History of Artificial Intelligence (AI)

Early Concepts & Origins

- **1950**: Alan Turing asked the question — *“Can machines think?”* and published *"Computing Machinery and Intelligence"*, proposing the **Turing Test** to evaluate a machine's ability to exhibit human-like intelligence.
- **1956**: The term **"Artificial Intelligence"** was coined by **John McCarthy** during the Dartmouth Conference—considered the official birth of AI.

📘 Symbolic AI Era (1950s–1960s)

- Focus on symbolic reasoning and logic-based programs.
- AI systems attempted to replicate human knowledge using **rules and symbols.**
- **Symbols** are facts like father(John, Mary) which means "John is Mary's father".
- **Rules** are logic like: If A is B's father and B is C's mother, then A is C's grandfather.
- **Problem**: Computers were very slow and had little memory, so progress was limited.

⚙️ Expert Systems & AI Winter (1970s–1980s)

- Development of **expert systems**—rule-based systems used in medicine, finance, etc.
- Rule-based AI uses simple "if-then" rules to make decisions.
- **Limitations**: Could not handle uncertainty or adapt to new data.
- **AI Winter**: Funding and interest dropped due to high expectations and slow progress.

📊 Machine Learning Emerges (1990s)

- Instead of just programming rules, scientists taught computers to **learn from data**.
- Algorithms like **decision trees**, **support vector machines**, and **neural networks** gained popularity.
- **Data availability** and **processing power** started improving.

🤖 Deep Learning Revolution (2000s–2010s)

- Rise of **deep neural networks** inspired by the human brain.
- Major breakthroughs in **image recognition**, **speech recognition**, and **natural language processing (NLP)**.
- Tech giants like Google, Facebook, and OpenAI invested heavily in AI research.

📜 Transformer Models & GPT Era (2018–Present)

- Introduction of **Generative Pre-trained Transformers (GPT)** by OpenAI.
- **GPT-3** and similar models demonstrated powerful language understanding and generation capabilities.
- Applications: chatbots, writing assistants, translation tools, creative content generation.

🌍 AI's Impact Today

- Used across industries: healthcare, finance, transportation, and retail.
- Powers **autonomous vehicles**, **virtual assistants**, **predictive analytics**, and more.

⚖️ Challenges & Ethics

- Concerns: **job loss**, **bias in algorithms**, **privacy**, **autonomous weapons**.
- Emphasis on **ethical AI** and responsible development.

What is Artificial Intelligence (AI)

Artificial Intelligence (AI) is the ability of machines or computers to perform tasks that normally require human intelligence, such as learning, reasoning, problem-solving, and understanding language.

- AI helps machines **learn from data** and improve over time.
- It can **understand and interpret** human language (like Siri or Alexa).
- AI can **recognize images** and **patterns** (used in facial recognition).
- It makes **decisions or predictions** based on data.
- AI is used in many fields like healthcare, finance, gaming, and self-driving cars.

How AI Works (Basic Working)

1. **Data Collection**: AI systems collect large amounts of data (images, text, numbers).
2. **Data Processing**: The data is cleaned and prepared for analysis.
3. **Learning**: AI models learn patterns from data using algorithms (e.g., machine learning).
4. **Decision Making**: The AI uses what it learned to make predictions or decisions.
5. **Improvement**: AI systems improve by learning from new data continuously.

Types of AI

Aspect	Capability	Functionality
Meaning	Refers to **how intelligent** the AI is (its power).	Refers to **how the AI behaves or operates.**
Focus	Focuses on the **level of intelligence** AI can achieve.	Focuses on **tasks it can perform** and **how it reacts.**
Types	Narrow AI, General AI, Super AI	Reactive Machines, Limited Memory, Theory of Mind, Self-aware
Example	Siri (Narrow AI), Theoretical AI doctor (General AI)	Deep Blue (Reactive), Self-driving car (Limited Memory)
Usage	Tells **what the AI is capable of doing** overall	Tells **how the AI makes decisions or learns**

1. Types of AI Based on Capability

1.1 Narrow AI (Weak AI)

- **Definition:** AI designed to perform a specific task or a narrow range of tasks. It cannot do anything outside its programmed function.
- **Characteristics:** Task-specific, no generalization, mimics human performance, lacks consciousness.
- **Example:** Voice assistants like Siri, Alexa, and recommendation systems on Netflix.
- **Usage:** Used in virtual assistants, spam filters, image recognition.

1.2 General AI (Strong AI)

- **Definition:** AI with human-like intelligence that can understand, learn, and apply knowledge across a wide variety of tasks, similar to a human being.
- **Characteristics:** Broad intelligence, self-learning, human-like reasoning, adaptable across domains.
- **Example:** Still theoretical; no real-world examples yet.
- **Usage:** AI systems that could autonomously diagnose and solve complex medical issues.

1.3 Super AI (Artificial Superintelligence)

- **Definition:** AI that surpasses human intelligence and can perform any intellectual task better than humans.
- **Characteristics:** Beyond human intelligence, fully autonomous, independent decision-making, ethical concerns.
- **Example:** This is futuristic and does not exist yet.
- **Usage:** Potential to revolutionize every industry by making breakthroughs in science, medicine, and technology far beyond human capability.

Aspect	Narrow AI (Weak AI)	General AI (Strong AI)
Definition	AI designed for a **specific task** only.	AI that can **learn and perform any intellectual task** like a human.
Scope	**Limited** to one area.	**Broad** and flexible across multiple areas.
Learning Ability	Learns **only within its task boundaries.**	Can **self-learn and generalize** knowledge.
Examples	Siri, Alexa, Google Translate, spam filters	Still **theoretical**, no real-world examples yet.
Human-like Thinking	No – mimics human behavior in a **narrow task only**.	Yes – aims for **full human-level intelligence**.
Usage	Common in **everyday applications.**	Expected in **future smart systems** like AI doctors or AI teachers.

2. Types of AI Based on Functionality

2.1 Reactive Machines

- **Definition:** AI systems that can only react to current inputs and do not store past experiences.
- **Characteristics:** No memory, simple decision-making, lacks contextual understanding.
- **Example:** IBM's Deep Blue chess-playing computer.
- **Usage:** Used in static environments like basic game-playing and pattern recognition.

2.2 Limited Memory

- **Definition:** AI that can use past experiences or data to make better decisions but has a limited memory.
- **Characteristics:** Short-term memory, needs training data, improves adaptability over time.
- **Example:** Self-driving cars that observe recent surroundings to navigate safely.
- **Usage:** Used in autonomous vehicles, recommendation engines, fraud detection systems.

2.3 Theory of Mind

- **Definition:** AI that can understand emotions, beliefs, and intentions of humans and other entities.
- **Characteristics:** Social awareness, emotional intelligence, complex decision-making.
- **Example:** Sophia the Robot
- **Usage:** Expected in future human-robot interaction, social robots, and empathetic AI assistants.

2.4 Self-aware AI

- **Definition:** AI with self-consciousness and awareness of its own existence.
- Characteristics: Self-consciousness, autonomous decision making, raises ethical and moral concerns.
- **Example:** Purely theoretical and does not exist yet.
- **Usage:** Could lead to fully autonomous machines with self-understanding, but raises complex ethical issues.

AI Agent

An **AI agent** is a software that interacts with the environment, collects information, and acts to achieve goals automatically.

How AI Agents Work

1. **Perceive Environment**: Gather data using sensors, user input, or databases.
2. **Process & Decide**: Analyze data using rules or machine learning.
3. **Act**: Perform tasks like answering questions or controlling devices.
4. **Learn & Improve**: Use past experiences to perform better (reinforcement learning).

Architecture of AI Agents

1. **Profiling Module** – Understands role and collects data.
2. **Memory Module** – Stores and recalls past experiences.
3. **Planning Module** – Makes decisions and plans actions.
4. **Action Module** – Executes the planned actions.

Structure of an AI Agent

- **Agent =** Architecture + Agent Program
    - **Architecture**: Hardware (e.g., sensors, actuators).
    - **Agent Program**: Software logic that maps inputs to actions.

Types of AI Agents

1. Simple Reflex Agents: Acts only based on the current input.

- Uses if-condition-then-action rules; no memory of past actions.

2. Model-Based Reflex Agents: Maintains an internal state (memory).

- Handles partially observable environments using past data.

3. Goal-Based Agents: Chooses actions to achieve a specific goal.

- Uses search and planning to decide the next step.

4. Utility-Based Agents: Picks actions that maximize utility (happiness).

- Considers quality of outcomes, not just goal achievement.

5. Learning Agents: Can learn from experience and improve performance.

- Has four parts: Learning element, Critic, Performance element, Problem generator.

6. Multi-Agent Systems (MAS): Involves multiple agents working together or in competition.

- Types: Cooperative / Competitive, Homogeneous / Heterogeneous.

7. Hierarchical Agents: Structured in levels with different responsibilities.

- High-level agents set goals; low-level agents execute tasks.

Agent Classifications

- **Reactive**: Respond immediately to current sensory input without thinking ahead.
- **Proactive**: Take initiative and plan actions to achieve long-term goals for future.
- **Fixed Environment**: Stable and predictable Environment, not change unexpectedly.
- **Dynamic Environment**: Continuously changes over time, often in unpredictable ways.
- **Single-Agent**: Involves only one agent making decisions and acting in environment.
- **Multi-Agent**: Multiple agents operate in the same environment, either cooperating or competing.

Advantages of AI Agents

- **Autonomous:** Can operate without human intervention, saving time and effort.
- **Efficient:** Perform repetitive or complex tasks quickly and accurately.
- **Adaptive:** Can learn and improve their performance over time.
- **Scalable:** Can handle multiple tasks or work continuously without fatigue.

Disadvantages of AI Agents

- **Limited Understanding:** May fail in unexpected or complex situations without proper programming.
- **Dependence on Data:** Require large amounts of data and quality input to perform well.
- **Lack of Creativity:** Cannot think creatively or intuitively like humans.
- **Security Risks:** Vulnerable to hacking or malicious manipulation in some applications.

Algorithms in AI

Algorithms in AI are step-by-step instructions or rules that an AI system follows to solve problems, make decisions, or learn from data. They define how the AI processes information to achieve specific tasks or goals.

- They help machines **learn patterns** from data (like in machine learning).
- They guide decision-making based on input data or environment.
- Different algorithms are used for different tasks: classification, prediction, optimization, etc.
- Examples include **search algorithms**, **learning algorithms**, **optimization algorithms**, and **neural networks**.

Types of AI Algorithms

1. **Search Algorithms:** Explore possible solutions to find the best or goal state.
    - **Examples**: Breadth-First Search (BFS), Depth-First Search (DFS), A* Algorithm.
2. **Optimization Algorithms:** Optimize a function by finding maximum/minimum values.
    - **Examples**: Genetic Algorithm, Gradient Descent.
3. **Machine Learning Algorithms:** Learn patterns from data to make predictions
    - **Supervised**: Linear Regression, Decision Trees.
    - **Unsupervised**: K-Means, PCA.
    - **Reinforcement**: Q-Learning, Deep Q-Network (DQN).
4. **Neural Networks / Deep Learning**: Mimic brain neurons for complex tasks.
    - **Examples**: CNN (image), RNN (sequential data), LSTM (memory handling).
5. **Rule-Based Algorithms**: Use IF-THEN rules for decision-making.
    - **Examples**: Expert Systems, Decision Tables.
6. **Evolutionary Algorithms:** Inspired by natural selection, they evolve solutions over generations.
    - **Examples**: Genetic Algorithm, Particle Swarm Optimization.
7. **Probabilistic Algorithms:** Work with uncertainty and make predictions using probabilities.
    - **Examples**: Bayesian Networks, Hidden Markov Models.

Search Algorithm in AI

A search algorithm is a method used by AI agents to find a path from a start state to a goal state within a state space by exploring possible states and actions.

Basic Terms:

1. **State Space** – Set of all possible states the agent can be in.
2. **Search Space** – Subset of the state space that is actually explored during the search process.
3. **Start State** – The initial point of the search.
4. **Goal State** – The destination or condition the agent is trying to reach.
5. **Solution** – A sequence of actions (a plan) that leads from the start to the goal.
6. **Fringe / Frontier**– A data structure (stack, queue, or priority queue) used to store unexplored states.
7. **Path Cost (g(x))** – The total cost of a path from the start state to a given state x
8. **Heuristic Function (h(x))** – Provides an estimated cost from state x to the goal.
9. **Evaluation Function (f(x))** – In informed searches, a function like f(x) = g(x) + h(x) is used to prioritize states.
10. **Completeness** – Whether the algorithm always finds a solution (if one exists).
11. **Optimality** – Whether the algorithm always finds the least-cost path.
12. **Time Complexity** – Efficiency of the algorithm in terms of computation time.
13. **Space Complexity** – Efficiency of the algorithm in terms of memory usage.
14. **Irrevocability** – A condition where once an action is taken, it cannot be undone. (Not using Backtracking)
15. **Dead End** – A state from which no goal state can be reached.

Generate and Test (G&T)

Generate and Test is a simple problem-solving method in AI where possible solutions are generated and tested to check if they meet the goal condition.

Steps Involved:

1. **Generate** – Create a possible solution from the search space.
2. **Test** – Check if it satisfies the goal or constraints.
3. **Repeat** – Continue until a valid solution is found or all options are exhausted.

Properties

- **Uninformed Search**: No heuristic is used; blindly generates candidates.
- **Brute-force Approach**: Tries all possible solutions until one is valid.
- **Test Function Required**: A function is needed to check if a solution meets the goal.
- **Non-redundant**: It should avoid generating the same solution multiple times.
- **Completeness**: Will find a solution if one exists and search space is finite.
- **Correctness**: The test must accurately verify valid solutions.
- **Termination**: Stops on finding a solution or after exhausting all options.

Types of Search Algorithms

A. Based on Direction of Reasoning

1. **Data-Driven Search (Forward Search)**
    - Starts from the **initial state** and applies actions to reach the **goal**.
    - Example: Problem-solving when input is known but output (goal) is unknown.
    - Common in **pathfinding** problems.
2. **Goal-Driven Search (Backward Search)**
    - Starts from the **goal state** and works backwards to find the **start state**.
    - Example: Proof systems or theorem proving.
    - Useful when the goal is well-defined or has limited possibilities.

B. Based on Use of Heuristics

1. Uninformed (Blind) Search Algorithms

These algorithms do not use any domain-specific knowledge or heuristics.

- **Depth-First Search (DFS)**: Uses stack; explores deep branches first.
- **Depth-Limited Search (DLS)**: DFS with a fixed depth limit to prevent going too deep or into infinite paths.
- **Breadth-First Search (BFS)**: Uses queue; Explores all nodes at the current depth before moving to the next level
- **Iterative Deepening Depth-First Search (IDDFS)**: Repeatedly applies DLS with increasing depth limits. Combines advantages of DFS and BFS.

- **Uniform Cost Search (UCS)**: Expands the node with the **lowest path cost (g(x))**. Guarantees optimal solution if cost is positive.

2. Informed (Heuristic) Search Algorithms

These use a heuristic to estimate the cost from the current state to the goal.

- **Greedy Best-First Search** – Selects node closest to goal based on h(x).
- **A* Search Algorithm** – Uses **Evaluation Function:** f(x) = g(x) + h(x) for Tree and Graph
    - g(x) = cost so far (from start to current node)
    - h(x) = estimated cost from current node to goal
    - **A* Tree Search** – Doesn't check for revisited states
    - **A* Graph Search** – Keeps track of explored states to avoid loops

Feature	Uninformed Search	Informed (Heuristic) Search
Knowledge	Does not use any domain-specific knowledge	Uses problem-specific heuristic information
Guidance	Explores blindly, no guidance to goal	Guided by heuristic to reach goal faster
Examples	BFS, DFS, UCS, DLS, IDS	Greedy Best-First Search, A* Search
Evaluation Function	No evaluation function used	Uses evaluation function: `f(n) = g(n) + h(n)`
Efficiency	Often less efficient and slower	More efficient in terms of time and space
Optimality	Some are optimal (e.g., UCS, BFS)	A* is optimal if heuristic is admissible and consistent
Heuristic Function	Not used	Required (e.g., `h(n)`)
Use Case	When no extra knowledge is available	When heuristic knowledge is available

Trees

A tree is a hierarchical structure used to represent problems where decisions branch into sub-decisions. It is commonly used in search algorithms, decision-making, data storage, and optimization.

- Represents state space in search algorithms
- Supports recursive traversal and backtracking
- Avoids cycles, making it ideal for tree search algorithms
- Efficient in organizing data for quick access and search

Basic Terms of Tree (Related to Algorithms):

1. **Node** – A fundamental unit of a tree that contains a value or data.
2. **Root** – The topmost node of a tree from which all nodes descend.
3. **Edge** – The connection between two nodes in a tree.
4. **Parent** – A node that has one or more child nodes directly connected below it.
5. **Child** – A node that descends directly from another node (its parent).
6. **Sibling** – Nodes that share the same parent node.
7. **Leaf (External Node)** – A node with no children; an endpoint of a tree.
8. **Internal Node** – A node that has at least one child.
9. **Subtree** – A tree consisting of a node and all its descendants.
10. **Depth (Level)** – The number of edges from the root node to a particular node.
11. **Height** – The length of the longest path from a node down to a leaf.
12. **Degree** – The number of children a node has.
13. **Branching Factor** – The maximum number of children any node in the tree can have.
14. **Binary Tree** – A tree where each node has at most two children.
15. **Full Binary Tree** – A binary tree in which every node has either 0 or 2 children.
16. **Complete Binary Tree** – A binary tree where all levels except possibly the last are fully filled, and all nodes are as far left as possible.
17. **Balanced Tree** – A tree where the height difference between the left and right subtrees of every node is small (often at most 1).
18. **Traversal** – The method of visiting all nodes in a tree systematically (Preorder, Inorder, Postorder).
19. **Search Tree** – A tree structured to allow efficient searching (e.g., Binary Search Tree).
20. **Forest** – A collection of disjoint trees.

Types of Tree-Based Algorithms:

- **Tree Search Algorithms** (used in AI):
    - **Depth-First Search (DFS)** – Explores as deep as possible first
    - **Breadth-First Search (BFS)** – Explores all nodes level by level
    - *A Tree Search** – Uses cost functions (f(x) = g(x) + h(x))
    - **Greedy Tree Search** – Chooses path that seems best (h(x))

- **Decision Trees** – Used in machine learning for classification
- **Minimax Tree** – Used in games and adversarial search
- **Binary Search Tree (BST)** – Used for efficient data lookup
- **Spanning Trees** – Used in networking algorithms (e.g., MST)

Breadth-First Search (BFS)

BFS is a search algorithm that explores all the nodes at the present depth level before moving on to nodes at the next level. It uses a **queue** data structure to keep track of nodes to visit.

- BFS explores nodes level by level.
- It uses a **queue** to keep track of nodes to visit.
- It guarantees the shortest path (in terms of the number of edges) if the cost between nodes is equal.
- **Space Complexity:** O(b^d) **—** Because it stores all nodes at the current depth level in the queue.
- **Time Complexity: O(b^d) —** Because BFS explores all nodes level by level, and the number of nodes grows exponentially with the depth.
- Where **b** = branching factor (average number of children per node),
  and **d** = depth of the shallowest solution

How BFS Works

1. **Initialize:** Start by putting the start node into a queue.
2. **Explore:** Remove the node at the front of the queue.
3. **Goal Check:** If this node is the goal, stop and return the path.
4. **Expand:** If not, add all its unexplored neighbors (children) to the back of the queue.
5. **Repeat:** Continue the process until you find the goal or the queue is empty (meaning no solution).

Advantages of BFS:

- **Complete:** It will find a solution if one exists.
- **Optimal:** Finds the shortest path in terms of number of steps (if all edges have the same cost).
- **Systematic:** Explores level by level, so it won't get stuck deep in a wrong branch.

Disadvantages of BFS:

- **Memory Intensive:** Stores all nodes in the current level, so it can use a lot of memory.
- **Slow:** Explores all nodes level-wise, so can be slow if the search space is large.
- **Not suitable** if the search tree is very deep or infinite.

Depth-First Search (DFS)

DFS is a search algorithm that explores as far as possible along each branch before backtracking. It uses a stack (or recursion) to keep track of nodes to visit.

- DFS explores nodes by going deep down one branch before exploring others.
- It uses a stack or recursion for node tracking.
- It does **not** guarantee the shortest path.
- **Space Complexity:** O(b * m) — because it stores nodes along the current path
- **Time Complexity:** O(b^m) — because, in the worst case, DFS may explore all nodes down to maximum depth *m*.
- *b* = average number of children per node (branching factor), *m* = maximum depth of the tree.

How DFS Works

1. **Initialize:** Push the start node onto the stack.
2. **Explore:** Pop the top node from the stack.
3. **Goal Check:** If this node is the goal, stop and return the path.
4. **Expand:** If not, push all its unexplored neighbors (children) onto the stack.
5. **Repeat:** Continue until the goal is found or the stack is empty (no solution).

Advantages of DFS

- Uses less memory compared to BFS (stores only the current path).
- Can find a solution without exploring all nodes.
- Easy to implement using recursion or a stack.

Disadvantages of DFS

- Not guaranteed to find the shortest path.
- Can get stuck going deep in infinite or very deep branches.
- Not complete for infinite-depth or cyclic graphs unless cycle checking is implemented.

Depth-Limited Search (DLS)

DLS is a variation of DFS where the search is limited to a fixed maximum depth **L** to avoid going too deep or infinitely deep in the search tree.

- **Space Complexity:** O(L) — It is a DFS variant and stores only the current path.
- **Time Complexity:** O(b^L)— Expands nodes up to the depth limit L, where b is the branching factor
- **How it works:** DFS is performed but nodes deeper than the limit **L** are not expanded.

Advantages:

- Prevents infinite loops in infinite-depth spaces.
- Uses less memory than BFS.

Disadvantages:

- Not complete if the solution is beyond the depth limit.
- Choosing the right depth limit can be hard.

Use case: When you know or can estimate a reasonable depth limit for the solution.

Iterative Deepening Depth-First Search (IDDFS)

IDDFS combines the space efficiency of DFS with the completeness of BFS by repeatedly performing DLS with increasing depth limits from 0 up to the solution depth.

- **Space Complexity:** O(d) — Like DFS, it only stores the current path up to depth ddd
- **Time Complexity:** O(b^d)— Although nodes at shallower depths are revisited multiple times, the total cost is dominated by the deepest level **d**.
- **How it works**
  - Run DLS with depth limit = 0.
  - If no solution, run DLS with depth limit = 1.
  - Increase depth limit by 1 and repeat until the goal is found.

Advantages

- Complete like BFS (will find a solution if it exists).
- Uses less memory like DFS.
- Finds the shallowest solution.

Disadvantages

- Repeats search on nodes at shallower depths multiple times (but this overhead is usually small).

Use case: When the depth of the solution is unknown and the search tree is large or infinite.

Uniform-Cost Search (UCS)

UCS is a search algorithm that expands the node with the lowest total path cost (g(n)) first. It is an uninformed (blind) search but considers cost, unlike BFS or DFS.

- Uses a Priority Queue (usually a min-heap) to select the node with the lowest path cost.
- Suitable when path costs vary between nodes.
- Guarantees the least-cost (optimal) path to the goal.
- Works like BFS when all step costs are equal.

How Uniform-Cost Search (UCS) Works:

1. **Initialize**: Insert the start node into a priority queue with a total path cost of **0**.
2. **Pick Node**: Remove the node with the lowest total path cost from the priority queue.
3. **Goal Test**: If this node is the goal, return the path and stop.
4. **Expand Node**: If not, expand the node — for each child (neighbor):
    - Calculate the cumulative cost to reach that neighbor.
    - If the neighbor is not in the queue or a lower-cost path is found, add/update it in the queue.
5. **Repeat**: Continue steps 2–4 until the goal node is found or the queue becomes empty.

Advantages:

- Complete: Finds a solution if one exists.
- Optimal: Always finds the least-cost path.
- Works with variable path costs.

Disadvantages:

- Can be slow in large graphs.
- High memory usage due to storing many paths in the priority queue.
- Not suitable if the goal is deep and costs are small.

Use Case: You need the lowest-cost solution, not just the one with the fewest steps.

Greedy Best-First Search (GBFS)

Greedy Best-First Search is a **heuristic search algorithm** that selects the node that appears to be **closest to the goal** based on a heuristic function.

- **Data Structure Used: Priority Queue**, ordered by h(n) (not total path cost).
- It uses a **heuristic function** h(n) that estimates the cost from node n to the goal
- It **greedily picks** the node with the lowest h(n) value.
- **Time Complexity**: O(b^m), where b = branching factor and m = maximum depth of the search space
- **Space Complexity**: O(b^m), because it stores all generated nodes in the queue.

How GBFS Works (Step-by-Step):

1. **Initialize**: Start from the initial node and put it into a priority queue based on h(n).
2. **Select**: Remove the node with the **lowest heuristic value**.
3. **Goal Check**: If it is the goal, return the path.
4. **Expand**: Otherwise, expand the node and add its neighbors to the queue with their h(n) values.
5. **Repeat**: Continue until the goal is found or the queue is empty.

Advantages:

- Faster than uninformed search if the heuristic is good.
- Requires less memory than BFS in some cases.

Disadvantages:

- **Not optimal** – may miss the best (lowest-cost) path.
- **Not complete** – might not find a solution if it gets stuck in loops or dead ends.
- Depends heavily on the **quality of the heuristic**.

Use Case:

Use GBFS when:

- You have a **good heuristic** that can estimate distance to the goal.
- **Speed is more important than optimality**.

A* (A-Star) Search Algorithm

A* is a **best-first search algorithm** that finds the **shortest and most optimal path** to the goal by combining **f(n) = g(n) + h(n)**

- **Actual cost from start**: g(n)
- **Estimated cost to goal** (heuristic): h(n)
- It selects the node with the **lowest total estimated cost** by **f(n) = g(n) + h(n)**
- **Data Structure Used: Priority Queue** (sorted by f(n))
- **Time Complexity**:
    - Worst case: **O(b^d)** if heuristic is poor
    - Best case (with good heuristic): much better
- **Space Complexity**: **O(b^d)**
  (It stores all generated nodes in memory)

How A* Works (Step-by-Step):

1. **Start**: Put the start node into a **priority queue**, with f(n) = g(n) + h(n), where g(n) = 0 and h(n) is the heuristic.
2. **Select**: Pick the node with the **lowest f(n)** from the queue.
3. **Goal Test**: If it's the goal node, return the path.
4. **Expand**: **Expand**: Generate all valid neighbours. For each neighbour, calculate g(n) and f(n). If it's not in the queue or has a lower cost, add or update it in the queue.
5. **Repeat**: Continue until the goal is found or the queue is empty.

Advantages:

- **Optimal** (if heuristic is admissible and consistent)
- **Complete** (guaranteed to find a solution if one exists)
- **Flexible** – balances between DFS, BFS, UCS based on the heuristic

Disadvantages:

- **Memory intensive** – stores all visited nodes
- **Performance** depends on the **quality of the heuristic**

Use Case: Use A* when you need the **shortest and optimal path**

Machine Learning

Machine Learning (ML) is a branch of AI focused on building systems that **learn from data** and **improve automatically** over time without being explicitly programmed for every task.

- Learns from past data to make better decisions or predictions.
- No need for manual rules; it finds patterns automatically.
- Improves accuracy as more data is provided.
- Detects hidden patterns that humans may miss.
- Mainly used for prediction tasks like spam detection or price forecasting.
- Handles large amounts of data efficiently.
- Adapts to new data over time.
- Used in many areas like healthcare, finance, and online services.

Stages of Machine Learning

1. **Data Collection**: Gather data from various sources like databases, files, images, sounds, or web scraping. Organize data into usable formats (e.g., CSV).
2. **Data Pre-processing**: Clean the data by removing duplicates, fixing errors, handling missing values, and formatting data properly.
3. **Choosing the Right Model**: Select a machine learning algorithm (e.g., linear regression, decision trees, neural networks) based on data type, problem, size, and resources.
4. **Training the Model**: Use the prepared data to train the model so it learns patterns and relationships.
5. **Evaluating the Model**: Test the trained model on unseen data to measure accuracy and performance.
6. **Hyperparameter Tuning and Optimization**: Adjust model parameters and use techniques like cross-validation to improve performance.
7. **Predictions and Deployment**: Use the optimized model to predict outcomes on new data and integrate it into real-world applications for decision-making.

Types of Machine Learning

1. **Supervised Learning:** The model is trained on labeled data, where the input data is paired with correct output labels. The goal is to learn a mapping from inputs to outputs.
    - **Example:** Email spam detection (emails labeled as spam or not spam)
    - **Usage:** Classification (e.g., handwriting recognition), regression (e.g., house price prediction).
    - **Algorithms:** Linear Regression, Decision Trees, Random Forests, KNN, SVM
2. **Unsupervised Learning:** The model is trained on unlabeled data and tries to find patterns or groupings without predefined labels.
    - **Example:** Customer segmentation based on purchase behavior.
    - **Usage:** Clustering (e.g., market segmentation), dimensionality reduction (e.g., PCA for visualization).
    - **Algorithms:** K-Means Clustering, Hierarchical Clustering, DBSCAN, PCA
3. **Semi-Supervised Learning:** Uses a small amount of labeled data combined with a large amount of unlabeled data to improve learning accuracy.
    - **Example:** Image recognition with few labeled images and many unlabeled ones.
    - **Usage:** Helpful when labeling data is expensive or time-consuming.
    - **Algorithms:** Label Propagation, Semi-Supervised SVM, Self-training models
4. **Reinforcement Learning:** Learns by trial and error, receiving rewards or penalties from the environment.
    - **Example:** Teaching a robot to walk, training AI to play chess.
    - **Usage:** Used in robotics, gaming AI, and autonomous systems.
    - **Algorithms:** Q-Learning, Deep Q Networks (DQN), Policy Gradient Methods.

Supervised Learning

Supervised Learning is a type of machine learning where a model is trained on a labeled dataset. In this dataset, each input is paired with the correct output (label). The model learns the relationship between the input features and the output labels so that it can accurately predict the output for new, unseen data.

- The dataset contains input-output pairs; inputs are features, outputs are labels or target values.
- It is mainly used for: Classification (predicting categories), Regression (predicting continuous values)
- The learning process minimizes the difference between the predicted output and actual output (error).
- Model performance is evaluated using: Accuracy, precision, recall (for classification), Error metrics like MSE, RMSE (for regression)
- Requires a large and representative labeled dataset to perform well.

How Supervised Learning Works

1. **Data Collection:** Gather a labeled dataset, where each data point has inputs (features) and a known output (label).
2. **Data Preparation:** Clean the data, handle missing values, normalize or scale features, and split the data into training and testing sets.
3. **Model Selection:** Choose a suitable supervised learning algorithm based on the **problem type** (classification or regression) and data characteristics.
4. **Training:** The model learns by adjusting its parameters to minimize the error on the training data using techniques like gradient descent.
5. **Evaluation:** Test the model on the unseen test data to measure how well it predicts new examples.
6. **Prediction:** Use the trained model to predict outputs for new input data.

Types of Supervised Learning Problems

1. **Classification:** The model predicts which **category or group** an input belongs to. The output is a **label** or **class**.
    - *Example:* Email spam detection, image recognition, medical diagnosis
2. **Regression:** The model predicts a **continuous numeric value** based on input data.
    - *Example:* Predicting house prices, stock market forecasting, temperature prediction.

Common Supervised Learning Algorithms

- **Linear Regression:**
  Predicts a continuous value based on a linear combination of input features.
- **Logistic Regression:**
  Predicts probability of class membership, used for binary classification problems.
- **Decision Trees:**
  Models decisions and their possible consequences in a tree-like structure.
- **Random Forests:**
  Ensemble of decision trees to improve accuracy and reduce overfitting.
- **Support Vector Machines (SVM):**
  Finds the best boundary (hyperplane) that separates classes.
- **K-Nearest Neighbors (KNN):**
  Classifies data points based on the classes of nearest neighbors.

Advantages

- Produces accurate and reliable models with labeled data.
- Easy to evaluate and interpret using known outputs.
- Effective for a wide range of real-world problems.

Disadvantages

- Requires large amounts of labeled data, which may be expensive or time-consuming to collect.
- Performance depends heavily on data quality and label accuracy.
- Can overfit if the model is too complex or the data is noisy.

Unsupervised Learning

Unsupervised Learning is a type of machine learning where the model is trained on unlabeled data. This means the dataset contains only input features without any corresponding output labels. The model tries to identify patterns, structures, or groupings in the data without being told what to look for.

- The dataset contains only input data (features), without known outputs or labels.
- It is mainly used for: Clustering (grouping similar data points), Dimensionality Reduction (simplifying data while preserving structure).
- The learning process aims to discover hidden patterns or groupings within the dataset.
- Model performance is evaluated using metrics like: Silhouette Score, Inertia, or visual inspection for clustering; reconstruction error for dimensionality reduction.
- Suitable for exploratory data analysis and when labeled data is unavailable or costly to obtain.

How Unsupervised Learning Works

1. **Data Collection**: Gather an unlabeled dataset, where each data point has input features but no corresponding output or label.
2. **Data Preparation**: Clean the data, handle missing values, and normalize or scale features.
3. **Model Selection**: Choose a suitable unsupervised learning algorithm based on the problem type (clustering, dimensionality reduction).
4. **Training**: The model identifies patterns or structures in the data without guidance, using mathematical techniques (e.g., distance metrics, matrix decomposition).
5. **Evaluation**: Analyze the results to assess the quality of the discovered structure using clustering validation metrics or visual inspection.
6. **Application**: Use the learned patterns to segment customers, detect anomalies, compress data, or as preprocessing for supervised learning.

Types of Unsupervised Learning Problems

1. **Clustering**: The model groups data points into clusters based on their similarity.
    - **Example**: Customer segmentation, market basket analysis, social network analysis.
2. **Dimensionality Reduction**: The model reduces the number of input variables while preserving important information.
    - **Example**: Data visualization, feature compression, noise reduction.

Common Unsupervised Learning Algorithms

- **K-Means Clustering**: Partitions data into K clusters by minimizing the variance within each cluster.
- **Hierarchical Clustering**: Builds a tree (dendrogram) of nested clusters based on data similarity.
- **DBSCAN** (Density-Based Spatial Clustering of Applications with Noise): Groups data based on density, useful for arbitrary-shaped clusters.
- **PCA (Principal Component Analysis)**: Reduces dimensionality by transforming data into new features (principal components) that capture the most variance.
- **t-SNE (t-Distributed Stochastic Neighbor Embedding)**: A nonlinear dimensionality reduction technique for visualizing high-dimensional data.

Advantages

- Can discover hidden patterns in data without the need for labeled outputs.
- Useful for exploratory data analysis, customer segmentation, and preprocessing.
- Helpful when labeling data is difficult, expensive, or impossible.

Disadvantages

- Hard to evaluate performance since there are no true labels.
- Interpretability of the model outputs can be challenging.
- May find patterns that are not meaningful, especially in noisy or irrelevant data.
- Choosing the right number of clusters or components is often difficult and subjective.

Semi-Supervised Learning

Semi-Supervised Learning is a type of machine learning that combines a small amount of labeled data with a large amount of unlabeled data during training. This approach leverages the efficiency of unlabeled data to improve learning accuracy when labeled data is scarce or expensive to obtain.

- The dataset contains a small portion of labeled data and a large portion of unlabeled data.
- It is mainly used when labeling data is costly or time-consuming, but a large quantity of unlabeled data is available.
- The learning process uses the labeled data to guide the learning and the unlabeled data to discover data structure and enhance generalization.
- Model performance is evaluated using standard metrics: Accuracy, Precision, Recall, F1-score for classification; MSE, RMSE for regression (on the labeled validation/test set).
- Provides a good balance between supervised and unsupervised learning, reducing the dependency on fully labeled datasets.

How Semi-Supervised Learning Works

1. **Data Collection**: Gather a dataset that includes a small set of labeled examples and a large set of unlabeled examples.
2. **Data Preparation**: Clean the data, handle missing values, and normalize features. Split the labeled data into training and validation/test sets.
3. **Model Initialization**: Start training with the labeled data using a supervised learning algorithm.
4. **Incorporate Unlabeled Data**: Use the model's predictions on unlabeled data to help improve training (e.g., pseudo-labeling, self-training).
5. **Refinement**: Retrain or fine-tune the model using both labeled and confidently predicted pseudo-labeled data.
6. **Evaluation**: Use the labeled validation/test set to assess model performance.
7. **Prediction**: Apply the trained model to predict outputs for new input data.

Types of Semi-Supervised Learning Approaches

1. **Self-Training**: The model trained on labeled data predicts labels for unlabeled data, and the most confident predictions are added to the training set.
    - *Example*: Sentiment analysis with few labeled reviews and many unlabeled ones.

2. **Co-Training**: Two models are trained on different views of the data and teach each other by sharing confident predictions.
    - *Example*: Web page classification using content and hyperlink structure.
3. **Graph-Based Methods**: Represent data as a graph and use label propagation techniques to spread labels from labeled to unlabeled points.
    - *Example*: Social network analysis.
4. **Semi-Supervised Generative Models**: Use techniques like Variational Autoencoders (VAE) or GANs that can generate data representations using both labeled and unlabeled data.

Common Algorithms Used in Semi-Supervised Learning

- **Semi-Supervised SVM**: Uses both labeled and unlabeled data to find a decision boundary that respects the data structure.
- **Label Propagation and Label Spreading**: Graph-based methods that spread labels through data points.
- **Self-Training with Neural Networks**: Trains a model on labeled data, uses it to label unlabeled data, and retrains with high-confidence examples.
- **Generative Adversarial Networks (GANs)**: Modified to incorporate a small amount of labeled data for better representation learning.

Advantages

- Reduces the need for large labeled datasets, saving time and cost.
- Improves performance over purely supervised or unsupervised methods when labeled data is limited.
- Takes advantage of large amounts of unlabeled data that are often readily available.
- Can produce more robust models by learning the underlying structure of the data.

Disadvantages

- Quality of pseudo-labels from the model can affect performance if incorrect labels are introduced.
- Sensitive to noise and outliers in the unlabeled data.
- Requires careful confidence thresholding to avoid training on incorrect predictions.
- Implementation and tuning can be complex compared to purely supervised or unsupervised models.