

**Course Name: -BCAAIML**

**Subject Name: - Machine Learning Techniques**

**Course Code: -BCAAIML501**

**Sem: 5<sup>th</sup>**

### **Unit-IV**

#### **Dimensionality Reduction and Evolutionary Models**

Dimensionality reduction refers to the process of reducing the number of input variables or features in a dataset while preserving as much relevant information as possible. This helps in simplifying models, reducing computation, and improving model performance.

##### **Subpoints:**

###### **a) Linear Discriminant Analysis (LDA)**

- A supervised technique used to find a linear combination of features that characterizes or separates classes.
- Maximizes the **between-class variance** and minimizes the **within-class variance**.
- Commonly used in pattern recognition and classification problems.

###### **b) Locally Linear Embedding (LLE)**

- A **non-linear** dimensionality reduction technique.
- Preserves **local neighborhood structure** of data.
- Each data point is reconstructed from its nearest neighbors using linear coefficients, and a low-dimensional representation preserves these coefficients.

###### **c) Isomap (Isometric Mapping)**

- An extension of classical **Multidimensional Scaling (MDS)**.
- Preserves **geodesic distances** (shortest path along a manifold) between data points.
- Useful for unfolding non-linear manifolds in high-dimensional data.

###### **d) Least Squares Optimization**

- A mathematical method to minimize the **sum of squared differences** between observed and predicted values.
- Used in fitting models (e.g., linear regression) and in optimization problems within dimensionality reduction.

#### **Evolutionary Models**

Evolutionary models are optimization algorithms inspired by the process of **natural selection and genetics**. They are particularly useful in search spaces that are complex, noisy, or lack a clear mathematical formulation.

**Subpoints:**

### a) Evolutionary Learning

- A learning paradigm based on **evolutionary computation**.
- Algorithms evolve solutions over generations using mechanisms like selection, mutation, and crossover.

### b) Genetic Algorithms (GAs)

- A popular type of evolutionary algorithm.
- Works with a **population of solutions** (chromosomes) which evolve over generations.
- Applies biological processes like **selection, crossover, and mutation**.

### c) Genetic Offspring

- New individuals (solutions) created by **recombining parent chromosomes**.
- Represent the next generation in the evolutionary cycle.
- Intended to inherit better traits from parents.

### d) Genetic Operators

- **Selection:** Chooses the fittest individuals for reproduction.
- **Crossover:** Combines parts of two parents to create a new individual.
- **Mutation:** Randomly alters parts of an individual to maintain genetic diversity.

### e) Using Genetic Algorithms

- Applied in optimization problems, neural network training, game strategy evolution, etc.
- GAs do not require gradient information and can handle **discrete, non-linear, and multi-modal** functions.

## Reinforcement Learning

Reinforcement Learning (RL) is a type of machine learning where agents learn to take actions in an environment to maximize a reward signal over time.

**Subpoints:**

### a) Overview

- Involves **agents, states, actions, rewards, and environment**.
- The goal is to learn a policy that maximizes cumulative reward.
- Learning is based on **trial and error**.

### b) Getting Lost Example

- A common introductory example where an agent (e.g., a robot) learns to find a path in a maze.

- Demonstrates exploration vs. exploitation and delayed reward.

### c) Markov Decision Process (MDP)

- A formal framework for RL problems.
- Defined by a tuple  $(S, A, P, R, \gamma)$ :
  - **S**: Set of states
  - **A**: Set of actions
  - **P**: Transition probabilities
  - **R**: Reward function
  - $\gamma$ : Discount factor
- Assumes the **Markov Property**, meaning the next state depends only on the current state and action.

### Linear Discriminant Analysis (LDA)

**Linear Discriminant Analysis (LDA)** is a **supervised** dimensionality reduction and classification technique used in machine learning and statistics. It aims to project data onto a lower-dimensional space where the **classes are most separable**.

#### Key Concepts:

1. **Class Separability:**  
LDA seeks directions (linear combinations of features) that **maximize the distance between class means** while **minimizing the spread within each class**.
2. **Supervised Method:**  
Unlike PCA, which is unsupervised, LDA uses **class label information** to find the best projection.
3. **Dimensionality Reduction:**  
LDA reduces the number of features while retaining the most **discriminative information**. It is especially useful when the number of features is very large compared to the number of observations.

#### Mathematical Formulation:

LDA tries to maximize the following objective function:

$$J(w) = w^T S_B w / w^T S_W w$$

Where:

- $w$  = projection vector
- $S_B$  = **between-class scatter matrix**
- $S_W$  = **within-class scatter matrix**

#### Steps in LDA:

1. **Compute the mean vectors** for each class.
2. **Compute the within-class scatter matrix (SW)** and **between-class scatter matrix (SB)**.

3. Compute eigenvalues and eigenvectors of the matrix  $SW^{-1}SBS^T$ .
4. Select top k eigenvectors to form a transformation matrix.
5. Project the original data onto this new subspace.

### Applications of LDA:

- Face recognition
- Text classification
- Medical diagnosis
- Gene expression analysis

### Advantages:

- Reduces **overfitting** by reducing the number of dimensions.
- Improves **classification accuracy**.
- Simple and computationally efficient.

### Locally Linear Embedding (LLE)

**Locally Linear Embedding (LLE)** is an **unsupervised, nonlinear** dimensionality reduction technique that is particularly useful for unfolding or flattening data that lies on a **nonlinear manifold**.

It focuses on preserving the **local geometry** (neighborhood relationships) of the data, unlike global techniques such as PCA or LDA.

### Key Concepts:

- LLE assumes that each data point and its neighbors lie on or near a locally linear patch of the manifold.
- The algorithm preserves the **linear relationships between each point and its nearest neighbors** when mapping to a lower-dimensional space.

### Working Steps of LLE:

1. **Neighborhood Identification:**
  - For each data point, identify its **k nearest neighbors** using Euclidean distance.
2. **Compute Reconstruction Weights:**
  - Each point is reconstructed as a **weighted linear combination of its neighbors**.
  - Weights are computed such that the reconstruction error is minimized:

$$\min_W \sum_i \|X_i - \sum_j W_{ij} X_j\|^2$$

where  $X_j$  are the neighbors of  $X_i$ .

3. **Low-Dimensional Embedding:**
  - Find low-dimensional representations  $Y_i$  that preserve these weights:

$$\min Y \sum i \|Y_i - \sum j W_{ij} Y_j\|^2$$

- This ensures that the **same local geometry** is preserved in the lower-dimensional space.

### Applications:

- **Image processing** (e.g., face manifold learning)
- **Text and document visualization**
- **Genomic data analysis**
- **Pattern recognition**

### Advantages:

- Captures **nonlinear structures** in data.
- Preserves **local neighborhood relationships** effectively.
- No assumptions about global structure or linearity.:.

### Isomap (Isometric Mapping)

Isomap is a **nonlinear dimensionality reduction** technique that extends **Multidimensional Scaling (MDS)** by incorporating **geodesic distances** (i.e., shortest paths along a curved surface or manifold).

It is particularly effective at **preserving global geometry** in data that lies on a nonlinear manifold.

### Key Concepts:

- Unlike PCA, which preserves Euclidean distances, Isomap preserves **geodesic distances**.
- Geodesic distance: the shortest path between two points **along the manifold**, not through space.

### Steps in Isomap:

1. **Construct Neighborhood Graph:**
  - Use **k-nearest neighbors (k-NN)** or  **$\epsilon$ -radius** to define a graph where nodes are connected to nearby points.
2. **Compute Geodesic Distances:**
  - Use algorithms like **Dijkstra's** or **Floyd-Warshall** to compute the **shortest path distances** between all pairs of points in the graph.
3. **Apply Classical MDS:**
  - Perform **Multidimensional Scaling** on the distance matrix to find a lower-dimensional embedding that preserves those distances.

### Applications:

- **Facial image manifolds**

- Handwritten digit visualization
- Speech recognition
- Bioinformatics

#### Advantages:

- Captures both **local** and **global structure**.
- Handles **nonlinear relationships** in data effectively.

#### Limitations:

- Sensitive to the choice of **k** or  **$\epsilon$**  (neighborhood size).
- High computational cost for **large datasets**.
- Poor performance on **noisy or disconnected manifolds**.

### Least Squares Optimization

**Least Squares Optimization** is a **mathematical approach** used to find the best-fitting solution by minimizing the **sum of the squared differences** (errors) between observed and predicted values.

It's widely used in **regression analysis**, curve fitting, and various optimization problems.

#### Objective Function:

$$\text{Minimize} \sum_{i=1}^n (y_i - f(x_i))^2$$

Where:

- $y_i$ : observed value
- $f(x_i)$ : predicted value
- The difference  $y_i - f(x_i)$  is called the **residual**.

#### Types:

##### 1. Linear Least Squares:

- Model:  $y = X\beta + \epsilon$
- Solution:

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

- Used in **Linear Regression**.

##### 2. Nonlinear Least Squares:

- The function  $f(x)$  is **nonlinear** in parameters.
- Solved using **iterative algorithms** like **Gauss-Newton** or **Levenberg–Marquardt**.

#### Applications:

- **Linear and polynomial regression**

- Physics/engineering model fitting
- Data approximation and smoothing
- Computer vision tasks

### Evolutionary Learning

**Evolutionary Learning** is a type of machine learning that is inspired by the principles of **natural evolution** such as **selection**, **mutation**, **recombination (crossover)**, and **survival of the fittest**. It is commonly used for **optimization problems**, especially when the search space is large, complex, or poorly understood.

It falls under the broader category of **Evolutionary Computation**, which includes algorithms like Genetic Algorithms, Genetic Programming, Evolution Strategies, and Differential Evolution.

### Core Concepts:

1. **Population-Based** **Search:**  
A population of candidate solutions (individuals) is maintained rather than a single solution.
2. **Fitness** **Evaluation:**  
Each individual is evaluated using a **fitness function**, which measures how well it solves the given problem.
3. **Selection:** **Selection:**  
Individuals with better fitness are more likely to be selected for reproduction.
4. **Reproduction and Variation:**
  - **Crossover (Recombination):** Combines parts of two parent solutions to create new offspring.
  - **Mutation:** Introduces small random changes to individuals to maintain genetic diversity.
5. **Survivor** **Selection:**  
A new generation is formed, typically by keeping the best solutions or using probabilistic rules.
6. **Termination** **Condition:**  
The algorithm stops after a fixed number of generations or when a satisfactory solution is found.

### Working Steps of Evolutionary Learning Algorithm:

1. **Initialize** a random population of solutions.
2. **Evaluate** the fitness of each individual.
3. **Select** parents based on fitness.
4. **Apply crossover and mutation** to generate offspring.
5. **Evaluate** the offspring's fitness.
6. **Replace** some or all of the population with new offspring.
7. Repeat steps 3–6 until a stopping criterion is met.

### Applications:

- Neural network optimization
- Game playing agents
- Robotics and control systems
- Automated software testing
- Symbolic regression and program synthesis

### Advantages:

- Works well with **complex, nonlinear, and multimodal functions.**
- No need for gradient information.
- Can handle **discrete and continuous variables.**
- Can escape local minima.

### Genetic Algorithms (GAs)

**Genetic Algorithms** are a type of **evolutionary algorithm** inspired by the principles of **natural selection and genetics**. They are used for solving **optimization and search problems** by evolving a population of candidate solutions.

### Key Components of Genetic Algorithms:

1. **Population:**  
A set of candidate solutions (often represented as chromosomes or strings).
2. **Chromosome** **Representation:**  
Solutions can be encoded as **binary strings, real-valued vectors, or other structured formats**, depending on the problem.
3. **Fitness** **Function:**  
Evaluates how well a solution solves the problem. Higher fitness indicates a better solution.
4. **Selection:**  
Chooses individuals (parents) based on fitness to reproduce the next generation.  
Examples: **Roulette Wheel Selection, Tournament Selection.**
5. **Crossover** **(Recombination):**  
Combines two parent chromosomes to produce new offspring.  
Types:
  - **Single-point crossover**
  - **Two-point crossover**
  - **Uniform crossover**
6. **Mutation:**  
Randomly alters genes in a chromosome to introduce diversity.  
Example: flipping a bit in binary encoding.
7. **Replacement:**  
Offspring replace some or all of the current population based on strategy (e.g., elitism, generational, steady-state).

### Genetic Algorithm Process:

1. Initialize a random population.
2. Evaluate fitness of each individual.
3. Select parents based on fitness.

4. Apply crossover and mutation to create offspring.
5. Evaluate offspring fitness.
6. Replace some population members with offspring.
7. Repeat steps 3–6 for several generations or until convergence.

#### **Advantages:**

- Works well in **large, complex, and poorly understood search spaces**.
- Does not require derivative or gradient information.
- Can find **global optima** in multimodal functions.
- Applicable to **discrete and continuous** optimization problems.

#### **Limitations:**

- May require **many evaluations** of the fitness function.
- Can converge **slowly** or to local optima.
- **Parameter tuning** (population size, crossover/mutation rate) is crucial.

#### **Genetic Offspring**

**Genetic Offspring** are the **new solutions** or individuals produced during the **reproduction process** in a Genetic Algorithm.

They result from the application of **crossover** and **mutation operators** to selected parent solutions.

#### **Details of Genetic Offspring:**

##### **1. Generated by Crossover:**

- Combines genetic information from **two parents**.
- Inherits traits from both, potentially creating better-performing solutions.

<b>Example</b>	<b>(Binary):</b>	
Parent	1:	101010
Parent	2:	110011
Offspring: 101011 (after crossover at 5th position)		

##### **2. Modified by Mutation:**

- Introduces **random changes** to ensure diversity in the population.
- Prevents **premature convergence** to local optima.

<b>Example</b>	<b>(Mutation):</b>
Offspring: 101011 → Mutated Offspring: 100011 (1st bit flipped)	

##### **3. Evaluated Using the Fitness Function:**

- If the offspring performs better than its parents, it may survive and contribute to the next generation.

#### **Role in Evolutionary Learning:**

- Offspring are the **driving force of evolution** in GAs.
- Their **quality** determines how quickly and effectively the algorithm finds optimal solutions.
- Maintaining **diversity among offspring** is key to avoiding stagnation in the search process.

## Genetic Operators

**Genetic Operators** are the core mechanisms in a **Genetic Algorithm (GA)** that simulate the process of **natural evolution**. These operators are used to create new individuals (solutions) from existing ones, guiding the search toward better solutions over successive generations.

There are three main types of genetic operators:

### **Selection**

To choose individuals (parents) from the current population based on their **fitness**, giving higher chances to better-performing individuals.

#### **Common Selection Techniques:**

- **Roulette Wheel Selection:**
  - Probability of selection is **proportional to fitness**.
  - Like spinning a wheel where each slice represents an individual's fitness share.
- **Tournament Selection:**
  - Randomly pick a group of individuals and select the **best among them**.
  - Simple and effective; allows controlling selection pressure.
- **Rank Selection:**
  - Individuals are ranked, and selection is based on **ranking** instead of raw fitness.
  - Reduces the risk of premature convergence.
- **Elitism:**
  - Automatically passes the **best individuals** to the next generation without change.

### Crossover (Recombination)

#### **Purpose:**

To combine the genetic material of two parent solutions to produce **offspring** that inherit traits from both.

#### **Types of Crossover:**

- **Single-Point Crossover:**
  - Cut the parent strings at a single point and **swap tails**.
  - Example:
    - Parent1: 110|010
    - Parent2: 001|101
    - Offspring: 110101, 001010

- **Two-Point Crossover:**
  - Swap the middle segments between two cut points.
- **Uniform Crossover:**
  - Randomly exchange genes from parents at each position with equal probability.
- **Arithmetic/Blend Crossover (for real-valued):**
  - Combine real-valued genes using a **linear combination**.

## Mutation

To introduce **random variations** in individuals, helping maintain **genetic diversity** and explore new regions of the search space.

### Types of Mutation:

- **Bit-Flip Mutation (binary encoding):**
  - Randomly flip bits from 0 to 1 or 1 to 0.
- **Swap Mutation (permutation encoding):**
  - Swap the position of two genes.
- **Gaussian Mutation (real-valued):**
  - Add a small **random value** drawn from a Gaussian distribution to genes.
- Mutation rates are kept **low** (e.g., 0.001 to 0.05) to prevent random search behavior.

### Why Genetic Operators Matter:

- They **balance exploration and exploitation**:
  - **Selection + Crossover:** Exploit current good solutions.
  - **Mutation:** Explore new areas of the solution space.
- Proper tuning of operators is crucial for **effective convergence** and avoiding **premature stagnation**.

## Using Genetic Algorithms (GAs)

Genetic Algorithms are **metaheuristic optimization techniques** that mimic the process of natural evolution. They are used when traditional optimization methods fail, especially in **complex, nonlinear, discontinuous, or high-dimensional** search spaces.

### Key Steps in Using Genetic Algorithms

1. **Problem Definition**
  - Clearly define the **optimization problem**, including:
    - **Objective function** (to be minimized or maximized)
    - **Constraints** (if any)
    - **Decision variables** (parameters to be optimized)
2. **Encoding Solutions**
  - Choose a way to represent potential solutions (**chromosomes**):
    - **Binary encoding:** e.g., 10101001

- **Real-valued encoding:** e.g., [2.3, -4.1, 1.7]
- **Permutation encoding** (e.g., for TSP): [4, 1, 3, 2]

### 3. Initialize Population

- Randomly generate a population of candidate solutions.
- Size of the population (e.g., 50–500) affects convergence speed and diversity.

### 4. Fitness Evaluation

- Define a **fitness function** that evaluates how "good" each solution is.
- May involve running simulations, calculations, or evaluating a mathematical formula.

### 5. Selection

- Choose the best individuals to become **parents**.
- Techniques: **Roulette Wheel, Tournament, Rank-based, Elitism**.

### 6. Crossover (Recombination)

- Combine parent chromosomes to produce new offspring.
- Helps mix and pass beneficial traits.
- Types: **Single-point, Two-point, Uniform, Blend (BLX- $\alpha$ )** for real numbers.

### 7. Mutation

- Apply random changes to offspring genes.
- Introduces diversity and prevents premature convergence.
- Keep mutation rate low (e.g., 0.01–0.05).

### 8. Replacement

- Decide which individuals will survive into the next generation.
- Strategies:
  - **Generational** (replace whole population)
  - **Steady-state** (replace few worst)
  - **Elitism** (keep top performers intact)

### 9. Termination Criteria

- Stop after:
  - A fixed number of generations
  - No significant improvement over time
  - Reaching a satisfactory fitness level

### 10. Return Best Solution

- Output the best-performing chromosome as the **final solution** to the problem.

## Example Applications of GAs

Field	Example Use Case
Engineering	Structural design, parameter tuning
AI/ML	Hyperparameter optimization, feature selection
Robotics	Path planning, control system optimization
Finance	Portfolio optimization, trading strategies
Bioinformatics	Gene sequence alignment, protein folding
Game Development	Evolving AI behavior, strategy design

## Advantages of Using Genetic Algorithms

- **No derivative needed** (non-gradient optimization)
- Can handle **discrete, continuous, or mixed variables**

- Effective in **multimodal and noisy environments**
- Capable of avoiding **local minima**

## Reinforcement Learning (RL)

**Reinforcement Learning** is a type of machine learning where an **agent** learns to make decisions by interacting with an **environment**, aiming to **maximize cumulative rewards** over time.

Unlike supervised learning, RL doesn't rely on labeled data — instead, the agent learns from **trial and error** through **rewards and punishments**.

---

## 💡 Key Concepts in Reinforcement Learning

### 1. Agent

The learner or decision maker (e.g., a robot, game bot, or program).

### 2. Environment

The external system with which the agent interacts.

### 3. State (S)

A representation of the environment at a specific time.

### 4. Action (A)

A move the agent can make in a given state.

### 5. Reward (R)

A scalar feedback signal that evaluates the action taken — positive for good behavior, negative for bad.

### 6. Policy ( $\pi$ )

A strategy or mapping from states to actions:

$$\pi(s) = a \backslash \pi(s) = a$$

It tells the agent what action to take in a given state.

### 7. Value Function (V)

Measures how good a state (or state-action pair) is, in terms of expected cumulative reward.

## Reinforcement Learning Cycle:

1. The agent observes the current **state**.
2. It selects an **action** based on its policy.
3. The action is executed in the environment.
4. The environment returns a **new state** and a **reward**.
5. The agent updates its knowledge (policy or value function).

This cycle continues until the task ends or a stopping condition is reached.

### Types of Reinforcement Learning

#### a) Model-Free vs Model-Based

- **Model-Free:** Learns directly from interaction (e.g., Q-learning, SARSA).
- **Model-Based:** Builds a model of the environment to simulate outcomes and plan actions.

#### b) Value-Based vs Policy-Based

- **Value-Based:** Learns the value of states or actions (e.g., Q-learning).
- **Policy-Based:** Directly learns the policy (e.g., REINFORCE).
- **Actor-Critic:** Combines both value and policy learning.

### Popular Algorithms

Algorithm	Type	Description
Q-Learning	Model-free	Learns action-value function $Q(s, a)$
SARSA	Model-free	Similar to Q-learning but on-policy
DQN	Deep RL	Deep Q-Network using neural networks
Policy Gradient	Policy-based	Learns policy directly
A3C, PPO	Actor-Critic	Combines value and policy estimation

### Example: "Getting Lost" Problem

A simple RL example where a robot tries to find its way out of a maze:

- It starts at a random location (state).
- Moves in directions (actions).
- Gets negative reward for hitting walls, positive reward for reaching the goal.
- Through trial and error, it **learns an optimal path** over time.

This example demonstrates key RL ideas like:

- Exploration vs. exploitation
- Delayed rewards
- Policy improvement

### Advantages of Reinforcement Learning

- Learns from **experience**, not explicit instruction.
- Can solve **complex sequential decision problems**.
- Highly effective in **robotics, games, and real-time systems**.

## Challenges

- **Exploration vs Exploitation:** Balancing trying new actions vs. using known good ones.
- **Sparse or Delayed Rewards:** Learning can be slow if feedback is rare or delayed.
- **Computational cost:** Especially in deep RL with neural networks.
- May get stuck in **local optima** or overfit to certain behaviors.

## Applications

- **Game playing** (e.g., AlphaGo, Atari games)
- **Robotics** (motion control, navigation)
- **Self-driving cars**
- **Recommendation systems**
- **Finance and trading bots**

## "Getting Lost" Example

The “**Getting Lost**” example is a classic introductory problem used to demonstrate how reinforcement learning works through trial and error.

### Scenario:

A **robot (agent)** is placed in a maze-like environment and has to **find its way to a goal (exit)**. Initially, it has **no map** or idea about the layout.

### Agent's Actions:

- Move **up, down, left, or right**.

## Markov – Overview

The term **Markov** comes from **Andrey Markov**, a Russian mathematician, and is most commonly associated with **Markov Processes**, which are used to model **random systems that undergo transitions** from one state to another over time.

In the context of **Reinforcement Learning** and **probabilistic modeling**, the **Markov Property** and **Markov Decision Processes (MDPs)** are fundamental.

## Markov Property

A system satisfies the **Markov Property** if the **future state** depends **only on the current state**, and **not on the sequence of events** that preceded it.

### **Formal Definition:**

A process has the **Markov Property** if:

$$P(S_{t+1}|S_t, S_{t-1}, \dots, S_0) = P(S_{t+1}|S_t)P(S_{\{t+1\}} | S_t, S_{\{t-1\}}, \dots, S_0) = P(S_{\{t+1\}} | S_t)$$

This means the next state  $S_{t+1}$  is **conditionally independent** of the past states given the current state  $S_t$ .

### **Markov Chain**

A **Markov Chain** is a stochastic model that describes a sequence of states with the **Markov Property**.

#### **Characteristics:**

- Finite set of **states**.
- A **transition probability matrix** defines the probability of moving from one state to another.
- No **actions** or **rewards** (unlike MDPs).

#### **Example:**

Suppose you are modeling weather:

#### **Current State Next State Probability**

Sunny	Sunny	0.8
Sunny	Rainy	0.2
Rainy	Sunny	0.4
Rainy	Rainy	0.6

### **Markov Decision Process (MDP)**

An **MDP** is an extension of a Markov Chain that includes **actions** and **rewards**, making it suitable for **Reinforcement Learning**.

#### **Defined by a 5-tuple:**

$$\text{MDP} = (S, A, P, R, \gamma) \quad \text{MDP} = (S, A, P, R, \gamma)$$

#### **Element Description**

**S** Set of states

**A** Set of actions

**P** Transition probability:  $P(s'|s, a)$

**R** Reward function:  $R(s, a, s')$

### Element Description

$\gamma$  Discount factor ( $0 < \gamma < 1$ )

### Why Markov Models Matter in AI/ML:

- They **simplify the modeling** of sequential decision-making problems.
- Core of **Reinforcement Learning algorithms**.
- Used in **robotics, game AI, recommendation systems**, etc.

### Decision Process – Overview

A **Decision Process** is a mathematical model used to represent **sequential decision-making problems**. In such problems, an agent interacts with an environment over time, **choosing actions** based on the current state, and **receiving rewards** based on those actions.

In the context of machine learning and artificial intelligence, this concept is formalized as a **Markov Decision Process (MDP)**.

### Markov Decision Process (MDP)

A **Markov Decision Process** is a framework for modeling situations where outcomes are partly random and partly under the control of a decision-maker (agent).

### Components of an MDP:

An MDP is defined by a **5-tuple**:

$$\text{MDP} = (S, A, P, R, \gamma)$$

Component	Description
S (States)	Set of all possible states the agent can be in.
A (Actions)	Set of actions the agent can take.
P (Transition Probability)	$P(s')$
R (Reward Function)	$R(s, a, s')$ : expected reward received after transitioning from $s$ to $s'$ using action $a$ .
$\gamma$ (Discount Factor)	A value $0 < \gamma \leq 1$ that determines how future rewards are valued relative to immediate rewards.

### How It Works (Decision Process Flow):

1. The agent **observes the current state**  $s$ .
2. It chooses an **action**  $a$  based on a policy  $\pi$ .
3. The environment responds with a **reward**  $R$  and transitions to a **new state**  $s'$ .
4. The agent updates its knowledge to improve future decisions.
5. This process repeats for each time step.

## Goal of the Agent:

To learn an **optimal policy**  $\pi^*$  that maximizes the **expected cumulative reward** over time:

$$\pi^* = \arg\max_{\pi} \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)]$$

## Example: Robot Navigation

- **States:** Each cell in a grid.
- **Actions:** Move Up, Down, Left, Right.
- **Transition:** Moving from one cell to another (with some probability of error).
- **Reward:** +10 for reaching goal, -1 for hitting wall, 0 otherwise.
- **Objective:** Find a path that maximizes reward (fastest to goal with fewest penalties).

## Applications of Decision Processes:

Domain	Use Case
Robotics	Motion planning, navigation
Healthcare	Treatment planning over time
Game AI	Decision-making in strategic games
Finance	Investment decisions, trading bots
Operations Research	Inventory control, scheduling