Artificial Intelligence

5 Units

Complete Notes

If you find these notes helpful, please follow me on LinkedIn

Sarthak Sobti

Click on the link below:

www.linkedin.com/in/**sarthak-sobti-65bab51b4**

Artificial Intelligence

Unit 1

1. Introduction to Artificial Intelligence

- Artificial Intelligence (AI) refers to the simulation of human intelligence in machines that are programmed to think and act like humans.

- It involves the development of algorithms and computer programs that can perform tasks that typically require human intelligence such as visual perception, speech recognition, decision-making, and language translation.

- The focus of artificial intelligence is towards understanding human behaviour and performance.

- This can be done by creating computers with human-like intelligence and capabilities.

- This includes natural language processing, facial analysis and robotics.

---

2. Applications of Artificial Intelligence

1. Artificial Intelligence in E-Commerce

- Artificial Intelligence helps to make appropriate suggestions and recommendations as per the user search history and view preferences.
- There are also AI chatbots that are used to provide customer support instantly and help to reduce complaints and queries to a great extent.

2. AI in Education Purpose

- It helps the faculty as well as the students by making course recommendations, Analysing some data and some decisions about the student, etc.
- Making automated messages to the students, and parents regarding any vacation, and test results are done by Artificial Intelligence these days.

3. Healthcare

- Artificial Intelligence uses the medical history and current situation of a particular human being to predict future diseases.

- Artificial Intelligence is also used to find the current vacant beds in the hospitals of a city that saves the time of patients who are in emergency conditions.

4. Automobiles

- AI is used to detect the traffic on the street and provide the best route out of the present all routes to the driver.
- It uses sensors, GPS technology, and control signals to bring the vehicle the best path.

5. Surveillance

- Artificial intelligence is also used in the field of surveillance by recognizing far faces and objects.
- Then the event recognition capabilities are used to enhance these faces and objects. This helps the military to protect their areas and prevent any attack in real time.

---

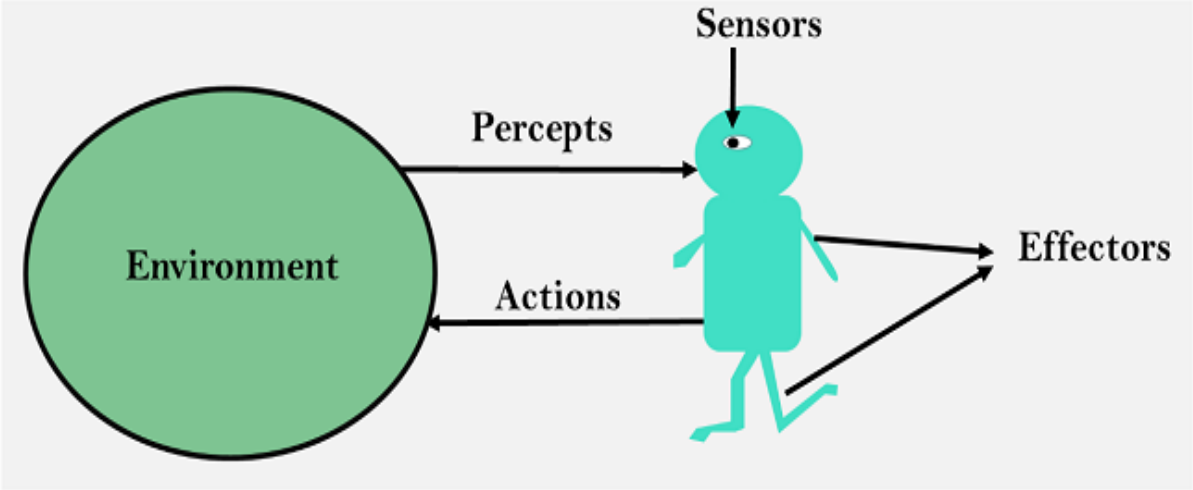
3. Intelligent Agents and its Structure

1. Agents

An agent can be anything that perceive its environment through sensors and act upon that environment through actuators. An Agent runs in the cycle of perceiving, thinking, and acting.

2. Basic Terminology :

- **Sensor**: Sensor is a device which detects the change in the environment and sends the information to other electronic devices. An agent observes its environment through sensors.
- **Actuators**: Actuators are the component of machines that converts energy into motion. The actuators are only responsible for moving and controlling a system. An actuator can be an electric motor, gears, rails, etc.
- **Effectors**: Effectors are the devices which affect the environment. Effectors can be legs, wheels, arms, fingers, wings, fins, and display screen.



3. Intelligent Agents

An intelligent agent is an autonomous entity which act upon an environment using sensors and actuators for achieving goals. An intelligent agent may learn from the environment to achieve their goals. A thermostat is an example of an intelligent agent.

Following are the main four rules for an AI agent:

- **Rule 1**: An AI agent must have the ability to perceive the environment.
- **Rule 2**: The observation must be used to make decisions.
- **Rule 3**: Decision should result in an action.
- **Rule 4**: The action taken by an AI agent must be a rational action.

4. Structure of AI Agent

To understand the structure of Intelligent Agents, we should be familiar with Architecture and Agent programs.

- **Architecture** is the machinery that the agent executes on. It is a device with sensors and actuators, for example, a robotic car, a camera, and a PC.

- **Agent function** is used to map a percept sequence to an action.

- An **agent program** is an implementation of an agent function.

$$Agent = Architecture + Agent\ Program$$

5. Types of AI Agents

1. **Simple Reflex agent**

- The Simple reflex agents are the simplest agents. These agents take decisions on the basis of the current percepts and ignore the rest of the percept history.
- These agents only succeed in the fully observable environment.
- The Simple reflex agent does not consider any part of percepts history during their decision and action process.
- The Simple reflex agent works on Condition-action rule, which means it maps the current state to action. Such as a Room Cleaner agent, it works only if there is dirt in the room.
- For example, a thermostat turns the heater on or off based on the current temperature.

2. **Model-based reflex agent**

- The Model-based agent can work in a partially observable environment and track the situation.
- It maintains an internal state that keeps track of part of the history.
- It uses “a model of the world” to handle more complex environments.
- It considers updating their internal state based on how their actions change the environment.
- For example, a robot vacuum which remembers where it has already cleaned to avoid redundant cleaning.

3. **Goal-based agents**

- The knowledge of the current state environment is not always sufficient to decide for an agent to what to do.
- The agent needs to know its goal which describes desirable situations.
- Goal-based agents expand the capabilities of the model-based agent by having the "goal" information.
- They choose an action, so that they can achieve the goal.
- For example, a GPS navigation system plans the best route to a destination.

4. **Utility-based agents**

- These agents are similar to the goal-based agent but provide an extra component of utility measurement which makes them different by providing a measure of success at a given state.
- Utility-based agent act based on not only goals but also the best way to achieve the goal.
- The Utility-based agent is useful when there are multiple possible alternatives, and an agent has to choose in order to perform the best action.
- For example, an investment advisor chooses investment strategies based on risk and return to maximize profit.

5. **Learning Agents**

- A learning agent in AI is the type of agent which can learn from its past experiences, or it has learning capabilities.
- It starts to act with basic knowledge and then able to act and adapt automatically through learning.
- Learning agents are able to learn, analyze performance, and look for new ways to improve the performance.
- For example, a recommendation system learns user preferences to provide better recommendations over time.

---

4. Understanding PEAS

PEAS System is used to categorize similar agents together. The PEAS system delivers the performance measure with respect to the environment, actuators, and sensors of the respective agent. Most of the highest performing agents are Rational Agents.

Rational Agent: The rational agent considers all possibilities and chooses to perform a highly efficient action. For example, it chooses the shortest path with low cost for high efficiency. PEAS stands for a Performance measure, Environment, Actuator, Sensor:

1. **Performance Measure**: Performance measure is the unit to define the success of an agent. Performance varies with agents based on their different precepts.

2. **Environment**: Environment is the surrounding of an agent at every instant. It keeps changing with time if the agent is set in motion. There are 5 major types of environments:
    - Fully Observable & Partially Observable
    - Episodic & Sequential
    - Static & Dynamic
    - Discrete & Continuous
    - Deterministic & Stochastic

3. **Actuator**: An actuator is a part of the agent that delivers the output of action to the environment.

4. **Sensor**: Sensors are the receptive parts of an agent that takes in the input for the agent.

Agent	Performance Measure	Environment	Actuator	Sensor
Hospital Management System	Patient's health, Admission process, Payment	Hospital, Doctors, Patients	Prescription, Diagnosis, Scan report	Symptoms, Patient's response
Automated Car Drive	The comfortable trip, Safety, Maximum Distance	Roads, Traffic, Vehicles	Steering wheel, Accelerator, Brake, Mirror	Camera, GPS, Odometer
Subject Tutoring	Maximize scores, Improvement is students	Classroom, Desk, Chair, Board, Staff, Students	Smart displays, Corrections	Eyes, Ears, Notebooks
Part-picking robot	Percentage of parts in correct bins	Conveyor belt with parts; bins	Jointed arms and hand	Camera, joint angle sensors
Satellite image analysis system	Correct image categorization	Downlink from orbiting satellite	Display categorization of scene	Color pixel arrays

5. Computer Vision

Computer vision is a field of artificial intelligence (AI) that uses machine learning and neural networks to teach computers and systems to derive meaningful information from digital images, videos and other visual inputs—and to make recommendations or take actions when they see defects or issues.

Computer vision teaches computers to see by showing them lots of pictures. To recognize things like car tires, they need to study many tire images and learn the differences. The more they practice, the better they get at spotting tires, even ones with defects.

Here are some examples of computer vision:

- **Facial recognition**: Identifying individuals through visual analysis.
- **Self-driving cars**: Using computer vision to navigate and avoid obstacles.
- **Robotic automation**: Enabling robots to perform tasks and make decisions based on visual input.

---

5. Natural Language Processing

- Natural Language Processing (NLP) is a subfield of artificial intelligence that deals with the interaction between computers and humans in natural language.

- It involves the use of computational techniques to process and analyse natural language data, such as text and speech, with the goal of understanding the meaning behind the language.

- NLP is used in a wide range of applications, including machine translation, sentiment analysis, speech recognition, chatbots, and text classification.

Some common techniques used in NLP include:

1. **Tokenization**: the process of breaking text into individual words or phrases.

2. **Part-of-speech tagging**: the process of labelling each word in a sentence with its grammatical part of speech.

3. **Named entity recognition**: the process of identifying and categorizing named entities, such as people, places, and organizations, in text.

4. **Sentiment analysis**: the process of determining the sentiment of a piece of text, such as whether it is positive, negative, or neutral.

5. **Machine translation**: the process of automatically translating text from one language to another.

6. **Text classification**: the process of categorizing text into predefined categories or topics.

Unit 2

1. Search Algorithms

Artificial Intelligence is the study of building agents that act rationally. Most of the time, these agents perform some kind of search algorithm in the background in order to achieve their tasks.

- A search problem consists of:
  - **A State Space**. Set of all possible states where you can be.
  - **A Start State**. The state from where the search begins.
  - **A Goal State**. A function that looks at the current state returns whether or not it is the goal state.

- The Solution to a search problem is a sequence of actions, called the plan that transforms the start state to the goal state.

- This plan is achieved through search algorithms.

- Based on the search problems we can classify the search algorithms into **Uninformed Search (Blind Search)** and **Informed Search (Heuristic Search)** algorithms.

- The 5 components in a Search Algorithm are:

    - **STATE**: state in the state space to which the node corresponds
    - **PARENT-NODE**: the node in the search tree that generated this node
    - **ACTION**: the action that was applied to the parent to generate the node
    - **PATH-COST**: the cost of the path from the initial state to the node.
    - **DEPTH**: the number of steps along the path from the initial state

---

2. Evaluation

Search Trees are evaluated based on :

- **Completeness**: does it always find a solution if one exists
- **Time complexity**: No. of nodes generated.
- **Space complexity**: maximum number of nodes in memory
- **Optimality**: does it always find a least cost solution
- **Systematicity**: does it visit each state at most once.

---

3. Uninformed Search

- Uninformed search in AI refers to a type of search algorithm that does not use additional information to guide the search process.

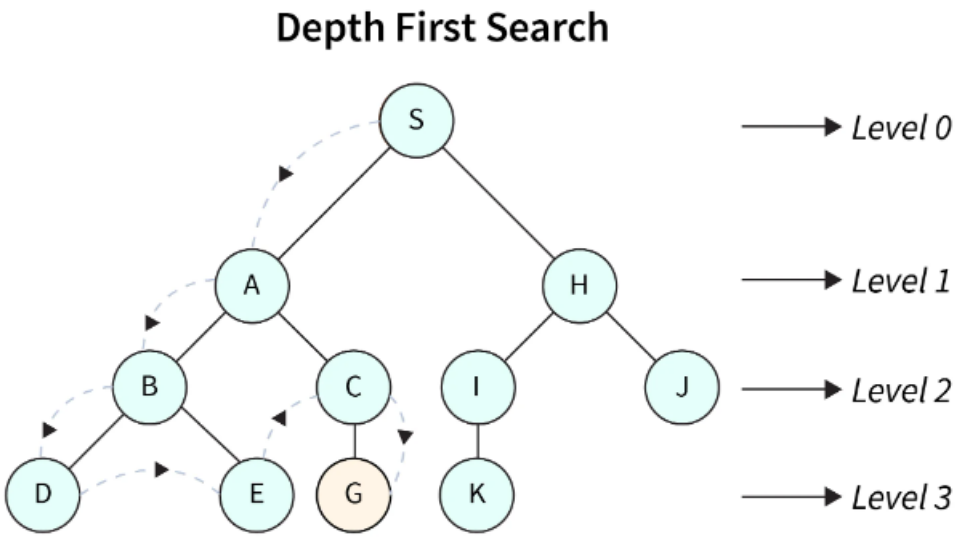
- Instead, these algorithms explore the search space in a systematic, but blind, manner without considering the cost of reaching the goal or the likelihood of finding a solution.

- Examples of uninformed search algorithms include -
  - Depth First Search
  - Breadth First Search
  - Uniform Cost Search

1. Depth First Search

Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node and goes through the branch nodes and then returns. It is implemented using a stack data structure that works on the concept of last in, first out (LIFO).

As DFS traverses the tree “deepest node first”, it would always pick the deeper branch until it reaches the solution.



m = max. depth of the search tree = the number of levels of the search tree.

b = branching factor (average no. of successors/children each node has)

Time complexity: Equivalent to the number of nodes traversed in DFS.

$$O(b^m)$$

Space complexity: Equivalent to how large can the fringe get.

$$O(b \times m)$$

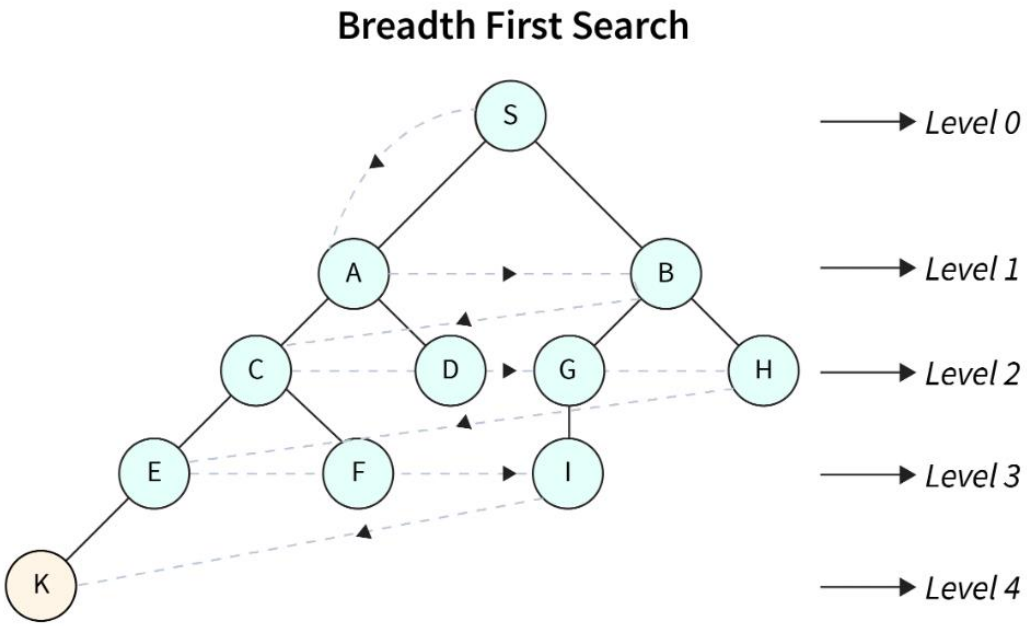
Completeness: DFS is complete if the search tree is finite, meaning for a given finite search tree, DFS will come up with a solution if it exists.

Optimality: DFS is not optimal, meaning the number of steps in reaching the solution, or the cost spent in reaching it is high.

2. Breadth First Search

Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root and explores all of the neighbour nodes at the present depth prior to moving on to the nodes at the next depth level. It uses the queue data structure that works on the first in, first out (FIFO) concept. It is a complete algorithm as it returns a solution if a solution exists.

As BFS traverses the tree “shallowest node first”, it would always pick the shallower branch until it reaches the solution.



d = the depth of the shallowest solution.

b = branching factor (average no. of successors/children each node has)

Time complexity: Equivalent to the number of nodes traversed in BFS until the shallowest solution.

$$O(b^d)$$

Space complexity: Equivalent to how large can the fringe get.

$$O(b^d)$$

Completeness: BFS is complete, meaning for a given search tree, BFS will come up with a solution if it exists.

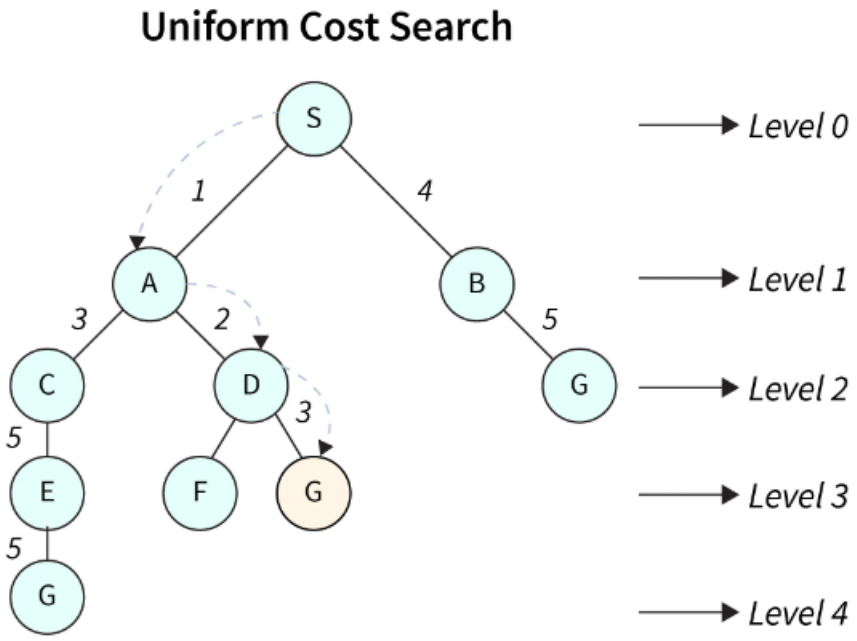
Optimality: BFS is optimal as long as the costs of all edges are equal.

3. Uniform Cost Search

UCS is different from BFS and DFS because here the costs come into play. In other words, traversing via different edges might not have the same cost. The goal is to find a path where the cumulative sum of costs is the least.

Cost of a node is defined as:

```
cost(node) = cumulative cost of all nodes from root
cost(root) = 0
```



The cost of each node is the cumulative cost of reaching that node from the root. Based on the UCS strategy, the path with the least cumulative cost is chosen. Note that due to the many options in the fringe, the algorithm explores most of them so long as their cost is low and discards them when a lower-cost path is found; these discarded traversals are not shown below.

Let C = cost of solution.

ε = arcs cost.

Then $\frac{C}{\varepsilon}$ = effective depth

Time complexity: $O(b^{\frac{C}{\varepsilon}})$

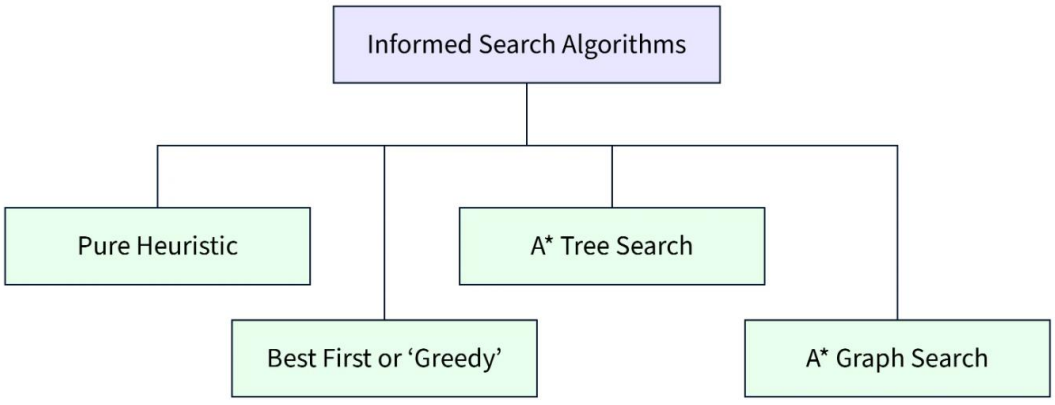
Space complexity: $O(b^{\frac{C}{\varepsilon}})$

---

4. Informed Search

- Informed search algorithms are a type of search algorithm that uses heuristic functions to guide the search process. For example, a heuristic function calculates the cost of moving from a starting state to a goal state.
- Informed search algorithms can select search paths more likely to lead to the desired state.

- As a result, the search process is improved, making it quicker and more accurate for AI systems to make decisions.

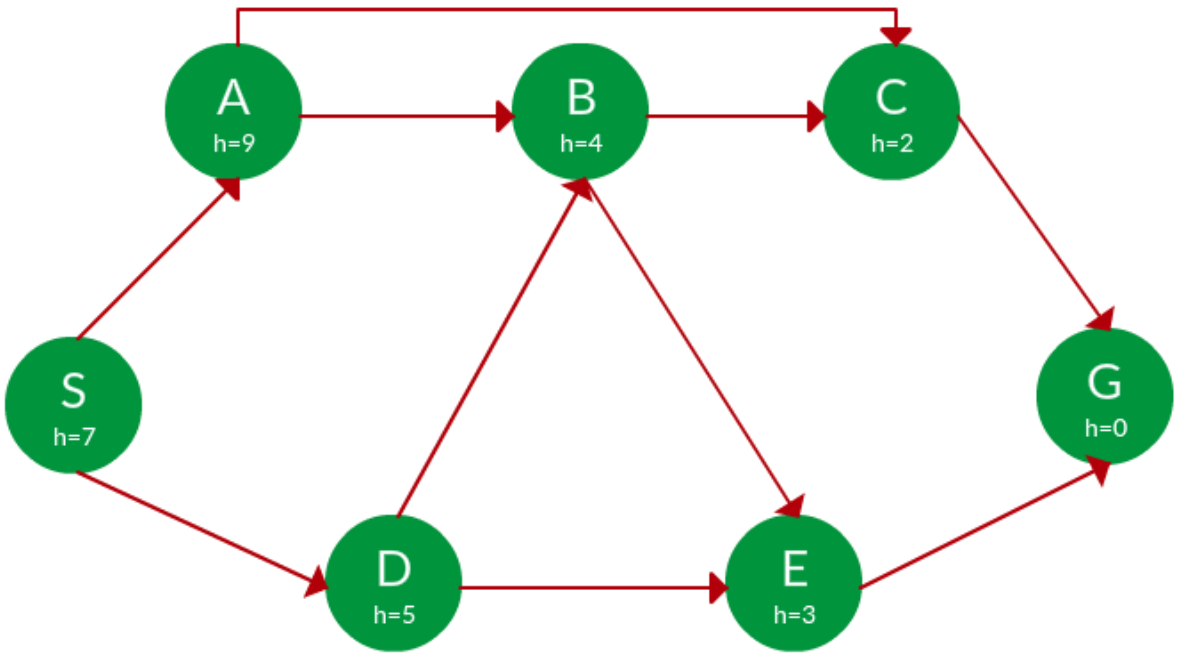


1. Greedy Best First Search

- A heuristic function calculates the cost of moving from a starting state to a goal state.
- In search algorithms, heuristics come in two types: acceptable and inadmissible.
- An acceptable heuristic never overestimates the cost to reach the goal. It always gives a lower bound on the cost.
- In contrast, an inadmissible heuristic may overestimate the cost, leading to suboptimal solutions.
- In greedy search, we expand the node closest to the goal node. The "closeness" is estimated by a heuristic h(x).
- A heuristic h is defined as-
  h(x) = Estimate of distance of node x from the goal node.
  Lower the value of h(x), closer is the node from the goal.
- **Strategy**: Expand the node closest to the goal state, i.e. expand the node with a lower h value.

Example:

Question. Find the path from S to G using greedy search. The heuristic values h of each node below the name of the node.



Solution. Starting from S, we can traverse to A(h=9) or D(h=5). We choose D, as it has the lower heuristic cost. Now from D, we can move to B(h=4) or E(h=3). We choose E with a lower heuristic cost. Finally, from E, we go to G(h=0).

Path: S -> D -> E -> G

Advantage: Works well with informed search problems, with fewer steps to reach a goal.

Disadvantage: Can turn into unguided DFS in the worst case.

Time complexity: The worst-case time complexity of the Greedy best-first search in artificial intelligence is $O(b^m)$.

Space Complexity: The worst-case space complexity of Greedy best-first search in artificial intelligence is $O(b^m)$, where m is the maximum depth of the search space.

Optimal: Not complete, hence not optimal.

2. A* Tree Search

- A* Tree Search, known as A* Search, combines the strengths of uniform-cost search and greedy search.

- To find the best path from the starting state to the desired state, the A* search algorithm investigates all potential paths in a graph or grid. The algorithm calculates the cost of each potential move at each stage using the following two criteria:
  - How much it costs to reach the present node?
  - The approximate cost from the present node to the goal.

- A heuristic function is used to determine the estimated cost and estimate the distance between the current node and the desired state.

  The following points should be noted wrt. heuristics in A* search.

$$f(x) = g(x) + h(x)$$

  Here, $h(x)$ is called the **forward cost** and is an estimate of the distance of the current node from the goal node.

  And $g(x)$ is called the **backward cost** and is the cumulative cost of a node from the root node.

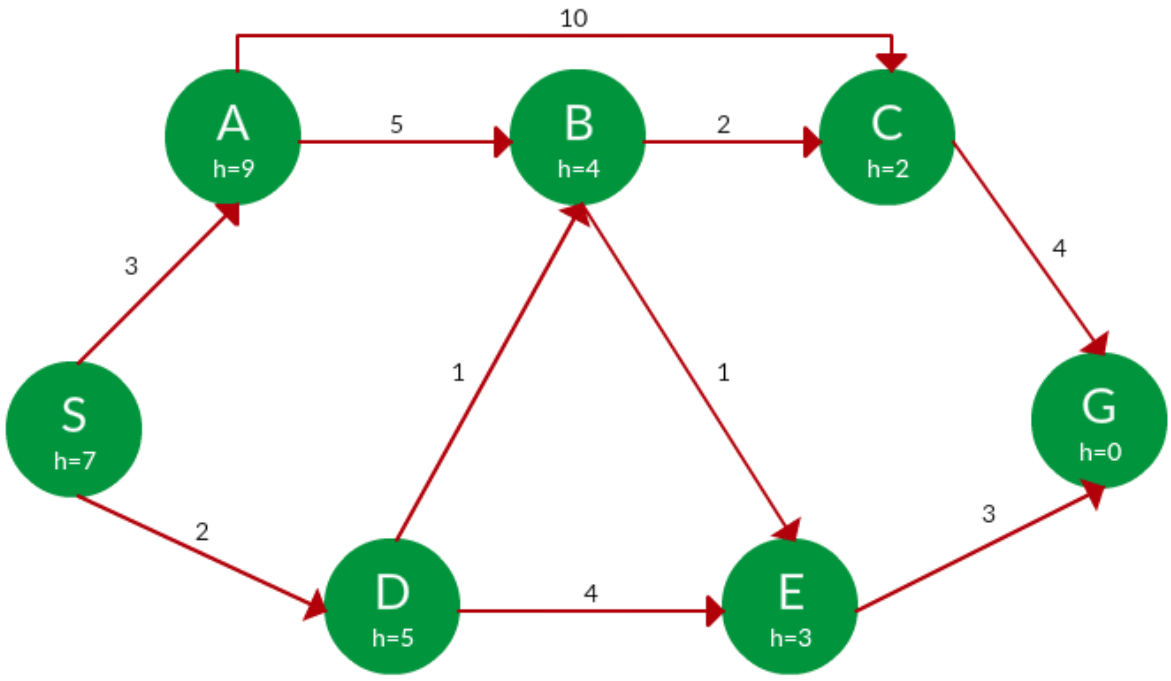
- **Strategy**: Choose the node with the lowest f(x) value.

Example:

Question. Find the path to reach from S to G using A* search.

Solution. Starting from S, the algorithm computes g(x) + h(x) for all nodes in the fringe at each step, choosing the node with the lowest sum. The entire work is shown in the table below.

Note that in the fourth set of iterations, we get two paths with equal summed cost f(x), so we expand them both in the next set. The path with a lower cost on further expansion is the chosen path



Path	h(x)	g(x)	f(x)
S	7	0	7
S -> A	9	3	12
S -> D	5	2	7
S -> D -> B	4	2 + 1 = 3	7
S -> D -> E	3	2 + 4 = 6	9
S -> D -> B -> C	2	3 + 2 = 5	7
S -> D -> B -> E	3	3 + 1 = 4	7
S -> D -> B -> C -> G	0	5 + 4 = 9	9
S -> D -> B -> E -> G	0	4 + 3 = 7	7

Path: S -> D -> B -> E -> G

Cost: 7

Complete: Yes

Optimal: Depends on two properties of the heuristic function -

- Admissibility
- Consistency

1. **Admissible Heuristics**:

- A* search is optimal only when for all nodes, the forward cost for a node h(x) underestimates the actual cost h*(x) to reach the goal. This property of A* heuristic is called admissibility.

- **Admissibility**:

$$0 \leq h(x) \leq h^*(x)$$

- A heuristic function h(n) is admissible for every node n, if -

$$h(n) \leq h^*(n)$$

$h(n)$ is the estimated cost to reach the goal from n

$h^*(n)$ is the true cost to reach the goal from n

- An admissible Heuristic never overestimate the cost to reach the goal i.e., it is optimal.

If $h(n)$ is admissible, A* is optimal.

2. **Consistent Heuristic**:

- $h(n)$ is consistent if for every node n and for every successor node n, the triangular inequality (each side of the triangle

cannot be more than the sum of the other two sides) is satisfied -

$$h(n) \leq c(n, a, n') + h(n')$$

$h(n)$ = estimated cost of reaching the goal from n

$c(n, a, n')$ = step cost of getting to n'

$h(n')$ = estimated cost of reaching the goal from n'

- Every heuristic which is consistent is also admissible.

  If $h(n)$ is consistent, A* is cost optimal.

Disadvantage: It keeps all the generated nodes in memory. Therefore, runs out of space long before it runs out of time. As a result, even if it finds optimal solution, it is not practical for many large-scale problems.

---

5. Local Search

- A **local search algorithm** in artificial intelligence works by starting with an initial solution and then making minor adjustments to it in the hopes of discovering a better one.

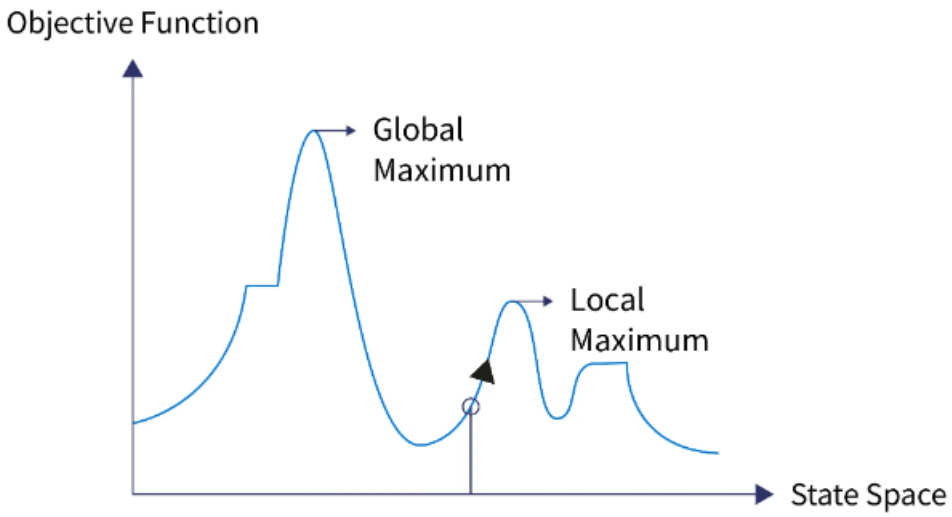
- Every time the algorithm iterates, the current solution is assessed, and a small modification to the current solution creates a new solution.

- The current solution is then compared to the new one, and if the new one is superior, it replaces the old one.

- This process keeps going until a satisfactory answer is discovered or a predetermined stopping criterion is satisfied.

There are several types of local search algorithms, such as Hill Climbing, Local Beam Search, Travelling Salesman Problem, etc.

Hill Climbing (Greedy Local Search or Steepest Ascent)



A one-dimensional state-space landscape in which elevation corresponds to the objective function

The term "hill climbing" refers to the analogous activity of ascending a hill to reach its summit (the optimal solution). The algorithm operates by making incremental progress (or "climbs") towards increasing values of the objective function up until it hits a local maximum beyond which no additional gains are possible.

The basic steps of the hill climbing algorithm are as follows:

1. Start with an initial solution.
2. Evaluate the objective function to determine the quality of the solution.
3. Generate a set of candidate solutions by making small modifications to the current solution.
4. Evaluate the objective function for each candidate solution.
5. Select the candidate solution that improves the objective function the most.
6. Repeat steps 3-5 until no further improvement can be made.

Disadvantages:

1. Hill Climbing can get stuck in local optima, meaning that it may not find the global optimum of the problem.

2. The algorithm is sensitive to the choice of initial solution, and a poor initial solution may result in a poor final solution.

3. It may be less effective than other optimization algorithms, such as genetic algorithms or simulated annealing, for certain types of problems.

---

6. Adversarial Search (Game Playing)

Adversarial search is used in scenarios where two or more players are competing against each other, like in games. The goal is to find the best move for a player, considering that the opponent is also trying to make the best moves to win. Imagine you're playing a game of chess. You want to make a move that puts you in the best position to win, but you also need to think about how your opponent will respond. Adversarial search helps you decide on the best move by considering both your actions and your opponent's actions.

1. Key Concepts

1. **Multi-agent Environment**: Unlike single-agent search problems, adversarial search involves two or more agents with conflicting goals, often seen in games like chess, tic-tac-toe, and checkers.

2. **Game Theory**: Adversarial search is based on game theory, where each player's goal is to maximize their own benefits while minimizing the opponent's.

3. **Zero-Sum Games**: Many adversarial search problems are zero-sum games, where one player's gain is exactly balanced by the other player's loss. Examples include chess and tic-tac-toe.

4. **Game Tree**: A game tree represents all possible moves in a game. Nodes in the tree represent game states, and edges represent player moves. Algorithms like Minimax and Alpha-Beta Pruning traverse this tree to determine optimal moves.

5. **Elements of Game Formalization**:
    - **Initial State**: The starting configuration of the game.
    - **Player(s)**: The agents participating in the game.
    - **Actions**: Possible moves a player can make.
    - **Result(s, a)**: The outcome of a move from a given state.
    - **Terminal-Test(s)**: A test to check if the game is over.
    - **Utility(s, p)**: The payoff or value of the game outcome for a player.

2. Minimax Algorithm

The Minimax algorithm is used in decision-making and game theory to find the optimal move for a player, assuming that the opponent also plays optimally. Here's how it works in simple terms:

1. **Two Players**: Think of a two-player game like chess or tic-tac-toe, where one player tries to maximize their score (let's call this player "Max") and the other tries to minimize it ("Min").

2. **Game Tree**: Imagine a tree where each node represents a game state. The root is the current game state, and each branch represents a possible move.

3. **Simulate Moves**:
    - Max makes a move, then Min responds with their move, then Max again, and so on.
    - This continues until the game reaches a terminal state (end of the game).

4. **Score Terminal States**:
    - At the end of the game, assign a score based on who wins. For example, +1 for a win, -1 for a loss, and 0 for a draw.

5. **Backpropagate Scores**:
    - Starting from the terminal states, move up the tree:
        - If it's Max's turn, choose the move with the highest score.
        - If it's Min's turn, choose the move with the lowest score.
    - This way, each node gets a score representing the best possible outcome from that position, assuming both players play optimally.

6. **Optimal Move**:
    - At the root node (current game state), Max chooses the move that leads to the highest score.

Time complexity: $O(b^m)$

Space complexity: $O(bm)$

b – denotes the legal moves available at each point,

m – denotes the maximum depth of the tree.

3. Alpha-Beta Pruning

Alpha-Beta Pruning is an optimization for the Minimax algorithm. It reduces the number of nodes evaluated in the game tree, making the algorithm faster without affecting the final decision.

1. **Alpha and Beta Values**:
    - **Alpha**: The best value that Max (the maximizer) can guarantee at that level or above.
    - **Beta**: The best value that Min (the minimizer) can guarantee at that level or below.

2. **Pruning**:
    - As you evaluate the game tree, you keep track of these alpha and beta values.

- **Prune** (cut off) branches that can't possibly influence the final decision. If you find a branch where:
    - **Max's Turn**: The current score is greater than or equal to Beta, stop exploring that branch.
    - **Min's Turn**: The current score is less than or equal to Alpha, stop exploring that branch.

Example

Let's add Alpha-Beta Pruning to our tic-tac-toe example:

1. **Set Initial Alpha and Beta**:
    - Alpha = $-\infty$ (worst case for Max)
    - Beta = $+\infty$ (worst case for Min)
2. **Traverse the Tree**:
    - As you simulate moves, update Alpha and Beta.
    - If Max finds a move with a value >= Beta, prune the rest of the moves at that level.
    - If Min finds a move with a value <= Alpha, prune the rest of the moves at that level.

Unit 3

1. Knowledge Representation and Reasoning

- Knowledge representation and reasoning (KR, KRR) is the part of Artificial intelligence which concerned with AI agents thinking and how thinking contributes to intelligent behaviour of agents.

- It is responsible for representing information about the real world so that a computer can understand and can utilize this knowledge to solve the complex real-world problems such as diagnosis a medical condition or communicating with humans in natural language.

- It is also a way which describes how we can represent knowledge in artificial intelligence.

- Knowledge representation is not just storing data into some database, but it also enables an intelligent machine to learn from that knowledge and experiences so that it can behave intelligently like a human.

---

2. Propositional Logic

Propositional logic (PL) is the simplest form of logic where all the statements are made by propositions. A proposition is a declarative statement which is either true or false. It is a technique of knowledge representation in logical and mathematical form.

Example:

a) It is Sunday.

b) The Sun rises from West (False proposition)

c) 3 + 3 = 7 (False proposition)

d) 5 is a prime number.

There are two types of Propositions:

a. **Atomic Propositions**

b. **Compound propositions**

- **Atomic Proposition:** Atomic propositions are the simple propositions. It consists of a single proposition symbol. These are the sentences which must be either true or false.

  Example:

  a) 2 + 2 is 4, it is an atomic proposition as it is a **true** fact.

  b) "The Sun is cold" is also a proposition as it is a **false** fact.

- **Compound proposition:** Compound propositions are constructed by combining simpler or atomic propositions, using parenthesis and logical connectives.

  Example:

  a) "It is raining today, and street is wet."

  b) "Ankit is a doctor, and his clinic is in Mumbai."

---

3. Theory of First Order Logic (FOL)

First-order logic extends propositional logic by incorporating quantifiers and predicates, allowing for more expressive statements about the world. The key components of FOL include constants, variables, predicates, functions, quantifiers, and logical connectives.

1. **Constants**: Constants represent specific objects within the domain of discourse. For example, in a given domain, Alice, 2, and New York could be constants.

2. **Variables**: Variables stand for placeholders for objects in the domain. Commonly used symbols for variables include x, y, and z.

3. **Predicates**: Predicates are functions that return true or false, representing properties of objects or relationships between them. For example, $Likes(Alice, Bob)$ indicates that Alice likes Bob, and $GreaterThan(x, 2)$ means that x is greater than 2.

4. **Functions**: Functions map objects to other objects. For instance, $MotherOf(x)$ might denote the mother of x.

5. **Quantifiers**: Quantifiers specify the scope of variables. The two main quantifiers are:

   - **Universal Quantifier (∀)**: Indicates that a predicate applies to all elements in the domain.

     For example, ∀x (Person(x) → Mortal(x)) means "All persons are mortal."

   - **Existential Quantifier (∃)**: Indicates that there is at least one element in the domain for which the predicate holds.

     For example, ∃x (Person(x) ∧ Likes(x, IceCream)) means "There exists a person who likes ice cream."

6. **Logical Connectives**: Logical connectives include conjunction (∧), disjunction (∨), implication (→), biconditional (↔), and negation (¬). These connectives are used to form complex logical statements.

---

4. Logical Connectives

Logical connectives are used to connect two simpler propositions or representing a sentence logically. We can create compound propositions with the help of logical connectives. There are mainly five connectives, which are given as follows:

1. AND (∧) or Conjunction

- The AND connective is true if and only if both propositions it connects are true.

- Example:
    - Let P = "It is raining."
    - Let Q = "I have an umbrella."
    - "P ∧ Q" means "It is raining AND I have an umbrella."

2. OR (∨) or Disjunction

- The OR connective is true if at least one of the propositions it connects is true.

- **Example**:
    - Let P = "It is raining."
    - Let Q = "I have an umbrella."
    - "P ∨ Q" means "It is raining OR I have an umbrella."

3. NOT (¬) or Negation

- The NOT connective inverts the truth value of the proposition it precedes.

- **Example**:
  - Let P = "It is raining."
  - "¬P" means "It is NOT raining."

Conjunction Truth Table

P	Q	P ∧ Q
T	T	T
T	F	F
F	T	F
F	F	F

Disjunction Truth Table

P	Q	P ∨ Q
T	T	T
T	F	T
F	T	T
F	F	F

Negation Truth Table

P	~P
T	F
F	T

Implication Truth Table

P	Q	$P \rightarrow Q$
T	T	T
T	F	F
F	T	T
F	F	T

Double Implication Truth Table

P	Q	$P \leftrightarrow Q$
T	T	T
T	F	F
F	T	F
F	F	T

4. IMPLIES (→) or Implication

- The IMPLIES connective is true if either the first proposition is false or the second proposition is true. It is only false if the first proposition is true and the second is false.

- **Example**:
  - Let P = "It is raining."
  - Let Q = "The ground is wet."
  - "P → Q" means "If it is raining, then the ground is wet."

5. BICONDITIONAL (↔) or Double Implication

- **Definition**: The BICONDITIONAL connective is true if both propositions have the same truth value (both true or both false).

- **Example**:
  - Let P = "It is raining."
  - Let Q = "The ground is wet."
  - "P ↔ Q" means "It is raining if and only if the ground is wet."

---

5. Inference in First Order Logic

Inference in First-Order Logic involves deriving new sentences from existing ones. The process uses a set of rules to manipulate the logic formulas to arrive at conclusions. The main inferences used in First Order Logic are explained below:

1. Universal Instantiation (UI)

Universal Instantiation is the process of applying a general statement to a specific case. If something is true for all elements in a domain, it must be true for any specific element in that domain.

Rule:

If $\forall x\ P(x)$ is true (for all x, P(x) is true), then $P(a)$ is true for any specific a.

Example:

Given: $\forall x\ (Human(x) \rightarrow Mortal(x))$

This statement means "for all x, if x is a human, then x is mortal."

Let's instantiate x with "Socrates":

$$Human(Socrates) \rightarrow Mortal(Socrates)$$

This means that since we know that all humans are mortal, we can say that if Socrates is a human, then Socrates is mortal.

2. Existential Instantiation (EI)

Existential Instantiation is about taking a general statement that says something exists and identifying a specific example of that thing.

Rule:

If $\exists x\ P(x)$ is true (there exists some x such that $P(x)$ is true), then $P(c)$ is true for some specific c.

Example:

Given: $\exists x\ (Bird(x)\ \wedge\ CanFly(x))$

This statement means "there exists some x such that x is a bird and x can fly."

Let's instantiate x with "Tweety":

$$Bird(Tweety)\ \wedge\ CanFly(Tweety)$$

This means that there is at least one bird that can fly, and we are naming that bird Tweety. It's like saying, "We know there's a bird that can fly, let's call that bird Tweety."

3. Universal Generalization (UG)

Universal Generalization allows us to take a specific case and generalize it to all cases if the specific case is arbitrary and not unique.

Rule:

If $P(a)$ is true for any arbitrary a, then $\forall x\ P(x)$ is true.

Example:

Given: $Human(Socrates) \rightarrow Mortal(Socrates)$

If we can say this is true for any human, not just Socrates, we generalize to:

$$\forall x\ (Human(x) \rightarrow Mortal(x))$$

This means that if we can prove that any human being is mortal (not just Socrates), then we can say that all humans are mortal. If the specific human (Socrates) is arbitrary, we can generalize the statement.

4. Existential Generalization (EG)

Existential Generalization allows us to take a specific case and conclude that there exists at least one such case.

Rule:

If $P(a)$ is true for some specific a, then $\exists x\ P(x)$ is true.

Example:

Given: $Bird(Tweety) \wedge CanFly(Tweety)$

This specific statement means "Tweety is a bird and Tweety can fly."

We can generalize to:

$$\exists x \, (Bird(x) \, \wedge \, CanFly(x))$$

This means that there exists at least one bird that can fly. We've identified Tweety as a specific example, so we can conclude that there is some bird that can fly.

5. Modus Ponens (MP)

Modus Ponens is a fundamental rule of logic that allows us to conclude the consequent of a conditional statement if the antecedent is true.

Rule:

If $P \, \rightarrow \, Q$ and P are both true, then Q is true.

Example:

Given: $Human(Socrates) \, \rightarrow \, Mortal(Socrates)$

Given: $Human(Socrates)$

Since Socrates is human and if all humans are mortal, we infer:

$$Mortal(Socrates)$$

This means that if Socrates is human and we know that all humans are mortal, we can conclude that Socrates must be mortal.

8. Resolution (RES)

Resolution is a rule used in automated theorem proving to derive a new clause by eliminating a pair of complementary literals. he resolution rule states that if you have two clauses, one containing a literal Q and the other containing its negation $\neg Q$, you can combine these clauses by removing the complementary literals Q and $\neg Q$. The remaining parts of the clauses form a new clause.

Rule:

If $(P \vee Q)$ and $(\neg Q \vee R)$ are true, then $(P \vee R)$ is true.

Example:

Given: $Human(Socrates) \vee Animal(Socrates)$

Given: $\neg Animal(Socrates) \vee Mortal(Socrates)$

We can infer: $Human(Socrates) \vee Mortal(Socrates)$

We eliminate the complementary literals ($Animal(Socrates)$ and $\neg Animal(Socrates)$) to combine the remaining parts of the clauses. This results in the conclusion that either Socrates is human or Socrates is mortal.

6. Forward and Backward Chaining

Forward Chaining

Forward chaining is a method of reasoning in AI where we start with the available facts and apply inference rules to extract more data until we reach the desired goal.

Working:

1. **Start with Facts**: Begin with a set of known facts.

2. **Apply Rules**: Look at the inference rules (if-then statements) and see if any of the current facts trigger these rules.

3. **Generate New Facts**: When a rule is triggered (the "if" part is true), the result (the "then" part) is added to the list of known facts.

4. **Repeat**: Continue applying rules to the newly generated facts.

5. **Stop**: The process stops when the goal is reached or no more rules can be applied.

Example: Imagine a simple medical diagnosis system.

Facts:

1. Symptom: Cough
2. Symptom: Fever

Rules:

1. If a person has a cough and fever, they might have the flu.
2. If a person has the flu, they should rest.

Process:

1. Initial Facts:
    - Cough
    - Fever
2. Apply Rule 1:
    - Since the person has a cough and fever, we conclude they might have the flu.
3. New Fact:
    - Flu
4. Apply Rule 2:
    - Since the person has the flu, we conclude they should rest.
5. Conclusion:
    - The person should rest.

In forward chaining, we start with initial symptoms and apply rules to derive a treatment.

Backward Chaining

Backward chaining is a method of reasoning where we start with the goal and work backward to determine which facts must be true to achieve that goal.

Working:

1. **Start with Goal**: Begin with the goal you want to achieve.

2. **Find Rules**: Look for rules that can lead to the goal.

3. **Break Down**: For each rule, identify the conditions (the "if" part) that need to be true for the rule to apply.

4. **Solve Sub-goals**: Treat each condition as a new goal and repeat the process.

5. **Verify Facts**: Continue until you either verify the starting facts that lead to the goal or find that it's not achievable.

Example: Using the same medical diagnosis system, let's see if a person should rest.

Goal:

1. The person should rest.

Rules:

1. If a person has the flu, they should rest.
2. If a person has a cough and fever, they might have the flu.

Process:

1. Goal:
    - Should the person rest?
2. Find Rule for Goal:
    - Rule 1: If a person has the flu, they should rest.
    - New Sub-Goal: Does the person have the flu?
3. Find Rule for Sub-Goal:
    - Rule 2: If a person has a cough and fever, they might have the flu.
    - New Sub-Goals:
        - Does the person have a cough?
        - Does the person have a fever?
4. Check Facts:
    - Known facts: The person has a cough and a fever.
    - These facts support the sub-goal that the person has the flu.

5. Conclusion:
    - Since the person has a cough and a fever, they might have the flu.
    - Since the person has the flu, they should rest.

In backward chaining, we start with the goal (should rest) and work backwards to check if the facts (cough and fever) support it.

---

7. Probabilistic Reasoning

Probabilistic Reasoning in Artificial Intelligence (AI) refers to the use of probability theory to model and manage uncertainty in decision-making processes. This approach is fundamental in creating intelligent systems that can operate effectively in complex, real-world environments where information is often incomplete or noisy. Unlike deterministic systems, which operate under the assumption of complete and exact information, probabilistic systems acknowledge that the real world is fraught with uncertainties. By employing probabilities, AI systems can make informed decisions even in the face of ambiguity.

1. Bayes' Theorem

One of the foundational tools in probabilistic reasoning is **Bayes' Theorem**, which relates conditional probabilities and helps update our beliefs based on new evidence. It is formulated as:

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

Where:

- $P(A|B)$ is the probability of A given B (Posterior probability)
- $P(B|A)$ is the probability of B given A (Likelihood Probability)
- $P(A)$ is the prior probability of A (Prior Probability)
- $P(B)$ is the prior probability of B (Marginal Probability)

Example: Medical Diagnosis

Let's say a doctor wants to determine the probability that a patient has a disease D given a positive test result T. We have the following information:

- $P(D)$: Prior probability of the disease (e.g., 0.01, or 1%).
- $P(T|D)$: Probability of a positive test given the disease (e.g., 0.95, or 95%).
- $P(T)$: Overall probability of a positive test (e.g., 0.05, or 5%).

Using Bayes' Theorem, we can calculate:

$$P(D|T) = P(T|D) \cdot \frac{P(D)}{P(T)} = 0.95 \times \frac{0.01}{0.05} = 0.19$$

This means that given a positive test result, there is a 19% probability that the patient has the disease.

2. Bayesian Networks

Bayesian Networks are graphical models that represent the probabilistic relationships among a set of variables. These networks simplify complex probabilistic reasoning by breaking down the joint probability distribution into simpler, conditional probabilities.

Example: Weather Prediction

Consider a simple Bayesian Network for weather prediction with three variables:

- **Cloudy** (C)
- **Sprinkler** (S)
- **Rain** (R)

The network structure might look like this:

- $C \rightarrow S$
- $C \rightarrow R$

This means the probability of the sprinkler being on or it raining depends on whether it is cloudy. We use the conditional probabilities $P(S|C)$ and $P(R|C)$ to reason about the likelihood of different weather conditions.

3. Markov Models

Markov Models are mathematical models used to represent systems that transition from one state to another, where the probability of moving to a new state depends only on the current state, not on the sequence of events that preceded it. This characteristic is known as the **Markov Property**.

Example: Weather Prediction

Consider a simple weather system with two states: "Sunny" and "Rainy". The transition probabilities are:

- $P(Sunny \mid Sunny) = 0.8$
- $P(Rainy \mid Sunny) = 0.2$
- $P(Sunny \mid Rainy) = 0.3$
- $P(Rainy \mid Rainy) = 0.7$

This means:

- If it is sunny today, there is an 80% chance it will be sunny tomorrow and a 20% chance it will be rainy.
- If it is rainy today, there is a 30% chance it will be sunny tomorrow and a 70% chance it will be rainy.

4. Hidden Markov Models (HMMs)

Hidden Markov Models are an extension of Markov Models where the system being modeled has hidden states. The states themselves are not directly observable, but each state produces

an observable output (or observation). HMMs are particularly useful for problems where we want to infer the hidden states based on observable data.

Key Concepts

1. **Hidden States**: The actual states of the system, which are not directly observable.

2. **Observations**: The data we can observe, which are generated by the hidden states.

3. **Transition Probabilities**: The probabilities of moving from one hidden state to another.

4. **Emission Probabilities**: The probabilities of observing a particular observation given a specific hidden state.

Example: Weather Prediction with Umbrellas

Imagine a scenario where you want to predict the weather (hidden state) based on whether someone carries an umbrella (observable state).

1. **Hidden States**: "Sunny", "Rainy"
2. **Observable States**: "Umbrella", "No Umbrella"

Transition Probabilities:

- $P(Sunny \mid Sunny) = 0.7$
- $P(Rainy \mid Sunny) = 0.3$
- $P(Sunny \mid Rainy) = 0.4$
- $P(Rainy \mid Rainy) = 0.6$

Emission Probabilities:

- $P(Umbrella \mid Sunny) = 0.1$
- $P(No\ Umbrella \mid Sunny) = 0.9$
- $P(Umbrella \mid Rainy) = 0.8$
- $P(No\ Umbrella \mid Rainy) = 0.2$

If you see someone with an umbrella, you can use the HMM to infer that it is more likely to be rainy. Conversely, if you see no umbrella, it is more likely to be sunny. HMMs help in making such predictions by using the observable data to estimate the hidden states.

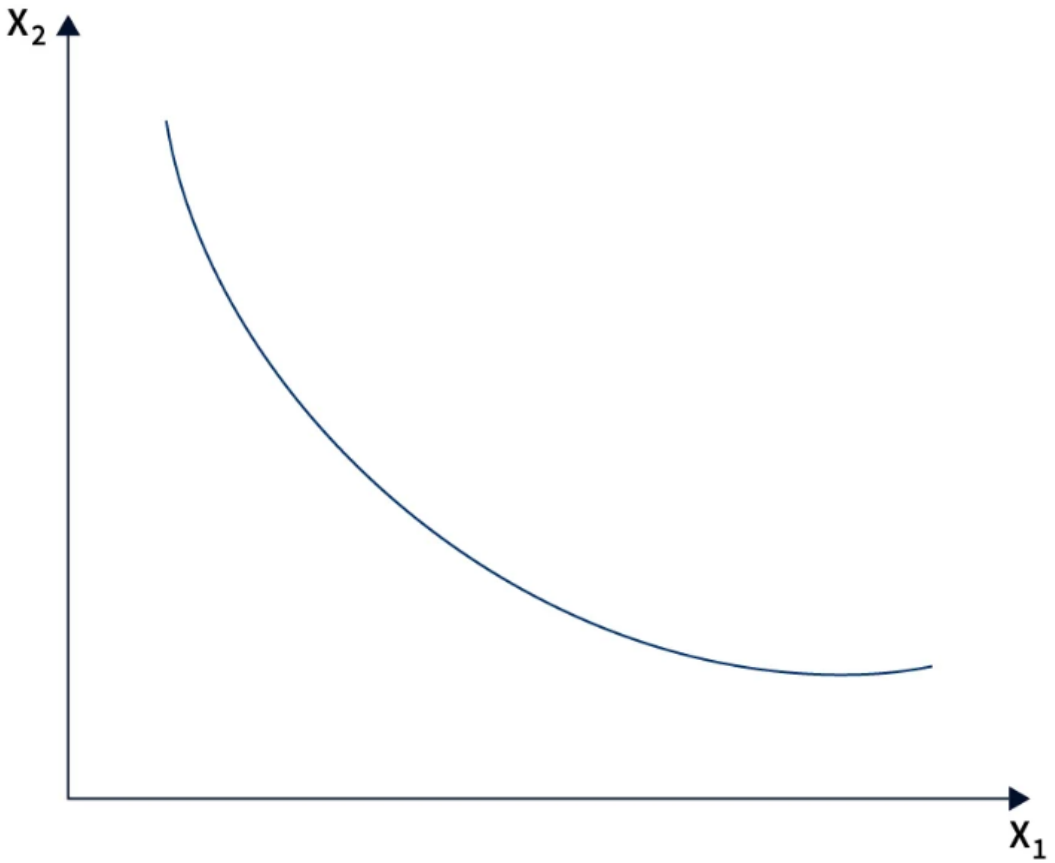
---

8. Utility Theory

Utility theory in artificial intelligence is a mathematical framework used to model decision-making under uncertainty. At its core, utility theory helps AI systems make decisions that maximize a specific goal, referred to as **utility**.

Utility Function

A **utility function** is a mathematical function used in Artificial Intelligence (AI) to represent a system's preferences or objectives. It assigns a numerical value, referred to as **utility**, to different outcomes based on their **satisfaction level**. It maps different outcomes to their corresponding utility values, where **higher utility values represent more desirable outcomes**. The utility function is subjective and can vary from one agent or system to another.



Unit 4

1. Machine Learning and its Approaches

Machine learning is the branch of Artificial Intelligence that focuses on developing models and algorithms that let computers learn from data and improve from previous experience without being explicitly programmed for every task. In simple words, ML teaches the systems to think and understand like humans by learning from the data.

Approaches

Some of the main types of machine learning algorithms are as follows:

1. Supervised Learning
2. Unsupervised Learning
3. Reinforcement Learning

1. Supervised Learning

- **Definition**: Supervised learning involves training a model on a labelled dataset, meaning that each training example is paired with an output label. The model learns to make predictions or decisions based on the input-output pairs.

- **How it works**: The algorithm tries to find patterns in the training data that map the input data (features) to the desired output (labels). It then uses this learned mapping to predict the output for new, unseen data.

- **Example**: Predicting house prices based on features like size, location, and number of bedrooms. If you have a dataset of houses with these features and their corresponding prices, a supervised learning algorithm can learn to predict the price of a new house based on its features.

- **Common algorithms**: Linear regression, logistic regression.

2. Unsupervised Learning

- **Definition**: Unsupervised learning involves training a model on data that does not have labelled responses. The model tries to find hidden patterns or intrinsic structures in the input data.

- **How it works**: The algorithm analyzes the data to identify patterns, group similar data points together, or reduce the data's dimensionality. It does this without any specific guidance on what to look for.

- **Example**: Clustering customers into different segments based on their purchasing behaviour. If you have a dataset of customer transactions, an unsupervised learning algorithm can group customers with similar buying habits together.

- **Common algorithms**: K-means clustering, hierarchical clustering.

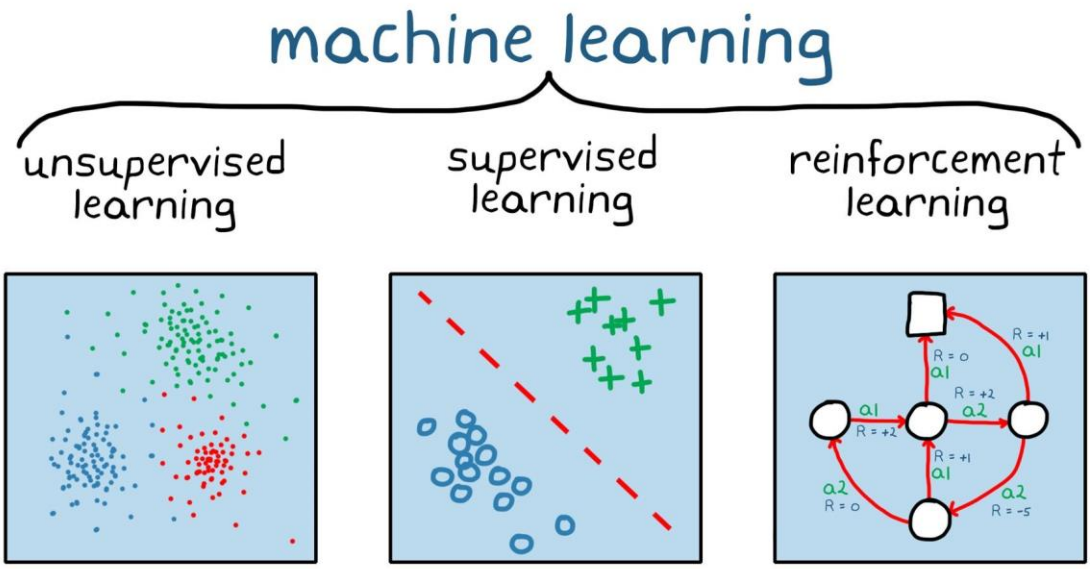
3. Reinforcement Learning

- **Definition**: Reinforcement learning involves training a model to make a sequence of decisions by rewarding desired behaviours and/or punishing undesired ones. The model learns to achieve a goal by interacting with an environment.

- **How it works**: The algorithm receives feedback in the form of rewards or penalties as it navigates through an environment. It aims to maximize the total reward over time by learning the best actions to take in different situations.

- **Example**: Training a robot to navigate through a maze. The robot receives a reward for moving closer to the goal and a penalty for hitting obstacles. Over time, it learns the optimal path to take.

- **Common algorithms**: Q-learning, Deep Q-Networks (DQN)



---

3. Statistical Learning Models

Statistical learning models in machine learning use statistical methods to make predictions or decisions based on data.

These models are fundamental to understanding patterns and relationships within data and form the backbone of many machine learning algorithms.

Supervised learning is typically divided into two main categories:

1. **Regression**: In regression, the algorithm learns to predict a continuous output value, such as predicting house prices based on features like size, bedrooms, and location. Algorithms include Linear Regression, Decision Tree, etc.

2. **Classification**: In classification, the algorithm learns to predict a categorical output variable or class label, such as classifying emails as spam or not. Algorithms include Logistic Regression, K-Nearest Neighbours (KNN), etc.

1. Linear Regression

- Linear regression is a type of supervised machine learning algorithm that computes the linear relationship between the dependent variable and one or more independent features by fitting a linear equation to observed data.

- When there is only one independent feature, it is known as Simple Linear Regression, and when there is more than one feature, it is known as Multiple Linear Regression.

- Similarly, when there is only one dependent variable, it is considered Univariate Linear Regression, while when there are more than one dependent variables, it is known as Multivariate Regression.

Simple Linear Regression

This is the simplest form of linear regression, and it involves only one independent variable and one dependent variable. The equation for simple linear regression is:

$$y = \beta_0 + \beta_1 X + \epsilon$$

where:

- Y is the dependent variable
- X is the independent variable
- β_0 is the intercept
- β_1 is the slope

Multiple Linear Regression

This involves more than one independent variable and one dependent variable. The equation for multiple linear regression is:

$$y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots \beta_n X_n + \epsilon$$

where:

- Y is the dependent variable
- $X_1, X_2, \ldots, X_n$ are the independent variables
- β_0 is the intercept
- $\beta_1, \beta_2, \ldots, \beta_n$ are the slopes

2. Logistic Regression

- Logistic regression is used for binary classification where we use sigmoid function, that takes input as independent variables and produces a probability value between 0 and 1.

- For example, we have two classes Class 0 and Class 1 if the value of the logistic function for an input is greater than 0.5 (threshold value) then it belongs to Class 1 otherwise it belongs to Class 0.

- It is referred to as regression because it is the extension of linear regression but is mainly used for classification problems.

- The sigmoid function is a mathematical function used to map the predicted values to probabilities.

- It maps any real value into another value within a range of 0 and 1. The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the “S” form. The S-form curve is called the Sigmoid function or the logistic function.

- In logistic regression, we use the concept of the threshold value, which defines the probability of either 0 or 1. Such as values above the threshold value tends to 1, and a value below the threshold values tends to 0.

- On the basis of the categories, Logistic Regression can be classified into three types:

    - **Binomial:** In binomial Logistic regression, there can be only two possible types of the dependent variables, such as 0 or 1, Pass or Fail, etc.

    - **Multinomial:** In multinomial Logistic regression, there can be 3 or more possible unordered types of the dependent variable, such as “Cat”, “Dogs”, or “Sheep”

    - **Ordinal:** In ordinal Logistic regression, there can be 3 or more possible ordered types of dependent variables, such as “Low”, “Medium”, or “High”.

3. Decision Trees

- Decision tree is a tree-like structure that is used to model decisions and their possible consequences.

- Each internal node in the tree represents a decision, while each leaf node represents a possible outcome.

- Decision trees can be used to model complex relationships between input features and output variables.

Structure of a Decision Tree:

- **Root Node**: The topmost node in a decision tree, representing the entire dataset.
- **Internal Nodes**: Nodes that represent decisions based on feature values.
- **Leaf Nodes**: Terminal nodes that represent the final classification or decision.

Algorithm:

1. **Select the Best Feature**: Choose the feature that best separates the data based on a criterion like Gini impurity or information gain.
2. **Split the Data**: Partition the dataset into subsets based on the selected feature.
3. **Repeat**: Recursively apply steps 1 and 2 to each subset until a stopping criterion is met (e.g., maximum depth or minimum samples per node).
4. **Assign Labels**: Assign a class label to each leaf node based on the majority class of the data points in that node.

Example:

Consider a small dataset to predict whether a student will pass or fail based on their study hours and attendance.

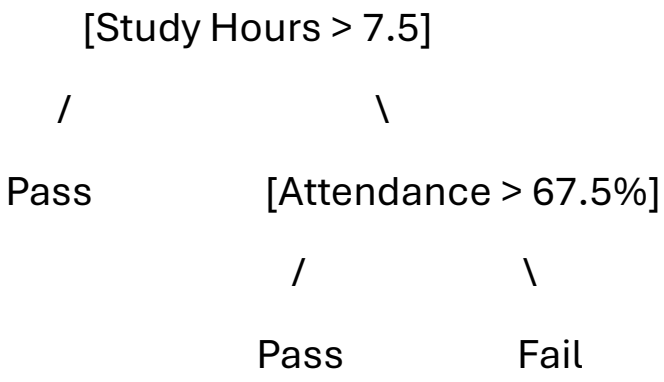
Study Hours	Attendance	Pass/Fail
10	90%	Pass
9	80%	Pass
8	85%	Pass
7	70%	Fail
6	60%	Fail
5	65%	Fail

Steps:

1. **Select the Best Feature**:
    - Compute information gain for "Study Hours" and "Attendance".
    - Assume "Study Hours" provides higher information gain.
2. **Split the Data**:
    - Split based on "Study Hours" threshold (e.g., 7.5).
    - Nodes:
        - Study Hours > 7.5: {10, 9, 8}
        - Study Hours ≤ 7.5: {7, 6, 5}

3. **Further Splits**:
    - For Study Hours > 7.5, all are "Pass".
    - For Study Hours ≤ 7.5, split further based on "Attendance".
        - Attendance > 67.5%: "Pass"
        - Attendance ≤ 67.5%: "Fail"

Tree Structure:



Solution:

- For a student with 9 study hours and 85% attendance:
    - Study Hours > 7.5 → Pass.
- For a student with 6 study hours and 70% attendance:
    - Study Hours ≤ 7.5 → Check Attendance → Attendance > 67.5% → Pass.

Unit 5

1. Pattern Recognition

Pattern Recognition is the process of identifying patterns and regularities in data. It involves the classification or categorization of data based on the input features. Pattern recognition systems are designed to recognize complex patterns in data and make decisions based on these patterns.

Key Concepts:

1. **Patterns**: Recognizable regularities or structures in data.

2. **Features**: Attributes or characteristics used to represent the data.

3. **Classification**: Assigning data to one of the predefined categories.

4. **Clustering**: Grouping data into clusters where members of a cluster are more similar to each other than to those in other clusters.

5. **Regression**: Predicting a continuous output based on input features.

2. Design Principles of Pattern Recognition System

1. Data Collection and Preprocessing:

- **Data Collection**: Gather raw data from various sources.
- **Preprocessing**: Clean the data, handle missing values, normalize or standardize features, and extract relevant features.

2. Feature Extraction and Selection:

- **Feature Extraction**: Transform raw data into a set of measurable features that can be used for classification or clustering.
- **Feature Selection**: Choose the most relevant features that contribute to the performance of the pattern recognition model, reducing dimensionality and improving efficiency.

3. Model Selection:

- **Choice of Algorithm**: Select an appropriate algorithm based on the nature of the problem and the type of data (e.g., decision trees, neural networks, support vector machines).
- **Model Training**: Train the model using a training dataset, where the model learns to recognize patterns and make predictions.

4. Evaluation and Validation:

- **Performance Metrics**: Use metrics such as accuracy, precision, recall, F1 score, and confusion matrix to evaluate the performance of the model.
- **Cross-validation**: Implement techniques like k-fold cross-validation to assess the model's performance on different subsets of data, ensuring generalizability.

5. Deployment and Maintenance:

- **Implementation**: Deploy the model into a real-world application where it can process new data and make predictions.
- **Monitoring**: Continuously monitor the model's performance and update it as needed to handle new patterns or changes in data distribution.

---

3. Statistical Pattern Recognition

Statistical pattern recognition is a subfield of pattern recognition that focuses on using statistical techniques to recognize patterns and make decisions based on data. This approach relies on probability theory and statistical methods to model and analyze data, often involving assumptions about the underlying distribution of the data.

Common Techniques in Statistical Pattern Recognition

1. Linear Discriminant Analysis (LDA)
2. Quadratic Discriminant Analysis (QDA)
3. Gaussian Mixture Models (GMM)
4. Hidden Markov Models (HMM)
5. Support Vector Machines (SVM)

Applications of Statistical Pattern Recognition

- **Speech Recognition**: Identifying spoken words based on audio features.

- **Image Recognition**: Classifying objects or patterns in images.

- **Biometrics**: Recognizing individuals based on physiological or behavioral characteristics (e.g., fingerprint, facial recognition).

- **Medical Diagnosis**: Classifying medical conditions based on patient data.

---

4. Parameter Estimation Techniques

- Parameter estimation involves using sample data to estimate the parameters of a distribution or model.

- Parameters are descriptive measures of an entire population, but their exact values are often unknown because measuring the entire population is impractical.

- Instead, we take a random sample from the population and compute parameter estimates based on that sample.

- Two widely-used methods for parameter estimation are: **Principal Component Analysis (PCA)** and **Linear Discriminant Analysis (LDA)**.

1. Principal Component Analysis (PCA)

- Principal Component Analysis (PCA) is a popular technique for dimensionality reduction that transforms the original features into a new set of uncorrelated features called principal components.

- These components are ordered by the amount of variance they explain in the data.

- The first principal component captures the most variance, the second captures the second most, and so on.

- By selecting the top few principal components that capture the majority of the variance, we can reduce the dimensionality of the dataset while retaining most of the important information.

- PCA is particularly useful for visualizing high-dimensional data and for noise reduction.

- PCA is widely used in fields such as image compression, noise reduction, and data visualization, as it simplifies complex datasets while preserving their essential structure.

2. Linear Discriminant Analysis (LDA)

- Linear Discriminant Analysis (LDA) is a technique used to improve the separation between different groups or classes in data.

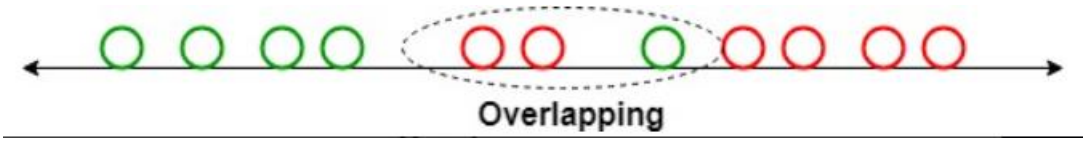
- Imagine you have a set of data points belonging to different classes (e.g., different species of flowers). LDA helps to find the best line or plane that separates these classes as clearly as possible.

- It does this by looking at how the data points are spread out within each class and how the means of different classes are spread out from each other.

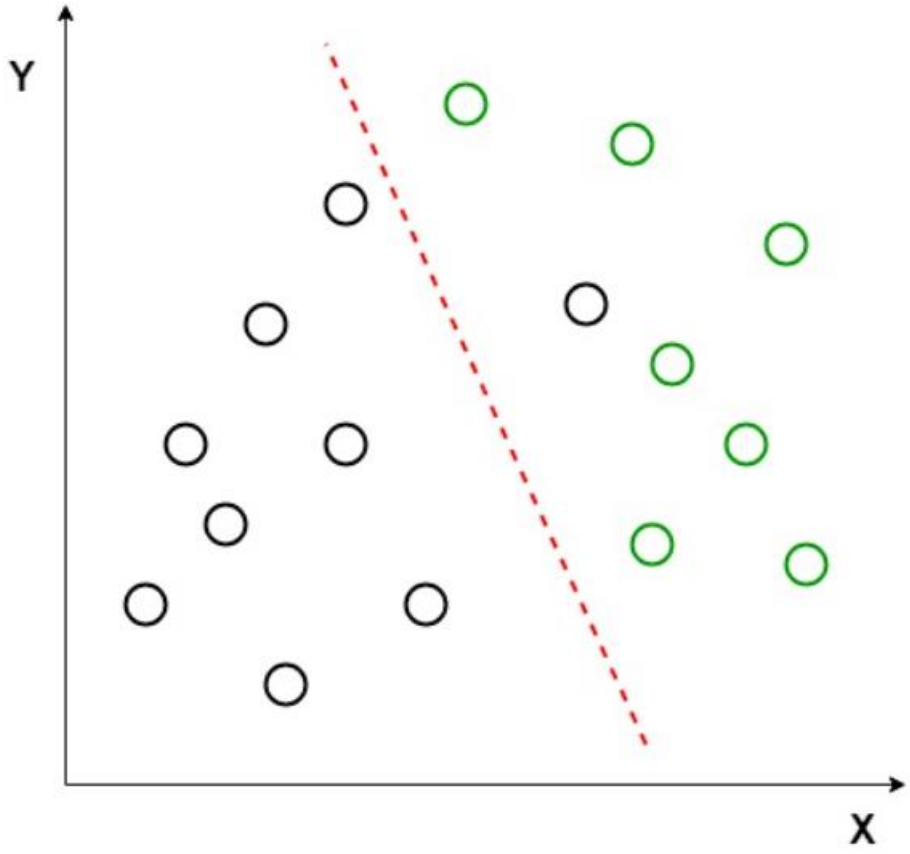
- The goal is to maximize the distance between the means of the classes while minimizing the spread within each class.

- This way, when the data is projected onto this new line or plane, the classes are more distinct and easier to classify.

Explaination:



Suppose we have two sets of data points belonging to two different classes that we want to classify.

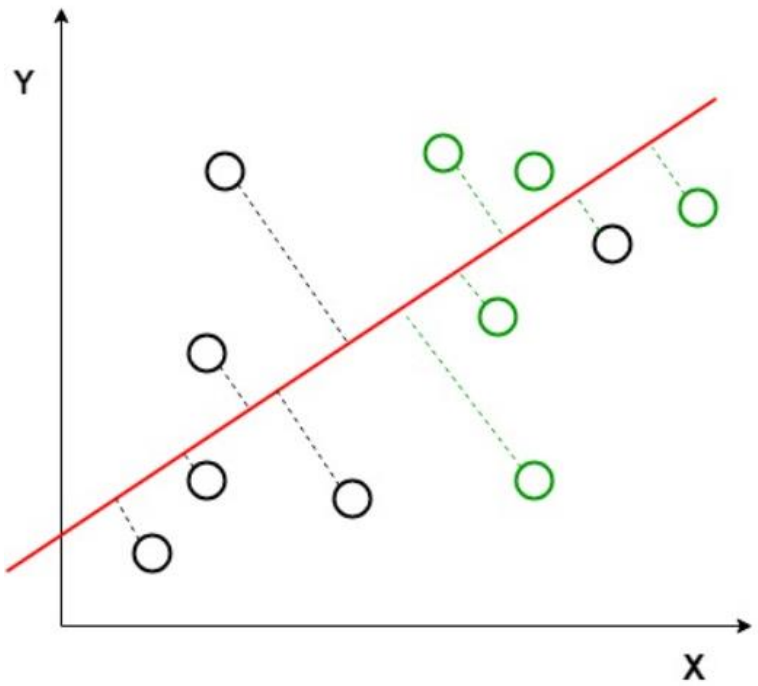


As shown in the given 2D graph, when the data points are plotted on the 2D plane, there's no straight line that can separate the two classes of data points completely. Hence, in this case, LDA (Linear Discriminant Analysis) is used which reduces the 2D graph into a 1D graph in order to maximize the separability between the two classes.

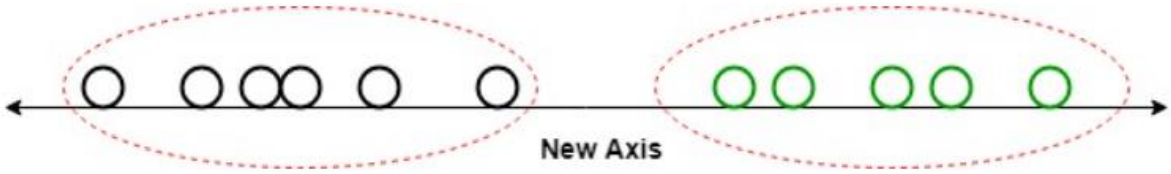
Here, Linear Discriminant Analysis uses both axes (X and Y) to create a new axis and projects data onto a new axis in a way to maximize the separation of the two categories and hence, reduces the 2D graph into a 1D graph.

Two criteria are used by LDA to create a new axis:

1. Maximize the distance between the means of the two classes.
2. Minimize the variation within each class.



In the above graph, it can be seen that a new axis (in red) is generated and plotted in the 2D graph such that it maximizes the distance between the means of the two classes and minimizes the variation within each class. In simple terms, this newly generated axis increases the separation between the data points of the two classes. After generating this new axis using the above-mentioned criteria, all the data points of the classes are plotted on this new axis and are shown in the figure given below.



---

5. Classification Techniques

As the name suggests, Classification is the task of “classifying things” into sub-categories. Classification is part of supervised machine learning in which we put labelled data for training. Classification is a process of categorizing data or objects into predefined classes or categories based on their features or attributes.

1. Support Vector Machine (SVM)

- Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems.

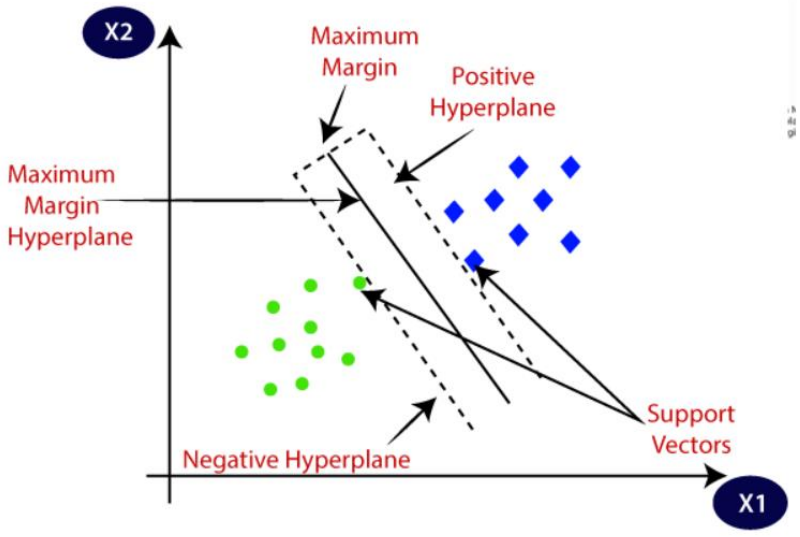
- Primarily, it is used for Classification problems in Machine Learning.

- The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

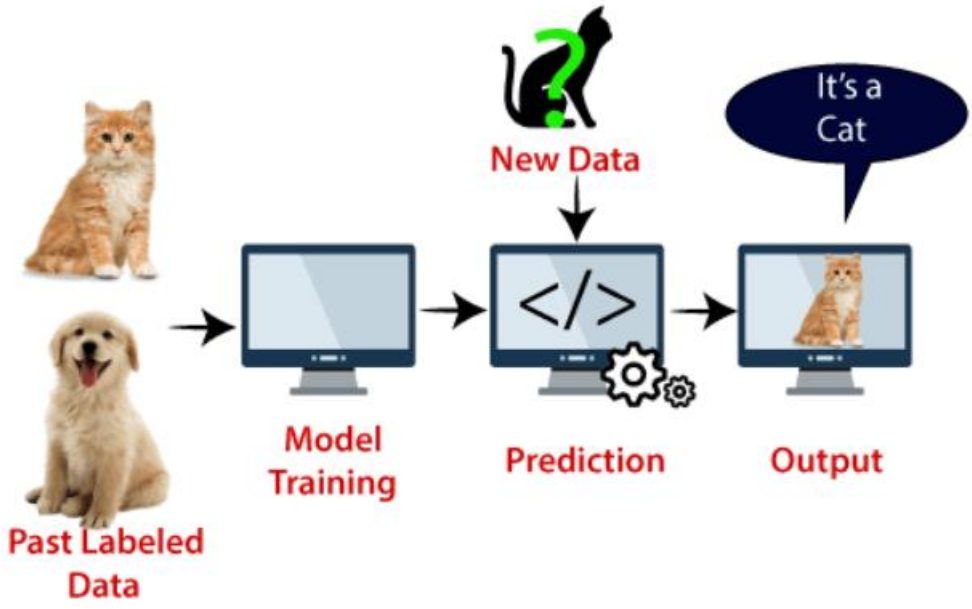
- SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine.

- SVM algorithm can be used for **Face detection, image classification, text categorization,** etc.

- Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:



Example: SVM can be understood with the example that we have used in the KNN classifier. Suppose we see a strange cat that also has some features of dogs, so if we want a model that can accurately identify whether it is a cat or dog, so such a model can be created by using the SVM algorithm. We will first train our model with lots of images of cats and dogs so that it can learn about different features of cats and dogs, and then we test it with this strange creature. So as support vector creates a decision boundary between these two data (cat and dog) and choose extreme cases (support vectors), it will see the extreme case of cat and dog. On the basis of the support vectors, it will classify it as a cat. Consider the below diagram:



2. k-Nearest Neighbours (KNN) Algorithm

- k-Nearest Neighbors (k-NN) is a simple and intuitive machine learning algorithm used for classification and regression tasks.

- The basic idea of k-NN is to classify a data point based on the majority class of its k closest neighbors in the feature space. To make a prediction for a new data point, k-NN calculates the distance between this point and all other points in the training dataset using a distance metric, typically Euclidean distance.

- It then identifies the k closest data points (neighbors) and assigns the class that is most common among these neighbors.

- The value of k is a critical parameter that determines the algorithm's performance: a small k can lead to a noisy decision boundary, while a large k might smooth out the boundaries too much, potentially leading to underfitting.

Steps:

1. **Choose the number of neighbors k**: Decide how many neighbors to consider when making the prediction.

2. **Calculate the distance**: Measure the distance between the new data point and all the points in the training dataset. Common distance metrics include Euclidean distance, Manhattan distance, and Minkowski distance.

3. **Find the nearest neighbors**: Identify the k points in the training dataset that are closest to the new data point.

4. **Vote for the classes** (classification) or **average the values** (regression): For classification, assign the class that is most frequent among the k nearest neighbors. For regression, compute the average of the values of the kkk nearest neighbors.

Example:

We have a small dataset of students' study hours and their pass/fail results in an exam. We want to predict whether a new student who studied for 3.5 hours will pass or fail.

Training Data:

Study Hours	Pass/Fail
1.0	Fail
2.0	Fail
3.0	Fail
4.0	Pass
5.0	Pass
6.0	Pass

Let's use $k = 3$ to classify the new student.

Step-by-Step Process:

1. **Calculate the Distance**: We will use the Euclidean distance to calculate the distance between the new data point (3.5 hours) and all the points in the training dataset.

$$Distance = \sqrt{(3.5 - Study\ Hours)^2}$$

- Distance to 1.0 hours: $\sqrt{(3.5 - 1.0)^2} = \sqrt{(2.5)^2} = 2.5$
- Distance to 2.0 hours: $\sqrt{(3.5 - 2.0)^2} = \sqrt{(1.5)^2} = 1.5$
- Distance to 3.0 hours: $\sqrt{(3.5 - 3.0)^2} = \sqrt{(0.5)^2} = 0.5$
- Distance to 4.0 hours: $\sqrt{(3.5 - 4.0)^2} = \sqrt{(0.5)^2} = 0.5$
- Distance to 5.0 hours: $\sqrt{(3.5 - 5.0)^2} = \sqrt{(1.5)^2} = 1.5$
- Distance to 6.0 hours: $\sqrt{(3.5 - 6.0)^2} = \sqrt{(2.5)^2} = 2.5$

2. **Find the Nearest Neighbors**: The three nearest neighbors are the ones with the smallest distances:
    - 3.0 hours (0.5 distance, Fail)
    - 4.0 hours (0.5 distance, Pass)
    - 2.0 hours (1.5 distance, Fail)

3. **Vote for the Classes**: Among the three nearest neighbors:
    - Fail: 2 votes (3.0 hours and 2.0 hours)
    - Pass: 1 vote (4.0 hours)

The majority class among the neighbors is "Fail".

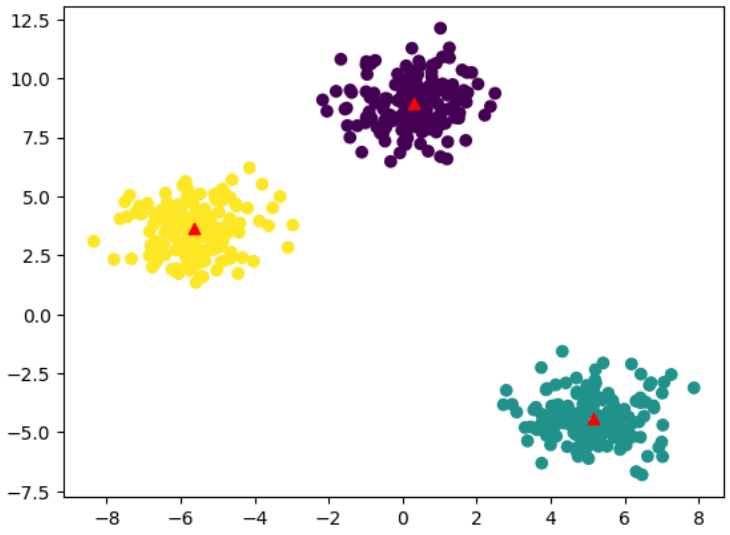
Prediction:

Based on the k-NN algorithm with $k = 3$, we predict that the student who studied for 3.5 hours will "Fail" the exam.

---

6. K-means Clustering

- K-Means Clustering is an Unsupervised Machine Learning algorithm, which groups the unlabelled dataset into different clusters.
- It starts by randomly assigning the clusters centroid in the space.
- Then each data point is assigned to one of the clusters based on its distance from centroid of the cluster.
- After assigning each point to one of the cluster, new cluster centroids are assigned which are the averages of the items categorized in that cluster so far.
- This process runs iteratively until it finds good cluster.
- **Demerit**: When data points overlapped this clustering is not suitable.
- **Merit**: K Means is faster as compared to other clustering techniques.



Example:

Consider a simple dataset of points in a 2D space: (2,3), (3,3), (6,6), (8,8).

Step-by-Step Process

1. **Initialization**:
    - Choose $K = 2$.
    - Randomly select initial centroids, say $(2, 3)$ and $(6, 6)$.

2. **Assignment**:
    - Calculate the Euclidean distance from each point to the centroids.
    - Assign each point to the nearest centroid.

Point	Distance to $(2,3)$	Distance to $(6,6)$	Assigned Cluster
$(2,3)$	0	5	1
$(3,3)$	1	4.24	1
$(6,6)$	5	0	2
$(8,8)$	7.81	2.83	2

3. **Update Centroids**:

Calculate the new centroids by averaging the points in each cluster.

- **Cluster 1**: Points $(2,3)$ and $(3,3)$.

$$\text{New Centroid} = \left(\frac{2+3}{2}, \frac{3+3}{2}\right) = (2.5, 3)$$

- **Cluster 2**: Points $(6,6)$ and $(8,8)$

$$\text{New Centroid} = \left(\frac{6+8}{2}, \frac{6+8}{2}\right) = (7, 7)$$

4. **Repeat**:

- Reassign points based on the new centroids.
- Update centroids again if necessary and repeat until centroids no longer change significantly.

Confused? Let’s have a short map of what you have studied so far:

Supervised Learning:

- **Regression:**
  - Linear Regression
- **Classification:**
  - Logistic Regression
  - Support Vector Machine (SVM)
  - Decision Trees
  - Random Forest
  - k-Nearest Neighbors (k-NN)
  - Neural Networks
  - Naive Bayes

Unsupervised Learning:

- **Clustering:**
  - K-Means
  - Hierarchical Clustering
- **Dimensionality Reduction:**
  - Principal Component Analysis (PCA)
  - Linear Discriminant Analysis (LDA)

Always chant

Hare Krishna Hare Krishna
Krishna Krishna Hare Hare
Hare Ram Hare Ram
Ram Ram Hare Hare

And be happy

By Sarthak Sobti