
Interpolation	Predicting values within the range of known data points	Estimating unknown values between observed points
Extrapolation	Predicting values outside the range of known data points	Estimating future or unseen values beyond the observed range

Types of Time Series:

1. Auto Regressive Model
2. MA (Moving Average) Model
    a. Statistical MA Models (Used in Time Series Forecasting)

    b. Smoothing MA Models (Used for Trend Detection)

3. ARMA

1. Auto Regressive Model: Predicts future values based **only on past values** of the variable itself.

How It Works:

It assumes that past values **have a linear influence** on the current value.

Example:

If you're predicting sales, AR says:

"Today's sales depend on yesterday's, the day before, etc."

Mathematical Formula:

$$Y_t = c + \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \cdots + \phi_p Y_{t-p} + \epsilon_t$$

- Y_t: Value at time t
- c: Constant term
- ϕ: Coefficients for past values
- ϵ_t: Error/noise at time t

Intuition:

If people bought 100 units yesterday, they'll likely buy around 100 today (plus some random error). The AR model **learns how strongly** past days impact today.

When to Use:

- When your data is **stationary** (no trend/seasonality)
- Strong correlation with past values
- ACF (Autocorrelation Function) shows slow decay

2. MA (Moving Average) Model: Instead of using past values, it uses past forecast errors to predict the current value.

How It Works:

MA assumes that unexpected shocks (errors) in the past affect the current value. These errors represent information that was not captured previously.

Mathematical Formula:

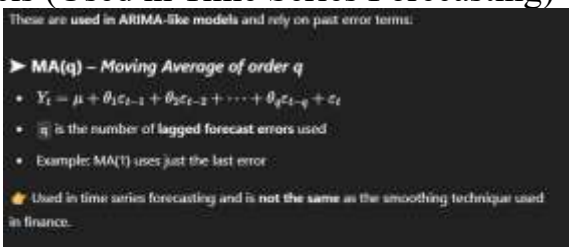
$$Y_t = \mu + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q} + \varepsilon_t$$

- Y_t: Value at time t
- μ: Mean of the series
- θ: Coefficients for past errors
- ε_t: White noise (error) at time t

Intuition:
Let's say yesterday you predicted 100 units sold, but the actual was 110 → error = +10.
An MA model says:
"Today's sales might increase based on that error — maybe something changed that I didn't capture earlier."
So it **learns from its past mistakes**.

Types of MA (Moving Average) Model:

1. Statistical MA Models (Used in Time Series Forecasting)



2. Smoothing MA Models (Used for Trend Detection)

These are the ones commonly used in **data smoothing or financial analysis.**

Type	What It Does	Example Use
Simple Moving Average (SMA)	Average of the last N points	Smoothing stock prices
Weighted Moving Average (WMA)	Recent points have more weight	Short-term price analysis
Exponential Moving Average (EMA)	Exponentially decreasing weights	Responsive smoothing
Double EMA (DEMA) / Triple EMA (TEMA)	Reduces lag further using layered EMAs	Faster reaction to data changes

3. ARMA: It is the combination of both AR(autoregressive) and MA(Moving average).

It's useful when your time series shows **both autocorrelation** and **random noise** patterns.

How It Works:
ARMA models use past values AND past errors to capture dependencies in time series data.

Mathematical Formula:

$$Y_t = c + \sum_{i=1}^{p} \phi_i Y_{t-i} + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j} + \varepsilon_t$$

Where:

- Y_t: Value at time t
- c: Constant
- ϕ_i: AR coefficients (past values)
- θ_j: MA coefficients (past errors)
- ε_t: White noise/error

Intuition:

- **AR** part: "If sales were high yesterday, they may stay high today."
- **MA** part: "If my prediction was too low yesterday, I'll adjust today's forecast upward."

ARMA Model Variants (Based on Order):

Model	Description	Formula
ARMA(1, 0)	Pure AR model (only uses past values)	$Y_t = \phi_1 Y_{t-1} + \varepsilon_t$
ARMA(0, 1)	Pure MA model (only uses past errors)	$Y_t = \theta_1 \varepsilon_{t-1} + \varepsilon_t$
ARMA(1, 1)	Uses both past value and error (classic ARMA)	$Y_t = \phi_1 Y_{t-1} + \theta_1 \varepsilon_{t-1} + \varepsilon_t$
ARMA(p, q)	General form with p AR terms and q MA terms	$Y_t = \sum_{i=1}^{p} \phi_i Y_{t-i} + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j} + \varepsilon_t$

Extended Variants of ARMA:

Model	Purpose
ARIMA (AutoRegressive Integrated Moving Average)	ARMA for **non-stationary** data (includes differencing)
SARIMA (Seasonal ARIMA)	ARIMA + handling **seasonality**
ARIMAX	ARIMA with **exogenous variables** (like external factors)
SARIMAX	SARIMA with **exogenous variables**

4.ARIMA (AutoRegressive Integrated Moving Average): ARIMA is an extension of ARMA that can handle non-stationary data by introducing a differencing step (the "I" part: Integrated).

How It Works:
ARIMA combines 3 parts:

1. **AR (p)** – AutoRegression (past values)
2. **I (d)** – Integration (**differencing** to make data stationary)
3. **MA (q)** – Moving Average (past forecast errors)

Mathematical Intuition:

If:

- Y_t is your original time series
- You apply **d differencing steps:**

$$Y_t' = Y_t - Y_{t-1} \quad \text{(1st difference)}$$

Then ARIMA models:

$$Y_t' = c + \sum_{i=1}^{p} \phi_i Y_{t-i}' + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j} + \varepsilon_t$$

📌 **Example:**
Imagine sales are growing steadily over time (trend). ARMA can't model it well unless the trend is removed.
ARIMA will **difference** the data (remove the trend), and then apply AR and MA.

5. SARIMA (Seasonal ARIMA): SARIMA is **ARIMA + Seasonality**.

It handles:

- **Trend** (like ARIMA)
- **Seasonal patterns** (e.g., monthly sales peaks)

Notation:

SARIMA is usually written as:

$$\text{SARIMA}(p, d, q) \times (P, D, Q, s)$$

Where:

- `p, d, q` = ARIMA terms
- `P, D, Q` = **Seasonal** AR, I, MA terms
- `s` = **Seasonal period** (e.g., `12` for monthly data with yearly seasonality)

Example:

For monthly sales data showing yearly peaks:

$$\text{SARIMA}(1, 1, 1) \times (1, 1, 1, 12)$$

This means:

- Differencing once to remove trend
- Another seasonal difference every 12 steps to remove **seasonal trend**
- Apply AR and MA on both regular and seasonal components

Mathematics Intuition:

It uses:

- **Regular ARMA components** after d differencing
- **Seasonal ARMA components** after D seasonal differencing

SARIMA models both:

- **Short-term memory** (via AR/MA)
- **Repeating long-term seasonal patterns** (via seasonal AR/MA)

6.SARIMAX: SARIMA is ARIMA + Seasonality.

It handles:

- Trend (like ARIMA)
- Seasonal patterns (e.g., monthly sales peaks)

Notation:

SARIMA is usually written as:

$$\text{SARIMA}(p,d,q) \times (P,D,Q,s)$$

Where:

- `p, d, q` = ARIMA terms
- `P, D, Q` = **Seasonal** AR, I, MA terms
- `s` = **Seasonal period** (e.g., `12` for monthly data with yearly seasonality)

Example:

For monthly sales data showing yearly peaks:

$$\text{SARIMA}(1,1,1) \times (1,1,1,12)$$

This means:

- Differencing once to remove trend
- Another seasonal difference every 12 steps to remove **seasonal trend**
- Apply AR and MA on both regular and seasonal components

Mathematics Intuition:

Math Intuition:

$$Y_t = SARIMA\text{-based prediction} + \beta_1 X_{1,t} + \beta_2 X_{2,t} + \ldots + \varepsilon_t$$

Where:

- Y_t is the target (e.g., sales)
- X are external inputs (e.g., ads, temp)

It uses:

- **Regular ARMA components** after d differencing
- **Seasonal ARMA components** after D seasonal differencing

Types of Time Series Models

Model	What It Does	How It Works	Maths / Intuition
1. AR (AutoRegressive)	Uses **past values** to predict the future	Current value depends on a **linear combo of past values**	$Y_t = c + \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \cdots + \epsilon_t$
2. MA (Moving Average)	Uses **past forecast errors** to predict the future	Sums up past errors with weights	$Y_t = \mu + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \ldots$
3. ARMA (AR + MA)	Combines AR and MA models	Uses both **past values + past errors**	$Y_t = c + \phi Y_{t-1} + \theta \epsilon_{t-1} + \epsilon_t$
4. ARIMA (AR + I + MA)	Adds **differencing** to make non-stationary data stationary	Handles **trend + noise**	ARMA on **differenced data**: $\Delta Y_t = Y_t - Y_{t-1}$
5. SARIMA	ARIMA + **seasonality**	Adds seasonal terms (lags like 12, 24 etc.)	Includes seasonal AR/MA terms: e.g., $Y_t = Y_{t-12} + \ldots$
6. Prophet (by Facebook)	Forecasts with **trend + season + holidays**	Additive model with automatic season detection	$y(t) = g(t) + s(t) + h(t) + \epsilon_t$
7. LSTM (Deep Learning)	Learns from **long-term patterns**	Uses memory cells to track sequences	Backpropagation through time (BPTT); neural network logic
8. Exponential Smoothing (ETS)	Forecasts by giving more weight to **recent observations**	Weighted average with exponentially decreasing weights	Simple: $\hat{y}_t = \alpha y_{t-1} + (1-\alpha)\hat{y}_{t-1}$
9. VAR (Vector AutoRegression)	For multivariate time series	Each variable depends on **itself and other variables' pasts**	$Y_t = c + A_1 Y_{t-1} + A_2 Y_{t-2} + \epsilon_t$ where Y_t is a vector

Goal	Suggested Model
Trend + Seasonality	ARIMA, SARIMA, Prophet
Complex, long-term patterns	LSTM, RNNs
Fast, smooth forecasts	Exponential Smoothing
Multivariate forecasting	VAR

REINFORCEMENT LEARNING

Reinforcement Learning: Reinforcement Learning is a type of machine learning where an agent learns to make decisions by interacting with an environment. The agent takes actions, receives rewards or penalties, and learns to maximize long-term rewards.

Why is it Used?

- To train systems that can learn from trial and error
- To optimize decision-making in environments where explicit supervision isn't feasible
- Suitable when feedback is delayed rather than instant

KEY CONCEPTS:

Term	Meaning
Agent	**The learner or decision-maker**
Environment	**The world with which the agent interacts**
Action (A)	**What the agent can do**
State (S)	**Current situation returned by the environment**
Reward (R)	**Feedback signal from the environment**
Policy (π)	**The strategy the agent follows**
Value Function (V)	**Expected reward from a state**
Q-Value Function (Q)	**Expected reward from a state-action pair**

Mathematical Intuition:

- The goal is to **maximize cumulative future rewards:**

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$$

Where:

- G_t: return at time t
- R: reward
- γ: discount factor $(0 < \gamma < 1)$, determines how much future rewards are valued

Q-learning (off-policy) update rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

Real-Life Scenario:
Imagine teaching a robot vacuum:

- It tries different paths to clean the house.
- When it avoids obstacles and covers more floor, it gets positive reward.
- When it bumps into walls, it gets negative reward.
- Over time, it learns the optimal path.

Concept	Description
RL	Learning through trial and error
Goal	Maximize long-term reward
Use Case	Dynamic, interactive systems (robotics, games, etc.)
Algorithm	Q-Learning, Deep Q Network (DQN), Policy Gradient, etc.

I have now completed all the core concepts of Machine Learning, including a comprehensive overview of the latest algorithms. For practical implementation, feel free to visit my GitHub — you'll find all the resources, code, and projects I've worked on available there.

GitHub link: Vishnu Kumar D S

Happy Learning!

By: Vishnu Kumar D S