

UNIT- I

Introduction: AI problems, Agents and Environments, Structure of Agents, Problem Solving Agents **Basic Search Strategies:** Problem Spaces, Uninformed Search (Breadth First, Depth-First Search, Depth-first with Iterative Deepening), Heuristic Search (Hill Climbing, Generic Best-First, A*), Constraint Satisfaction (Backtracking, Local Search)

Introduction:

- Artificial Intelligence is concerned with the design of intelligence in an artificial device. The term was coined by John McCarthy in 1956.
- Intelligence is the ability to acquire, understand and apply the knowledge to achieve goals in the world.
- AI is the study of the mental faculties through the use of computational models
- AI is the study of intellectual/mental processes as computational processes.
- AI program will demonstrate a high level of intelligence to a degree that equals or exceeds the intelligence required of a human in performing some task.
- AI is unique, sharing borders with Mathematics, Computer Science, Philosophy, Psychology, Biology, Cognitive Science and many others.
- Although there is no clear definition of AI or even Intelligence, it can be described as an attempt to build machines that like humans can think and act, able to learn and use knowledge to solve problems on their own.

Sub Areas of AI:

1) Game Playing

Deep Blue Chess program beat world champion Gary Kasparov

2) Speech Recognition

PEGASUS spoken language interface to American Airlines' EASY SABRE reservation system, which allows users to obtain flight information and make reservations over the

telephone. The 1990s has seen significant advances in speech recognition so that limited systems are now successful.

3) **Computer Vision**

Face recognition programs in use by banks, government, etc. The ALVINN system from CMU autonomously drove a van from Washington, D.C. to San Diego (all but 52 of 2,849 miles), averaging 63 mph day and night, and in all weather conditions. Handwriting recognition, electronics and manufacturing inspection, photo interpretation, baggage inspection, reverse engineering to automatically construct a 3D geometric model.

4) **Expert Systems**

Application-specific systems that rely on obtaining the knowledge of human experts in an area and programming that knowledge into a system.

- a. **Diagnostic Systems:** MYCIN system for diagnosing bacterial infections of the blood and suggesting treatments. Intellipath pathology diagnosis system (AMA approved). Pathfinder medical diagnosis system, which suggests tests and makes diagnoses. Whirlpool customer assistance center.
- b. **System Configuration**
DEC's XCON system for custom hardware configuration. Radiotherapy treatment planning.
- c. **Financial Decision Making**
Credit card companies, mortgage companies, banks, and the U.S. government employ AI systems to detect fraud and expedite financial transactions. For example, AMEX credit check.
- d. **Classification Systems**
Put information into one of a fixed set of categories using several sources of information. E.g., financial decision making systems. NASA developed a system for classifying very faint areas in astronomical images into either stars or galaxies with very high accuracy by learning from human experts' classifications.

5) **Mathematical Theorem Proving**

Use inference methods to prove new theorems.

6) **Natural Language Understanding**

AltaVista's translation of web pages. Translation of Caterpillar Truck manuals into 20 languages.

7) Scheduling and Planning

Automatic scheduling for manufacturing. DARPA's DART system used in Desert Storm and Desert Shield operations to plan logistics of people and supplies. American Airlines rerouting contingency planner. European space agency planning and scheduling of spacecraft assembly, integration and verification.

8) Artificial Neural Networks:

9) Machine Learning

Applications of AI:

AI algorithms have attracted close attention of researchers and have also been applied successfully to solve problems in engineering. Nevertheless, for large and complex problems, AI algorithms consume considerable computation time due to stochastic feature of the search approaches

1. Business; financial strategies
2. Engineering: check design, offer suggestions to create new product, expert systems for all engineering problems
3. Manufacturing: assembly, inspection and maintenance
4. Medicine: monitoring, diagnosing
5. Education: in teaching
6. Fraud detection
7. Object identification
8. Information retrieval
9. Space shuttle scheduling

Building AI Systems:

1) Perception

Intelligent biological systems are physically embodied in the world and experience the world through their sensors (senses). For an autonomous vehicle, input might be images from a camera and range information from a rangefinder. For a medical diagnosis

system, perception is the set of symptoms and test results that have been obtained and input to the system manually.

2) Reasoning

Inference, decision-making, classification from what is sensed and what the internal "model" is of the world. Might be a neural network, logical deduction system, Hidden Markov Model induction, heuristic searching a problem space, Bayes Network inference, genetic algorithms, etc. Includes areas of knowledge representation, problem solving, decision theory, planning, game theory, machine learning, uncertainty reasoning, etc.

3) Action

Biological systems interact within their environment by actuation, speech, etc. All behavior is centered around actions in the world. Examples include controlling the steering of a Mars rover or autonomous vehicle, or suggesting tests and making diagnoses for a medical diagnosis system. Includes areas of robot actuation, natural language generation, and speech synthesis.

The definitions of AI:

<p>a) "The exciting new effort to make computers think . . . <i>machines with minds</i>, in the full and literal sense" (Haugeland, 1985)</p> <p>"The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning..."(Bellman, 1978)</p>	<p>b) "The study of mental faculties through the use of computational models" (Charniak and McDermott, 1985)</p> <p>"The study of the computations that make it possible to perceive, reason, and act" (Winston, 1992)</p>
---	--

<p>c) "The art of creating machines that perform functions that require intelligence when performed by people" (Kurzweil, 1990)</p> <p>"The study of how to make computers do things at which, at the moment, people are better" (Rich and Knight, 1991)</p>	<p>d) "A field of study that seeks to explain and emulate intelligent behavior in terms of computational processes" (Schalkoff, 1990)</p> <p>"The branch of computer science that is concerned with the automation of intelligent behavior" (Luger and Stubblefield, 1993)</p>
--	--

The definitions on the top, **(a)** and **(b)** are concerned with **reasoning**, whereas those on the bottom, **(c)** and **(d)** address **behavior**. The definitions on the left, **(a)** and **(c)** measure success in terms of human performance, and those on the right, **(b)** and **(d)** measure the ideal concept of intelligence called rationality

Intelligent Systems:

In order to design intelligent systems, it is important to categorize them into four categories (Luger and Stubblefield 1993), (Russell and Norvig, 2003)

1. Systems that think like humans
2. Systems that think rationally
3. Systems that behave like humans
4. Systems that behave rationally

	Human-Like	Rationality
Think:	<p>Cognitive Science Approach</p> <p><i>"Machines that think like humans"</i></p>	<p>Laws of thought Approach</p> <p><i>"Machines that think Rationally"</i></p>
Act:	<p>Turing Test Approach</p> <p><i>"Machines that behave like humans"</i></p>	<p>Rational Agent Approach</p> <p><i>"Machines that behave Rationally"</i></p>

Cognitive Science: Think Human-Like

- a. Requires a model for human cognition. Precise enough models allowsimulation by computers.
- b. Focus is not just on behavior and I/O, but looks like reasoning process.
- c. Goal is not just to produce human-like behavior but to produce a sequence of steps of thereasoning process, similar to the steps followed by a human in solving the same task.

Laws of thought: Think Rationally

- a. The study of mental faculties through the use of computational models; that it is, thestudy of computations that make it possible to perceive reason and act.
- b. Focus is on inference mechanisms that are probably correct and guarantee an optimal solution.
- c. Goal is to formalize the reasoning process as a system of logical rules and procedures ofinference.
- d. Develop systems of representation to allow inferences to be like

“Socrates is a man. All men are mortal. Therefore Socrates is mortal”

Turing Test: Act Human-Like

- a. The art of creating machines that perform functions requiring intelligence when performed by people; that it is the study of, how to make computers do things which, at the moment, people do better.
- b. Focus is on action, and not intelligent behavior centered around the representation of the world
- c. Example: Turing Test
 - 3 rooms contain: a person, a computer and an interrogator.
 - The interrogator can communicate with the other 2 by teletype (to avoidthe machine imitate the appearance of voice of the person)
 - The interrogator tries to determine which the person is and which themachine is.

- The machine tries to fool the interrogator to believe that it is the human, and the person also tries to convince the interrogator that it is the human.
- If the machine succeeds in fooling the interrogator, then conclude that the machine is intelligent.

Rational agent: Act Rationally

- Tries to explain and emulate intelligent behavior in terms of computational process; that it is concerned with the automation of the intelligence.
- Focus is on systems that act sufficiently if not optimally in all situations.
- Goal is to develop systems that are rational and sufficient

Agents and Environments:

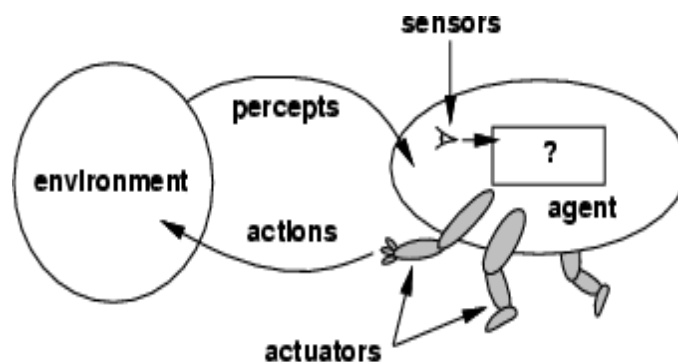


Fig 2.1: Agents and Environments

Agent:

An *Agent* is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.

- ✓ A *human agent* has eyes, ears, and other organs for sensors and hands, legs, mouth, and other body parts for actuators.
- ✓ A *robotic agent* might have cameras and infrared range finders for sensors and various motors for actuators.
- ✓ A *software agent* receives keystrokes, file contents, and network packets as

sensory inputs and acts on the environment by displaying on the screen, writing files, and sending network packets.

Percept:

We use the term percept to refer to the agent's perceptual inputs at any given instant.

Percept Sequence:

An agent's percept sequence is the complete history of everything the agent has ever perceived.

Agent function:

Mathematically speaking, we say that an agent's behavior is described by the agent function that maps any **given percept sequence to an action**.

Agent program

Internally, the agent function for an artificial agent will be implemented by an agent program. It is important to keep these two ideas distinct. The agent function is an abstract

mathematical description; the agent program is a concrete implementation, running on the agent architecture.

To illustrate these ideas, we will use a very simple example—the vacuum-cleaner world shown in **Fig 2.1.5**. This particular world has just two locations: squares A and B. The vacuum agent perceives which square it is in and whether there is dirt in the square. It can choose to move left, move right, suck up the dirt, or do nothing. One very simple agent function is the following: if the current square is dirty, then suck, otherwise move to the other square. A partial tabulation of this agent function is shown in **Fig 2.1.6**.

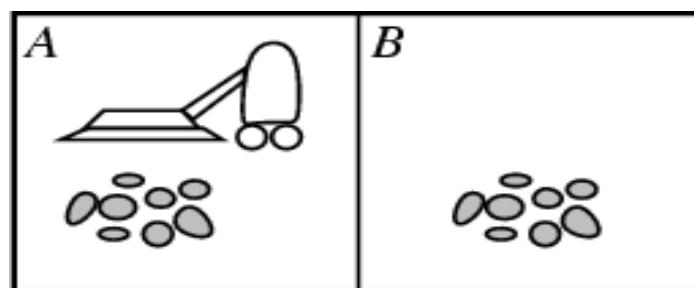


Fig 2.1.5: A vacuum-cleaner world with just two locations.

Agent function

Percept Sequence	Action
[A, Clean]	Right
[A, Dirty]	Suck
[B, Clean]	Left
[B, Dirty]	Suck
[A, Clean], [A, Clean]	Right
[A, Clean], [A, Dirty]	Suck
...	

Fig 2.1.6: Partial tabulation of a simple agent function for the example: vacuum-cleaner world shown in the Fig2.1.5

Function **REFLEX-VACCUM-AGENT** ([location, status]) returns an

action If **status=Dirty** then return **Suck**

else if **location = A** then return **Right**

else if **location = B** then return **Left**

Fig 2.1.6(i): The REFLEX-VACCUM-AGENT program is invoked for each new percept (location, status) and returns **an** action each time

- A **Rational agent** is one that does the right thing. we say that the right action is the one that will cause the agent to be most successful. That leaves us with the problem of deciding how and when to evaluate the agent's success.
We use the term performance measure for the how—the criteria that determine how successful an agent is.

- ✓ Ex-Agent cleaning the dirty floor
- ✓ Performance Measure-Amount of dirt collected
- ✓ When to measure-Weekly for better results

What is rational at any given time depends on four things:

- The performance measure defining the criterion of success
- The agent’s prior knowledge of the environment
- The actions that the agent can perform
- The agent’s percept sequence up to now.

Omniscience ,Learning and Autonomy:

- We need to distinguish between rationality and omniscience. An **Omniscient agent** knows the actual outcome of its actions and can act accordingly but omniscience is impossible in reality.
- Rational agent not only gathers information but also **learns** as much as possible from what it perceives.
- If an agent just relies on the prior knowledge of its designer rather than its own percepts then the agent lacks **autonomy**.
- A system is autonomous to the extent that its behavior is determined by its own experience.
- A rational agent should be autonomous.
 - E.g., a clock(lacks autonomy)
- No input (percepts)
- Run only by its own algorithm (prior knowledge)
- No learning, no experience, etc.

ENVIRONMENTS:

The Performance measure, the environment and the agents actuators and sensors comes under the heading task environment. We also call this as PEAS(Performance,Environment,Actuators,Sensors)

Agent Type	Performance Measure	Environment	Actuators	Sensors
Taxi driver	Safe, fast, legal, comfortable trip, maximize profits	Roads, other traffic, pedestrians, customers	Steering, accelerator, brake, signal, horn, display	Cameras, sonar, speedometer, GPS, odometer, accelerometer, engine sensors, keyboard

Figure 2.4 PEAS description of the task environment for an automated taxi.

Other PEAS Examples

Agent Type	Performance Measure	Environment	Actuators	Sensors
Medical diagnosis system	healthy patient, costs, lawsuits	patient, hospital, stuff	display questions, tests, diagnoses, treatments, referrals	keyboard entry of symptoms, findings, patient's answers
Satellite image analysis system	correct image categorization	downlink from orbiting satellite	display categorization of scene	color pixel arrays
Part-picking robot	percentage of parts in correct bins	conveyor belt with parts, bins	jointed arm and hand	camera, joint angle sensors
Refinery controller	purity, yield, safety	refinery, operators	valves pumps, heaters displays	temperature, pressure, chemical sensors
Interactive English tutor	student's score on test	set of students, testing agency	display exercises, suggestions, corrections	keyboard entry

Environment-Types:

1. Accessible vs. inaccessible or Fully observable vs Partially Observable:

If an agent sensor can sense or access the complete state of an environment at each point of time then it is a fully observable environment, else it is partially observable.

2. Deterministic vs. Stochastic:

If the next state of the environment is completely determined by the current state and the actions selected by the agents, then we say the environment is deterministic

3. Episodic vs. nonepisodic:

- The agent's experience is divided into "episodes." Each episode consists of the agent perceiving and then acting. The quality of its action depends just on the episode itself, because subsequent episodes do not depend on what actions occur in previous episodes.
- Episodic environments are much simpler because the agent does not need to think ahead.

4. Static vs. dynamic.

If the environment can change while an agent is deliberating, then we say the environment is dynamic for that agent; otherwise it is static.

5. Discrete vs. continuous:

If there are a limited number of distinct, clearly defined percepts and actions we say that the environment is discrete. Otherwise, it is continuous.

Examples of task environments

Task Environment	Observable	Deterministic	Episodic	Static	Discrete	Agents
Crossword puzzle	Fully	Deterministic	Sequential	Static	Discrete	Single
Chess with a clock	Fully	Strategic	Sequential	Semi	Discrete	Multi
Poker	Partially	Strategic	Sequential	Static	Discrete	Multi
Backgammon	Fully	Stochastic	Sequential	Static	Discrete	Multi
Taxi driving	Partially	Stochastic	Sequential	Dynamic	Continuous	Multi
Medical diagnosis	Partially	Stochastic	Sequential	Dynamic	Continuous	Single
Image-analysis	Fully	Deterministic	Episodic	Semi	Continuous	Single
Part-picking robot	Partially	Stochastic	Episodic	Dynamic	Continuous	Single
Refinery controller	Partially	Stochastic	Sequential	Dynamic	Continuous	Single
Interactive English tutor	Partially	Stochastic	Sequential	Dynamic	Discrete	Multi

Figure 2.6 Examples of task environments and their characteristics.

STRUCTURE OF INTELLIGENT AGENTS

- The job of AI is to design the agent program: a function that implements the agent mapping from percepts to actions. We assume this program will run on some sort of ARCHITECTURE computing device, which we will call the architecture.
- The architecture might be a plain computer, or it might include special-purpose hardware for certain tasks, such as processing camera images or filtering audio input. It might also include software that provides a degree of insulation between the raw computer and the agent program, so that we can program at a higher level. In general, the architecture makes the percepts from the sensors available to the program, runs the program, and feeds the program's action choices to the effectors as they are generated.
- The relationship among agents, architectures, and programs can be summed up as follows: agent = architecture + program

Agent Type	Percepts	Actions	Goals	Environment
Medical diagnosis system	Symptoms, findings, patient's answers	Questions, tests, treatments	Healthy patient, minimize costs	Patient, hospital
Satellite image analysis system	Pixels of varying intensity, color	Print a categorization of scene	Correct categorization	Images from orbiting satellite
Part-picking robot	Pixels of varying intensity	Pick up parts and sort into bins	Place parts in correct bins	Conveyor belt with parts
Refinery controller	Temperature, pressure readings	Open, close valves; adjust temperature	Maximize purity, yield, safety	Refinery
Interactive English tutor	Typed words	Print exercises, suggestions, corrections	Maximize student's score on test	Set of students

Figure 2.3 Examples of agent types and their PAGE descriptions.

Agent programs:

- Intelligent agents accept percepts from an environment and generates actions. The early versions of agent programs will have a very simple form (Figure 2.4)
- Each will use some internal data structures that will be updated as new percepts arrive.
- These data structures are operated on by the agent's decision-making procedures to generate an action choice, which is then passed to the architecture to be executed

```

function TABLE-DRIVEN-AGENT(percept) returns action
  static: percepts, a sequence, initially empty
           table, a table, indexed by percept sequences, initially fully specified

  append percept to the end of percepts
  action ← LOOKUP(perceptstable)
  return action

```

Figure 2.5 An agent based on a prespecified lookup table. It keeps track of the percept sequence and just looks up the best action.

Types of agents:

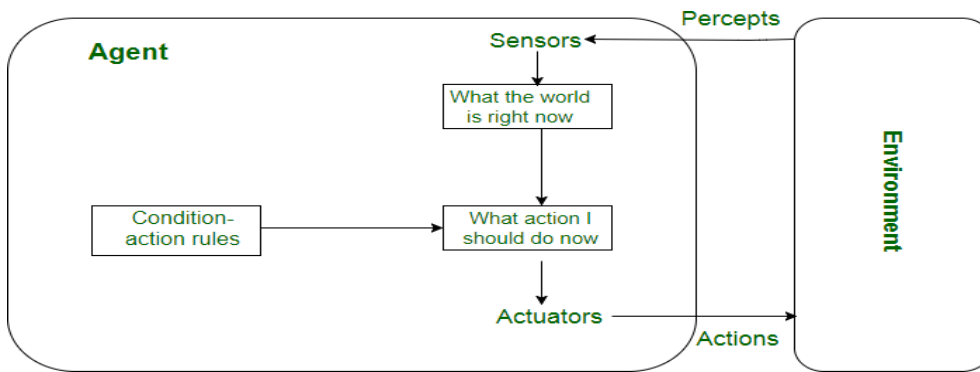
Agents can be grouped into four classes based on their degree of perceived intelligence and capability :

- Simple Reflex Agents

- Model-Based Reflex Agents
- Goal-Based Agents
- Utility-Based Agents

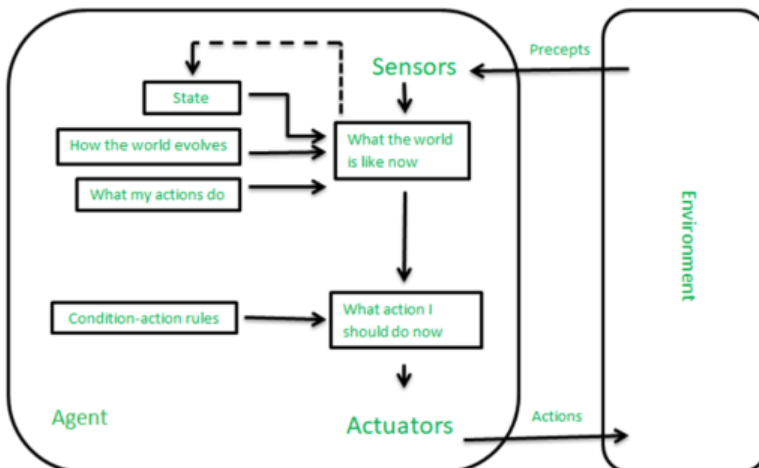
Simple reflex agents:

- Simple reflex agents ignore the rest of the percept history and act only on the basis of the current percept.
- The agent function is based on the condition-action rule.
- If the condition is true, then the action is taken, else not. This agent function only succeeds when the environment is fully observable.



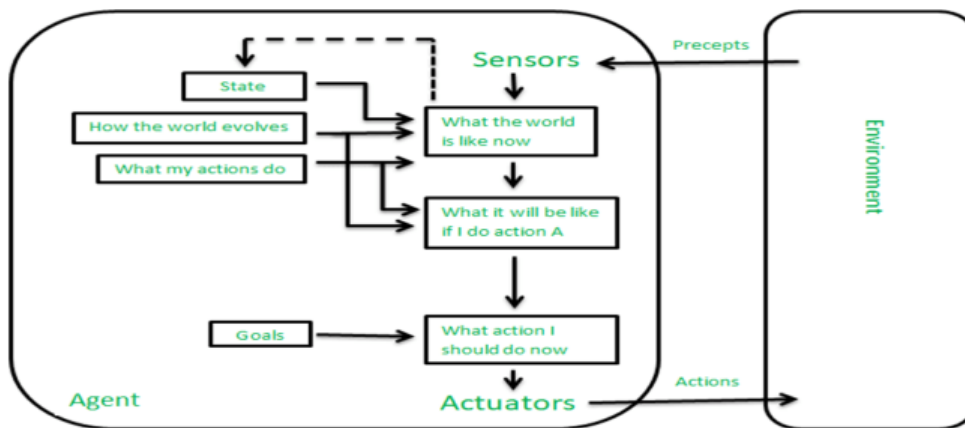
Model-based reflex agents:

- The Model-based agent can work in a partially observable environment, and track the situation.
- A model-based agent has two important factors:
 - Model: It is knowledge about "how things happen in the world," so it is called a Model-based agent.
 - Internal State: It is a representation of the current state based on percept history.



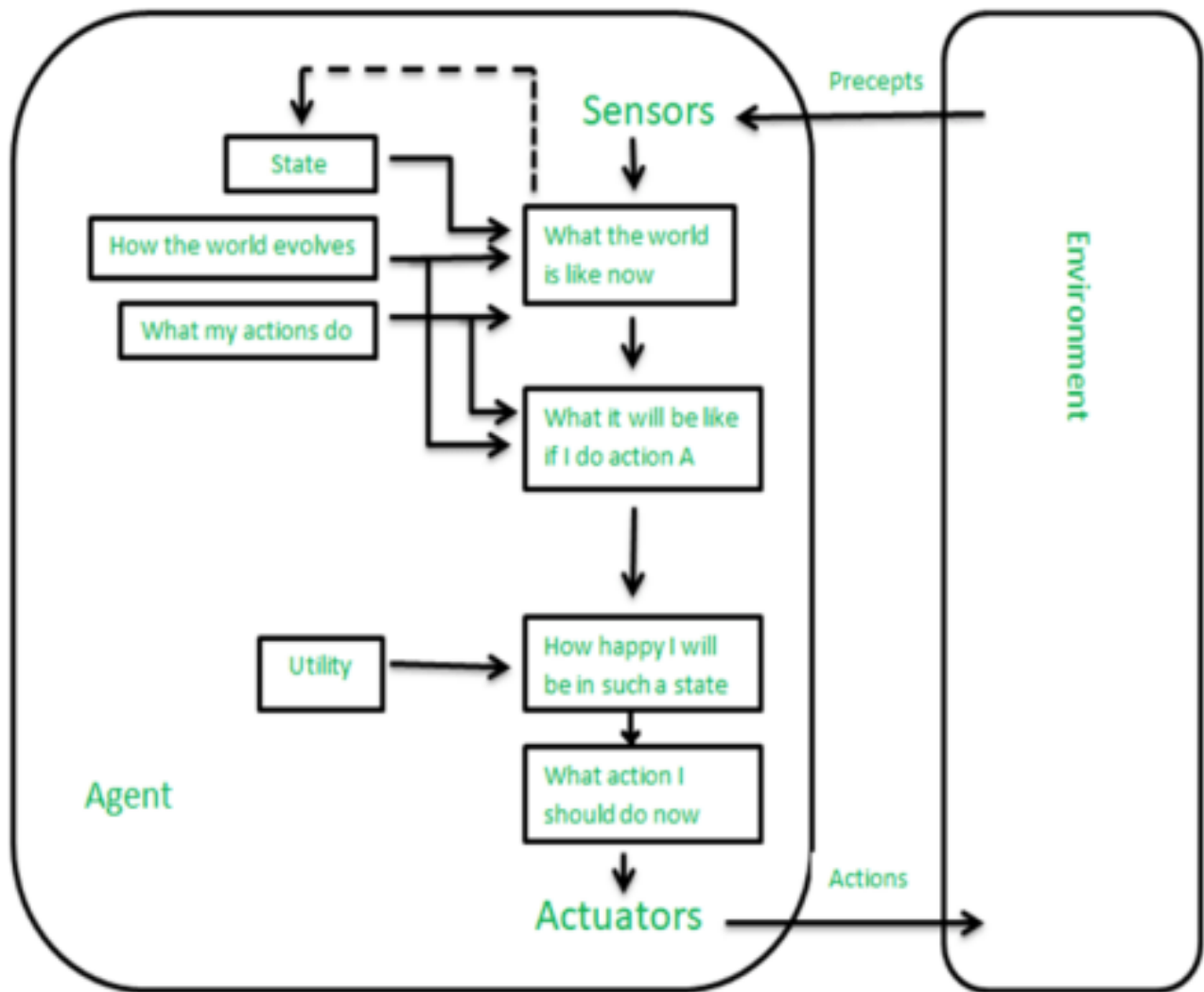
Goal-based agents:

- A goal-based agent has an agenda.
- It operates based on a goal in front of it and makes decisions based on how best to reach that goal.
- A goal-based agent operates as a search and planning function, meaning it targets the goal ahead and finds the right action in order to reach it.
- Expansion of model-based agent.



Utility-based agents:

- A utility-based agent is an agent that acts based not only on what the goal is, but the best way to reach that 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.
- The term utility can be used to describe how "happy" the agent is.



Problem Solving Agents:

- Problem solving agent is a goal-based agent.
- Problem solving agents decide what to do by finding sequence of actions that lead to desirable states.

Goal Formulation:

It organizes the steps required to formulate/ prepare one goal out of multiple goals available.

Problem Formulation:

It is a process of deciding what actions and states to consider to follow goal formulation. The process of looking for a best sequence to achieve a goal is called

Search.

A search algorithm takes a problem as input and returns a solution in the form of action sequences. Once the solution is found the action it recommends can be carried out. This is called **Execution phase.**

Well Defined problems and solutions:

A problem can be defined formally by 4 components:

- The **initial state** of the agent is the state where the agent starts in. In this case, the initial state can be

described as In: Arad

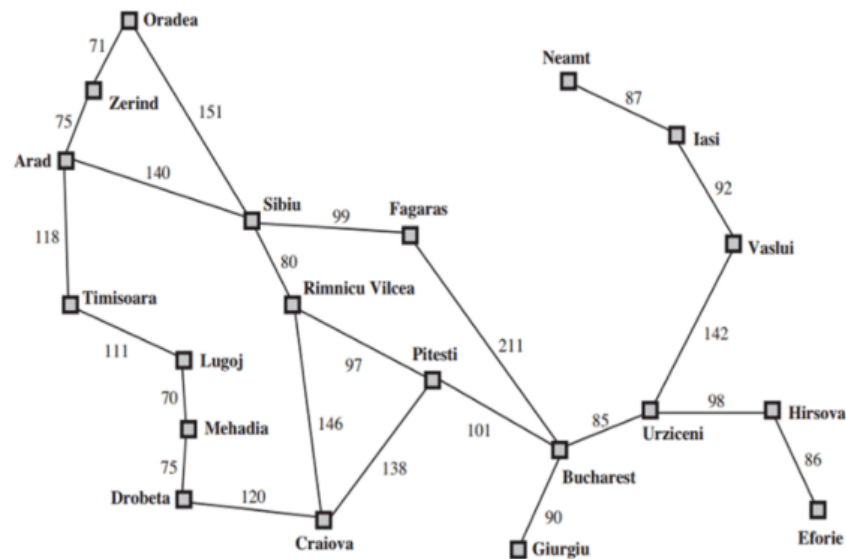
- The possible **actions** available to the agent, corresponding to each of the state the agent resides in.

For example, $ACTIONS(In: Arad) = \{Go: Sibiu, Go: Timisoara, Go: Zerind\}$.

Actions are also known as operations.

- A description of what each action does. the formal name for this is **Transition model**, Specified by the function $Result(s,a)$ that returns the state that results from the action a in state s .

We also use the term Successor to refer to any state reachable from a given state by a single action. For EX: $Result(In(Arad),GO(Zerind))=In(Zerind)$



Together the initial state, actions and transition model implicitly defines the **state space** of the problem. State space: set of all states reachable from the initial state by any sequence of actions

- **The goal test**, determining whether the current state is a goal state. Here, the goal state is $\{In: Bucharest\}$
- **The path cost function**, which determine the cost of each path, which is reflecting in the performance measure.

we define the cost function as $c(s, a, s')$, where s is the current state and a is the action performed by the agent to reach state s' .

```

function SIMPLE-PROBLEM-SOLVING-AGENT(p) returns an action
  inputs: p, a percept
  static: s, an action sequence, initially empty
           state, some description of the current world state
           g, a goal, initially null
           problem, a problem formulation

  state ← UPDATE-STATE(state, p)
  if s is empty then
    g ← FORMULATE-GOAL(state)
    problem ← FORMULATE-PROBLEM(state, g)
    s ← SEARCH(problem)
  action ← RECOMMENDATION(s, state)
  s ← REMAINDER(s, state)
  return action

```

Figure 3.1 A simple problem-solving agent.

Example –

8 puzzle problem

Initial State

2	8	3
1	6	4
7		5

Goal State

1	2	3
8		4
7	6	5

- **States:** a state description specifies the location of each of the eight tiles in one of the ninesquares. For efficiency, it is useful to include the location of the blank.
- **Actions:** blank moves left, right, up, or down.
- **Transition Model:** Given a state and action, this returns the resulting state. For example if we apply left to the start state the resulting state has the 5 and the blank switched.
- **Goal test:** state matches the goal configuration shown in fig.
- **Path cost:** each step costs 1, so the path cost is just the length of the path.

State Space Search/Problem Space Search:

The state space representation forms the basis of most of the AI methods.

- Formulate a problem as a **state space search** by showing the legal problem states, the legal operators, and the initial and goal states.
- A **state** is defined by the specification of the values of all attributes of interest in the world
- An **operator** changes one state into the other; it has a precondition which is the value of certain attributes prior to the application of the operator, and a set of effects, which are the attributes altered by the operator
- The **initial state** is where you start
- The **goal state** is the partial description of the solution

Formal Description of the problem:

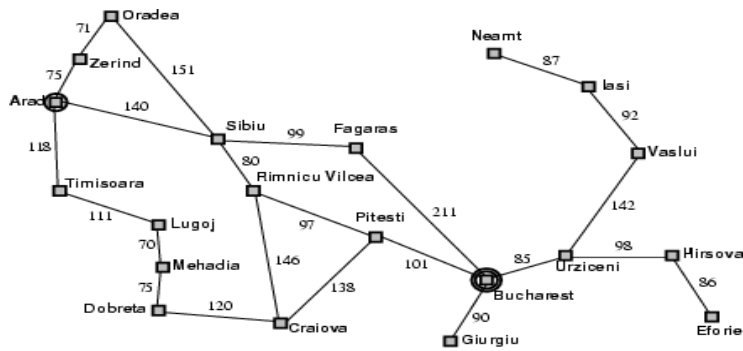
1. Define a state space that contains all the possible configurations of the relevant objects.
 2. Specify one or more states within that space that describe possible situations from which the problem solving process may start (**initial state**)
 3. Specify one or more states that would be acceptable as solutions to the problem. (**goal states**)
- Specify a set of rules that describe the actions (**operations**) available

State-Space Problem Formulation:

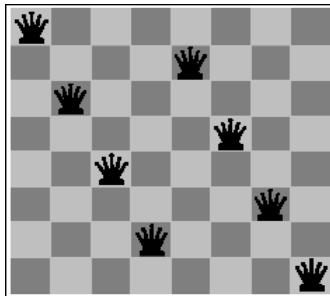
Example: A problem is defined by four items:

1. **initial state** e.g., "at Arad"
2. **actions or successor function** : $S(x)$ = set of action–state pairs e.g., $S(\text{Arad}) = \{ \langle \text{Arad} \rightarrow \text{Zerind}, \text{Zerind} \rangle, \dots \}$
3. **goal test (or set of goal states)**
e.g., $x = \text{"at Bucharest"}$, $\text{Checkmate}(x)$
4. **path cost (additive)**
e.g., sum of distances, number of actions executed, etc.
 $c(x, a, y)$ is the step cost, assumed to be ≥ 0

A solution is a sequence of actions leading from the initial state to a goal state



Example: 8-queens problem



1. **Initial State:** Any arrangement of 0 to 8 queens on board.
2. **Operators:** add a queen to any square.
3. **Goal Test:** 8 queens on board, none attacked.
4. **Path cost:** not applicable or Zero (because only the final state counts, search cost might be of interest).

Search strategies:

Search: Searching is a step by step procedure to solve a search-problem in a given search space. A search problem can have three main factors:

Search Space: Search space represents a set of possible solutions, which a system may have.

Start State: It is a state from where agent begins the search.

Goal test: It is a function which observe the current state and returns whether the goal state is achieved or not.

Properties of Search Algorithms

Which search algorithm one should use will generally depend on the problem domain. There are four important factors to consider:

1. **Completeness** – Is a solution guaranteed to be found if at least one solution exists?
2. **Optimality** – Is the solution found guaranteed to be the best (or lowest cost) solution if there exists more than one solution?
3. **Time Complexity** – The upper bound on the time required to find a solution, as a function of the complexity of the problem.
4. **Space Complexity** – The upper bound on the storage space (memory) required at any point during the search, as a function of the complexity of the problem.

State Spaces versus Search Trees:

- State Space
 - Set of valid states for a problem
 - Linked by operators
 - e.g., 20 valid states (cities) in the Romanian travel problem
- Search Tree
 - Root node = initial state
 - Child nodes = states that can be visited from parent
 - Note that the depth of the tree can be infinite
 - E.g., via repeated states
 - Partial search tree
 - Portion of tree that has been expanded so far
 - Fringe
 - Leaves of partial search tree, candidates

for expansion Search trees = data structure to search state-space

Searching

Many traditional search algorithms are used in AI applications. For complex problems, the traditional algorithms are unable to find the solution within some practical time and space limits. Consequently, many special techniques are developed; using heuristic functions. The algorithms that use heuristic

functions are called heuristic algorithms. Heuristic algorithms are not really intelligent; they appear to be intelligent because they achieve better performance.

Heuristic algorithms are more efficient because they take advantage of feedback from the data to direct the search path.

Uninformed search

Also called blind, exhaustive or brute-force search, uses no information about the problem to guide the search and therefore may not be very efficient.

Informed Search:

Also called heuristic or intelligent search, uses information about the problem to guide the search, usually guesses the distance to a goal state and therefore efficient, but the search may not be always possible.

Uninformed Search (Blind searches):

1. Breadth First Search:

- One simple search strategy is a breadth-first search. In this strategy, the root node is expanded first, then all the nodes generated by the root node are expanded next, and then their successors, and so on.
- In general, all the nodes at depth d in the search tree are expanded before the nodes at depth $d + 1$.

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier) /* chooses the shallowest node in frontier. */
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child ← CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
        if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
        frontier ← INSERT(child, frontier)
```

Figure 3.11 Breadth-first search on a graph.

BFS illustrated:

Step 1: Initially frontier contains only one node corresponding to the source state A.

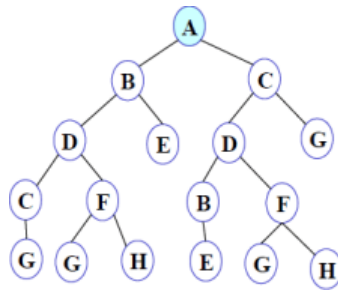


Figure 1

Frontier: A

Step 2: A is removed from fringe. The node is expanded, and its children B and C are generated. They are placed at the back of fringe.

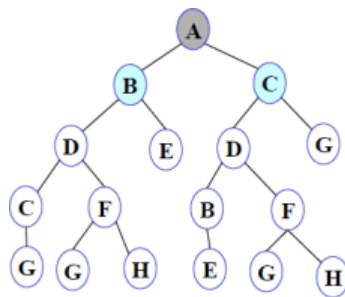
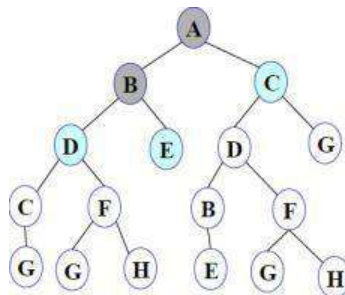


Figure 2

Frontier: B C

Step 3: Node B is removed from fringe and is expanded. Its children D, E are generated and



putat the back of fringe.

Figure 3

Frontier: C D E

Step 4: Node C is removed from fringe and is expanded. Its children D and G are added to theback of fringe.

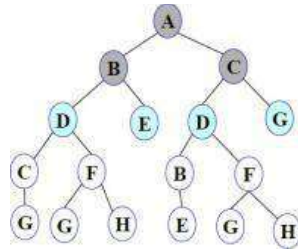


Figure 4

Frontier: D E D G

Step 5: Node D is removed from fringe. Its children C and F are generated and added to the back of fringe.

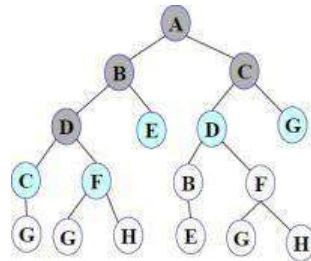


Figure 5

Frontier: E D G C F

Step 6: Node E is removed from fringe. It has no children.

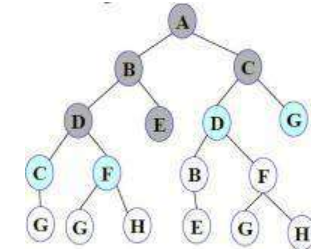


Figure 6

Frontier: D G C F

Step 7: D is expanded; B and F are put in OPEN.

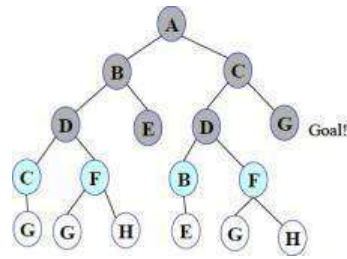


Figure 7

Frontier: G C F B F

Step 8: G is selected for expansion. **It is found to be a goal node.** So the algorithm returns the path A C G by following the parent pointers of the node corresponding to G. The algorithm terminates.

Breadth first search is:

- One of the simplest search strategies
- Complete. If there is a solution, BFS is guaranteed to find it.
- If there are multiple solutions, then a minimal solution will be found
- The algorithm is optimal (i.e., admissible) if all operators have the same cost. Otherwise, breadth first search finds a solution with the shortest path length.
- **Time complexity** : $O(b^d)$
- **Space complexity** : $O(b^d)$
- **Optimality** : Yes

b - branching factor (maximum no of successors of any node), d – Depth of the shallowest goal node

Advantages: ***Maximum length of any path (m) in search space***

- BFS will provide a solution if any solution exists.
- If there are more than one solutions for a given problem, then BFS will provide the minimal solution which requires the least number of steps.

Disadvantages:

- Requires the generation and storage of a tree whose size is exponential the depth of the shallowest goal node.
- The breadth first search algorithm cannot be effectively used unless the search space is quite small.

Applications Of Breadth-First Search Algorithm

GPS Navigation systems: Breadth-First Search is one of the best algorithms used to find neighboring locations by using the GPS system.

Broadcasting: Networking makes use of what we call as packets for communication. These packets follow a traversal method to reach various networking nodes. One of the most commonly used traversal

methods is Breadth-First Search. It is being used as an algorithm that is used to communicate broadcasted packets across all the nodes in a network.

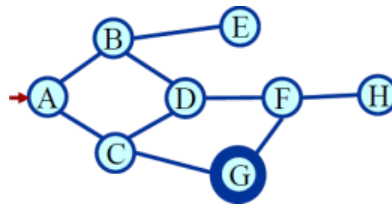
Depth- First- Search.

We may sometimes search the goal along the largest depth of the tree, and move up only when further traversal along the depth is not possible. We then attempt to find alternative offspring of the parent of the node (state) last visited. If we visit the nodes of a tree using the above principle to search the goal, the traversal made is called depth first traversal and consequently the search strategy is called *depth first search*.

function DEPTH-LIMITED-SEARCH(*problem, limit*) **returns** a solution, or failure/cutoff
return RECURSIVE-DLS(MAKE-NODE(*problem.INITIAL-STATE*), *problem, limit*)

function RECURSIVE-DLS(*node, problem, limit*) **returns** a solution, or failure/cutoff
if *problem.GOAL-TEST*(*node.STATE*) **then return** SOLUTION(*node*)
else if *limit* = 0 **then return** *cutoff*
else
 cutoff_occurred? ← false
 for each *action* **in** *problem.ACTIONS*(*node.STATE*) **do**
 child ← CHILD-NODE(*problem, node, action*)
 result ← RECURSIVE-DLS(*child, problem, limit - 1*)
 if *result* = *cutoff* **then** *cutoff_occurred?* ← true
 else if *result* ≠ *failure* **then return** *result*
 if *cutoff_occurred?* **then return** *cutoff* **else return** *failure*

DFS illustrated:



A State Space Graph

Step 1: Initially fringe contains only the node for A.

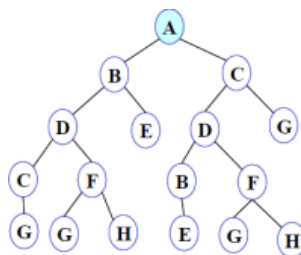


Figure 1

FRINGE: A

Step 2: A is removed from fringe. A is expanded and its children B and C are put in front of fringe.

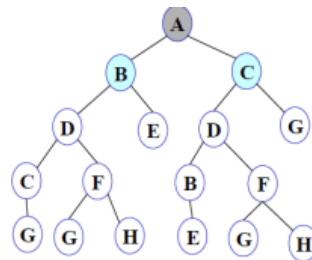


Figure 2

FRINGE: B C

Step 3: Node B is removed from fringe, and its children D and E are pushed in front of fringe.

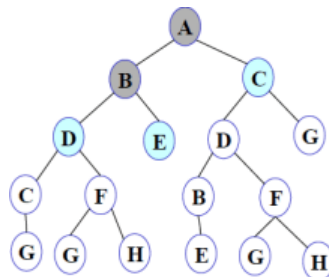


Figure 3

FRINGE: D E C

Step 4: Node D is removed from fringe. C and F are pushed in front of fringe.

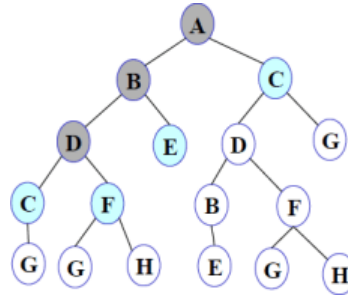


Figure 4

FRINGE: C F E C

Step 5: Node C is removed from fringe. Its child G is pushed in front of fringe.

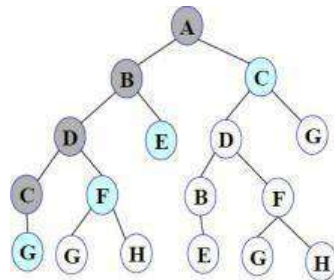


Figure 5

FRINGE: G F E C

Step 6: Node G is expanded and found to be a goal node.

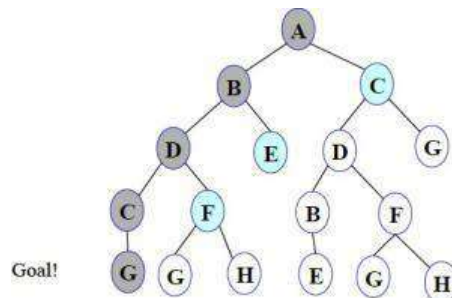


Figure 6

FRINGE: G F E C

The solution path A-B-D-C-G is returned and the algorithm terminates.

Depth first search

1. takes exponential time.
2. If N is the maximum depth of a node in the search space, in the worst case the algorithm will take time $O(b^d)$.
3. The space taken is linear in the depth of the search tree, $O(bN)$.

Note that the time taken by the algorithm is related to the maximum depth of the search tree. If the search tree has infinite depth, the algorithm may not terminate. This can happen if the search space is infinite. It can also happen if the search space contains cycles. The latter case can be handled by checking for cycles in the algorithm. Thus **Depth First Search is not complete**.

Iterative Deeping DFS

- The iterative deepening algorithm is a combination of DFS and BFS algorithms.
- This search algorithm finds out the best depth limit and does it by gradually increasing the limit until a goal is found.
- This algorithm performs depth-first search up to a certain "depth limit", and it keeps increasing the depth limit after each iteration until the goal node is found.

Advantages:

- It combines the benefits of BFS and DFS search algorithm in terms of fast search and memory efficiency.

Disadvantages:

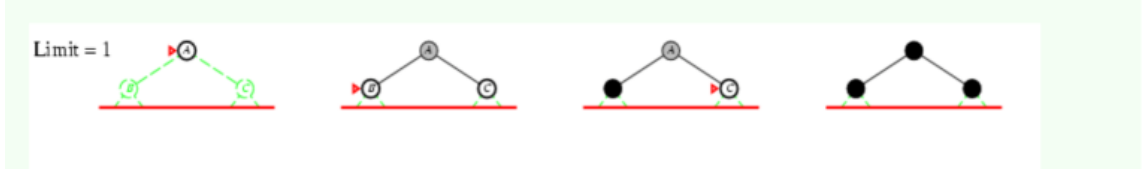
- The main drawback of IDDFS is that it repeats all the work of the previous phase.

Iterative deepening search $L=0$

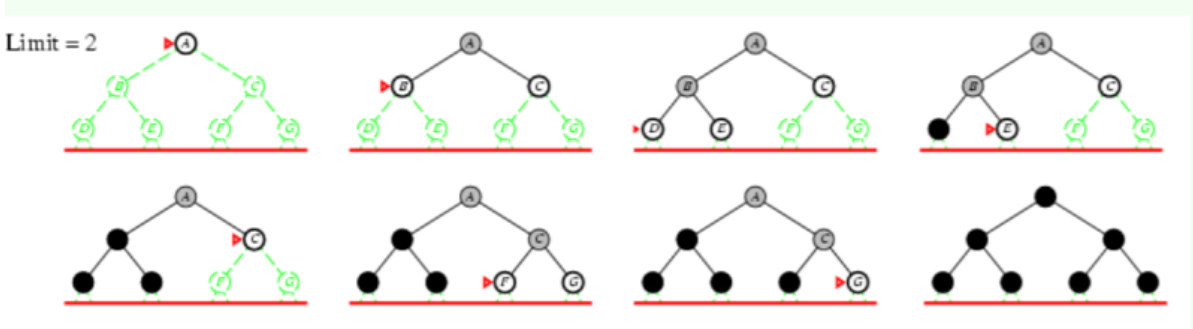
Limit = 0



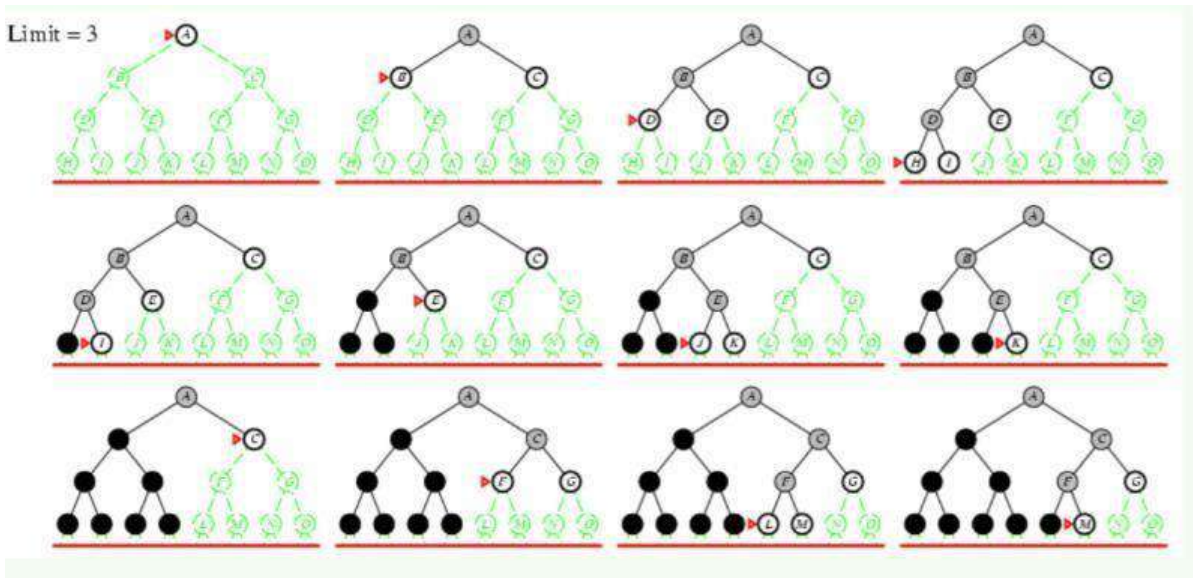
Iterative deepening search L=1



Iterative deepening search L=2



Iterative Deepening Search L=3



M is the goal node. So we stop there.

Complete: Yes

Time: $O(bd)$

Space: $O(bd)$

Optimal: Yes, if step cost = 1 or increasing function of depth.

Conclusion:

- We can conclude that IDS is a hybrid search strategy between BFS and DFS inheriting their advantages.
- IDS is faster than BFS and DFS.
- It is said that “IDS is the preferred uniformed search method when there is a large search space and the depth of the solution is not known”

A comparison table between DFS, BFS and IDDFS

	Time Complexity	Space Complexity	When to Use ?
DFS	$O(b^d)$	$O(d)$	=> Don't care if the answer is closest to the starting vertex/root. => When graph/tree is not very big/infinite.
BFS	$O(b^d)$	$O(b^d)$	=> When space is not an issue => When we do care/want the closest answer to the root.
IDDFS	$O(b^d)$	$O(bd)$	=> You want a BFS, you don't have enough memory, and somewhat slower performance is accepted. In short, you want a BFS + DFS.

Informed search/Heuristic search

A *heuristic* is a method that

- might not always find the best solution **but** is guaranteed to find a good solution in reasonable time. By sacrificing completeness it increases efficiency.
- Useful in solving tough problems which
 - could not be solved any other way.
 - solutions take an infinite time or very long time to compute.

Calculating Heuristic Value:

- 1. Euclidian distance- used to calculate straight line distance.
- 2. Manhattan distance- If we want to calculate vertical or horizontal

distance For ex: 8 puzzle problem

Source state

1	3	2
---	---	---

6	5	4
	8	7

destination state

1	2	3
4	5	6
7	8	

Then the Manhattan distance would be sum of the no of moves required to move each number from source state to destination state.

Number in 8 puzzle	1	2	3	4	5	6	7	8
No. of moves to reach destination	0	2	1	2	0	2	2	0

3. No. of misplaced tiles for 8 puzzle problem

Source state

1	3	2
6	5	4
	8	7

Destination state

1	2	3
4	5	6
7	8	

Here just calculate the number of tiles that have to be changed to reach goal state Here 1,5,8 need not be changed

2,3,4,6,7 should be changed, so the heuristic value will be 5(because 5 tiles have to be changed)

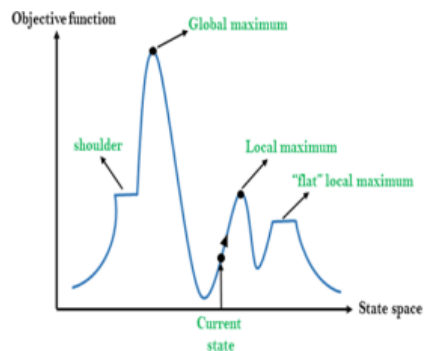
Hill Climbing Algorithm

- ✓ Hill climbing algorithm is a local search algorithm which continuously moves in the direction of increasing elevation/value to find the peak of the mountain or best solution to the problem. It terminates when it reaches a peak value where no neighbor has a higher value.

- ✓ It is also called greedy local search as it only looks to its good immediate neighbor state and not beyond that.
- ✓ Hill Climbing is mostly used when a good heuristic is available.
- ✓ In this algorithm, we don't need to maintain and handle the search tree or graph as it only keeps a single current state.

The idea behind hill climbing is as follows.

1. Pick a random point in the search space.
2. Consider all the neighbors of the current state.
3. Choose the neighbor with the best quality and move to that state.
4. Repeat 2 thru 4 until all the neighboring states are of lower quality.
5. Return the current state as the solution state.



Different regions in the state space landscape:

Local Maximum: Local maximum is a state which is better than its neighbor states, but there is also another state which is higher than it.

Global Maximum: Global maximum is the best possible state of state space landscape. It has the highest value of objective function.

Current state: It is a state in a landscape diagram where an agent is currently present.

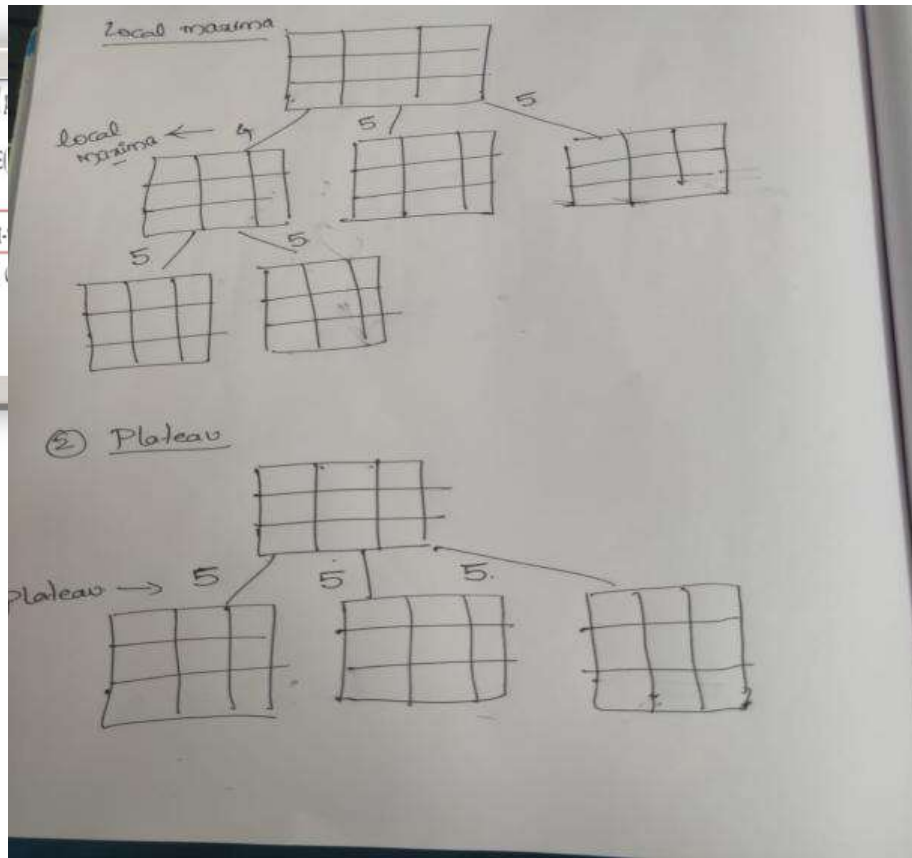
Flat local maximum: It is a flat space in the landscape where all the neighbor states of current states have the same value.

Shoulder: It is a plateau region which has an uphill edge.

Algorithm for Hill Climbing

```

function HILL-CLIMBING(p)
  current ← MAKE-NODE(p)
  loop do
    neighbor ← a highest-
    if neighbor.VALUE > current.VALUE
      current ← neighbor
  
```



Problems in Hill Climbing Algorithm:

: Hill-climbing

This simple policy has three well-known drawbacks:

1. **Local Maxima:** a local maximum as opposed to global maximum.
2. **Plateaus:** An area of the search space where evaluation function is flat, thus requiring random walk.
3. **Ridge:** Where there are steep slopes and the search direction is not towards the top but towards the side.

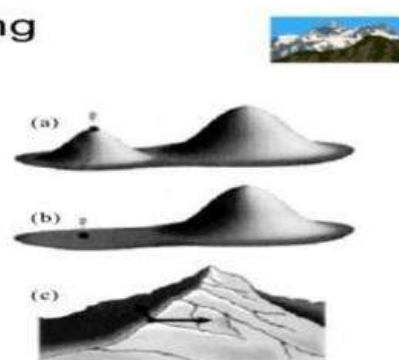


Figure 5.9 Local maxima, Plateaus and ridge situation for Hill Climbing

Simulated annealing search

A hill-climbing algorithm that never makes “downhill” moves towards states with lower value (or higher cost) is guaranteed to be incomplete, because it can get stuck on a local maximum. In contrast, a purely random walk – that is, moving to a successor chosen uniformly at random from the set of successors – is complete, but extremely inefficient. Simulated annealing is an algorithm that combines hill-climbing with a random walk in some way that yields both efficiency and completeness.

The simulated annealing algorithm is quite similar to hill climbing. Instead of picking the best move, however, it picks the random move. If the move improves the situation, it is always accepted. Otherwise, the algorithm accepts the move with some probability less than 1. The

probability decreases exponentially with the “badness” of the move – the amount ΔE by which the evaluation is worsened. The probability also decreases as the “temperature” T goes down: “bad” moves are more likely to be allowed at the start when temperature is high, and they become more unlikely as T decreases. One can prove that if the schedule lowers T slowly enough, the algorithm will find a global optimum with probability approaching 1.

Simulated annealing was first used extensively to solve VLSI layout problems. It has been applied widely to factory scheduling and other large-scale optimization tasks.

Best First Search:

- A combination of depth first and breadth first searches.
- Depth first is good because a solution can be found without computing all nodes and breadth first is good because it does not get trapped in dead ends.
- The best first search allows us to switch between paths thus gaining the benefit of both approaches. At each step the most promising node is chosen. If one of the nodes chosen generates nodes that are less promising it is possible to choose another at the same level and in effect the search changes from depth to breadth. If on analysis these are no better than this previously unexpanded node and branch is not forgotten and the search method reverts to the

OPEN is a priority queue of nodes that have been evaluated by the heuristic function but

which have not yet been expanded into successors. The most promising nodes are at the front.

CLOSED are nodes that have already been generated and these nodes must be stored because a graph is being used in preference to a tree.

Algorithm:

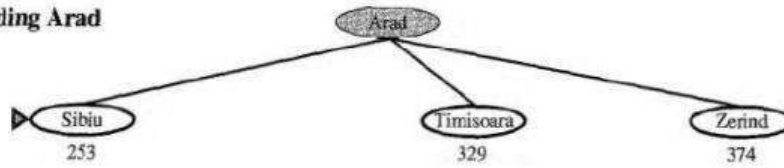
1. Start with OPEN holding the initial state
2. Until a goal is found or there are no nodes left on open do.
 - Pick the best node on OPEN
 - Generate its successors
 - For each successor Do
 - If it has not been generated before ,evaluate it ,add it to OPEN and record its parent
 - If it has been generated before change the parent if this new path is better and in that case update the cost of getting to any successor nodes.
3. If a goal is found or no more nodes left in OPEN, quit, else return to 2.

Example:

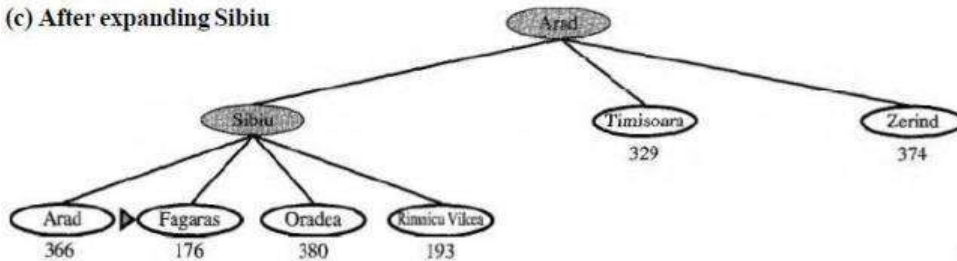
(a) The initial state



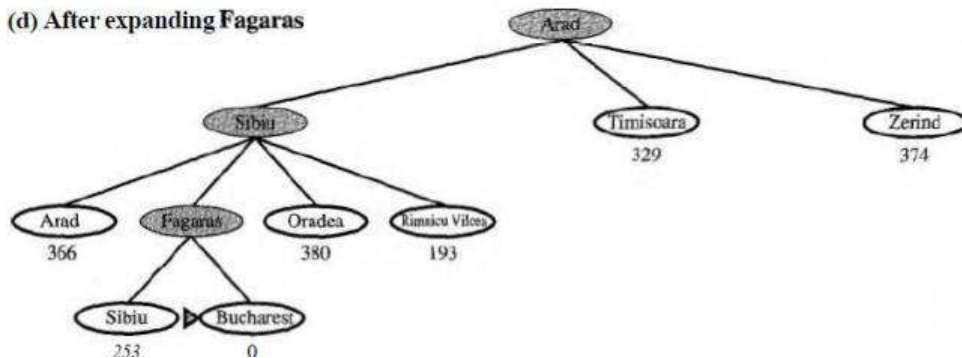
(b) After expanding Arad



(c) After expanding Sibiu



(d) After expanding Fagaras



1. It is not optimal.
2. It is incomplete because it can start down an infinite path and never return to try other possibilities.
3. The worst-case time complexity for greedy search is $O(b^m)$, where m is the maximum depth of the search space.
4. Because greedy search retains all nodes in memory, its space complexity is the same as its time complexity.

A* Algorithm

The Best First algorithm is a simplified form of the A* algorithm.

The **A* search algorithm** (pronounced "Ay-star") is a tree search algorithm that finds a path from a given initial node to a given goal node (or one passing a given goal test). It employs a "heuristic estimate" which ranks each node by an estimate of the best route that goes through

that node. It visits the nodes in order of this heuristic estimate.

Similar to greedy best-first search but is more accurate because A* takes into account the nodes that have already been traversed.

From A* we note that $f = g + h$ where

g is a measure of the distance/cost to go from the initial node to the current node

h is an estimate of the distance/cost to solution from the current node.

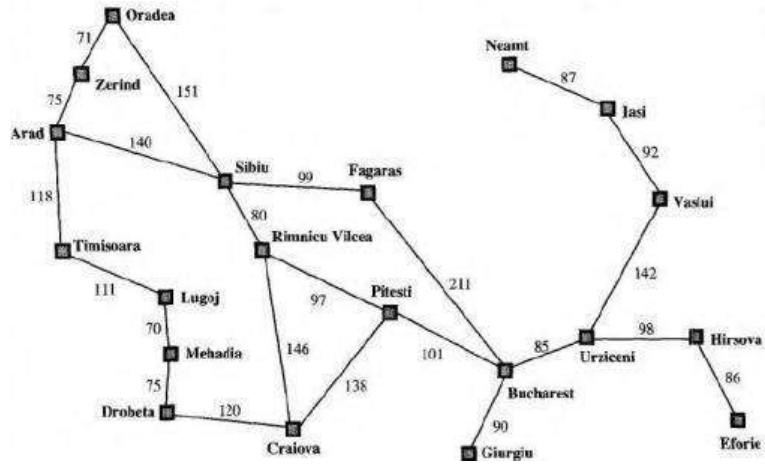
Thus f is an estimate of how long it takes to go from the initial node to the solution

Algorithm:

1. Initialize : Set OPEN = (S); CLOSED = ()
 $g(s) = 0, f(s) = h(s)$
2. Fail : If OPEN = (), Terminate and fail.
3. Select : select the minimum cost state, n, from OPEN,
save n in CLOSED
4. Terminate : If n ∈ G, Terminate with success and return f(n)
5. Expand : for each successor, m, of n
 - a) If $m \in [OPEN \cup CLOSED]$ Set $g(m) = g(n) + c(n, m)$ Set $f(m) = g(m) + h(m)$
Insert m in OPEN
 - b) If $m \in [OPEN \cup CLOSED]$
Set $g(m) = \min \{ g(m), g(n) + c(n, m) \}$ Set $f(m) = g(m) + h(m)$
If f(m) has decreased and $m \in CLOSED$
Move m to OPEN.

Description:

- A* begins at a selected node. Applied to this node is the "cost" of entering this node (usually zero for the initial node). A* then estimates the distance to the goal node from the current node. This estimate and the cost added together are the heuristic which is assigned to the path leading to this node. The node is then added to a priority queue, oftencalled "open".
- The algorithm then removes the next node from the priority queue (because of the way a priority queue works, the node removed will have the lowest heuristic). If the queue is empty, there is no path from the initial node to the goal node and the algorithm stops. If the node is the goal node, A* constructs and outputs the successful path and stops.
- If the node is not the goal node, new nodes are created for all admissible adjoining nodes;the exact way of doing this depends on the problem at hand. For each successive node, A* calculates the "cost" of entering the node and saves it with the node. This cost is calculated from the cumulative sum of costs stored with its ancestors, plus the cost of the operation which reached this new node.
- The algorithm also maintains a 'closed' list of nodes whose adjoining nodes have been checked. If a newly generated node is already in this list with an equal or lower cost, no further processing is done on that node or with the path associated with it. If a node in the closed list matches the new one, but has been stored with a *higher* cost, it is removed from the closed list, and processing continues on the new node.
- Next, an estimate of the new node's distance to the goal is added to the cost to form the heuristic for that node. This is then added to the 'open' priority queue, unless an identical node is found there.
- Once the above three steps have been repeated for each new adjoining node, the original node taken from the priority queue is added to the 'closed' list. The next node is then popped from the priority queue and the process is repeated



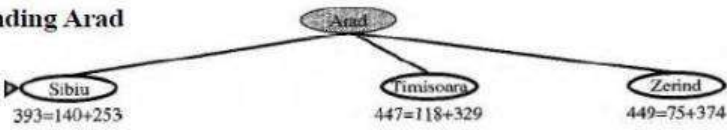
The heuristic costs from each city to Bucharest:

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

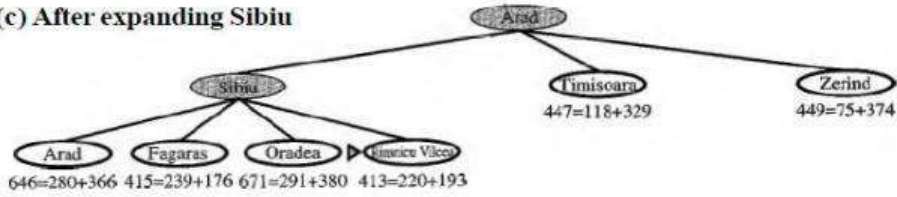
(a) The initial state

$$366=0+366$$

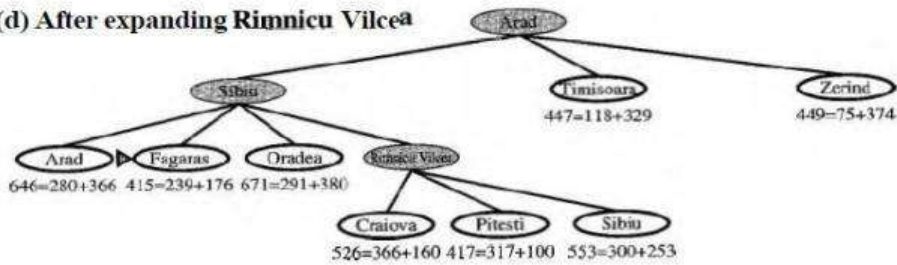
(b) After expanding Arad



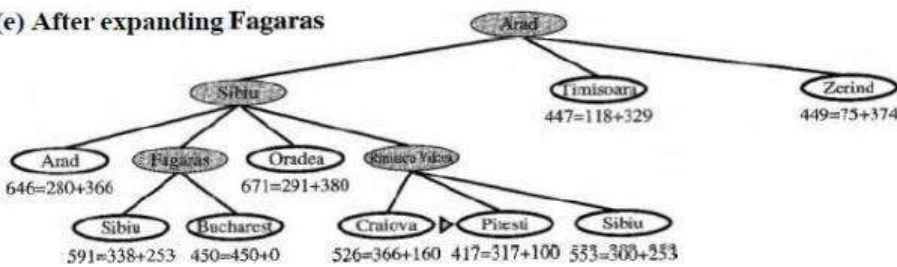
(c) After expanding Sibiu



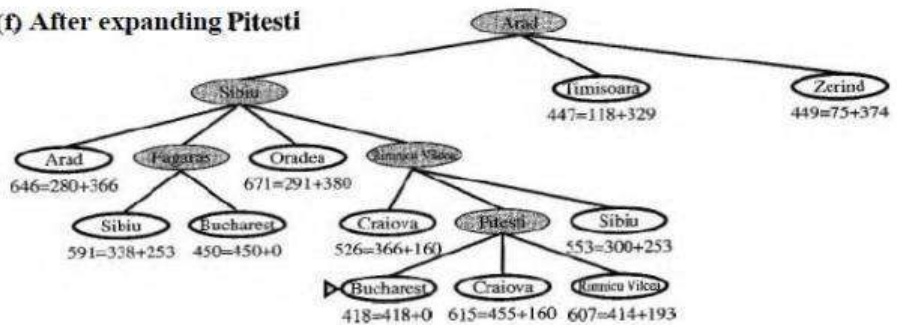
(d) After expanding Rimnicu Vilcea



(e) After expanding Fagaras



(f) After expanding Pitesti



A* search properties:

- The algorithm A* is admissible. This means that provided a solution exists, the first solution found by A* is an optimal solution. A* is admissible under the following conditions:
 - Heuristic function: for every node n , $h(n) \leq h^*(n)$.
 - A* is also complete.
- A* is optimally efficient for a given heuristic.
- A* is much more efficient than uninformed search.

Constraint Satisfaction Problems

<https://www.cnblogs.com/RDaneelOlivaw/p/8072603.html>

Sometimes a problem is not embedded in a long set of action sequences but requires picking the best option from available choices. A good general-purpose problem solving technique is to list the constraints of a situation (either negative constraints, like limitations, or positive elements that you want in the final solution). Then pick the choice that satisfies most of the constraints.

Formally speaking, a **constraint satisfaction problem (or CSP)** is defined by a set of variables, $X_1; X_2; \dots$

$; X_n$, and a set of constraints, $C_1; C_2; \dots; C_m$. Each variable X_i has a nonempty domain D_i of possible values. Each constraint C_i involves some subset of variables and specifies the allowable combinations of values for that subset. A state of the problem is defined by an assignment of values to some or all of the variables, $\{X_i = v_i; X_j = v_j; \dots\}$. An assignment that does not violate any constraints is called a consistent or

legal assignment. A complete assignment is one in which every variable is mentioned, and a solution to a CSP is a complete assignment that satisfies all the constraints. Some CSPs also require a solution that maximizes an objective function.

CSP can be given an **incremental formulation** as a standard search problem as follows:

1. **Initial state:** the empty assignment fg , in which all variables are unassigned.
2. **Successor function:** a value can be assigned to any unassigned variable, provided that it does not conflict with previously assigned variables.

3. **Goal test:** the current assignment is complete.
4. **Path cost:** a constant cost for every step

Examples:

1. The best-known category of continuous-domain CSPs is that of **linear programming** problems, where constraints must be linear inequalities forming a *convex* region.

$$\begin{array}{r}
 T \ W \ O \\
 + \ T \ W \ O \\
 \hline
 F \ O \ U \ R
 \end{array}$$

2. **Crypt arithmetic** puzzles.

Example: The map coloring problem.

The task of coloring each region red, green or blue in such a way that no neighboring regions have the same color.

We are given the task of coloring each region red, green, or blue in such a way that the neighboring regions must not have the same color.

To formulate this as CSP, we define the variable to be the regions: WA, NT, Q, NSW, V, SA, and T. The domain of each variable is the set {red, green, blue}. The constraints require neighboring regions to have distinct colors: for example, the allowable combinations for WA and NT are the pairs {(red,green),(red,blue),(green,red),(green,blue),(blue,red),(blue,green)}. (The constraint can also be represented as the inequality $WA \neq NT$). There are many possible solutions,

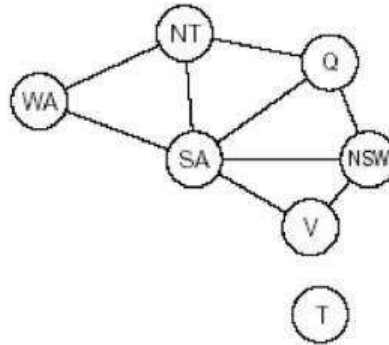


Variables WA, NT, Q, NSW, V, SA, T
 Domains $D_i = \{red, green, blue\}$
 Constraints: adjacent regions must have different colors
 e.g., $WA \neq NT$ (if the language allows this), or
 $(WA, NT) \in \{(red, green), (red, blue), (green, red), (green, blue), \dots\}$

such as {WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = red}. Map of Australia showing each of its states and territories

Constraint Graph: A CSP is usually represented as an undirected graph, called constraint graph where the nodes are the variables and the edges are the binary constraints.

Constraint graph: nodes are variables, arcs show constraints



The map-coloring problem represented as a constraint

graph. CSP can be viewed as a standard search

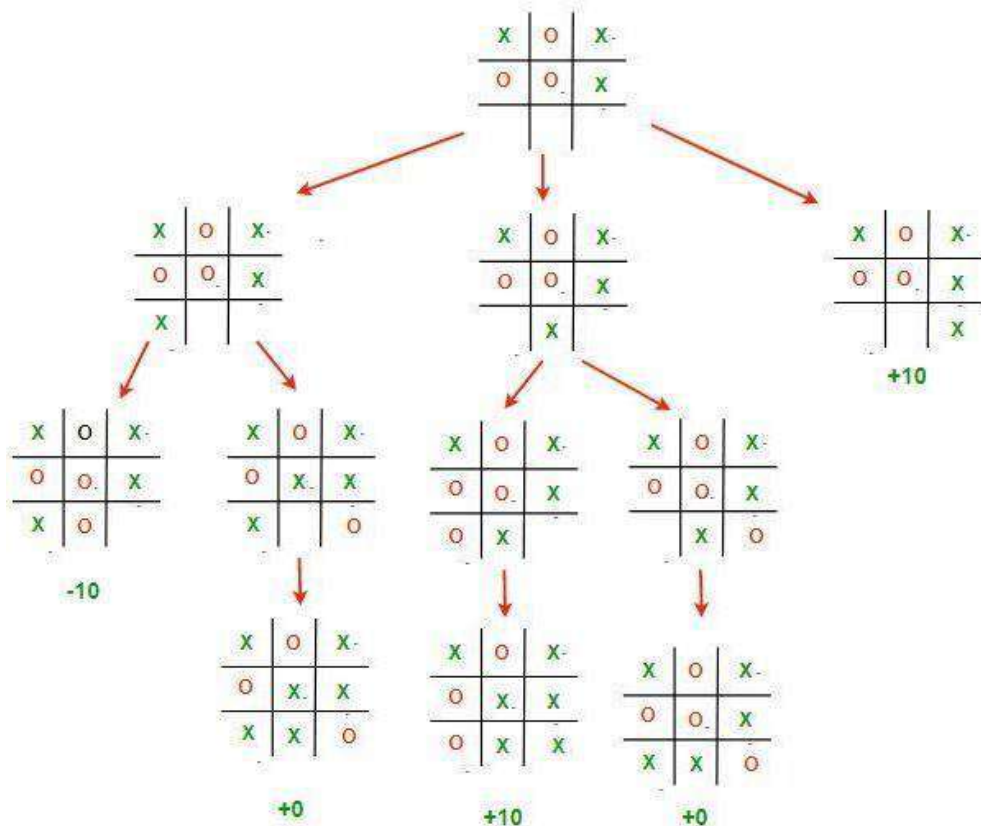
problem as follows:

- > **Initial state** : the empty assignment {}, in which all variables are unassigned.
- > **Successor function**: a value can be assigned to any unassigned variable, provided that it does not conflict with previously assigned variables.
- > **Goal test**: the current assignment is complete.
- > **Path cost**: a constant cost (E.g., 1) for every step.

UNIT II

Advanced Search: Constructing Search Trees, Stochastic Search, AO* Search Implementation, Minimax Search, Alpha-Beta Pruning Basic Knowledge Representation and Reasoning: Propositional Logic, First-Order Logic, Forward Chaining and Backward Chaining, Introduction to Probabilistic Reasoning, Bayes Theorem

Constructing Search Trees:



Game Playing

Adversarial search, or game-tree search, is a technique for analyzing an adversarial game in order to try to determine who can win the game and what moves the players should make in order to win. Adversarial search is one of the oldest topics in Artificial Intelligence. The original ideas for adversarial search were developed by Shannon in 1950 and independently by Turing in 1951, in the context of the game of chess—and their ideas still form the basis for the techniques used today.

2- Person Games:

- Players: We call them Max and Min.
- Initial State: Includes board position and whose turn it is.

- Operators: These correspond to legal moves.
- Terminal Test: A test applied to a board position which determines whether the game is over. In chess, for example, this would be a checkmate or stalemate situation.
- Utility Function: A function which assigns a numeric value to a terminal state. For example, in chess the outcome is win (+1), lose (-1) or draw (0). Note that by convention, we always measure utility relative to Max.

Mini Max Algorithm:

1. Generate the whole game tree.
2. Apply the utility function to leaf nodes to get their values.
3. Use the utility of nodes at level n to derive the utility of nodes at level $n-1$.
4. Continue backing up values towards the root (one layer at a time).
5. Eventually the backed up values reach the top of the tree, at which point Max chooses the move that yields the highest value. This is called the minimax decision because it maximises the utility for Max on the assumption that Min will play perfectly to minimise it.

Algorithm: MINIMAX (Depth-First Version)

To determine the minimax value $V(J)$, do the following:

1. If J is terminal, return $V(J) = e(J)$; otherwise
2. Generate J 's successors J_1, J_2, \dots, J_b .
3. Evaluate $V(J_1), V(J_2), \dots, V(J_b)$ from left to right.
4. If J is a MAX node, return $V(J) = \max[V(J_1), \dots, V(J_b)]$.
5. If J is a MIN node, return $V(J) = \min[V(J_1), \dots, V(J_b)]$.

```

function MINIMAX-DECISION(state) returns an action
  v ← MAX-VALUE(state)
  return the action in SUCCESSORS(state) with value v

function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← -∞
  for a, s in SUCCESSORS(state) do
    v ← MAX(v, MIN-VALUE(s))
  return v

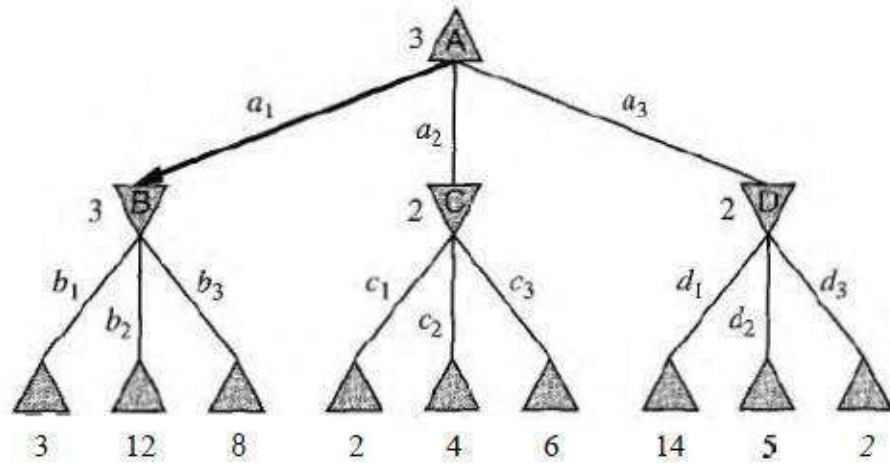
function MIN-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← ∞
  for a, s in SUCCESSORS(state) do
    v ← MIN(v, MAX-VALUE(s))
  return v

```

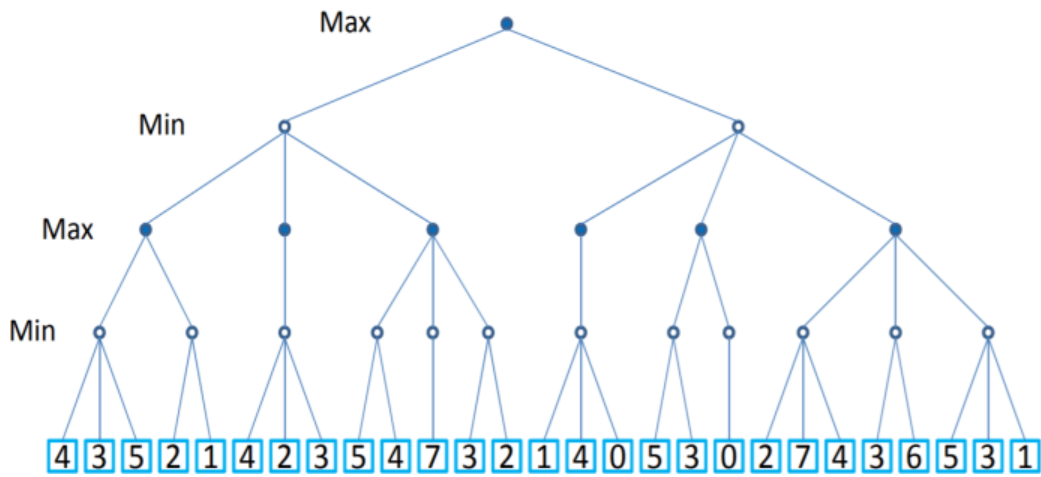
Example:

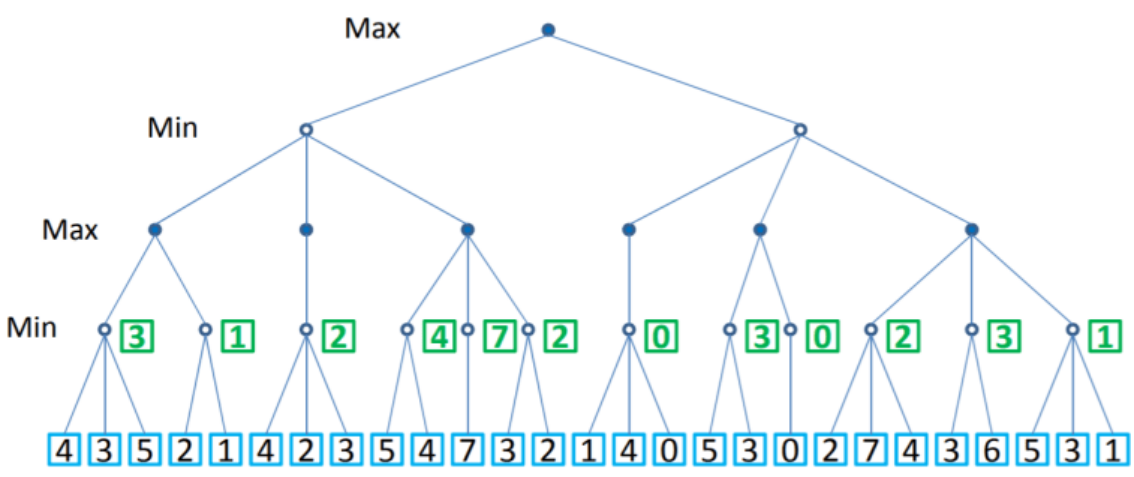
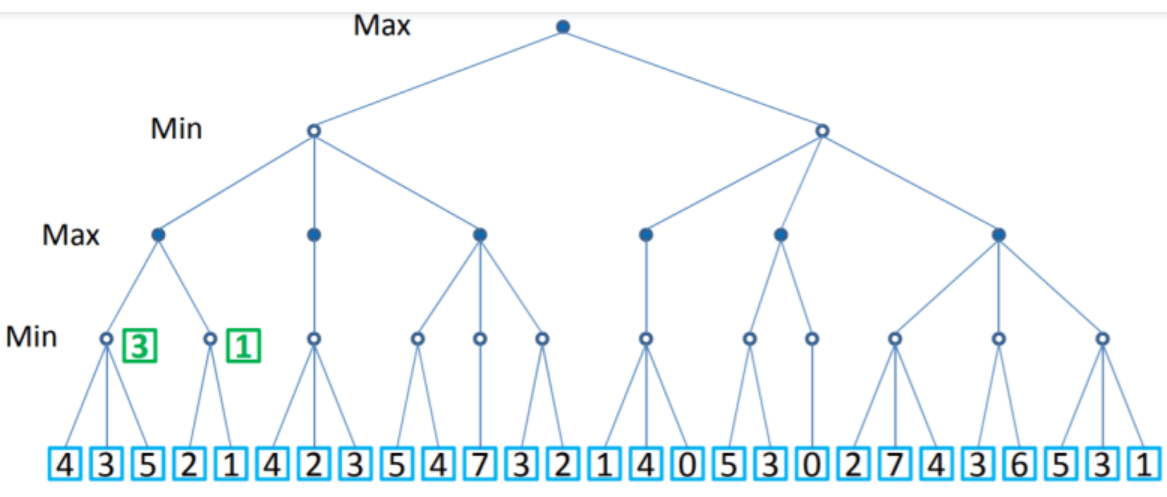
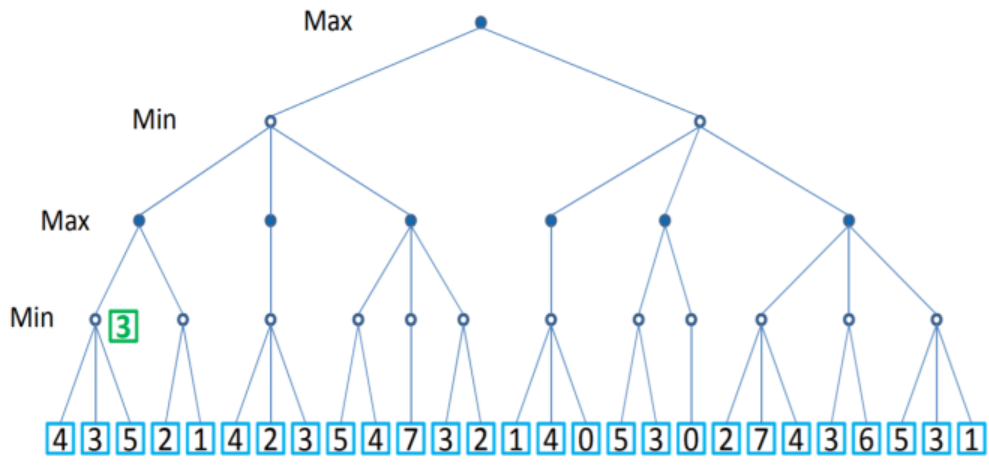
MAX

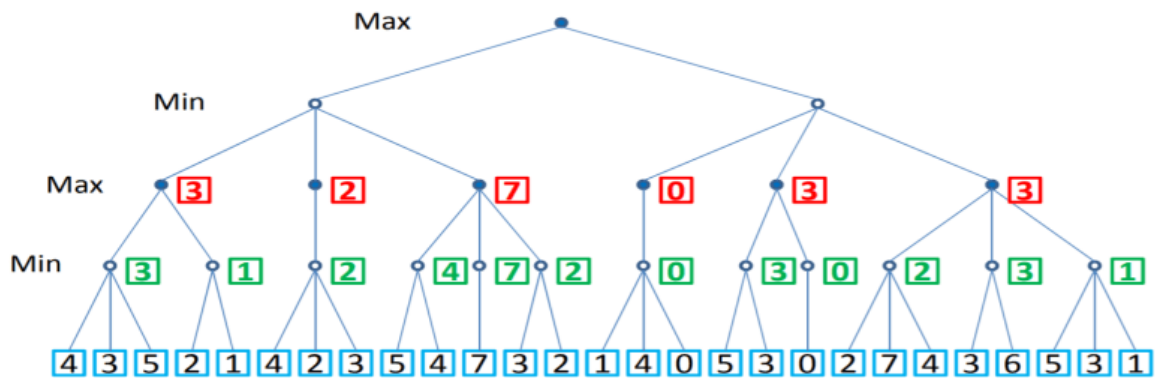
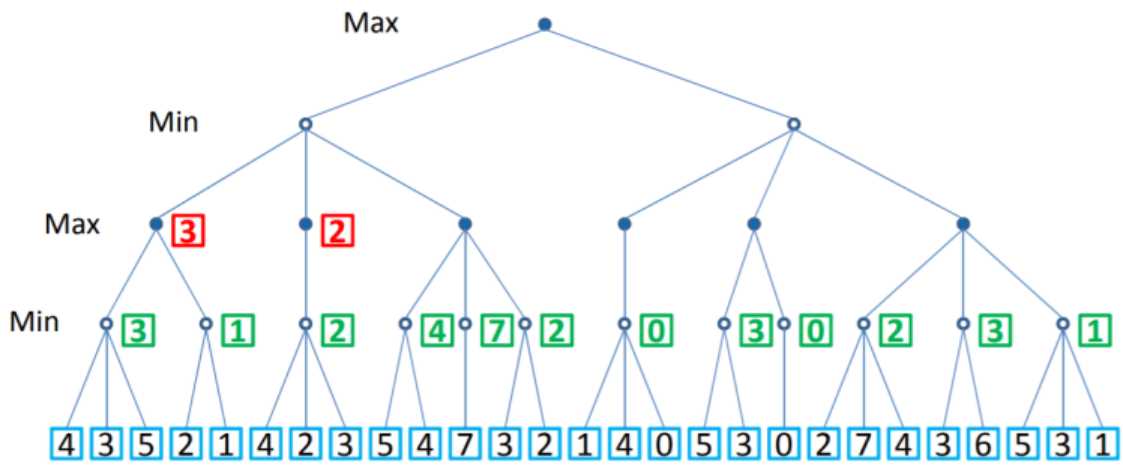
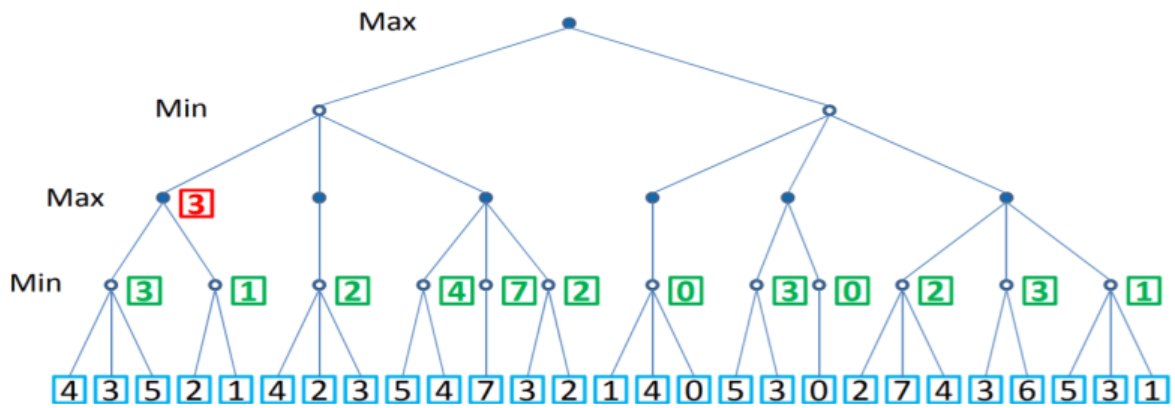
MIN

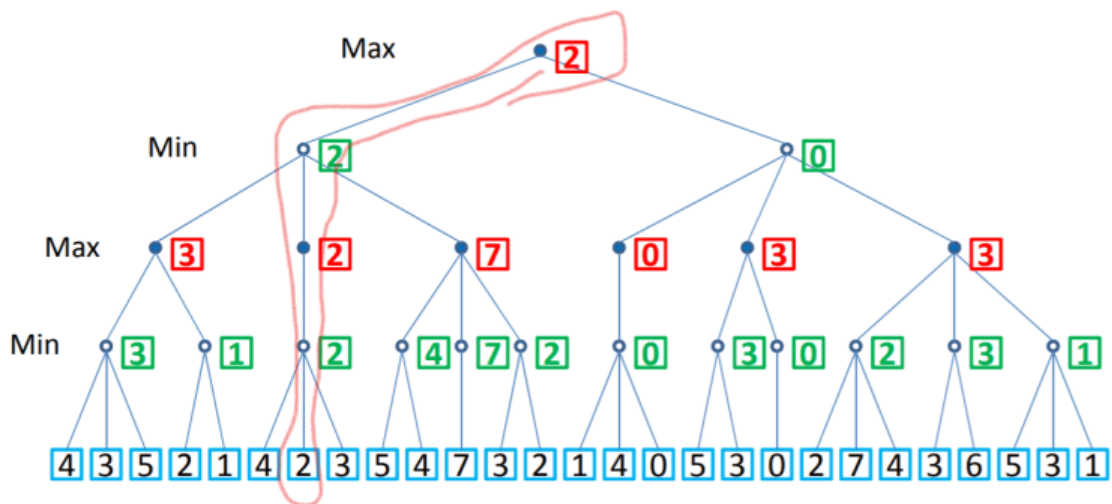
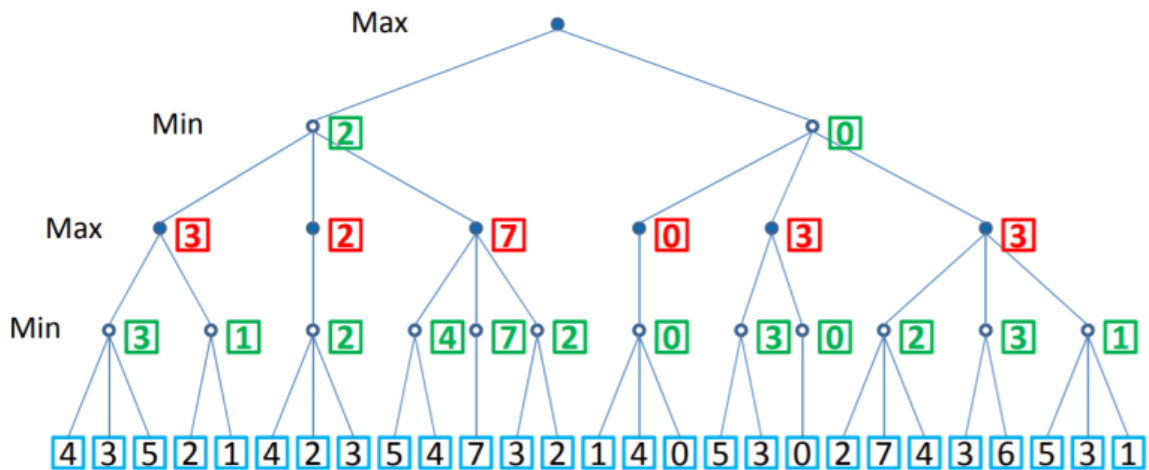


Example:









Properties of minimax:

- Complete : Yes (if tree is finite)
- Optimal : Yes (against an optimal opponent)
- Time complexity : $O(b^m)$
- Space complexity : $O(bm)$ (depth-first exploration)
- For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games
 → exact solution completely infeasible.

Limitations

- Not always feasible to traverse entire tree
- Time limitations

Alpha-Beta pruning algorithm:

- **Pruning:** eliminating a branch of the search tree from consideration without exhaustive examination of each node
- **Pruning:** the basic idea is to prune portions of the search tree that cannot improve the utility value of the max or min node, by just considering the values of nodes seen so far.
- *Alpha-beta pruning* is used on top of minimax search to detect paths that do not need to be explored. The intuition is:
 - The MAX player is always trying to maximize the score. Call this α .
 - The MIN player is always trying to minimize the score. Call this β .
- **Alpha cutoff:** Given a Max node n , cutoff the search below n (i.e., don't generate or examine any more of n 's children) if $\alpha(n) \geq \beta(n)$
(α increases and passes β from below)
- **Beta cutoff:** Given a Min node n , cutoff the search below n (i.e., don't generate or examine any more of n 's children) if $\beta(n) \leq \alpha(n)$
(β decreases and passes α from above)
- Carry α and β values down during search. Pruning occurs whenever $\alpha \geq \beta$

Algorithm:

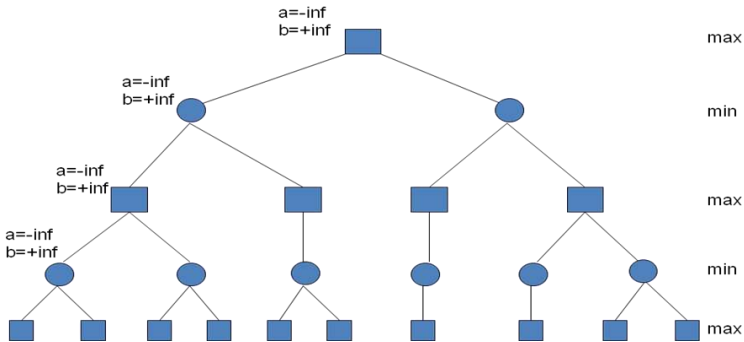
```
function ALPHA-BETA-SEARCH(state) returns an action
  inputs: state, current state in game
   $v \leftarrow \text{MAX-VALUE}(\textit{state}, -\infty, +\infty)$ 
  return the action in SUCCESSORS(state) with value  $v$ 

function MAX-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
  inputs: state, current state in game
          $\alpha$ , the value of the best alternative for MAX along the path to state
          $\beta$ , the value of the best alternative for MIN along the path to state
  if TERMINAL-TEST(state) then return UTILITY(state)
   $v \leftarrow -\infty$ 
  for  $a, s$  in SUCCESSORS(state) do
     $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s, a, \beta))$ 
    if  $v \geq \beta$  then return  $v$ 
   $\alpha \leftarrow \text{MAX}(\alpha, v)$ 
  return  $v$ 

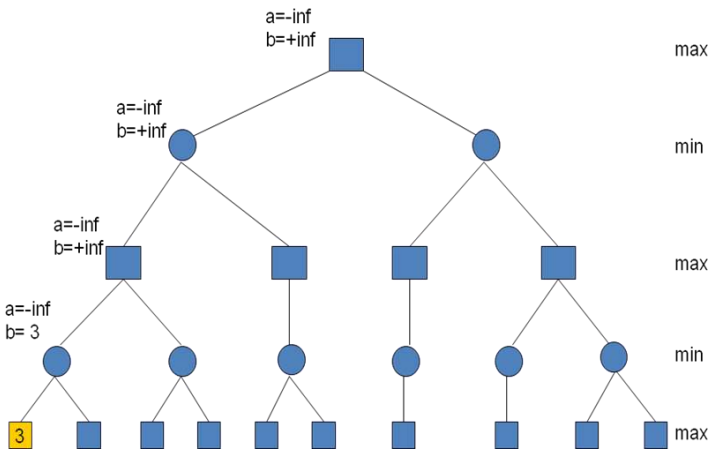
function MIN-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
  inputs: state, current state in game
          $\alpha$ , the value of the best alternative for MAX along the path to state
          $\beta$ , the value of the best alternative for MIN along the path to state
  if TERMINAL-TEST(state) then return UTILITY(state)
   $v \leftarrow +\infty$ 
  for  $a, s$  in SUCCESSORS(state) do
     $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(s, a, \beta))$ 
    if  $v \leq \alpha$  then return  $v$ 
   $\beta \leftarrow \text{MIN}(\beta, v)$ 
  return  $v$ 
```

Example:

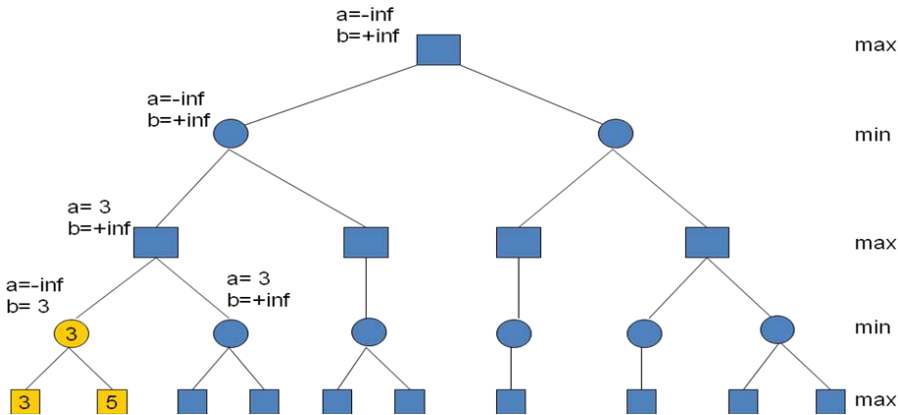
- 1) Setup phase: Assign to each left-most (or right-most) internal node of the tree, variables: $\alpha = -\text{infinity}$, $\beta = +\text{infinity}$



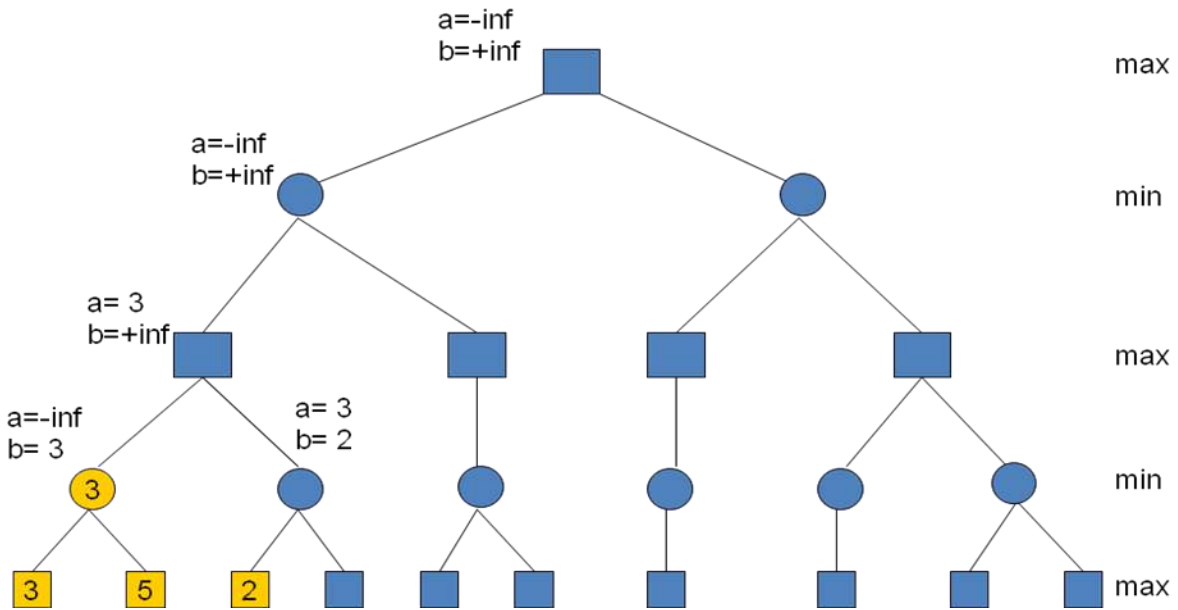
- 2) Look at first computed final configuration value. It's a 3. Parent is a min node, so set the beta (min) value to 3.



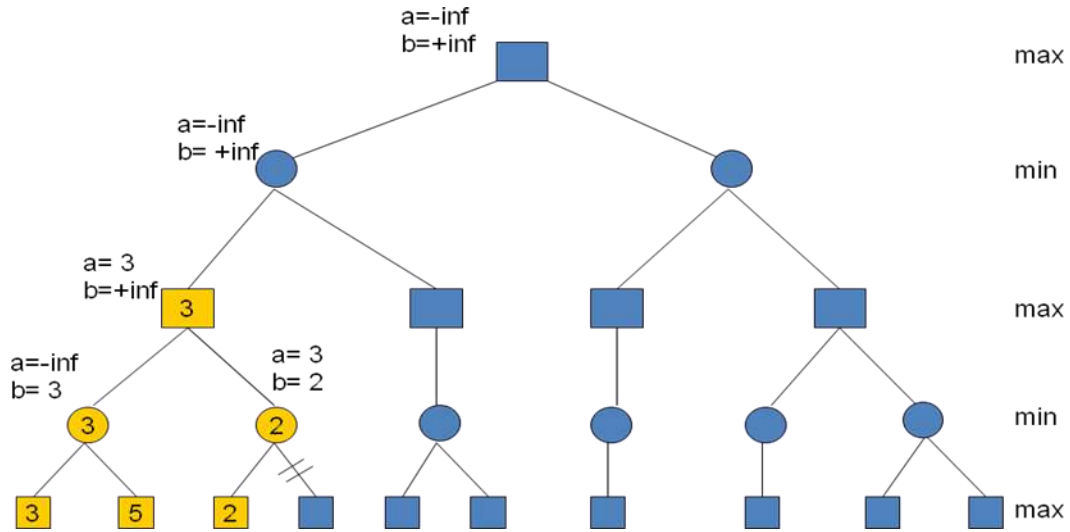
3) Look at next value, 5. Since parent is a min node, we want the minimum of 3 and 5 which is 3. Parent min node is done – fill alpha (max) value of its parent max node. Always set alpha for max nodes and beta for min nodes. Copy the state of the max parent node into the second unevaluated min child.



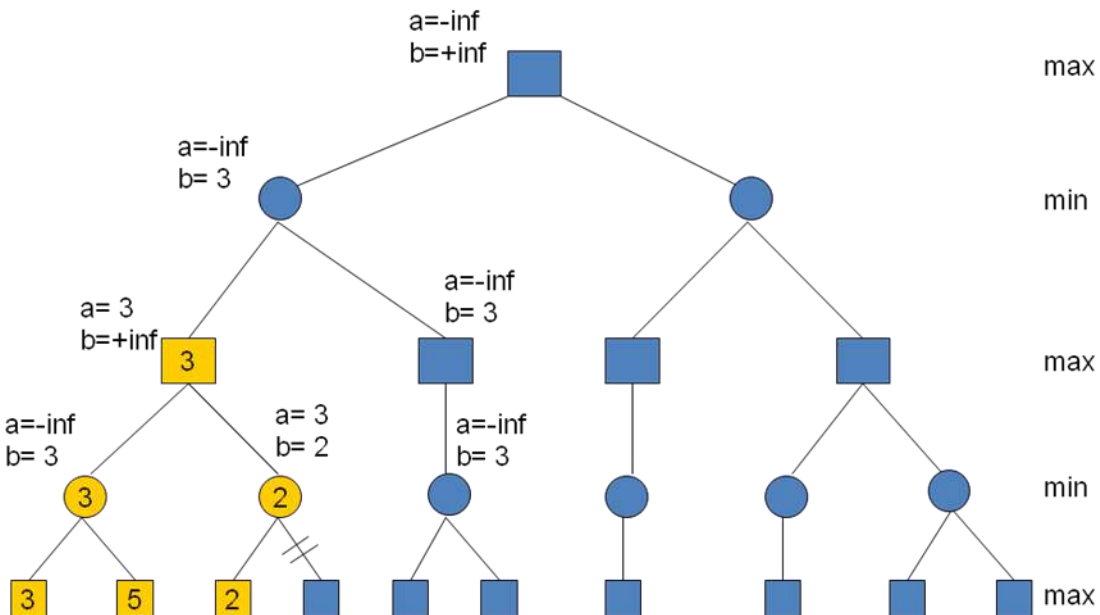
4) Look at next value, 2. Since parent node is min with $b = +\text{inf}$, 2 is smaller, change b .



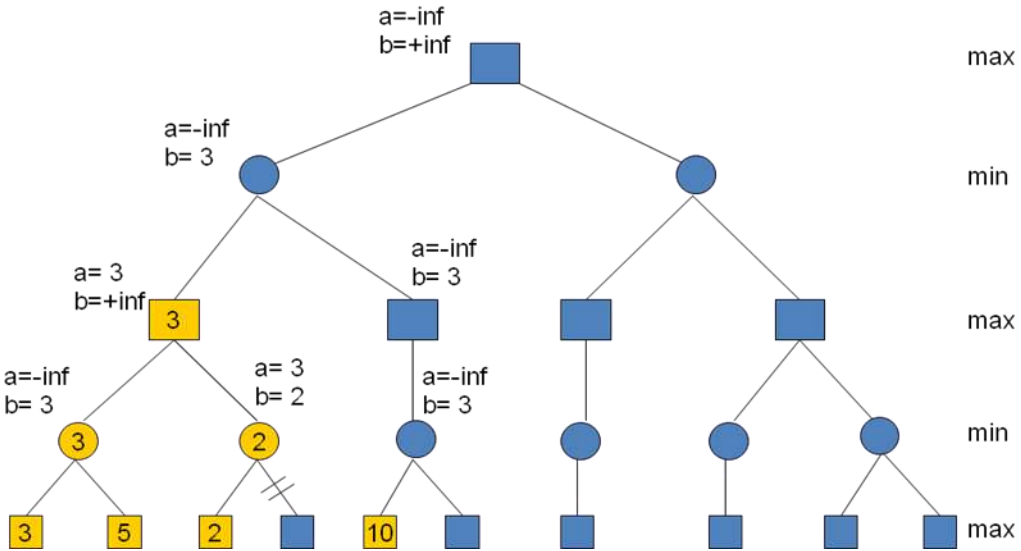
5) Now, the min parent node has a max value of 3 and min value of 2. The value of the 2nd child does not matter. If it is >2 , 2 will be selected for min node. If it is <2 , it will be selected for min node, but since it is <3 it will not get selected for the parent max node. Thus, we prune the right subtree of the min node. Propagate max value up the tree.



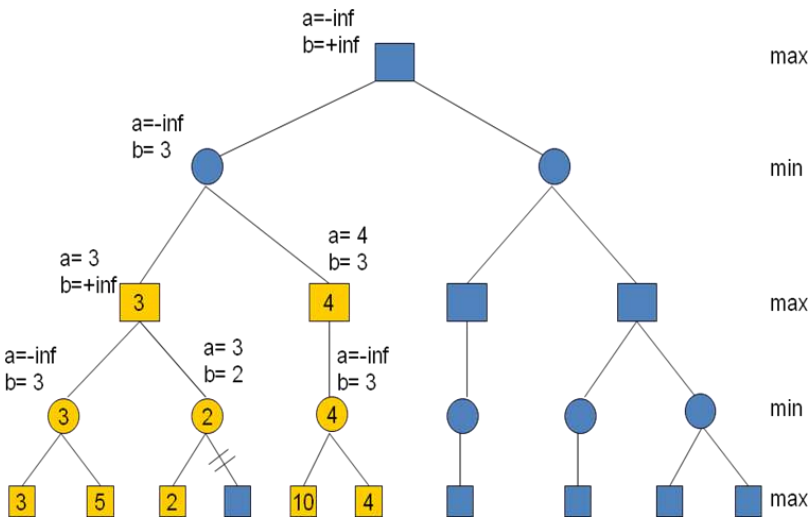
6) Max node is now done and we can set the beta value of its parent and propagate node state to sibling subtree's left-most path.



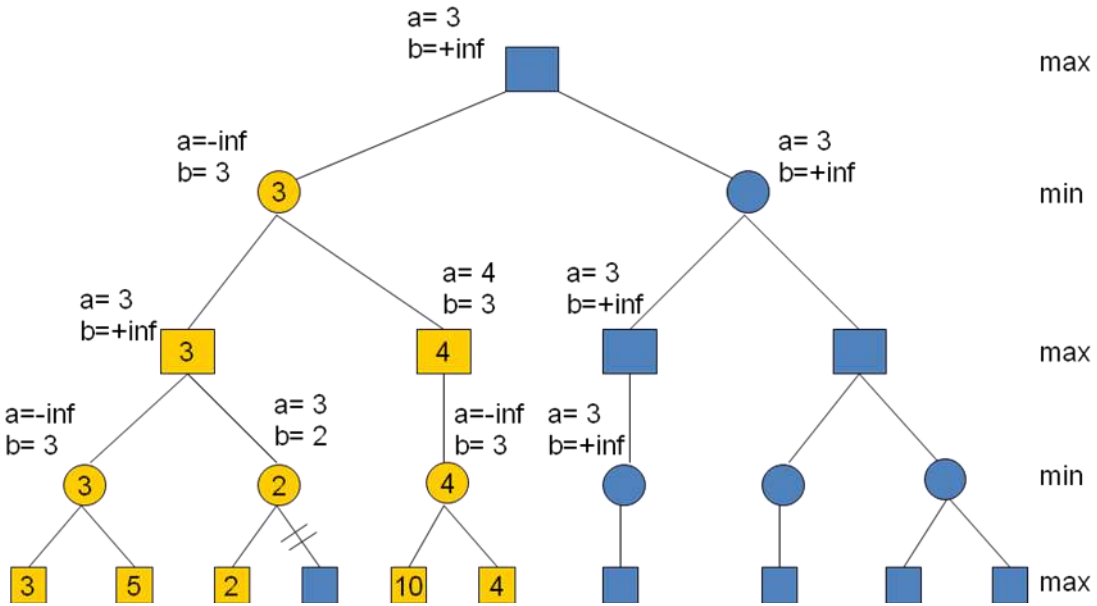
7) The next node is 10. 10 is not smaller than 3, so state of parent does not change. We still have to look at the 2nd child since alpha is still $-\text{inf}$.



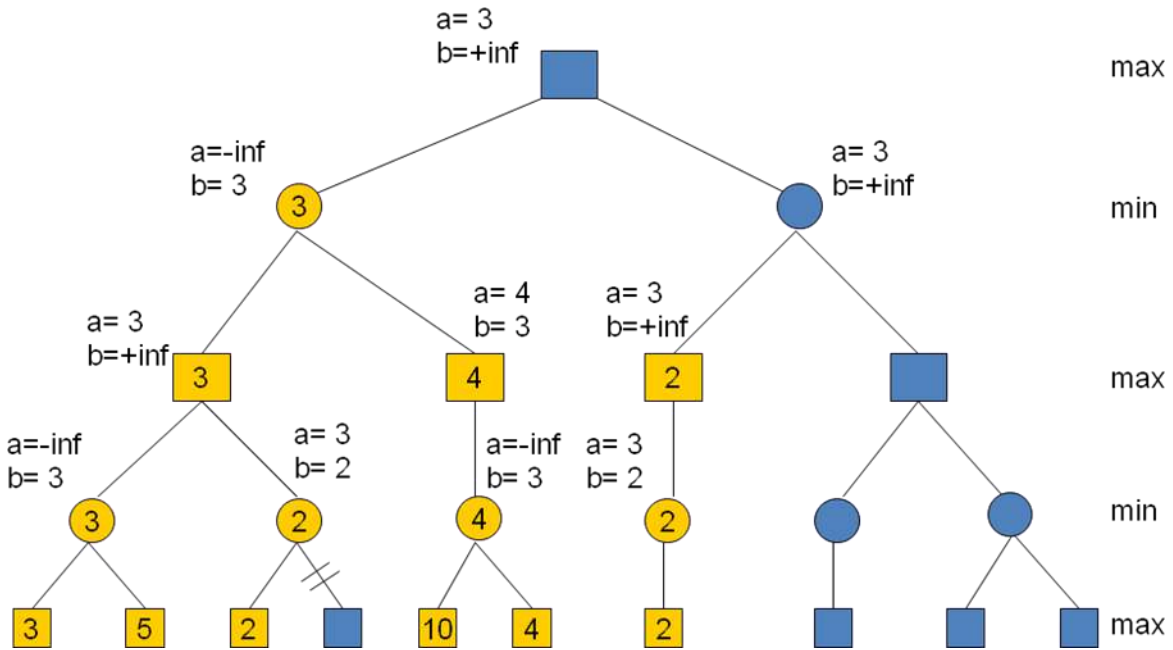
8) The next node is 4. Smallest value goes to the parent min node. Min subtree is done, so the parent max node gets the alpha (max) value from the child. Note that if the max node had a 2nd subtree, we can prune it since $a > b$.



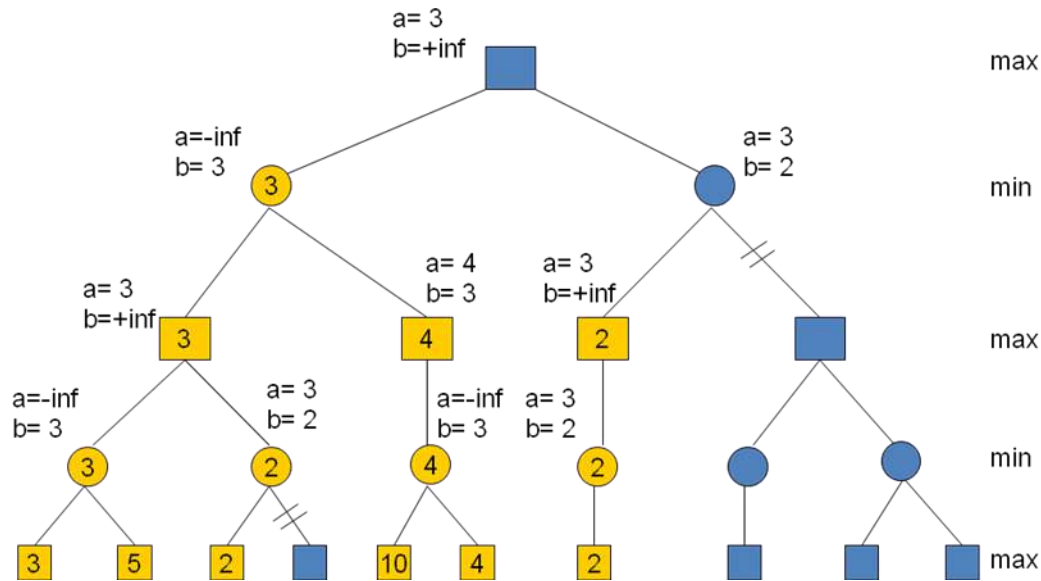
9) Continue propagating value up the tree, modifying the corresponding alpha/beta values. Also propagate the state of root node down the left-most path of the right subtree.



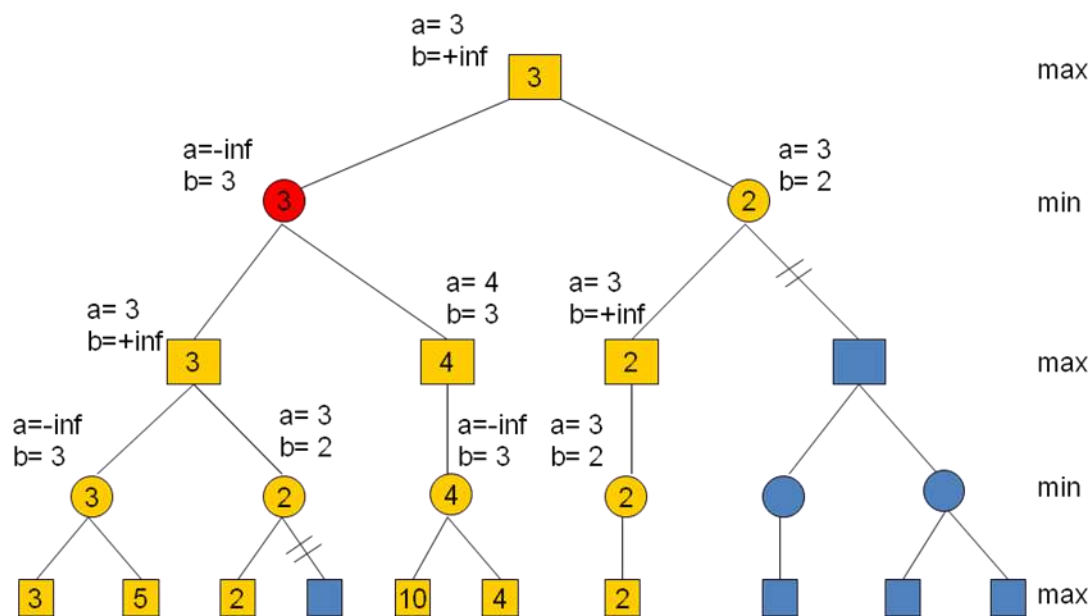
10) Next value is a 2. We set the beta (min) value of the min parent to 2. Since no other children exist, we propagate the value up the tree.



11) We have a value for the 3rd level max node, now we can modify the beta (min) value of the min parent to 2. Now, we have a situation that $a > b$ and thus the value of the rightmost subtree of the min node does not matter, so we prune the whole subtree.



12) Finally, no more nodes remain, we propagate values up the tree. The root has a value of 3 that comes from the left-most child. Thus, the player should choose the left-most child's move in order to maximize his/her winnings. As you can see, the result is the same as with the mini-max example, but we did not visit all nodes of the tree.



AO* Search: (And-Or) Graph:

The Depth first search and Breadth first search given earlier for OR trees or graphs can be easily adopted by AND-OR graph. The main difference lies in the way termination conditions are determined, since all goals following an AND nodes must be realized; where as a single goal node following an OR node will do. So for this purpose we are using AO* algorithm.

Like A* algorithm here we will use two arrays and one heuristic function.

OPEN:

It contains the nodes that has been traversed but yet not been marked solvable or unsolvable.

CLOSE:

It contains the nodes that have already been processed.
6 7: The distance from current node to goal node.

Algorithm:

Step 1: Place the starting node into OPEN.

Step 2: Compute the most promising solution tree say T0.

Step 3: Select a node n that is both on OPEN and a member of T0. Remove it from OPEN and place it in

CLOSE

Step 4: If n is the terminal goal node then levelled n as solved and levelled all the ancestors of n as solved. If the starting node is marked as solved then success and exit.

Step 5: If n is not a solvable node, then mark n as unsolvable. If starting node is marked as unsolvable, then return failure and exit.

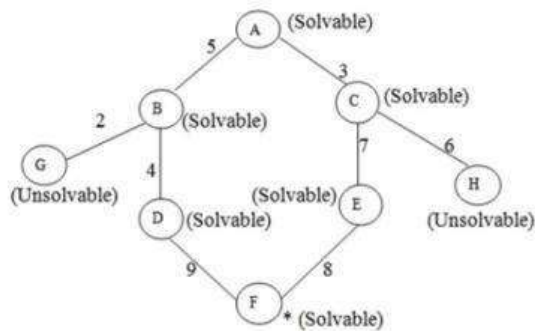
Step 6: Expand n. Find all its successors and find their h (n) value, push them into OPEN.

Step 7: Return to Step 2.

Step 8: Exit.

Implementation:

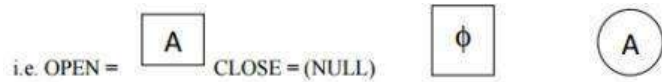
Let us take the following example to implement the AO* algorithm.



Figure

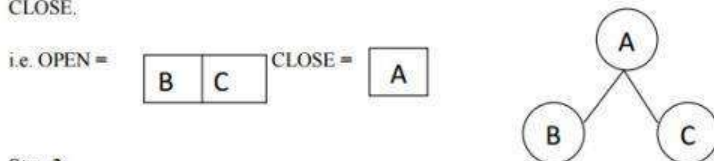
Step 1:

In the above graph, the solvable nodes are A, B, C, D, E, F and the unsolvable nodes are G, H. Take A as the starting node. So place A into OPEN.



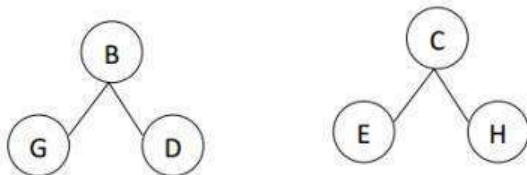
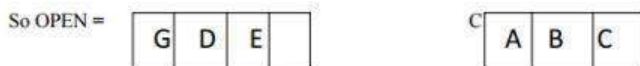
Step 2:

The children of A are B and C which are solvable. So place them into OPEN and place A into the CLOSE.



Step 3:

Now process the nodes B and C. The children of B and C are to be placed into OPEN. Also remove B and C from OPEN and place them into CLOSE.



(O)

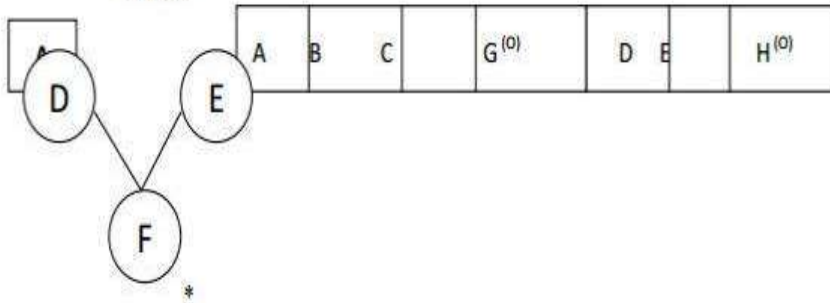
'O' indicated that the nodes G and H are unsolvable.

Step 4:

As the nodes G and H are unsolvable, so place them into CLOSE directly and process the nodes D and E.

i.e. OPEN =

CLOSE =



Step 5:

Now we have been reached at our goal state. So place F into CLOSE.

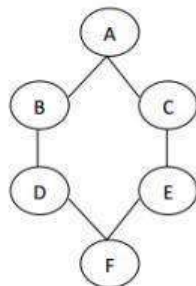


i.e. CLOSE =

Step 6:

Success and Exit

AO* Graph:



Figure

Advantages:

It is an optimal algorithm.

If traverse according to the ordering of nodes. It can be used for both OR and AND graph.

Disadvantages:

Sometimes for unsolvable nodes, it can't find the optimal path. Its complexity is than other algorithms.

BASIC KNOWLEDGE REPRESENTATION AND REASONING:

- Humans are best at understanding, reasoning, and interpreting knowledge. Human knows things, which is knowledge and as per their knowledge they perform various actions in the real world.
- But how machines do all these things comes under knowledge representation
 - **There are three factors which are put into the machine, which makes it valuable:**
 - **Knowledge:** The information related to the environment is stored in the machine.
 - **Reasoning:** The ability of the machine to understand the stored knowledge.
 - **Intelligence:** The ability of the machine to make decisions on the basis of the stored information.
 - A knowledge representation language is defined by two aspects:
 - The syntax of a language describes the possible configurations that can constitute sentences.
 - The semantics determines the facts in the world to which the sentences refer.
 - For example, the syntax of the language of arithmetic expressions says that if x and y are expressions denoting numbers, then $x > y$ is a sentence about numbers. The semantics of the languagesays that $x > y$ is false when y is a bigger number than x , and true otherwise From the syntax and semantics, we can derive an inference mechanism for an agent that uses the language.
 - Recall that the semantics of the language determine the fact to which a given sentence refers. Facts are part of the world,
- whereas their representations must be encoded in some way that can be physically stored within an agent. We cannot put the world inside a computer (nor can we put it inside a human), so all reasoning mechanisms must operate on representations of facts, rather than on the facts themselves. Because sentences are physical configurations of

parts of the agent,

Reasoning must be a process of constructing new physical configurations from old ones.

Proper reasoning should ensure that the new configurations represent facts that actually follow from the facts that the old configurations represent.

- We want to generate new sentences that are necessarily true, given that the old sentences are true. This relation between sentences is called entailment.

- In mathematical notation, the relation of entailment between a knowledge base KB and

a sentence α is pronounced "KB entails α " and written as

- An inference procedure can do one of two things:
- given a knowledge base KB, it can generate new sentences α that purport to be entailed by KB.
- E.g., $x + y = 4$ entails $4 = x + y$
- Entailment is a relationship between sentences (i.e., syntax) that is based on semantics

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

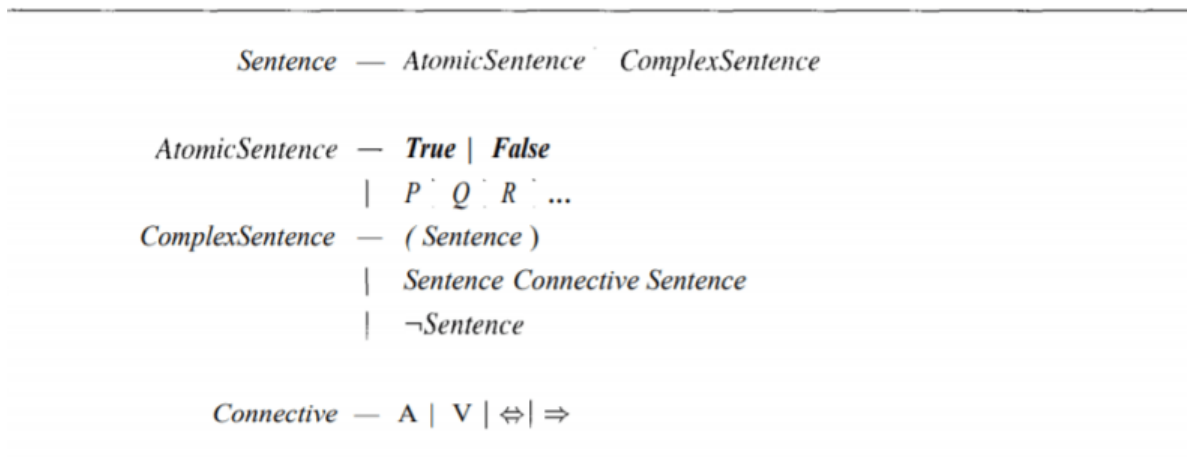


Figure 6.8 A BNF (Backus-Naur Form) grammar of sentences in propositional logic.

Syntax of propositional logic:

- The symbols of propositional logic are the logical constants True and False,

proposition symbols such as P and Q, the logical connectives \wedge , \vee , \Leftrightarrow , \Rightarrow and parentheses,

- All sentences are made by putting these symbols together using the following rules:
- The logical constants True and False are sentences by themselves.
- A propositional symbol such as P or Q is a sentence by itself.
- Wrapping parentheses around a sentence yields a sentence, for example, $(P \wedge Q)$. A sentence can be formed by combining simpler sentences with one of the five logical connectives:

1. Negation: A sentence such as $\neg P$ is called negation of P. A literal can be either Positive literal or negative literal.

Example: P = Today is not Sunday $\rightarrow \neg p$

1. Conjunction: A sentence which has \wedge connective such as, $P \wedge Q$ is called a conjunction. Example: Rohan is intelligent and hardworking. It can be written as,

P = Rohan is intelligent,

Q = Rohan is hardworking. $\rightarrow P \wedge Q$.

2. Disjunction: A sentence which has \vee connective, such as $P \vee Q$. is called disjunction, where P and Q are the propositions.

3. Example: "Ritika is a doctor or Engineer",

Here P = Ritika is Doctor. Q = Ritika is Engineer, so we can write it as $P \vee Q$.

4. Implication: A sentence such as $P \rightarrow Q$, is called an implication. Implications are also known as if-then rules. It can be represented as

If it is raining, then the street is wet.

Let P = It is raining, and Q = Street is wet, so it is represented as $P \rightarrow Q$

5. Biconditional: A sentence such as $P \Leftrightarrow Q$ is a Biconditional sentence, example If I am breathing, then I am alive

P = I am breathing, Q = I am alive, it can be represented

as $P \Leftrightarrow Q$. Precedence of connectives:

Precedence Operators

First Precedence

Parenthesis Second

Precedence

Negation

Third Precedence

Conjunction(AND)

Fourth Precedence

Disjuncti

on(OR) Fifth Precedence

Implicati

on

Six Precedence

Biconditional

Precedence of connectives:

Semantics

- The semantics of propositional logic is also quite straightforward. We define it by specifying the interpretation of the proposition symbols and constants, and specifying the meanings of the logical connectives.

Validity

- Truth tables can be used not only to define the connectives, but also to test for valid sentences.
- Given a sentence, we make a truth table with one row for each of the possible combinations of truth values for the proposition symbols in the sentence.
- If the sentence is true in every row, then the sentence is valid. For example, the sentence $((P \vee H) \wedge \neg H) \Rightarrow P$

Translating English into logic:

- User defines semantics of each propositional symbol
- P: It is Hot
- Q: It is Humid

- R:It is raining

1. If it is humid

then it is hot $Q \rightarrow P$

.If it is hot and humid , then

it is raining $(P \wedge Q) \rightarrow R$

Limitations of Propositional logic:

- In propositional logic, we can only represent the facts, which are either true or false.
- PL is not sufficient to represent the complex sentences or natural language statements.
- The propositional logic has very limited expressive power.
- Consider the following sentence, which we cannot represent using PL logic.
- "Some humans are intelligent", or "Sachin likes cricket"

First-order logic:

Advantages of Propositional Logic

- The declarative nature of propositional logic, specify that knowledge and inference are separate, and inference is entirely domain-independent. Propositional logic is a declarative language because its semantics is based on a truth relation between sentences and possible worlds.
- It also has sufficient expressive power to deal with partial information, using disjunction and negation.
- Propositional logic has a third COMPOSITIONALITY property that is desirable in representation languages, namely, compositionality. In a compositional language, the meaning of a sentence is a function of the meaning of its parts. For example, the meaning of "S1,4A S1,2" is related to the meanings of "S1,4" and "S1,2."

Drawbacks of Propositional Logic

Propositional logic lacks the expressive power to concisely describe an environment with many objects.

For example, we were forced to write a separate rule about breezes and pits for each square, such as $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$.

In English, it seems easy enough to say, "Squares adjacent to pits are breezy."

The syntax and semantics of English somehow make it possible to describe the environment concisely

SYNTAX AND SEMANTICS OF FIRST-ORDER LOGIC

Models for first-order logic :

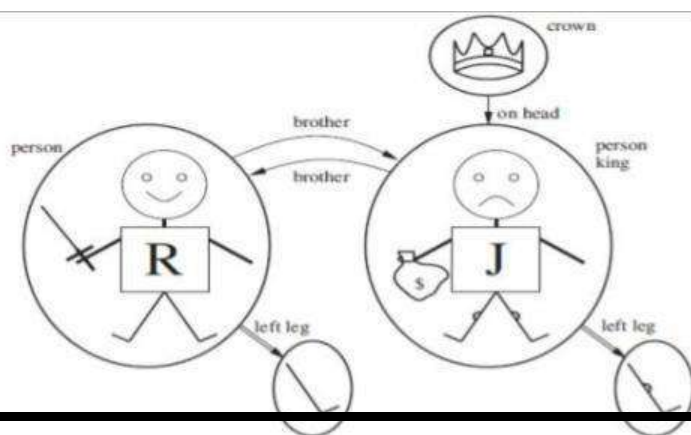
The models of a logical language are the formal structures that constitute the possible worlds under consideration. Each model links the vocabulary of the logical sentences to elements of the possible world, so that the truth of any sentence can be determined. Thus, models for propositional logic link proposition symbols to predefined truth values. Models for first-order logic have objects. The domain of a model is the set of objects or domain elements it contains. The domain is required to be nonempty—every possible world must contain at least one object.

A relation is just the set of tuples of objects that are related.

Unary Relation: Relations relates to single Object Binary Relation: Relation Relates to multiple objects Certain kinds of relationships are best considered as functions, in that a given object must be related to exactly one object.

For Example:

Richard the Lionheart, King of England from 1189 to 1199; His younger brother, the evil King John, who ruled from 1199 to 1215; the left legs of Richard and John; crown



Unary Relation : John is a king Binary Relation :crown is on head of john , Richard is brother ofjohn The unary "left leg" function includes the following mappings: (Richard the Lionheart) ->Richard's left leg (King John) ->Johns left Leg

Symbols and interpretations

Symbols are the basic syntactic elements of first-order logic. Symbols stand for objects, relations, and functions.

The symbols are of three kinds: Constant symbols which stand for objects; Example: John, Richard Predicate symbols, which stand for relations; Example: OnHead, Person, King, and Crown

Function symbols, which stand for functions. Example: left leg Symbols will begin with uppercase letters.

Interpretation The semantics must relate sentences to models in order to determine truth. For this to happen, we need an interpretation that specifies exactly which objects, relations and functions are referred to by the constant, predicate, and function symbols.

For Example:

Richard refers to Richard the Lionheart and John refers to the evil king John. Brother refers to the brotherhood relation OnHead refers to the "on head relation that holds between the crown and King John; Person, King, and Crown refer to the sets of objects that are persons, kings, and crowns. LeftLeg refers to the "left leg" function,

The truth of any sentence is determined by a model and an interpretation for the sentence's symbols. Therefore, entailment, validity, and so on are defined in terms of all possible models and all possible interpretations. The number of domain elements in each model may be unbounded-for example, the domain elements may be integers or real numbers. Hence, the number of possible models is unbounded, as is the number of interpretations.

Term

A term is a logical expression that refers to an object. Constant symbols are therefore terms. Complex Terms A complex term is just a complicated kind of name. A complex term is formed by a function symbol followed by a parenthesized list of terms as arguments to the function symbol For example: "KingJohn's left leg" Instead of using a constant symbol, we use LeftLeg(John). The formal semantics of terms Consider a term $f(t_1, \dots, t_n)$. The function symbol refers to some function in the model (F); the argument terms refer to objects in the domain (call them d_1, \dots, d_n); and the term as a whole refers to the object that is the value of the function F applied to d_1, \dots, d_n . For example,; the LeftLeg function symbol refers to the function “ (King John) \rightarrow John's left leg” and John refers to King John, then LeftLeg(John) refers to KingJohn's left leg. In this way, the interpretation fixes the referent of every term.

Atomic sentences

An atomic sentence is formed from a predicate symbol followed by a parenthesized list of terms: ForExample: Brother(Richard, John).

Atomic sentences can have complex terms as arguments. For Example: Married (Father(Richard),Mother(John)).

An atomic sentence is true in a given model, under a given interpretation, if the relation referred to by the predicate symbol holds among the objects referred to by the arguments

Complex sentences Complex sentences can be constructed using logical Connectives, just as in propositional calculus. For Example:

- ✓ \neg Brother (LeftLeg(Richard), John)
- ✓ Brother (Richard , John) \wedge Brother (John, Richard)
- ✓ King(Richard) \vee King(John)
- ✓ \neg King(Richard) \Rightarrow King(John)

Quantifiers

Quantifiers express properties of entire collections of objects, instead of enumerating the objects by name.

First-order logic contains two standard quantifiers:

1. Universal Quantifier
2. Existential Quantifier

Universal Quantifier

Universal quantifier is defined as follows:

“Given a sentence $\forall x P$, where P is any logical expression, says that P is true for every object x .”

More precisely, $\forall x P$ is true in a given model if P is true in all possible **extended interpretations** constructed from the interpretation given in the model, where each extended interpretation specifies a domain element to which x refers.

For Example: “All kings are persons,” is written in first-order logic as

$\forall x \text{King}(x) \Rightarrow \text{Person}(x)$.

\forall is usually pronounced “For all”

Thus, the sentence says, “For all x , if x is a king, then x is a person.” The symbol x is called a variable. Variables are lowercase letters. A variable is a term all by itself, and can also serve as the argument of a function. A term with no variables is called a ground term.

Assume we can extend the interpretation in different ways: $x \rightarrow$ Richard the Lionheart, $x \rightarrow$ King John, $x \rightarrow$ Richard’s left leg, $x \rightarrow$ John’s left leg, $x \rightarrow$ the crown

The universally quantified sentence $\forall x \text{ King}(x) \Rightarrow \text{Person}(x)$ is true in the original model if the sentence $\text{King}(x)$

$\Rightarrow \text{Person}(x)$ is true under each of the five extended interpretations. That is, the universally quantified sentence is equivalent to asserting the following five sentences:

Richard the Lionheart is a king \Rightarrow Richard the Lionheart is a person. King John is a king \Rightarrow King John is a person. Richard’s left leg is a king \Rightarrow Richard’s left leg is a person. John’s left leg is a king \Rightarrow John’s left leg is a person. The crown is a king \Rightarrow the crown is a person.

Existential quantification (\exists)

Universal quantification makes statements about every object. Similarly, we can make a statement about some object in the universe without naming it, by using an existential quantifier.

“The sentence $\exists x P$ says that P is true for at least one object x . More precisely, $\exists x P$ is true in a given model if P is true in at least one extended interpretation that assigns x to a domain element.” $\exists x$ is pronounced “There exists an x such that . . .”

or “For some x . . .”.

For example, that King John has a crown on his head, we write $\exists x \text{Crown}(x) \wedge \text{OnHead}(x, \text{John})$ Given assertions:

Richard the Lionheart is a crown \wedge Richard the Lionheart is on John’s head; King John is a crown \wedge King John is on John’s head; Richard’s left leg is a crown \wedge Richard’s left leg is on John’s head; John’s left leg is a crown \wedge John’s left leg is on John’s head; The crown is a crown \wedge the crown is on John’s head. The fifth assertion is true in the model, so the original existentially quantified sentence is true in the model. Just as \Rightarrow appears to be the natural connective to use with \forall , \wedge is the natural connective to use with \exists .

Nested quantifiers

One can express more complex sentences using multiple quantifiers.

For example, “Brothers are siblings” can be written as $\forall x \forall y \text{ Brother}(x, y) \Rightarrow \text{Sibling}(x, y)$. Consecutive

quantifiers of the same type can be written as one quantifier with several variables.

For example, to say that siblinghood is a symmetric relationship, we can write $\forall x, y \text{ Sibling}(x, y)$

$\Leftrightarrow \text{Sibling}(y, x)$. In other cases we will have mixtures.

For example: 1. "Everybody loves somebody" means that for every person, there is someone that person loves: $\forall x \exists y \text{ Loves}(x, y)$. 2. On the other hand, to say "There is someone who is loved by everyone," we write $\exists y \forall x \text{ Loves}(x, y)$.

Connections between \forall and \exists

Universal and Existential quantifiers are actually intimately connected with each other, through negation.

Example assertions:

1. "Everyone dislikes medicine" is the same as asserting "there does not exist someone who likes medicine", and vice versa: " $\forall x \neg \text{Likes}(x, \text{medicine})$ " is equivalent to " $\neg \exists x \text{ Likes}(x, \text{medicine})$ ".

2. "Everyone likes ice cream" means that "there is no one who does not like ice cream": " $\forall x \text{ Likes}(x, \text{IceCream})$ " is equivalent to " $\neg \exists x \neg \text{Likes}(x, \text{IceCream})$ ".

Because \forall is really a conjunction over the universe of objects and \exists is a disjunction that they obey De Morgan's rules. The De Morgan rules for quantified and unquantified sentences are as follows:

Because \forall is really a conjunction over the universe of objects and \exists is a disjunction that they obey De Morgan's rules. The De Morgan rules for quantified and unquantified sentences are as follows:

$$\begin{array}{ll} \forall x \neg P \equiv \neg \exists x P & \neg(P \vee Q) \equiv \neg P \wedge \neg Q \\ \neg \forall x P \equiv \exists x \neg P & \neg(P \wedge Q) \equiv \neg P \vee \neg Q \\ \forall x P \equiv \neg \exists x \neg P & P \wedge Q \equiv \neg(\neg P \vee \neg Q) \\ \exists x P \equiv \neg \forall x \neg P & P \vee Q \equiv \neg(\neg P \wedge \neg Q) \dots \end{array}$$

Thus, Quantifiers are important in terms of readability.

Equality

First-order logic includes one more way to make atomic sentences, other than using a predicate and terms. We can use the equality symbol to signify that two terms refer to the same object.

For example,

“Father(John) =Henry” says that the object referred to by Father (John) and the object referred to byHenry are the same.

Because an interpretation fixes the referent of any term, determining the truth of an equality sentence is simply a matter of seeing that the referents of the two terms are the same object.The equality symbol can be used to state facts about a given function.It can also be used with negation to insist that two terms are not the same object.

For example,

“Richard has at least two brothers” can be written as, $\exists x, y \text{ Brother}(x, \text{Richard}) \wedge \text{Brother}(y, \text{Richard}) \wedge \neg(x=y)$. Thse $\exists x, y \text{ Brother}(x, \text{Richard}) \wedge \text{Brother}(y, \text{Richard})$ does not have the intended meaning.

In particular, it is true only in the model where Richard has only one brother considering the extended interpretation in which both x and y are assigned to King John. The addition of $\neg(x=y)$ rules out such models.

<i>Sentence</i>	→	<i>AtomicSentence</i> <i>ComplexSentence</i>
<i>AtomicSentence</i>	→	<i>Predicate</i> <i>Predicate(Term, ...)</i> <i>Term = Term</i>
<i>ComplexSentence</i>	→	(<i>Sentence</i>) <i>Sentence</i>
		\neg <i>Sentence</i>
		<i>Sentence</i> \wedge <i>Sentence</i>
		<i>Sentence</i> \vee <i>Sentence</i>
		<i>Sentence</i> \Rightarrow <i>Sentence</i>
		<i>Sentence</i> \Leftrightarrow <i>Sentence</i>
		<i>Quantifier</i> <i>Variable, ...</i> <i>Sentence</i>
<i>Term</i>	→	<i>Function(Term, ...)</i>
		<i>Constant</i>
		<i>Variable</i>
<i>Quantifier</i>	→	\forall \exists
<i>Constant</i>	→	<i>A</i> <i>X₁</i> <i>John</i> ...
<i>Variable</i>	→	<i>a</i> <i>x</i> <i>s</i> ...
<i>Predicate</i>	→	<i>True</i> <i>False</i> <i>After</i> <i>Loves</i> <i>Raining</i> ...
<i>Function</i>	→	<i>Mother</i> <i>LeftLeg</i> ...
OPERATOR PRECEDENCE	:	$\neg, =, \wedge, \vee, \Rightarrow, \Leftrightarrow$

Backus Naur Form for First Order Logic

USING FIRST ORDER LOGIC Assertions and queries in first-order

logicAssertions:

Sentences are added to a knowledge base using TELL, exactly as in propositional logic. Such sentences are called assertions.

For example,

John is a king, TELL (KB, King (John)). Richard is a person. TELL (KB, Person (Richard)). All kings are persons: TELL (KB, $\forall x \text{ King}(x) \Rightarrow \text{Person}(x)$).

Asking Queries:

We can ask questions of the knowledge base using ASK. Questions asked with ASK are called queries or goals.

For example,

ASK (KB, King (John)) returns true.

Any query that is logically entailed by the knowledge base should be answered

affirmatively. For example, given the two preceding assertions, the query:

“ASK (KB, Person (John))” should also return true.

Substitution or binding list

We can ask quantified queries, such as ASK (KB, $\exists x \text{ Person}(x)$).

The answer is true, but this is perhaps not as helpful as we would like. It is rather like answering “Can you tell me the time?” with “Yes.”

If we want to know what value of x makes the sentence true, we will need a different function, ASKVARS, which we call with ASKVARS (KB, Person(x)) and which yields a stream of answers.

In this case there will be two answers: $\{x/\text{John}\}$ and $\{x/\text{Richard}\}$. Such an answer is called a

substitution or binding list.

ASKVARS is usually reserved for knowledge bases consisting solely of Horn clauses, because in such knowledge bases every way of making the query true will bind the variables to specific values.

The kinship domain

The objects in Kinship domain are people.

We have two unary predicates, Male and Female.

Kinship relations—parenthood, brotherhood, marriage, and so on—are represented by binary predicates: Parent, Sibling, Brother, Sister, Child, Daughter, Son, Spouse, Wife, Husband, Grandparent, Grandchild, Cousin, Aunt, and Uncle.

We use functions for Mother and Father, because every person has exactly one of each of these.

We can represent each function and predicate, writing down what we know in terms of the other symbols.

For example:-

1. one's mother is one's female parent: $\forall m, c \text{ Mother}(c)=m \Leftrightarrow \text{Female}(m) \wedge \text{Parent}(m, c)$
2. One's husband is one's male spouse: $\forall w, h \text{ Husband}(h,w) \Leftrightarrow \text{Male}(h) \wedge \text{Spouse}(h,w)$
3. Male and female are disjoint categories: $\forall x \text{ Male}(x) \Leftrightarrow \neg \text{Female}(x)$
4. Parent and child are inverse relations: $\forall p, c \text{ Parent}(p, c) \Leftrightarrow \text{Child}(c, p)$
5. A grandparent is a parent of one's parent: $\forall g, c \text{ Grandparent}(g, c) \Leftrightarrow \exists p \text{ Parent}(g, p) \wedge \text{Parent}(p, c)$
6. A sibling is another child of one's parents: $\forall x, y \text{ Sibling}(x, y) \Leftrightarrow x \neq y \wedge \exists p \text{ Parent}(p, x) \wedge \text{Parent}(p, y)$

Axioms:

Each of these sentences can be viewed as an axiom of the kinship domain. Axioms are commonly associated with purely mathematical domains. They provide the basic factual information

from which useful conclusions can be derived.

Kinship axioms are also definitions; they have the form $\forall x, y P(x, y) \Leftrightarrow \dots$

The axioms define the Mother function, Husband, Male, Parent, Grandparent, and Sibling predicates in terms of other predicates.

Our definitions “bottom out” at a basic set of predicates (Child, Spouse, and Female) in terms of which the others are ultimately defined. This is a natural way in which to build up the representation of a domain, and it is analogous to the way in which software packages are built up by successive definitions of subroutines from primitive library functions.

Theorems:

Not all logical sentences about a domain are axioms. Some are theorems—that is, they are entailed by the axioms.

For example, consider the assertion that siblinghood is symmetric: $\forall x, y \text{ Sibling}(x, y) \Leftrightarrow \text{Sibling}(y, x)$.

It is a theorem that follows logically from the axiom that defines siblinghood. If we ASK the knowledge base this sentence, it should return true. From a purely logical point of view, a knowledge base need contain only axioms and no theorems, because the theorems do not increase the set of conclusions that follow from the knowledge base. From a practical point of view, theorems are essential to reduce the computational cost of deriving new sentences. Without them, a reasoning system has to start from first principles every time.

Axioms :Axioms without Definition

Not all axioms are definitions. Some provide more general information about certain predicates without

constituting a definition. Indeed, some predicates have no complete definition because we do not know enough to characterize them fully.

For example, there is no obvious definitive way to complete the sentence

$\forall x \text{ Person}(x) \Leftrightarrow \dots$

Fortunately, first-order logic allows us to make use of the Person predicate without completely defining it. Instead, we can write partial specifications of properties that every person has and properties that make something a person:

$\forall x \text{Person}(x) \Rightarrow \dots \forall x \dots \Rightarrow \text{Person}(x)$.

Axioms can also be “just plain facts,” such as Male (Jim) and Spouse (Jim, Laura). Such facts form the descriptions of specific problem instances, enabling specific questions to be answered. The answers to these questions will then be theorems that follow from the axioms

Numbers, sets, and lists Number theory

Numbers are perhaps the most vivid example of how a large theory can be built up from NATURAL NUMBERS a tiny kernel of axioms. We describe here the theory of natural numbers or non-negative integers. We need:

predicate NatNum that will be true of natural numbers;

PEANO AXIOMS constant symbol, 0; One function symbol, S (successor). The Peano axioms define natural numbers and addition.

Natural numbers are defined recursively: $\text{NatNum}(0)$. $\forall n \text{NatNum}(n) \Rightarrow \text{NatNum}(S(n))$.

That is, 0 is a natural number, and for every object n, if n is a natural number, then S(n) is a natural number.

So the natural numbers are 0, S(0), S(S(0)), and so on. We also need axioms to constrain the successor function: $\forall n 0 \neq S(n)$. $\forall m, n m \neq n \Rightarrow S(m) \neq S(n)$.

Now we can define addition in terms of the successor function: $\forall m \text{NatNum}(m) \Rightarrow + (0, m) = m$.
 $\forall m, n \text{NatNum}(m) \wedge \text{NatNum}(n) \Rightarrow + (S(m), n) = S(+ (m, n))$

The first of these axioms says that adding 0 to any natural number m gives m itself.

Addition is represented using the binary function symbol “+” in the term + (m, 0);

To make our sentences about numbers easier to read, we allow the use of infix notation. We

can also write $S(n)$ as $n + 1$, so the second axiom becomes :

$$\forall m, n \in \text{NatNum} \quad (m) \wedge \text{NatNum}(n) \Rightarrow (m + 1) + n = (m + n) + 1 .$$

This axiom reduces addition to repeated application of the successor function. Once we have addition, it is straightforward to define multiplication as repeated addition, exponentiation as repeated multiplication, integer division and remainders, prime numbers, and so on. Thus, the whole of number theory (including cryptography) can be built up from one constant, one function, one predicate and four axioms.

Sets

The domain of sets is also fundamental to mathematics as well as to commonsense reasoning. Sets can be represented as individual sets, including empty sets.

Sets can be built up by:

adding an element to a set or

Taking the union or intersection of

two sets. Operations that can be

performed on sets are:

To know whether an element is a member of a set Distinguish sets from objects that are not sets.

Vocabulary of set theory:

The empty set is a constant written as $\{ \}$. There is one unary predicate, Set , which is true of sets.

The binary predicates are

$x \in s$ (x is a member of set s) $s_1 \subseteq s_2$ (set s_1 is a subset, not necessarily proper, of set s_2).

The binary functions are

$s_1 \cap s_2$ (the intersection of two sets), $s_1 \cup s_2$ (the union of two sets), and $\{x|s\}$ (the set resulting from adjoining element x to set s).

Forward Chaining and backward chaining in AI

Inference engine:

The inference engine is the component of the intelligent system in artificial intelligence, which applies logical rules to the knowledge base to infer new information from known facts. The first inference engine was part of the expert system. Inference engine commonly proceeds in two modes, which are:

- a. **Forward chaining**
- b. **Backward**

chainingHorn Clause

and Definite clause:

Horn clause and definite clause are the forms of sentences, which enables knowledge base to use a more restricted and efficient inference algorithm. Logical inference algorithms use forward and backward chaining approaches, which require KB in the form of the **first-order definite clause**.

Definite clause: A clause which is a disjunction of literals with **exactly one positive literal** is known as a definite clause or strict horn clause.

Horn clause: A clause which is a disjunction of literals with **at most one positive literal** is known as horn clause. Hence all the definite clauses are horn clauses.

Example: $(\neg p \vee \neg q \vee k)$. It has only one positive

literal k. It is equivalent to $p \wedge q \rightarrow k$.

A. Forward Chaining

Forward chaining is also known as a forward deduction or forward reasoning method when using an inference engine. Forward chaining is a form of reasoning which start with atomic sentences in the knowledge base and applies inference rules (Modus Ponens) in the forward direction to extract more data until a goal is reached.

The Forward-chaining algorithm starts from known facts, triggers all rules whose premises are satisfied, and add their conclusion to the known facts. This process repeats until the problem is solved.

Properties of Forward-Chaining:

- It is a down-up approach, as it moves from bottom to top.
- It is a process of making a conclusion based on known facts or data, by starting from the initial state and reaches the goal state.
- Forward-chaining approach is also called as data-driven as we reach to the goal using available data.
- Forward -chaining approach is commonly used in the expert system, such as CLIPS, business, and production rule systems.

Consider the following famous example which we will use in both approaches:

Facts Conversion into FOL:

- It is a crime for an American to sell weapons to hostile nations. (Let's say p, q, and r are variables)

American (p) A weapon(q) A sells (p, q, r) A hostile(r) → Criminal(p) ... (1)

- Country A has some missiles. **?p Owns(A, p) A Missile(p)**. It can be written in two definite clauses by using Existential Instantiation, introducing new Constant T1. **Owns(A, T1)**

Missile(T1) (3)

- All of the missiles were sold to country A by Robert.

?p Missiles(p) A Owns (A, p) → Sells (Robert, p, A) (4)

- Missiles are weapons.

Missile(p) → Weapons (p) (5)

- Enemy of America is known as hostile.

Enemy(p, America) → Hostile(p) (6)

- Country A is an enemy of America.

Enemy (A, America) (7)

- Robert is American

American(Robert) (8) Forward chaining proof:

Step-1:

In the first step we will start with the known facts and will choose the sentences which do not have implications, such as: **American(Robert)**, **Enemy(A, America)**, **Owns(A, T1)**, and **Missile(T1)**. All these facts will be represented as below.



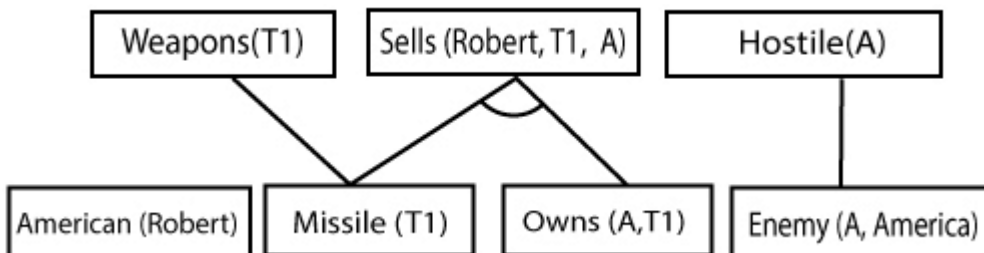
Step-2:

At the second step, we will see those facts which infer from available facts and with satisfied premises. Rule-(1) does not satisfy premises, so it will not be added in the first iteration.

Rule-(2) and (3) are already added.

Rule-(4) satisfy with the substitution $\{p/T1\}$, so **Sells (Robert, T1, A)** is added, which infers from the conjunction of Rule (2) and (3).

Rule-(6) is satisfied with the substitution (p/A) , so **Hostile(A)** is added and which infers from Rule-(7).

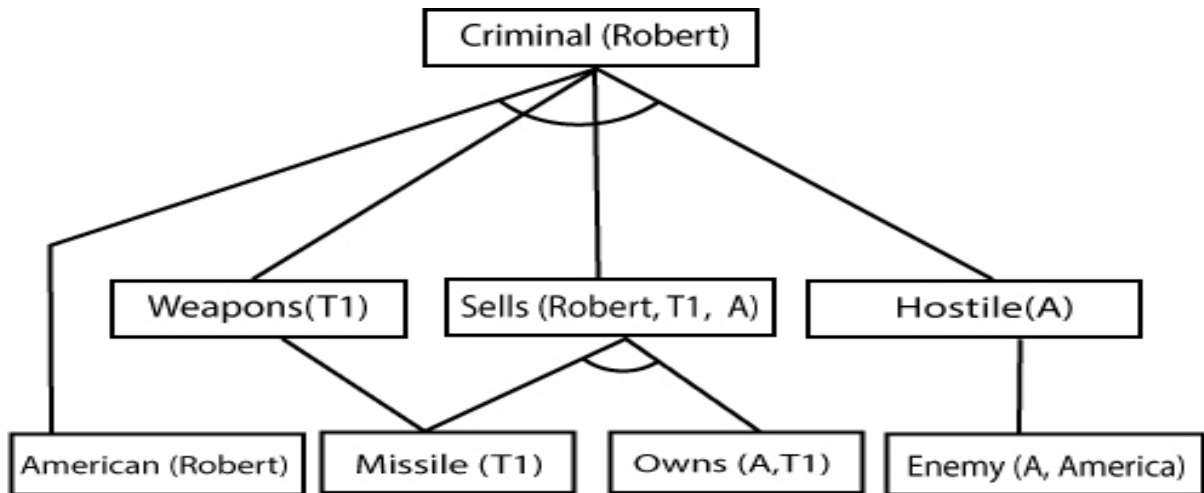


Step-3:

At step-3, as we can check Rule-(1) is satisfied with the substitution $\{p/Robert, q/T1, r/A\}$, so we can add **Criminal(Robert)** which infers all the available facts. And hence we reached our goal statement.

Hence it is proved that Robert is Criminal using forward chaining approach.

Backward Chaining:



Backward-chaining is also known as a backward deduction or backward reasoning method when using an inference engine. A backward chaining algorithm is a form of reasoning, which starts with the goal and works backward, chaining through rules to find known facts that support the goal.

Properties of backward chaining:

- It is known as a top-down approach.
- Backward-chaining is based on modus ponens inference rule.
- In backward chaining, the goal is broken into sub-goal or sub-goals to prove the facts true.
- It is called a goal-driven approach, as a list of goals decides which rules are selected and used.
- Backward -chaining algorithm is used in game theory, automated theorem proving tools, inference engines, proof assistants, and various AI applications.
- The backward-chaining method mostly used a **depth-first search** strategy for proof.

Example:

In backward-chaining, we will use the same above example, and will rewrite all the rules.

- **American (p) A weapon(q) A sells (p, q, r) A hostile(r) → Criminal(p) ...**(1)**Owns(A, T1) (2)**
- **Missile(T1)**
- **?p Missiles(p) A Owns (A, p) → Sells (Robert, p, A).....**(4)
- **Missile(p) → Weapons (p)**(5)

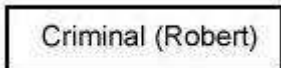
- **Enemy(p, America) → Hostile(p)(6)**
- **Enemy (A, America)..... (7)**
- **American(Robert). (8)**

Backward-Chaining proof:

In Backward chaining, we will start with our goal predicate, which is **Criminal(Robert)**, and then infer further rules.

Step-1:

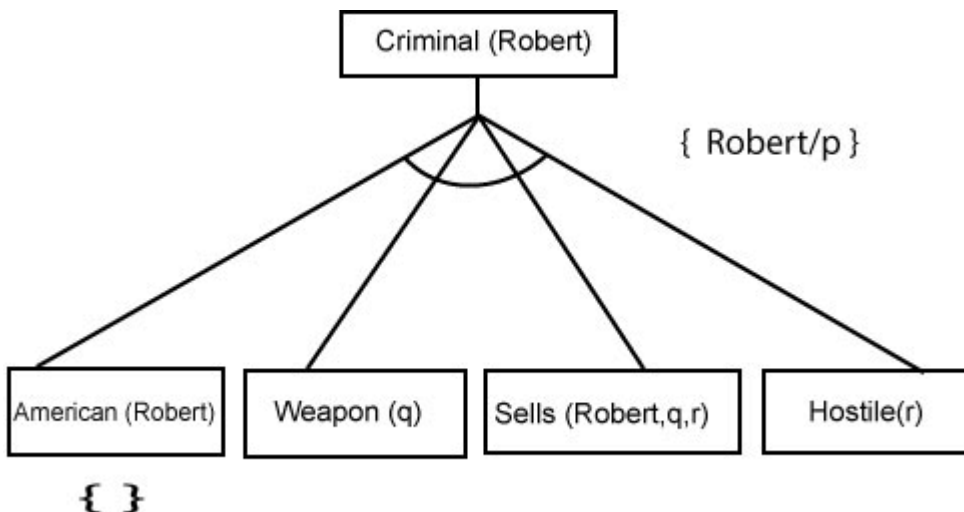
At the first step, we will take the goal fact. And from the goal fact, we will infer other facts, and at last, we will prove those facts true. So our goal fact is "Robert is Criminal," so following is the predicate of it.



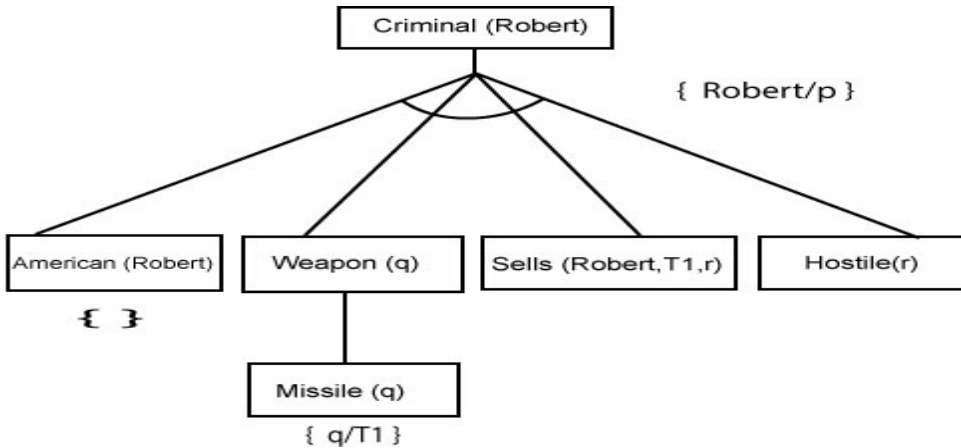
Step-2:

At the second step, we will infer other facts from goal fact which satisfies the rules. So as we can see in Rule-1, the goal predicate Criminal (Robert) is present with substitution {Robert/P}. So we will add all the conjunctive facts below the first level and will replace p with Robert.

Here we can see American (Robert) is a fact, so it is proved here.

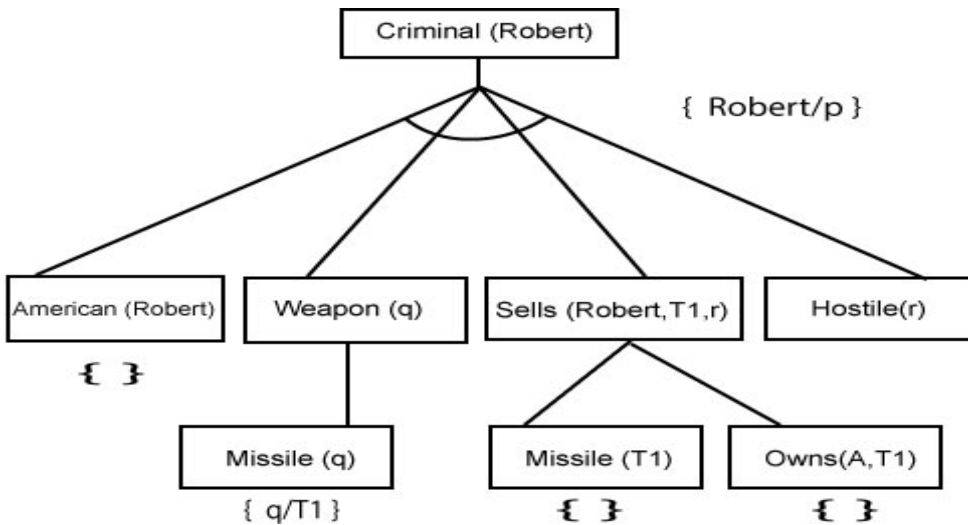


Step-3: At step-3, we will extract further fact Missile(q) which infer from Weapon(q), as it satisfies Rule-(5). Weapon (q) is also true with the substitution of a constant T1 at q.



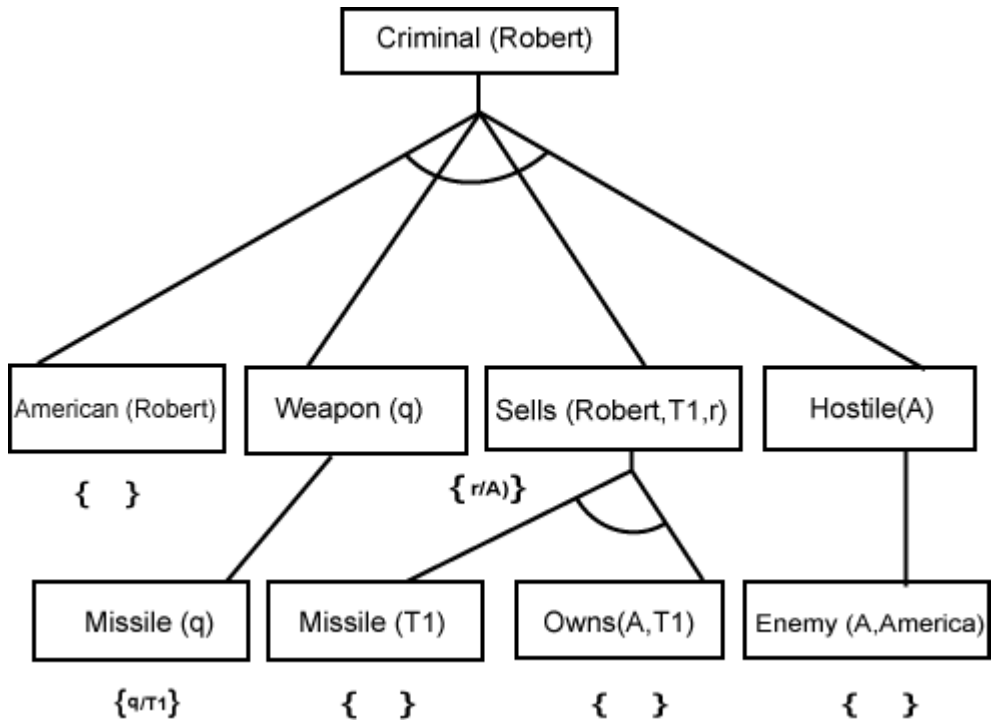
Step-4:

At step-4, we can infer facts Missile(T1) and Owns(A, T1) from Sells(Robert, T1, r) which satisfies the **Rule- 4**, with the substitution of A in place of r. So these two statements are proved here.



Step-5:

At step-5, we can infer the fact **Enemy(A, America)** from **Hostile(A)** which satisfies Rule- 6. Andhence all the statements are proved true using backward chaining.



Difference between backward chaining and forward chaining

No.	Forward Chaining	Backward Chaining
1	Forward chaining starts from known facts and applies inference rule to extract more data unit it reaches to the goal.	Backward chaining starts from the goal and works backward through inference rules to find the required facts that support the goal.
2	It is a bottom-up approach	It is a top-down approach
3	Forward chaining is known as data-driven inference technique as we reach to the goal using the available data.	Backward chaining is known as goal-driven technique as we start from the goal and divide into sub-goal to extract the facts.

4	Forward chaining reasoning applies a breadth-first search strategy.	Backward chaining reasoning applies a depth-first search strategy.
5	Forward chaining tests for all the available rules	Backward chaining only tests for few required rules.
6	Forward chaining is suitable for the planning, monitoring, control, and interpretation application.	Backward chaining is suitable for diagnostic, prescription, and debugging application.
7	Forward chaining can generate an infinite number of possible conclusions.	Backward chaining generates a finite number of possible conclusions.
8	It operates in the forward direction.	It operates in the backward direction.
9	Forward chaining is aimed for any conclusion.	Backward chaining is only aimed for the required data.

Basic probability notation

- **Prior probability** :We will use the notation $P(A)$ for the unconditional or prior probability that the proposition A is true.
- For example, if $Cavity$ denotes the proposition that a particular patient has a cavity, $P(Cavity) = 0.1$ means that in the absence of any other information, the agent will assign a probability of 0.1 (a 10% chance)
- It is important to remember that $P(A)$ can only be used when there is no other information. As soon as some new information B is known, we have to reason with the **conditional probability** of A given B instead of $P(A)$ to the event of the patient's having a cavity.
- Propositions can also include equalities involving so-called random variables.
- For example, if we are concerned about the random

variable Weather, we might have $P(\text{Weather} = \text{Sunny}) = 0.7$

$P(\text{Weather} =$

$\text{Rain}) = 0.2$

$P(\text{Weather} =$

$\text{Cloudy}) = 0.08$

$P(\text{Weather} =$

$\text{Snow}) = 0.02$

Each random variable X has a domain of possible values (x_1, \dots, x_n) that it can take on.

- We can view proposition symbols as random variables as well, if we assume that they have a domain $\{\text{true}, \text{false}\}$.
- Thus, the expression $P(\text{Cavity})$ can be viewed as shorthand for $P(\text{Cavity} = \text{true})$.
- Similarly, $P(\neg \text{Cavity})$ is shorthand for $P(\text{Cavity} = \text{false})$.
- Sometimes, we will want to talk about the probabilities of all the possible values of a random variable. In this case, we will use an expression such as $P(\text{Weather})$

• for example, we would write $P(\text{Weather}) = (0.7, 0.2, 0.08, 0.02)$ This statement defines a probability distribution

- We can also use logical connectives to make more complex sentences and assign probabilities to them.

For example, $P(\text{Cavity} \wedge \neg \text{Insured})$

Conditional probability:

- Once the agent has obtained some evidence concerning the previously unknown propositions making up the domain, prior probabilities are no longer applicable.

Instead, we use conditional or posterior probabilities, with the notation $P(A|B)$

- This is read as "the probability of A given that all we know is B ."
- $P(B|A)$ means "Event B given Event A "
- In other words, event A has already happened, now what is the chance of event B ?
- $P(B|A)$ is also called the "Conditional Probability" of B given A .

Ex: Drawing 2 Kings from a Deck

- **Event A is drawing a King first, and Event B is drawing a King second.**
- For the first card the chance of drawing a King is 4 out of 52 (there are 4 Kings in a

deck of 52 cards):

- $P(A) = 4/52$
- But after removing a King from the deck the probability of the 2nd card drawn is less likely to be a King (only 3 of the 51 cards left are Kings):

- $P(B|A) = 3/51$

And so: $P(A \text{ and } B) = P(A) \times P(B|A) = (4/52) \times (3/51) = 12/2652 = 1/221$

- So the chance of getting 2 Kings is 1 in 221, or about 0.5

BAYES Theorem:

- Bayes' Theorem is a way of finding a probability when we know certain other probabilities.

The formula is

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

- Which tells us: how often A happens given that B happens, written $P(A|B)$,
- When we know: How often B happens given that A happens, written $P(B|A)$
- and how likely A is on its own, written $P(A)$
- and how likely B is on its own,

written $P(B)$ Example:

- Dangerous fires are rare (1%)
- But smoke is fairly common (10%) due to barbecues, and 90% of dangerous fires

make smoke We can then discover the **probability of dangerous Fire when there is**

Smoke:

$$P(\text{Fire} | \text{Smoke}) = \frac{P(\text{Fire}) P(\text{Smoke} | \text{Fire})}{P(\text{Smoke})}$$

$$= 1\% \times 90 / 10\%$$

$$= 9\%$$

So it is still worth checking out any smoke

to be sure. Example 2:

You are planning a picnic today, but the morning is cloudy. Oh no! 50% of all rainy days start off cloudy!

But cloudy mornings are common (about 40% of days start cloudy)

And this is usually a dry month (only 3 of 30 days tend to be rainy, or 10%). What is the chance of rain during the day?

We will use Rain to mean rain during the day, and Cloud to mean cloudy morning. The chance of Rain given Cloud is written

$P(\text{Rain} | \text{Cloud})$

So let's put that in the formula:

$P(\text{Rain} | \text{Cloud}) = \frac{P(\text{Rain and Cloud})}{P(\text{Cloud})}$

$P(\text{Cloud} | \text{Rain}) / P(\text{Cloud})$ $P(\text{Rain})$ is

Probability of Rain = 10%

$P(\text{Cloud} | \text{Rain})$ is Probability of Cloud, given that Rain happens =

50% $P(\text{Cloud})$ is Probability of Cloud = 40%

$P(\text{Rain} | \text{Cloud}) = 0.1 \times 0.5 / 0.4 = .125$

Or a 12.5% chance of rain. Not too bad, let's have a picnic!

UNIT- III

Machine-Learning : Introduction. Machine Learning Systems, Forms of Learning: Supervised and Unsupervised Learning, reinforcement – theory of learning – feasibility of learning – Data Preparation– training versus testing and split.

What is Machine Learning ?

Arthur Samuel, a pioneer in the field of artificial intelligence and computer gaming, coined the term “**Machine Learning**”. He defined machine learning as – a “**Field of study that gives computers the capability to learn without being explicitly programmed**”. In a very layman’s manner, Machine Learning(ML) can be explained as automating and improving the learning process of computers based on their experiences without being actually programmed i.e. without any human assistance. The process starts with feeding good quality data and then training our machines(computers) by building machine learning models using the data and different algorithms. The choice of algorithms depends on what type of data do we have and what kind of task we are trying to automate. **Example: Training of students during exams.** While preparing for the exams students don’t actually cram the subject but try to learn it with complete understanding. Before the examination, they feed their machine(brain) with a good amount of high-quality data (questions and answers from different books or teachers’ notes, or online video lectures). Actually, they are training their brain with input as well as output i.e. what kind of approach or logic do they have to solve a different kinds of questions. Each time they solve practice test papers and find the performance (accuracy /score) by comparing answers with the answer key given, Gradually, the performance keeps on increasing, gaining more confidence with the adopted approach. That’s how actually models are built, train machine with data (both inputs and outputs are given to the model), and when the time comes test on data (with input only) and achieve our model scores by comparing its answer with the actual output which has not been fed while training. Researchers are working with assiduous efforts to improve algorithms, and techniques so that these

models perform even much better.



Basic Difference in ML and Traditional Programming?

Traditional Programming: We feed in DATA (Input) + PROGRAM (logic), run it on the machine, and get the output.

Machine Learning: We feed in DATA(Input) + Output, run it on the machine during training and the machine creates its own program(logic), which can be evaluated while testing.

What does exactly learning mean for a computer? A computer is said to be learning from **Experiences** with respect to some class of **Tasks** if its performance in a given task improves with the Experience.

A computer program is said to learn from experience **E** with respect to some class of tasks **T** and performance measure **P**, if its performance at tasks in **T**, as measured by **P**, improves with experience **E** **Example:** playing checkers. **E** = the experience of playing many games of checkers **T** = the task of playing checkers. **P** = the probability that the program will win the next game In general, any machine learning problem can be assigned to one of two broad classifications: Supervised learning and Unsupervised learning.

How does ML work?

Gathering past data in any form suitable for processing. The better the quality of data, the more suitable it will be for modeling

Data Processing – Sometimes, the data collected is in raw form and it needs to be pre-processed. Example: Some tuples may have missing values for certain attributes, and, in this case, it has to be filled with suitable values in order to perform machine learning or any form of data mining. Missing values for numerical attributes such as the price of the house may be replaced with the mean value of the attribute whereas missing values for categorical attributes may be replaced with the attribute with the highest mode. This invariably depends on the types of filters we use. If data is in the form of text or images then converting it to numerical form will be required, be it a list or array or matrix. Simply, Data is to be made relevant and consistent. It is to be converted into a format understandable by the machine

- Divide the input data into training, cross-validation, and test sets. The ratio between the respective sets must be 6:2:2

- Building models with suitable algorithms and techniques on the training set.
- Testing our conceptualized model with data that was not fed to the model at the time of training and evaluating its performance using metrics such as F1 score, precision, and recall.
- Linear Algebra
- Statistics and Probability
- Calculus
- Graph theory
- Programming Skills – Languages such as Python, R, MATLAB, C++, or Octave
-

Machine learning is programming computers to optimize a performance criterion using example data or past experience . We have a model defined up to some parameters, and learning is the execution of a computer program to optimize the parameters of the model using the training data or past experience. The model may be predictive to make predictions in the future, or descriptive to gain knowledge from data.

- The field of study known as machine learning is concerned with the question of how to construct computer programs that automatically improve with experience.

Definition of learning: A computer program is said to *learn* from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks T, as measured by P , improves with experience E.

Examples

- Handwriting recognition learning problem
 - Task T : Recognizing and classifying handwritten words within images
 - Performance P : Percent of words correctly classified
 - Training experience E : A dataset of handwritten words with given classifications
- A robot driving learning problem
 - Task T : Driving on highways using vision sensors
 - Performance P : Average distance traveled before an error
 - Training experience E : A sequence of images and steering commands recorded while observing a human driver

Definition: A computer program which learns from experience is called a machine learning program or simply a learning program .

Classification of Machine Learning

Machine learning implementations are classified into four major categories, depending on the nature of the learning “signal” or “response” available to a learning system which are as follows:

A. Supervised learning:

Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs. The given data is labeled . Both *classification* and *regression* problems are supervised learning problems .

- Example — Consider the following data regarding patients entering a clinic . The data consists of the gender and age of the patients and each patient is labeled as “healthy” or “sick”.

gender	age	label
M	48	sick
M	67	sick
F	53	healthy
M	49	sick
F	32	healthy
M	34	healthy
M	21	healthy

Unsupervised learning:

Unsupervised learning is a type of machine learning algorithm used to draw inferences from datasets consisting of input data without labeled responses. In unsupervised learning algorithms, classification or categorization is not included in the observations. Example: Consider the following data regarding patients entering a clinic. The data consists of the gender and age of the patients.

gender	age
M	48
M	67
F	53
M	49
F	34
M	21

As a kind of learning, it resembles the methods humans use to figure out that certain objects or events are from the same class, such as by observing the degree of similarity between objects. Some recommendation systems that you find on the web in the form of marketing automation are based on this type of learning.

C. Reinforcement learning:

Reinforcement learning is the problem of getting an agent to act in the world so as to maximize its rewards.

A learner is not told what actions to take as in most forms of machine learning but instead must discover which actions yield the most reward by trying them. For example — Consider teaching a dog a new trick: we cannot tell him what to do, what not to do, but we can reward/punish it if it does the right/wrong thing.

When watching the video, notice how the program is initially clumsy and unskilled but steadily improves with training until it becomes a champion.

Semi-supervised learning:

Where an incomplete training signal is given: a training set with some (often many) of the target outputs missing. There is a special case of this principle known as Transduction where the entire set of problem instances is known at learning time, except that part of the targets are missing. Semi-supervised learning is an approach to machine learning that combines small labeled data with a large amount of unlabeled data during training. Semi-supervised learning falls between unsupervised learning and supervised learning.

Supervised learning

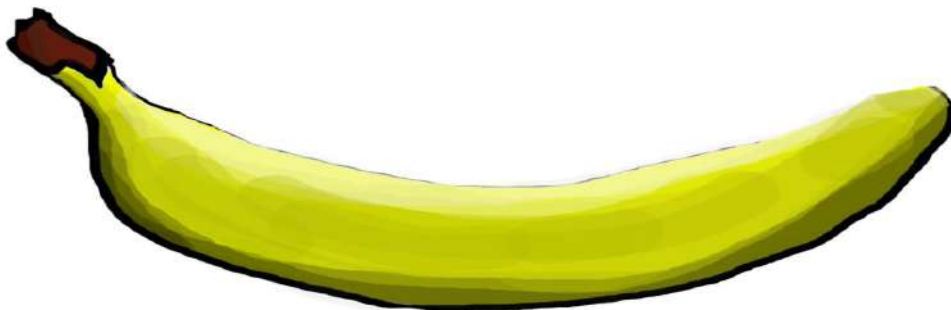
Supervised learning, as the name indicates, has the presence of a supervisor as a teacher. Basically supervised learning is when we teach or train the machine using data that is well labelled. Which means some data is already tagged with the correct answer. After that, the machine is provided with a new set of examples(data) so that the supervised learning algorithm analyses the training data(set of training examples) and produces a correct outcome from labelled data.

For instance, suppose you are given a basket filled with different kinds of fruits. Now the first step is to train the machine with all the different fruits one by one like this:



- If the shape of the object is rounded and has a depression at the top, is red in color, then it will be labeled as –**Apple**.
- If the shape of the object is a long curving cylinder having Green-Yellow color, then it will be labeled as –**Banana**.

Now suppose after training the data, you have given a new separate fruit, say Banana from the basket, and asked to identify it.



Since the machine has already learned the things from previous data and this time has to use it wisely. It will first classify the fruit with its shape and color and would confirm the fruit name as BANANA and put it in the Banana category. Thus the machine learns the things from training data(basket containing fruits) and then applies the knowledge to test data(new fruit).

Supervised learning is the types of machine learning in which machines are trained using well "labelled" training data, and on basis of that data, machines predict the output. The labelled data means some input data is already tagged with the correct output.

In supervised learning, the training data provided to the machines work as the supervisor that teaches the machines to predict the output correctly. It applies the same concept as a student learns in the supervision of the teacher.

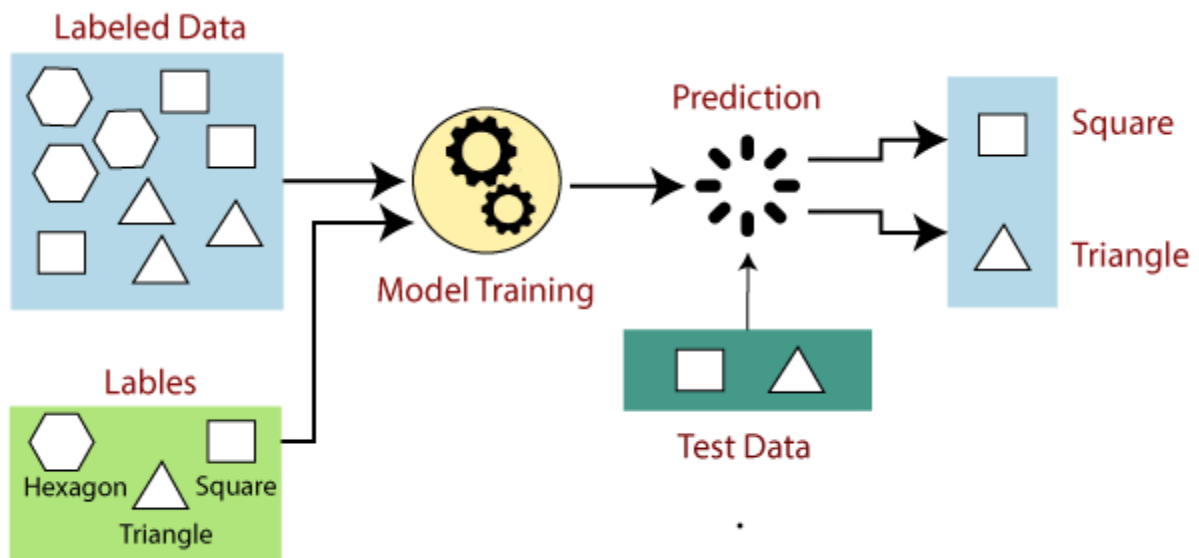
Supervised learning is a process of providing input data as well as correct output data to the machine learning model. The aim of a supervised learning algorithm is to **find a mapping function to map the input variable(x) with the output variable(y)**.

In the real-world, supervised learning can be used for **Risk Assessment, Image classification, Fraud Detection, spam filtering**, etc.

How Supervised Learning Works?

In supervised learning, models are trained using labelled dataset, where the model learns about each type of data. Once the training process is completed, the model is tested on the basis of test data (a subset of the training set), and then it predicts the output.

The working of Supervised learning can be easily understood by the below example and diagram:



Suppose we have a dataset of different types of shapes which includes square, rectangle, triangle, and Polygon. Now the first step is that we need to train the model for each shape.

- If the given shape has four sides, and all the sides are equal, then it will be labelled as a **Square**.
- If the given shape has three sides, then it will be labelled as a **triangle**.
- If the given shape has six equal sides then it will be labelled as **hexagon**.

Now, after training, we test our model using the test set, and the task of the model is to identify the shape.

The machine is already trained on all types of shapes, and when it finds a new shape, it classifies the shape on the bases of a number of sides, and predicts the output.

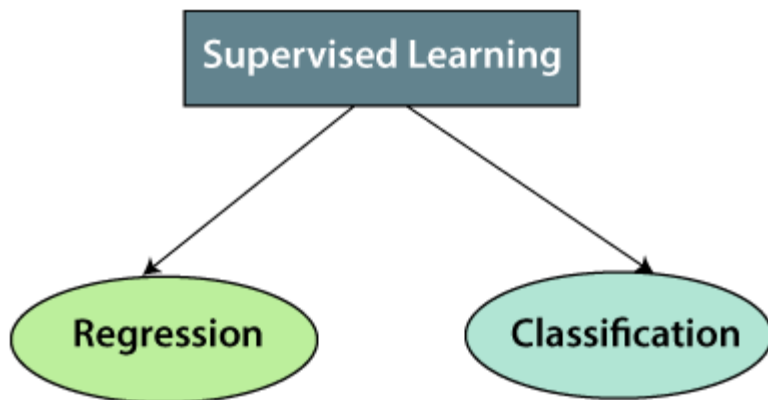
Steps Involved in Supervised Learning:

- First Determine the type of training dataset
- Collect/Gather the labelled training data.

- Split the training dataset into training **dataset, test dataset, and validation dataset**.
- Determine the input features of the training dataset, which should have enough knowledge so that the model can accurately predict the output.
- Determine the suitable algorithm for the model, such as support vector machine, decision tree, etc.
- Execute the algorithm on the training dataset. Sometimes we need validation sets as the control parameters, which are the subset of training datasets.
- Evaluate the accuracy of the model by providing the test set. If the model predicts the correct output, which means our model is accurate.

Types of supervised Machine learning Algorithms:

Supervised learning can be further divided into two types of problems:



1. Regression

Regression algorithms are used if there is a relationship between the input variable and the output variable. It is used for the prediction of continuous variables, such as Weather forecasting, Market Trends, etc. Below are some popular Regression algorithms which come under supervised learning:

- Linear Regression
- Regression Trees
- Non-Linear Regression
- Bayesian Linear Regression
- Polynomial Regression

2. Classification

Classification algorithms are used when the output variable is categorical, which means there are two classes such as Yes-No, Male-Female, True-false, etc.

Spam Filtering,

- Random Forest
- Decision Trees
- Logistic Regression
- Support vector Machines

Advantages of Supervised learning:

- With the help of supervised learning, the model can predict the output on the basis of prior experiences.
- In supervised learning, we can have an exact idea about the classes of objects.
- Supervised learning model helps us to solve various real-world problems such as **fraud detection, spam filtering,** etc.

Disadvantages of supervised learning:

- Supervised learning models are not suitable for handling the complex tasks.
- Supervised learning cannot predict the correct output if the test data is different from the training dataset.
- Training required lots of computation times.
- In supervised learning, we need enough knowledge about the classes of object.

Supervised learning is classified into two categories of algorithms:

- **Classification:** A classification problem is when the output variable is a category, such as “Red” or “blue” , “disease” or “no disease”.
- **Regression:** A regression problem is when the output variable is a real value, such as “dollars” or “weight”.

Supervised learning deals with or learns with “labeled” data. This implies that some data is already tagged with the correct answer.

Types:-

- Regression
- Logistic Regression
- Classification
- Naive Bayes Classifiers
- K-NN (k nearest neighbors)
- Decision Trees
- Support Vector Machine

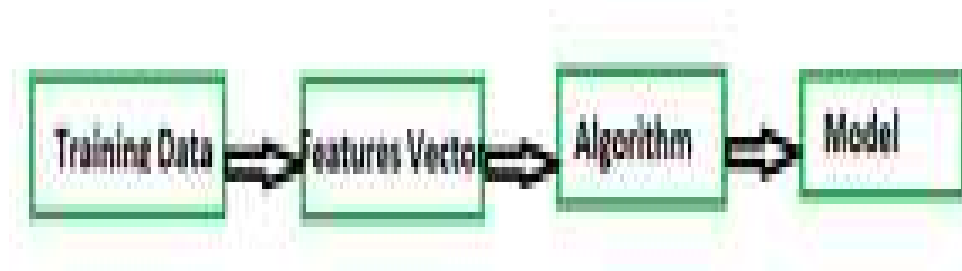
Advantages:-

- Supervised learning allows collecting data and produces data output from previous experiences.
- Helps to optimize performance criteria with the help of experience.

- Supervised machine learning helps to solve various types of real-world computation problems.

Disadvantages:-

- Classifying big data can be challenging.
- Training for supervised learning needs a lot of computation time. So, it requires a lot of time.

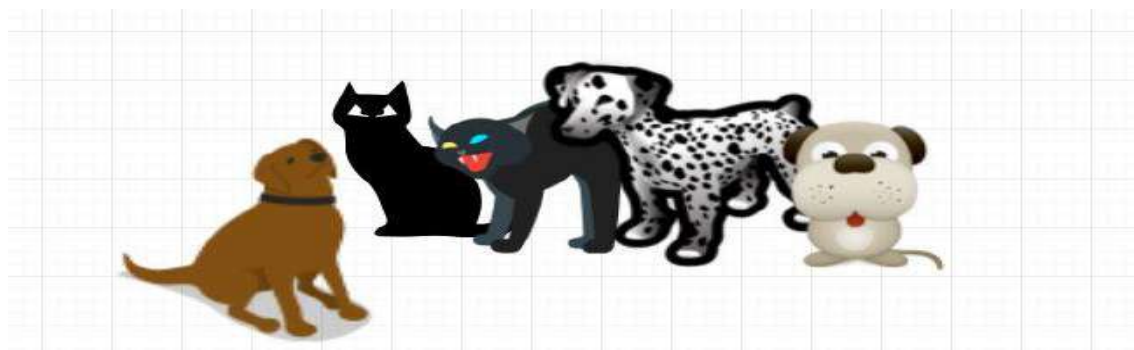


Steps

Unsupervised learning

Unsupervised learning is the training of a machine using information that is neither classified nor labeled and allowing the algorithm to act on that information without guidance. Here the task of the machine is to group unsorted information according to similarities, patterns, and differences without any prior training of data.

Unlike supervised learning, no teacher is provided that means no training will be given to the machine. Therefore the machine is restricted to find the hidden structure in unlabeled data by itself. **For instance**, suppose it is given an image having both dogs and cats which it has never seen.



Thus the machine has no idea about the features of dogs and cats so we can't categorize it as 'dogs and cats'. But it can categorize them according to their similarities, patterns, and differences, i.e., we can easily categorize the above picture into two parts. The first may contain all pics having **dogs** in them and the second part may contain all pics having **cats** in them. Here you didn't learn anything before, which means no training data or examples.

It allows the model to work on its own to discover patterns and information that was previously undetected. It mainly deals with unlabelled data.

Unsupervised learning is classified into two categories of algorithms:

- **Clustering:** A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior.
- **Association:** An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y.

In the previous topic, we learned supervised machine learning in which models are trained using labeled data under the supervision of training data. But there may be many cases in which we do not have labeled data and need to find the hidden patterns from the given dataset. So, to solve such types of cases in machine learning, we need unsupervised learning techniques.

What is Unsupervised Learning?

As the name suggests, unsupervised learning is a machine learning technique in which models are not supervised using training dataset. Instead, models itself find the hidden patterns and insights from the given data. It can be compared to learning which takes place in the human brain while learning new things. It can be defined as:

Unsupervised learning is a type of machine learning in which models are trained using unlabeled dataset and are allowed to act on that data without any supervision.

Unsupervised learning cannot be directly applied to a regression or classification problem because unlike supervised learning, we have the input data but no corresponding output data. The goal of unsupervised learning is to **find the underlying structure of dataset, group that data according to similarities, and represent that dataset in a compressed format.**

Example: Suppose the unsupervised learning algorithm is given an input dataset containing images of different types of cats and dogs. The algorithm is never trained upon the given dataset, which means it does not have any idea about the features of the dataset. The task of the unsupervised learning algorithm is to identify the image features on their own. Unsupervised learning algorithm will perform this task by clustering the image dataset into the groups according to similarities between images.

Why use Unsupervised Learning?

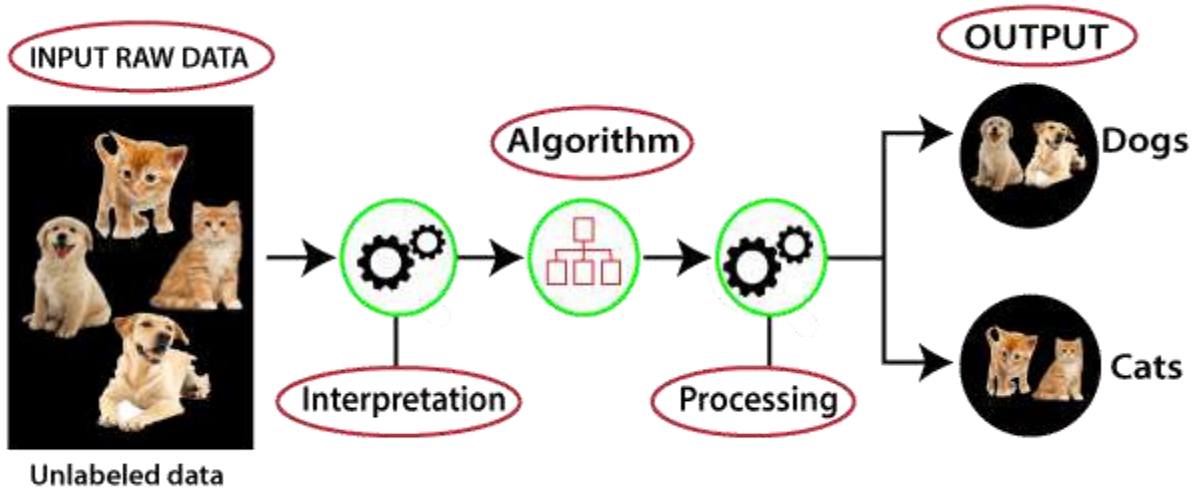
Below are some main reasons which describe the importance of Unsupervised Learning:

- Unsupervised learning is helpful for finding useful insights from the data.
- Unsupervised learning is much similar as a human learns to think by their own experiences, which makes it closer to the real AI.
- Unsupervised learning works on unlabeled and uncategorized data which make unsupervised learning more important.

- In real-world, we do not always have input data with the corresponding output so to solve such cases, we need unsupervised learning.

Working of Unsupervised Learning

Working of unsupervised learning can be understood by the below diagram:

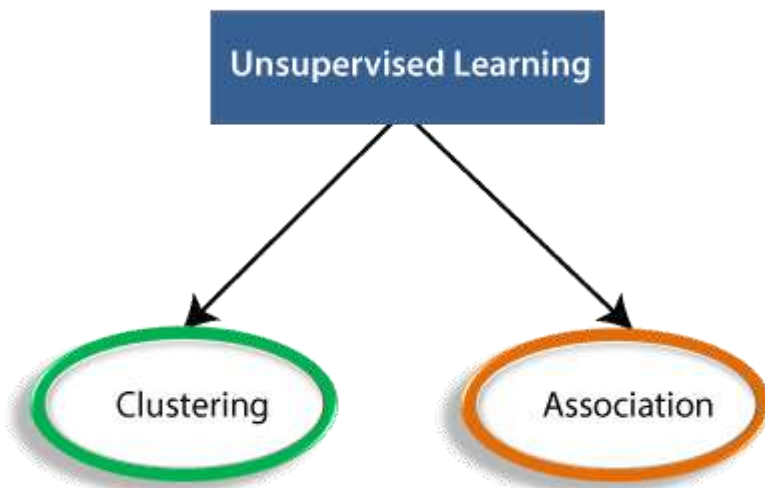


Here, we have taken an unlabeled input data, which means it is not categorized and corresponding outputs are also not given. Now, this unlabeled input data is fed to the machine learning model in order to train it. Firstly, it will interpret the raw data to find the hidden patterns from the data and then will apply suitable algorithms such as k-means clustering, Decision tree, etc.

Once it applies the suitable algorithm, the algorithm divides the data objects into groups according to the similarities and difference between the objects.

Types of Unsupervised Learning Algorithm:

The unsupervised learning algorithm can be further categorized into two types of problems:



- **Clustering:** Clustering is a method of grouping the objects into clusters such that objects with most similarities remains into a group and has less or no similarities with the objects of another group. Cluster analysis finds the commonalities between the data objects and categorizes them as per the presence and absence of those commonalities.
- **Association:** An association rule is an unsupervised learning method which is used for finding the relationships between variables in the large database. It determines the set of items that occurs together in the dataset. Association rule makes marketing strategy more effective. Such as people who buy X item (suppose a bread) are also tend to purchase Y (Butter/Jam) item. A typical example of Association rule is Market Basket Analysis.

Unsupervised Learning algorithms:

Below is the list of some popular unsupervised learning algorithms:

- **K-means clustering**
- **KNN (k-nearest neighbors)**
- **Hierarchical clustering**
- **Anomaly detection**
- **Neural Networks**
- **Principle Component Analysis**
- **Independent Component Analysis**
- **Apriori algorithm**
- **Singular value decomposition**

Advantages of Unsupervised Learning

- Unsupervised learning is used for more complex tasks as compared to supervised learning because, in unsupervised learning, we don't have labeled input data.
- Unsupervised learning is preferable as it is easy to get unlabeled data in comparison to labeled data.

Disadvantages of Unsupervised Learning

- Unsupervised learning is intrinsically more difficult than supervised learning as it does not have corresponding output.
- The result of the unsupervised learning algorithm might be less accurate as input data is not labeled, and algorithms do not know the exact output in advance.

Types of Unsupervised Learning:-

Clustering

1. Exclusive (partitioning)
2. Agglomerative
3. Overlapping
4. Probabilistic

Clustering Types:-

1. Hierarchical clustering
2. K-means clustering
3. Principal Component Analysis
4. Singular Value Decomposition
5. Independent Component Analysis

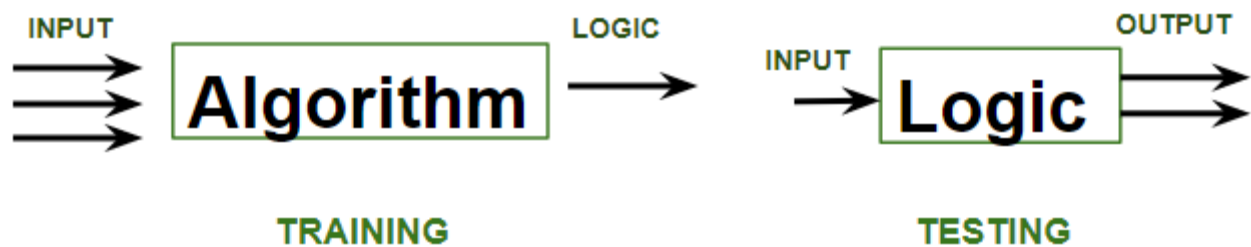
Supervised vs. Unsupervised Machine Learning

Parameters	Supervised machine learning	Unsupervised machine learning
Input Data	Algorithms are trained using labeled data.	Algorithms are used against data that is not labeled
Computational Complexity	Simpler method	Computationally complex
Accuracy	Highly accurate	Less accurate

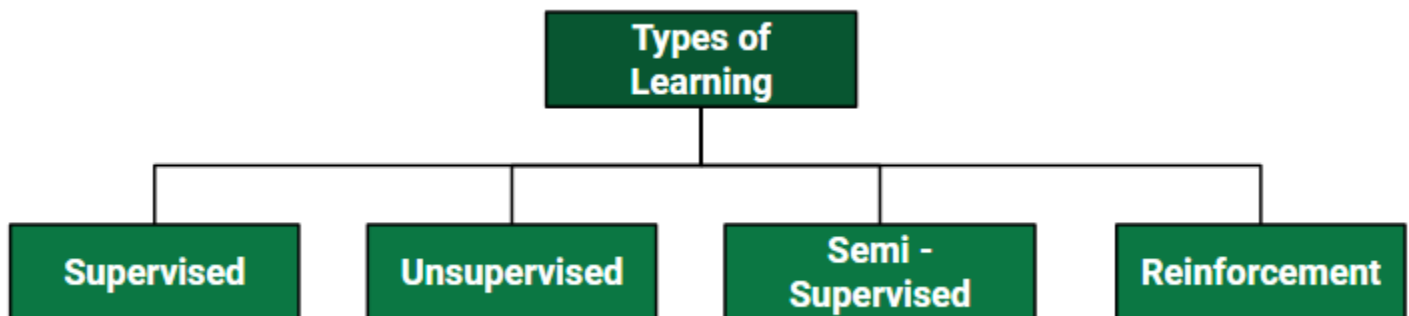
No. of classes	No. of classes is known	No. of classes is not known
Data Analysis	Uses offline analysis	Uses real-time analysis of data
Algorithms used	Linear and Logistics regression, Random forest, Support Vector Machine, Neural Network, etc.	K-Means clustering, Hierarchical clustering, Apriori algorithm, etc.

Types of Learning – Supervised Learning

Let us discuss what is learning for a machine is as shown below media as follows:



A machine is said to be learning from **past Experiences**(data feed-in) with respect to some class of **tasks** if its **Performance** in a given Task improves with the Experience. For example, assume that a machine has to predict whether a customer will buy a specific product let’s say “Antivirus” this year or not. The machine will do it by looking at the **previous knowledge/past experiences** i.e the data of products that the customer had bought every year and if he buys Antivirus every year, then there is a high probability that the customer is going to buy an antivirus this year as well. This is how machine learning works at the basic conceptual level.



Supervised learning is when the model is getting trained on a labelled dataset. A **labelled** dataset is one that has both input and output parameters. In this type of learning both training and validation, datasets are labelled as shown in the figures below.

User ID	Gender	Age	Salary	Purchased	Temperature	Pressure	Relative Humidity	Wind Direction	Wind Speed
15624510	Male	19	19000	0	10.69261758	986.882019	54.19337313	195.7150879	3.278597116
15810944	Male	35	20000	1	13.59184184	987.8729248	48.0648859	189.2951202	2.909167767
15668575	Female	26	43000	0	17.70494885	988.1119385	39.11965597	192.9273834	2.973036289
15603246	Female	27	57000	0	20.95430404	987.8500366	30.66273218	202.0752869	2.965289593
15804002	Male	19	76000	1	22.9278274	987.2833862	26.06723423	210.6589203	2.798230886
15728773	Male	27	58000	1	24.04233986	986.2907104	23.46918024	221.1188507	2.627005816
15598044	Female	27	84000	0	24.41475295	985.2338867	22.25082295	233.7911987	2.448749781
15694829	Female	32	150000	1	23.93361956	984.8914795	22.35178837	244.3504333	2.454271793
15600575	Male	25	33000	1	22.68800023	984.8461304	23.7538641	253.0864716	2.418341875
15727311	Female	35	65000	0	20.56425726	984.8380737	27.07867944	264.5071106	2.318677425
15570769	Female	26	80000	1	17.76400389	985.4262085	33.54900114	280.7827454	2.343950987
15606274	Female	26	52000	0	11.25680746	988.9386597	53.74139903	68.15406036	1.650191426
15746139	Male	20	86000	1	14.37810685	989.6819458	40.70884681	72.62069702	1.553469896
15704987	Male	32	18000	0	18.45114201	990.2960205	30.85038484	71.70604706	1.005017161
15628972	Male	18	82000	0	22.54895853	989.9562988	22.81738811	44.66042709	0.264133632
15697686	Male	29	80000	0	24.23155922	988.796875	19.74790765	318.3214111	0.329656571

Figure A: CLASSIFICATION

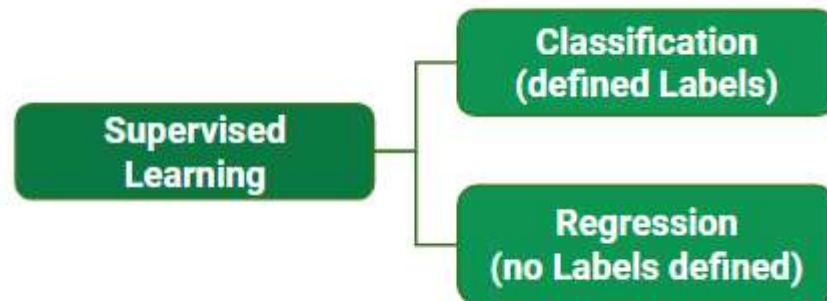
Figure B: REGRESSION

Both the above figures have labelled data set as follows:

- Figure A:** It is a dataset of a shopping store that is useful in predicting whether a customer will purchase a particular product under consideration or not based on his/ her gender, age, and salary.
Input: Gender, Age, Salary
Output: Purchased i.e. 0 or 1; 1 means yes the customer will purchase and 0 means that the customer won't purchase it.
- Figure B:** It is a Meteorological dataset that serves the purpose of predicting wind speed based on different parameters.
Input: Dew Point, Temperature, Pressure, Relative Humidity, Wind Direction
Output: Wind Speed

Training the system:

While training the model, data is usually split in the ratio of 80:20 i.e. 80% as training data and the rest as testing data. In training data, we feed input as well as output for 80% of data. The model learns from training data only. We use different machine learning algorithms(which we will discuss in detail in the next articles) to build our model. Learning means that the model will build some logic of its own. Once the model is ready then it is good to be tested. At the time of testing, the input is fed from the remaining 20% of data that the model has never seen before, the model will predict some value and we will compare it with the actual output and calculate the accuracy.



Types of Supervised Learning:

A. Classification: It is a Supervised Learning task where output is having defined labels(discrete value). For example in above Figure A, Output – Purchased has defined labels i.e. 0 or 1; 1 means the customer will purchase, and 0 means that the customer won't purchase. The goal here is to predict discrete values belonging to a particular class and evaluate them on the basis of accuracy. It can be either binary or multi-class classification. In **binary** classification, the model predicts either 0 or 1; yes or no but in the case of **multi-class** classification, the model predicts more than one class. **Example:** Gmail classifies mails in more than one class like social, promotions, updates, and forums.

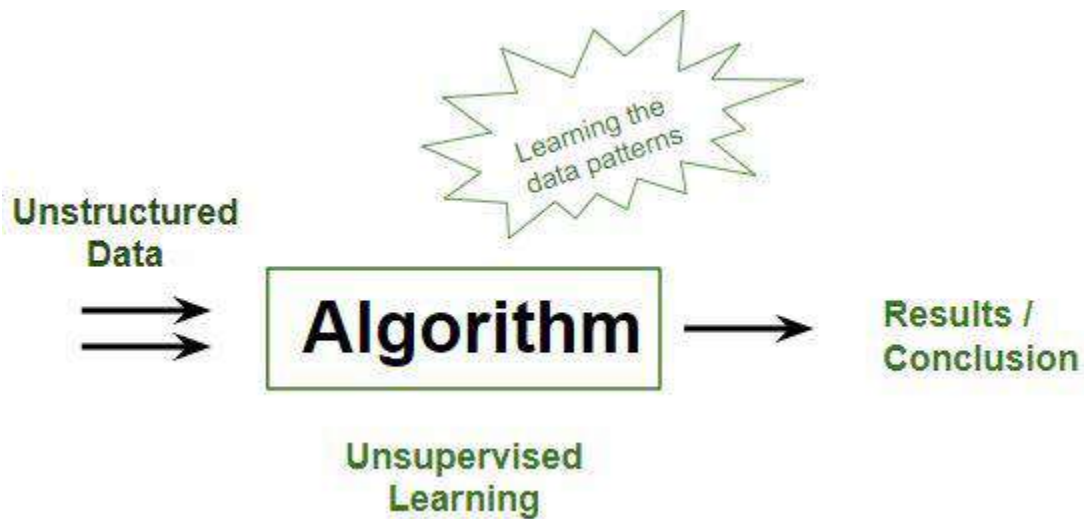
B. Regression: It is a Supervised Learning task where output is having continuous value. For example in above Figure B, Output – Wind Speed is not having any discrete value but is continuous in a particular range. The goal here is to predict a value as much closer to the actual output value as our model can and then evaluation is done by calculating the error value. The smaller the error the greater the accuracy of our regression model.

Example of Supervised Learning Algorithms:

- Linear Regression
- Logistic Regression
- Nearest Neighbor
- Gaussian Naive Bayes
- Decision Trees
- Support Vector Machine (SVM)
- Random Forest

Unsupervised Learning:

Or unsupervised machine learning analyzes and clusters unlabeled datasets using machine learning algorithms. These algorithms find hidden patterns and data without any human intervention, i.e., we don't give output to our model. The training model has only input parameter values and discovers the groups or patterns on its own. Data-set in Figure A is Mall data that contains information about its clients that subscribe to them. Once subscribed they are provided a membership card and the mall has complete information about the customer and his/her every purchase. Now using this data and unsupervised learning techniques, the mall can easily group clients based on the parameters we are feeding in.



CustomerID	Genre	Age	Annual Income (k\$)	Spending Score (1-100)
1	Male	19	15	39
2	Male	21	15	81
3	Female	20	16	6
4	Female	23	16	77
5	Female	31	17	40
6	Female	22	17	76
7	Female	35	18	6
8	Female	23	18	94
9	Male	64	19	3
10	Female	30	19	72
11	Male	67	19	14
12	Female	35	19	99
13	Female	58	20	15
14	Female	24	20	77
15	Male	37	20	13
16	Male	22	20	79
17	Female	35	21	35

Figure A

The input to the unsupervised learning models is as follows:

- **Unstructured data:** May contain noisy(meaningless) data, missing values, or unknown data
- **Unlabeled data:** Data only contains a value for input parameters, there is no targeted value(output). It is easy to collect as compared to the labeled one in the Supervised approach.

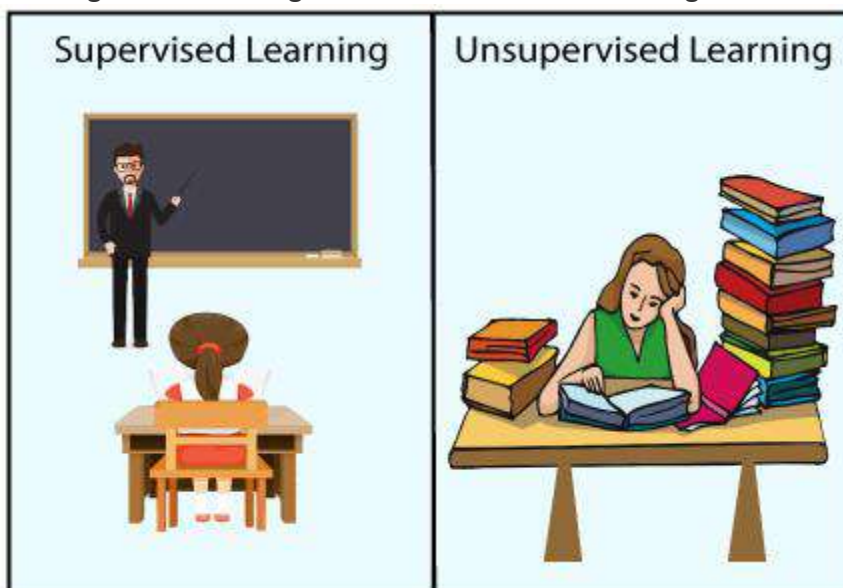


Types of Unsupervised Learning are as follows:

- **Clustering:** Broadly this technique is applied to group data based on different patterns, such as similarities or differences, our machine model finds. These algorithms are used to process raw, unclassified data objects into groups. For example, in the above figure, we have not given output parameter values, so this technique will be used to group clients based on the input parameters provided by our data.
- **Association:** This technique is a rule-based ML technique that finds out some very useful relations between parameters of a large data set. This technique is basically used for market basket analysis that helps to better understand the relationship between different products. For e.g. shopping stores use algorithms based on this technique to find out the relationship between the sale of one product w.r.t to another's sales based on customer behavior. Like if a customer buys milk, then he may also buy bread, eggs, or butter. Once trained well, such models can be used to increase their sales by planning different offers.

Some algorithms: *K-Means Clustering*

- *DBSCAN – Density-Based Spatial Clustering of Applications with Noise*
- *BIRCH – Balanced Iterative Reducing and Clustering using Hierarchies*
- *Hierarchical Clustering*
 - Supervised and Unsupervised learning are the two techniques of machine learning. But both the techniques are used in different scenarios and with different datasets. Below the explanation of both learning methods along with their difference table is given.



- **Supervised Machine Learning:**

- Supervised learning is a machine learning method in which models are trained using labeled data. In supervised learning, models need to find the mapping function to map the input variable (X) with the output variable (Y).

$$Y = f(X)$$

- Supervised learning needs supervision to train the model, which is similar to as a student learns things in the presence of a teacher. Supervised learning can be used for two types of problems: **Classification** and **Regression**.
- Example:** Suppose we have an image of different types of fruits. The task of our supervised learning model is to identify the fruits and classify them accordingly. So to identify the image in supervised learning, we will give the input data as well as output for that, which means we will train the model by the shape, size, color, and taste of each fruit. Once the training is completed, we will test the model by giving the new set of fruit. The model will identify the fruit and predict the output using a suitable algorithm.

Unsupervised Machine Learning:

- Unsupervised learning is another machine learning method in which patterns inferred from the unlabeled input data. The goal of unsupervised learning is to find the structure and patterns from the input data. Unsupervised learning does not need any supervision. Instead, it finds patterns from the data by its own.
- Learn more [Unsupervised Machine Learning](#)
- Unsupervised learning can be used for two types of problems: **Clustering** and **Association**.
- Example:** To understand the unsupervised learning, we will use the example given above. So unlike supervised learning, here we will not provide any supervision to the model. We will just provide the input dataset to the model and allow the model to find the patterns from the data. With the help of a suitable algorithm, the model will train itself and divide the fruits into different groups according to the most similar features between them.
- The main differences between Supervised and Unsupervised learning are given below:

Supervised Learning	Unsupervised Learning
Supervised learning algorithms are trained using labeled data.	Unsupervised learning algorithms are trained using unlabeled data.
Supervised learning model takes direct feedback to check if it is predicting correct output or not.	Unsupervised learning model does not take any feedback.
Supervised learning model predicts the output.	Unsupervised learning model finds the hidden patterns in data.
In supervised learning, input data is provided to the model along with the output.	In unsupervised learning, only input data is provided to the model.
The goal of supervised learning is to train the model so that it can predict the output when it is given new data.	The goal of unsupervised learning is to find the hidden patterns and useful insights from the unknown dataset.

Supervised learning needs supervision to train the model.	Unsupervised learning does not need any supervision to train the model.
Supervised learning can be categorized in Classification and Regression problems.	Unsupervised Learning can be classified in Clustering and Associations problems.
Supervised learning can be used for those cases where we know the input as well as corresponding outputs.	Unsupervised learning can be used for those cases where we have only input data and no corresponding output data.
Supervised learning model produces an accurate result.	Unsupervised learning model may give less accurate result as compared to supervised learning.
Supervised learning is not close to true Artificial intelligence as in this, we first train the model for each data, and then only it can predict the correct output.	Unsupervised learning is more close to the true Artificial Intelligence as it learns similarly as a child learns daily routine things by his experiences.
It includes various algorithms such as Linear Regression, Logistic Regression, Support Vector Machine, Multi-class Classification, Decision tree, Bayesian Logic, etc.	It includes various algorithms such as Clustering, KNN, and Apriori algorithm.

Semi-supervised Learning:

As the name suggests, its working lies between Supervised and Unsupervised techniques. We use these techniques when we are dealing with data that is a little bit labeled and the rest large portion of it is unlabeled. We can use the unsupervised techniques to predict labels and then feed these labels to supervised techniques. This technique is mostly applicable in the case of image data sets where usually all images are not labeled.



Reinforcement Learning:

In this technique, the model keeps on increasing its performance using Reward Feedback to learn the behavior or pattern. These algorithms are specific to a particular problem e.g. Google Self Driving car, AlphaGo where a bot competes with humans and even itself to get better and better performers in Go

Game. Each time we feed in data, they learn and add the data to their knowledge which is training data. So, the more it learns the better it gets trained and hence experienced.

- Agents observe input.
- An agent performs an action by making some decisions.
- After its performance, an agent receives a reward and accordingly reinforces and the model stores in state-action pair of information.
- Temporal Difference (TD)
- Q-Learning
- Deep Adversarial Networks

UNIT - IV

Supervised Learning:

Regression: Linear Regression, multi linear regression, Polynomial Regression, logistic regression, Non-linear Regression, Model evaluation methods. **Classification:** – support vector machines (SVM) , Naïve Bayes classification

Regression Analysis in Machine learning

Regression analysis is a statistical method to model the relationship between a dependent (target) and independent (predictor) variables with one or more independent variables. More specifically, Regression analysis helps us to understand how the value of the dependent variable is changing corresponding to an independent variable when other independent variables are held fixed. It predicts continuous/real values such as **temperature, age, salary, price**, etc.

We can understand the concept of regression analysis using the below example:

Example: Suppose there is a marketing company A, who does various advertisement every year and get sales on that. The below list shows the advertisement made by the company in the last 5 years and the corresponding sales:

Advertisement	Sales
\$90	\$1000
\$120	\$1300
\$150	\$1800
\$100	\$1200
\$130	\$1380
\$200	??

Now, the company wants to do the advertisement of \$200 in the year 2019 **and wants to know the prediction about the sales for this year**. So to solve such type of prediction problems in machine learning, we need regression analysis.

Regression is a supervised learning technique which helps in finding the correlation between variables and enables us to predict the continuous output variable based on the one or more predictor variables. It is mainly used for **prediction, forecasting, time series modeling, and determining the causal-effect relationship between variables.**

In Regression, we plot a graph between the variables which best fits the given datapoints, using this plot, the machine learning model can make predictions about the data. In simple words, "**Regression shows a line or curve that passes through all the datapoints on target-predictor graph in such a way that the vertical distance between the datapoints and the regression line is minimum.**" The distance between datapoints and line tells whether a model has captured a strong relationship or not.

Some examples of regression can be as:

- Prediction of rain using temperature and other factors
- Determining Market trends
- Prediction of road accidents due to rash driving.

Terminologies Related to the Regression Analysis:

- **Dependent Variable:** The main factor in Regression analysis which we want to predict or understand is called the dependent variable. It is also called **target variable**.
- **Independent Variable:** The factors which affect the dependent variables or which are used to predict the values of the dependent variables are called independent variable, also called as a **predictor**.
- **Outliers:** Outlier is an observation which contains either very low value or very high value in comparison to other observed values. An outlier may hamper the result, so it should be avoided.
- **Multicollinearity:** If the independent variables are highly correlated with each other than other variables, then such condition is called Multicollinearity. It should not be present in the dataset, because it creates problem while ranking the most affecting variable.
- **Underfitting and Overfitting:** If our algorithm works well with the training dataset but not well with test dataset, then such problem is called **Overfitting**. And if our algorithm does not perform well even with training dataset, then such problem is called **underfitting**.

Why do we use Regression Analysis?

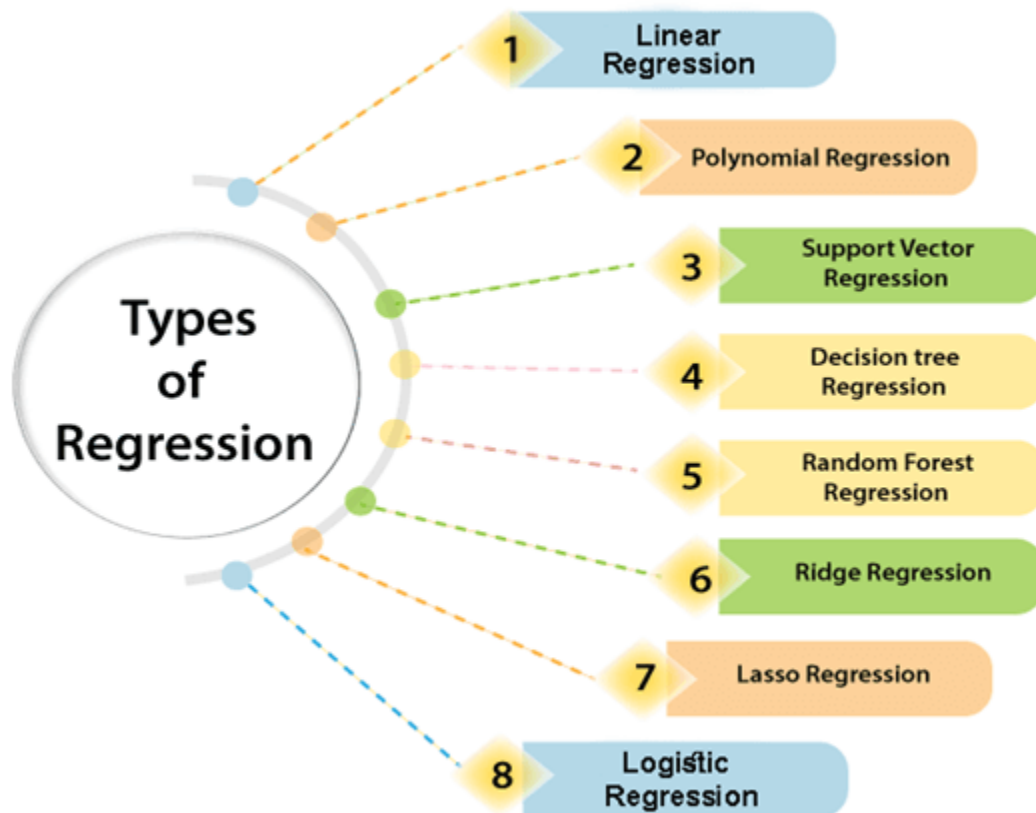
As mentioned above, Regression analysis helps in the prediction of a continuous variable. There are various scenarios in the real world where we need some future predictions such as weather condition, sales prediction, marketing trends, etc., for such case we need some technology which can make predictions more accurately. So for such case we need Regression analysis which is a statistical method and used in machine learning and data science. Below are some other reasons for using Regression analysis:

- Regression estimates the relationship between the target and the independent variable.
- It is used to find the trends in data.
- It helps to predict real/continuous values.
- By performing the regression, we can confidently determine the **most important factor, the least important factor, and how each factor is affecting the other factors.**

Types of Regression

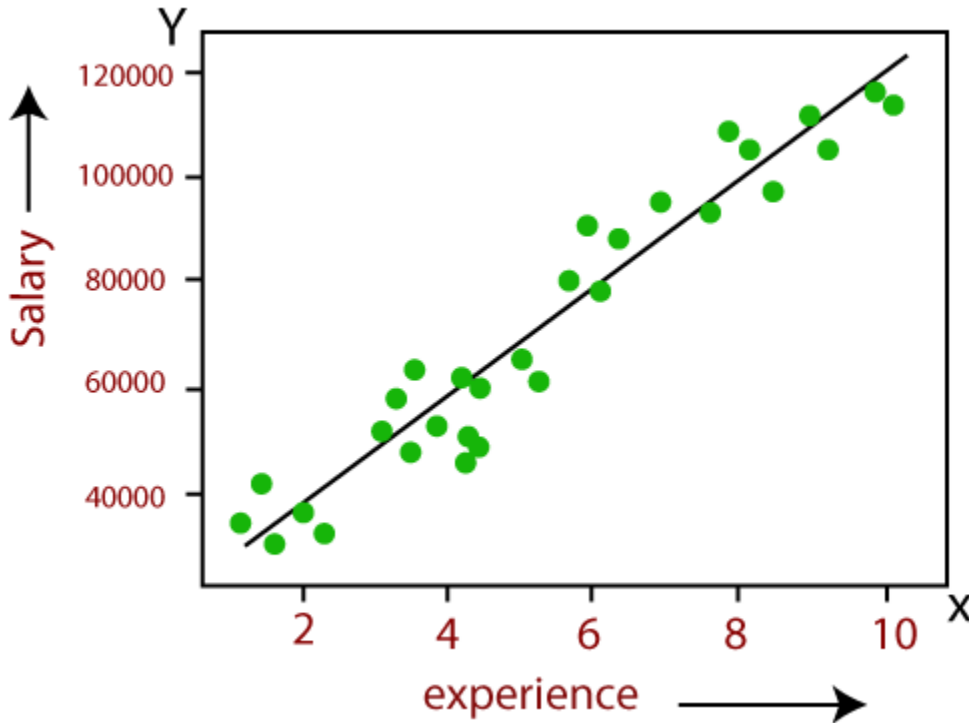
There are various types of regressions which are used in data science and machine learning. Each type has its own importance on different scenarios, but at the core, all the regression methods analyze the effect of the independent variable on dependent variables. Here we are discussing some important types of regression which are given below:

- **Linear Regression**
- **Logistic Regression**
- **Polynomial Regression**
- **Support Vector Regression**
- **Decision Tree Regression**
- **Random Forest Regression**
- **Ridge Regression**
- **Lasso Regression:**



Linear Regression:

- Linear regression is a statistical regression method which is used for predictive analysis.
- It is one of the very simple and easy algorithms which works on regression and shows the relationship between the continuous variables.
- It is used for solving the regression problem in machine learning.
- Linear regression shows the linear relationship between the independent variable (X-axis) and the dependent variable (Y-axis), hence called linear regression.
- If there is only one input variable (x), then such linear regression is called **simple linear regression**. And if there is more than one input variable, then such linear regression is called **multiple linear regression**.
- The relationship between variables in the linear regression model can be explained using the below image. Here we are predicting the salary of an employee on the basis of **the year of experience**.



- Below is the mathematical equation for Linear regression:

$$Y = aX + b$$

Here, Y = dependent variables (target variables),
 X = Independent variables (predictor variables),
 a and b are the linear coefficients

Some popular applications of linear regression are:

- Analyzing trends and sales estimates
- Salary forecasting
- Real estate prediction
- Arriving at ETAs in traffic.

Logistic Regression:

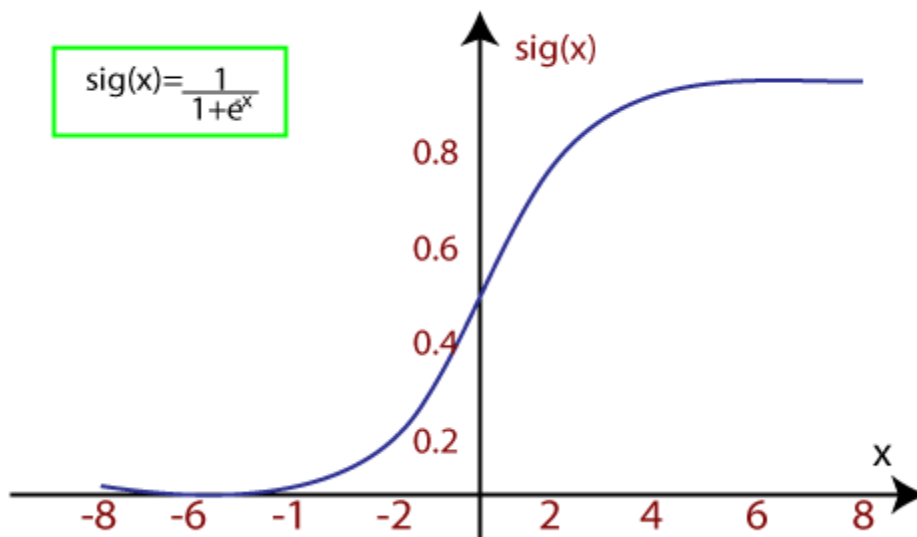
- Logistic regression is another supervised learning algorithm which is used to solve the classification problems. In **classification problems**, we have dependent variables in a binary or discrete format such as 0 or 1.
- Logistic regression algorithm works with the categorical variable such as 0 or 1, Yes or No, True or False, Spam or not spam, etc.
- It is a predictive analysis algorithm which works on the concept of probability.

- Logistic regression is a type of regression, but it is different from the linear regression algorithm in the term how they are used.
- Logistic regression uses **sigmoid function** or logistic function which is a complex cost function. This sigmoid function is used to model the data in logistic regression. The function can be represented as:

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

- $f(x)$ = Output between the 0 and 1 value.
- x = input to the function
- e = base of natural logarithm.

When we provide the input values (data) to the function, it gives the S-curve as follows:



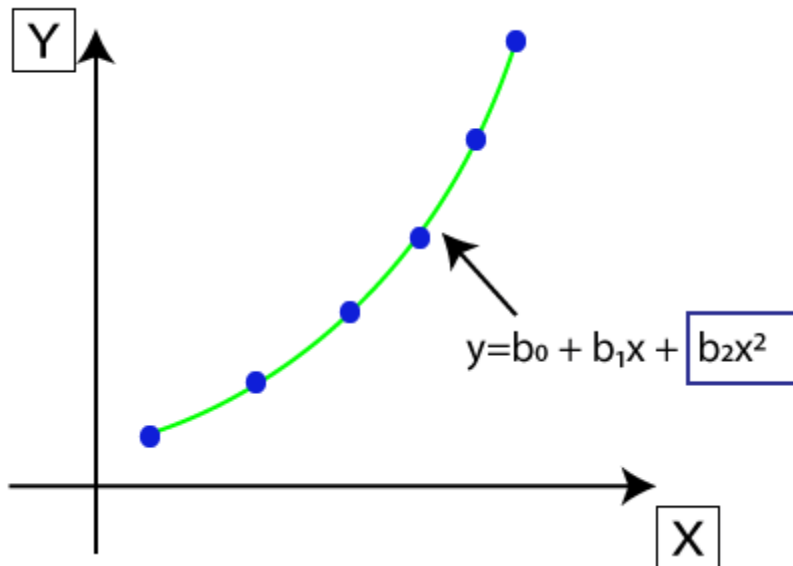
- It uses the concept of threshold levels, values above the threshold level are rounded up to 1, and values below the threshold level are rounded up to 0.

There are three types of logistic regression:

- **Binary(0/1, pass/fail)**
- **Multi(cats, dogs, lions)**
- **Ordinal(low, medium, high)**
-

Polynomial Regression:

- Polynomial Regression is a type of regression which models the **non-linear dataset** using a linear model.
- It is similar to multiple linear regression, but it fits a non-linear curve between the value of x and corresponding conditional values of y .
- Suppose there is a dataset which consists of datapoints which are present in a non-linear fashion, so for such case, linear regression will not best fit to those datapoints. To cover such datapoints, we need Polynomial regression.
- **In Polynomial regression, the original features are transformed into polynomial features of given degree and then modeled using a linear model.** Which means the datapoints are best fitted using a polynomial line.



- The equation for polynomial regression also derived from linear regression equation that means Linear regression equation $Y = b_0 + b_1x$, is transformed into Polynomial regression equation $Y = b_0 + b_1x + b_2x^2 + b_3x^3 + \dots + b_nx^n$.
- Here Y is the **predicted/target output**, b_0, b_1, \dots, b_n are the **regression coefficients**. x is our **independent/input variable**.
- The model is still linear as the coefficients are still linear with quadratic

Note: This is different from Multiple Linear regression in such a way that in Polynomial regression, a single element has different degrees instead of multiple variables with the same degree.

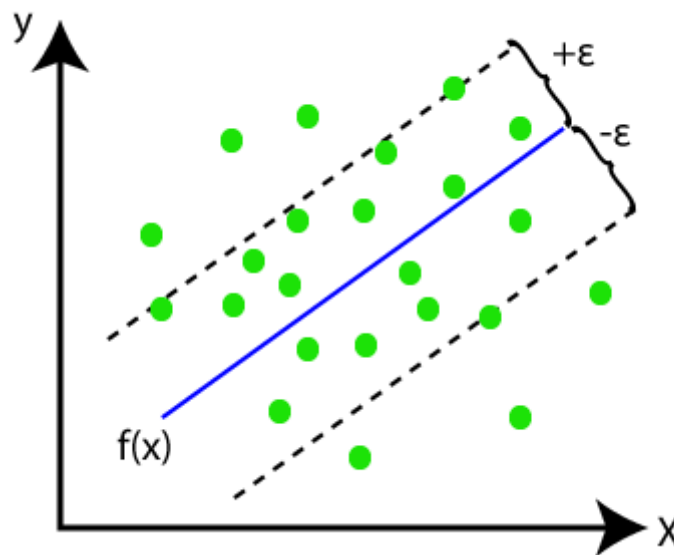
Support Vector Regression:

Support Vector Machine is a supervised learning algorithm which can be used for regression as well as classification problems. So if we use it for regression problems, then it is termed as Support Vector Regression.

Support Vector Regression is a regression algorithm which works for continuous variables. Below are some keywords which are used in **Support Vector Regression**:

- **Kernel:** It is a function used to map a lower-dimensional data into higher dimensional data.
- **Hyperplane:** In general SVM, it is a separation line between two classes, but in SVR, it is a line which helps to predict the continuous variables and cover most of the datapoints.
- **Boundary line:** Boundary lines are the two lines apart from hyperplane, which creates a margin for datapoints.
- **Support vectors:** Support vectors are the datapoints which are nearest to the hyperplane and opposite class.

In SVR, we always try to determine a hyperplane with a maximum margin, so that maximum number of datapoints are covered in that margin. *The main goal of SVR is to consider the maximum datapoints within the boundary lines and the hyperplane (best-fit line) must contain a maximum number of datapoints.* Consider the below image:

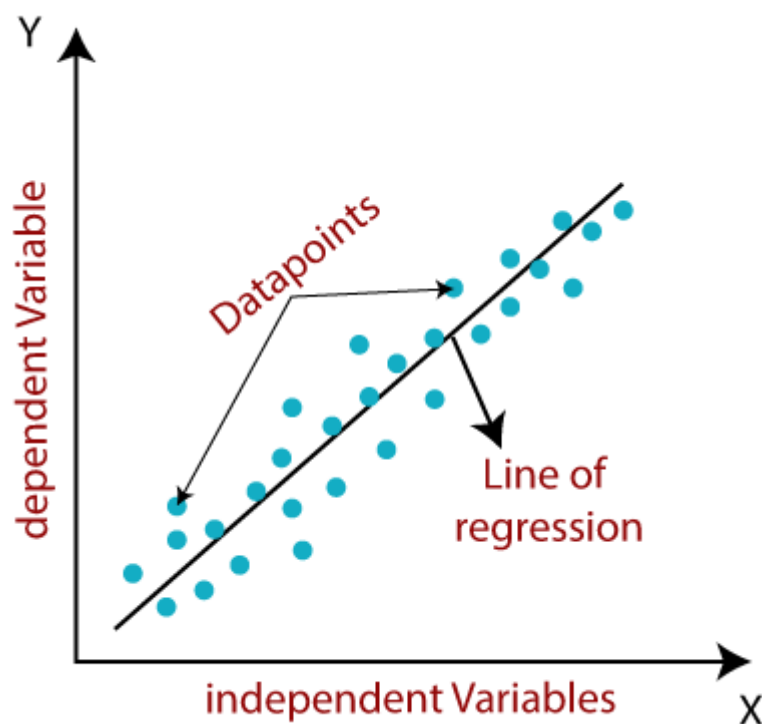


Here, the blue line is called hyperplane, and the other two lines are known as boundary lines.

Linear regression is one of the easiest and most popular Machine Learning algorithms. It is a statistical method that is used for predictive analysis. Linear regression makes predictions for continuous/real or numeric variables such as **sales, salary, age, product price**, etc.

Linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (x) variables, hence called as linear regression. Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.

The linear regression model provides a sloped straight line representing the relationship between the variables. Consider the below image:



$$y = a_0 + a_1x + \epsilon$$

Here,

Y=	Dependent	Variable	(Target	Variable)					
X=	Independent	Variable	(predictor	Variable)					
a0=	intercept	of the	line	(Gives an	additional	degree	of	freedom)	
a1	= Linear	regression	coefficient	(scale	factor	to	each	input	value).
ϵ	= random error								

The values for x and y variables are training datasets for Linear Regression model representation.

Types of Linear Regression

Linear regression can be further divided into two types of the algorithm:

- **Simple Linear Regression:**

If a single independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Simple Linear Regression.

- **Multiple Linear regression:**

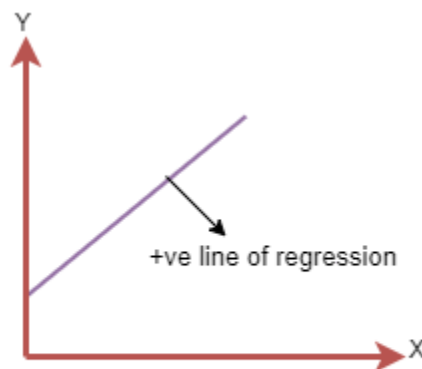
If more than one independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Multiple Linear Regression.

Linear Regression Line

A linear line showing the relationship between the dependent and independent variables is called a **regression line**. A regression line can show two types of relationship:

- **Positive Linear Relationship:**

If the dependent variable increases on the Y-axis and independent variable increases on X-axis, then such a relationship is termed as a Positive linear relationship.

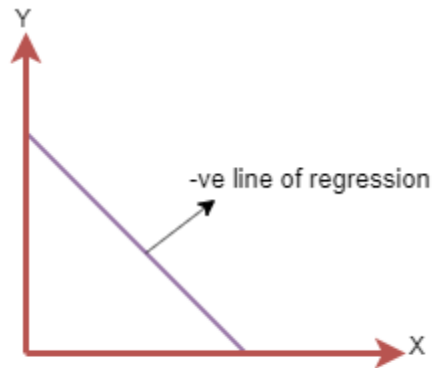


The line equation will be: $Y = a_0 + a_1X$

- **Negative Linear Relationship:**

○

If the dependent variable decreases on the Y-axis and independent variable increases on the X-axis, then such a relationship is called a negative linear relationship.



The line of equation will be: $Y = -a_0 + a_1X$

Finding the best fit line:

When working with linear regression, our main goal is to find the best fit line that means the error between predicted values and actual values should be minimized. The best fit line will have the least error.

The different values for weights or the coefficient of lines (a_0 , a_1) gives a different line of regression, so we need to calculate the best values for a_0 and a_1 to find the best fit line, so to calculate this we use cost function.

Cost function-

- The different values for weights or coefficient of lines (a_0 , a_1) gives the different line of regression, and the cost function is used to estimate the values of the coefficient for the best fit line.
- Cost function optimizes the regression coefficients or weights. It measures how a linear regression model is performing.
- We can use the cost function to find the accuracy of the **mapping function**, which maps the input variable to the output variable. This mapping function is also known as **Hypothesis function**.

For Linear Regression, we use the **Mean Squared Error (MSE)** cost function, which is the average of squared error occurred between the predicted values and actual values. It can be written as:

$$MSE = \frac{1}{N} \sum_{i=1}^n (y_i - (a_1x_i + a_0))^2$$

Where,

N=Total number of observation

Y_i = Actual value

$(a_1x_i+a_0)$ = Predicted value.

Residuals: The distance between the actual value and predicted values is called residual. If the observed points are far from the regression line, then the residual will be high, and so cost function will high. If the scatter points are close to the regression line, then the residual will be small and hence the cost function

UNIT- V

Unsupervised learning

Nearest neighbor models – K-means – clustering around medoids– silhouettes – hierarchical clustering – k-d trees ,Clustering trees – learning ordered rule lists – learning unordered rule .

Reinforcement learning- Example: Getting Lost -State and Action Spaces

K-Nearest Neighbor(KNN) Algorithm for Machine Learning

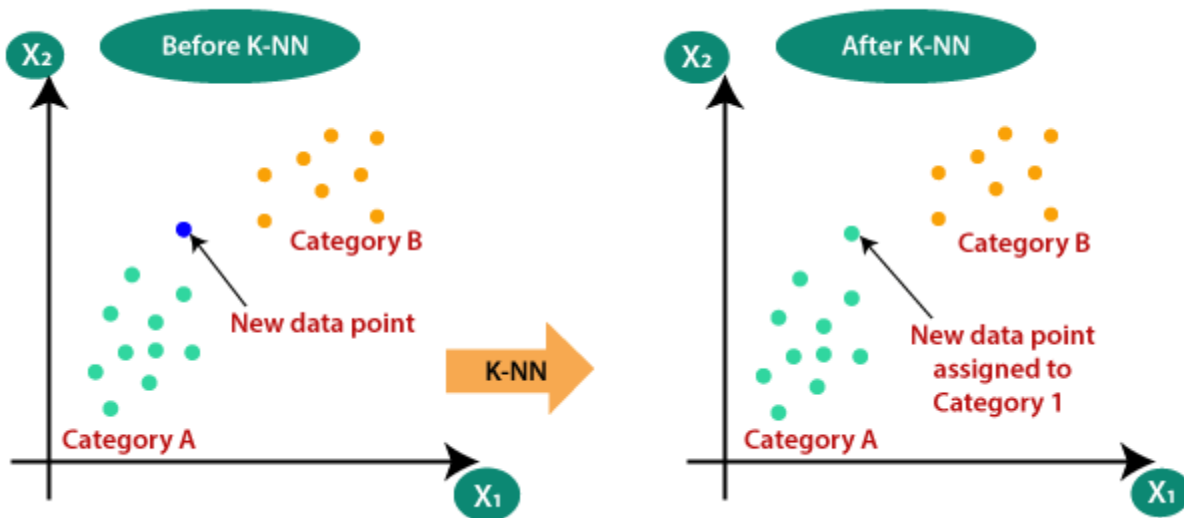
- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.
- It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.
- **Example:** Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.

KNN Classifier



Why do we need a K-NN Algorithm?

Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x_1 , so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:



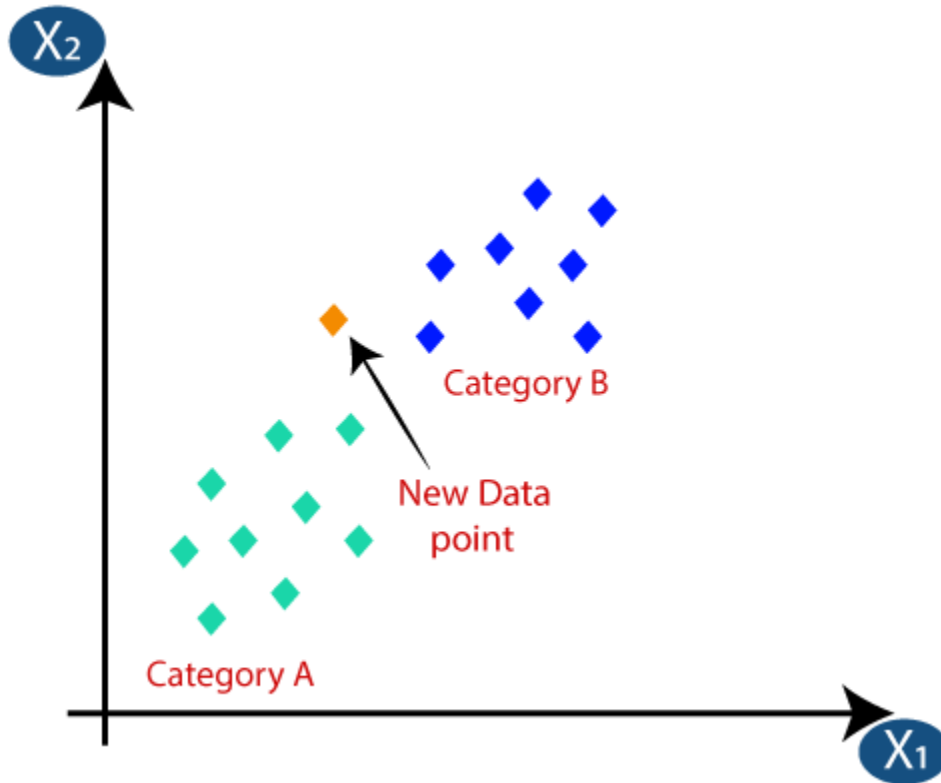
How does K-NN work?

The K-NN working can be explained on the basis of the below algorithm:

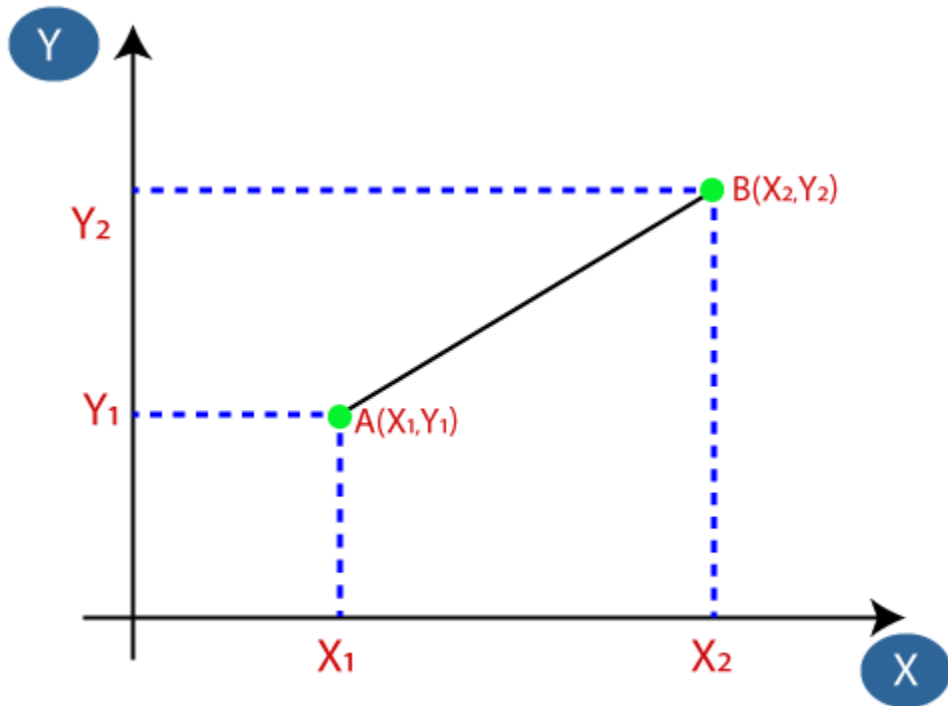
- **Step-1:** Select the number K of the neighbors
- **Step-2:** Calculate the Euclidean distance of K number of neighbors
- **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.

- **Step-4:** Among these k neighbors, count the number of the data points in each category.
- **Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.
- **Step-6:** Our model is ready.

Suppose we have a new data point and we need to put it in the required category. Consider the below image:



- firstly, we will choose the number of neighbors, so we will choose the $k=5$.
- Next, we will calculate the **Euclidean distance** between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:



$$\text{Euclidean Distance between } A_1 \text{ and } B_2 = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2}$$

- By calculating the Euclidean distance we got the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B. Consider the below image:



How to select the value of K in the K-NN Algorithm?

Below are some points to remember while selecting the value of K in the K-NN algorithm:

- As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A.
- There is no particular way to determine the best value for "K", so we need to try some values to find the best out of them. The most preferred value for K is 5.
- A very low value for K such as K=1 or K=2, can be noisy and lead to the effects of outliers in the model.
- Large values for K are good, but it may find some difficulties.

Advantages of KNN Algorithm:

- It is simple to implement.
- It is robust to the noisy training data
- It can be more effective if the training data is large.

Disadvantages of KNN Algorithm:

- Always needs to determine the value of K which may be complex some time.
- The computation cost is high because of calculating the distance between the data points for all the training samples.

K-Means Clustering Algorithm

K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science. In this topic, we will learn what is K-means clustering algorithm, how the algorithm works, along with the Python implementation of k-means clustering.

What is K-Means Algorithm?

K-Means Clustering is an [Unsupervised Learning algorithm](#), which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.

It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties.

It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.

It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters

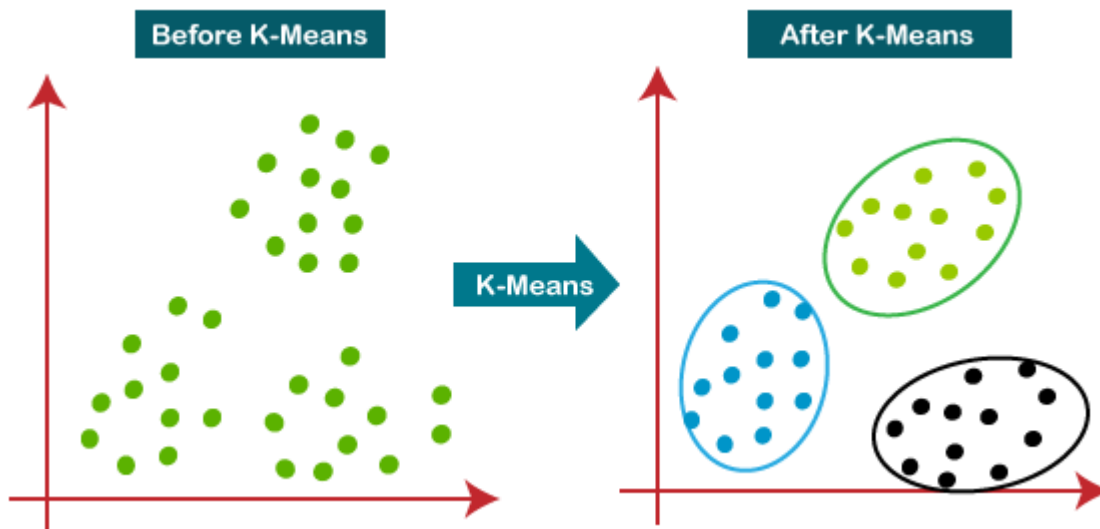
The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

The k-means clustering algorithm mainly performs two tasks:

- Determines the best value for K center points or centroids by an iterative process.
- Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

Hence each cluster has datapoints with some commonalities, and it is away from other clusters.

The below diagram explains the working of the K-means Clustering Algorithm:



How does the K-Means Algorithm Work?

The working of the K-Means algorithm is explained in the below steps:

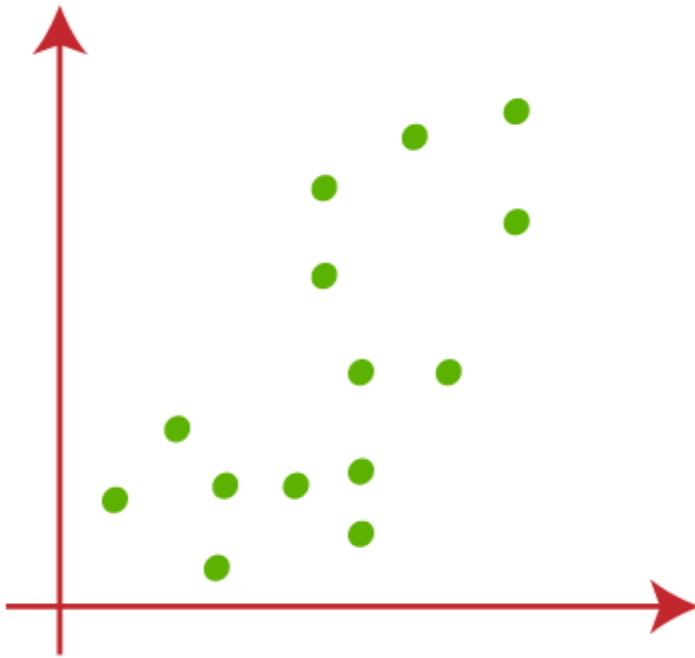
Step-1: Select the number K to decide the number of clusters.

Step-2: Select random K points or centroids. (It can be other from the input dataset).

Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.

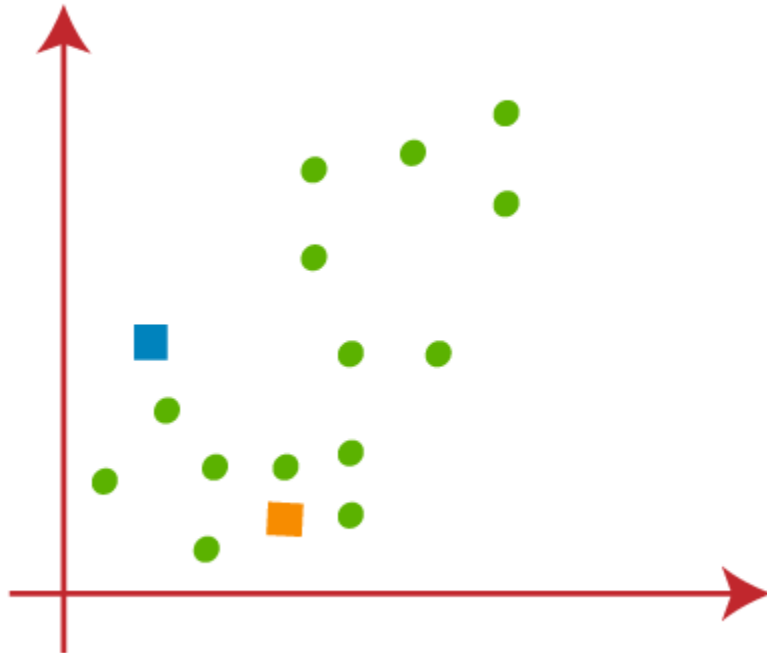
- **Step-4:** Calculate the variance and place a new centroid of each cluster.

- **Step-5:** Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.
- **Step-6:** If any reassignment occurs, then go to step-4 else go to FINISH.
- **Step-7:** The model is ready.
- Suppose we have two variables M1 and M2. The x-y axis scatter plot of these two variables is given below:

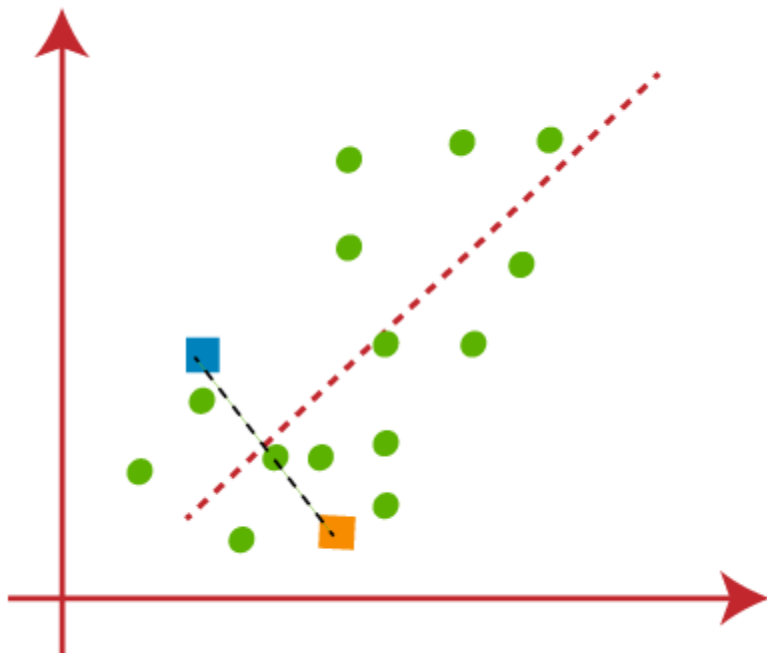


-
- Let's take number k of clusters, i.e., $K=2$, to identify the dataset and to put them into different clusters. It means here we will try to group these datasets into two different clusters.
- We need to choose some random k points or centroid to form the cluster. These points can be either the points from the dataset or any other point. So, here we are selecting the below two points as k

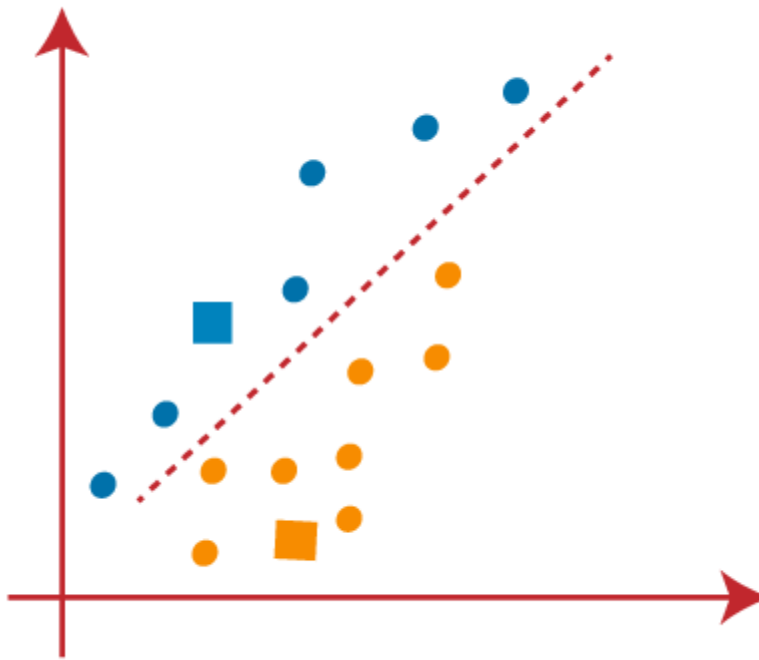
points, which are not the part of our dataset. Consider the below image:



- Now we will assign each data point of the scatter plot to its closest K-point or centroid. We will compute it by applying some mathematics that we have studied to calculate the distance between two points. So, we will draw a median between both the centroids. Consider the below image:

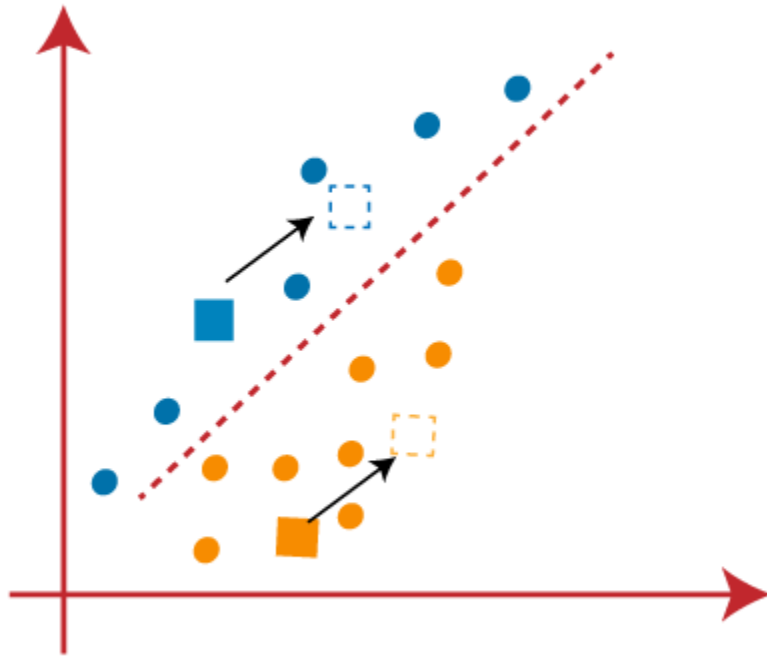


From the above image, it is clear that points left side of the line is near to the K1 or blue centroid, and points to the right of the line are close to the yellow centroid. Let's color them as blue and yellow for clear visualization.

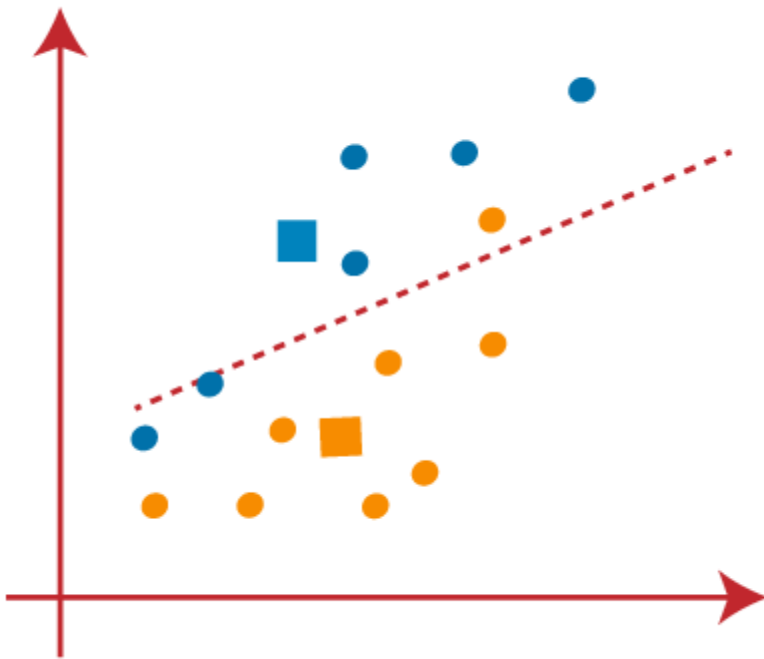


- As we need to find the closest cluster, so we will repeat the process by choosing **a new centroid**. To choose the new centroids, we will compute the center of gravity of these centroids, and will find new

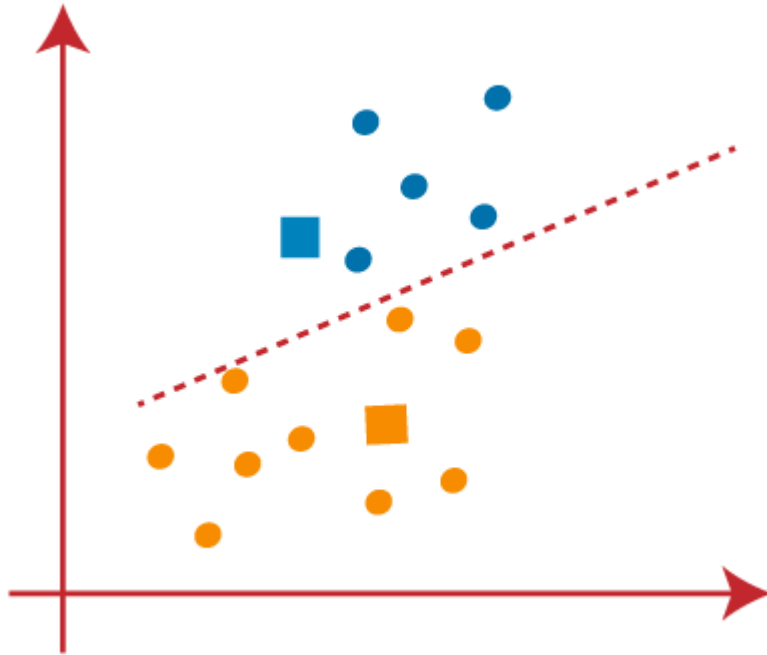
centroids as below:



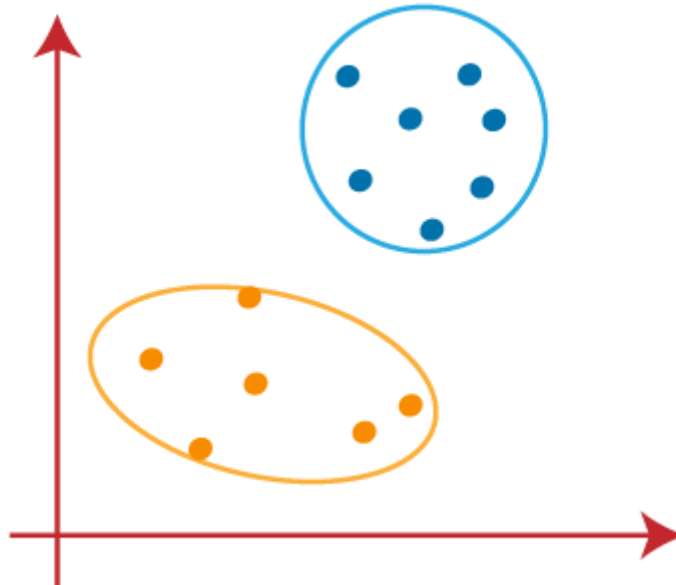
- Next, we will reassign each datapoint to the new centroid. For this, we will repeat the same process of finding a median line. The median will be like below image:



- From the above image, we can see, one yellow point is on the left side of the line, and two blue points are right to the line. So, these three points will be assigned to new centroids.



○



○

Hierarchical clustering is another unsupervised machine learning algorithm, which is used to group the unlabeled datasets into a cluster and also known as **hierarchical cluster analysis** or HCA.

In this algorithm, we develop the hierarchy of clusters in the form of a tree, and this tree-shaped structure is known as the **dendrogram**.

Sometimes the results of K-means clustering and hierarchical clustering may look similar, but they both differ depending on how they work. As there is no requirement to predetermine the number of clusters as we did in the K-Means algorithm.

The hierarchical clustering technique has two approaches:

1. **Agglomerative:** Agglomerative is a **bottom-up** approach, in which the algorithm starts with taking all data points as single clusters and merging them until one cluster is left.
2. **Divisive:** Divisive algorithm is the reverse of the agglomerative algorithm as it is a **top-down approach**.

Why hierarchical clustering?

As we already have other [clustering](#) algorithms such as [K-Means Clustering](#), then why we need hierarchical clustering? So, as we have seen in the K-means clustering that there are some challenges with this algorithm, which are a predetermined number of clusters, and it always tries to create the clusters of the same size. To solve these two challenges, we can opt for the hierarchical clustering algorithm because, in this algorithm, we don't need to have knowledge about the predefined number of clusters.

In this topic, we will discuss the Agglomerative Hierarchical clustering algorithm.

Agglomerative Hierarchical clustering

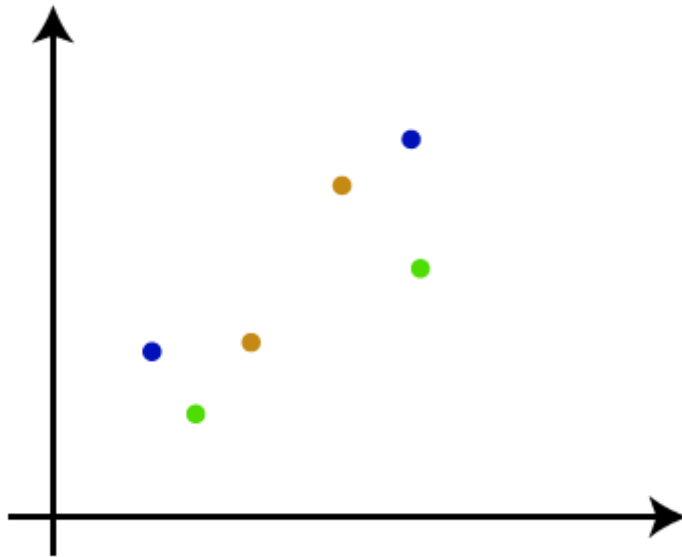
The agglomerative hierarchical clustering algorithm is a popular example of HCA. To group the datasets into clusters, it follows the **bottom-up approach**. It means, this algorithm considers each dataset as a single cluster at the beginning, and then start combining the closest pair of clusters together. It does this until all the clusters are merged into a single cluster that contains all the datasets.

This hierarchy of clusters is represented in the form of the dendrogram.

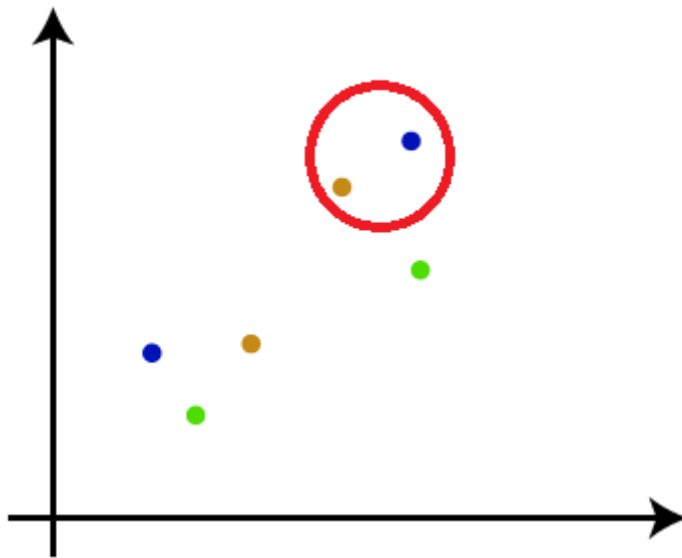
How the Agglomerative Hierarchical clustering Work?

The working of the AHC algorithm can be explained using the below steps:

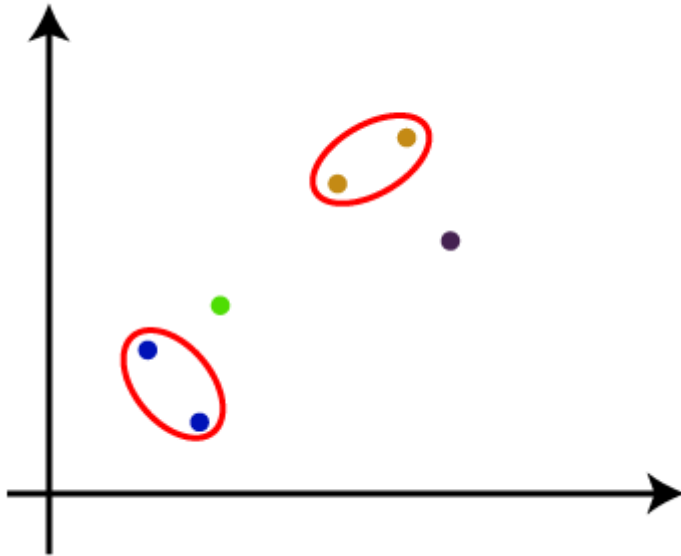
- **Step-1:** Create each data point as a single cluster. Let's say there are N data points, so the number of clusters will also be N .



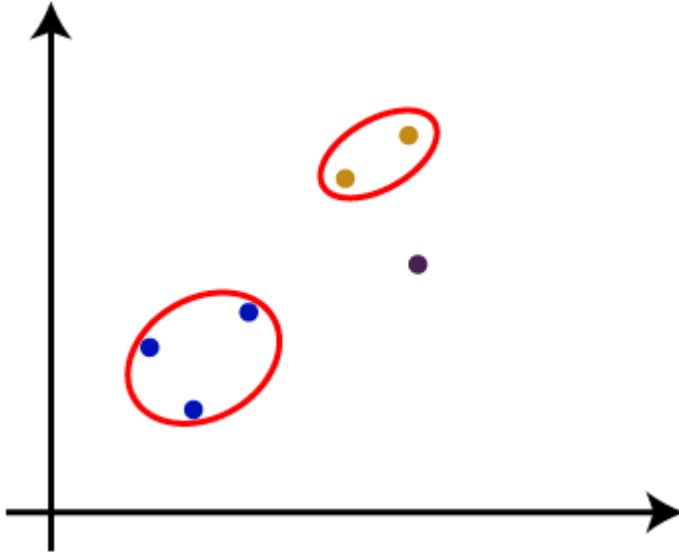
- **Step-2:** Take two closest data points or clusters and merge them to form one cluster. So, there will now be $N-1$ clusters.

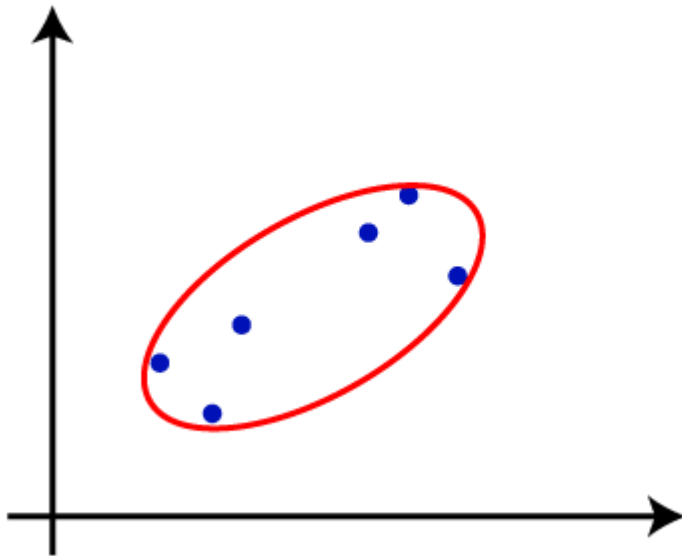
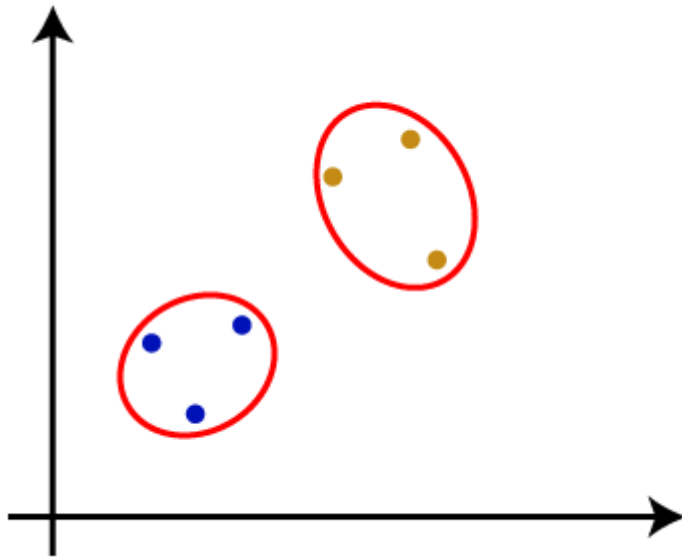


- **Step-3:** Again, take the two closest clusters and merge them together to form one cluster. There will be $N-2$ clusters.



- **Step-4:** Repeat Step 3 until only one cluster left. So, we will get the following clusters. Consider the below images:



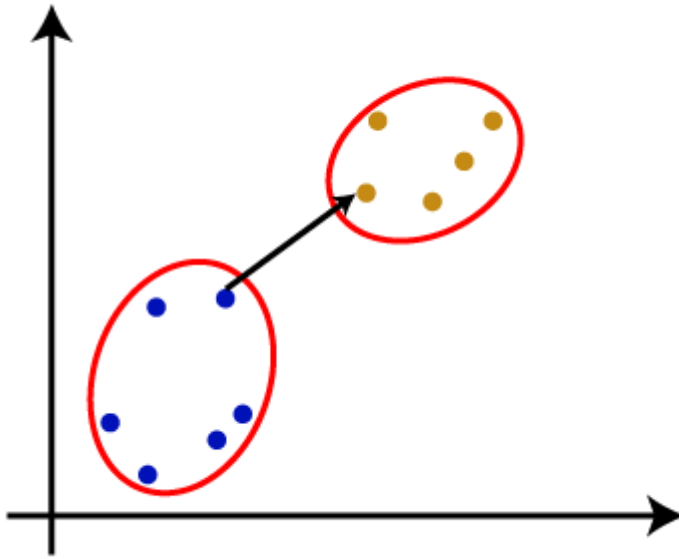


- **Step-5:** Once all the clusters are combined into one big cluster, develop the dendrogram to divide the clusters as per the problem.

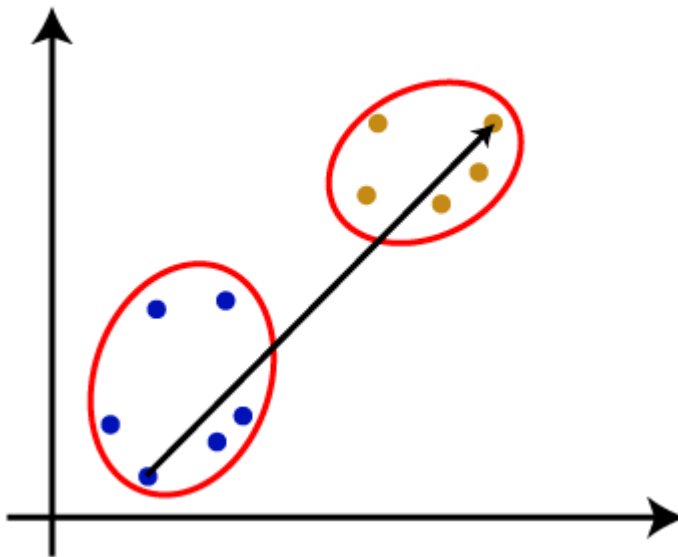
Measure for the distance between two clusters

As we have seen, the **closest distance** between the two clusters is crucial for the hierarchical clustering. There are various ways to calculate the distance between two clusters, and these ways decide the rule for clustering. These measures are called **Linkage methods**. Some of the popular linkage methods are given below:

1. **Single Linkage:** It is the Shortest Distance between the closest points of the clusters. Consider the below image:

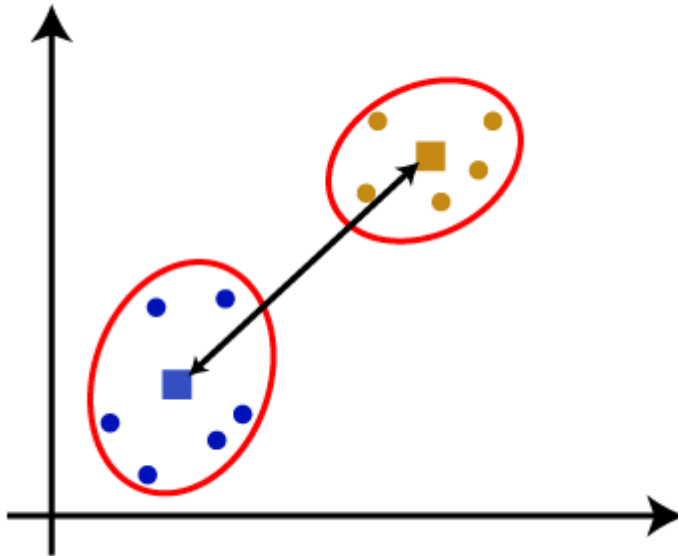


2. **Complete Linkage:** It is the farthest distance between the two points of two different clusters. It is one of the popular linkage methods as it forms tighter clusters than single-linkage.



3. **Average Linkage:** It is the linkage method in which the distance between each pair of datasets is added up and then divided by the total number of datasets to calculate the average distance between two clusters. It is also one of the most popular linkage methods.

4. **Centroid Linkage:** It is the linkage method in which the distance between the centroid of the clusters is calculated. Consider the below image:

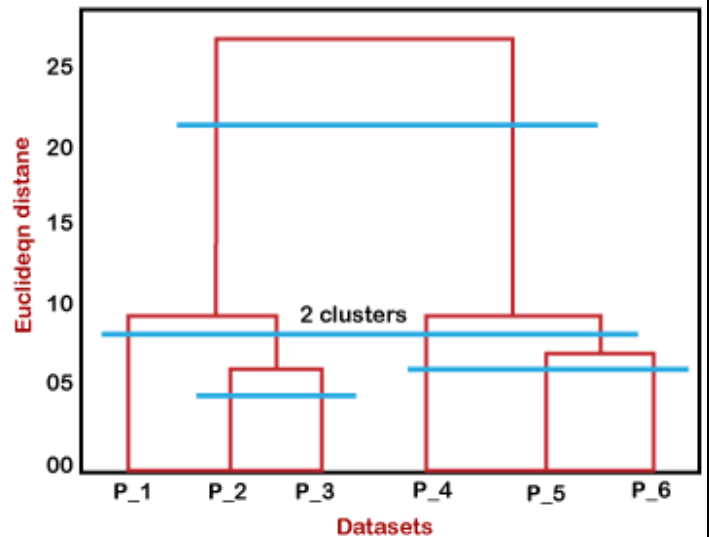
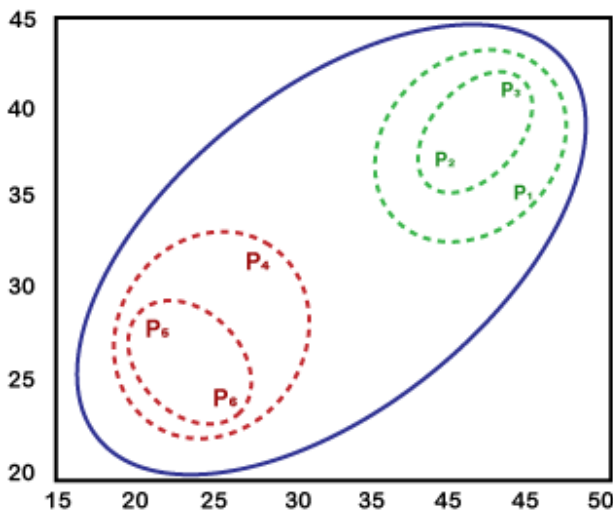


From the above-given approaches, we can apply any of them according to the type of problem or business requirement.

Working of Dendrogram in Hierarchical clustering

The dendrogram is a tree-like structure that is mainly used to store each step as a memory that the HC algorithm performs. In the dendrogram plot, the Y-axis shows the Euclidean distances between the data points, and the x-axis shows all the data points of the given dataset.

The working of the dendrogram can be explained using the below diagram:



In the above diagram, the left part is showing how clusters are created in agglomerative clustering, and the right part is showing the corresponding dendrogram.

- As we have discussed above, firstly, the datapoints P2 and P3 combine together and form a cluster, correspondingly a dendrogram is created, which connects P2 and P3 with a rectangular shape. The height is decided according to the Euclidean distance between the data points.
- In the next step, P5 and P6 form a cluster, and the corresponding dendrogram is created. It is higher than of previous, as the Euclidean distance between P5 and P6 is a little bit greater than the P2 and P3.
- Again, two new dendrograms are created that combine P1, P2, and P3 in one dendrogram, and P4, P5, and P6, in another dendrogram.
- At last, the final dendrogram is created that combines all the data points together.

Clustering or cluster analysis is a machine learning technique, which groups the unlabelled dataset. It can be defined as ***"A way of grouping the data points into different clusters, consisting of similar data points. The objects with the possible similarities remain in a group that has less or no similarities with another group."***

It does it by finding some similar patterns in the unlabelled dataset such as shape, size, color, behavior, etc., and divides them as per the presence and absence of those similar patterns.

It is an [unsupervised learning](#) method, hence no supervision is provided to the algorithm, and it deals with the unlabeled dataset.

The clustering technique is commonly used for **statistical data analysis**.

Note: Clustering is somewhere similar to the [classification algorithm](#), but the difference is the type of dataset that we are using. In classification, we work with the labeled data set, whereas in clustering, we work with the unlabelled dataset.

Example: Let's understand the clustering technique with the real-world example of Mall: When we visit any shopping mall, we can observe that the things with similar usage are grouped together. Such as the t-shirts are grouped in one section, and trousers are at other sections, similarly, at vegetable sections, apples, bananas, Mangoes, etc., are grouped in separate sections, so that we can easily find out the things. The clustering technique also works in the same way. Other examples of clustering are grouping documents according to the topic.

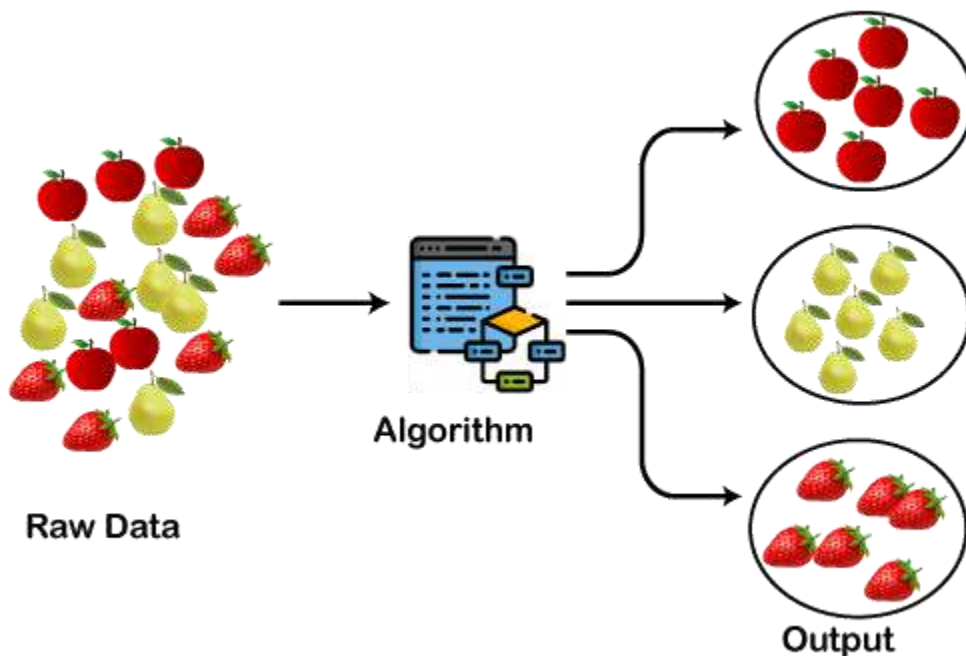
The clustering technique can be widely used in various tasks. Some most common uses of this technique are:

- Market Segmentation
- Statistical data analysis
- Social network analysis

- Image segmentation
- Anomaly detection, etc.

Apart from these general usages, it is used by the **Amazon** in its recommendation system to provide the recommendations as per the past search of products. **Netflix** also uses this technique to recommend the movies and web-series to its users as per the watch history.

The below diagram explains the working of the clustering algorithm. We can see the different fruits are divided into several groups with similar properties.



Types of Clustering Methods

The clustering methods are broadly divided into **Hard clustering** (datapoint belongs to only one group) and **Soft Clustering** (data points can belong to another group also). But there are also other various approaches of Clustering exist. Below are the main clustering methods used in Machine learning:

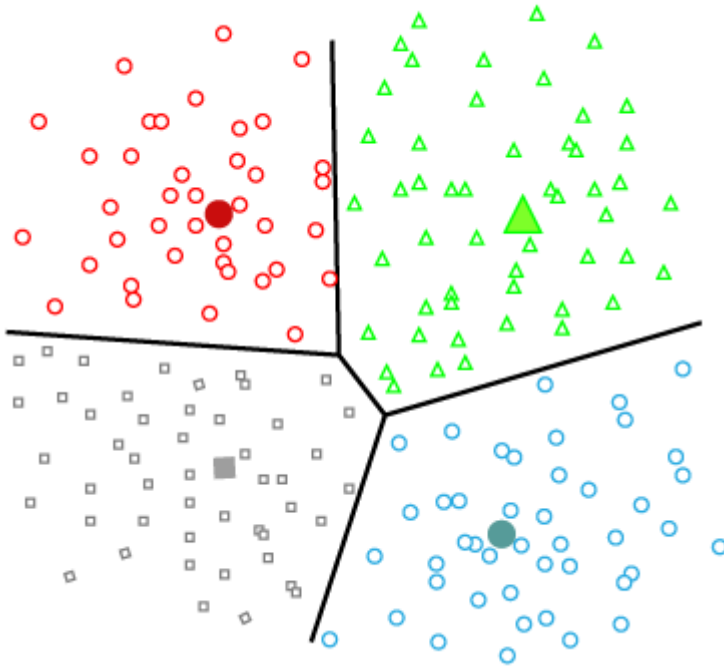
1. **Partitioning Clustering**
2. **Density-Based Clustering**
3. **Distribution Model-Based Clustering**
4. **Hierarchical Clustering**
5. **Fuzzy Clustering**

Partitioning Clustering

It is a type of clustering that divides the data into non-hierarchical groups. It is also known as

the **centroid-based method**. The most common example of partitioning clustering is the **K-Means Clustering algorithm**.

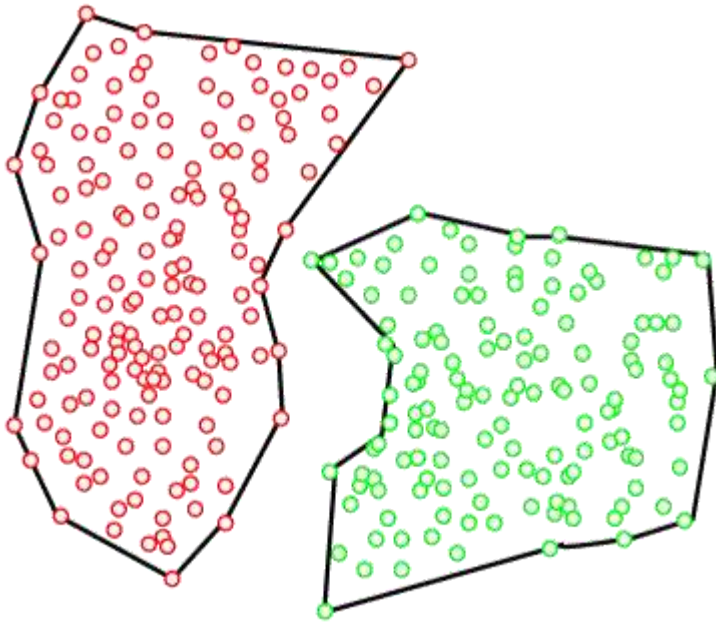
In this type, the dataset is divided into a set of k groups, where K is used to define the number of pre-defined groups. The cluster center is created in such a way that the distance between the data points of one cluster is minimum as compared to another cluster centroid.



Density-Based Clustering

The density-based clustering method connects the highly-dense areas into clusters, and the arbitrarily shaped distributions are formed as long as the dense region can be connected. This algorithm does it by identifying different clusters in the dataset and connects the areas of high densities into clusters. The dense areas in data space are divided from each other by sparser areas.

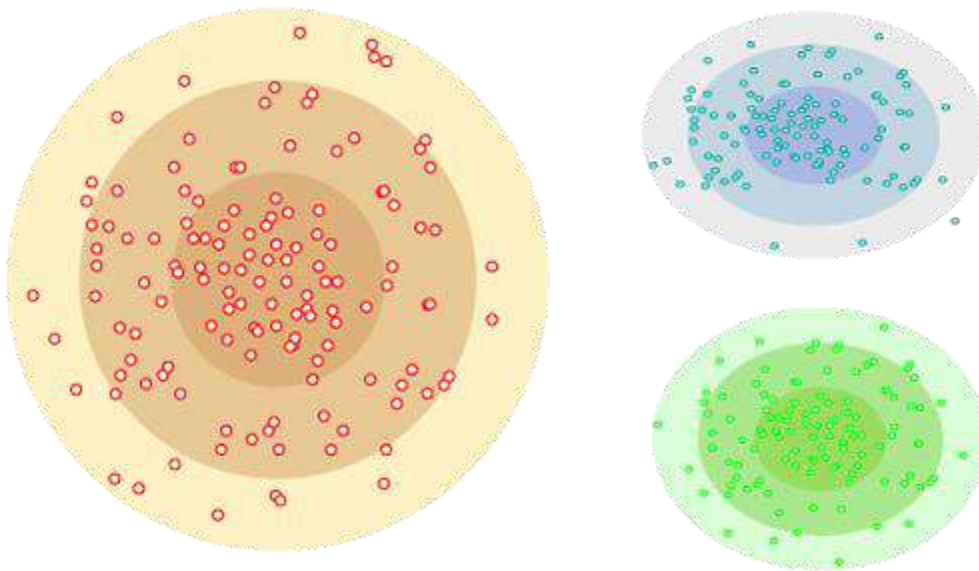
These algorithms can face difficulty in clustering the data points if the dataset has varying densities and high dimensions.



Distribution Model-Based Clustering

In the distribution model-based clustering method, the data is divided based on the probability of how a dataset belongs to a particular distribution. The grouping is done by assuming some distributions commonly **Gaussian Distribution**.

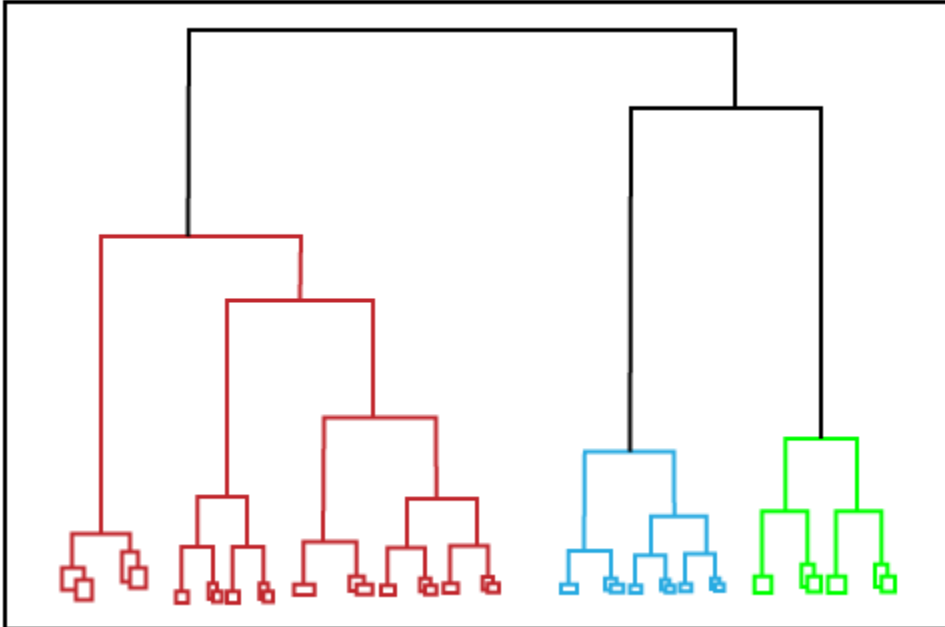
The example of this type is the **Expectation-Maximization Clustering algorithm** that uses Gaussian Mixture Models (GMM).



Hierarchical Clustering

Hierarchical clustering can be used as an alternative for the partitioned clustering as there is no

requirement of pre-specifying the number of clusters to be created. In this technique, the dataset is divided into clusters to create a tree-like structure, which is also called a **dendrogram**. The observations or any number of clusters can be selected by cutting the tree at the correct level. The most common example of this method is the **Agglomerative Hierarchical algorithm**.



Fuzzy Clustering

Fuzzy clustering is a type of soft method in which a data object may belong to more than one group or cluster. Each dataset has a set of membership coefficients, which depend on the degree of membership to be in a cluster. **Fuzzy C-means algorithm** is the example of this type of clustering; it is sometimes also known as the Fuzzy k-means algorithm.

Clustering Algorithms

The Clustering algorithms can be divided based on their models that are explained above. There are different types of clustering algorithms published, but only a few are commonly used. The clustering algorithm is based on the kind of data that we are using. Such as, some algorithms need to guess the number of clusters in the given dataset, whereas some are required to find the minimum distance between the observation of the dataset.

Here we are discussing mainly popular Clustering algorithms that are widely used in machine learning:

1. **K-Means algorithm:** The k-means algorithm is one of the most popular clustering algorithms. It classifies the dataset by dividing the samples into different clusters of equal variances. The number of clusters must be specified in this algorithm. It is fast with fewer computations required, with the linear complexity of $O(n)$.
2. **Mean-shift algorithm:** Mean-shift algorithm tries to find the dense areas in the smooth density of data

points. It is an example of a centroid-based model, that works on updating the candidates for centroid to be the center of the points within a given region.

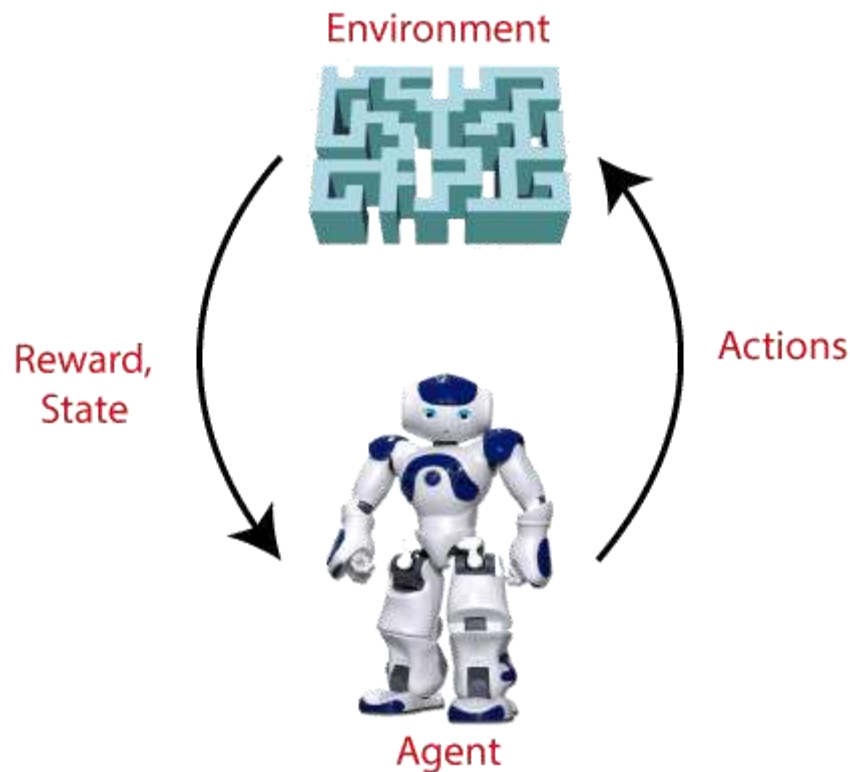
3. **DBSCAN Algorithm:** It stands for **Density-Based Spatial Clustering of Applications with Noise**. It is an example of a density-based model similar to the mean-shift, but with some remarkable advantages. In this algorithm, the areas of high density are separated by the areas of low density. Because of this, the clusters can be found in any arbitrary shape.
4. **Expectation-Maximization Clustering using GMM:** This algorithm can be used as an alternative for the k-means algorithm or for those cases where K-means can be failed. In GMM, it is assumed that the data points are Gaussian distributed.
5. **Agglomerative Hierarchical algorithm:** The Agglomerative hierarchical algorithm performs the bottom-up hierarchical clustering. In this, each data point is treated as a single cluster at the outset and then successively merged. The cluster hierarchy can be represented as a tree-structure.
6. **Affinity Propagation:** It is different from other clustering algorithms as it does not require to specify the number of clusters. In this, each data point sends a message between the pair of data points until convergence. It has $O(N^2T)$ time complexity, which is the main drawback of this algorithm.

Applications of Clustering

Below are some commonly known applications of clustering technique in Machine Learning:

- **In Identification of Cancer Cells:** The clustering algorithms are widely used for the identification of cancerous cells. It divides the cancerous and non-cancerous data sets into different groups.
- **In Search Engines:** Search engines also work on the clustering technique. The search result appears based on the closest object to the search query. It does it by grouping similar data objects in one group that is far from the other dissimilar objects. The accurate result of a query depends on the quality of the clustering algorithm used.
- **Customer Segmentation:** It is used in market research to segment the customers based on their choice and preferences.
- **In Biology:** It is used in the biology stream to classify different species of plants and animals using the image recognition technique.
- **In Land Use:** The clustering technique is used in identifying the area of similar lands use in the GIS database. This can be very useful to find that for what purpose the particular land should be used, that means for which purpose it is more suitable.

-
- Reinforcement Learning is a feedback-based Machine learning technique in which an agent learns to behave in an environment by performing the actions and seeing the results of actions. For each good action, the agent gets positive feedback, and for each bad action, the agent gets negative feedback or penalty.
 - In Reinforcement Learning, the agent learns automatically using feedbacks without any labeled data, unlike [supervised learning](#).
 - Since there is no labeled data, so the agent is bound to learn by its experience only.
 - RL solves a specific type of problem where decision making is sequential, and the goal is long-term, such as **game-playing, robotics**, etc.
 - The agent interacts with the environment and explores it by itself. The primary goal of an agent in reinforcement learning is to improve the performance by getting the maximum positive rewards.
 - The agent learns with the process of hit and trial, and based on the experience, it learns to perform the task in a better way. Hence, we can say that ***"Reinforcement learning is a type of machine learning method where an intelligent agent (computer program) interacts with the environment and learns to act within that."*** How a Robotic dog learns the movement of his arms is an example of Reinforcement learning.
 - It is a core part of [Artificial intelligence](#), and all [AI agent](#) works on the concept of reinforcement learning. Here we do not need to pre-program the agent, as it learns from its own experience without any human intervention.
 - **Example:** Suppose there is an AI agent present within a maze environment, and his goal is to find the diamond. The agent interacts with the environment by performing some actions, and based on those actions, the state of the agent gets changed, and it also receives a reward or penalty as feedback.
 - The agent continues doing these three things (**take action, change state/remain in the same state, and get feedback**), and by doing these actions, he learns and explores the environment.
 - The agent learns that what actions lead to positive feedback or rewards and what actions lead to negative feedback penalty. As a positive reward, the agent gets a positive point, and as a penalty, it gets a negative point.



Terms used in Reinforcement Learning

- **Agent():** An entity that can perceive/explore the environment and act upon it.
- **Environment():** A situation in which an agent is present or surrounded by. In RL, we assume the stochastic environment, which means it is random in nature.
- **Action():** Actions are the moves taken by an agent within the environment.
- **State():** State is a situation returned by the environment after each action taken by the agent.
- **Reward():** A feedback returned to the agent from the environment to evaluate the action of the agent.
- **Policy():** Policy is a strategy applied by the agent for the next action based on the current state.
- **Value():** It is expected long-term return with the discount factor and opposite to the short-term reward.
- **Q-value():** It is mostly similar to the value, but it takes one additional parameter as a current action (a).

Key Features of Reinforcement Learning

- In RL, the agent is not instructed about the environment and what actions need to be taken.
- It is based on the hit and trial process.
- The agent takes the next action and changes states according to the feedback of the previous action.
- The agent may get a delayed reward.

- The environment is stochastic, and the agent needs to explore it to reach to get the maximum positive rewards.

Approaches to implement Reinforcement Learning

There are mainly three ways to implement reinforcement-learning in ML, which are:

1. **Value-based:**

The value-based approach is about to find the optimal value function, which is the maximum value at a state under any policy. Therefore, the agent expects the long-term return at any state(s) under policy π .

2. **Policy-based:**

Policy-based approach is to find the optimal policy for the maximum future rewards without using the value function. In this approach, the agent tries to apply such a policy that the action performed in each step helps to maximize the future reward.

The policy-based approach has mainly two types of policy:

- **Deterministic:** The same action is produced by the policy (π) at any state.
- **Stochastic:** In this policy, probability determines the produced action.

3. **Model-based:** In the model-based approach, a virtual model is created for the environment, and the agent explores that environment to learn it. There is no particular solution or algorithm for this approach because the model representation is different for each environment.

Elements of Reinforcement Learning

There are four main elements of Reinforcement Learning, which are given below:

1. Policy
2. Reward Signal
3. Value Function
4. Model of the environment

1) Policy: A policy can be defined as a way how an agent behaves at a given time. It maps the perceived states of the environment to the actions taken on those states. A policy is the core element of the RL as it alone can define the behavior of the agent. In some cases, it may be a simple function or a lookup table, whereas, for other cases, it may involve general computation as a search process. It could be deterministic or a stochastic policy:

For **deterministic** policy: $a = \pi(s)$
For stochastic policy: $\pi(a | s) = P[A_t = a | S_t = s]$

2) Reward Signal: The goal of reinforcement learning is defined by the reward signal. At each state, the environment sends an immediate signal to the learning agent, and this signal is known as a **reward signal**. These rewards are given according to the good and bad actions taken by the agent. The agent's main objective is to maximize the total number of rewards for good actions. The reward signal can change the policy, such as if an action selected by the agent leads to low reward, then the policy may change to select other actions in the future.

3) Value Function: The value function gives information about how good the situation and action are and how much reward an agent can expect. A reward indicates the **immediate signal for each good and bad action**, whereas a value function specifies **the good state and action for the future**. The value function depends on the reward as, without reward, there could be no value. The goal of estimating values is to achieve more rewards.

4) Model: The last element of reinforcement learning is the model, which mimics the behavior of the environment. With the help of the model, one can make inferences about how the environment will behave. Such as, if a state and an action are given, then a model can predict the next state and reward.

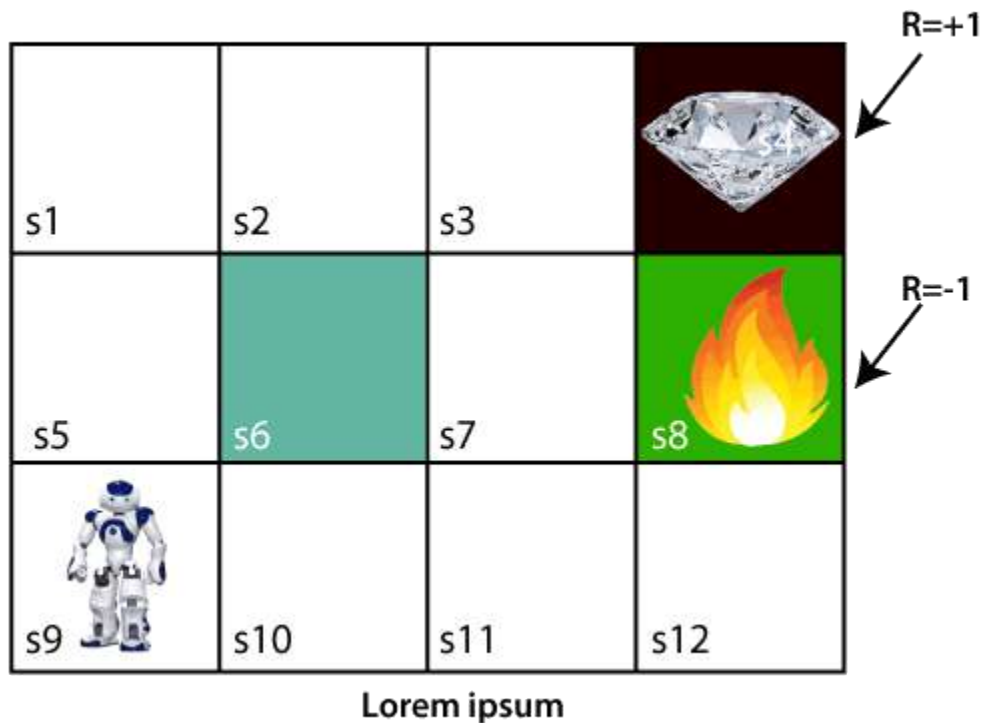
The model is used for planning, which means it provides a way to take a course of action by considering all future situations before actually experiencing those situations. The approaches for solving the RL problems **with the help of the model** are termed as the **model-based approach**. Comparatively, an approach **without using a model** is called a **model-free approach**.

How does Reinforcement Learning Work?

To understand the working process of the RL, we need to consider two main things:

- **Environment:** It can be anything such as a room, maze, football ground, etc.
- **Agent:** An intelligent agent such as AI robot.

Let's take an example of a maze environment that the agent needs to explore. Consider the below image:







In the above image, the agent is at the very first block of the maze. The maze is consisting of an S_6 block, which is a **wall**, S_8 a **fire pit**, and S_4 a **diamond block**.







The agent cannot cross the S_6 block, as it is a solid wall. If the agent reaches the S_4 block, then get the **+1 reward**; if it reaches the fire pit, then gets **-1 reward point**. It can take four actions: **move up, move down, move left, and move right**.

The agent can take any path to reach to the final point, but he needs to make it in possible fewer steps. Suppose the agent considers the path **S9-S5-S1-S2-S3**, so he will get the +1-reward point.

The agent will try to remember the preceding steps that it has taken to reach the final step. To memorize the steps, it assigns 1 value to each previous step. Consider the below step:

V=1 s1	V=1 s2	V=1 s3	 s4
V=1 s5		s7	 s8
 V=1 s9	s10	s11	s12

Now, the agent has successfully stored the previous steps assigning the 1 value to each previous block. But what will the agent do if he starts moving from the block, which has 1 value block on both sides? Consider the below diagram:

  V=1 s1	V=1 s2	V=1 s3	 s4
 V=1 s5		s7	 s8
V=1 s9	s10	s11	s12

It will be a difficult condition for the agent whether he should go up or down as each block has the same value. So, the above approach is not suitable for the agent to reach the destination. Hence to solve the problem, we will use the **Bellman equation**, which is the main concept behind reinforcement learning.

The Bellman Equation

The Bellman equation was introduced by the Mathematician **Richard Ernest Bellman in the year 1953**, and hence it is called as a Bellman equation. It is associated with dynamic programming and used to calculate the values of a decision problem at a certain point by including the values of previous states.

It is a way of calculating the value functions in dynamic programming or environment that leads to modern reinforcement learning.

The key-elements used in Bellman equations are:

- Action performed by the agent is referred to as "a"
- State occurred by performing the action is "s."
- The reward/feedback obtained for each good and bad action is "R."
- A discount factor is Gamma " γ ."

The Bellman equation can be written as:

$$V(s) = \max [R(s,a) + \gamma V(s')]$$

Where,

$V(s)$ = value calculated at a particular point.

$R(s,a)$ = Reward at a particular state s by performing an action.

γ = Discount factor

$V(s')$ = The value at the previous state.

In the above equation, we are taking the max of the complete values because the agent tries to find the optimal solution always.

So now, using the Bellman equation, we will find value at each state of the given environment. We will start from the block, which is next to the target block.

For 1st block:

- Action performed by the agent is referred to as "a"

- State occurred by performing the action is "s."
- The reward/feedback obtained for each good and bad action is "R."
- A discount factor is Gamma " γ ."

The Bellman equation can be written as:

$$V(s) = \max [R(s,a) + \gamma V(s')]$$

Where,

$V(s)$ = value calculated at a particular point.

$R(s,a)$ = Reward at a particular state s by performing an action.

γ = Discount factor

$V(s')$ = The value at the previous state.

In the above equation, we are taking the max of the complete values because the agent tries to find the optimal solution always.

So now, using the Bellman equation, we will find value at each state of the given environment. We will start from the block, which is next to the target block.

For 1st block:

$V(s_3) = \max [R(s,a) + \gamma V(s')]$, here $V(s') = 0$ because there is no further state to move.

$$V(s_3) = \max [R(s,a)] \Rightarrow V(s_3) = \max [1] \Rightarrow \mathbf{V(s_3) = 1.}$$

For 2nd block:

$V(s_2) = \max [R(s,a) + \gamma V(s')]$, here $\gamma = 0.9$ (lets), $V(s') = 1$, and $R(s, a) = 0$, because there is no reward at this state.

$$V(s_2) = \max [0.9(1)] \Rightarrow V(s_2) = \max [0.9] \Rightarrow \mathbf{V(s_2) = 0.9}$$

For 3rd block:

$V(s_1) = \max [R(s,a) + \gamma V(s')]$, here $\gamma = 0.9$ (lets), $V(s') = 0.9$, and $R(s, a) = 0$, because there is no reward at this state also.

$$V(s_1) = \max [0.9(0.9)] \Rightarrow V(s_1) = \max [0.81] \Rightarrow \mathbf{V(s_1) = 0.81}$$

For 4th block:

$V(s_5) = \max [R(s,a) + \gamma V(s')]$, here $\gamma = 0.9$ (lets), $V(s') = 0.81$, and $R(s, a) = 0$, because there is no reward at this state also.




$$V(s_5) = \max[0.9(0.81)] \Rightarrow V(s_5) = \max[0.81] \Rightarrow \mathbf{V(s_5) = 0.73}$$

For 5th block:





$V(s_9) = \max [R(s,a) + \gamma V(s')]$, here $\gamma = 0.9$ (lets), $V(s') = 0.73$, and $R(s, a) = 0$, because there is no reward at this state also.

$$V(s_9) = \max[0.9(0.73)] \Rightarrow V(s_9) = \max[0.66] \Rightarrow \mathbf{V(s_9) = 0.66}$$



Consider the below image:

V=0.81 s1	V=0.9 s2	V=1 s3	 s4
V=0.73 s5		s7	 s8
 V=0.66 s9	s10	s11	s12

Now, we will move further to the 6th block, and here agent may change the route because it always tries to find the optimal path. So now, let's consider from the block next to the fire pit.

$V=0.81$ s1	$V=0.9$ s2	$V=1$ s3	 s4
$V=0.73$ s5		 s7	 s8
$V=0.66$ s9	s10	s11	s12

Now, the agent has three options to move; if he moves to the blue box, then he will feel a bump if he moves to the fire pit, then he will get the -1 reward. But here we are taking only positive rewards, so for this, he will move to upwards only. The complete block values will be calculated using this formula. Consider the below image:

V=0.81 s1	V=0.9 s2	V=1 s3	 s4
V=0.73 s5	s6	V=0.9 s7	 s8
V=0.66 s9	V=0.73 s10	V=0.81 s11	V=0.73 s12

Types of Reinforcement learning

There are mainly two types of reinforcement learning, which are:

- **Positive Reinforcement**
- **Negative Reinforcement**

Positive Reinforcement:

The positive reinforcement learning means adding something to increase the tendency that expected behavior would occur again. It impacts positively on the behavior of the agent and increases the strength of the behavior.

This type of reinforcement can sustain the changes for a long time, but too much positive reinforcement may lead to an overload of states that can reduce the consequences.

Negative Reinforcement:

The negative reinforcement learning is opposite to the positive reinforcement as it increases the tendency that the specific behavior will occur again by avoiding the negative condition.

It can be more effective than the positive reinforcement depending on situation and behavior, but it provides reinforcement only to meet minimum behavior.

How to represent the agent state?

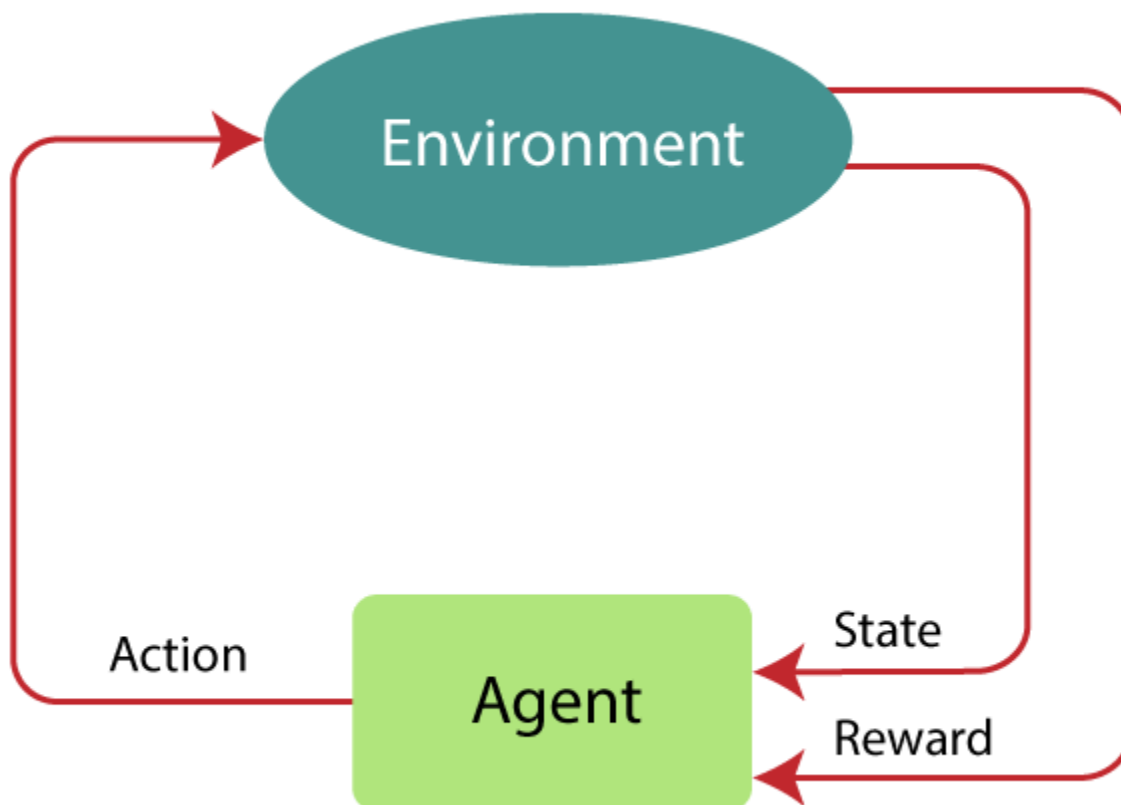
We can represent the agent state using the **Markov State** that contains all the required information from the history. The State S_t is Markov state if it follows the given condition:

$$P[S_{t+1} | S_t] = P[S_{t+1} | S_1, \dots, S_t]$$

The Markov state follows the **Markov property**, which says that the future is independent of the past and can only be defined with the present. The RL works on fully observable environments, where the agent can observe the environment and act for the new state. The complete process is known as Markov Decision process, which is explained below:

Markov Decision Process

Markov Decision Process or MDP, is used to **formalize the reinforcement learning problems**. If the environment is completely observable, then its dynamic can be modeled as a **Markov Process**. In MDP, the agent constantly interacts with the environment and performs actions; at each action, the environment responds and generates a new state.



MDP is used to describe the environment for the RL, and almost all the RL problem can be formalized using MDP.

MDP contains a tuple of four elements (S, A, P_a, R_a) :

- A set of finite States S
- A set of finite Actions A
- Rewards received after transitioning from state S to state S' , due to action a .
- Probability P_a .

MDP uses **Markov property**, and to better understand the MDP, we need to learn about it.

Markov Property:

It says that *"If the agent is present in the current state S_1 , performs an action a_1 and move to the state s_2 , then the state transition from s_1 to s_2 only depends on the current state and future action and states do not depend on past actions, rewards, or states."*

Or, in other words, as per Markov Property, the current state transition does not depend on any past action or state. Hence, MDP is an RL problem that satisfies the Markov property. Such as in a **Chess game, the players only focus on the current state and do not need to remember past actions or states.**

Finite MDP:

A finite MDP is when there are finite states, finite rewards, and finite actions. In RL, we consider only the finite MDP.

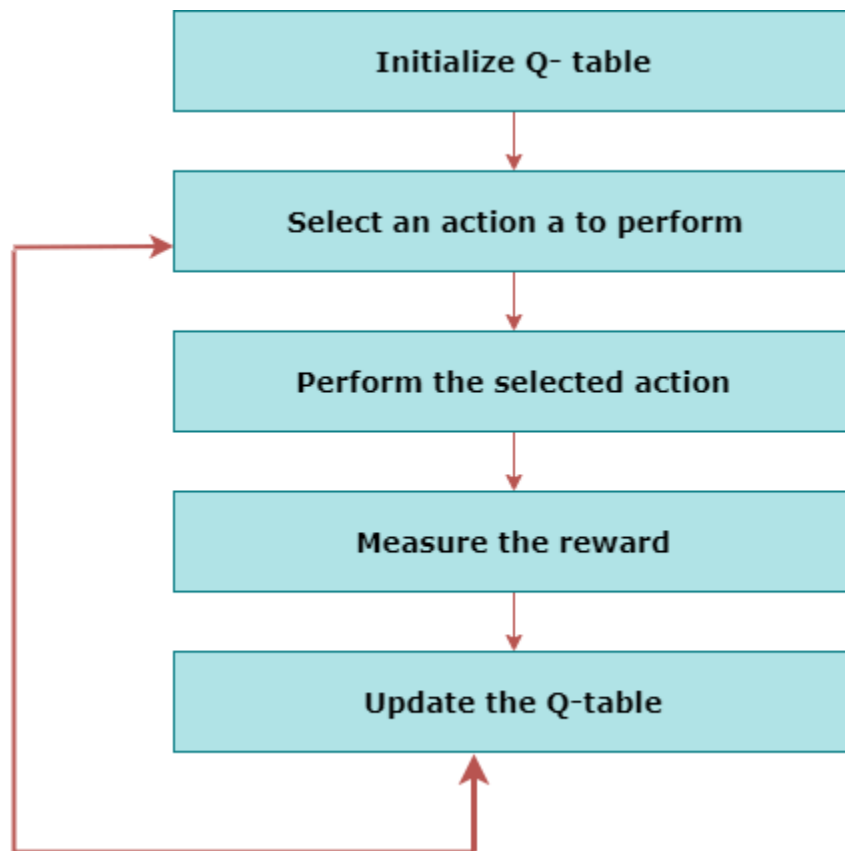
Markov Process:

Markov Process is a memoryless process with a sequence of random states S_1, S_2, \dots, S_t that uses the Markov Property. Markov process is also known as Markov chain, which is a tuple (S, P) on state S and transition function P . These two components (S and P) can define the dynamics of the system.

Reinforcement Learning Algorithms

Reinforcement learning algorithms are mainly used in AI applications and gaming applications. The main used algorithms are:

- **Q-Learning:**
 - Q-learning is an **Off policy RL algorithm**, which is used for the temporal difference Learning. The temporal difference learning methods are the way of comparing temporally successive predictions.
 - It learns the value function $Q(S, a)$, which means how good to take action "**a**" at a particular state "**s**."
 - The below flowchart explains the working of Q- learning:



- **State Action Reward State action (SARSA):**

- SARSA stands for **State Action Reward State action**, which is an **on-policy** temporal difference learning method. The on-policy control method selects the action for each state while learning using a specific policy.
- The goal of SARSA is to calculate the **$Q \pi (s, a)$ for the selected current policy π and all pairs of $(s-a)$.**
- The main difference between Q-learning and SARSA algorithms is that **unlike Q-learning, the maximum reward for the next state is not required for updating the Q-value in the table.**
- In SARSA, new action and reward are selected using the same policy, which has determined the original action.
- The SARSA is named because it uses the quintuple **$Q(s, a, r, s', a')$.**
- Where, **s: original state**
- **a: Original action**

- **r: reward observed while following the states**
s' and a': New state, action pair.

- **Deep Q Neural Network (DQN):**
 - As the name suggests, DQN is a **Q-learning using Neural networks**.
 - For a big state space environment, it will be a challenging and complex task to define and update a Q-table.
 - To solve such an issue, we can use a DQN algorithm. Where, instead of defining a Q-table, neural network approximates the Q-values for each action and state.

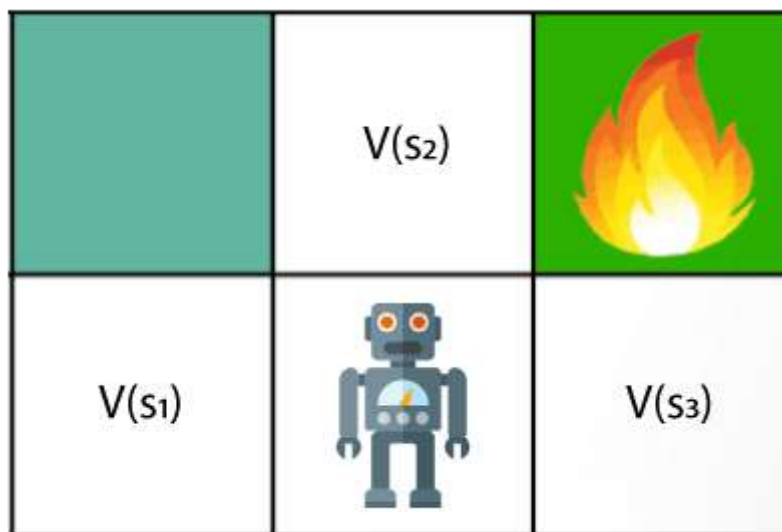
Now, we will expand the Q-learning.

Q-Learning Explanation:

- Q-learning is a popular model-free reinforcement learning algorithm based on the Bellman equation.
- **The main objective of Q-learning is to learn the policy which can inform the agent that what actions should be taken for maximizing the reward under what circumstances.**
- It is an **off-policy RL** that attempts to find the best action to take at a current state.
- The goal of the agent in Q-learning is to maximize the value of Q.
- The value of Q-learning can be derived from the Bellman equation. Consider the Bellman equation given below:

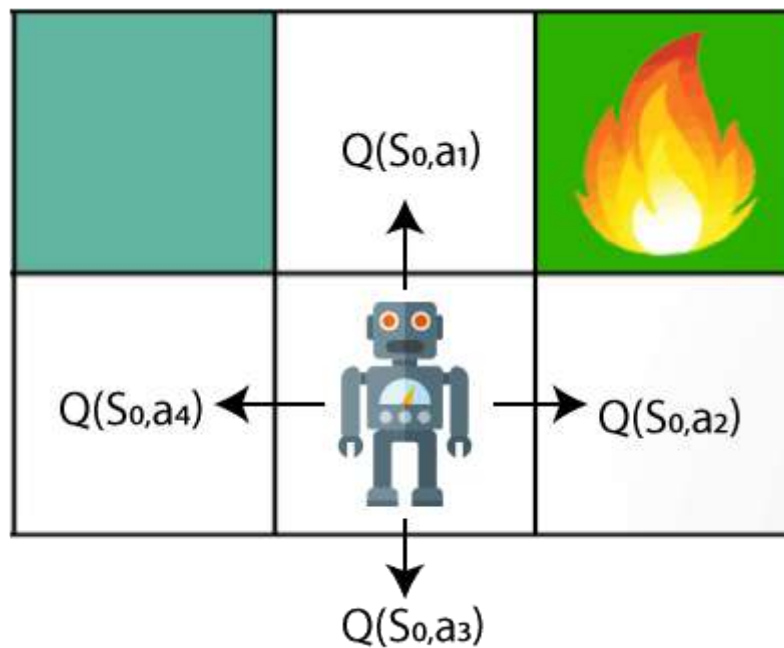
$$V(s) = \max [R(s,a) + \gamma \sum_{s'} P(s, a, s')V(s')]$$

In the equation, we have various components, including reward, discount factor (γ), probability, and end states s' . But there is no any Q-value is given so first consider the below image:



In the above image, we can see there is an agent who has three values options, $V(s_1)$, $V(s_2)$, $V(s_3)$. As this is MDP, so agent only cares for the current state and the future state. The agent can go to any direction (Up,

Left, or Right), so he needs to decide where to go for the optimal path. Here agent will take a move as per probability bases and changes the state. But if we want some exact moves, so for this, we need to make some changes in terms of Q-value. Consider the below image:



Q- represents the quality of the actions at each state. So instead of using a value at each state, we will use a pair of state and action, i.e., $Q(s, a)$. Q-value specifies that which action is more lubricative than others, and according to the best Q-value, the agent takes his next move. The Bellman equation can be used for deriving the Q-value.

To perform any action, the agent will get a reward $R(s, a)$, and also he will end up on a certain state, so the Q - value equation will be:

$$Q(S, a) = R(s, a) + \gamma \sum_{s'} P(s, a, s') V(s')$$

Hence, we can say that, $V(s) = \max [Q(s, a)]$

$$Q(S, a) = R(s, a) + \gamma \sum_{s'} (P(s, a, s') \max_{a'} Q(s', a'))$$

The above formula is used to estimate the Q-values in Q-Learning.

What is 'Q' in Q-learning?

The Q stands for **quality** in **Q-learning**, which means it specifies the quality of an action taken by the agent.

Q-table:

A Q-table or matrix is created while performing the Q-learning. The table follows the state and action pair, i.e., [s, a], and initializes the values to zero. After each action, the table is updated, and the q-values are stored within the table.

The RL agent uses this Q-table as a reference table to select the best action based on the q-values.

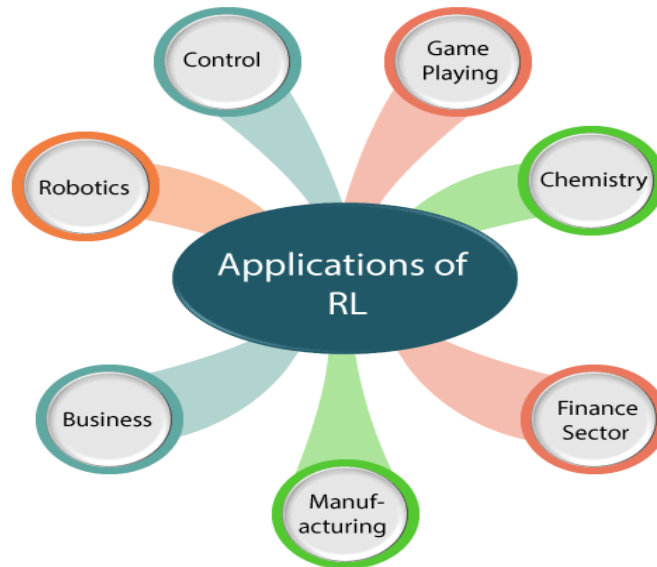
Difference between Reinforcement Learning and Supervised Learning

The Reinforcement Learning and Supervised Learning both are the part of machine learning, but both types of learnings are far opposite to each other. The RL agents interact with the environment, explore it, take action, and get rewarded. Whereas supervised learning algorithms learn from the labeled dataset and, on the basis of the training, predict the output.

The difference table between RL and Supervised learning is given below:

Reinforcement Learning	Supervised Learning
RL works by interacting with the environment.	Supervised learning works on the existing dataset.
The RL algorithm works like the human brain works when making some decisions.	Supervised Learning works as when a human learns things in the supervision of a guide.
There is no labeled dataset is present	The labeled dataset is present.
No previous training is provided to the learning agent.	Training is provided to the algorithm so that it can predict the output.
RL helps to take decisions sequentially.	In Supervised learning, decisions are made when input is given.

Reinforcement Learning Applications



1. **Robotics:**

- a. RL is used in **Robot navigation, Robo-soccer, walking, juggling**, etc.

2. **Control:**

- . RL can be used for **adaptive control** such as Factory processes, admission control in telecommunication, and Helicopter pilot is an example of reinforcement learning.

3. **Game Playing:**

- . RL can be used in **Game playing** such as tic-tac-toe, chess, etc.

4. **Chemistry:**

- . RL can be used for optimizing the chemical reactions.

5. **Business:**

- . RL is now used for business strategy planning.

6. **Manufacturing:**

In various automobile manufacturing companies, the robots use deep reinforcement learning to pick goods and put them in some containers.

7. **Finance Sector:**

The RL is currently used in the finance sector for evaluating trading strategies.

Conclusion:

From the above discussion, we can say that Reinforcement Learning is one of the most interesting and useful parts of Machine learning. In RL, the agent explores the environment by exploring it without any human intervention. It is the main learning algorithm that is used in Artificial Intelligence. But there are some cases where it should not be used, such as if you have enough data to solve the problem, then other ML algorithms can be used more efficiently. The main issue with the RL algorithm is that some of the parameters may affect the speed of the learning, such as delayed feedback.