

What is Artificial Intelligence?

In today's world, technology is growing very fast, and we are getting in touch with different new technologies day by day.

Here, one of the booming technologies of computer science is Artificial Intelligence which is ready to create a new revolution in the world by making intelligent machines. The Artificial Intelligence is now all around us. It is currently working with a variety of subfields, ranging from general to specific, such as self-driving cars, playing chess, proving theorems, playing music, Painting, etc.

AI is one of the fascinating and universal fields of Computer science which has a great scope in future. AI holds a tendency to cause a machine to work as a human.

Artificial Intelligence is composed of two words **Artificial** and **Intelligence**, where Artificial defines "*man-made*," and intelligence defines "*thinking power*", hence AI means "*a man-made thinking power*."

So, we can define AI as:

"It is a branch of computer science by which we can create intelligent machines which can behave like a human, think like humans, and able to make decisions."

Artificial Intelligence exists when a machine can have human based skills such as learning, reasoning, and solving problems

With Artificial Intelligence you do not need to preprogram a machine to do some work, despite that you can create a machine with programmed algorithms which can work with own intelligence, and that is the awesomeness of AI.

It is believed that AI is not a new technology, and some people says that as per Greek myth, there were Mechanical men in early days which can work and behave like humans.

Why Artificial Intelligence?

Before Learning about Artificial Intelligence, we should know that what is the importance of AI and why should we learn it. Following are some main reasons to learn about AI:

- With the help of AI, you can create such software or devices which can solve real-world problems very easily and with accuracy such as health issues, marketing, traffic issues, etc.

- With the help of AI, you can create your personal virtual Assistant, such as Cortana, Google Assistant, Siri, etc.
- With the help of AI, you can build such Robots which can work in an environment where survival of humans can be at risk.
- AI opens a path for other new technologies, new devices, and new Opportunities.

Goals of Artificial Intelligence

Following are the main goals of Artificial Intelligence:

1. Replicate human intelligence
2. Solve Knowledge-intensive tasks
3. An intelligent connection of perception and action
4. Building a machine which can perform tasks that requires human intelligence such as:
 - Proving a theorem
 - Playing chess
 - Plan some surgical operation
 - Driving a car in traffic
5. Creating some system which can exhibit intelligent behavior, learn new things by itself, demonstrate, explain, and can advise to its user.

What Comprises to Artificial Intelligence?

Artificial Intelligence is not just a part of computer science even it's so vast and requires lots of other factors which can contribute to it. To create the AI first we should know that how intelligence is composed, so the Intelligence is an intangible part of our brain which is a combination of **Reasoning, learning, problem-solving perception, language understanding, etc.** To achieve the above factors for a machine or software Artificial Intelligence requires the following discipline:

- Mathematics
- Biology
- Psychology
- Sociology
- Computer Science
- Neurons Study
- Statistics

Advantages of Artificial Intelligence

Following are some main advantages of Artificial Intelligence:

- **High Accuracy with less errors:** AI machines or systems are prone to less errors and high accuracy as it takes decisions as per pre-experience or information.
- **High-Speed:** AI systems can be of very high-speed and fast-decision making, because of that AI systems can beat a chess champion in the Chess game.
- **High reliability:** AI machines are highly reliable and can perform the same action multiple times with high accuracy.

- **Useful for risky areas:** AI machines can be helpful in situations such as defusing a bomb, exploring the ocean floor, where to employ a human can be risky.
- **Digital Assistant:** AI can be very useful to provide digital assistant to the users such as AI technology is currently used by various E-commerce websites to show the products as per customer requirement.
- **Useful as a public utility:** AI can be very useful for public utilities such as a self-driving car which can make our journey safer and hassle-free, facial recognition for security purpose, Natural language processing to communicate with the human in human-language, etc.

Disadvantages of Artificial Intelligence

Every technology has some disadvantages, and the same goes for Artificial intelligence. Being so advantageous technology still, it has some disadvantages which we need to keep in our mind while creating an AI system. Following are the disadvantages of AI:

- **High Cost:** The hardware and software requirement of AI is very costly as it requires lots of maintenance to meet current world requirements.
- **Can't think out of the box:** Even we are making smarter machines with AI, but still they cannot work out of the box, as the robot will only do that work for which they are trained, or programmed.
- **No feelings and emotions:** AI machines can be an outstanding performer, but still it does not have the feeling so it cannot make any kind of emotional attachment with human, and may sometime be harmful for users if the proper care is not taken.
- **Increase dependency on machines:** With the increment of technology, people are getting more dependent on devices and hence they are losing their mental capabilities.
- **No Original Creativity:** As humans are so creative and can imagine some new ideas but still AI machines cannot beat this power of human intelligence and cannot be creative and imaginative.

Prerequisite

Before learning about Artificial Intelligence, you must have the fundamental knowledge of following so that you can understand the concepts easily:

- Any computer language such as C, C++, Java, Python, etc. (knowledge of Python will be an advantage)
- Knowledge of essential Mathematics such as derivatives, probability theory, etc.

Application of AI

Artificial Intelligence has various applications in today's society. It is becoming essential for today's time because it can solve complex problems with an efficient way in multiple industries, such as Healthcare, entertainment, finance, education, etc. AI is making our daily life more comfortable and fast.

Following are some sectors which have the application of Artificial Intelligence:

1. AI in Astronomy

- Artificial Intelligence can be very useful to solve complex universe problems. AI technology can be helpful for understanding the universe such as how it works, origin, etc.

2. AI in Healthcare

- In the last, five to ten years, AI becoming more advantageous for the healthcare industry and going to have a significant impact on this industry.
- Healthcare Industries are applying AI to make a better and faster diagnosis than humans. AI can help doctors with diagnoses and can inform when patients are worsening so that medical help can reach to the patient before hospitalization.

3. AI in Gaming

- AI can be used for gaming purpose. The AI machines can play strategic games like chess, where the machine needs to think of a large number of possible places.

4. AI in Finance

- AI and finance industries are the best matches for each other. The finance industry is implementing automation, chatbot, adaptive intelligence, algorithm trading, and machine learning into financial processes.

5. AI in Data Security

- The security of data is crucial for every company and cyber-attacks are growing very rapidly in the digital world. AI can be used to make your data more safe and secure. Some examples such as AEG bot, AI2 Platform, are used to determine software bug and cyber-attacks in a better way.

6. AI in Social Media

- Social Media sites such as Facebook, Twitter, and Snapchat contain billions of user profiles, which need to be stored and managed in a very efficient way. AI can organize and manage massive amounts of data. AI can analyze lots of data to identify the latest trends, hashtag, and requirement of different users.

7. AI in Travel & Transport

- AI is becoming highly demanding for travel industries. AI is capable of doing various travel related works such as from making travel arrangement to suggesting the hotels, flights, and best routes to the customers. Travel industries are using AI-powered chatbots which can make human-like interaction with customers for better and fast response.

8. AI in Automotive Industry

- Some Automotive industries are using AI to provide virtual assistant to their user for better performance. Such as Tesla has introduced TeslaBot, an intelligent virtual assistant.
- Various Industries are currently working for developing self-driven cars which can make your journey more safe and secure.

9. AI in Robotics:

- Artificial Intelligence has a remarkable role in Robotics. Usually, general robots are programmed such that they can perform some repetitive task, but with the help of AI, we can create intelligent robots which can perform tasks with their own experiences without pre-programmed.
- Humanoid Robots are best examples for AI in robotics, recently the intelligent Humanoid robot named as Erica and Sophia has been developed which can talk and behave like humans.

10. AI in Entertainment

- We are currently using some AI based applications in our daily life with some entertainment services such as Netflix or Amazon. With the help of ML/AI algorithms, these services show the recommendations for programs or shows.

11. AI in Agriculture

- Agriculture is an area which requires various resources, labor, money, and time for best result. Now a day's agriculture is becoming digital, and AI is emerging in this field. Agriculture is applying AI as agriculture robotics, soil and crop monitoring, predictive analysis. AI in agriculture can be very helpful for farmers.

12. AI in E-commerce

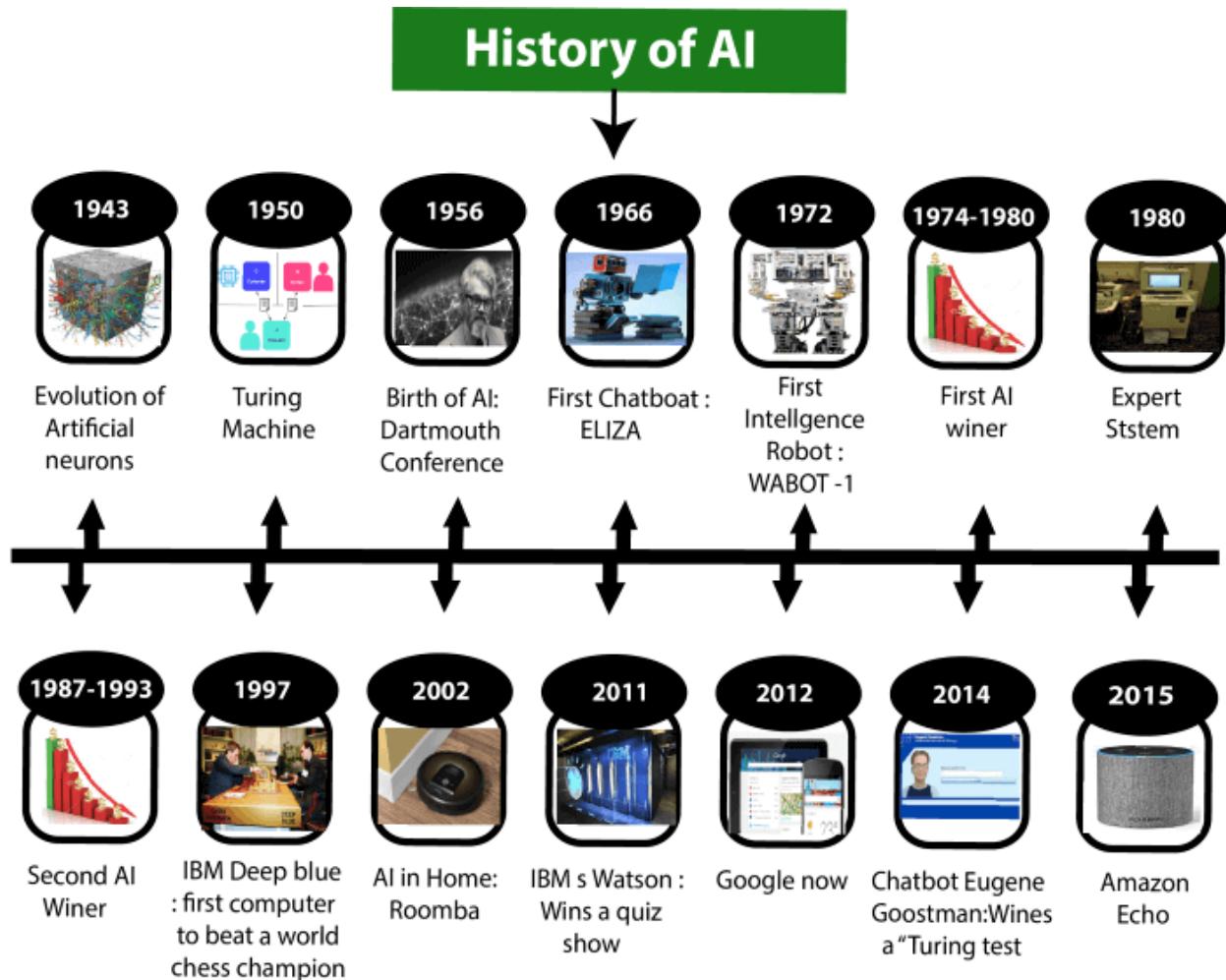
- AI is providing a competitive edge to the e-commerce industry, and it is becoming more demanding in the e-commerce business. AI is helping shoppers to discover associated products with recommended size, color, or even brand.

13. AI in education:

- AI can automate grading so that the tutor can have more time to teach. AI chatbot can communicate with students as a teaching assistant.
- AI in the future can be work as a personal virtual tutor for students, which will be accessible easily at any time and any place.

History of Artificial Intelligence

Artificial Intelligence is not a new word and not a new technology for researchers. This technology is much older than you would imagine. Even there are the myths of Mechanical men in Ancient Greek and Egyptian Myths. Following are some milestones in the history of AI which defines the journey from the AI generation to till date development.



Maturation of Artificial Intelligence (1943-1952)

- **Year 1943:** The first work which is now recognized as AI was done by Warren McCulloch and Walter Pitts in 1943. They proposed a model of **artificial neurons**.
- **Year 1949:** Donald Hebb demonstrated an updating rule for modifying the connection strength between neurons. His rule is now called **Hebbian learning**.
- **Year 1950:** The Alan Turing who was an English mathematician and pioneered Machine learning in 1950. Alan Turing publishes "**Computing Machinery and Intelligence**" in which he proposed a test. The test can check the machine's ability to exhibit intelligent behavior equivalent to human intelligence, called a **Turing test**.

The birth of Artificial Intelligence (1952-1956)

- **Year 1955:** An Allen Newell and Herbert A. Simon created the "first artificial intelligence program" which was named as "**Logic Theorist**". This program had proved 38 of 52 Mathematics theorems, and find new and more elegant proofs for some theorems.
- **Year 1956:** The word "Artificial Intelligence" first adopted by American Computer scientist John McCarthy at the Dartmouth Conference. For the first time, AI coined as an academic field.

At that time high-level computer languages such as FORTRAN, LISP, or COBOL were invented. And the enthusiasm for AI was very high at that time.

The golden years-Early enthusiasm (1956-1974)

- **Year 1966:** The researchers emphasized developing algorithms which can solve mathematical problems. Joseph Weizenbaum created the first chatbot in 1966, which was named as ELIZA.
- **Year 1972:** The first intelligent humanoid robot was built in Japan which was named as WABOT-1.

The first AI winter (1974-1980)

- The duration between years 1974 to 1980 was the first AI winter duration. AI winter refers to the time period where computer scientist dealt with a severe shortage of funding from government for AI researches.
- During AI winters, an interest of publicity on artificial intelligence was decreased.

A boom of AI (1980-1987)

- **Year 1980:** After AI winter duration, AI came back with "Expert System". Expert systems were programmed that emulate the decision-making ability of a human expert.
- In the Year 1980, the first national conference of the American Association of Artificial Intelligence **was held at Stanford University**.

The second AI winter (1987-1993)

- The duration between the years 1987 to 1993 was the second AI Winter duration.
- Again Investors and government stopped in funding for AI research as due to high cost but not efficient result. The expert system such as XCON was very cost effective.

The emergence of intelligent agents (1993-2011)

- **Year 1997:** In the year 1997, IBM Deep Blue beats world chess champion, Gary Kasparov, and became the first computer to beat a world chess champion.
- **Year 2002:** for the first time, AI entered the home in the form of Roomba, a vacuum cleaner.
- **Year 2006:** AI came in the Business world till the year 2006. Companies like Facebook, Twitter, and Netflix also started using AI.

Deep learning, big data and artificial general intelligence (2011-present)

- **Year 2011:** In the year 2011, IBM's Watson won Jeopardy, a quiz show, where it had to solve the complex questions as well as riddles. Watson had proved that it could understand natural language and can solve tricky questions quickly.
- **Year 2012:** Google has launched an Android app feature "Google Now", which was able to provide information to the user as a prediction.
- **Year 2014:** In the year 2014, Chatbot "Eugene Goostman" won a competition in the infamous "Turing test."
- **Year 2018:** The "Project Debater" from IBM debated on complex topics with two master debaters and also performed extremely well.
- Google has demonstrated an AI program "Duplex" which was a virtual assistant and which had taken hairdresser appointment on call, and lady on other side didn't notice that she was talking with the machine.

Now AI has developed to a remarkable level. The concept of Deep learning, big data, and data science are now trending like a boom. Nowadays companies like Google, Facebook, IBM, and Amazon are working with AI and creating amazing devices. The future of Artificial Intelligence is inspiring and will come with high intelligence.

Agents in Artificial Intelligence

An AI system can be defined as the study of the rational agent and its environment. The agents sense the environment through sensors and act on their environment through actuators. An AI agent can have mental properties such as knowledge, belief, intention, etc.

What is an Agent?

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

- **Human-Agent:** A human agent has eyes, ears, and other organs which work for sensors and hand, legs, vocal tract work for actuators.
- **Robotic Agent:** A robotic agent can have cameras, infrared range finder, NLP for sensors and various motors for actuators.
- **Software Agent:** Software agent can have keystrokes, file contents as sensory input and act on those inputs and display output on the screen.

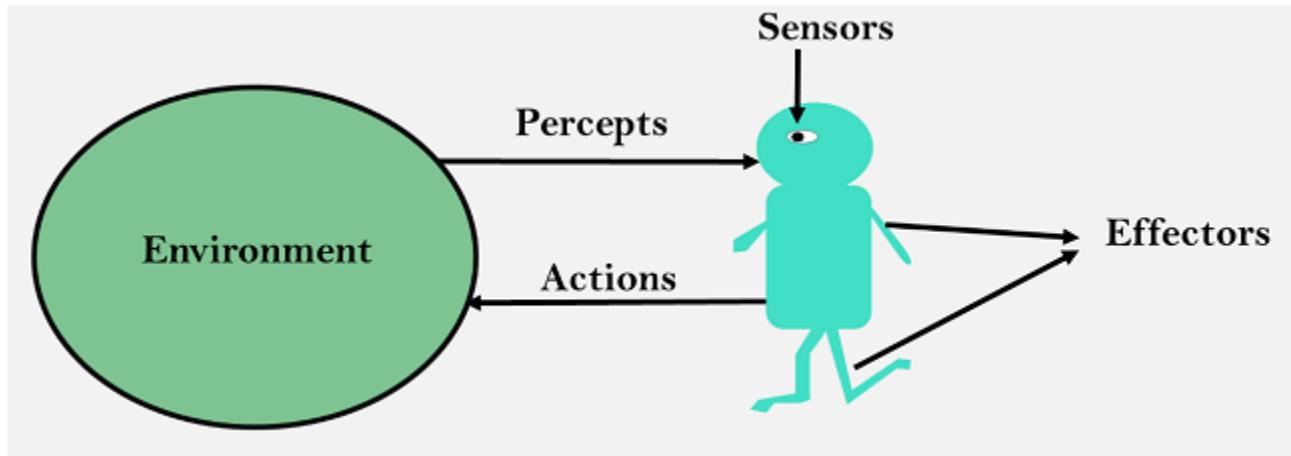
Hence the world around us is full of agents such as thermostat, cellphone, camera, and even we are also agents.

Before moving forward, we should first know about sensors, effectors, and actuators.

Sensor: Sensor is a device which detects the change in the environment and sends the information to other electronic devices. An agent observes its environment through sensors.

Actuators: Actuators are the component of machines that converts energy into motion. The actuators are only responsible for moving and controlling a system. An actuator can be an electric motor, gears, rails, etc.

Effectors: Effectors are the devices which affect the environment. Effectors can be legs, wheels, arms, fingers, wings, fins, and display screen.



Intelligent Agents:

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

Following are the main four rules for an AI agent:

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

Rational Agent:

A rational agent is an agent which has clear preference, models uncertainty, and acts in a way to maximize its performance measure with all possible actions.

A rational agent is said to perform the right things. AI is about creating rational agents to use for game theory and decision theory for various real-world scenarios.

For an AI agent, the rational action is most important because in AI reinforcement learning algorithm, for each best possible action, agent gets the positive reward and for each wrong action, an agent gets a negative reward.

Note: Rational agents in AI are very similar to intelligent agents.

Rationality:

The rationality of an agent is measured by its performance measure. Rationality can be judged on the basis of following points:

- Performance measure which defines the success criterion.
- Agent prior knowledge of its environment.
- Best possible actions that an agent can perform.
- The sequence of percepts.

Note: Rationality differs from Omnicience because an Omnicient agent knows the actual outcome of its action and act accordingly, which is not possible in reality.

Structure of an AI Agent

The task of AI is to design an agent program which implements the agent function. The structure of an intelligent agent is a combination of architecture and agent program. It can be viewed as:

1. $\text{Agent} = \text{Architecture} + \text{Agent program}$

Following are the main three terms involved in the structure of an AI agent:

Architecture: Architecture is machinery that an AI agent executes on.

Agent Function: Agent function is used to map a percept to an action.

1. $f: P^* \rightarrow A$

Agent program: Agent program is an implementation of agent function. An agent program executes on the physical architecture to produce function f .

PEAS Representation

PEAS is a type of model on which an AI agent works upon. When we define an AI agent or rational agent, then we can group its properties under PEAS representation model. It is made up of four words:

- **P:** Performance measure
- **E:** Environment
- **A:** Actuators
- **S:** Sensors

Here performance measure is the objective for the success of an agent's behavior.

PEAS for self-driving cars:

Let's suppose a self-driving car then PEAS representation will be:

Performance: Safety, time, legal drive, comfort

Environment: Roads, other vehicles, road signs, pedestrian

Actuators: Steering, accelerator, brake, signal, horn

Sensors: Camera, GPS, speedometer, odometer, accelerometer, sonar.

Example of Agents with their PEAS representation

Agent	Performance measure	Environment	Actuators	Sensors
1. Medical Diagnose	<ul style="list-style-type: none">o Healthy patiento Minimized cost	<ul style="list-style-type: none">o Patiento Hospitalo Staff	<ul style="list-style-type: none">o Testso Treatments	Keyboard (Entry of symptoms)
2. Vacuum Cleaner	<ul style="list-style-type: none">o Cleanlinesso Efficiencyo Battery lifeo Security	<ul style="list-style-type: none">o Roomo Tableo Wood flooro Carpeto Various obstacles	<ul style="list-style-type: none">o Wheelso Brusheso Vacuum Extractor	<ul style="list-style-type: none">o Camerao Dirt detection sensoro Cliff sensoro Bump Sensoro Infrared Wall Sensor
3. Part -picking Robot	<ul style="list-style-type: none">o Percentage of parts in correct bins.	<ul style="list-style-type: none">o Conveyor belt with parts,o Bins	<ul style="list-style-type: none">o Jointed Armso Hand	<ul style="list-style-type: none">o Camerao Joint angle sensors.

Agent Environment in AI

An environment is everything in the world which surrounds the agent, but it is not a part of an agent itself. An environment can be described as a situation in which an agent is present.

The environment is where agent lives, operate and provide the agent with something to sense and act upon it. An environment is mostly said to be non-feministic.

Features of Environment

As per Russell and Norvig, an environment can have various features from the point of view of an agent:

1. Fully observable vs Partially Observable
2. Static vs Dynamic
3. Discrete vs Continuous
4. Deterministic vs Stochastic
5. Single-agent vs Multi-agent

6. Episodic vs sequential
7. Known vs Unknown
8. Accessible vs Inaccessible

1. 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**.
- A fully observable environment is easy as there is no need to maintain the internal state to keep track history of the world.
- An agent with no sensors in all environments then such an environment is called as **unobservable**.

2. Deterministic vs Stochastic:

- If an agent's current state and selected action can completely determine the next state of the environment, then such environment is called a deterministic environment.
- A stochastic environment is random in nature and cannot be determined completely by an agent.
- In a deterministic, fully observable environment, agent does not need to worry about uncertainty.

3. Episodic vs Sequential:

- In an episodic environment, there is a series of one-shot actions, and only the current percept is required for the action.
- However, in Sequential environment, an agent requires memory of past actions to determine the next best actions.

4. Single-agent vs Multi-agent

- If only one agent is involved in an environment, and operating by itself then such an environment is called single agent environment.
- However, if multiple agents are operating in an environment, then such an environment is called a multi-agent environment.
- The agent design problems in the multi-agent environment are different from single agent environment.

5. Static vs Dynamic:

- If the environment can change itself while an agent is deliberating then such environment is called a dynamic environment else it is called a static environment.
- Static environments are easy to deal because an agent does not need to continue looking at the world while deciding for an action.
- However for dynamic environment, agents need to keep looking at the world at each action.
- Taxi driving is an example of a dynamic environment whereas Crossword puzzles are an example of a static environment.

6. Discrete vs Continuous:

- If in an environment there are a finite number of percepts and actions that can be performed within it, then such an environment is called a discrete environment else it is called continuous environment.
- A chess game comes under discrete environment as there is a finite number of moves that can be performed.
- A self-driving car is an example of a continuous environment.

7. Known vs Unknown

- Known and unknown are not actually a feature of an environment, but it is an agent's state of knowledge to perform an action.
- In a known environment, the results for all actions are known to the agent. While in unknown environment, agent needs to learn how it works in order to perform an action.
- It is quite possible that a known environment to be partially observable and an Unknown environment to be fully observable.

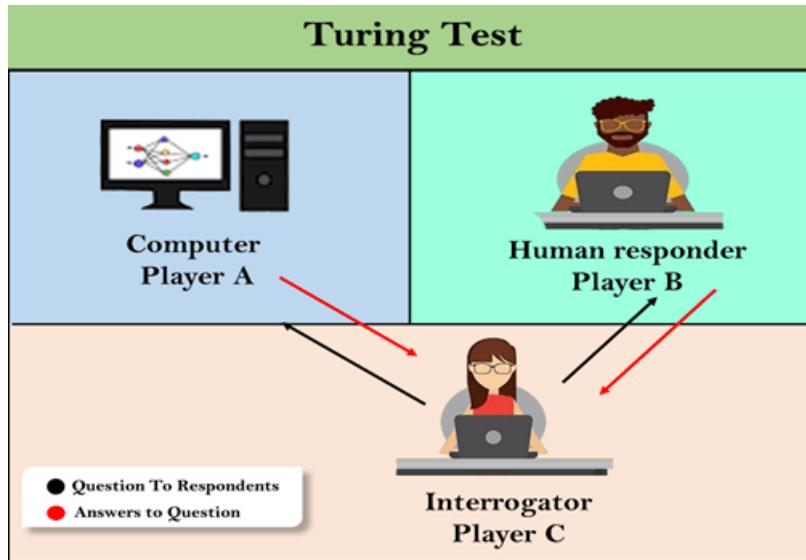
8. Accessible vs Inaccessible

- If an agent can obtain complete and accurate information about the state's environment, then such an environment is called an Accessible environment else it is called inaccessible.
- An empty room whose state can be defined by its temperature is an example of an accessible environment.
- Information about an event on earth is an example of Inaccessible environment.

Turing Test in AI

In 1950, Alan Turing introduced a test to check whether a machine can think like a human or not, this test is known as the Turing Test. In this test, Turing proposed that the computer can be said to be an intelligent if it can mimic human response under specific conditions.

Turing Test was introduced by Turing in his 1950 paper, "Computing Machinery and Intelligence," which considered the question, "Can Machine think?"



The Turing test is based on a party game "Imitation game," with some modifications. This game involves three players in which one player is Computer, another player is human responder, and the third player is a human Interrogator, who is isolated from other two players and his job is to find that which player is machine among two of them.

Consider, Player A is a computer, Player B is human, and Player C is an interrogator. Interrogator is aware that one of them is machine, but he needs to identify this on the basis of questions and their responses.

The conversation between all players is via keyboard and screen so the result would not depend on the machine's ability to convert words as speech.

The test result does not depend on each correct answer, but only how closely its responses like a human answer. The computer is permitted to do everything possible to force a wrong identification by the interrogator.

The questions and answers can be like:

Interrogator: Are you a computer?

PlayerA (Computer): No

Interrogator: Multiply two large numbers such as (256896489*456725896)

Player A: Long pause and give the wrong answer.

In this game, if an interrogator would not be able to identify which is a machine and which is human, then the computer passes the test successfully, and the machine is said to be intelligent and can think like a human.

"In 1991, the New York businessman Hugh Loebner announces the prize competition, offering a \$100,000 prize for the first computer to pass the Turing test. However, no AI program to till date, come close to passing an undiluted Turing test".

Chatbots to attempt the Turing test:

ELIZA: ELIZA was a Natural language processing computer program created by Joseph Weizenbaum. It was created to demonstrate the ability of communication between machine and humans. It was one of the first chatterbots, which has attempted the Turing Test.

Parry: Parry was a chatterbot created by Kenneth Colby in 1972. Parry was designed to simulate a person with **Paranoid schizophrenia** (most common chronic mental disorder). Parry was described as "ELIZA with attitude." Parry was tested using a variation of the Turing Test in the early 1970s.

Eugene Goostman: **Eugene Goostman was a chatbot developed in Saint Petersburg in 2001. This bot has competed in the various number of Turing Test. In June 2012, at an event, Goostman won the competition promoted as largest-ever Turing test content, in which it has convinced 29% of judges that it was a human. Goostman resembled as a 13-year old virtual boy.**

The Chinese Room Argument:

There were many philosophers who really disagreed with the complete concept of Artificial Intelligence. The most famous argument in this list was "**Chinese Room**."

In the year **1980**, **John Searle** presented "**Chinese Room**" thought experiment, in his paper "**Mind, Brains, and Program**," which was against the validity of Turing's Test. According to his argument, "**Programming a computer may make it to understand a language, but it will not produce a real understanding of language or consciousness in a computer.**"

He argued that Machine such as ELIZA and Parry could easily pass the Turing test by manipulating keywords and symbol, but they had no real understanding of language. So it cannot be described as "thinking" capability of a machine such as a human.

Features required for a machine to pass the Turing test:

- **Natural language processing:** NLP is required to communicate with Interrogator in general human language like English.
- **Knowledge representation:** To store and retrieve information during the test.
- **Automated reasoning:** To use the previously stored information for answering the questions.
- **Machine learning:** To adapt new changes and can detect generalized patterns.
- **Vision (For total Turing test):** To recognize the interrogator actions and other objects during a test.
- **Motor Control (For total Turing test):** To act upon objects if requested.

Problem-solving agent

- The problem-solving agent performs precisely by defining problems and its several solutions.
- According to psychology, “*a problem-solving refers to a state where we wish to reach to a definite goal from a present state or condition.*”
- According to computer science, *a problem-solving is a part of artificial intelligence which encompasses a number of techniques such as algorithms, heuristics to solve a problem.*
- Therefore, a problem-solving agent is a **goal-driven agent** and focuses on satisfying the goal.
- **Steps performed by Problem-solving agent**
- **Goal Formulation:** It is the first and simplest step in problem-solving. It organizes the steps/sequence required to formulate one goal out of multiple goals as well as actions to achieve that goal. Goal formulation is based on the current situation and the agent’s performance measure (discussed below).
- **Problem Formulation:** It is the most important step of problem-solving which decides what actions should be taken to achieve the formulated goal. There are following five components involved in problem formulation:
 - **Initial State:** It is the starting state or initial step of the agent towards its goal.
 - **Actions:** It is the description of the possible actions available to the agent.
 - **Transition Model:** It describes what each action does.
 - **Goal Test:** It determines if the given state is a goal state.
 - **Path cost:** It assigns a numeric cost to each path that follows the goal. The problem-solving agent selects a cost function, which reflects its performance measure.
Remember, **an optimal solution has the lowest path cost among all the solutions.**
- **Note:** **Initial state, actions, and transition model** together define the **state-space** of the problem implicitly. State-space of a problem is a set of all states which can be reached from the initial state followed by any sequence of actions. The state-space forms a directed map or graph where nodes are the states, links between the nodes are actions, and the path is a sequence of states connected by the sequence of actions.
- **Search:** It identifies all the best possible sequence of actions to reach the goal state from the current state. It takes a problem as an input and returns solution as its output.
- **Solution:** It finds the best algorithm out of various algorithms, which may be proven as the best optimal solution.
- **Execution:** It executes the best optimal solution from the searching algorithms to reach the goal state from the current state.

Properties of Search Algorithms:

Completeness: A search algorithm is said to be complete if it guarantees to return a solution if at least any solution exists for any random input.

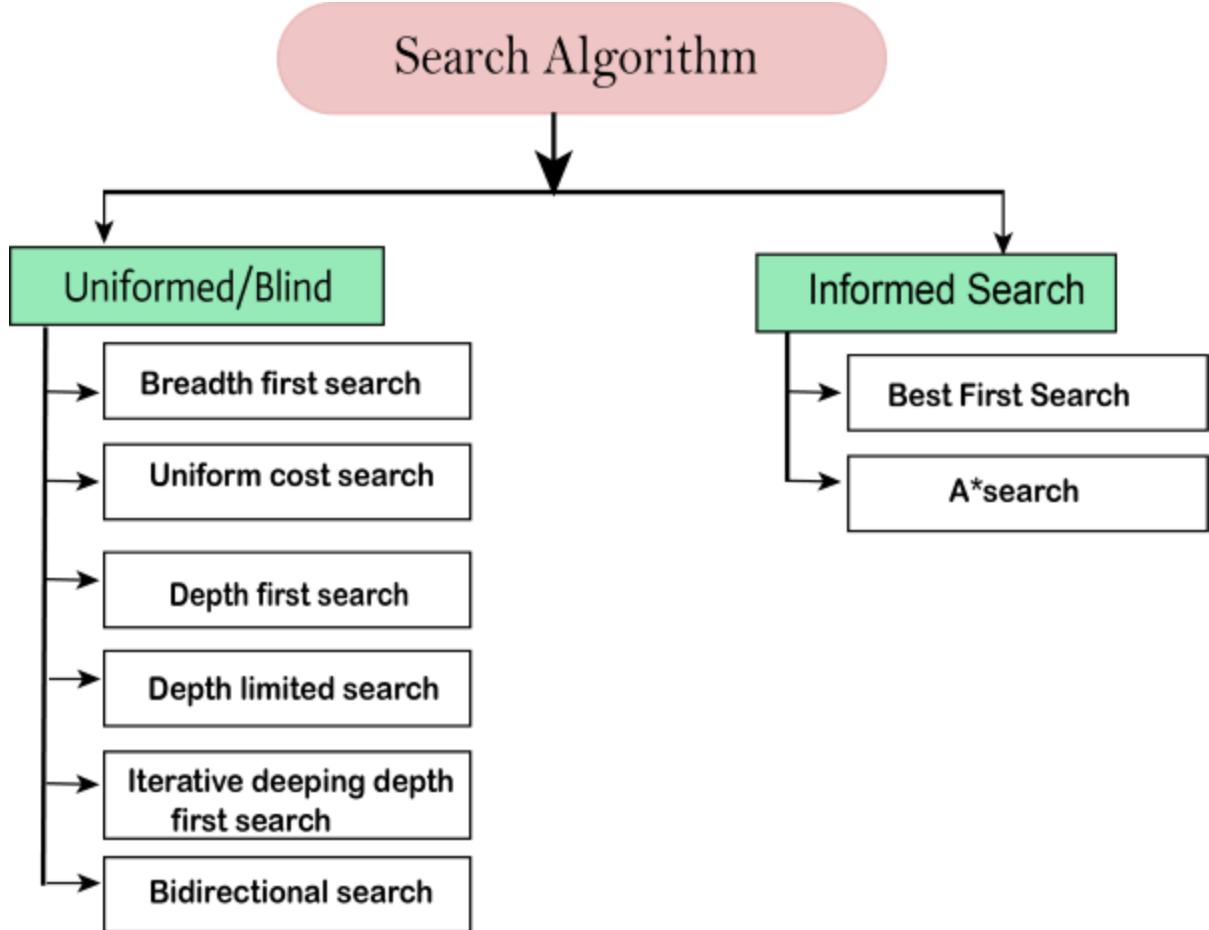
Optimality: If a solution found for an algorithm is guaranteed to be the best solution (lowest path cost) among all other solutions, then such a solution for is said to be an optimal solution.

Time Complexity: Time complexity is a measure of time for an algorithm to complete its task.

Space Complexity: It is the maximum storage space required at any point during the search, as the complexity of the problem.

Types of search algorithms

Based on the search problems we can classify the search algorithms into uninformed (Blind search) search and informed search (Heuristic search) algorithms.



Uninformed/Blind Search:

The uninformed search does not contain any domain knowledge such as closeness, the location of the goal. It operates in a brute-force way as it only includes information about how to traverse the tree and how to identify leaf and goal nodes. Uninformed search applies a way in which search tree is searched without any information about

the search space like initial state operators and test for the goal, so it is also called blind search. It examines each node of the tree until it achieves the goal node.

It can be divided into five main types:

- Breadth-first search
- Uniform cost search
- Depth-first search
- Iterative deepening depth-first search
- Bidirectional Search

Informed Search

Informed search algorithms use domain knowledge. In an informed search, problem information is available which can guide the search. Informed search strategies can find a solution more efficiently than an uninformed search strategy. Informed search is also called a Heuristic search.

A heuristic is a way which might not always be guaranteed for best solutions but guaranteed to find a good solution in reasonable time.

Informed search can solve much complex problem which could not be solved in another way.

An example of informed search algorithms is a traveling salesman problem.

1. Greedy Search
2. A* Search

Uninformed Search Algorithms

Uninformed search is a class of general-purpose search algorithms which operates in brute force-way. Uninformed search algorithms do not have additional information about state or search space other than how to traverse the tree, so it is also called blind search.

Following are the various types of uninformed search algorithms:

1. Breadth-first Search
2. Depth-first Search
3. Depth-limited Search
4. Iterative deepening depth-first search

5. **Uniform cost search**
6. **Bidirectional Search**

1. Breadth-first Search:

- Breadth-first search is the most common search strategy for traversing a tree or graph. This algorithm searches breadthwise in a tree or graph, so it is called breadth-first search.
- BFS algorithm starts searching from the root node of the tree and expands all successor node at the current level before moving to nodes of next level.
- The breadth-first search algorithm is an example of a general-graph search algorithm.
- Breadth-first search implemented using FIFO queue data structure.
- **Advantages:** 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:

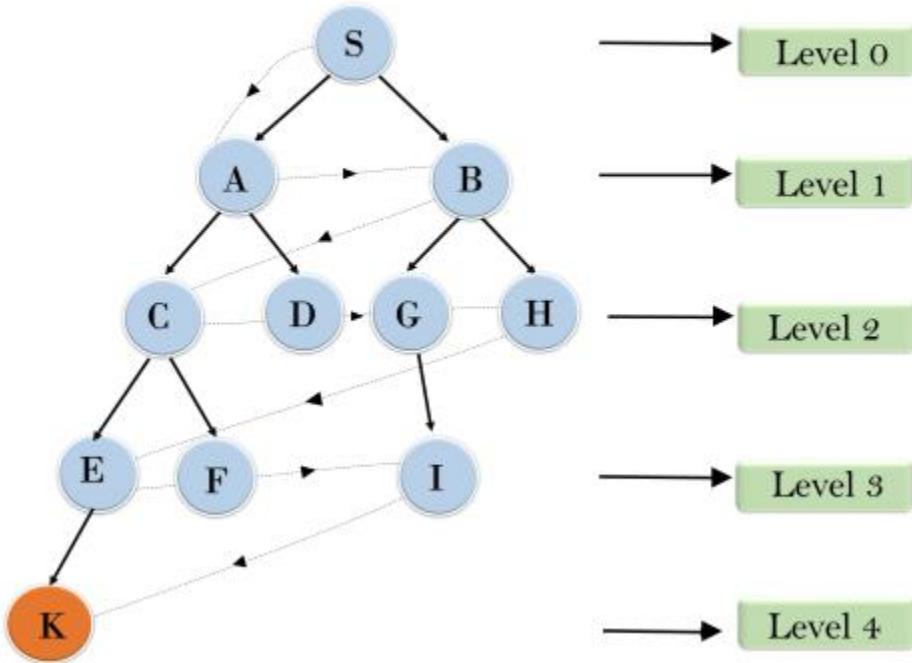
- It requires lots of memory since each level of the tree must be saved into memory to expand the next level.
- BFS needs lots of time if the solution is far away from the root node.

Example:

In the below tree structure, we have shown the traversing of the tree using BFS algorithm from the root node S to goal node K. BFS search algorithm traverse in layers, so it will follow the path which is shown by the dotted arrow, and the traversed path will be:

1. S---> A--->B---->C--->D---->G--->H--->E---->F---->I---->K

Breadth First Search



Time Complexity: Time Complexity of BFS algorithm can be obtained by the number of nodes traversed in BFS until the shallowest Node. Where the d = depth of shallowest solution and b is a node at every state.

$$T(b) = 1+b^2+b^3+\dots+b^d = O(b^d)$$

Space Complexity: Space complexity of BFS algorithm is given by the Memory size of frontier which is $O(b^d)$.

Completeness: BFS is complete, which means if the shallowest goal node is at some finite depth, then BFS will find a solution.

Optimality: BFS is optimal if path cost is a non-decreasing function of the depth of the node.

2. Depth-first Search

- Depth-first search is a recursive algorithm for traversing a tree or graph data structure.
- It is called the depth-first search because it starts from the root node and follows each path to its greatest depth node before moving to the next path.
- DFS uses a stack data structure for its implementation.
- The process of the DFS algorithm is similar to the BFS algorithm.

Note: Backtracking is an algorithm technique for finding all possible solutions using recursion.

Advantage:

- DFS requires very less memory as it only needs to store a stack of the nodes on the path from root node to the current node.
- It takes less time to reach to the goal node than BFS algorithm (if it traverses in the right path).

Disadvantage:

- There is the possibility that many states keep re-occurring, and there is no guarantee of finding the solution.
- DFS algorithm goes for deep down searching and sometime it may go to the infinite loop.

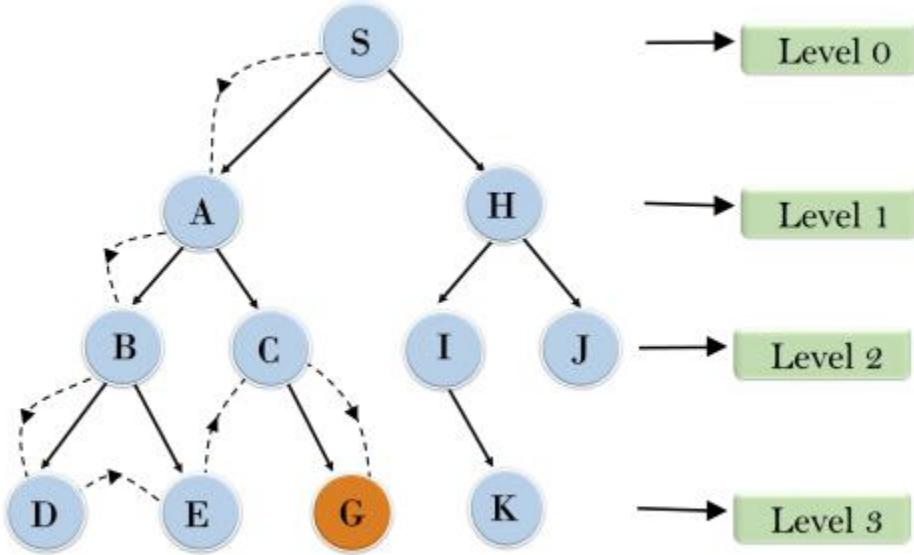
Example:

In the below search tree, we have shown the flow of depth-first search, and it will follow the order as:

Root node--->Left node ----> right node.

It will start searching from root node S, and traverse A, then B, then D and E, after traversing E, it will backtrack the tree as E has no other successor and still goal node is not found. After backtracking it will traverse node C and then G, and here it will terminate as it found goal node.

Depth First Search



Completeness: DFS search algorithm is complete within finite state space as it will expand every node within a limited search tree.

Time Complexity: Time complexity of DFS will be equivalent to the node traversed by the algorithm. It is given by:

$$T(n) = 1 + n^2 + n^3 + \dots + n^m = O(n^m)$$

Where, m = maximum depth of any node and this can be much larger than d (Shallowest solution depth)

Space Complexity: DFS algorithm needs to store only single path from the root node, hence space complexity of DFS is equivalent to the size of the fringe set, which is **O(bm)**.

Optimal: DFS search algorithm is non-optimal, as it may generate a large number of steps or high cost to reach to the goal node.

Informed Search Algorithms

So far we have talked about the uninformed search algorithms which looked through search space for all possible solutions of the problem without having any additional knowledge about search space. But informed search algorithm contains an array of knowledge such as how far we are from the goal, path cost, how to reach to goal node, etc. This knowledge help agents to explore less to the search space and find more efficiently the goal node.

The informed search algorithm is more useful for large search space. Informed search algorithm uses the idea of heuristic, so it is also called Heuristic search.

Heuristics function: Heuristic is a function which is used in Informed Search, and it finds the most promising path. It takes the current state of the agent as its input and produces the estimation of how close agent is from the goal. The heuristic method, however, might not always give the best solution, but it guaranteed to find a good solution in reasonable time. Heuristic function estimates how close a state is to the goal. It is represented by $h(n)$, and it calculates the cost of an optimal path between the pair of states. The value of the heuristic function is always positive.

Admissibility of the heuristic function is given as:

$$1. \quad h(n) \leq h^*(n)$$

Here $h(n)$ is heuristic cost, and $h^*(n)$ is the estimated cost. Hence heuristic cost should be less than or equal to the estimated cost.

Pure Heuristic Search:

Pure heuristic search is the simplest form of heuristic search algorithms. It expands nodes based on their heuristic value $h(n)$. It maintains two lists, OPEN and CLOSED list. In the CLOSED list, it places those nodes which have already expanded and in the OPEN list, it places nodes which have yet not been expanded.

On each iteration, each node n with the lowest heuristic value is expanded and generates all its successors and n is placed to the closed list. The algorithm continues until a goal state is found.

In the informed search we will discuss two main algorithms which are given below:

- **Best First Search Algorithm(Greedy search)**
- **A* Search Algorithm**

1.) Best-first Search Algorithm (Greedy Search):

Greedy best-first search algorithm always selects the path which appears best at that moment. It is the combination of depth-first search and breadth-first search algorithms. It uses the heuristic function and search. Best-first search allows us to take the advantages of both algorithms. With the help of best-first search, at each step, we can choose the most promising node. In the best first search algorithm, we expand the node which is closest to the goal node and the closest cost is estimated by heuristic function, i.e.

$$1. \quad f(n) = g(n).$$

Where, $h(n)$ = estimated cost from node n to the goal.

The greedy best first algorithm is implemented by the priority queue.

Best first search algorithm:

- **Step 1:** Place the starting node into the OPEN list.
- **Step 2:** If the OPEN list is empty, Stop and return failure.
- **Step 3:** Remove the node n , from the OPEN list which has the lowest value of $h(n)$, and places it in the CLOSED list.
- **Step 4:** Expand the node n , and generate the successors of node n .
- **Step 5:** Check each successor of node n , and find whether any node is a goal node or not. If any successor node is goal node, then return success and terminate the search, else proceed to Step 6.
- **Step 6:** For each successor node, algorithm checks for evaluation function $f(n)$, and then check if the node has been in either OPEN or CLOSED list. If the node has not been in both list, then add it to the OPEN list.
- **Step 7:** Return to Step 2.

Advantages:

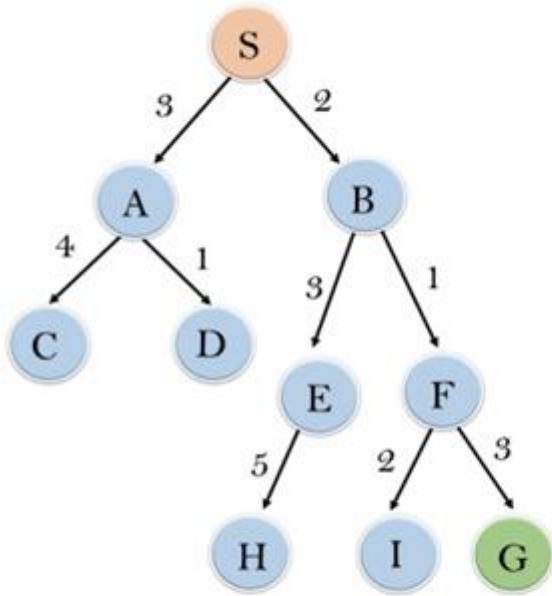
- Best first search can switch between BFS and DFS by gaining the advantages of both the algorithms.
- This algorithm is more efficient than BFS and DFS algorithms.

Disadvantages:

- It can behave as an unguided depth-first search in the worst case scenario.
- It can get stuck in a loop as DFS.
- This algorithm is not optimal.

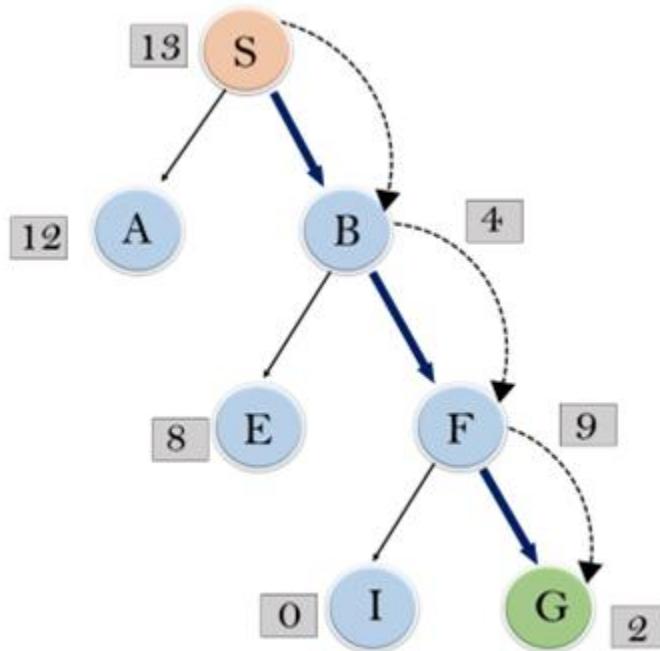
Example:

Consider the below search problem, and we will traverse it using greedy best-first search. At each iteration, each node is expanded using evaluation function $f(n)=h(n)$, which is given in the below table.



node	H (n)
A	12
B	4
C	7
D	3
E	8
F	2
H	4
I	9
S	13
G	0

In this search example, we are using two lists which are **OPEN** and **CLOSED** Lists. Following are the iteration for traversing the above example



Expand the nodes of S and put in the CLOSED list

Initialization: Open [A, B], Closed [S]

Iteration 1: Open [A], Closed [S, B]

Iteration 2: Open [E, F, A], Closed [S, B]
: Open [E, A], Closed [S, B, F]

Iteration 3: Open [I, G, E, A], Closed [S, B, F]
: Open [I, E, A], Closed [S, B, F, G]

Hence the final solution path will be: **S**----> **B**---->**F**----> **G**

Time Complexity: The worst case time complexity of Greedy best first search is $O(b^m)$.

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

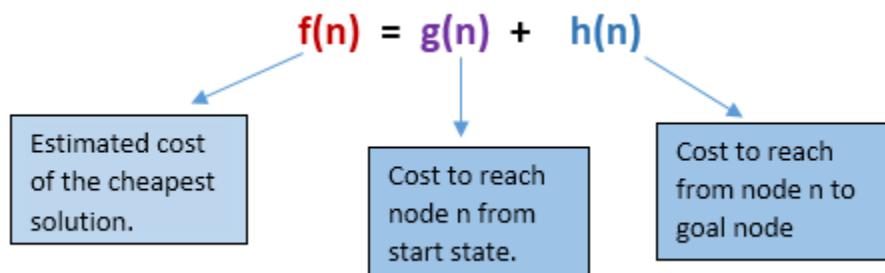
Complete: Greedy best-first search is also incomplete, even if the given state space is finite.

Optimal: Greedy best first search algorithm is not optimal.

2.) A* Search Algorithm:

A* search is the most commonly known form of best-first search. It uses heuristic function $h(n)$, and cost to reach the node n from the start state $g(n)$. It has combined features of UCS and greedy best-first search, by which it solve the problem efficiently. A* search algorithm finds the shortest path through the search space using the heuristic function. This search algorithm expands less search tree and provides optimal result faster. A* algorithm is similar to UCS except that it uses $g(n)+h(n)$ instead of $g(n)$.

In A* search algorithm, we use search heuristic as well as the cost to reach the node. Hence we can combine both costs as following, and this sum is called as a **fitness number**.



At each point in the search space, only those node is expanded which have the lowest value of $f(n)$, and the algorithm terminates when the goal node is found.

Algorithm of A* search:

Step1: Place the starting node in the OPEN list.

Step 2: Check if the OPEN list is empty or not, if the list is empty then return failure and stops.

Step 3: Select the node from the OPEN list which has the smallest value of evaluation function ($g+h$), if node n is goal node then return success and stop, otherwise

Step 4: Expand node n and generate all of its successors, and put n into the closed list. For each successor n' , check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.

Step 5: Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest $g(n')$ value.

Step 6: Return to **Step 2**.

Advantages:

- A* search algorithm is the best algorithm than other search algorithms.
- A* search algorithm is optimal and complete.
- This algorithm can solve very complex problems.

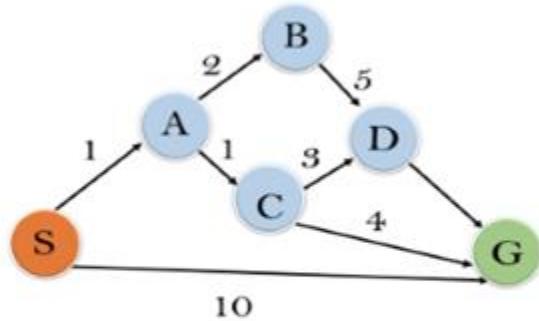
Disadvantages:

- It does not always produce the shortest path as it mostly based on heuristics and approximation.
- A* search algorithm has some complexity issues.
- The main drawback of A* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.

Example:

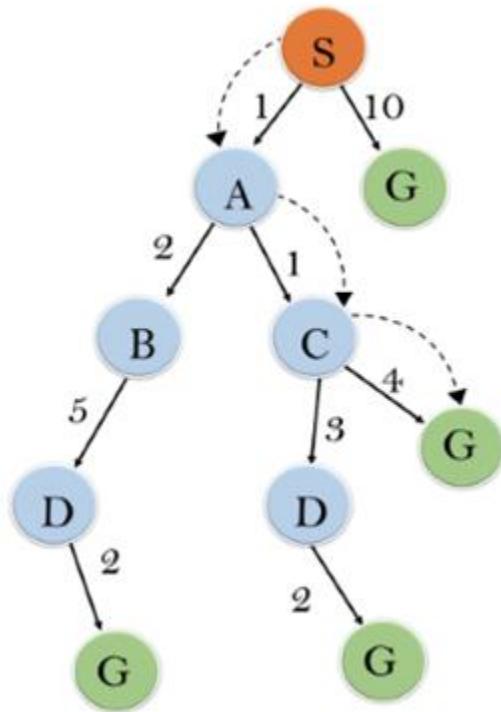
In this example, we will traverse the given graph using the A* algorithm. The heuristic value of all states is given in the below table so we will calculate the $f(n)$ of each state using the formula $f(n) = g(n) + h(n)$, where $g(n)$ is the cost to reach any node from start state.

Here we will use OPEN and CLOSED list.



State	$h(n)$
S	5
A	3
B	4
C	2
D	6
G	0

Solution:



Initialization: $\{(S, 5)\}$

Iteration1: $\{(S \rightarrow A, 4), (S \rightarrow G, 10)\}$

Iteration2: $\{(S \rightarrow A \rightarrow C, 4), (S \rightarrow A \rightarrow B, 7), (S \rightarrow G, 10)\}$

Iteration3: {(S--> A-->C--->G, 6), (S--> A-->C--->D, 11), (S--> A-->B, 7), (S-->G, 10)}

Iteration 4 will give the final result, as **S--->A--->C--->G** it provides the optimal path with cost 6.

Points to remember:

- A* algorithm returns the path which occurred first, and it does not search for all remaining paths.
- The efficiency of A* algorithm depends on the quality of heuristic.
- A* algorithm expands all nodes which satisfy the condition $f(n) \leq h(n)$

Complete: A* algorithm is complete as long as:

- Branching factor is finite.
- Cost at every action is fixed.

Optimal: A* search algorithm is optimal if it follows below two conditions:

- **Admissible:** the first condition requires for optimality is that $h(n)$ should be an admissible heuristic for A* tree search. An admissible heuristic is optimistic in nature.
- **Consistency:** Second required condition is consistency for only A* graph-search.

If the heuristic function is admissible, then A* tree search will always find the least cost path.

Time Complexity: The time complexity of A* search algorithm depends on heuristic function, and the number of nodes expanded is exponential to the depth of solution d . So the time complexity is $O(b^d)$, where b is the branching factor.

Space Complexity: The space complexity of A* search algorithm is **$O(b^d)$**

Hill Climbing Algorithm in Artificial Intelligence

- 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.

- Hill climbing algorithm is a technique which is used for optimizing the mathematical problems. One of the widely discussed examples of Hill climbing algorithm is Traveling-salesman Problem in which we need to minimize the distance traveled by the salesman.
- It is also called greedy local search as it only looks to its good immediate neighbor state and not beyond that.
- A node of hill climbing algorithm has two components which are state and value.
- 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.

Features of Hill Climbing:

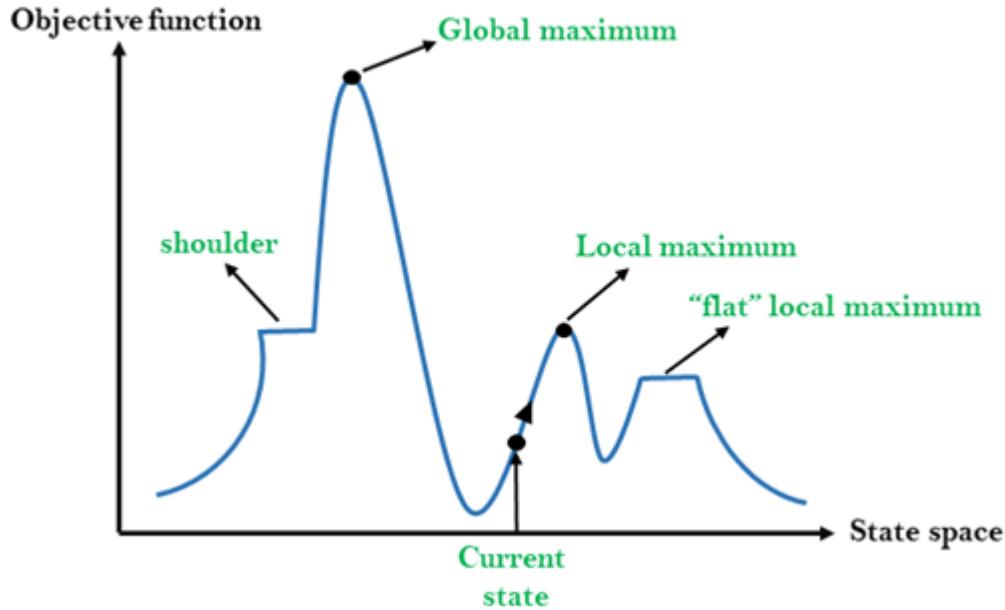
Following are some main features of Hill Climbing Algorithm:

- **Generate and Test variant:** Hill Climbing is the variant of Generate and Test method. The Generate and Test method produce feedback which helps to decide which direction to move in the search space.
- **Greedy approach:** Hill-climbing algorithm search moves in the direction which optimizes the cost.
- **No backtracking:** It does not backtrack the search space, as it does not remember the previous states.

State-space Diagram for Hill Climbing:

The state-space landscape is a graphical representation of the hill-climbing algorithm which is showing a graph between various states of algorithm and Objective function/Cost.

On Y-axis we have taken the function which can be an objective function or cost function, and state-space on the x-axis. If the function on Y-axis is cost then, the goal of search is to find the global minimum and local minimum. If the function of Y-axis is Objective function, then the goal of the search is to find the global maximum and local maximum.



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.

Types of Hill Climbing Algorithm:

- Simple hill Climbing:
- Steepest-Ascent hill-climbing:
- Stochastic hill Climbing:

1. Simple Hill Climbing:

Simple hill climbing is the simplest way to implement a hill climbing algorithm. **It only evaluates the neighbor node state at a time and selects the first one which optimizes current cost and set it as a current state.** It only checks it's one

successor state, and if it finds better than the current state, then move else be in the same state. This algorithm has the following features:

- Less time consuming
- Less optimal solution and the solution is not guaranteed

Algorithm for Simple Hill Climbing:

- **Step 1:** Evaluate the initial state, if it is goal state then return success and Stop.
- **Step 2:** Loop Until a solution is found or there is no new operator left to apply.
- **Step 3:** Select and apply an operator to the current state.
- **Step 4:** Check new state:
 - a. If it is goal state, then return success and quit.
 - b. Else if it is better than the current state then assign new state as a current state.
 - c. Else if not better than the current state, then return to step2.
- **Step 5:** Exit.

2. Steepest-Ascent hill climbing:

The steepest-Ascent algorithm is a variation of simple hill climbing algorithm. This algorithm examines all the neighboring nodes of the current state and selects one neighbor node which is closest to the goal state. This algorithm consumes more time as it searches for multiple neighbors

Algorithm for Steepest-Ascent hill climbing:

- **Step 1:** Evaluate the initial state, if it is goal state then return success and stop, else make current state as initial state.
- **Step 2:** Loop until a solution is found or the current state does not change.
 - a. Let SUCC be a state such that any successor of the current state will be better than it.
 - b. For each operator that applies to the current state:
 - a. Apply the new operator and generate a new state.
 - b. Evaluate the new state.
 - c. If it is goal state, then return it and quit, else compare it to the SUCC.
 - d. If it is better than SUCC, then set new state as SUCC.

- e. If the SUCC is better than the current state, then set current state to SUCC.
- o **Step 5:** Exit.

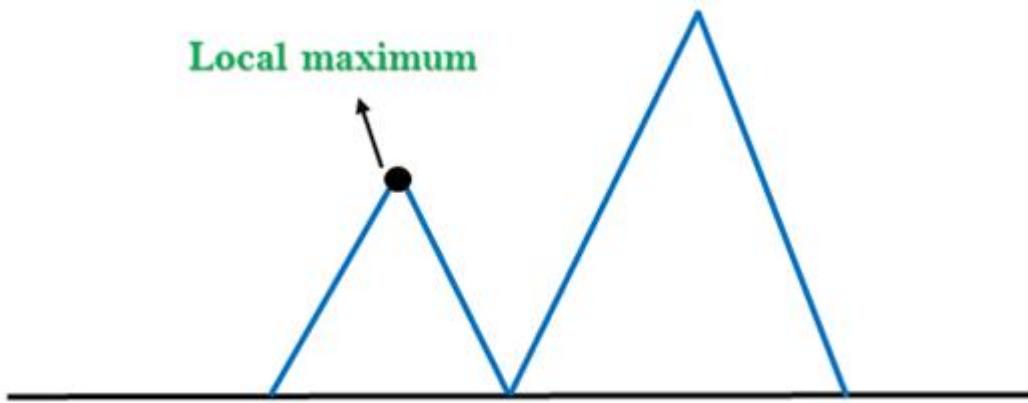
3. Stochastic hill climbing:

Stochastic hill climbing does not examine for all its neighbor before moving. Rather, this search algorithm selects one neighbor node at random and decides whether to choose it as a current state or examine another state.

Problems in Hill Climbing Algorithm:

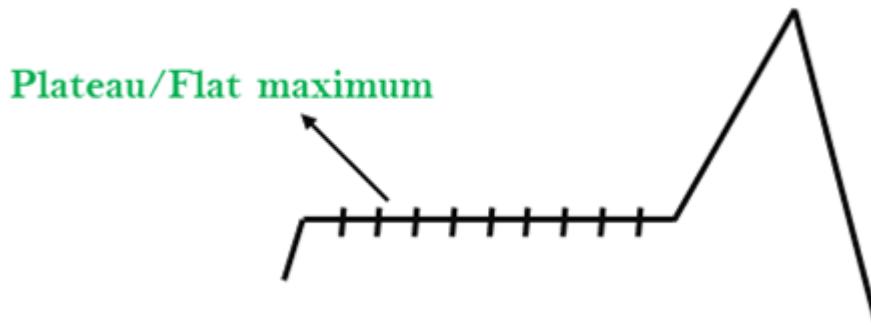
1. Local Maximum: A local maximum is a peak state in the landscape which is better than each of its neighboring states, but there is another state also present which is higher than the local maximum.

Solution: Backtracking technique can be a solution of the local maximum in state space landscape. Create a list of the promising path so that the algorithm can backtrack the search space and explore other paths as well.



2. Plateau: A plateau is the flat area of the search space in which all the neighbor states of the current state contains the same value, because of this algorithm does not find any best direction to move. A hill-climbing search might be lost in the plateau area.

Solution: The solution for the plateau is to take big steps or very little steps while searching, to solve the problem. Randomly select a state which is far away from the current state so it is possible that the algorithm could find non-plateau region.



3. Ridges: A ridge is a special form of the local maximum. It has an area which is higher than its surrounding areas, but itself has a slope, and cannot be reached in a single move.

Solution: With the use of bidirectional search, or by moving in different directions, we can improve this problem.

Ridge



Simulated Annealing:

A hill-climbing algorithm which never makes a move towards a lower value guaranteed to be incomplete because it can get stuck on a local maximum. And if algorithm applies a random walk, by moving a successor, then it may complete but not efficient. **Simulated Annealing** is an algorithm which yields both efficiency and completeness.

In mechanical term **Annealing** is a process of hardening a metal or glass to a high temperature then cooling gradually, so this allows the metal to reach a low-energy crystalline state. The same process is used in simulated annealing in which the algorithm picks a random move, instead of picking the best move. If the random move improves the state, then it follows the same path. Otherwise, the algorithm follows the path which has a probability of less than 1 or it moves downhill and chooses another path.

Adversarial Search

Adversarial search is a search, where we examine the problem which arises when we try to plan ahead of the world and other agents are planning against us.

- In previous topics, we have studied the search strategies which are only associated with a single agent that aims to find the solution which often expressed in the form of a sequence of actions.
- But, there might be some situations where more than one agent is searching for the solution in the same search space, and this situation usually occurs in game playing.
- The environment with more than one agent is termed as **multi-agent environment**, in which each agent is an opponent of other agent and playing against each other. Each agent needs to consider the action of other agent and effect of that action on their performance.
- So, **Searches in which two or more players with conflicting goals are trying to explore the same search space for the solution, are called adversarial searches, often known as Games.**
- Games are modeled as a Search problem and heuristic evaluation function, and these are the two main factors which help to model and solve games in AI.

	Deterministic	Chance Moves
Perfect information	Chess, Checkers, go, Othello	Backgammon, monopoly
Imperfect information	Battleships, blind, tic-tac-toe	Bridge, poker, scrabble, nuclear war

Types of Games in AI:

- **Perfect information:** A game with the perfect information is that in which agents can look into the complete board. Agents have all the information about

the game, and they can see each other moves also. Examples are Chess, Checkers, Go, etc.

- **Imperfect information:** If in a game agents do not have all information about the game and not aware with what's going on, such type of games are called the game with imperfect information, such as tic-tac-toe, Battleship, blind, Bridge, etc.
- **Deterministic games:** Deterministic games are those games which follow a strict pattern and set of rules for the games, and there is no randomness associated with them. Examples are chess, Checkers, Go, tic-tac-toe, etc.
- **Non-deterministic games:** Non-deterministic are those games which have various unpredictable events and has a factor of chance or luck. This factor of chance or luck is introduced by either dice or cards. These are random, and each action response is not fixed. Such games are also called as stochastic games. Example: Backgammon, Monopoly, Poker, etc.

Note: In this topic, we will discuss deterministic games, fully observable environment, zero-sum, and where each agent acts alternatively.

Mini-Max Algorithm in Artificial Intelligence

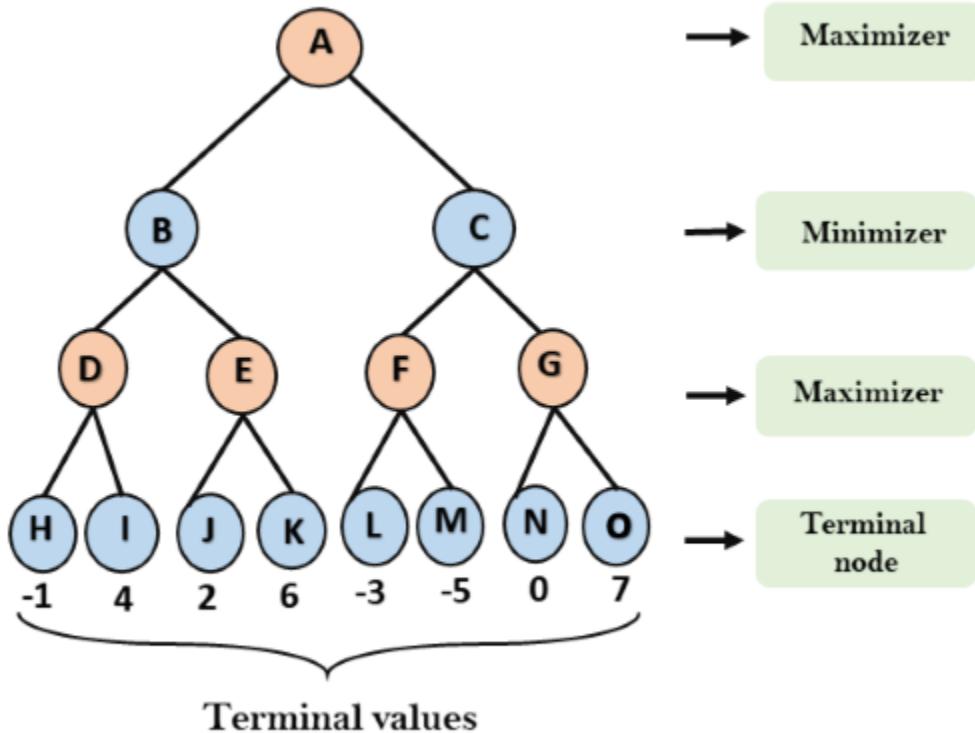
- Mini-max algorithm is a recursive or backtracking algorithm which is used in decision-making and game theory. It provides an optimal move for the player assuming that opponent is also playing optimally.
- Mini-Max algorithm uses recursion to search through the game-tree.
- Min-Max algorithm is mostly used for game playing in AI. Such as Chess, Checkers, tic-tac-toe, go, and various tow-players game. This Algorithm computes the minimax decision for the current state.
- In this algorithm two players play the game, one is called MAX and other is called MIN.
- Both the players fight it as the opponent player gets the minimum benefit while they get the maximum benefit.
- Both Players of the game are opponent of each other, where MAX will select the maximized value and MIN will select the minimized value.
- The minimax algorithm performs a depth-first search algorithm for the exploration of the complete game tree.

- The minimax algorithm proceeds all the way down to the terminal node of the tree, then backtrack the tree as the recursion.

Working of Min-Max Algorithm:

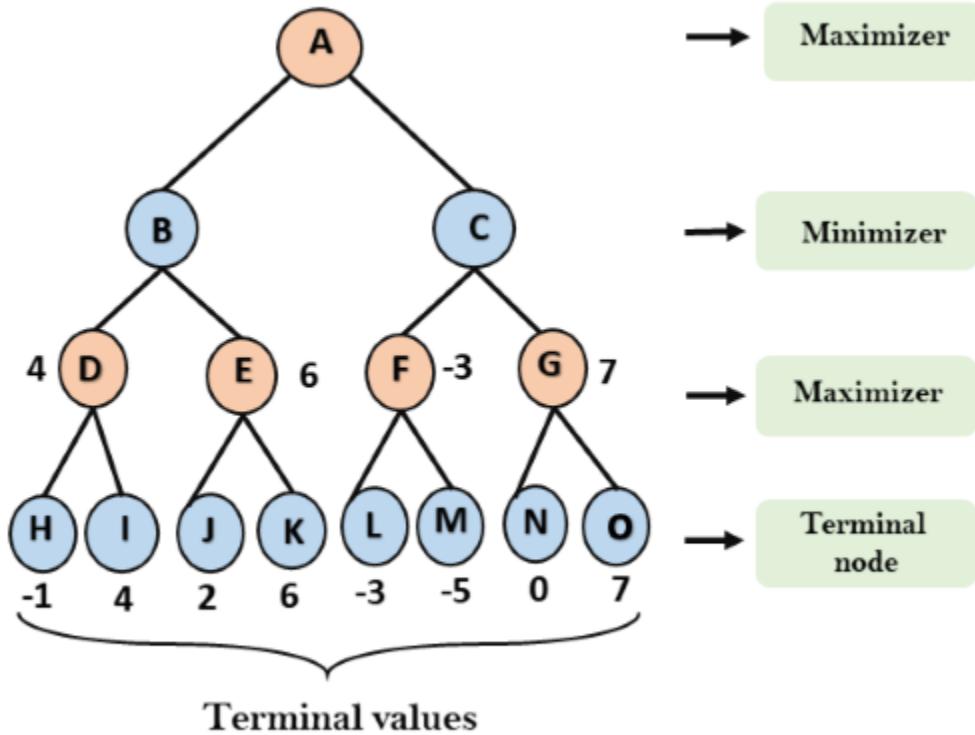
- The working of the minimax algorithm can be easily described using an example. Below we have taken an example of game-tree which is representing the two-player game.
- In this example, there are two players one is called Maximizer and other is called Minimizer.
- Maximizer will try to get the Maximum possible score, and Minimizer will try to get the minimum possible score.
- This algorithm applies DFS, so in this game-tree, we have to go all the way through the leaves to reach the terminal nodes.
- At the terminal node, the terminal values are given so we will compare those value and backtrack the tree until the initial state occurs. Following are the main steps involved in solving the two-player game tree:

Step-1: In the first step, the algorithm generates the entire game-tree and apply the utility function to get the utility values for the terminal states. In the below tree diagram, let's take A is the initial state of the tree. Suppose maximizer takes first turn which has worst-case initial value = -infinity, and minimizer will take next turn which has worst-case initial value = +infinity.



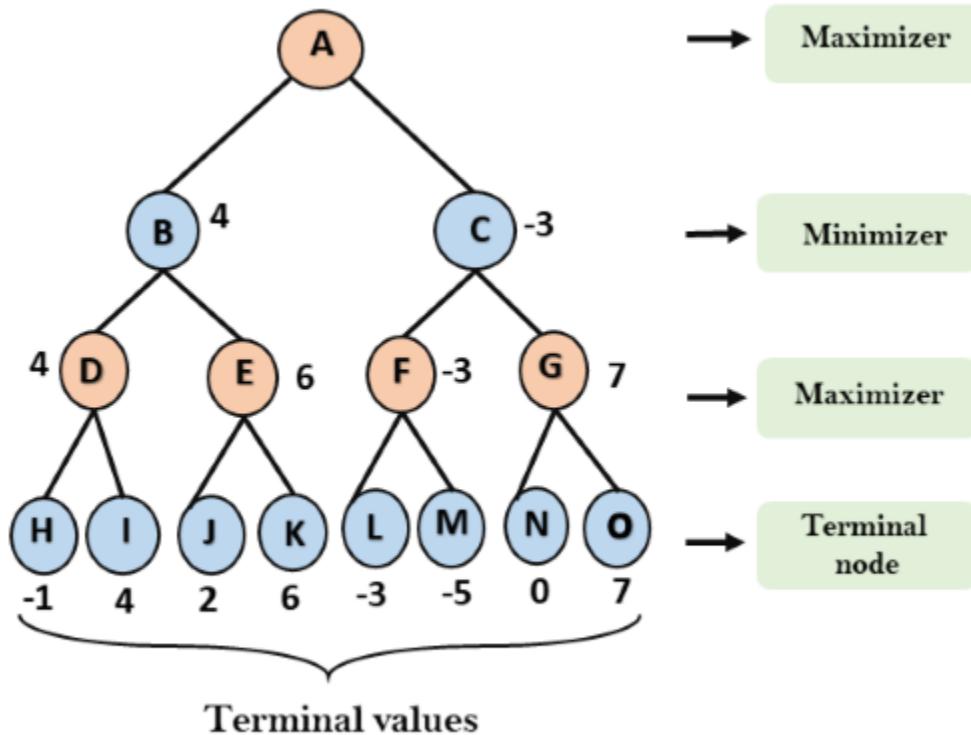
Step 2: Now, first we find the utilities value for the Maximizer, its initial value is $-\infty$, so we will compare each value in terminal state with initial value of Maximizer and determines the higher nodes values. It will find the maximum among the all.

- For node D $\max(-1, -\infty) \Rightarrow \max(-1, 4) = 4$
- For Node E $\max(2, -\infty) \Rightarrow \max(2, 6) = 6$
- For Node F $\max(-3, -\infty) \Rightarrow \max(-3, -5) = -3$
- For node G $\max(0, -\infty) = \max(0, 7) = 7$



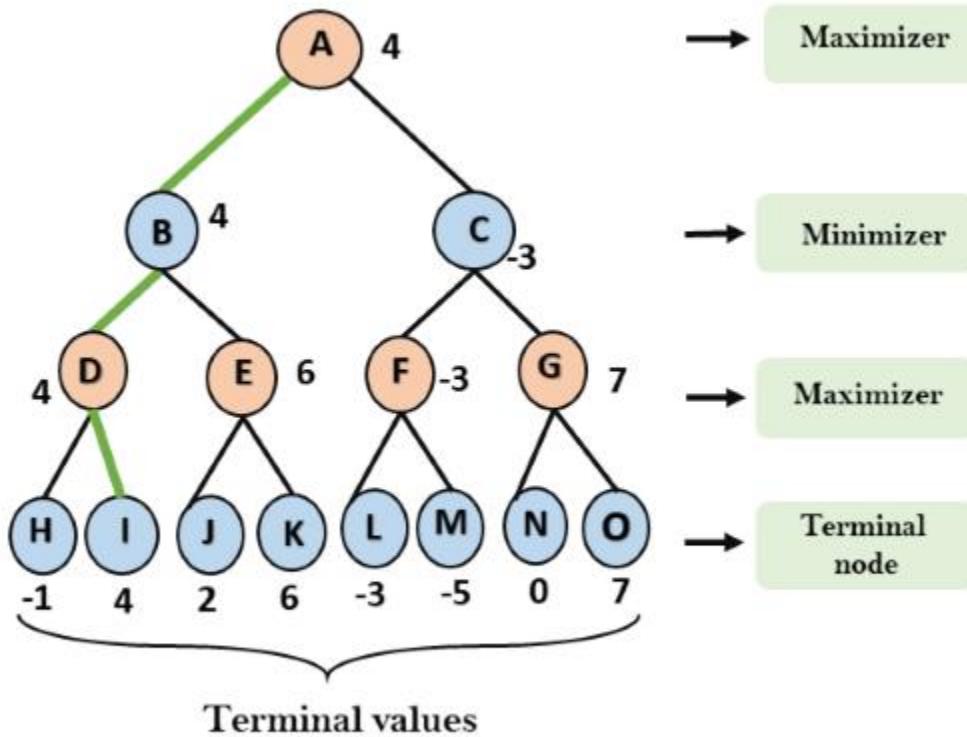
Step 3: In the next step, it's a turn for minimizer, so it will compare all nodes value with $+\infty$, and will find the 3rd layer node values.

- For node B= $\min(4,6) = 4$
- For node C= $\min (-3, 7) = -3$



Step 3: Now it's a turn for Maximizer, and it will again choose the maximum of all nodes value and find the maximum value for the root node. In this game tree, there are only 4 layers, hence we reach immediately to the root node, but in real games, there will be more than 4 layers.

- For node A $\max(4, -3) = 4$



That was the complete workflow of the minimax two player game.

Properties of Mini-Max algorithm:

- **Complete-** Min-Max algorithm is Complete. It will definitely find a solution (if exist), in the finite search tree.
- **Optimal-** Min-Max algorithm is optimal if both opponents are playing optimally.
- **Time complexity-** As it performs DFS for the game-tree, so the time complexity of Min-Max algorithm is $O(b^m)$, where b is branching factor of the game-tree, and m is the maximum depth of the tree.
- **Space Complexity-** Space complexity of Mini-max algorithm is also similar to DFS which is $O(bm)$.

Limitation of the minimax Algorithm:

The main drawback of the minimax algorithm is that it gets really slow for complex games such as Chess, go, etc. This type of games has a huge branching factor, and the

player has lots of choices to decide. This limitation of the minimax algorithm can be improved from **alpha-beta pruning** which we have discussed in the next topic.

Alpha-Beta Pruning

- Alpha-beta pruning is a modified version of the minimax algorithm. It is an optimization technique for the minimax algorithm.
- As we have seen in the minimax search algorithm that the number of game states it has to examine are exponential in depth of the tree. Since we cannot eliminate the exponent, but we can cut it to half. Hence there is a technique by which without checking each node of the game tree we can compute the correct minimax decision, and this technique is called **pruning**. This involves two threshold parameter Alpha and beta for future expansion, so it is called **alpha-beta pruning**. It is also called as **Alpha-Beta Algorithm**.
- Alpha-beta pruning can be applied at any depth of a tree, and sometimes it not only prune the tree leaves but also entire sub-tree.
- The two-parameter can be defined as:
 - a. **Alpha:** The best (highest-value) choice we have found so far at any point along the path of Maximizer. The initial value of alpha is **$-\infty$** .
 - b. **Beta:** The best (lowest-value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is **$+\infty$** .
- The Alpha-beta pruning to a standard minimax algorithm returns the same move as the standard algorithm does, but it removes all the nodes which are not really affecting the final decision but making algorithm slow. Hence by pruning these nodes, it makes the algorithm fast.

Note: To better understand this topic, kindly study the minimax algorithm.

Condition for Alpha-beta pruning:

The main condition which required for alpha-beta pruning is:

$$1. \quad \alpha >= \beta$$

Key points about alpha-beta pruning:

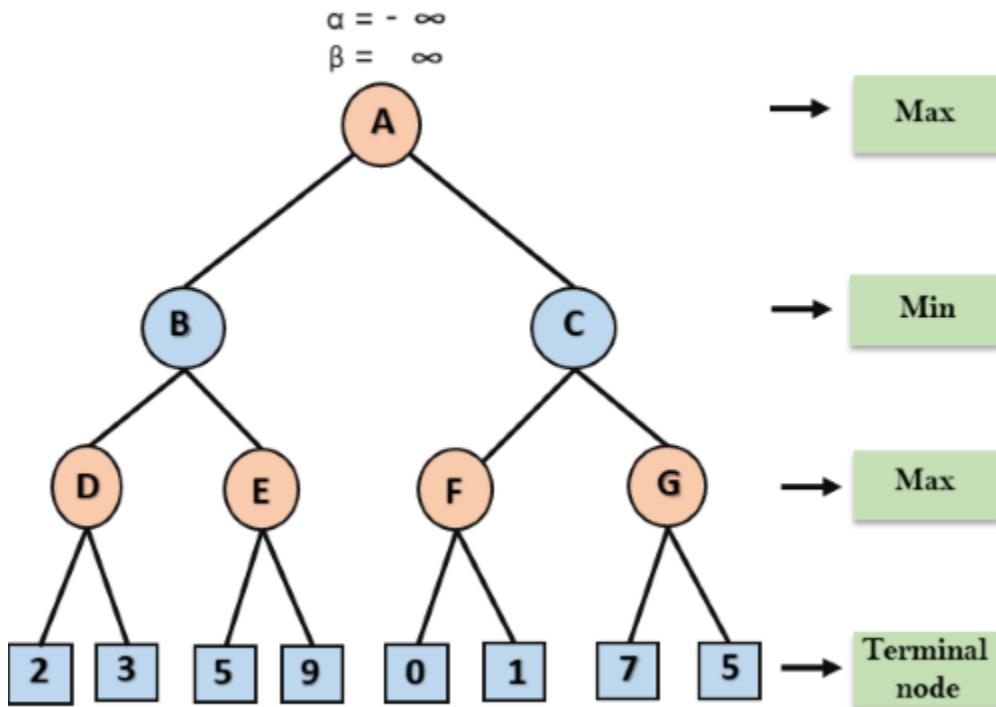
- The Max player will only update the value of alpha.

- The Min player will only update the value of beta.
- While backtracking the tree, the node values will be passed to upper nodes instead of values of alpha and beta.
- We will only pass the alpha, beta values to the child nodes.

Working of Alpha-Beta Pruning:

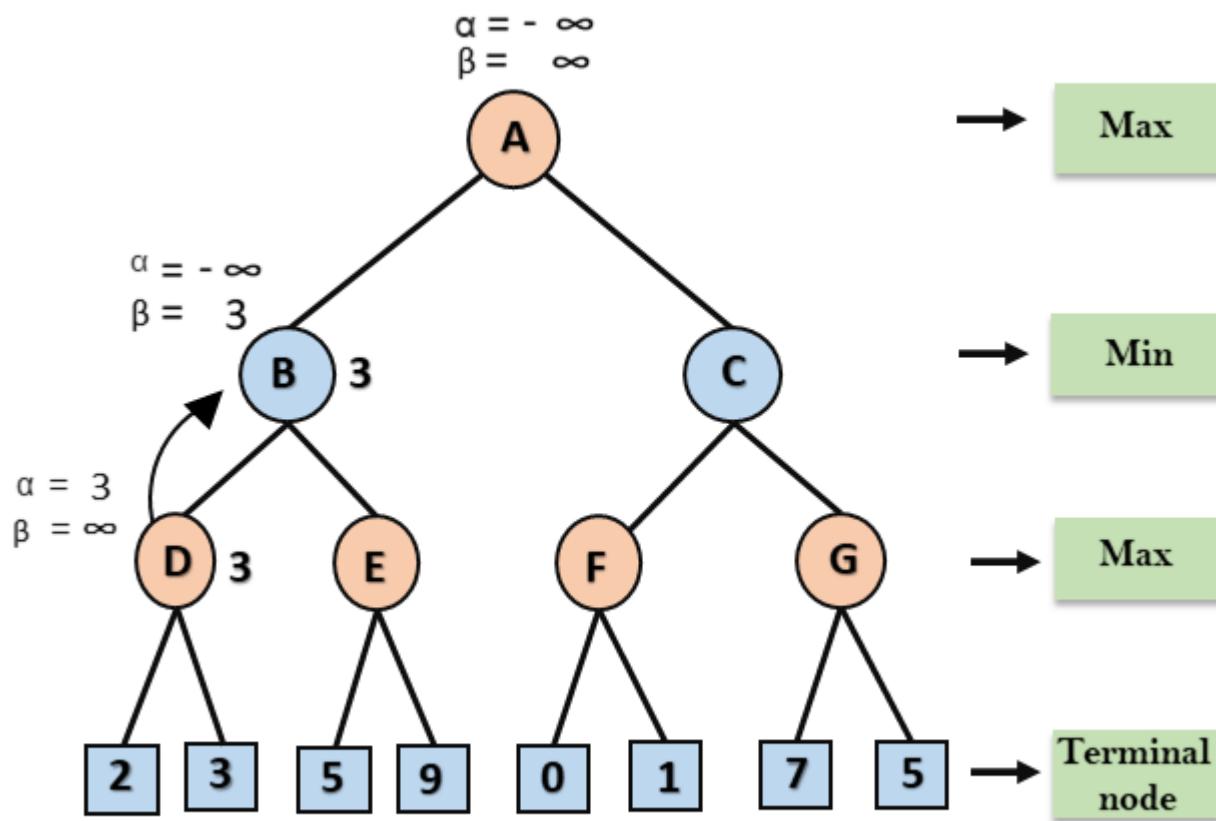
Let's take an example of two-player search tree to understand the working of Alpha-beta pruning

Step 1: At the first step the, Max player will start first move from node A where $\alpha = -\infty$ and $\beta = +\infty$, these value of alpha and beta passed down to node B where again $\alpha = -\infty$ and $\beta = +\infty$, and Node B passes the same value to its child D.



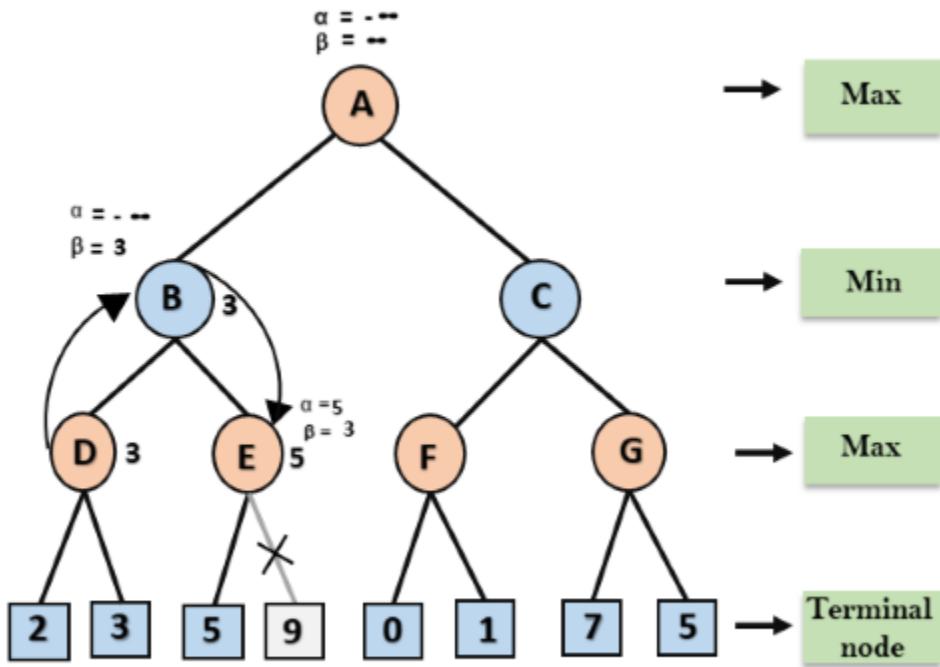
Step 2: At Node D, the value of α will be calculated as its turn for Max. The value of α is compared with firstly 2 and then 3, and the max (2, 3) = 3 will be the value of α at node D and node value will also 3.

Step 3: Now algorithm backtrack to node B, where the value of β will change as this is a turn of Min, Now $\beta = +\infty$, will compare with the available subsequent nodes value, i.e. $\min (\infty, 3) = 3$, hence at node B now $\alpha = -\infty$, and $\beta = 3$.



In the next step, algorithm traverse the next successor of Node B which is node E, and the values of $\alpha = -\infty$, and $\beta = 3$ will also be passed.

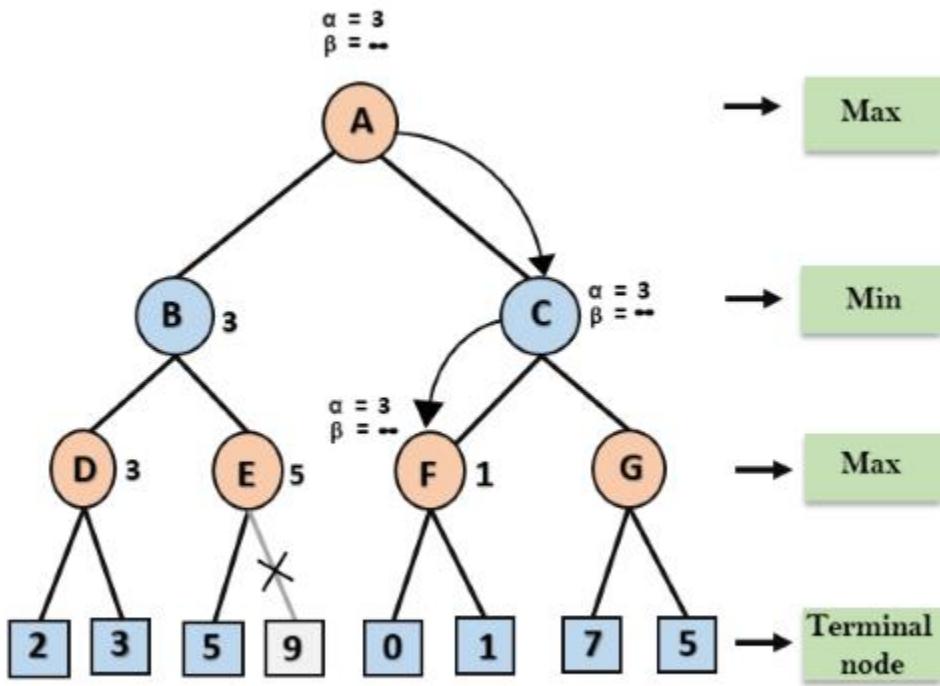
Step 4: At node E, Max will take its turn, and the value of alpha will change. The current value of alpha will be compared with 5, so $\max(-\infty, 5) = 5$, hence at node E $\alpha = 5$ and $\beta = 3$, where $\alpha \geq \beta$, so the right successor of E will be pruned, and algorithm will not traverse it, and the value at node E will be 5.



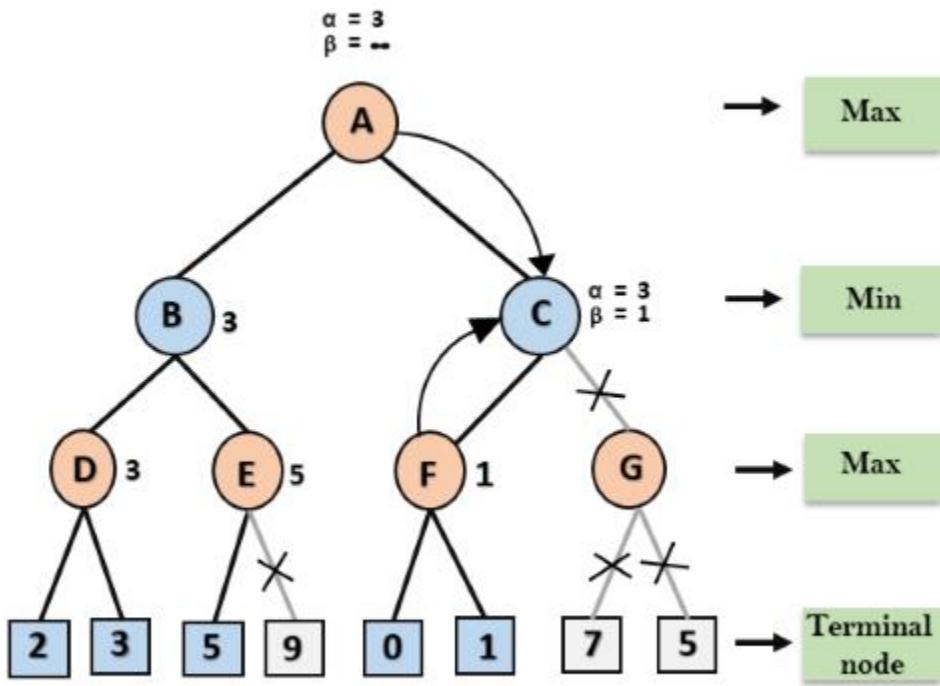
Step 5: At next step, algorithm again backtrack the tree, from node B to node A. At node A, the value of alpha will be changed the maximum available value is 3 as $\max(-\infty, 3) = 3$, and $\beta = +\infty$, these two values now passes to right successor of A which is Node C.

At node C, $\alpha = 3$ and $\beta = +\infty$, and the same values will be passed on to node F.

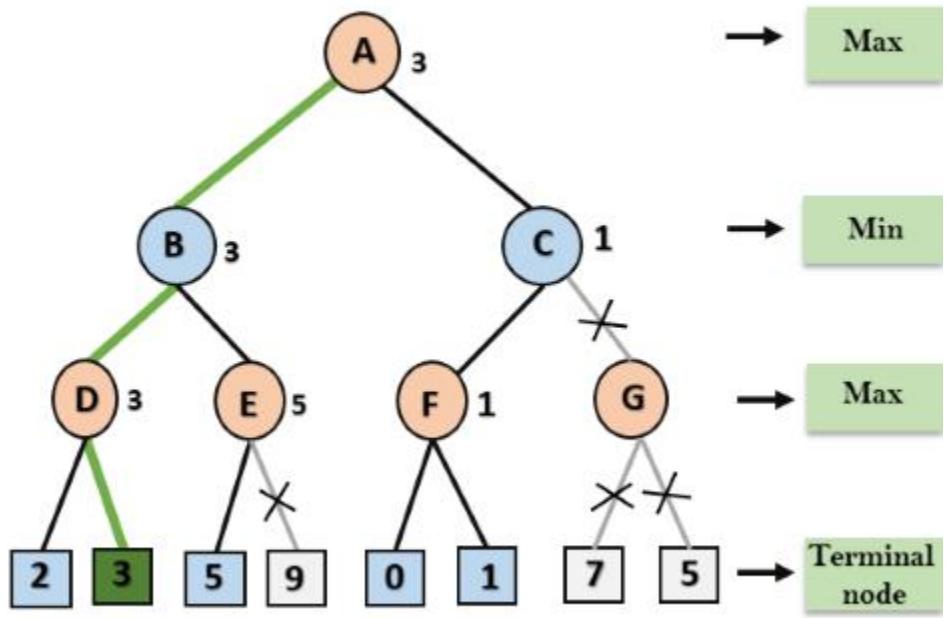
Step 6: At node F, again the value of α will be compared with left child which is 0, and $\max(3, 0) = 3$, and then compared with right child which is 1, and $\max(3, 1) = 3$ still α remains 3, but the node value of F will become 1.



Step 7: Node F returns the node value 1 to node C, at C $\alpha = 3$ and $\beta = +\infty$, here the value of beta will be changed, it will compare with 1 so $\min(\infty, 1) = 1$. Now at C, $\alpha = 3$ and $\beta = 1$, and again it satisfies the condition $\alpha \geq \beta$, so the next child of C which is G will be pruned, and the algorithm will not compute the entire sub-tree G.



Step 8: C now returns the value of 1 to A here the best value for A is max (3, 1) = 3. Following is the final game tree which is showing the nodes which are computed and nodes which has never computed. Hence the optimal value for the maximizer is 3 for this example.



What is knowledge representation?

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 and reasoning.** Hence we can describe Knowledge representation as following:

- Knowledge representation and reasoning (KR, KRR) is the part of Artificial intelligence which concerned with AI agents thinking and how thinking contributes to intelligent behavior of agents.
- It is responsible for representing information about the real world so that a computer can understand and can utilize this knowledge to solve the complex real world problems such as diagnosis a medical condition or communicating with humans in natural language.
- It is also a way which describes how we can represent knowledge in artificial intelligence. Knowledge representation is not just storing data into some database, but it also enables an intelligent machine to learn from that knowledge and experiences so that it can behave intelligently like a human.

What to Represent:

Following are the kind of knowledge which needs to be represented in AI systems:

- **Object:** All the facts about objects in our world domain. E.g., Guitars contains strings, trumpets are brass instruments.
- **Events:** Events are the actions which occur in our world.
- **Performance:** It describe behavior which involves knowledge about how to do things.
- **Meta-knowledge:** It is knowledge about what we know.
- **Facts:** Facts are the truths about the real world and what we represent.
- **Knowledge-Base:** The central component of the knowledge-based agents is the knowledge base. It is represented as KB. The Knowledgebase is a group of the Sentences (Here, sentences are used as a technical term and not identical with the English language).

Knowledge: Knowledge is awareness or familiarity gained by experiences of facts, data, and situations. Following are the types of knowledge in artificial intelligence:

Types of knowledge

Following are the various types of knowledge:



1. Declarative Knowledge:

- Declarative knowledge is to know about something.
- It includes concepts, facts, and objects.
- It is also called descriptive knowledge and expressed in declarative sentences.
- It is simpler than procedural language.

2. Procedural Knowledge

- It is also known as imperative knowledge.
- Procedural knowledge is a type of knowledge which is responsible for knowing how to do something.
- It can be directly applied to any task.
- It includes rules, strategies, procedures, agendas, etc.

- Procedural knowledge depends on the task on which it can be applied.

3. Meta-knowledge:

- Knowledge about the other types of knowledge is called Meta-knowledge.

4. Heuristic knowledge:

- Heuristic knowledge is representing knowledge of some experts in a filed or subject.
- Heuristic knowledge is rules of thumb based on previous experiences, awareness of approaches, and which are good to work but not guaranteed.

5. Structural knowledge:

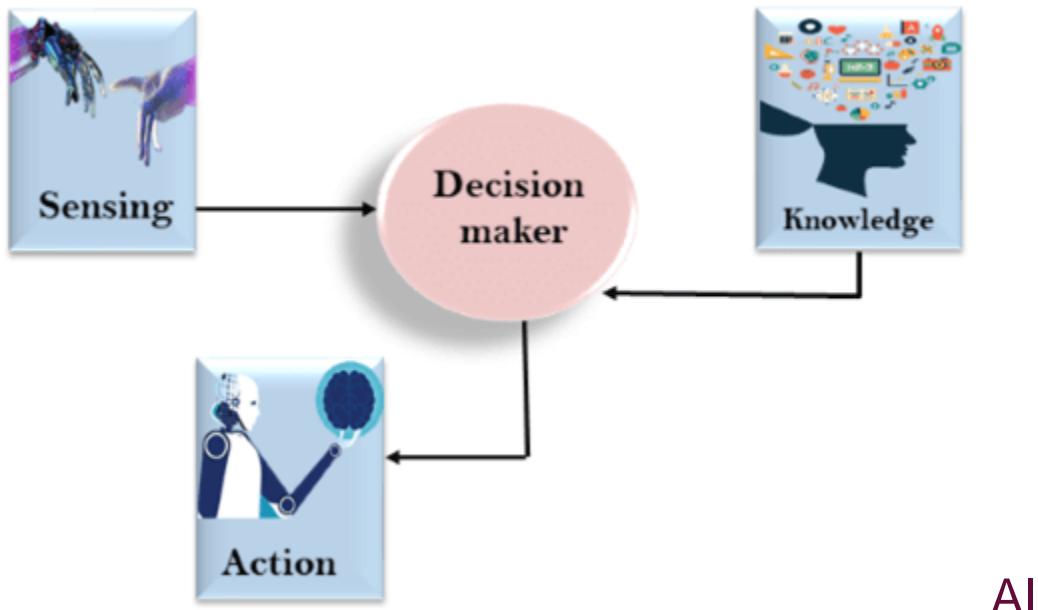
- Structural knowledge is basic knowledge to problem-solving.
- It describes relationships between various concepts such as kind of, part of, and grouping of something.
- It describes the relationship that exists between concepts or objects.

The relation between knowledge and intelligence:

Knowledge of real-worlds plays a vital role in intelligence and same for creating artificial intelligence. Knowledge plays an important role in demonstrating intelligent behavior in AI agents. An agent is only able to accurately act on some input when he has some knowledge or experience about that input.

Let's suppose if you met some person who is speaking in a language which you don't know, then how you will able to act on that. The same thing applies to the intelligent behavior of the agents.

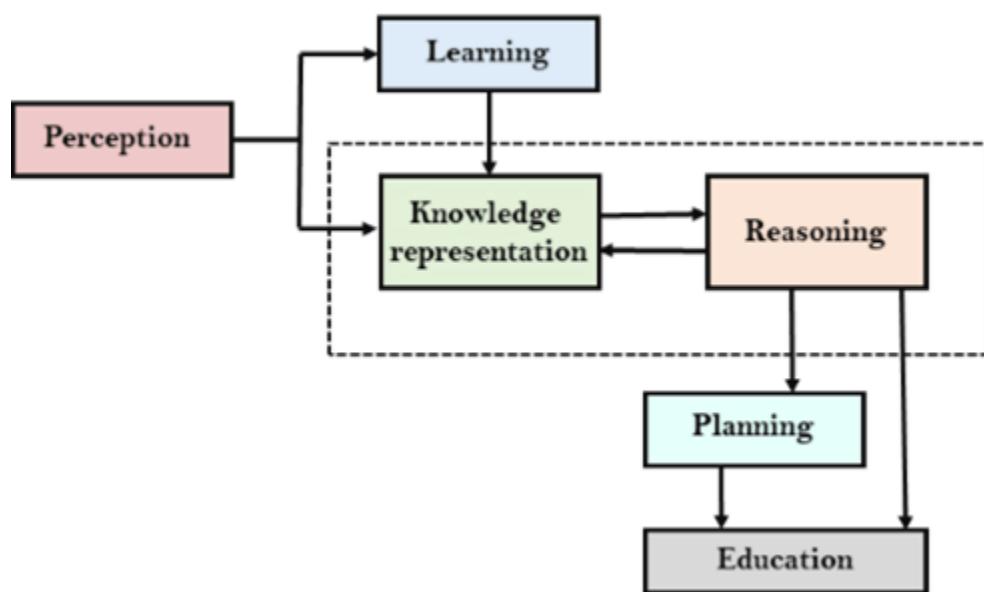
As we can see in below diagram, there is one decision maker which act by sensing the environment and using knowledge. But if the knowledge part will not present then, it cannot display intelligent behavior.



AI knowledge cycle:

An Artificial intelligence system has the following components for displaying intelligent behavior:

- Perception
- Learning
- Knowledge Representation and Reasoning
- Planning
- Execution



The above diagram is showing how an AI system can interact with the real world and what components help it to show intelligence. AI system has Perception component by which it retrieves information from its environment. It can be visual, audio or another form of sensory input. The learning component is responsible for learning from data captured by Perception component. In the complete cycle, the main components are knowledge representation and Reasoning. These two components are involved in showing the intelligence in machine-like humans. These two components are independent with each other but also coupled together. The planning and execution depend on analysis of Knowledge representation and reasoning.

Approaches to knowledge representation:

There are mainly four approaches to knowledge representation, which are given below:

1. Simple relational knowledge:

- It is the simplest way of storing facts which uses the relational method, and each fact about a set of the object is set out systematically in columns.
- This approach of knowledge representation is famous in database systems where the relationship between different entities is represented.
- This approach has little opportunity for inference.

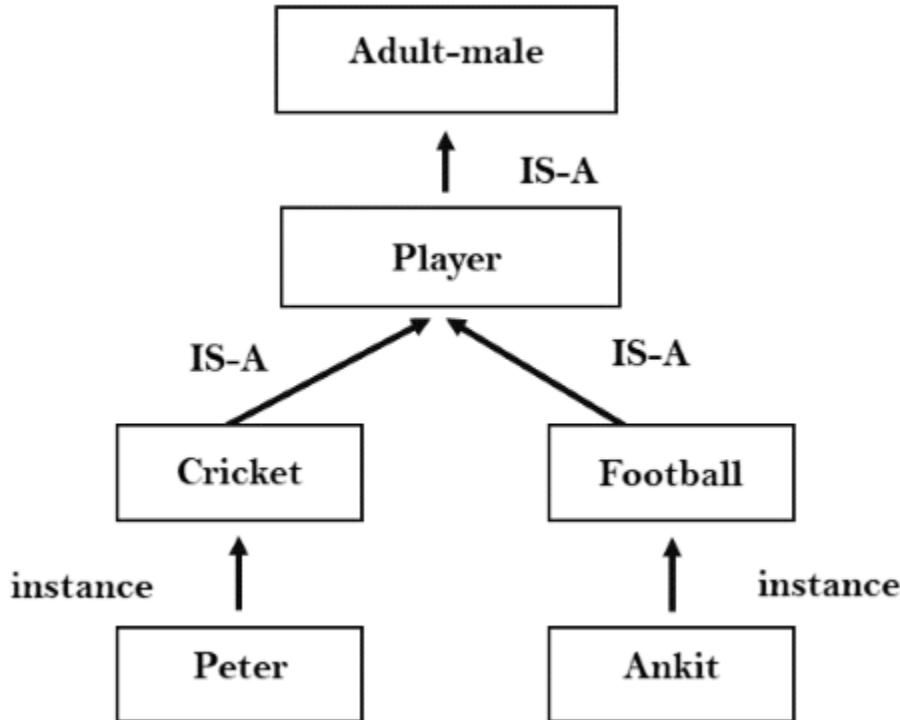
Example: The following is the simple relational knowledge representation.

Player	Weight	Age
Player1	65	23
Player2	58	18
Player3	75	24

2. Inheritable knowledge:

- In the inheritable knowledge approach, all data must be stored into a hierarchy of classes.
- All classes should be arranged in a generalized form or a hierachal manner.
- In this approach, we apply inheritance property.
- Elements inherit values from other members of a class.

- This approach contains inheritable knowledge which shows a relation between instance and class, and it is called instance relation.
- Every individual frame can represent the collection of attributes and its value.
- In this approach, objects and values are represented in Boxed nodes.
- We use Arrows which point from objects to their values.
- **Example:**



3. Inferential knowledge:

- Inferential knowledge approach represents knowledge in the form of formal logics.
- This approach can be used to derive more facts.
- It guaranteed correctness.
- **Example:** Let's suppose there are two statements:
 - a. Marcus is a man
 - b. All men are mortal
 Then it can represent as;

man(Marcus)
 $\forall x = \text{man}(x) \rightarrow \text{mortal}(x)$

4. Procedural knowledge:

- Procedural knowledge approach uses small programs and codes which describes how to do specific things, and how to proceed.
- In this approach, one important rule is used which is **If-Then rule**.
- In this knowledge, we can use various coding languages such as **LISP language** and **Prolog language**.
- We can easily represent heuristic or domain-specific knowledge using this approach.
- But it is not necessary that we can represent all cases in this approach.

Requirements for knowledge Representation system:

A good knowledge representation system must possess the following properties.

1. **1. Representational Accuracy:**

KR system should have the ability to represent all kind of required knowledge.

2. **2. Inferential Adequacy:**

KR system should have ability to manipulate the representational structures to produce new knowledge corresponding to existing structure.

3. **3. Inferential Efficiency:**

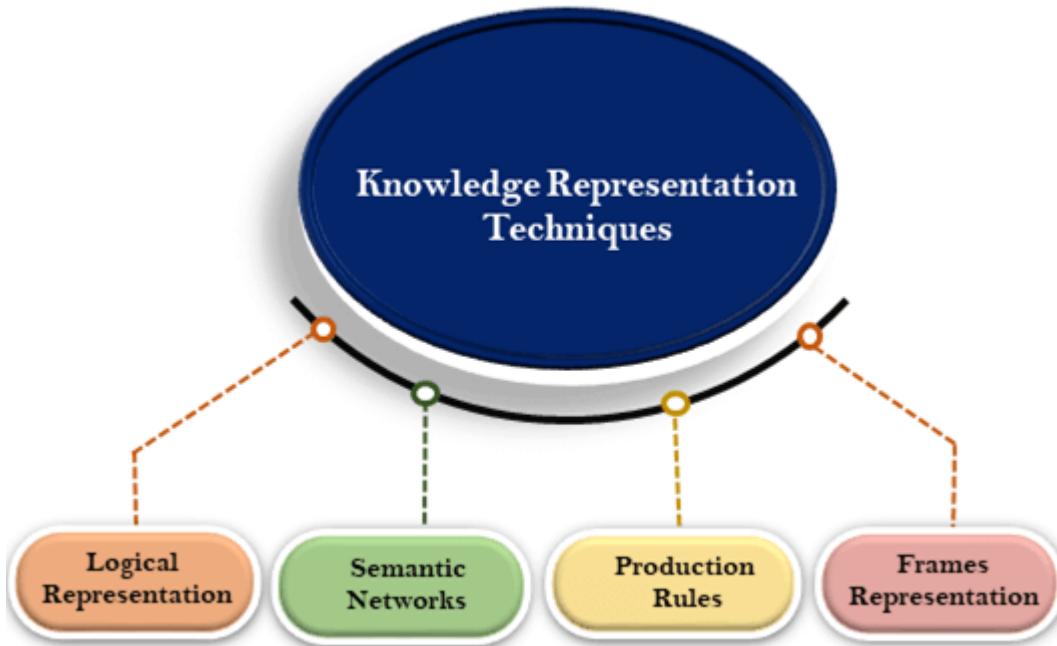
The ability to direct the inferential knowledge mechanism into the most productive directions by storing appropriate guides.

4. **4. Acquisitional efficiency-** The ability to acquire the new knowledge easily using automatic methods.

Techniques of knowledge representation

There are mainly four ways of knowledge representation which are given as follows:

1. Logical Representation
2. Semantic Network Representation
3. Frame Representation
4. Production Rules



1. Logical Representation

Logical representation is a language with some concrete rules which deals with propositions and has no ambiguity in representation. Logical representation means drawing a conclusion based on various conditions. This representation lays down some important communication rules. It consists of precisely defined syntax and semantics which supports the sound inference. Each sentence can be translated into logics using syntax and semantics.

Syntax:

- Syntaxes are the rules which decide how we can construct legal sentences in the logic.
- It determines which symbol we can use in knowledge representation.
- How to write those symbols.

Semantics:

- Semantics are the rules by which we can interpret the sentence in the logic.
- Semantic also involves assigning a meaning to each sentence.

Logical representation can be categorised into mainly two logics:

- a. Propositional Logics
- b. Predicate logics

Advantages of logical representation:

1. Logical representation enables us to do logical reasoning.
2. Logical representation is the basis for the programming languages.

Disadvantages of logical Representation:

1. Logical representations have some restrictions and are challenging to work with.
2. Logical representation technique may not be very natural, and inference may not be so efficient.

Note: Do not be confused with logical representation and logical reasoning as logical representation is a representation language and reasoning is a process of thinking logically.

2. Semantic Network Representation

Semantic networks are alternative of predicate logic for knowledge representation. In Semantic networks, we can represent our knowledge in the form of graphical networks. This network consists of nodes representing objects and arcs which describe the relationship between those objects. Semantic networks can categorize the object in different forms and can also link those objects. Semantic networks are easy to understand and can be easily extended.

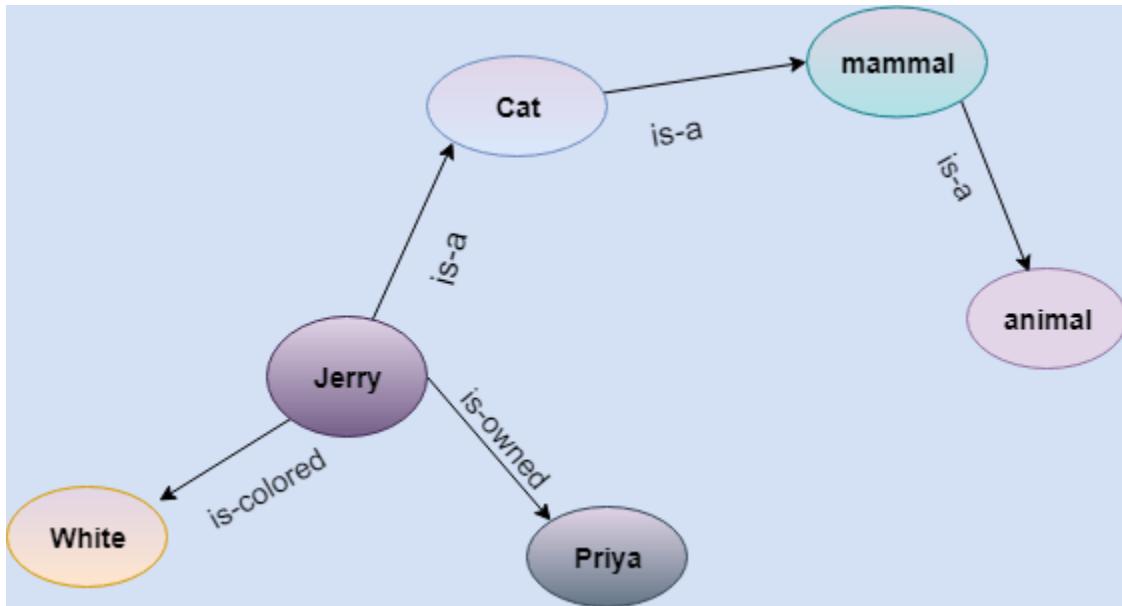
This representation consist of mainly two types of relations:

- a. IS-A relation (Inheritance)
- b. Kind-of-relation

Example: Following are some statements which we need to represent in the form of nodes and arcs.

Statements:

- a. Jerry is a cat.
- b. Jerry is a mammal
- c. Jerry is owned by Priya.
- d. Jerry is brown colored.
- e. All Mammals are animal.



In the above diagram, we have represented the different type of knowledge in the form of nodes and arcs. Each object is connected with another object by some relation.

Drawbacks in Semantic representation:

1. Semantic networks take more computational time at runtime as we need to traverse the complete network tree to answer some questions. It might be possible in the worst case scenario that after traversing the entire tree, we find that the solution does not exist in this network.
2. Semantic networks try to model human-like memory (Which has 1015 neurons and links) to store the information, but in practice, it is not possible to build such a vast semantic network.
3. These types of representations are inadequate as they do not have any equivalent quantifier, e.g., for all, for some, none, etc.
4. Semantic networks do not have any standard definition for the link names.
5. These networks are not intelligent and depend on the creator of the system.

Advantages of Semantic network:

1. Semantic networks are a natural representation of knowledge.
2. Semantic networks convey meaning in a transparent manner.
3. These networks are simple and easily understandable.

3. Frame Representation

A frame is a record like structure which consists of a collection of attributes and its values to describe an entity in the world. Frames are the AI data structure which divides knowledge into substructures by representing stereotypes situations. It consists of a collection of slots and slot values. These slots may be of any type and sizes. Slots have names and values which are called facets.

Facets: The various aspects of a slot is known as **Facets**. Facets are features of frames which enable us to put constraints on the frames. Example: IF-NEEDED facts are called when data of any particular slot is needed. A frame may consist of any number of slots, and a slot may include any number of facets and facets may have any number of values. A frame is also known as **slot-filter knowledge representation** in artificial intelligence.

Frames are derived from semantic networks and later evolved into our modern-day classes and objects. A single frame is not much useful. Frames system consist of a collection of frames which are connected. In the frame, knowledge about an object or event can be stored together in the knowledge base. The frame is a type of technology which is widely used in various applications including Natural language processing and machine visions.

Example: 1

Let's take an example of a frame for a book

Slots	Filters
Title	Artificial Intelligence
Genre	Computer Science
Author	Peter Norvig
Edition	Third Edition
Year	1996
Page	1152

Example 2:

Let's suppose we are taking an entity, Peter. Peter is an engineer as a profession, and his age is 25, he lives in city London, and the country is England. So following is the frame representation for this:

Slots	Filter
Name	Peter
Profession	Doctor
Age	25
Marital status	Single
Weight	78

Advantages of frame representation:

1. The frame knowledge representation makes the programming easier by grouping the related data.
2. The frame representation is comparably flexible and used by many applications in AI.
3. It is very easy to add slots for new attribute and relations.
4. It is easy to include default data and to search for missing values.
5. Frame representation is easy to understand and visualize.

Disadvantages of frame representation:

1. In frame system inference mechanism is not be easily processed.
2. Inference mechanism cannot be smoothly proceeded by frame representation.
3. Frame representation has a much generalized approach.

4. Production Rules

Production rules system consist of (**condition, action**) pairs which mean, "If condition then action". It has mainly three parts:

- The set of production rules
- Working Memory
- The recognize-act-cycle

In production rules agent checks for the condition and if the condition exists then production rule fires and corresponding action is carried out. The condition part of the rule determines which rule may be applied to a problem. And the action part carries out the associated problem-solving steps. This complete process is called a recognize-act cycle.

The working memory contains the description of the current state of problems-solving and rule can write knowledge to the working memory. This knowledge match and may fire other rules.

If there is a new situation (state) generates, then multiple production rules will be fired together, this is called conflict set. In this situation, the agent needs to select a rule from these sets, and it is called a conflict resolution.

Example:

- **IF (at bus stop AND bus arrives) THEN action (get into the bus)**
- **IF (on the bus AND paid AND empty seat) THEN action (sit down).**
- **IF (on bus AND unpaid) THEN action (pay charges).**
- **IF (bus arrives at destination) THEN action (get down from the bus).**

Advantages of Production rule:

1. The production rules are expressed in natural language.
2. The production rules are highly modular, so we can easily remove, add or modify an individual rule.

Disadvantages of Production rule:

1. Production rule system does not exhibit any learning capabilities, as it does not store the result of the problem for the future uses.
2. During the execution of the program, many rules may be active hence rule-based production systems are inefficient.

Propositional logic in Artificial intelligence

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

Example:

1. a) It is Sunday.
2. b) The Sun rises from West (False proposition)
3. c) $3+3= 7$ (False proposition)
4. d) 5 is a prime number.

Following are some basic facts about propositional logic:

- o Propositional logic is also called Boolean logic as it works on 0 and 1.
- o In propositional logic, we use symbolic variables to represent the logic, and we can use any symbol for representing a proposition, such A, B, C, P, Q, R, etc.
- o Propositions can be either true or false, but it cannot be both.
- o Propositional logic consists of an object, relations or function, and **logical connectives**.
- o These connectives are also called logical operators.
- o The propositions and connectives are the basic elements of the propositional logic.
- o Connectives can be said as a logical operator which connects two sentences.
- o A proposition formula which is always true is called **tautology**, and it is also called a valid sentence.
- o A proposition formula which is always false is called **Contradiction**.
- o A proposition formula which has both true and false values is called
- o Statements which are questions, commands, or opinions are not propositions such as "**Where is Rohini**", "**How are you**", "**What is your name**", are not propositions.

Syntax of propositional logic:

The syntax of propositional logic defines the allowable sentences for the knowledge representation. There are two types of Propositions:

a. **Atomic Propositions**

b. **Compound propositions**

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

Example:

1. a) **2+2 is 4**, it is an atomic proposition as it is a **true** fact.
 2. b) **"The Sun is cold"** is also a proposition as it is a **false** fact.
- o **Compound proposition:** Compound propositions are constructed by combining simpler or atomic propositions, using parenthesis and logical connectives.

Example:

1. a) **"It is raining today, and street is wet."**
2. b) **"Ankit is a doctor, and his clinic is in Mumbai."**

Logical Connectives:

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

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

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

Example: "Ritika is a doctor or Engineer",

Here P= Ritika is Doctor. Q= Ritika is Doctor, 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 $P \Leftrightarrow Q$.

Following is the summarized table for Propositional Logic Connectives:

Connective symbols	Word	Technical term	Example
\wedge	AND	Conjunction	$A \wedge B$
\vee	OR	Disjunction	$A \vee B$
\rightarrow	Implies	Implication	$A \rightarrow B$
\Leftrightarrow	If and only if	Biconditional	$A \Leftrightarrow B$
\neg or \sim	Not	Negation	$\neg A$ or $\sim B$

Truth Table:

In propositional logic, we need to know the truth values of propositions in all possible scenarios. We can combine all the possible combination with logical connectives, and the representation of these combinations in a tabular format is called **Truth table**. Following are the truth table for all logical connectives:

For Negation:

P	$\neg P$
True	False
False	True

For Conjunction:

P	Q	$P \wedge Q$
True	True	True
True	False	False
False	True	False
False	False	False

For disjunction:

P	Q	$P \vee Q$
True	True	True
False	True	True
True	False	True
False	False	False

For Implication:

P	Q	$P \rightarrow Q$
True	True	True
True	False	False
False	True	True
False	False	True

For Biconditional:

P	Q	$P \leftrightarrow Q$
True	True	True
True	False	False
False	True	False
False	False	True

Truth table with three propositions:

We can build a proposition composing three propositions P, Q, and R. This truth table is made-up of 8^n Tuples as we have taken three proposition symbols.

P	Q	R	$\neg R$	$P \vee Q$	$P \vee Q \rightarrow \neg R$
True	True	True	False	True	False
True	True	False	True	True	True
True	False	True	False	True	False
True	False	False	True	True	True
False	True	True	False	True	False
False	True	False	True	True	True
False	False	True	False	False	True
False	False	False	True	False	True

Precedence of connectives:

Just like arithmetic operators, there is a precedence order for propositional connectors or logical operators. This order should be followed while evaluating a propositional problem. Following is the list of the precedence order for operators:

Precedence	Operators
First Precedence	Parenthesis
Second Precedence	Negation
Third Precedence	Conjunction(AND)
Fourth Precedence	Disjunction(OR)
Fifth Precedence	Implication
Six Precedence	Biconditional

Note: For better understanding use parenthesis to make sure of the correct interpretations. Such as $\neg R \vee Q$, It can be interpreted as $(\neg R) \vee Q$.

Logical equivalence:

Logical equivalence is one of the features of propositional logic. Two propositions are said to be logically equivalent if and only if the columns in the truth table are identical to each other.

Let's take two propositions A and B, so for logical equivalence, we can write it as $A \Leftrightarrow B$. In below truth table we can see that column for $\neg A \vee B$ and $A \rightarrow B$, are identical hence A is Equivalent to B

A	B	$\neg A$	$\neg A \vee B$	$A \rightarrow B$
T	T	F	T	T
T	F	F	F	F
F	T	T	T	T
F	F	T	T	T

Properties of Operators:

- **Commutativity:**
 - $P \wedge Q = Q \wedge P$, or
 - $P \vee Q = Q \vee P$.
- **Associativity:**
 - $(P \wedge Q) \wedge R = P \wedge (Q \wedge R)$,
 - $(P \vee Q) \vee R = P \vee (Q \vee R)$
- **Identity element:**
 - $P \wedge \text{True} = P$,
 - $P \vee \text{True} = \text{True}$.
- **Distributive:**
 - $P \wedge (Q \vee R) = (P \wedge Q) \vee (P \wedge R)$.
 - $P \vee (Q \wedge R) = (P \vee Q) \wedge (P \vee R)$.
- **DE Morgan's Law:**
 - $\neg (P \wedge Q) = (\neg P) \vee (\neg Q)$
 - $\neg (P \vee Q) = (\neg P) \wedge (\neg Q)$.
- **Double-negation elimination:**
 - $\neg (\neg P) = P$.

Limitations of Propositional logic:

- We cannot represent relations like ALL, some, or none with propositional logic.
- Example:
- a. **All the girls are intelligent.**
 - b. **Some apples are sweet.**

- Propositional logic has limited expressive power.
- In propositional logic, we cannot describe statements in terms of their properties or logical relationships.

First-Order Logic in Artificial intelligence

In the topic of Propositional logic, we have seen that how to represent statements using propositional logic. But unfortunately, 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."**

To represent the above statements, PL logic is not sufficient, so we required some more powerful logic, such as first-order logic.

First-Order logic:

- First-order logic is another way of knowledge representation in artificial intelligence. It is an extension to propositional logic.
- FOL is sufficiently expressive to represent the natural language statements in a concise way.
- First-order logic is also known as **Predicate logic or First-order predicate logic**. First-order logic is a powerful language that develops information about the objects in a more easy way and can also express the relationship between those objects.
- First-order logic (like natural language) does not only assume that the world contains facts like propositional logic but also assumes the following things in the world:
 - **Objects:** A, B, people, numbers, colors, wars, theories, squares, pits, wumpus,
 - **Relations: It can be unary relation such as:** red, round, is adjacent, **or n-any relation such as:** the sister of, brother of, has color, comes between
 - **Function:** Father of, best friend, third inning of, end of,

- As a natural language, first-order logic also has two main parts:
 - Syntax**
 - Semantics**

Syntax of First-Order logic:

The syntax of FOL determines which collection of symbols is a logical expression in first-order logic. The basic syntactic elements of first-order logic are symbols. We write statements in short-hand notation in FOL.

Basic Elements of First-order logic:

Following are the basic elements of FOL syntax:

Constant	1, 2, A, John, Mumbai, cat,....
Variables	x, y, z, a, b,....
Predicates	Brother, Father, >,....
Function	sqrt, LeftLegOf,
Connectives	\wedge , \vee , \neg , \Rightarrow , \Leftrightarrow
Equality	$=$
Quantifier	\forall , \exists

Atomic sentences:

- Atomic sentences are the most basic sentences of first-order logic. These sentences are formed from a predicate symbol followed by a parenthesis with a sequence of terms.
- We can represent atomic sentences as **Predicate (term1, term2,, term n)**.

**Example: Ravi and Ajay are brothers: => Brothers(Ravi, Ajay).
Chinky is a cat: => cat (Chinky).**

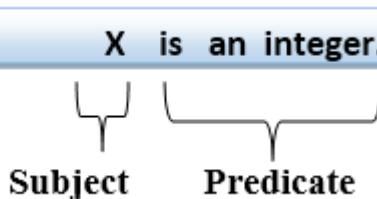
Complex Sentences:

- Complex sentences are made by combining atomic sentences using connectives.

First-order logic statements can be divided into two parts:

- Subject:** Subject is the main part of the statement.
- Predicate:** A predicate can be defined as a relation, which binds two atoms together in a statement.

Consider the statement: "x is an integer.", it consists of two parts, the first part x is the subject of the statement and second part "is an integer," is known as a predicate.



Quantifiers in First-order logic:

- A quantifier is a language element which generates quantification, and quantification specifies the quantity of specimen in the universe of discourse.
- These are the symbols that permit to determine or identify the range and scope of the variable in the logical expression. There are two types of quantifier:
 - Universal Quantifier, (for all, everyone, everything)**
 - Existential quantifier, (for some, at least one).**

Universal Quantifier:

Universal quantifier is a symbol of logical representation, which specifies that the statement within its range is true for everything or every instance of a particular thing.

The Universal quantifier is represented by a symbol \forall , which resembles an inverted A.

Note: In universal quantifier we use implication " \rightarrow ".

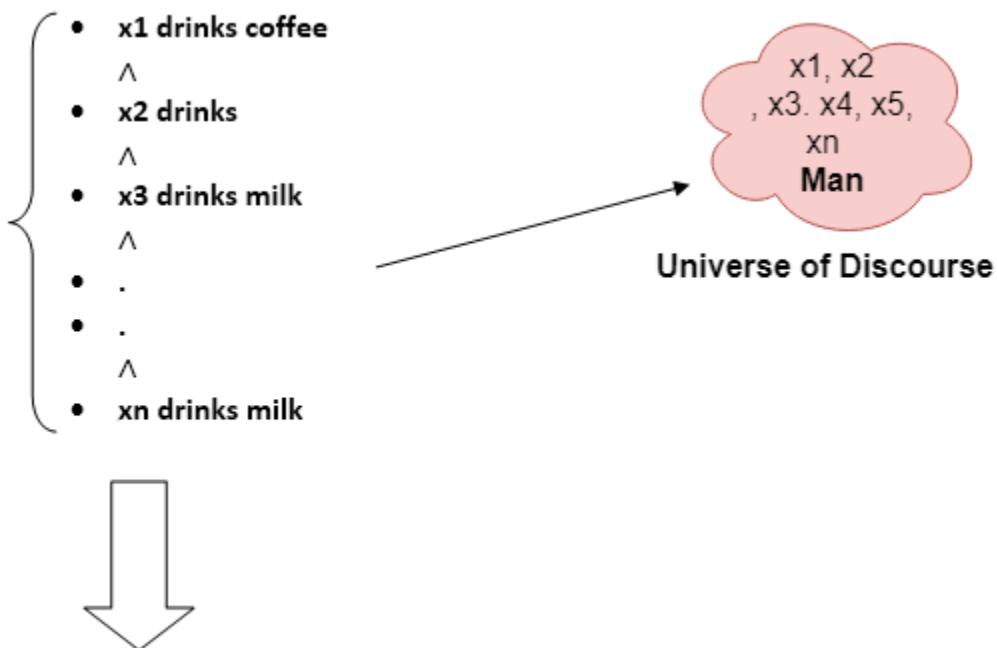
If x is a variable, then $\forall x$ is read as:

- **For all x**
- **For each x**
- **For every x.**

Example:

All man drink coffee.

Let a variable x which refers to a cat so all x can be represented in UOD as below:



So in shorthand notation, we can write it as :

$\forall x \text{ man}(x) \rightarrow \text{drink}(x, \text{coffee})$.

It will be read as: There are all x where x is a man who drink coffee.

Existential Quantifier:

Existential quantifiers are the type of quantifiers, which express that the statement within its scope is true for at least one instance of something.

It is denoted by the logical operator \exists , which resembles as inverted E. When it is used with a predicate variable then it is called as an existential quantifier.

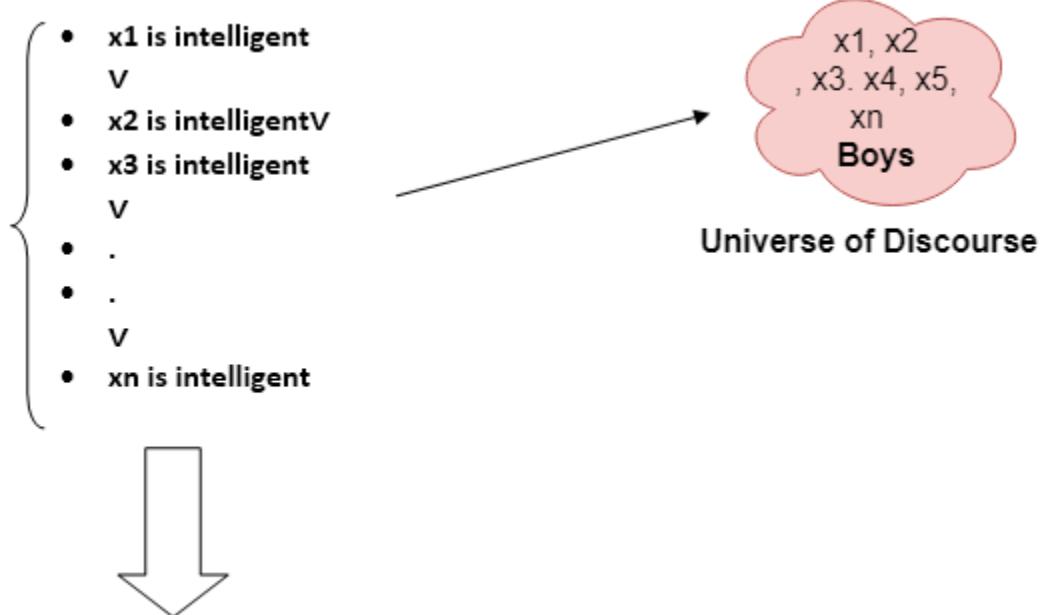
Note: In Existential quantifier we always use AND or Conjunction symbol (\wedge).

If x is a variable, then existential quantifier will be $\exists x$ or $\exists(x)$. And it will be read as:

- **There exists a 'x.'**
- **For some 'x.'**
- **For at least one 'x.'**

Example:

Some boys are intelligent.



So in short-hand notation, we can write it as:

$\exists x: \text{boys}(x) \wedge \text{intelligent}(x)$

It will be read as: There are some x where x is a boy who is intelligent.

Points to remember:

- The main connective for universal quantifier \forall is implication \rightarrow .
- The main connective for existential quantifier \exists is and \wedge .

Properties of Quantifiers:

- In universal quantifier, $\forall x \forall y$ is similar to $\forall y \forall x$.
- In Existential quantifier, $\exists x \exists y$ is similar to $\exists y \exists x$.
- $\exists x \forall y$ is not similar to $\forall y \exists x$.

Some Examples of FOL using quantifier:

1. All birds fly.

In this question the predicate is "**fly(bird)**."

And since there are all birds who fly so it will be represented as follows.

$$\forall x \text{ bird}(x) \rightarrow \text{fly}(x).$$

2. Every man respects his parent.

In this question, the predicate is "**respect(x, y)**," **where x=man, and y= parent.**

Since there is every man so will use \forall , and it will be represented as follows:

$$\forall x \text{ man}(x) \rightarrow \text{respects}(x, \text{parent}).$$

3. Some boys play cricket.

In this question, the predicate is "**play(x, y)**," where $x= \text{boys}$, and $y= \text{game}$. Since there are some boys so we will use \exists , and it will be represented as:

$$\exists x \text{ boys}(x) \rightarrow \text{play}(x, \text{cricket}).$$

4. Not all students like both Mathematics and Science.

In this question, the predicate is "**like(x, y)**," **where x= student, and y= subject.**

Since there are not all students, so we will use \forall with negation, so following representation for this:

$$\neg \forall (x) [\text{student}(x) \rightarrow \text{like}(x, \text{Mathematics}) \wedge \text{like}(x, \text{Science})].$$

5. Only one student failed in Mathematics.

In this question, the predicate is "**failed(x, y)**," **where x= student, and y= subject.**

Since there is only one student who failed in Mathematics, so we will use following representation for this:

$$\exists (x) [\text{student}(x) \rightarrow \text{failed}(x, \text{Mathematics}) \wedge \forall (y) [\neg(x==y) \wedge \text{student}(y) \rightarrow \neg \text{failed}(x, \text{Mathematics})]].$$

Free and Bound Variables:

The quantifiers interact with variables which appear in a suitable way. There are two types of variables in First-order logic which are given below:

Free Variable: A variable is said to be a free variable in a formula if it occurs outside the scope of the quantifier.

Example: $\forall x \exists (y) [P(x, y, z)]$, where z is a free variable.

Bound Variable: A variable is said to be a bound variable in a formula if it occurs within the scope of the quantifier.

Example: $\forall x [A(x) B(y)]$, here x and y are the bound variables.

Inference in First-Order Logic

Inference in First-Order Logic is used to deduce new facts or sentences from existing sentences. Before understanding the FOL inference rule, let's understand some basic terminologies used in FOL.

Substitution:

Substitution is a fundamental operation performed on terms and formulas. It occurs in all inference systems in first-order logic. The substitution is complex in the presence of quantifiers in FOL. If we write **F[a/x]**, so it refers to substitute a constant "a" in place of variable "x".

Note: First-order logic is capable of expressing facts about some or all objects in the universe.

Equality:

First-Order logic does not only use predicate and terms for making atomic sentences but also uses another way, which is equality in FOL. For this, we can use **equality symbols** which specify that the two terms refer to the same object.

Example: Brother (John) = Smith.

As in the above example, the object referred by the **Brother (John)** is similar to the object referred by **Smith**. The equality symbol can also be used with negation to represent that two terms are not the same objects.

Example: $\neg(x=y)$ which is equivalent to $x \neq y$.

FOL inference rules for quantifier:

As propositional logic we also have inference rules in first-order logic, so following are some basic inference rules in FOL:

- **Universal Generalization**
- **Universal Instantiation**
- **Existential Instantiation**
- **Existential introduction**

1. Universal Generalization:

- Universal generalization is a valid inference rule which states that if premise $P(c)$ is true for any arbitrary element c in the universe of discourse, then we can have a conclusion as $\forall x P(x)$.

$$\frac{P(c)}{\forall x P(x)}$$

- It can be represented as: $\frac{P(c)}{\forall x P(x)}$.
- This rule can be used if we want to show that every element has a similar property.
- In this rule, x must not appear as a free variable.

Example: Let's represent, $P(c)$: "**A byte contains 8 bits**", so for $\forall x P(x)$ "**All bytes contain 8 bits**.", it will also be true.

2. Universal Instantiation:

- Universal instantiation is also called as universal elimination or UI is a valid inference rule. It can be applied multiple times to add new sentences.
- The new KB is logically equivalent to the previous KB.
- As per UI, **we can infer any sentence obtained by substituting a ground term for the variable.**
- The UI rule state that we can infer any sentence $P(c)$ by substituting a ground term c (a constant within domain x) from $\forall x P(x)$ **for any object in the universe of discourse.**

$$\frac{\forall x P(x)}{P(c)}$$

- It can be represented as: $\frac{\forall x P(x)}{P(c)}$.

Example:1.

IF "Every person like ice-cream" $\Rightarrow \forall x P(x)$ so we can infer that "John likes ice-cream" $\Rightarrow P(c)$

Example: 2.

Let's take a famous example,

"All kings who are greedy are Evil." So let our knowledge base contains this detail as in the form of FOL:

$$\forall x \text{ king}(x) \wedge \text{greedy}(x) \rightarrow \text{Evil}(x),$$

So from this information, we can infer any of the following statements using Universal Instantiation:

- **King(John) \wedge Greedy (John) \rightarrow Evil (John),**
- **King(Richard) \wedge Greedy (Richard) \rightarrow Evil (Richard),**

- **King(Father(John)) \wedge Greedy (Father(John)) \rightarrow Evil (Father(John)),**

3. Existential Instantiation:

- Existential instantiation is also called as Existential Elimination, which is a valid inference rule in first-order logic.
- It can be applied only once to replace the existential sentence.
- The new KB is not logically equivalent to old KB, but it will be satisfiable if old KB was satisfiable.
- This rule states that one can infer $P(c)$ from the formula given in the form of $\exists x P(x)$ for a new constant symbol c .
- The restriction with this rule is that c used in the rule must be a new term for which $P(c)$ is true.

$$\frac{\exists x P(x)}{P(c)}$$

- It can be represented as: $\exists x P(x) \quad P(c)$

Example:

From the given sentence: **$\exists x \text{Crown}(x) \wedge \text{OnHead}(x, \text{John})$,**

So we can infer: **$\text{Crown}(K) \wedge \text{OnHead}(K, \text{John})$** , as long as K does not appear in the knowledge base.

- The above used K is a constant symbol, which is called **Skolem constant**.
- The Existential instantiation is a special case of **Skolemization process**.

4. Existential introduction

- An existential introduction is also known as an existential generalization, which is a valid inference rule in first-order logic.
- This rule states that if there is some element c in the universe of discourse which has a property P , then we can infer that there exists something in the universe which has the property P .

$$\frac{P(c)}{\exists x P(x)}$$

- It can be represented as: $\exists x P(x)$
- **Example: Let's say that,**
 "Priyanka got good marks in English."
 "Therefore, someone got good marks in English."

Generalized Modus Ponens Rule:

For the inference process in FOL, we have a single inference rule which is called Generalized Modus Ponens. It is lifted version of Modus ponens.

Generalized Modus Ponens can be summarized as, " P implies Q and P is asserted to be true, therefore Q must be True."

According to Modus Ponens, for atomic sentences p_1, p_1', q . Where there is a substitution θ such that $\text{SUBST}(\theta, p_1') = \text{SUBST}(\theta, p_1)$, it can be represented as:

$$\frac{p_1', p_2', \dots, p_n', (p_1 \wedge p_2 \wedge \dots \wedge p_n \Rightarrow q)}{\text{SUBST}(\theta, q)}$$

Example:

We will use this rule for Kings are evil, so we will find some x such that x is king, and x is greedy so we can infer that x is evil.

1. Here let say, p_1' is king(John) p_1 is king(x)
2. p_2' is Greedy(y) p_2 is Greedy(x)
3. θ is $\{x/John, y/John\}$ q is evil(x)
4. $\text{SUBST}(\theta, q)$.

Forward Chaining and backward chaining in AI

In artificial intelligence, forward and backward chaining is one of the important topics, but before understanding forward and backward chaining lets first understand that from where these two terms came.

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 chaining**

Horn 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 starts 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 adds 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:

Example:

"As per the law, it is a crime for an American to sell weapons to hostile nations. Country A, an enemy of America, has some missiles, and all the missiles were sold to it by Robert, who is an American citizen."

Prove that **"Robert is criminal."**

To solve the above problem, first, we will convert all the above facts into first-order definite clauses, and then we will use a forward-chaining algorithm to reach the goal.

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) \wedge weapon(q) \wedge sells (p, q, r) \wedge hostile(r) \rightarrow Criminal(p)
...(1)

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

Owns(A, T1)(2)

Missile(T1)(3)

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

?p Missiles(p) \wedge Owns (A, p) \rightarrow Sells (Robert, p, A)(4)

- Missiles are weapons.

Missile(p) \rightarrow Weapons (p)(5)

- Enemy of America is known as hostile.

Enemy(p, America) \rightarrow 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.

American (Robert)

Missile (T1)

Owns (A,T1)

Enemy (A, America)

Step-2:

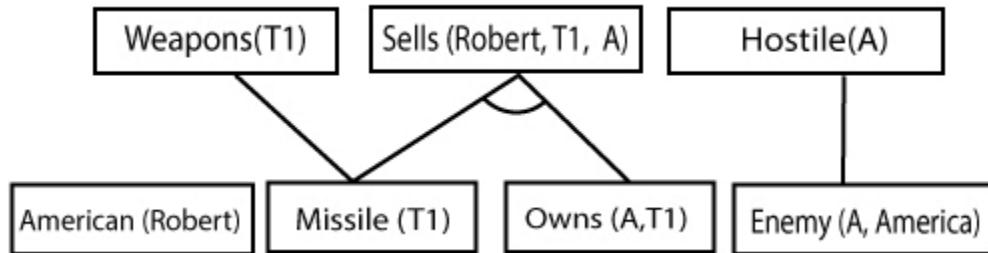
At the second step, we will see those facts which infer from available facts and with satisfied premises.

Rule-(1) does not satisfy premises, so it will not be added in the first iteration.

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

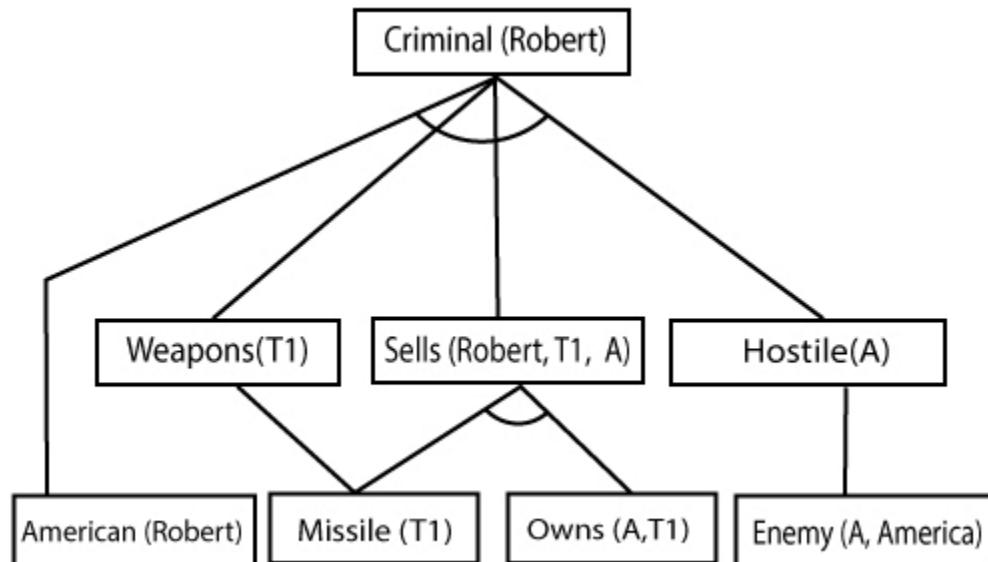
Rule-(4) satisfy with the substitution $\{p/T1\}$, **so Sells (Robert, T1, A)** is added, which infers from the conjunction of Rule (2) and (3).

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.

B. 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) \wedge weapon(q) \wedge sells (p, q, r) \wedge hostile(r) \rightarrow Criminal(p)**
...(1)
- **Owns(A, T1)**(2)
- **Missile(T1)**
- **?p Missiles(p) \wedge Owns (A, p) \rightarrow Sells (Robert, p, A)**(4)
- **Missile(p) \rightarrow Weapons (p)**(5)
- **Enemy(p, America) \rightarrow 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:

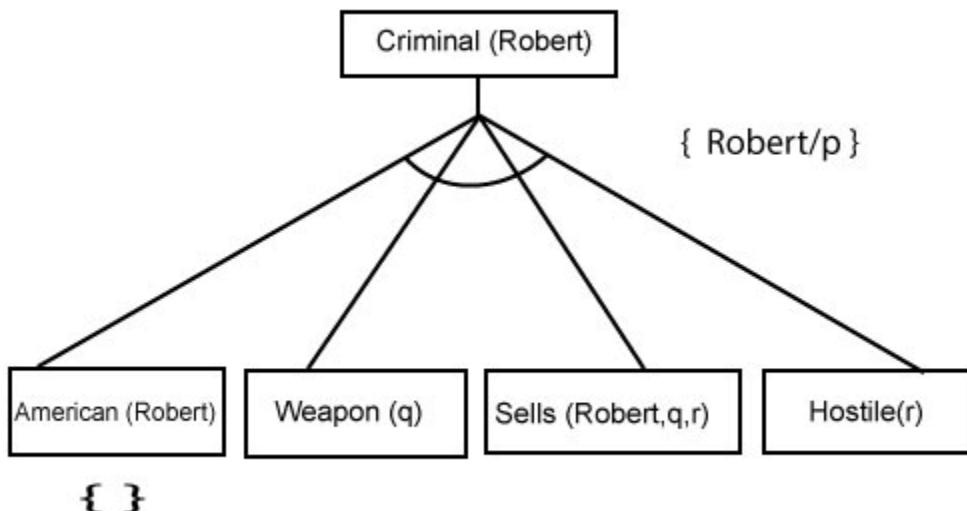
At the first step, we will take the goal fact. And from the goal fact, we will infer other facts, and at last, we will prove those facts true. So our goal fact is "Robert is Criminal," so following is the predicate of it.

Criminal (Robert)

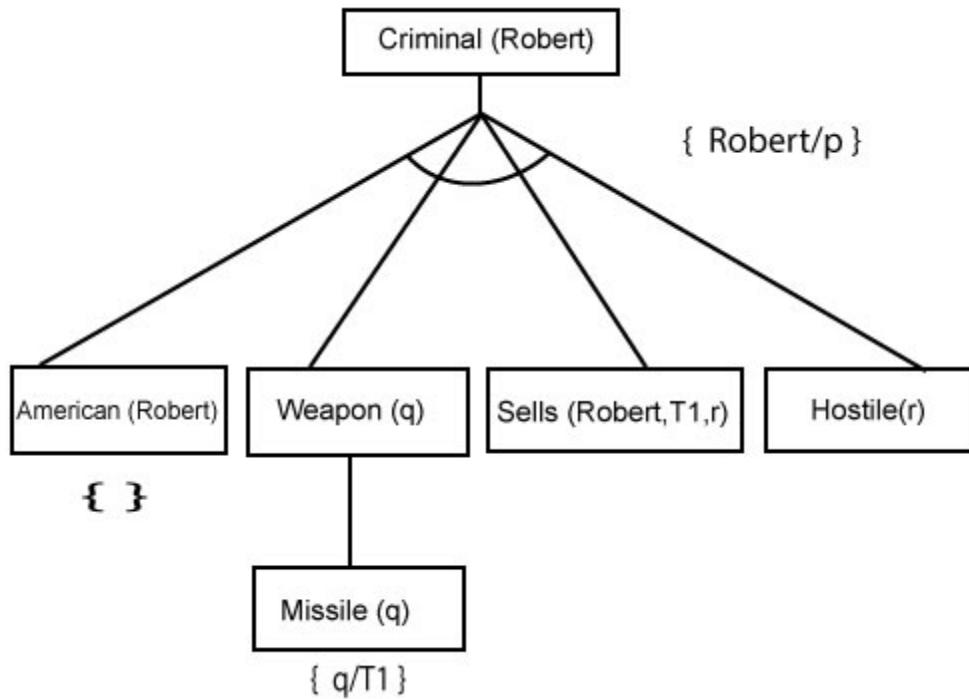
Step-2:

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.

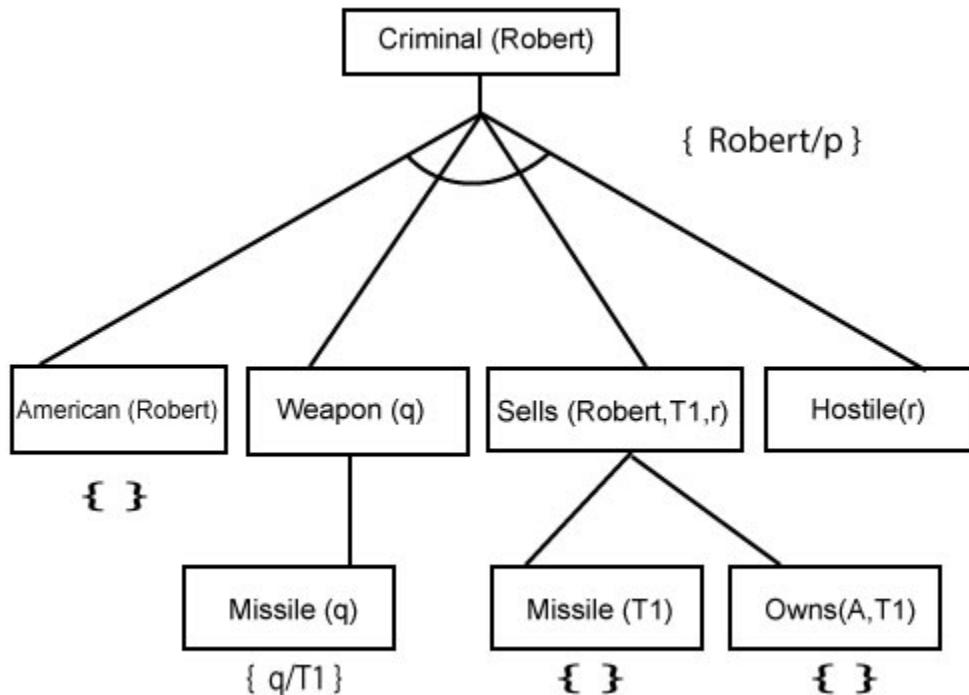


Step-3: At step-3, we will extract further fact Missile(q) which infer from Weapon(q), as it satisfies Rule-(5). Weapon (q) is also true with the substitution of a constant T1 at q.



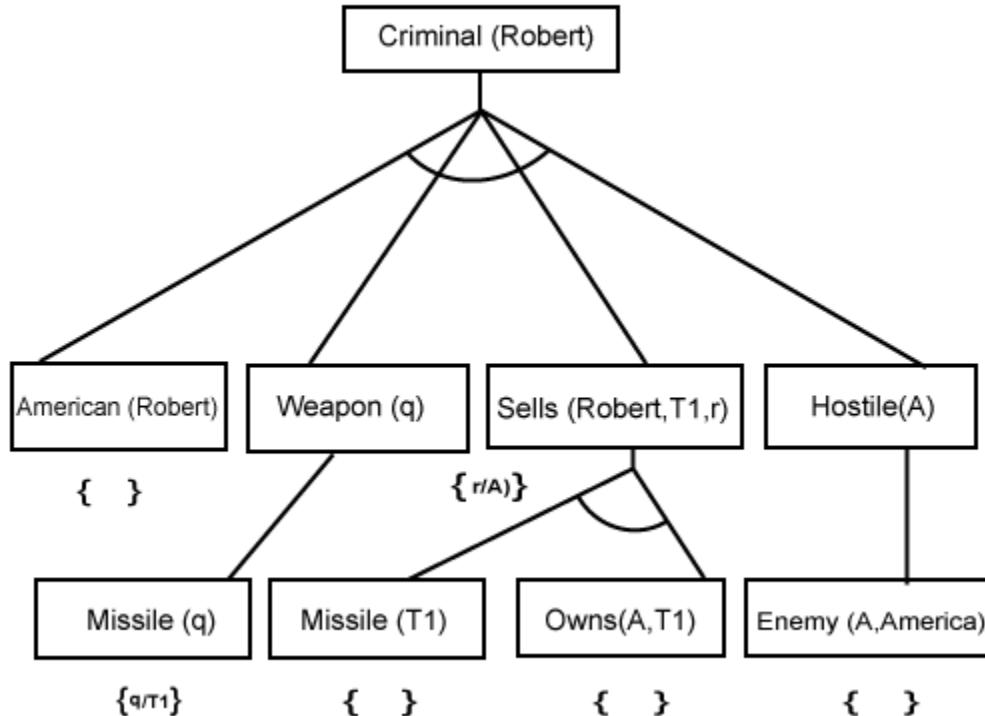
Step-4:

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. And hence all the statements are proved true using backward chaining.



Difference between backward chaining and forward chaining

Following is the difference between the forward chaining and backward chaining:

- Forward chaining as the name suggests, start from the known facts and move forward by applying inference rules to extract more data, and it continues until it reaches to the goal, whereas backward chaining starts from the goal, move backward by using inference rules to determine the facts that satisfy the goal.
- Forward chaining is called a **data-driven** inference technique, whereas backward chaining is called a **goal-driven** inference technique.
- Forward chaining is known as the **down-up** approach, whereas backward chaining is known as a **top-down** approach.

- Forward chaining uses **breadth-first search** strategy, whereas backward chaining uses **depth-first search** strategy.
- Forward and backward chaining both applies **Modus ponens** inference rule.
- Forward chaining can be used for tasks such as **planning, design process monitoring, diagnosis, and classification**, whereas backward chaining can be used for **classification and diagnosis tasks**.
- Forward chaining can be like an exhaustive search, whereas backward chaining tries to avoid the unnecessary path of reasoning.
- In forward-chaining there can be various ASK questions from the knowledge base, whereas in backward chaining there can be fewer ASK questions.
- Forward chaining is slow as it checks for all the rules, whereas backward chaining is fast as it checks few required rules only.

S. No.	Forward Chaining	Backward Chaining
1.	Forward chaining starts from known facts and applies inference rule to extract more data until it reaches to the goal.	Backward chaining starts from the goal and works backward using 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 it starts 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	Backward chaining is suitable for diagnostic, prescription

	planning, monitoring, control, and interpretation application.	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.

Probabilistic reasoning in Artificial intelligence

Uncertainty:

Till now, we have learned knowledge representation using first-order logic and propositional logic with certainty, which means we were sure about the predicates. With this knowledge representation, we might write $A \rightarrow B$, which means if A is true then B is true, but consider a situation where we are not sure about whether A is true or not then we cannot express this statement, this situation is called uncertainty.

So to represent uncertain knowledge, where we are not sure about the predicates, we need uncertain reasoning or probabilistic reasoning.

Causes of uncertainty:

Following are some leading causes of uncertainty to occur in the real world.

1. Information occurred from unreliable sources.
2. Experimental Errors
3. Equipment fault
4. Temperature variation
5. Climate change.

Probabilistic reasoning:

Probabilistic reasoning is a way of knowledge representation where we apply the concept of probability to indicate the uncertainty in knowledge. In probabilistic reasoning, we combine probability theory with logic to handle the uncertainty.

We use probability in probabilistic reasoning because it provides a way to handle the uncertainty that is the result of someone's laziness and ignorance.

In the real world, there are lots of scenarios, where the certainty of something is not confirmed, such as "It will rain today," "behavior of someone for some situations," "A match between two teams or two players." These are probable sentences for which we can assume that it will happen but not sure about it, so here we use probabilistic reasoning.

Need of probabilistic reasoning in AI:

- When there are unpredictable outcomes.
- When specifications or possibilities of predicates becomes too large to handle.
- When an unknown error occurs during an experiment.

In probabilistic reasoning, there are two ways to solve problems with uncertain knowledge:

- **Bayes' rule**
- **Bayesian Statistics**

Note: We will learn the above two rules in later chapters.

As probabilistic reasoning uses probability and related terms, so before understanding probabilistic reasoning, let's understand some common terms:

Probability: Probability can be defined as a chance that an uncertain event will occur. It is the numerical measure of the likelihood that an event will occur. The value of probability always remains between 0 and 1 that represent ideal uncertainties.

1. $0 \leq P(A) \leq 1$, where $P(A)$ is the probability of an event A.
1. $P(A) = 0$, indicates total uncertainty in an event A.
1. $P(A) = 1$, indicates total certainty in an event A.

We can find the probability of an uncertain event by using the below formula.

$$\text{Probability of occurrence} = \frac{\text{Number of desired outcomes}}{\text{Total number of outcomes}}$$

- $P(\neg A)$ = probability of a not happening event.

- $P(\neg A) + P(A) = 1$.

Event: Each possible outcome of a variable is called an event.

Sample space: The collection of all possible events is called sample space.

Random variables: Random variables are used to represent the events and objects in the real world.

Prior probability: The prior probability of an event is probability computed before observing new information.

Posterior Probability: The probability that is calculated after all evidence or information has taken into account. It is a combination of prior probability and new information.

Conditional probability:

Conditional probability is a probability of occurring an event when another event has already happened.

Let's suppose, we want to calculate the event A when event B has already occurred, "the probability of A under the conditions of B", it can be written as:

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

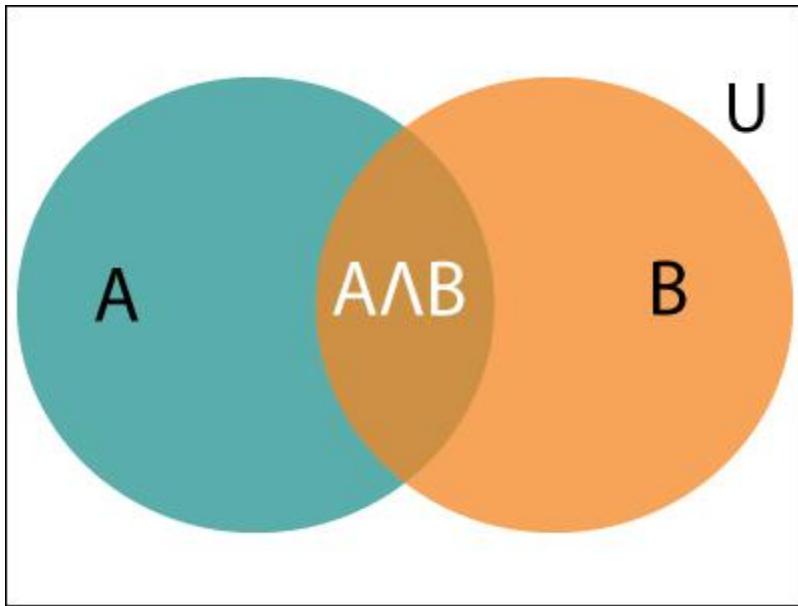
Where $P(A \wedge B)$ = Joint probability of a and B

$P(B)$ = Marginal probability of B.

If the probability of A is given and we need to find the probability of B, then it will be given as:

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

It can be explained by using the below Venn diagram, where B is occurred event, so sample space will be reduced to set B, and now we can only calculate event A when event B is already occurred by dividing the probability of **$P(A \wedge B)$ by $P(B)$** .



Example:

In a class, there are 70% of the students who like English and 40% of the students who like English and mathematics, and then what is the percent of students those who like English also like mathematics?

Solution:

Let, A is an event that a student likes Mathematics

B is an event that a student likes English.

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{0.4}{0.7} = 57\%$$

Hence, 57% are the students who like English also like Mathematics.

Bayes' theorem in Artificial intelligence

Bayes' theorem:

Bayes' theorem is also known as **Bayes' rule**, **Bayes' law**, or **Bayesian reasoning**, which determines the probability of an event with uncertain knowledge.

In probability theory, it relates the conditional probability and marginal probabilities of two random events.

Bayes' theorem was named after the British mathematician **Thomas Bayes**. The **Bayesian inference** is an application of Bayes' theorem, which is fundamental to Bayesian statistics.

It is a way to calculate the value of $P(B|A)$ with the knowledge of $P(A|B)$.

Bayes' theorem allows updating the probability prediction of an event by observing new information of the real world.

Example: If cancer corresponds to one's age then by using Bayes' theorem, we can determine the probability of cancer more accurately with the help of age.

Bayes' theorem can be derived using product rule and conditional probability of event A with known event B:

As from product rule we can write:

$$1. \quad P(A \wedge B) = P(A|B) P(B) \text{ or}$$

Similarly, the probability of event B with known event A:

$$1. \quad P(A \wedge B) = P(B|A) P(A)$$

Equating right hand side of both the equations, we will get:

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

The above equation (a) is called as **Bayes' rule** or **Bayes' theorem**. This equation is basic of most modern AI systems for **probabilistic inference**.

It shows the simple relationship between joint and conditional probabilities. Here,

$P(A|B)$ is known as **posterior**, which we need to calculate, and it will be read as Probability of hypothesis A when we have occurred an evidence B.

$P(B|A)$ is called the likelihood, in which we consider that hypothesis is true, then we calculate the probability of evidence.

$P(A)$ is called the **prior probability**, probability of hypothesis before considering the evidence

$P(B)$ is called **marginal probability**, pure probability of an evidence.

In the equation (a), in general, we can write $P(B) = P(A) * P(B|A)$, hence the Bayes' rule can be written as:

$$P(A_i|B) = \frac{P(A_i) \cdot P(B|A_i)}{\sum_{i=1}^k P(A_i) \cdot P(B|A_i)}$$

Where $A_1, A_2, A_3, \dots, A_n$ is a set of mutually exclusive and exhaustive events.

Applying Bayes' rule:

Bayes' rule allows us to compute the single term $P(B|A)$ in terms of $P(A|B)$, $P(B)$, and $P(A)$. This is very useful in cases where we have a good probability of these three terms and want to determine the fourth one. Suppose we want to perceive the effect of some unknown cause, and want to compute that cause, then the Bayes' rule becomes:

$$P(\text{cause}|\text{effect}) = \frac{P(\text{effect}|\text{cause}) P(\text{cause})}{P(\text{effect})}$$

Example-1:

Question: what is the probability that a patient has diseases meningitis with a stiff neck?

Given Data:

A doctor is aware that disease meningitis causes a patient to have a stiff neck, and it occurs 80% of the time. He is also aware of some more facts, which are given as follows:

- o The Known probability that a patient has meningitis disease is 1/30,000.
- o The Known probability that a patient has a stiff neck is 2%.

Let a be the proposition that patient has stiff neck and b be the proposition that patient has meningitis. , so we can calculate the following as:

$$P(a|b) = 0.8$$

$$P(b) = 1/30000$$

$$P(a) = .02$$

$$P(b|a) = \frac{P(a|b)P(b)}{P(a)} = \frac{0.8 \cdot (\frac{1}{30000})}{0.02} = 0.001333333.$$

Hence, we can assume that 1 patient out of 750 patients has meningitis disease with a stiff neck.

Example-2:

Question: From a standard deck of playing cards, a single card is drawn. The probability that the card is king is $4/52$, then calculate posterior probability $P(\text{King}|\text{Face})$, which means the drawn face card is a king card.

Solution:

$$P(\text{king}|\text{face}) = \frac{P(\text{Face}|\text{king}) \cdot P(\text{King})}{P(\text{Face})} \quad \dots \dots \text{(i)}$$

P(king): probability that the card is King = $4/52 = 1/13$

P(face): probability that a card is a face card= 3/13

$P(\text{Face}|\text{King})$: probability of face card when we assume it is a king = 1

Putting all values in equation (i) we will get:

$$P(\text{king} \mid \text{face}) = \frac{1 * \left(\frac{1}{13}\right)}{\left(\frac{3}{13}\right)} = 1/3, \text{ it is a probability that a face card is a king card.}$$

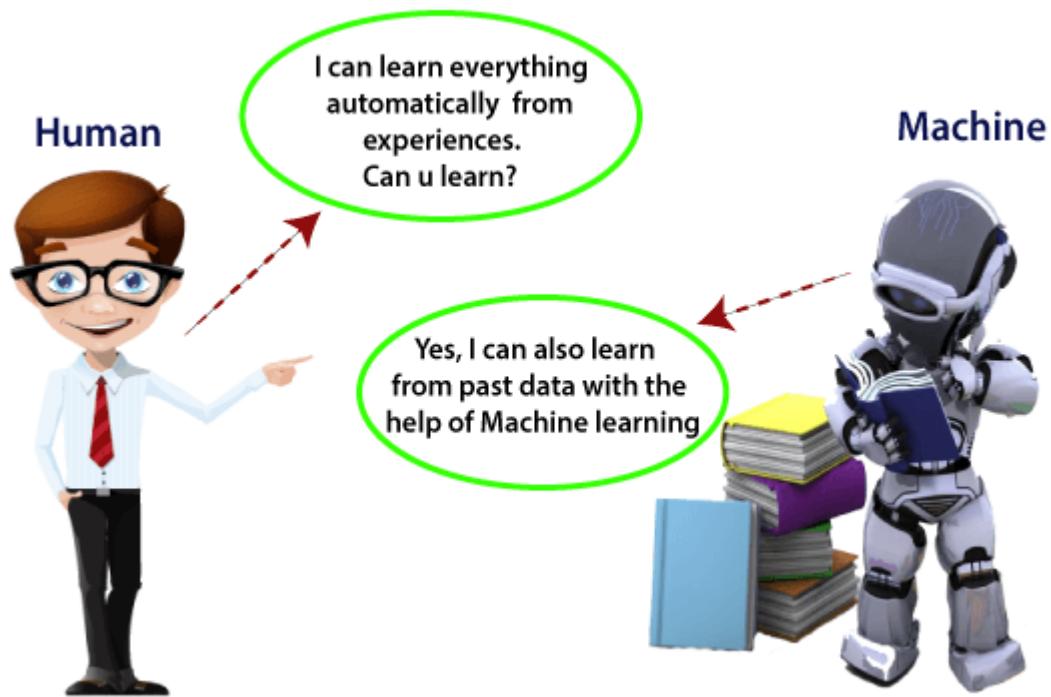
Application of Bayes' theorem in Artificial intelligence:

Following are some applications of Bayes' theorem:

- It is used to calculate the next step of the robot when the already executed step is given.
 - Bayes' theorem is helpful in weather forecasting.
 - It can solve the Monty Hall problem.

What is Machine Learning

In the real world, we are surrounded by humans who can learn everything from their experiences with their learning capability, and we have computers or machines which work on our instructions. But can a machine also learn from experiences or past data like a human does? So here comes the role of **Machine Learning**.



Machine Learning is said as a subset of **artificial intelligence** that is mainly concerned with the development of algorithms which allow a computer to learn from the data and past experiences on their own. The term machine learning was first introduced by **Arthur Samuel** in **1959**. We can define it in a summarized way as:

Machine learning enables a machine to automatically learn from data, improve performance from experiences, and predict things without being explicitly programmed.

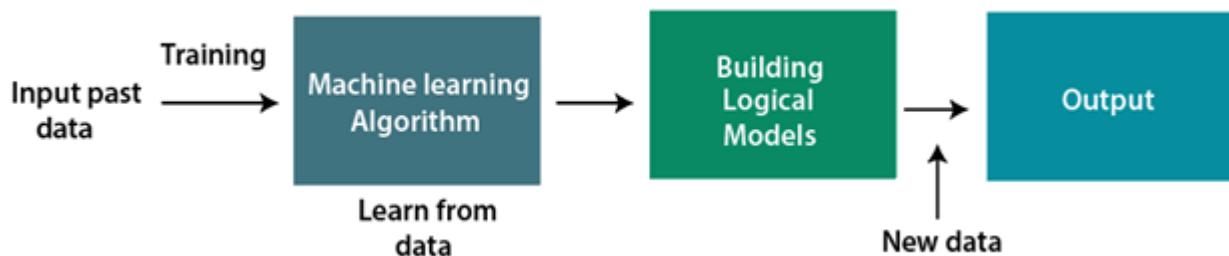
With the help of sample historical data, which is known as **training data**, machine learning algorithms build a **mathematical model** that helps in making predictions or decisions without being explicitly programmed. Machine learning brings computer science and statistics together for creating predictive models. Machine learning constructs or uses the algorithms that learn from historical data. The more we will provide the information, the higher will be the performance.

A machine has the ability to learn if it can improve its performance by gaining more data.

How does Machine Learning work

A Machine Learning system **learns from historical data, builds the prediction models, and whenever it receives new data, predicts the output for it**. The accuracy of predicted output depends upon the amount of data, as the huge amount of data helps to build a better model which predicts the output more accurately.

Suppose we have a complex problem, where we need to perform some predictions, so instead of writing a code for it, we just need to feed the data to generic algorithms, and with the help of these algorithms, machine builds the logic as per the data and predict the output. Machine learning has changed our way of thinking about the problem. The below block diagram explains the working of Machine Learning algorithm:



Features of Machine Learning:

- Machine learning uses data to detect various patterns in a given dataset.
- It can learn from past data and improve automatically.
- It is a data-driven technology.
- Machine learning is much similar to data mining as it also deals with the huge amount of the data.

Need for Machine Learning

The need for machine learning is increasing day by day. The reason behind the need for machine learning is that it is capable of doing tasks that are too complex for a person to implement directly. As a human, we have some limitations as we cannot access the huge amount of data manually, so for this, we need some computer systems and here comes the machine learning to make things easy for us.

We can train machine learning algorithms by providing them the huge amount of data and let them explore the data, construct the models, and predict the required output automatically. The performance of the machine learning algorithm depends on the amount of data, and it can be determined by the cost function. With the help of machine learning, we can save both time and money.

The importance of machine learning can be easily understood by its uses cases, Currently, machine learning is used in **self-driving cars**, **cyber fraud detection**, **face recognition**, and **friend suggestion by Facebook**, etc. Various top companies such as Netflix and Amazon have build machine learning models that are using a vast amount of data to analyze the user interest and recommend product accordingly.

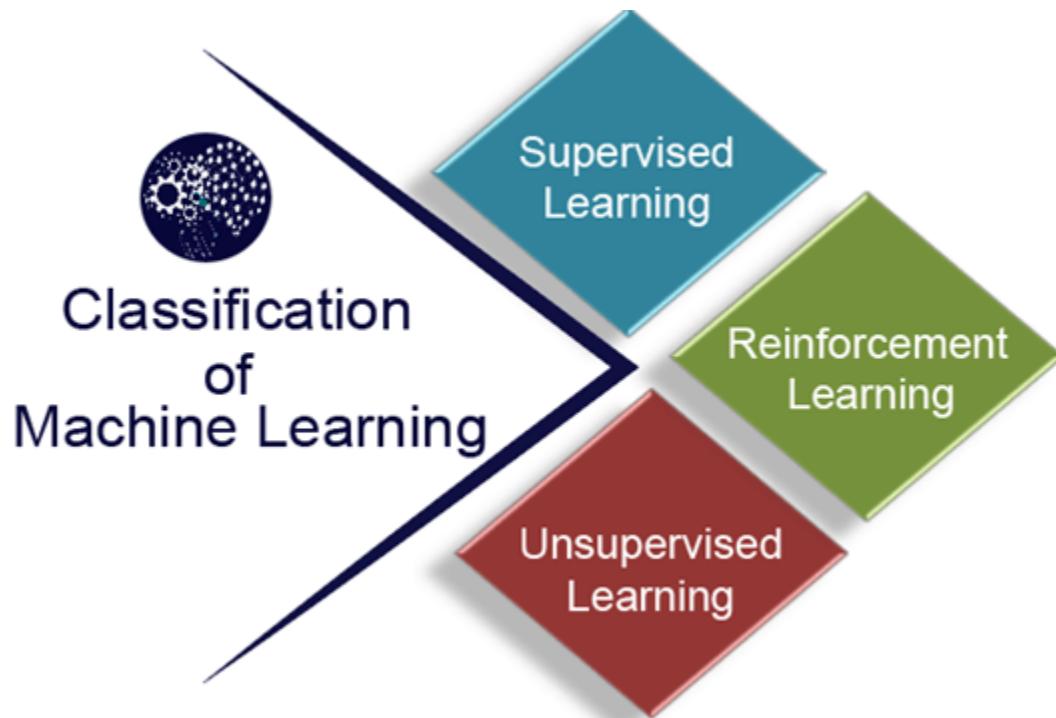
Following are some key points which show the importance of Machine Learning:

- Rapid increment in the production of data
- Solving complex problems, which are difficult for a human
- Decision making in various sector including finance
- Finding hidden patterns and extracting useful information from data.

Classification of Machine Learning

At a broad level, machine learning can be classified into three types:

1. **Supervised learning**
2. **Unsupervised learning**
3. **Reinforcement learning**



1) Supervised Learning

Supervised learning is a type of machine learning method in which we provide sample labeled data to the machine learning system in order to train it, and on that basis, it predicts the output.

The system creates a model using labeled data to understand the datasets and learn about each data, once the training and processing are done then we test the model by providing a sample data to check whether it is predicting the exact output or not.

The goal of supervised learning is to map input data with the output data. The supervised learning is based on supervision, and it is the same as when a student learns things in the supervision of the teacher. The example of supervised learning is **spam filtering**.

Supervised learning can be grouped further in two categories of algorithms:

- **Classification**
- **Regression**

2) Unsupervised Learning

Unsupervised learning is a learning method in which a machine learns without any supervision.

The training is provided to the machine with the set of data that has not been labeled, classified, or categorized, and the algorithm needs to act on that data without any supervision. The goal of unsupervised learning is to restructure the input data into new features or a group of objects with similar patterns.

In unsupervised learning, we don't have a predetermined result. The machine tries to find useful insights from the huge amount of data. It can be further classified into two categories of algorithms:

- **Clustering**
- **Association**

3) Reinforcement Learning

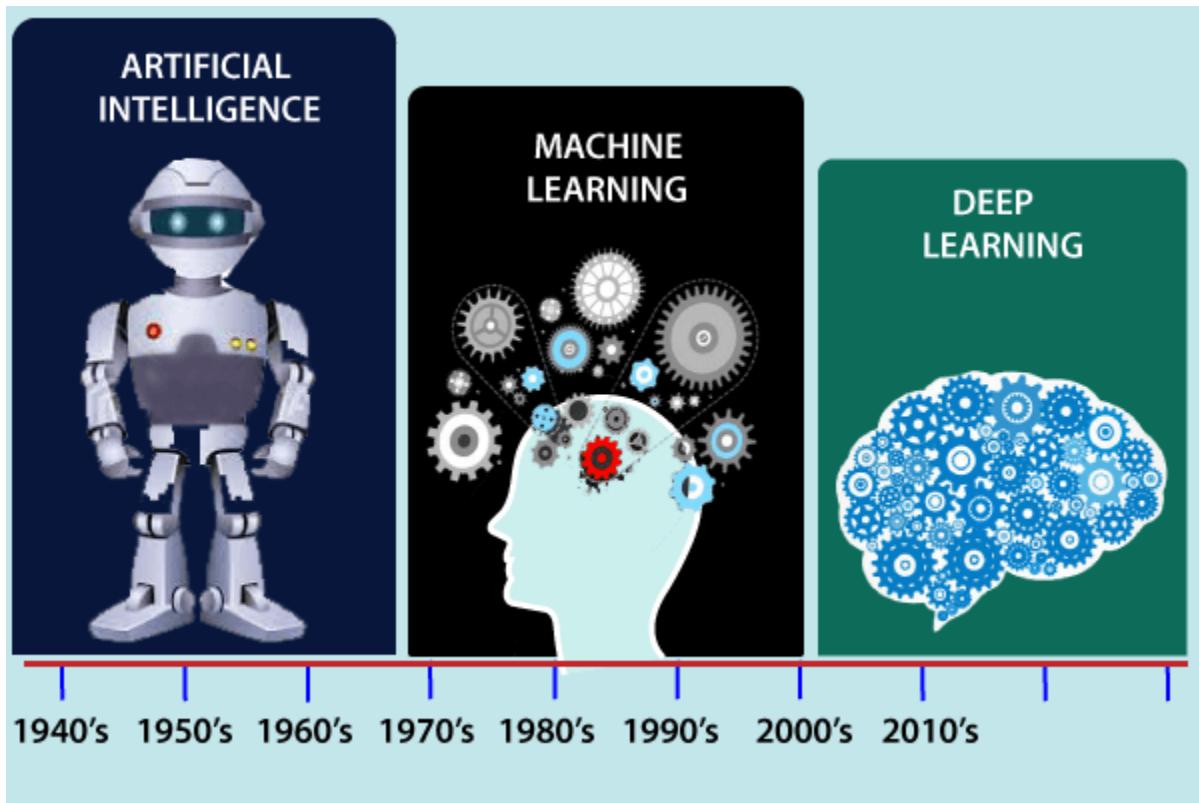
Reinforcement learning is a feedback-based learning method, in which a learning agent gets a reward for each right action and gets a penalty for each wrong action. The agent learns automatically with these feedbacks and improves its performance. In reinforcement learning, the agent interacts with the environment and explores it. The goal of an agent is to get the most reward points, and hence, it improves its performance.

The robotic dog, which automatically learns the movement of his arms, is an example of Reinforcement learning.

Note: We will learn about the above types of machine learning in detail in later chapters.

History of Machine Learning

Before some years (about 40-50 years), machine learning was science fiction, but today it is the part of our daily life. Machine learning is making our day to day life easy from **self-driving cars** to **Amazon virtual assistant "Alexa"**. However, the idea behind machine learning is so old and has a long history. Below some milestones are given which have occurred in the history of machine learning:



The early history of Machine Learning (Pre-1940):

- **1834:** In 1834, Charles Babbage, the father of the computer, conceived a device that could be programmed with punch cards. However, the machine was never built, but all modern computers rely on its logical structure.
- **1936:** In 1936, Alan Turing gave a theory that how a machine can determine and execute a set of instructions.

The era of stored program computers:

- **1940:** In 1940, the first manually operated computer, "ENIAC" was invented, which was the first electronic general-purpose computer. After that stored program computer such as EDSAC in 1949 and EDVAC in 1951 were invented.
- **1943:** In 1943, a human neural network was modeled with an electrical circuit. In 1950, the scientists started applying their idea to work and analyzed how human neurons might work.

Computer machinery and intelligence:

- **1950:** In 1950, Alan Turing published a seminal paper, "**Computer Machinery and Intelligence**," on the topic of artificial intelligence. **In his paper, he asked, "Can machines think?"**

Machine intelligence in Games:

- **1952:** Arthur Samuel, who was the pioneer of machine learning, created a program that helped an IBM computer to play a checkers game. It performed better more it played.
- **1959:** In 1959, the term "Machine Learning" was first coined by **Arthur Samuel.**

The first "AI" winter:

- The duration of 1974 to 1980 was the tough time for AI and ML researchers, and this duration was called as **AI winter.**
- In this duration, failure of machine translation occurred, and people had reduced their interest from AI, which led to reduced funding by the government to the researches.

Machine Learning from theory to reality

- **1959:** In 1959, the first neural network was applied to a real-world problem to remove echoes over phone lines using an adaptive filter.
- **1985:** In 1985, Terry Sejnowski and Charles Rosenberg invented a neural network **NETtalk**, which was able to teach itself how to correctly pronounce 20,000 words in one week.

- **1997:** The IBM's **Deep blue** intelligent computer won the chess game against the chess expert Garry Kasparov, and it became the first computer which had beaten a human chess expert.

Machine Learning at 21st century

- **2006:** In the year 2006, computer scientist Geoffrey Hinton has given a new name to neural net research as "**deep learning**," and nowadays, it has become one of the most trending technologies.
- **2012:** In 2012, Google created a deep neural network which learned to recognize the image of humans and cats in YouTube videos.
- **2014:** In 2014, the Chabot "**Eugen Goostman**" cleared the Turing Test. It was the first Chabot who convinced the 33% of human judges that it was not a machine.
- **2014: DeepFace** was a deep neural network created by Facebook, and they claimed that it could recognize a person with the same precision as a human can do.
- **2016: AlphaGo** beat the world's number second player **Lee sedol** at **Go game**. In 2017 it beat the number one player of this game **Ke Jie**.
- **2017:** In 2017, the Alphabet's Jigsaw team built an intelligent system that was able to learn the **online trolling**. It used to read millions of comments of different websites to learn to stop online trolling.

Machine Learning at present:

Now machine learning has got a great advancement in its research, and it is present everywhere around us, such as **self-driving cars**, **Amazon Alexa**, **Catboats**, **recommender system**, and many more. It includes **Supervised**, **unsupervised**, and **reinforcement learning with clustering**, **classification**, **decision tree**, **SVM algorithms**, etc.

Modern machine learning models can be used for making various predictions, including **weather prediction**, **disease prediction**, **stock market analysis**, etc.

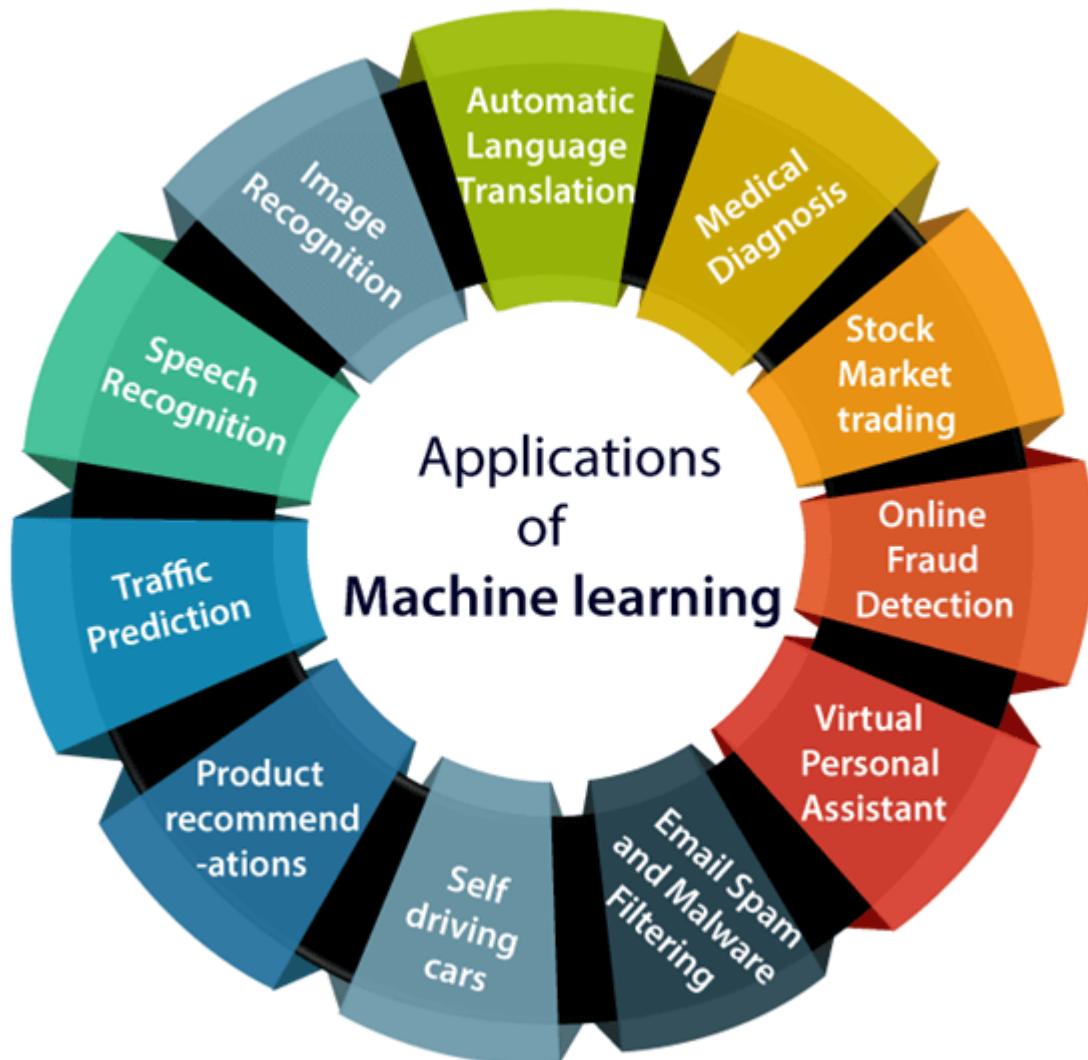
Prerequisites

Before learning machine learning, you must have the basic knowledge of followings so that you can easily understand the concepts of machine learning:

- Fundamental knowledge of probability and linear algebra.
- The ability to code in any computer language, especially in Python language.
- Knowledge of Calculus, especially derivatives of single variable and multivariate functions.

Applications of Machine learning

Machine learning is a buzzword for today's technology, and it is growing very rapidly day by day. We are using machine learning in our daily life even without knowing it such as Google Maps, Google assistant, Alexa, etc. Below are some most trending real-world applications of Machine Learning:



1. Image Recognition:

Image recognition is one of the most common applications of machine learning. It is used to identify objects, persons, places, digital images, etc. The popular use case of image recognition and face detection is, **Automatic friend tagging suggestion**:

Facebook provides us a feature of auto friend tagging suggestion. Whenever we upload a photo with our Facebook friends, then we automatically get a tagging suggestion with name, and the technology behind this is machine learning's **face detection** and **recognition algorithm**.

It is based on the Facebook project named "**Deep Face**," which is responsible for face recognition and person identification in the picture.

2. Speech Recognition

While using Google, we get an option of "**Search by voice**," it comes under speech recognition, and it's a popular application of machine learning.

Speech recognition is a process of converting voice instructions into text, and it is also known as "**Speech to text**", or "**Computer speech recognition**." At present, machine learning algorithms are widely used by various applications of speech recognition. **Google assistant**, **Siri**, **Cortana**, and **Alexa** are using speech recognition technology to follow the voice instructions.

3. Traffic prediction:

If we want to visit a new place, we take help of Google Maps, which shows us the correct path with the shortest route and predicts the traffic conditions.

It predicts the traffic conditions such as whether traffic is cleared, slow-moving, or heavily congested with the help of two ways:

- **Real Time location** of the vehicle from Google Map app and sensors
- **Average time has taken** on past days at the same time.

Everyone who is using Google Map is helping this app to make it better. It takes information from the user and sends back to its database to improve the performance.

4. Product recommendations:

Machine learning is widely used by various e-commerce and entertainment companies such as **Amazon**, **Netflix**, etc., for product recommendation to the user. Whenever we search for some product on Amazon, then we started getting an advertisement for the same product while internet surfing on the same browser and this is because of machine learning.

Google understands the user interest using various machine learning algorithms and suggests the product as per customer interest.

As similar, when we use Netflix, we find some recommendations for entertainment series, movies, etc., and this is also done with the help of machine learning.

5. Self-driving cars:

One of the most exciting applications of machine learning is self-driving cars. Machine learning plays a significant role in self-driving cars. Tesla, the most popular car manufacturing company is working on self-driving car. It is using unsupervised learning method to train the car models to detect people and objects while driving.

6. Email Spam and Malware Filtering:

Whenever we receive a new email, it is filtered automatically as important, normal, and spam. We always receive an important mail in our inbox with the important symbol and spam emails in our spam box, and the technology behind this is Machine learning. Below are some spam filters used by Gmail:

- Content Filter
- Header filter
- General blacklists filter
- Rules-based filters
- Permission filters

Some machine learning algorithms such as **Multi-Layer Perceptron**, **Decision tree**, and **Naïve Bayes classifier** are used for email spam filtering and malware detection.

7. Virtual Personal Assistant:

We have various virtual personal assistants such as **Google assistant**, **Alexa**, **Cortana**, **Siri**. As the name suggests, they help us in finding the information using our voice instruction. These assistants can help us in various ways just by our voice instructions such as Play music, call someone, Open an email, Scheduling an appointment, etc.

These virtual assistants use machine learning algorithms as an important part.

These assistant record our voice instructions, send it over the server on a cloud, and decode it using ML algorithms and act accordingly.

8. Online Fraud Detection:

Machine learning is making our online transaction safe and secure by detecting fraud transaction. Whenever we perform some online transaction, there may be various ways

that a fraudulent transaction can take place such as **fake accounts, fake ids, and steal money** in the middle of a transaction. So to detect this, **Feed Forward Neural network** helps us by checking whether it is a genuine transaction or a fraud transaction.

For each genuine transaction, the output is converted into some hash values, and these values become the input for the next round. For each genuine transaction, there is a specific pattern which gets change for the fraud transaction hence, it detects it and makes our online transactions more secure.

9. Stock Market trading:

Machine learning is widely used in stock market trading. In the stock market, there is always a risk of up and downs in shares, so for this machine learning's **long short term memory neural network** is used for the prediction of stock market trends.

10. Medical Diagnosis:

In medical science, machine learning is used for diseases diagnoses. With this, medical technology is growing very fast and able to build 3D models that can predict the exact position of lesions in the brain.

It helps in finding brain tumors and other brain-related diseases easily.

11. Automatic Language Translation:

Nowadays, if we visit a new place and we are not aware of the language then it is not a problem at all, as for this also machine learning helps us by converting the text into our known languages. Google's GNMT (Google Neural Machine Translation) provide this feature, which is a Neural Machine Learning that translates the text into our familiar language, and it called as automatic translation.

The technology behind the automatic translation is a sequence to sequence learning algorithm, which is used with image recognition and translates the text from one language to another language.

Unsupervised Machine Learning

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.



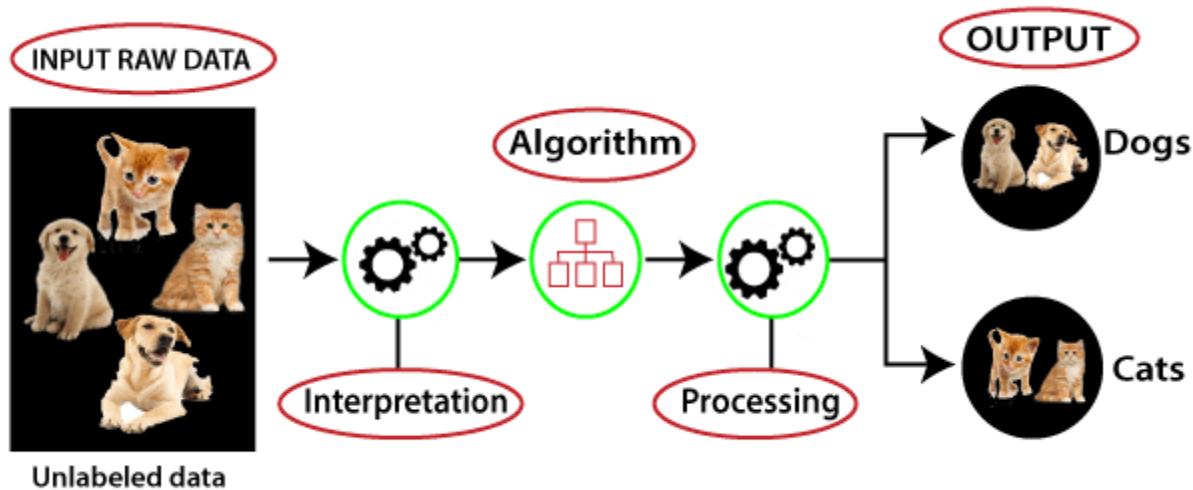
Why use Unsupervised Learning?

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:

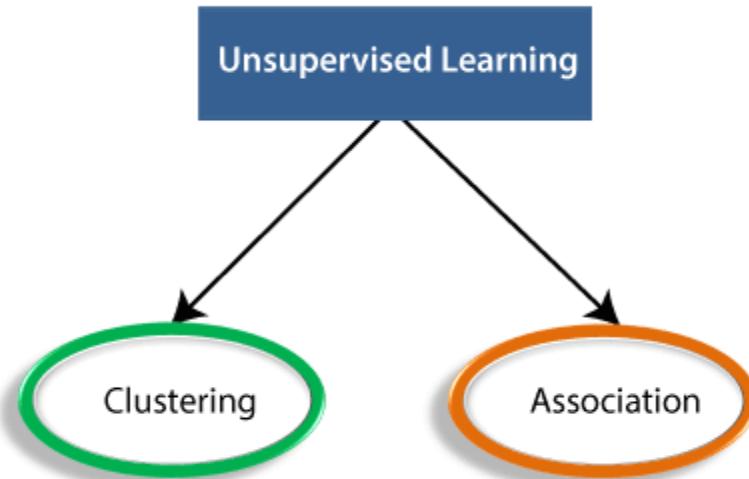


Here, we have taken an unlabeled input data, which means it is not categorized and corresponding outputs are also not given. Now, this unlabeled input data is fed to the machine learning model in order to train it. Firstly, it will interpret the raw data to find the hidden patterns from the data and 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.

Note: We will learn these algorithms in later chapters.

Unsupervised Learning algorithms:

Below is the list of some popular unsupervised learning algorithms:

- **K-means clustering**
- **KNN (k-nearest neighbors)**
- **Hierachal 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.

Clustering in Machine Learning

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.

After applying this clustering technique, each cluster or group is provided with a cluster-ID. ML system can use this id to simplify the processing of large and complex datasets.

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.

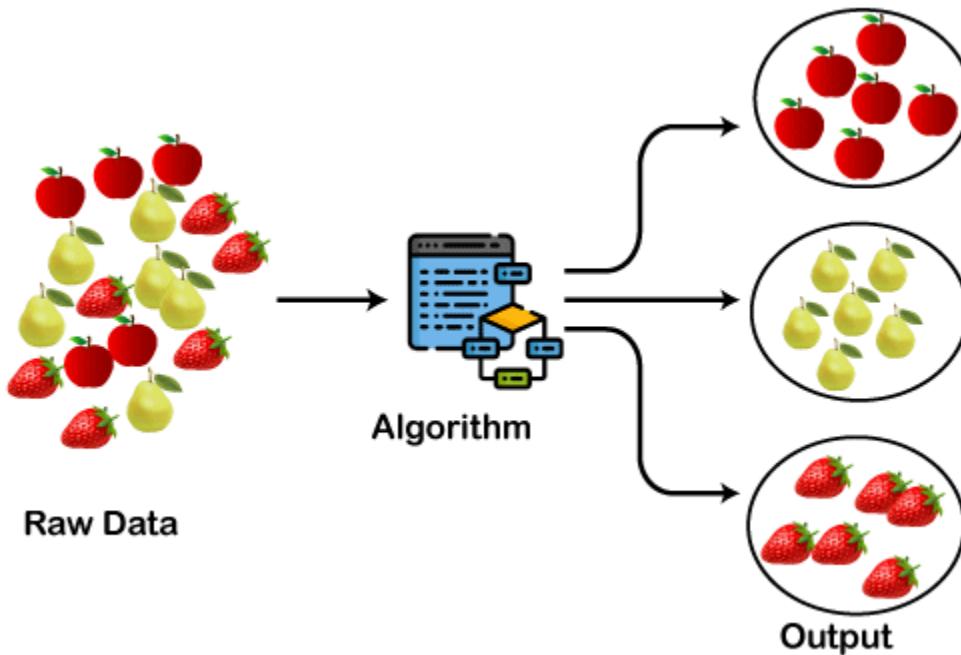
Example: Let's understand the clustering technique with the real-world example of Mall: When we visit any shopping mall, we can observe that the things with similar usage are grouped together. Such as the t-shirts are grouped in one section, and trousers are at other sections, similarly, at vegetable 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

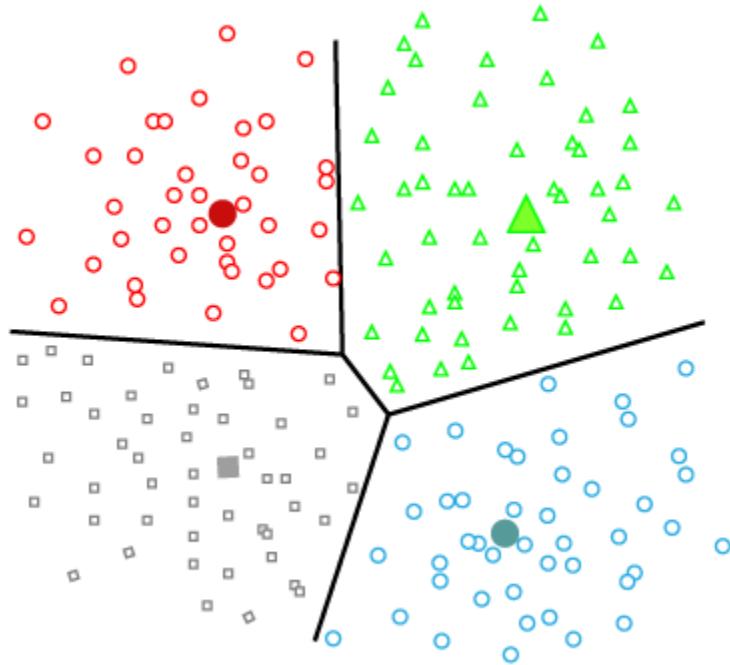
The clustering methods are broadly divided into **Hard clustering** (datapoint belongs to only one group) and **Soft Clustering** (data points can belong to another group also). But there are also other various approaches of Clustering exist. Below are the main clustering methods used in Machine learning:

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**](#).

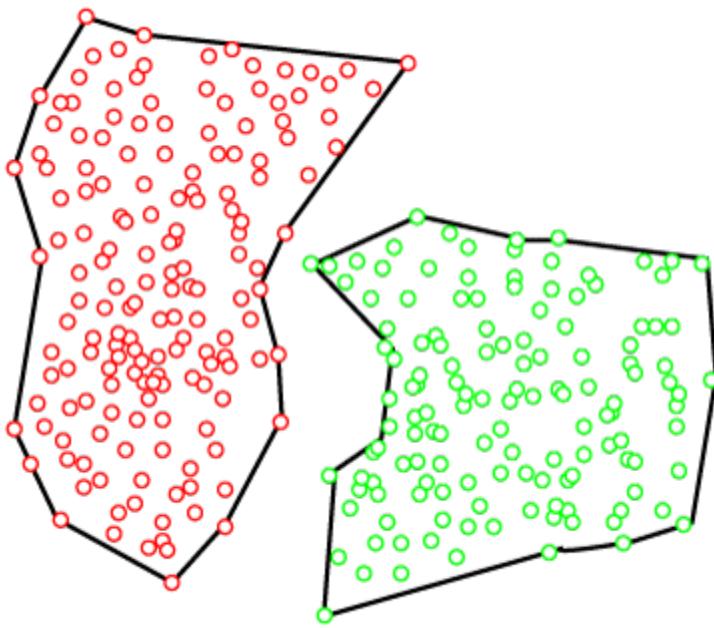
In this type, the dataset is divided into a set of k groups, where K is used to define the number of pre-defined groups. The cluster center is created in such a way that the distance between the data points of one cluster is minimum as compared to another cluster centroid.



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.

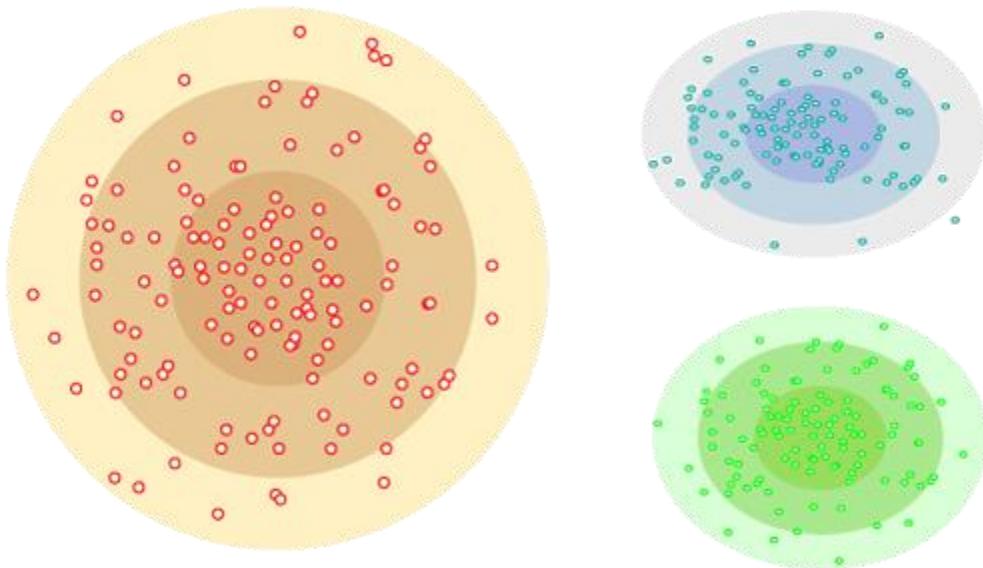
These algorithms can face difficulty in clustering the data points if the dataset has varying densities and high dimensions.



Distribution Model-Based Clustering

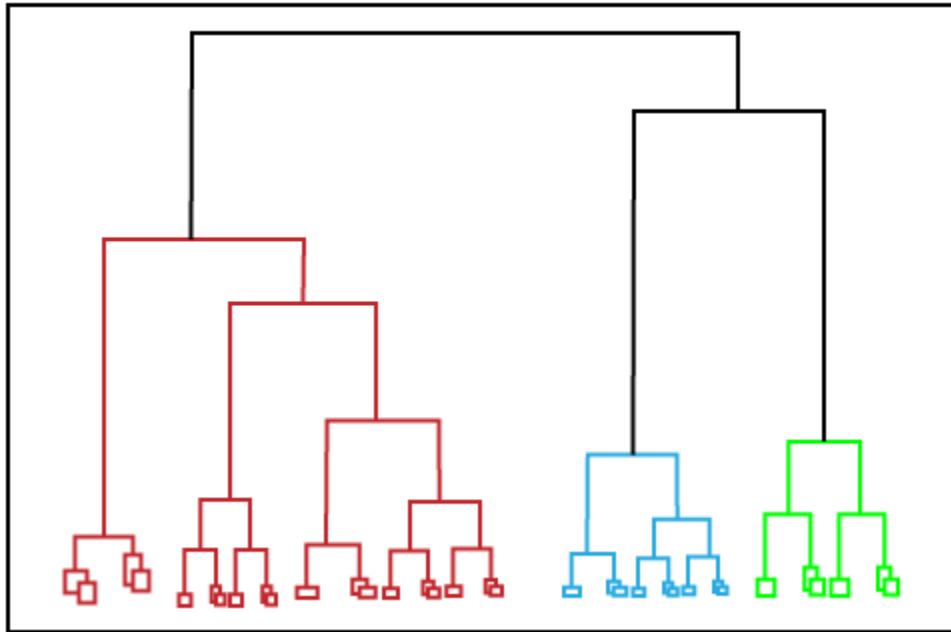
In the distribution model-based clustering method, the data is divided based on the probability of how a dataset belongs to a particular distribution. The grouping is done by assuming some distributions commonly **Gaussian Distribution**.

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.

Hierarchical Clustering in Machine Learning

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

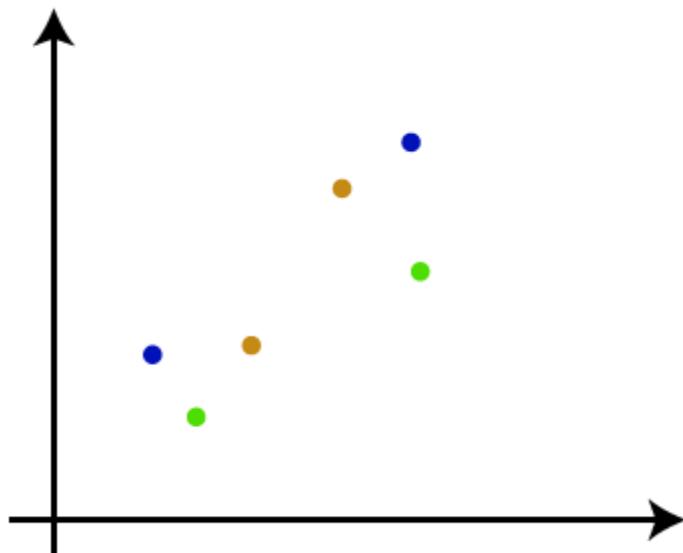
The agglomerative hierarchical clustering algorithm is a popular example of HCA. To group the datasets into clusters, it follows the **bottom-up approach**. It means, this algorithm considers each dataset as a single cluster at the beginning, and then start combining the closest pair of clusters together.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

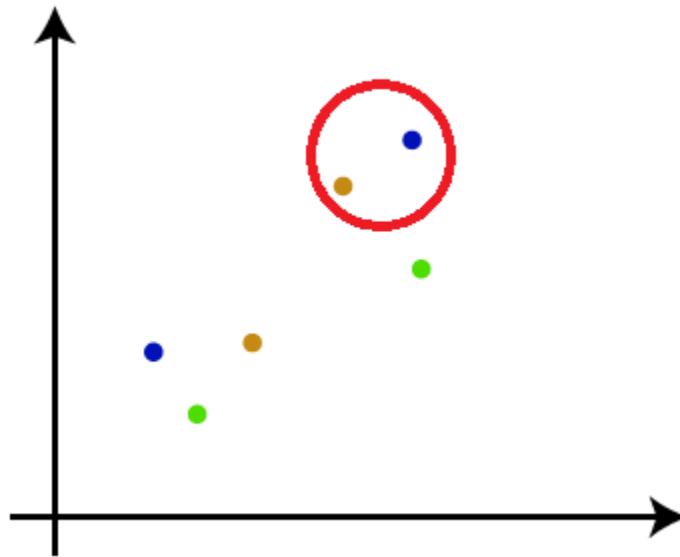
How the Agglomerative Hierarchical clustering Work?

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

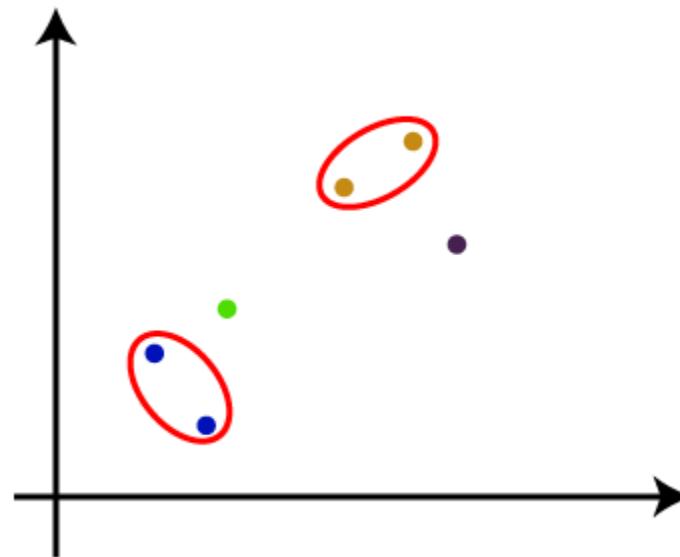
- **Step-1:** Create each data point as a single cluster. Let's say there are N data points, so the number of clusters will also be N .



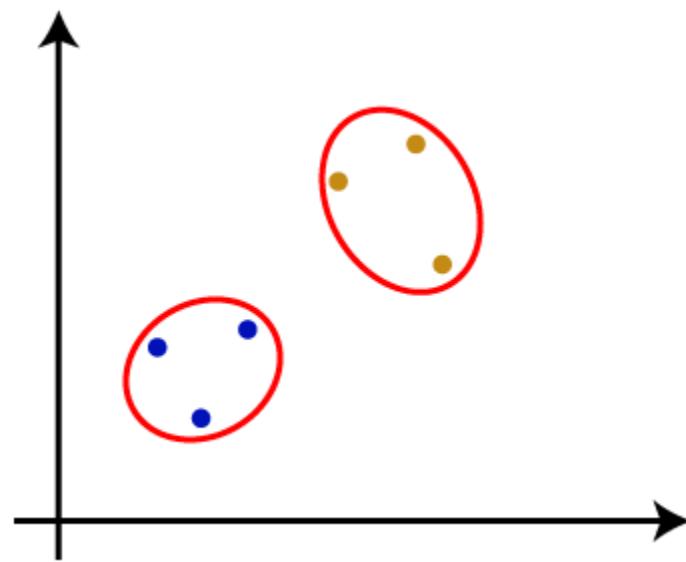
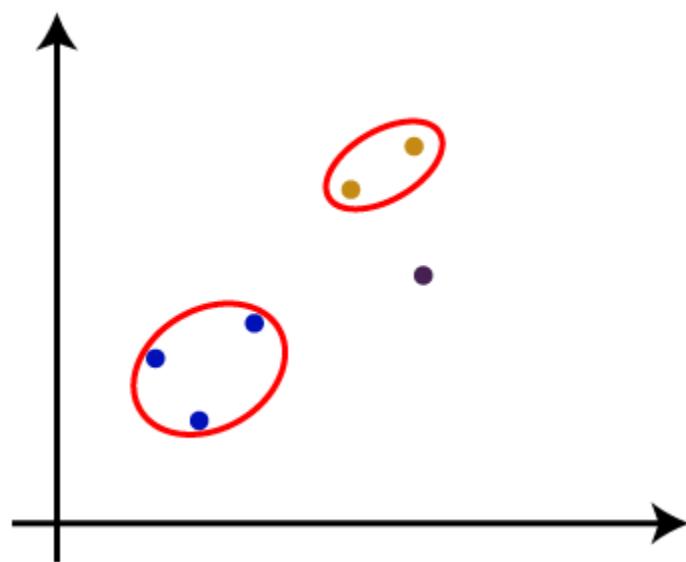
- **Step-2:** Take two closest data points or clusters and merge them to form one cluster. So, there will now be $N-1$ clusters.

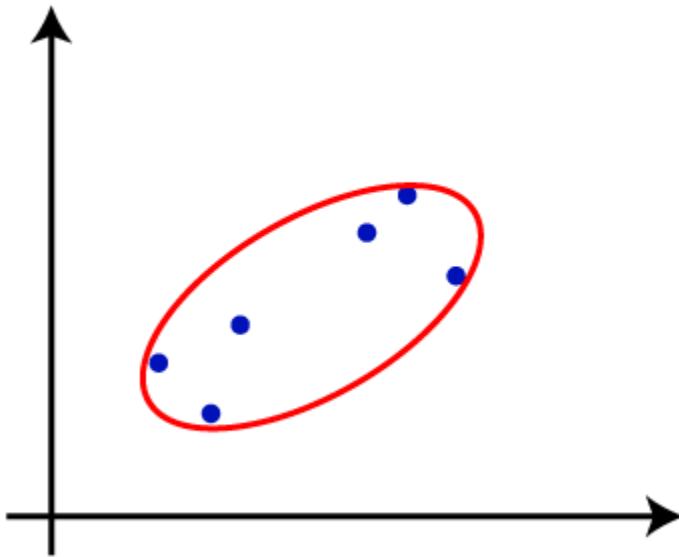


- **Step-3:** Again, take the two closest clusters and merge them together to form one cluster. There will be $N-2$ clusters.



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





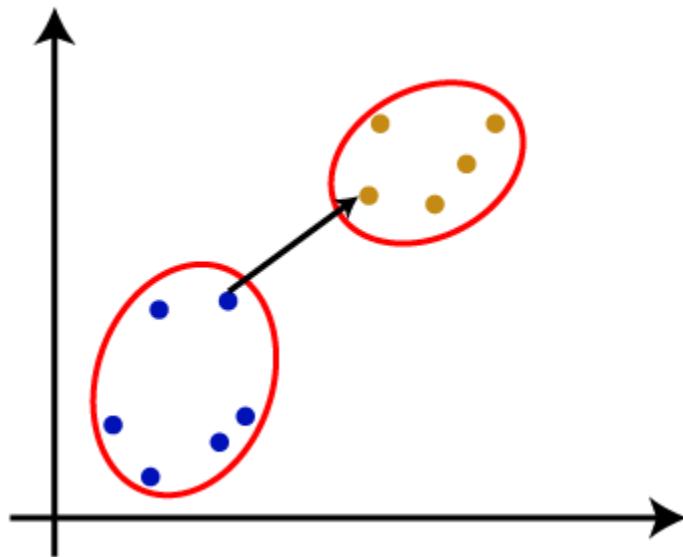
- **Step-5:** Once all the clusters are combined into one big cluster, develop the dendrogram to divide the clusters as per the problem.

Note: To better understand hierarchical clustering, it is advised to have a look on k-means clustering

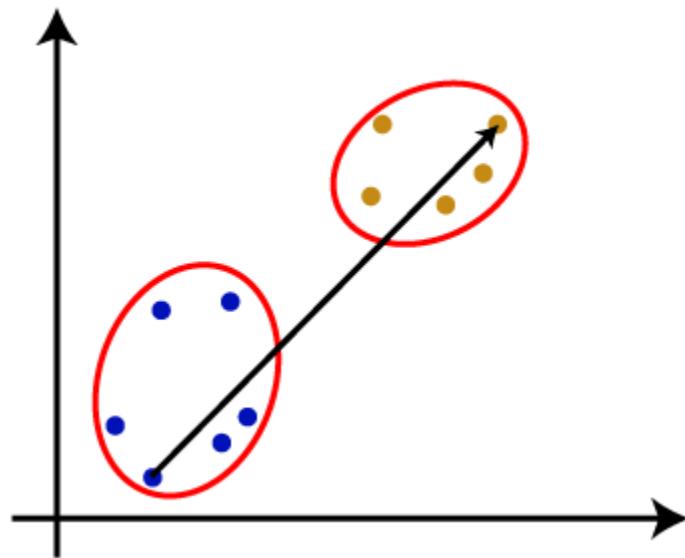
Measure for the distance between two clusters

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:

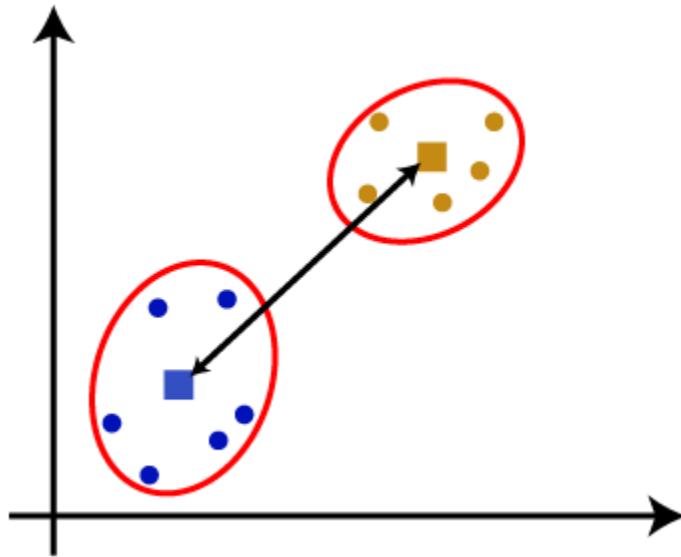


2. **Complete Linkage:** It is the farthest distance between the two points of two different clusters. It is one of the popular linkage methods as it forms tighter clusters than single-linkage.



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:

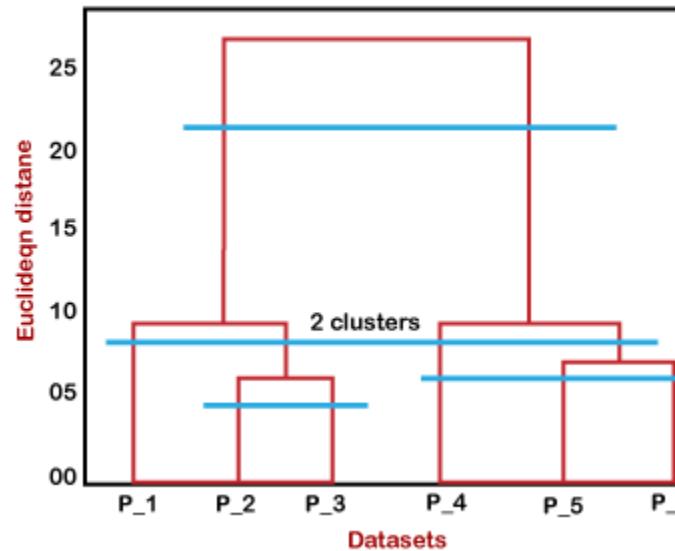
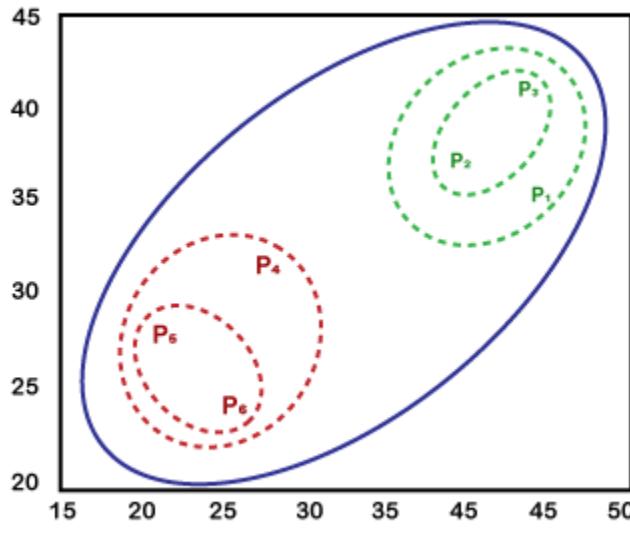


From the above-given approaches, we can apply any of them according to the type of problem or business requirement.

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 dendograms 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.

We can cut the dendrogram tree structure at any level as per our requirement.

Python Implementation of Agglomerative Hierarchical Clustering

Now we will see the practical implementation of the agglomerative hierarchical clustering algorithm using Python. To implement this, we will use the same dataset problem that we have used in the previous topic of K-means clustering so that we can compare both concepts easily.

The dataset is containing the information of customers that have visited a mall for shopping. So, the mall owner wants to find some patterns or some particular behavior of his customers using the dataset information.

Steps for implementation of AHC using Python:

The steps for implementation will be the same as the k-means clustering, except for some changes such as the method to find the number of clusters. Below are the steps:

1. **Data Pre-processing**
2. **Finding the optimal number of clusters using the Dendrogram**
3. **Training the hierarchical clustering model**
4. **Visualizing the clusters**

Data Pre-processing Steps:

In this step, we will import the libraries and datasets for our model.

- o **Importing the libraries**

1. # Importing the libraries
2. **import** numpy as nm
3. **import** matplotlib.pyplot as mtp
4. **import** pandas as pd

The above lines of code are used to import the libraries to perform specific tasks, such as **numpy** for the Mathematical operations, **matplotlib** for drawing the graphs or scatter plot, and **pandas** for importing the dataset.

- o **Importing the dataset**

1. # Importing the dataset
2. dataset = pd.read_csv('Mall_Customers_data.csv')

As discussed above, we have imported the same dataset of **Mall_Customers_data.csv**, as we did in k-means clustering. Consider the below output:

dataset - DataFrame

Index	CustomerID	Genre	Age	Annual Income (k\$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40
5	6	Female	22	17	76
6	7	Female	35	18	6
7	8	Female	23	18	94
8	9	Male	64	19	3
9	10	Female	30	19	72
10	11	Male	67	19	14
11	12	Female	35	19	99
12	13	Female	58	20	15
13	14	Female	24	20	77
14	15	Male	37	20	13

- **Extracting the matrix of features**

Here we will extract only the matrix of features as we don't have any further information about the dependent variable. Code is given below:

1. `x = dataset.iloc[:, [3, 4]].values`

Here we have extracted only 3 and 4 columns as we will use a 2D plot to see the clusters. So, we are considering the Annual income and spending score as the matrix of features.

Step-2: Finding the optimal number of clusters using the Dendrogram

Now we will find the optimal number of clusters using the Dendrogram for our model. For this, we are going to use **scipy** library as it provides a function that will directly return the dendrogram for our code. Consider the below lines of code:

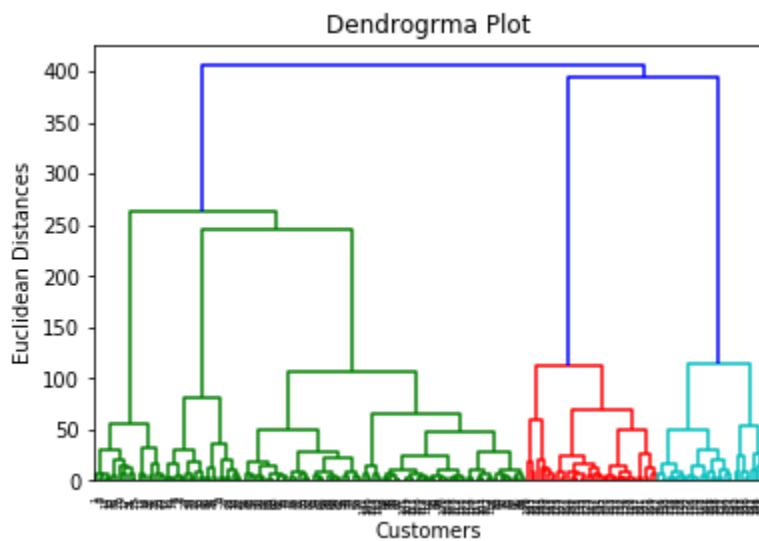
1. #Finding the optimal number of clusters using the dendrogram
2. **import** scipy.cluster.hierarchy as shc
3. dendro = shc.dendrogram(shc.linkage(x, method="ward"))
4. mtp.title("Dendrogram Plot")
5. mtp.ylabel("Euclidean Distances")
6. mtp.xlabel("Customers")
7. mtp.show()

In the above lines of code, we have imported the **hierarchy** module of **scipy** library. This module provides us a method **shc.dendrogram()**, which takes the **linkage()** as a parameter. The linkage function is used to define the distance between two clusters, so here we have passed the **x**(matrix of features), and method "**ward**," the popular method of linkage in hierarchical clustering.

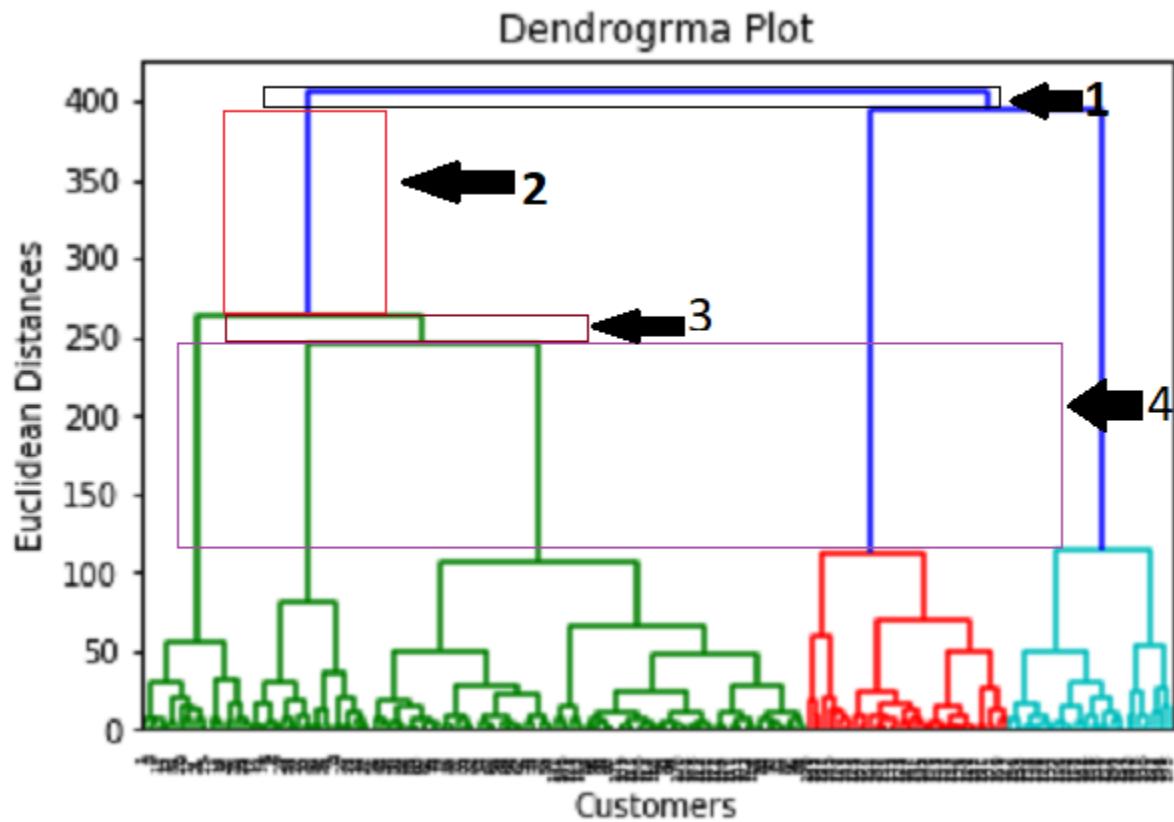
The remaining lines of code are to describe the labels for the dendrogram plot.

Output:

By executing the above lines of code, we will get the below output:



Using this Dendrogram, we will now determine the optimal number of clusters for our model. For this, we will find the **maximum vertical distance** that does not cut any horizontal bar. Consider the below diagram:



In the above diagram, we have shown the vertical distances that are not cutting their horizontal bars. As we can visualize, the 4th distance is looking the maximum, so according to this, **the number of clusters will be 5**(the vertical lines in this range). We can also take the 2nd number as it approximately equals the 4th distance, but we will consider the 5 clusters because the same we calculated in the K-means algorithm.

So, the optimal number of clusters will be 5, and we will train the model in the next step, using the same.

Step-3: Training the hierarchical clustering model

As we know the required optimal number of clusters, we can now train our model. The code is given below:

1. #training the hierarchical model on dataset
2. from sklearn.cluster **import** AgglomerativeClustering
3. hc= AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='ward')
4. y_pred= hc.fit_predict(x)

In the above code, we have imported the **AgglomerativeClustering** class of cluster module of scikit learn library.

Then we have created the object of this class named as **hc**. The AgglomerativeClustering class takes the following parameters:

- **n_clusters=5**: It defines the number of clusters, and we have taken here 5 because it is the optimal number of clusters.
- **affinity='euclidean'**: It is a metric used to compute the linkage.
- **linkage='ward'**: It defines the linkage criteria, here we have used the "ward" linkage. This method is the popular linkage method that we have already used for creating the Dendrogram. It reduces the variance in each cluster.

In the last line, we have created the dependent variable `y_pred` to fit or train the model. It does train not only the model but also returns the clusters to which each data point belongs.

After executing the above lines of code, if we go through the variable explorer option in our Spyder IDE, we can check the `y_pred` variable. We can compare the original dataset with the `y_pred` variable. Consider the below image:



As we can see in the above image, the **y_pred** shows the clusters value, which means the customer id 1 belongs to the 5th cluster (as indexing starts from 0, so 4 means 5th cluster), the customer id 2 belongs to 4th cluster, and so on.

Step-4: Visualizing the clusters

As we have trained our model successfully, now we can visualize the clusters corresponding to the dataset.

Here we will use the same lines of code as we did in k-means clustering, except one change. Here we will not plot the centroid that we did in k-means, because here we have used dendrogram to determine the optimal number of clusters. The code is given below:

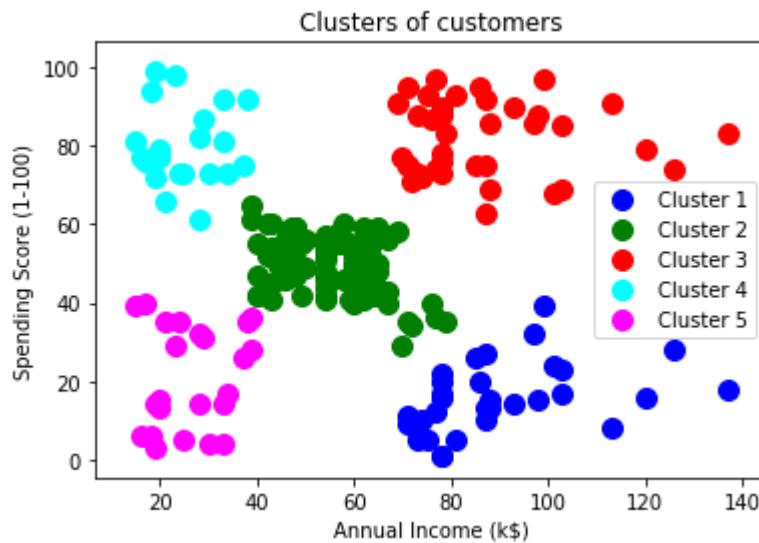
1. #visualizing the clusters
2. `mp.scatter(x[y_pred == 0, 0], x[y_pred == 0, 1], s = 100, c = 'blue', label = 'Cluster 1')`

```

3.     mtp.scatter(x[y_pred == 1, 0], x[y_pred == 1, 1], s = 100, c = 'green', label = 'Cluster 2')
4.     mtp.scatter(x[y_pred == 2, 0], x[y_pred == 2, 1], s = 100, c = 'red', label = 'Cluster 3')
5.     mtp.scatter(x[y_pred == 3, 0], x[y_pred == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
6.     mtp.scatter(x[y_pred == 4, 0], x[y_pred == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')
7.     mtp.title('Clusters of customers')
8.     mtp.xlabel('Annual Income (k$)')
9.     mtp.ylabel('Spending Score (1-100)')
10.    mtp.legend()
11.    mtp.show()

```

Output: By executing the above lines of code, we will get the below output:



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.

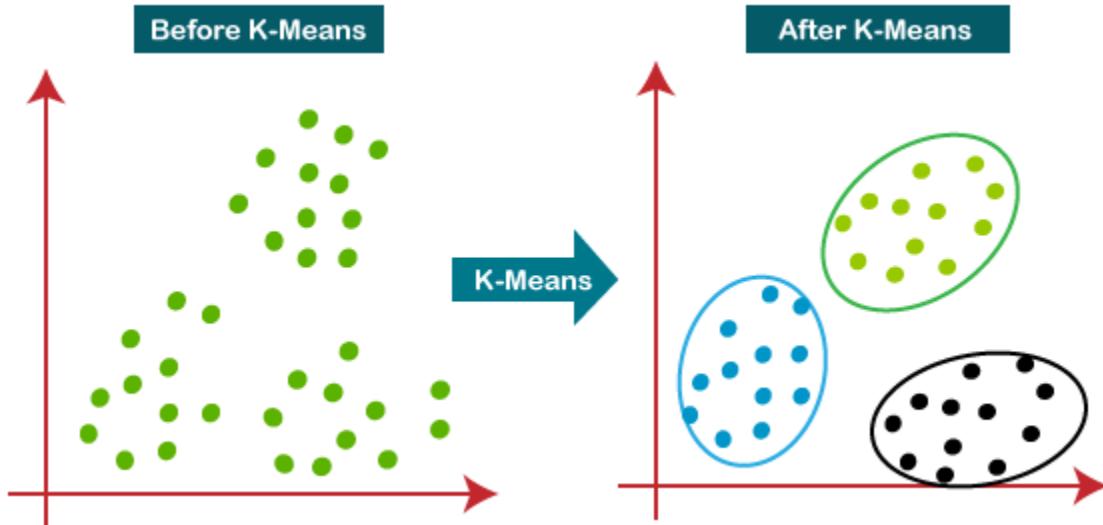
The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

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.

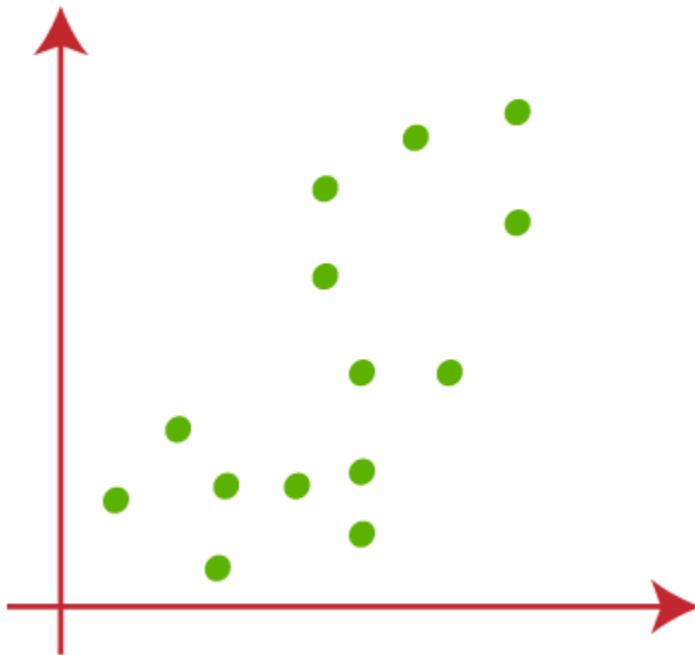
Step-5: Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.

Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.

Step-7: The model is ready.

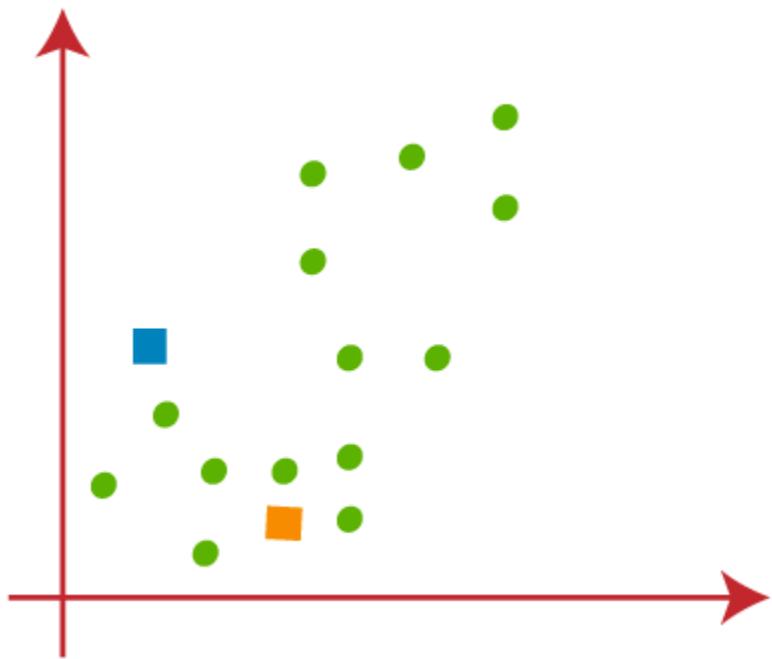
Let's understand the above steps by considering the visual plots:

Suppose we have two variables M1 and M2. The x-y axis scatter plot of these two variables is given below:



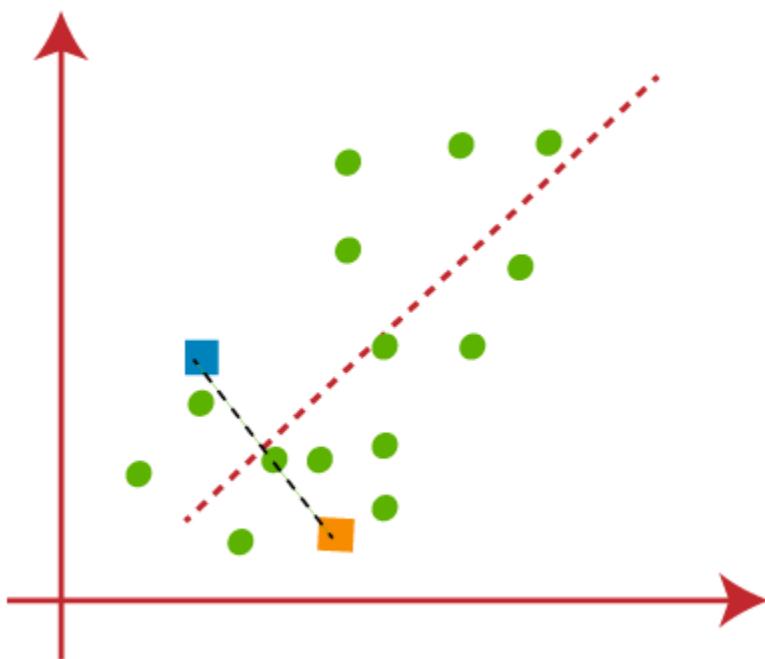
- Let's take number k of clusters, i.e., $K=2$, to identify the dataset and to put them into different clusters. It means here we will try to group these datasets into two different clusters.
- We need to choose some random k points or centroid to form the cluster. These points can be either the points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

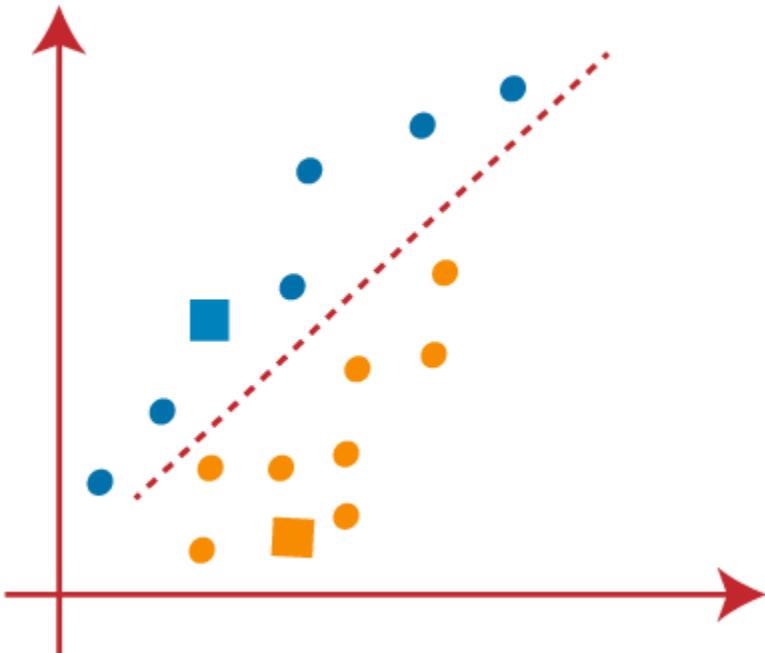


- Now we will assign each data point of the scatter plot to its closest K-point or centroid. We will compute it by applying some mathematics that we have studied to calculate the distance between two points. So, we will draw a median between

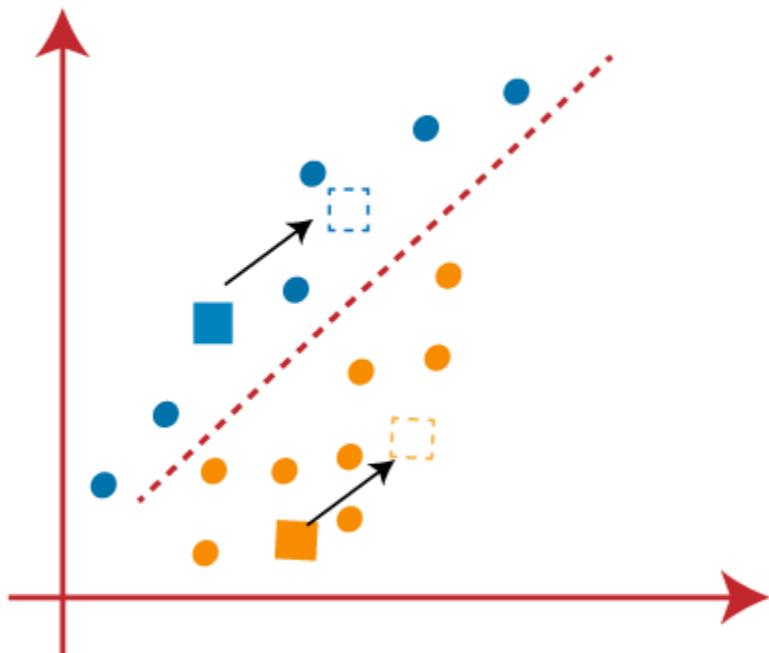
both the centroids. Consider the below image:



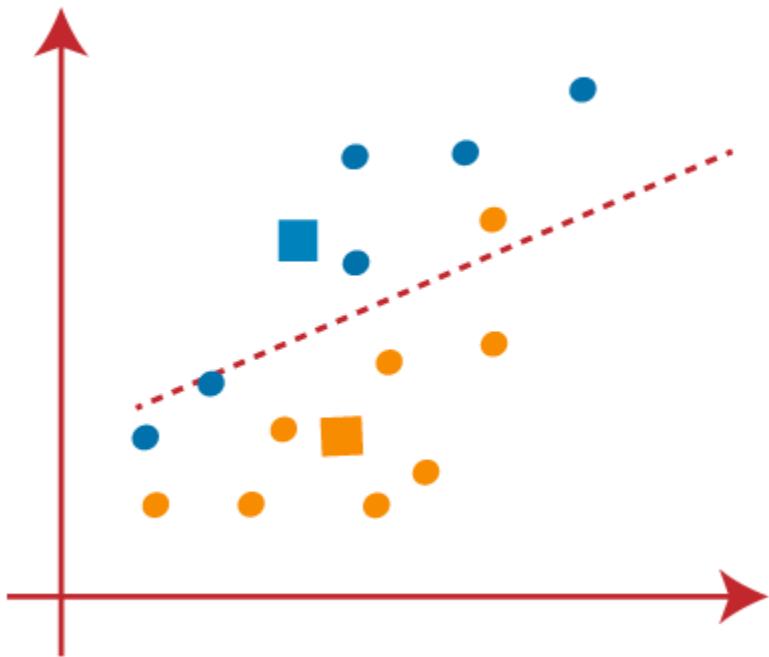
From the above image, it is clear that points left side of the line is near to the K1 or blue centroid, and points to the right of the line are close to the yellow centroid. Let's color them as blue and yellow for clear visualization.



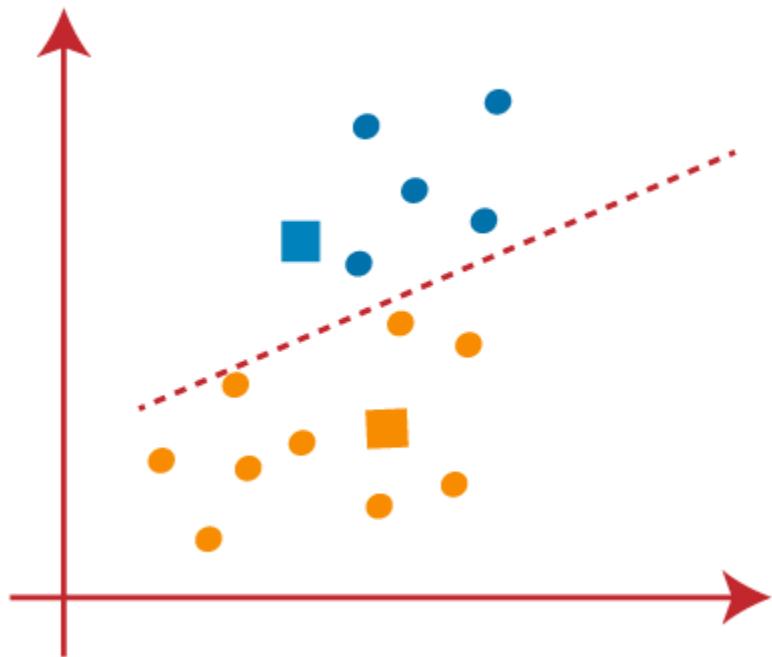
- As we need to find the closest cluster, so we will repeat the process by choosing **a new centroid**. To choose the new centroids, we will compute the center of gravity of these centroids, and will find new centroids as below:



- 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.

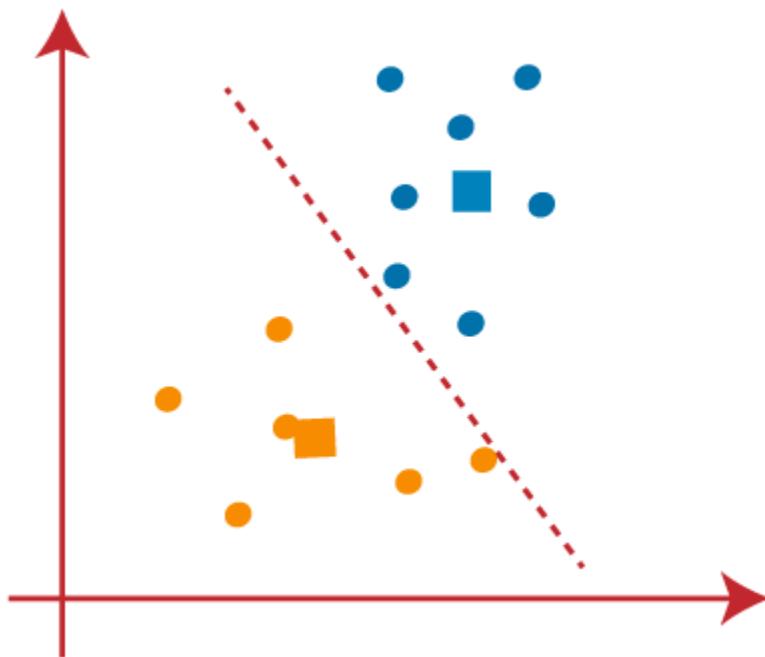


As reassignment has taken place, so we will again go to the step-4, which is finding new centroids or K-points.

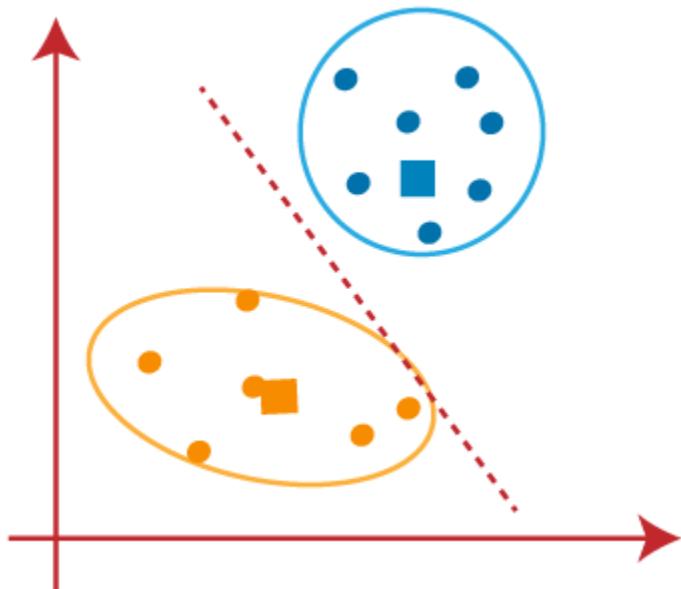
- We will repeat the process by finding the center of gravity of centroids, so the new centroids will be as shown in the below image:



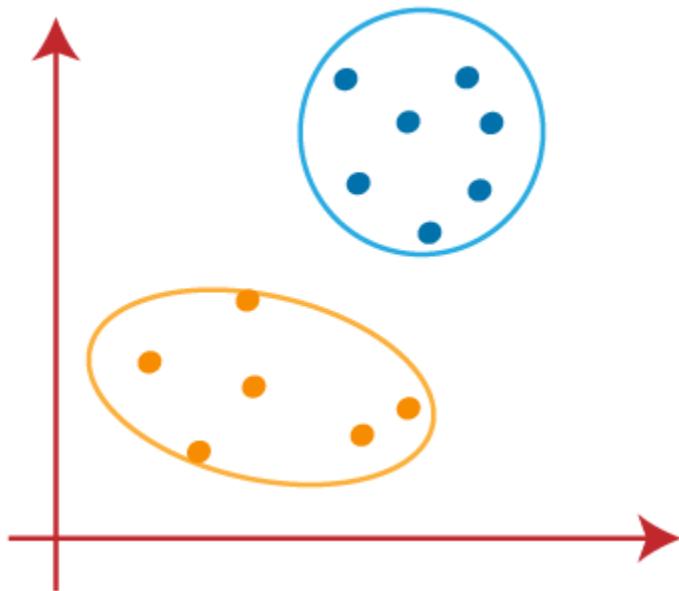
- As we got the new centroids so again will draw the median line and reassign the data points. So, the image will be:



- We can see in the above image; there are no dissimilar data points on either side of the line, which means our model is formed. Consider the below image:



As our model is ready, so we can now remove the assumed centroids, and the two final clusters will be as shown in the below image:



How to choose the value of "K number of clusters" in K-means Clustering?

The performance of the K-means clustering algorithm depends upon highly efficient clusters that it forms. But choosing the optimal number of clusters is a big task. There are some different ways to find the optimal number of clusters, but here we are discussing the most appropriate method to find the number of clusters or value of K. The method is given below:

Elbow Method

The Elbow method is one of the most popular ways to find the optimal number of clusters. This method uses the concept of WCSS value. **WCSS** stands for **Within Cluster Sum of Squares**, which defines the total variations within a cluster. The formula to calculate the value of WCSS (for 3 clusters) is given below:

$$\text{WCSS} = \sum_{\text{Pi in Cluster1}} \text{distance}(\text{Pi} | \text{C}_1)^2 + \sum_{\text{Pi in Cluster2}} \text{distance}(\text{Pi} | \text{C}_2)^2 + \sum_{\text{Pi in Cluster3}} \text{distance}(\text{Pi} | \text{C}_3)^2$$

In the above formula of WCSS,

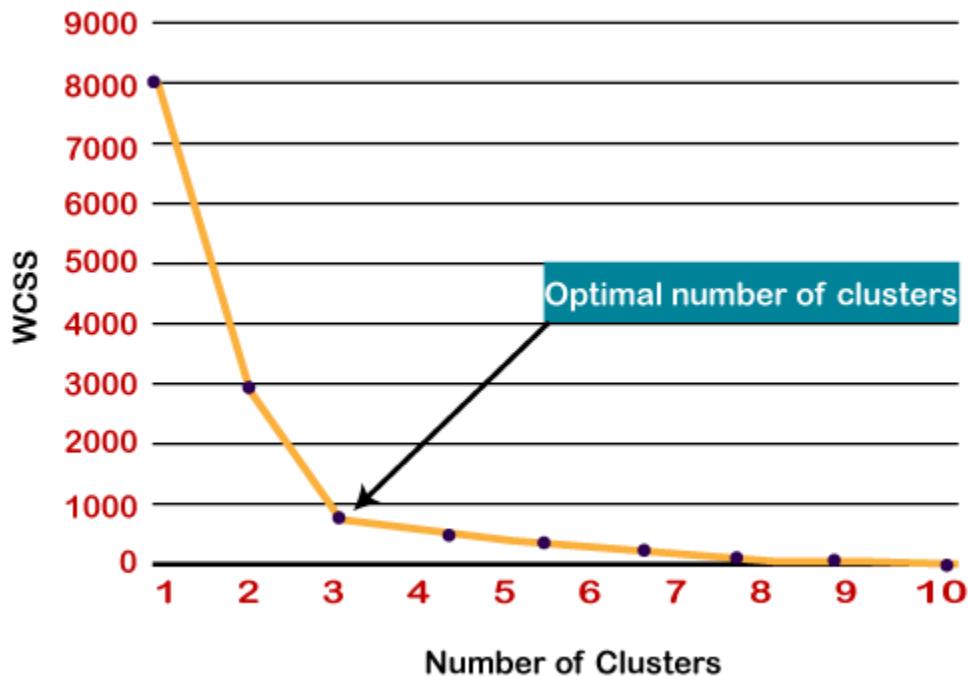
$\sum_{\text{Pi in Cluster1}} \text{distance}(\text{Pi} | \text{C}_1)^2$: It is the sum of the square of the distances between each data point and its centroid within a cluster1 and the same for the other two terms.

To measure the distance between data points and centroid, we can use any method such as Euclidean distance or Manhattan distance.

To find the optimal value of clusters, the elbow method follows the below steps:

- It executes the K-means clustering on a given dataset for different K values (ranges from 1-10).
- For each value of K, calculates the WCSS value.
- Plots a curve between calculated WCSS values and the number of clusters K.
- The sharp point of bend or a point of the plot looks like an arm, then that point is considered as the best value of K.

Since the graph shows the sharp bend, which looks like an elbow, hence it is known as the elbow method. The graph for the elbow method looks like the below image:



Note: We can choose the number of clusters equal to the given data points. If we choose the number of clusters equal to the data points, then the value of WCSS becomes zero, and that will be the endpoint of the plot.

Python Implementation of K-means Clustering Algorithm

In the above section, we have discussed the K-means algorithm, now let's see how it can be implemented using [Python](#).

Before implementation, let's understand what type of problem we will solve here. So, we have a dataset of **Mall_Customers**, which is the data of customers who visit the mall and spend there.

In the given dataset, we have **Customer_Id, Gender, Age, Annual Income (\$), and Spending Score** (which is the calculated value of how much a customer has spent in the mall, the more the value, the more he has spent). From this dataset, we need to calculate some patterns, as it is an unsupervised method, so we don't know what to calculate exactly.

The steps to be followed for the implementation are given below:

- **Data Pre-processing**
- **Finding the optimal number of clusters using the elbow method**
- **Training the K-means algorithm on the training dataset**
- **Visualizing the clusters**

Step-1: Data pre-processing Step

The first step will be the data pre-processing, as we did in our earlier topics of Regression and Classification. But for the clustering problem, it will be different from other models. Let's discuss it:

- **Importing Libraries**

As we did in previous topics, firstly, we will import the libraries for our model, which is part of data pre-processing. The code is given below:

1. `# importing libraries`
2. `import numpy as nm`
3. `import matplotlib.pyplot as mtp`
4. `import pandas as pd`

In the above code, the **numpy** we have imported for the performing mathematics calculation, **matplotlib** is for plotting the graph, and **pandas** are for managing the dataset.

- **Importing the Dataset:**

Next, we will import the dataset that we need to use. So here, we are using the **Mall_Customer_data.csv** dataset. It can be imported using the below code:

1. `# Importing the dataset`
2. `dataset = pd.read_csv('Mall_Customers_data.csv')`

By executing the above lines of code, we will get our dataset in the Spyder IDE. The dataset looks like the below image:

Index	CustomerID	Genre	Age	Annual Income (k\$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40
5	6	Female	22	17	76
6	7	Female	35	18	6
7	8	Female	23	18	94
8	9	Male	64	19	3
9	10	Female	30	19	72
10	11	Male	67	19	14
11	12	Female	35	19	99
12	13	Female	58	20	15
13	14	Female	24	20	77
14	15	Male	37	20	13
15	16	Male	22	20	79

From the above dataset, we need to find some patterns in it.

- **Extracting Independent Variables**

Here we don't need any dependent variable for data pre-processing step as it is a clustering problem, and we have no idea about what to determine. So we will just add a line of code for the matrix of features.

1. `x = dataset.iloc[:, [3, 4]].values`

As we can see, we are extracting only 3rd and 4th feature. It is because we need a 2d plot to visualize the model, and some features are not required, such as customer_id.

Step-2: Finding the optimal number of clusters using the elbow method

In the second step, we will try to find the optimal number of clusters for our clustering problem. So, as discussed above, here we are going to use the elbow method for this purpose.

As we know, the elbow method uses the WCSS concept to draw the plot by plotting WCSS values on the Y-axis and the number of clusters on the X-axis. So we are going to calculate the value for WCSS for different k values ranging from 1 to 10. Below is the code for it:

```
1.      #finding optimal number of clusters using the elbow method
2.      from sklearn.cluster import KMeans
3.      wcss_list= [] #Initializing the list for the values of WCSS
4.
5.      #Using for loop for iterations from 1 to 10.
6.      for i in range(1, 11):
7.          kmeans = KMeans(n_clusters=i, init='k-means++', random_state= 42)
8.          kmeans.fit(x)
9.          wcss_list.append(kmeans.inertia_)
10.         mtp.plot(range(1, 11), wcss_list)
11.         mtp.title('The Elbow Method Graph')
12.         mtp.xlabel('Number of clusters(k)')
13.         mtp.ylabel('wcss_list')
14.         mtp.show()
```

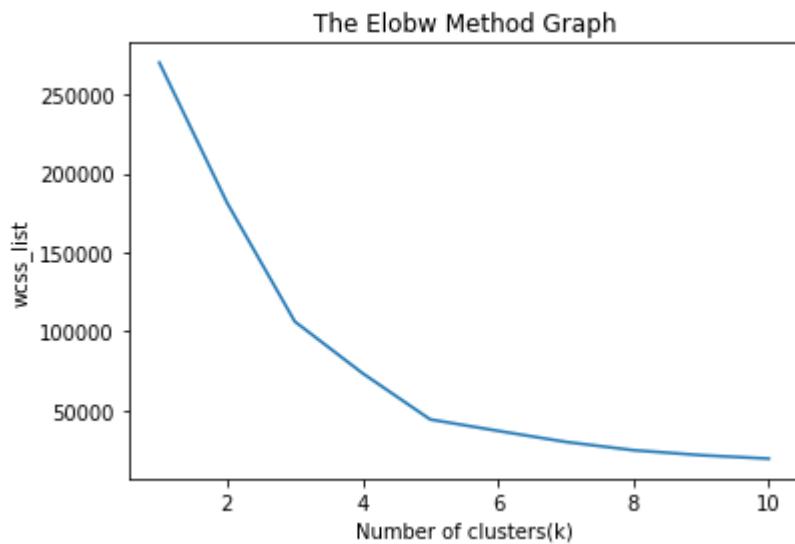
As we can see in the above code, we have used **the KMeans** class of `sklearn.cluster` library to form the clusters.

Next, we have created the **wcss_list** variable to initialize an empty list, which is used to contain the value of wcss computed for different values of k ranging from 1 to 10.

After that, we have initialized the for loop for the iteration on a different value of k ranging from 1 to 10; since for loop in Python, exclude the outbound limit, so it is taken as 11 to include 10th value.

The rest part of the code is similar as we did in earlier topics, as we have fitted the model on a matrix of features and then plotted the graph between the number of clusters and WCSS.

Output: After executing the above code, we will get the below output:



From the above plot, we can see the elbow point is at **5. So the number of clusters here will be 5.**

wcss_list - List (10 elements)

Index	Type	Size	Value
0	float64	1	269981.28
1	float64	1	181363.59595959596
2	float64	1	106348.37306211118
3	float64	1	73679.78903948834
4	float64	1	44448.45544793371
5	float64	1	37233.81451071001
6	float64	1	30259.65720728547
7	float64	1	25011.83934915659
8	float64	1	21850.165282585633
9	float64	1	19672.07284901432

Save and Close Close

Step- 3: Training the K-means algorithm on the training dataset

As we have got the number of clusters, so we can now train the model on the dataset.

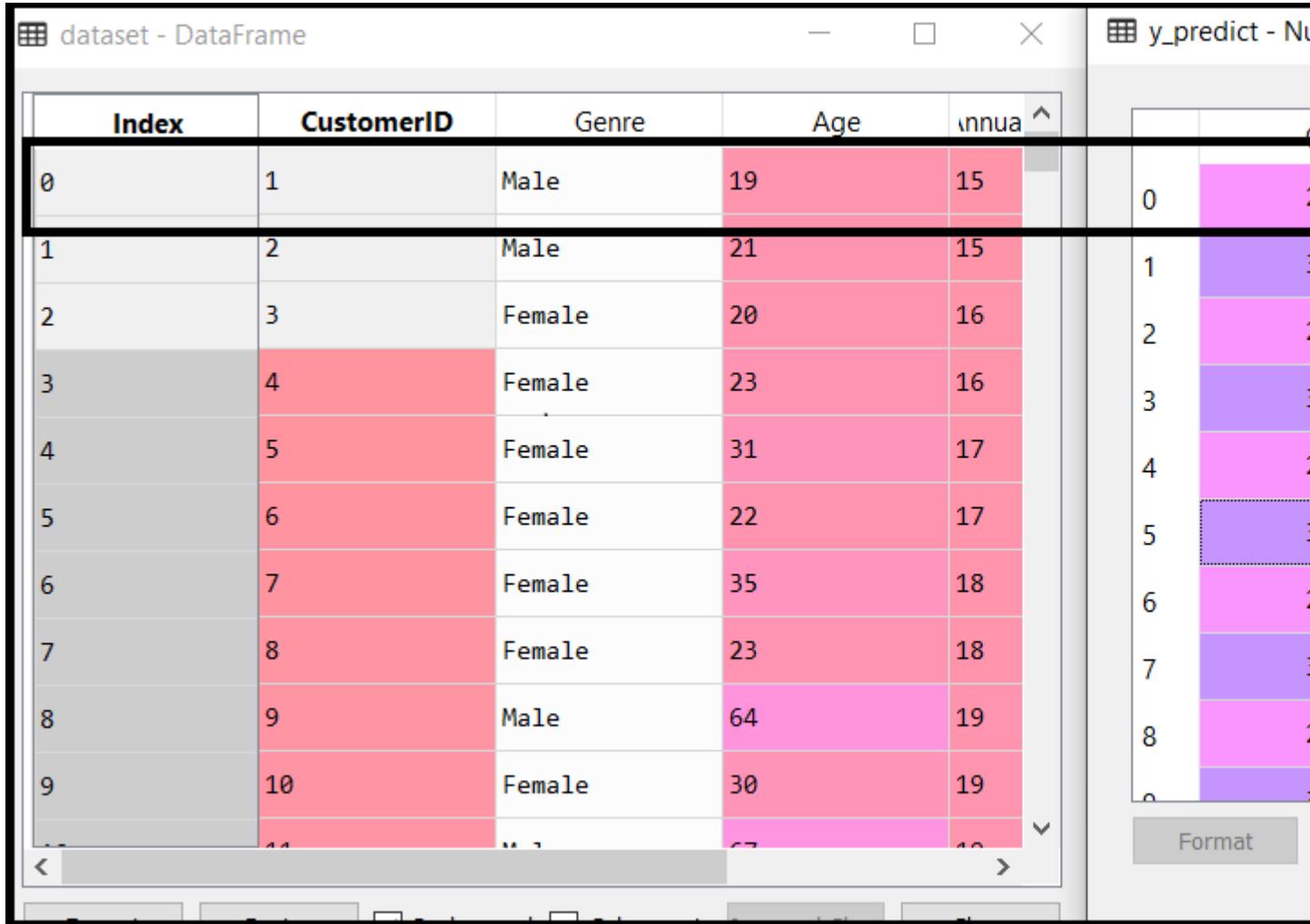
To train the model, we will use the same two lines of code as we have used in the above section, but here instead of using i, we will use 5, as we know there are 5 clusters that need to be formed. The code is given below:

1. #training the K-means model on a dataset
2. kmeans = KMeans(n_clusters=5, init='k-means++', random_state= 42)
3. y_predict= kmeans.fit_predict(x)

The first line is the same as above for creating the object of KMeans class.

In the second line of code, we have created the dependent variable **y_predict** to train the model.

By executing the above lines of code, we will get the **y_predict** variable. We can check it under **the variable explorer** option in the Spyder IDE. We can now compare the values of **y_predict** with our original dataset. Consider the below image:



From the above image, we can now relate that the CustomerID 1 belongs to a cluster 3(as index starts from 0, hence 2 will be considered as 3), and 2 belongs to cluster 4, and so on.

Step-4: Visualizing the Clusters

The last step is to visualize the clusters. As we have 5 clusters for our model, so we will visualize each cluster one by one.

To visualize the clusters will use scatter plot using `mpf.scatter()` function of matplotlib.

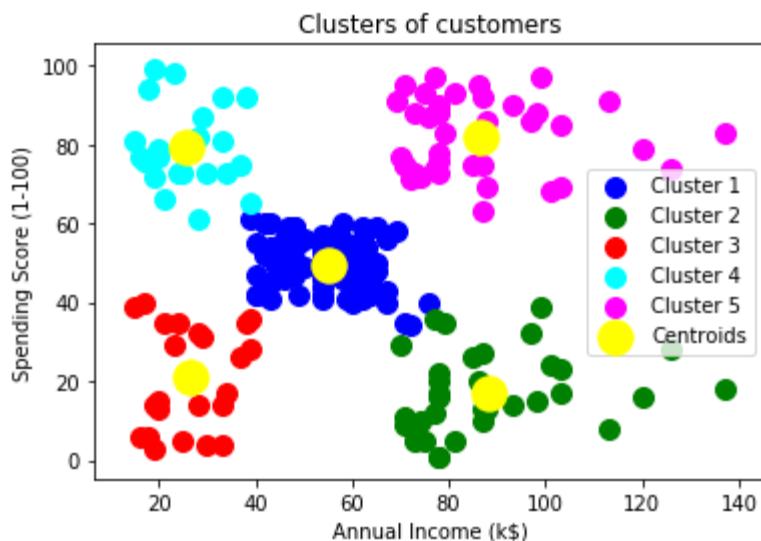
```

1.      #visualizing the clusters
2.      mtp.scatter(x[y_predict == 0, 0], x[y_predict == 0, 1], s = 100, c = 'blue', label
   l = 'Cluster 1') #for first cluster
3.      mtp.scatter(x[y_predict == 1, 0], x[y_predict == 1, 1], s = 100, c = 'green', label
   el = 'Cluster 2') #for second cluster
4.      mtp.scatter(x[y_predict == 2, 0], x[y_predict == 2, 1], s = 100, c = 'red', label
   = 'Cluster 3') #for third cluster
5.      mtp.scatter(x[y_predict == 3, 0], x[y_predict == 3, 1], s = 100, c = 'cyan', label
   l = 'Cluster 4') #for fourth cluster
6.      mtp.scatter(x[y_predict == 4, 0], x[y_predict == 4, 1], s = 100, c = 'magenta',
   label = 'Cluster 5') #for fifth cluster
7.      mtp.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 3
   00, c = 'yellow', label = 'Centroid')
8.      mtp.title('Clusters of customers')
9.      mtp.xlabel('Annual Income (k$)')
10.     mtp.ylabel('Spending Score (1-100)')
11.     mtp.legend()
12.     mtp.show()

```

In above lines of code, we have written code for each clusters, ranging from 1 to 5. The first coordinate of the `mtp.scatter`, i.e., `x[y_predict == 0, 0]` containing the x value for the showing the matrix of features values, and the `y_predict` is ranging from 0 to 1.

Output:



The output image is clearly showing the five different clusters with different colors. The clusters are formed between two parameters of the dataset; Annual income of customer and Spending. We can change the colors and labels as per the requirement or choice. We can also observe some points from the above patterns, which are given below:

- **Cluster1** shows the customers with average salary and average spending so we can categorize these customers as
- Cluster2 shows the customer has a high income but low spending, so we can categorize them as **careful**.
- Cluster3 shows the low income and also low spending so they can be categorized as sensible.
- Cluster4 shows the customers with low income with very high spending so they can be categorized as **careless**.
- Cluster5 shows the customers with high income and high spending so they can be categorized as target, and these customers can be the most profitable customers for the mall owner.

Silhouette Algorithm

One of the fundamental steps of an unsupervised learning algorithm is to determine the number of clusters into which the data may be divided. The silhouette algorithm is one of the many algorithms to determine the optimal number of clusters for an unsupervised learning technique.

In the Silhouette algorithm, we assume that the data has already been clustered into k clusters by a clustering technique(Typically K-Means Clustering technique). Then for each data point, we define the following:-

C(i) -The cluster assigned to the ith data point

$|C(i)|$ – The number of data points in the cluster assigned to the ith data point

$a(i)$ – It gives a measure of how well assigned the ith data point is to it's cluster

$b(i)$ – It is defined as the average dissimilarity to the closest cluster which is not it's cluster

The silhouette coefficient $s(i)$ is given by:-

We determine the average silhouette for each value of k and for the value of k which has the maximum value of $s(i)$ is considered the optimal number of clusters for the unsupervised learning algorithm.

Let us consider the following data:-

S.No	X1	X2
1.	-7.36	6.37
2.	3.08	-6.78
3.	5.03	-8.31
4.	-1.93	-0.92
5.	-8.86	6.60

We now iterate the values of k from 2 to 5. We assume that no practical data exists for which all the data points can be optimally clustered into 1 cluster.

ADVERTISING

We construct the following tables for each value of k :-

$k = 2$

S.No	$a(i)$	$b(i)$	$s(i)$
1.	5.31	14.1	0.62
2.	2.47	13.15	0.81
3.	2.47	14.97	0.84
4.	9.66	8.93	-0.076
5.	5.88	19.16	0.69

Average value of $s(i) = 0.58$

$k = 3$

S.No	a(i)	b(i)	s(i)
1.	1.52	9.09	0.83
2.	2.47	7.71	0.68
3.	2.47	10.15	0.76
4.	0	7.71	1
5.	1.52	17.93	0.92

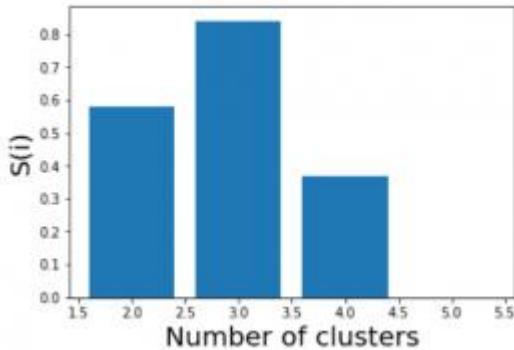
Average value of $s(i) = 0.84$
 $k = 4$

S.No	a(i)	b(i)	s(i)
1.	1.52	9.09	0.83
2.	infinite	2.47	0
3.	infinite	2.47	0
4.	infinite	7.71	0
5.	1.52	10.23	0.85

Average value of $s(i) = 0.37$
 $k = 5$

S.No	a(i)	b(i)	s(i)
1.	infinite	1.52	0
2.	infinite	2.47	0
3.	infinite	2.47	0
4.	infinite	7.71	0
5.	infinite	1.52	0

Average value of $s(i) = 0$



We see that the highest value of $s(i)$ exists for $k = 3$. Therefore we conclude that the optimal number of clusters for the given data is 3.

Fuzzy c-means clustering algorithm

This algorithm works by assigning membership to each data point corresponding to each cluster center on the basis

of distance between the cluster center and the data point. More the data is near to the cluster center more is its

membership towards the particular cluster center. Clearly, summation of membership of each data point should be

equal to one. After each iteration membership and cluster centers are updated according to the formula:

$$\mu_{ij} = 1 / \sum_{k=1}^c (d_{ij} / d_{ik})^{(2/m-1)}$$

$$v_j = \left(\sum_{i=1}^n (\mu_{ij})^m x_i \right) / \left(\sum_{i=1}^n (\mu_{ij})^m \right), \forall j = 1, 2, \dots, c$$

where,

' n ' is the number of data points. ' v_j ' represents the j^{th} cluster center. ' m ' is the fuzziness index $m \in [1, \infty]$. ' c ' represents the number of cluster center. ' μ_{ij} ' represents the membership of i^{th} data to j^{th} cluster center. ' d_{ij} ' represents the Euclidean distance between i^{th} data and j^{th} cluster center.

Main objective of fuzzy c-means algorithm is to minimize:

$$J(U, V) = \sum_{i=1}^n \sum_{j=1}^c (\mu_{ij})^m \|x_i - v_j\|^2$$

where,

$\|x_i - v_j\|$ is the Euclidean distance between i^{th} data and j^{th} cluster center.

Algorithmic steps for Fuzzy c-means clustering

Let $X = \{x_1, x_2, x_3, \dots, x_n\}$ be the set of data points and $V = \{v_1, v_2, v_3, \dots, v_c\}$ be the set of centers.

- 1) Randomly select 'c' cluster centers.
- 2) Calculate the fuzzy membership ' μ_{ij} ' using:

$$\mu_{ij} = 1 / \sum_{k=1}^c (d_{ij} / d_{ik})^{(2/m-1)}$$

- 3) Compute the fuzzy centers 'v' using:

$$v_j = \left(\sum_{i=1}^n (\mu_{ij})^m x_i \right) / \left(\sum_{i=1}^n (\mu_{ij})^m \right), \forall j = 1, 2, \dots, c$$

- 4) Repeat step 2) and 3) until the minimum 'J' value is achieved or $\|U^{(k+1)} - U^{(k)}\| < \beta$.

where,

' k ' is the iteration step.

' β ' is the termination criterion between [0, 1].

' $U = (\mu_{ij})^{n \times c}$ ' is the fuzzy membership matrix.

' J ' is the objective function.

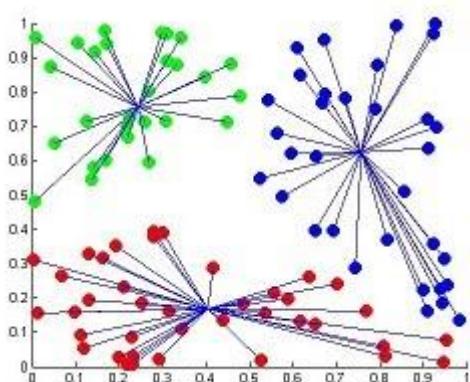


Fig I: Result of Fuzzy c-means clustering

Advantages

- 1) Gives best result for overlapped data set and comparatively better than k-means algorithm.
- 2) Unlike k-means where data point must exclusively belong to one cluster center here data point is assigned membership to each cluster center as a result of which data point may belong to more than one cluster center.

Disadvantages

- 1) Apriori specification of the number of clusters.
- 2) With lower value of β we get the better result but at the expense of more number of iteration.
- 3) Euclidean distance measures can unequally weight underlying factors.

DBSCAN Clustering in ML | Density based clustering

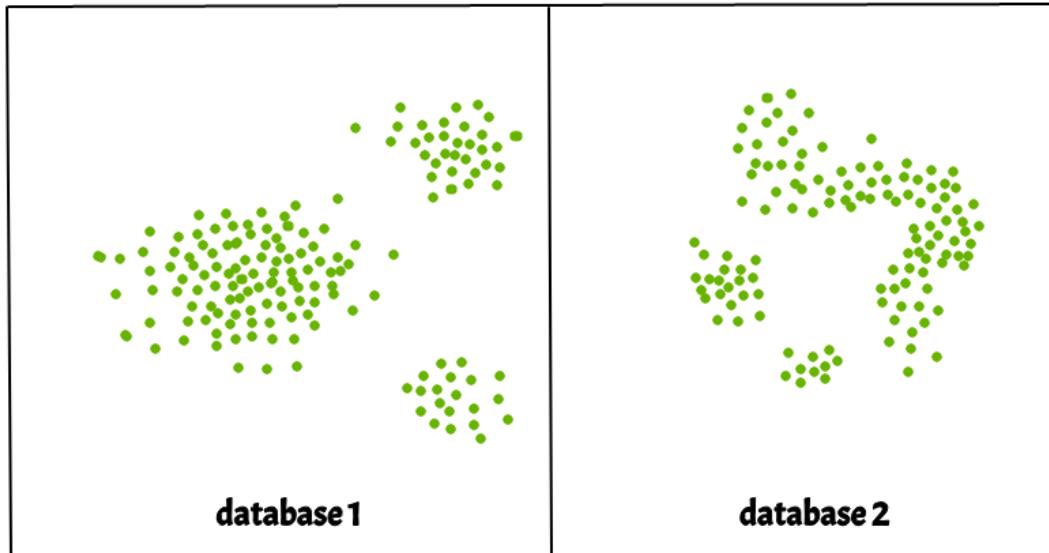
Clustering analysis or simply Clustering is basically an Unsupervised learning method that divides the data points into a number of specific batches or groups, such that the data points in the same groups have similar properties and data points in different groups have different properties in some sense. It comprises of many different methods based on different evolution.

E.g. K-Means (distance between points), Affinity propagation (graph distance), Mean-shift (distance between points), DBSCAN (distance between nearest points), Gaussian mixtures (Mahalanobis distance to centers), Spectral clustering (graph distance) etc.

Fundamentally, all clustering methods use the same approach i.e. first we calculate similarities and then we use it to cluster the data points into groups or batches. Here we will focus on **Density-based spatial clustering of applications with noise** (DBSCAN) clustering method.

Clusters are dense regions in the data space, separated by regions of the lower density of points. The **DBSCAN algorithm** is based on this intuitive notion of “clusters” and “noise”. The key idea is that for each point of a cluster, the

neighborhood of a given radius has to contain at least a minimum number of points.

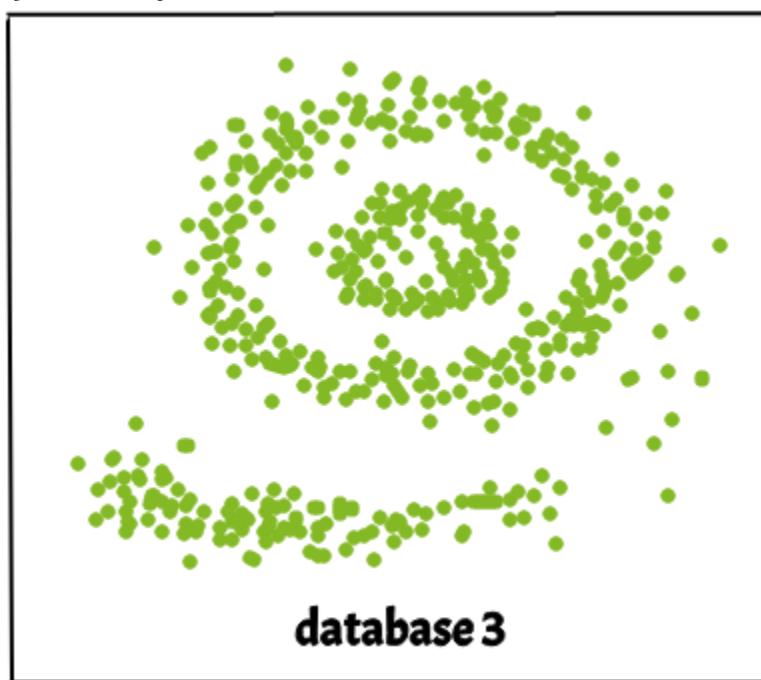


Why DBSCAN ?

Partitioning methods (K-means, PAM clustering) and hierarchical clustering work for finding spherical-shaped clusters or convex clusters. In other words, they are suitable only for compact and well-separated clusters. Moreover, they are also severely affected by the presence of noise and outliers in the data.

Real life data may contain irregularities, like –

- i) Clusters can be of arbitrary shape such as those shown in the figure below.
- ii) Data may contain noise.



The figure below shows a data set containing nonconvex clusters and

outliers/noises. Given such data, k-means algorithm has difficulties for identifying these clusters with arbitrary shapes.

DBSCAN algorithm requires two parameters –

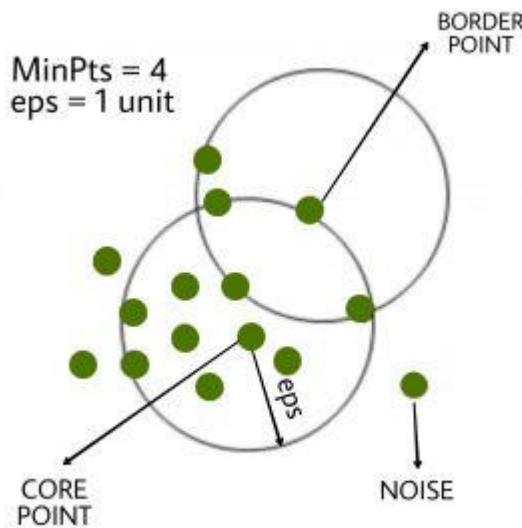
1. **eps** : It defines the neighborhood around a data point i.e. if the distance between two points is lower or equal to 'eps' then they are considered as neighbors. If the eps value is chosen too small then large part of the data will be considered as outliers. If it is chosen very large then the clusters will merge and majority of the data points will be in the same clusters. One way to find the eps value is based on the **k-distance graph**.
2. **MinPts**: Minimum number of neighbors (data points) within eps radius. Larger the dataset, the larger value of MinPts must be chosen. As a general rule, the minimum MinPts can be derived from the number of dimensions D in the dataset as, $\text{MinPts} \geq D+1$. The minimum value of MinPts must be chosen at least 3.

In this algorithm, we have 3 types of data points.

Core Point: A point is a core point if it has more than MinPts points within eps.

Border Point: A point which has fewer than MinPts within eps but it is in the neighborhood of a core point.

Noise or outlier: A point which is not a core point or border point.



DBSCAN algorithm can be abstracted in the following steps –

1. Find all the neighbor points within eps and identify the core points or visited with more than MinPts neighbors.

2. For each core point if it is not already assigned to a cluster, create a new cluster.
 3. Find recursively all its density connected points and assign them to the same cluster as the core point.
- A point a and b are said to be density connected if there exist a point c which has a sufficient number of points in its neighbors and both the points a and b are within the eps distance . This is a chaining process. So, if b is neighbor of c , c is neighbor of d , d is neighbor of e , which in turn is neighbor of a implies that b is neighbor of a .
4. Iterate through the remaining unvisited points in the dataset. Those points that do not belong to any cluster are noise.

Introduction to Dimensionality Reduction Technique

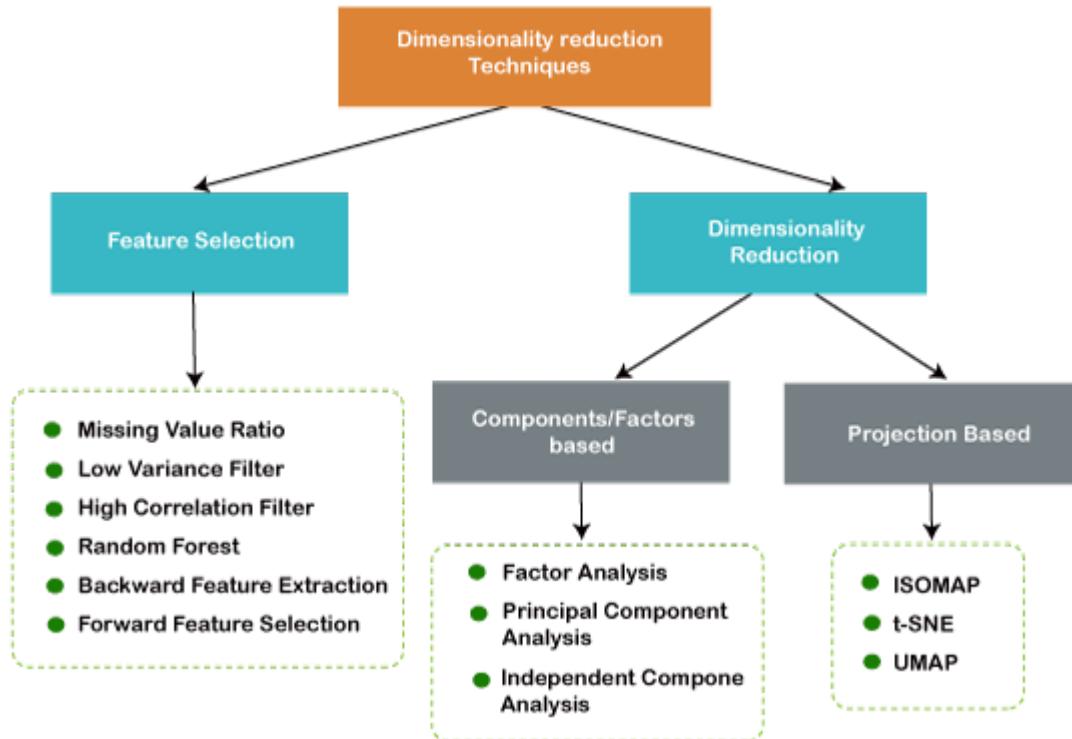
What is Dimensionality Reduction?

The number of input features, variables, or columns present in a given dataset is known as dimensionality, and the process to reduce these features is called dimensionality reduction.

A dataset contains a huge number of input features in various cases, which makes the predictive modeling task more complicated. Because it is very difficult to visualize or make predictions for the training dataset with a high number of features, for such cases, dimensionality reduction techniques are required to use.

Dimensionality reduction technique can be defined as, "***It is a way of converting the higher dimensions dataset into lesser dimensions dataset ensuring that it provides similar information.***" These techniques are widely used in [machine learning](#) for obtaining a better fit predictive model while solving the classification and regression problems.

It is commonly used in the fields that deal with high-dimensional data, such as **speech recognition, signal processing, bioinformatics, etc. It can also be used for data visualization, noise reduction, cluster analysis**, etc.



The Curse of Dimensionality

Handling the high-dimensional data is very difficult in practice, commonly known as the *curse of dimensionality*. If the dimensionality of the input dataset increases, any machine learning algorithm and model becomes more complex. As the number of features increases, the number of samples also gets increased proportionally, and the chance of overfitting also increases. If the machine learning model is trained on high-dimensional data, it becomes overfitted and results in poor performance.

Hence, it is often required to reduce the number of features, which can be done with dimensionality reduction.

Benefits of applying Dimensionality Reduction

Some benefits of applying dimensionality reduction technique to the given dataset are given below:

- By reducing the dimensions of the features, the space required to store the dataset also gets reduced.
- Less Computation training time is required for reduced dimensions of features.
- Reduced dimensions of features of the dataset help in visualizing the data quickly.

- **It removes the redundant features (if present) by taking care of multicollinearity.**

Disadvantages of dimensionality Reduction

There are also some disadvantages of applying the dimensionality reduction, which are given below:

- Some data may be lost due to dimensionality reduction.
- In the PCA dimensionality reduction technique, sometimes the principal components required to consider are unknown.

Approaches of Dimension Reduction

There are two ways to apply the dimension reduction technique, which are given below:

Feature Selection

Feature selection is the process of selecting the subset of the relevant features and leaving out the irrelevant features present in a dataset to build a model of high accuracy. In other words, it is a way of selecting the optimal features from the input dataset.

Three methods are used for the feature selection:

1. Filters Methods

In this method, the dataset is filtered, and a subset that contains only the relevant features is taken. Some common techniques of filters method are:

- **Correlation**
- **Chi-Square Test**
- **ANOVA**
- **Information Gain, etc.**

2. Wrappers Methods

The wrapper method has the same goal as the filter method, but it takes a machine learning model for its evaluation. In this method, some features are fed to the ML model, and evaluate the performance. The performance decides whether to add those features or remove to increase the accuracy of the model. This method is more accurate than the filtering method but complex to work. Some common techniques of wrapper methods are:

- Forward Selection
- Backward Selection
- Bi-directional Elimination

3. Embedded Methods: Embedded methods check the different training iterations of the machine learning model and evaluate the importance of each feature. Some common techniques of Embedded methods are:

- **LASSO**
- **Elastic Net**
- **Ridge Regression, etc.**

Feature Extraction:

Feature extraction is the process of transforming the space containing many dimensions into space with fewer dimensions. This approach is useful when we want to keep the whole information but use fewer resources while processing the information.

Some common feature extraction techniques are:

- a. Principal Component Analysis
- b. Linear Discriminant Analysis
- c. Kernel PCA
- d. Quadratic Discriminant Analysis

Common techniques of Dimensionality Reduction

- a. **Principal Component Analysis**
- b. **Backward Elimination**
- c. **Forward Selection**
- d. **Score comparison**
- e. **Missing Value Ratio**
- f. **Low Variance Filter**
- g. **High Correlation Filter**
- h. **Random Forest**
- i. **Factor Analysis**
- j. **Auto-Encoder**

Principal Component Analysis (PCA)

Principal Component Analysis is a statistical process that converts the observations of correlated features into a set of linearly uncorrelated features with the help of orthogonal transformation. These new transformed features are called the **Principal Components**. It is one of the popular tools that is used for exploratory data analysis and predictive modeling.

PCA works by considering the variance of each attribute because the high attribute shows the good split between the classes, and hence it reduces the dimensionality. Some real-world applications of PCA are **image processing, movie recommendation system, optimizing the power allocation in various communication channels**.

Backward Feature Elimination

The backward feature elimination technique is mainly used while developing Linear Regression or Logistic Regression model. Below steps are performed in this technique to reduce the dimensionality or in feature selection:

- In this technique, firstly, all the n variables of the given dataset are taken to train the model.
- The performance of the model is checked.
- Now we will remove one feature each time and train the model on n-1 features for n times, and will compute the performance of the model.
- We will check the variable that has made the smallest or no change in the performance of the model, and then we will drop that variable or features; after that, we will be left with n-1 features.
- Repeat the complete process until no feature can be dropped.

In this technique, by selecting the optimum performance of the model and maximum tolerable error rate, we can define the optimal number of features require for the machine learning algorithms.

Forward Feature Selection

Forward feature selection follows the inverse process of the backward elimination process. It means, in this technique, we don't eliminate the feature; instead, we will find the best features that can produce the highest increase in the performance of the model. Below steps are performed in this technique:

- We start with a single feature only, and progressively we will add each feature at a time.
- Here we will train the model on each feature separately.

- The feature with the best performance is selected.
- The process will be repeated until we get a significant increase in the performance of the model.

Missing Value Ratio

If a dataset has too many missing values, then we drop those variables as they do not carry much useful information. To perform this, we can set a threshold level, and if a variable has missing values more than that threshold, we will drop that variable. The higher the threshold value, the more efficient the reduction.

Low Variance Filter

As same as missing value ratio technique, data columns with some changes in the data have less information. Therefore, we need to calculate the variance of each variable, and all data columns with variance lower than a given threshold are dropped because low variance features will not affect the target variable.

High Correlation Filter

High Correlation refers to the case when two variables carry approximately similar information. Due to this factor, the performance of the model can be degraded. This correlation between the independent numerical variable gives the calculated value of the correlation coefficient. If this value is higher than the threshold value, we can remove one of the variables from the dataset. We can consider those variables or features that show a high correlation with the target variable.

Random Forest

Random Forest is a popular and very useful feature selection algorithm in machine learning. This algorithm contains an in-built feature importance package, so we do not need to program it separately. In this technique, we need to generate a large set of trees against the target variable, and with the help of usage statistics of each attribute, we need to find the subset of features.

Random forest algorithm takes only numerical variables, so we need to convert the input data into numeric data using **hot encoding**.

Factor Analysis

Factor analysis is a technique in which each variable is kept within a group according to the correlation with other variables, it means variables within a group can have a high correlation between themselves, but they have a low correlation with variables of other groups.

We can understand it by an example, such as if we have two variables Income and spend. These two variables have a high correlation, which means people with high

income spends more, and vice versa. So, such variables are put into a group, and that group is known as the **factor**. The number of these factors will be reduced as compared to the original dimension of the dataset.

Auto-encoders

One of the popular methods of dimensionality reduction is auto-encoder, which is a type of ANN or [artificial neural network](#), and its main aim is to copy the inputs to their outputs. In this, the input is compressed into latent-space representation, and output is occurred using this representation. It has mainly two parts:

- **Encoder:** The function of the encoder is to compress the input to form the latent-space representation.
- **Decoder:** The function of the decoder is to recreate the output from the latent-space representation

Principal Component Analysis

Principal Component Analysis is an unsupervised learning algorithm that is used for the dimensionality reduction in [machine learning](#). It is a statistical process that converts the observations of correlated features into a set of linearly uncorrelated features with the help of orthogonal transformation. These new transformed features are called the **Principal Components**. It is one of the popular tools that is used for exploratory data analysis and predictive modeling. It is a technique to draw strong patterns from the given dataset by reducing the variances.

PCA generally tries to find the lower-dimensional surface to project the high-dimensional data.

PCA works by considering the variance of each attribute because the high attribute shows the good split between the classes, and hence it reduces the dimensionality. Some real-world applications of PCA are [image processing](#), [movie recommendation system](#), [optimizing the power allocation in various communication channels](#). It is a feature extraction technique, so it contains the important variables and drops the least important variable.

The PCA algorithm is based on some mathematical concepts such as:

- Variance and Covariance
- Eigenvalues and Eigen factors

Some common terms used in PCA algorithm:

- **Dimensionality:** It is the number of features or variables present in the given dataset. More easily, it is the number of columns present in the dataset.

- **Correlation:** It signifies that how strongly two variables are related to each other. Such as if one changes, the other variable also gets changed. The correlation value ranges from -1 to +1. Here, -1 occurs if variables are inversely proportional to each other, and +1 indicates that variables are directly proportional to each other.
- **Orthogonal:** It defines that variables are not correlated to each other, and hence the correlation between the pair of variables is zero.
- **Eigenvectors:** If there is a square matrix M , and a non-zero vector v is given. Then v will be eigenvector if Av is the scalar multiple of v .
- **Covariance Matrix:** A matrix containing the covariance between the pair of variables is called the Covariance Matrix.

Principal Components in PCA

As described above, the transformed new features or the output of PCA are the Principal Components. The number of these PCs are either equal to or less than the original features present in the dataset. Some properties of these principal components are given below:

- The principal component must be the linear combination of the original features.
- These components are orthogonal, i.e., the correlation between a pair of variables is zero.
- The importance of each component decreases when going to 1 to n , it means the 1 PC has the most importance, and n PC will have the least importance.

Steps for PCA algorithm

1. Getting the dataset

Firstly, we need to take the input dataset and divide it into two subparts X and Y , where X is the training set, and Y is the validation set.

2. Representing data into a structure

Now we will represent our dataset into a structure. Such as we will represent the two-dimensional matrix of independent variable X . Here each row corresponds to the data items, and the column corresponds to the Features. The number of columns is the dimensions of the dataset.

3. Standardizing the data

In this step, we will standardize our dataset. Such as in a particular column, the

features with high variance are more important compared to the features with lower variance.

If the importance of features is independent of the variance of the feature, then we will divide each data item in a column with the standard deviation of the column. Here we will name the matrix as Z .

4. **Calculating the Covariance of Z**

To calculate the covariance of Z , we will take the matrix Z , and will transpose it. After transpose, we will multiply it by Z . The output matrix will be the Covariance matrix of Z .

5. **Calculating the Eigen Values and Eigen Vectors**

Now we need to calculate the eigenvalues and eigenvectors for the resultant covariance matrix Z . Eigenvectors or the covariance matrix are the directions of the axes with high information. And the coefficients of these eigenvectors are defined as the eigenvalues.

6. **Sorting the Eigen Vectors**

In this step, we will take all the eigenvalues and will sort them in decreasing order, which means from largest to smallest. And simultaneously sort the eigenvectors accordingly in matrix P of eigenvalues. The resultant matrix will be named as P^* .

7. **Calculating the new features Or Principal Components**

Here we will calculate the new features. To do this, we will multiply the P^* matrix to the Z . In the resultant matrix Z^* , each observation is the linear combination of original features. Each column of the Z^* matrix is independent of each other.

8. **Remove less or unimportant features from the new dataset.**

The new feature set has occurred, so we will decide here what to keep and what to remove. It means, we will only keep the relevant or important features in the new dataset, and unimportant features will be removed out.

Applications of Principal Component Analysis

- PCA is mainly used as the dimensionality reduction technique in various AI applications such **as computer vision, image compression, etc.**
- It can also be used for finding hidden patterns if data has high dimensions. Some fields where PCA is used are Finance, data mining, Psychology, etc.

Cross-Validation in Machine Learning

Cross-validation is a technique for validating the model efficiency by training it on the subset of input data and testing on previously unseen subset of the input data. ***We can also say that it is a technique to check how a statistical model generalizes to an independent dataset.***

In [machine learning](#), there is always the need to test the stability of the model. It means based only on the training dataset; we can't fit our model on the training dataset. For this purpose, we reserve a particular sample of the dataset, which was not part of the training dataset. After that, we test our model on that sample before deployment, and this complete process comes under cross-validation. This is something different from the general train-test split.

Hence the basic steps of cross-validations are:

- Reserve a subset of the dataset as a validation set.
- Provide the training to the model using the training dataset.
- Now, evaluate model performance using the validation set. If the model performs well with the validation set, perform the further step, else check for the issues.

Methods used for Cross-Validation

There are some common methods that are used for cross-validation. These methods are given below:

1. **Validation Set Approach**
2. **Leave-P-out cross-validation**
3. **Leave one out cross-validation**
4. **K-fold cross-validation**
5. **Stratified k-fold cross-validation**

Validation Set Approach

We divide our input dataset into a training set and test or validation set in the validation set approach. Both the subsets are given 50% of the dataset.

But it has one of the big disadvantages that we are just using a 50% dataset to train our model, so the model may miss out to capture important information of the dataset. It also tends to give the underfitted model.

Leave-P-out cross-validation

In this approach, the p datasets are left out of the training data. It means, if there are total n datapoints in the original input dataset, then $n-p$ data points will be used as the training dataset and the p data points as the validation set. This complete process is repeated for all the samples, and the average error is calculated to know the effectiveness of the model.

There is a disadvantage of this technique; that is, it can be computationally difficult for the large p .

Leave one out cross-validation

This method is similar to the leave- p -out cross-validation, but instead of p , we need to take 1 dataset out of training. It means, in this approach, for each learning set, only one datapoint is reserved, and the remaining dataset is used to train the model. This process repeats for each datapoint. Hence for n samples, we get n different training set and n test set. It has the following features:

- In this approach, the bias is minimum as all the data points are used.
- The process is executed for n times; hence execution time is high.
- This approach leads to high variation in testing the effectiveness of the model as we iteratively check against one data point.

K-Fold Cross-Validation

K-fold cross-validation approach divides the input dataset into K groups of samples of equal sizes. These samples are called **folds**. For each learning set, the prediction function uses $k-1$ folds, and the rest of the folds are used for the test set. This approach is a very popular CV approach because it is easy to understand, and the output is less biased than other methods.

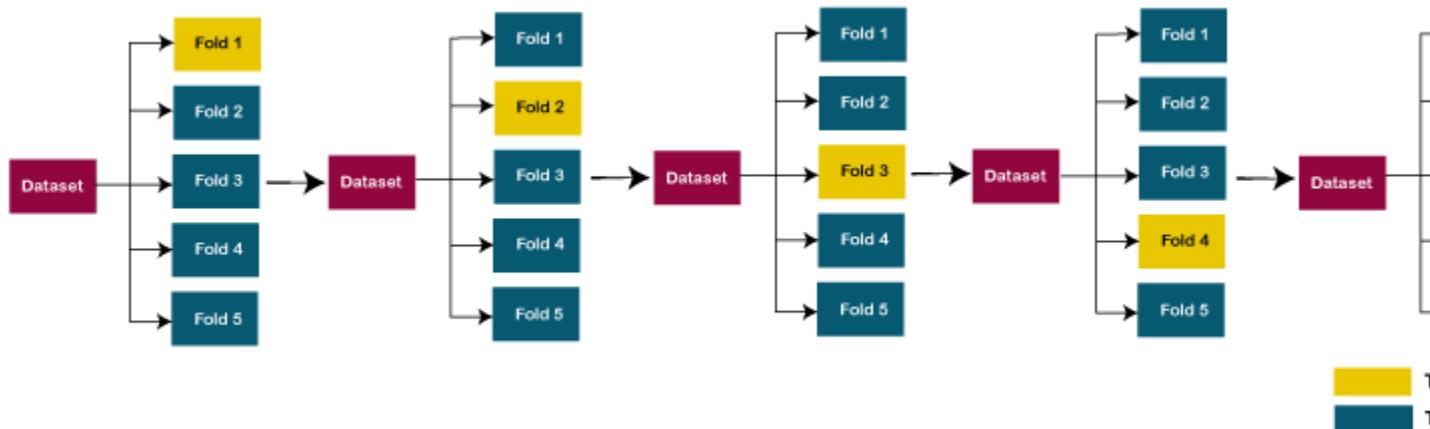
The steps for k-fold cross-validation are:

- Split the input dataset into K groups
- For each group:
 - Take one group as the reserve or test data set.
 - Use remaining groups as the training dataset
 - Fit the model on the training set and evaluate the performance of the model using the test set.

Let's take an example of 5-folds cross-validation. So, the dataset is grouped into 5 folds. On 1st iteration, the first fold is reserved for test the model, and rest are used to train the model. On 2nd iteration, the second fold is used to test the model, and rest are

used to train the model. This process will continue until each fold is not used for the test fold.

Consider the below diagram:



Stratified k-fold cross-validation

This technique is similar to k-fold cross-validation with some little changes. This approach works on stratification concept, it is a process of rearranging the data to ensure that each fold or group is a good representative of the complete dataset. To deal with the bias and variance, it is one of the best approaches.

It can be understood with an example of housing prices, such that the price of some houses can be much high than other houses. To tackle such situations, a stratified k-fold cross-validation technique is useful.

Holdout Method

This method is the simplest cross-validation technique among all. In this method, we need to remove a subset of the training data and use it to get prediction results by training it on the rest part of the dataset.

The error that occurs in this process tells how well our model will perform with the unknown dataset. Although this approach is simple to perform, it still faces the issue of high variance, and it also produces misleading results sometimes.

Comparison of Cross-validation to train/test split in Machine Learning

- **Train/test split:** The input data is divided into two parts, that are training set and test set on a ratio of 70:30, 80:20, etc. It provides a high variance, which is one of the biggest disadvantages.

- **Training Data:** The training data is used to train the model, and the dependent variable is known.
- **Test Data:** The test data is used to make the predictions from the model that is already trained on the training data. This has the same features as training data but not the part of that.
- **Cross-Validation dataset:** It is used to overcome the disadvantage of train/test split by splitting the dataset into groups of train/test splits, and averaging the result. It can be used if we want to optimize our model that has been trained on the training dataset for the best performance. It is more efficient as compared to train/test split as every observation is used for the training and testing both.

Limitations of Cross-Validation

There are some limitations of the cross-validation technique, which are given below:

- For the ideal conditions, it provides the optimum output. But for the inconsistent data, it may produce a drastic result. So, it is one of the big disadvantages of cross-validation, as there is no certainty of the type of data in machine learning.
- In predictive modeling, the data evolves over a period, due to which, it may face the differences between the training set and validation sets. Such as if we create a model for the prediction of stock market values, and the data is trained on the previous 5 years stock values, but the realistic future values for the next 5 years may drastically different, so it is difficult to expect the correct output for such situations.

Applications of Cross-Validation

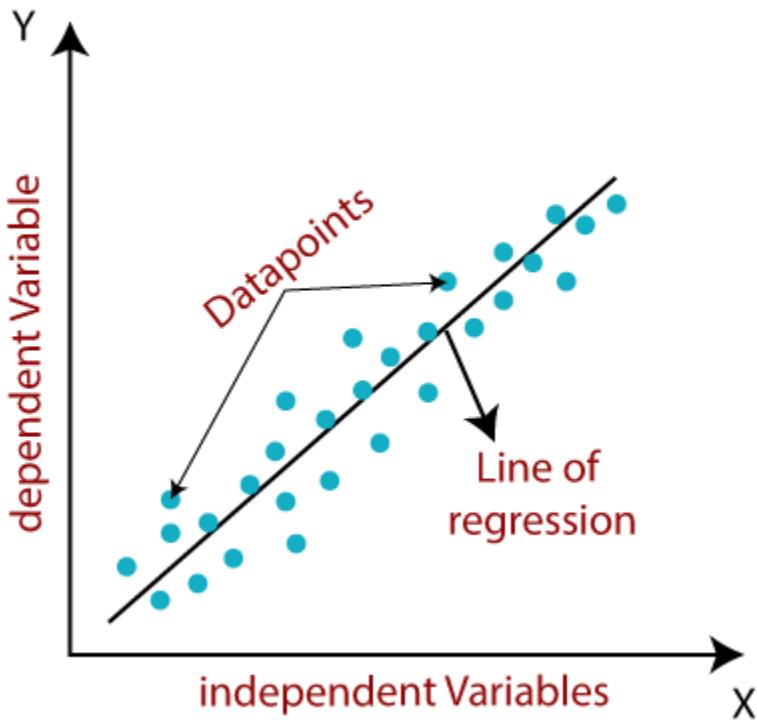
- This technique can be used to compare the performance of different predictive modeling methods.
- It has great scope in the medical research field.
- It can also be used for the meta-analysis, as it is already being used by the data scientists in the field of medical statistics.

Linear Regression in Machine Learning

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:



Mathematically, we can represent a linear regression as:

$$y = a_0 + a_1 x + \varepsilon$$

Here,

y = Dependent Variable (Target Variable)

X = Independent Variable (predictor Variable)

a_0 = intercept of the line (Gives an additional degree of freedom)

a_1 = 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:**

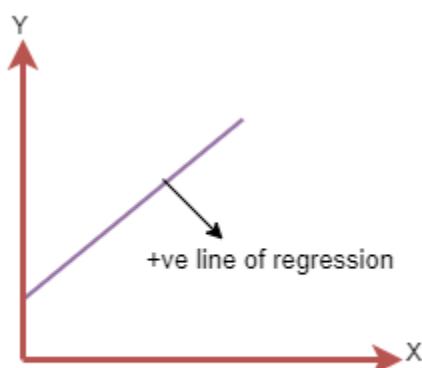
If more than one independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Multiple Linear Regression.

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.

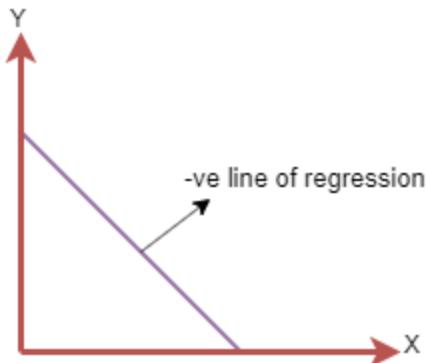


The line equation will be: $Y = a_0 + a_1 X$

- **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: $Y = -a_0 + a_1 X$

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:

For the above linear equation, MSE can be calculated as:

$$MSE = \frac{1}{N} \sum_{i=1}^n (y_i - (a_1 x_i + a_0))^2$$

Where,

N=Total number of observation

y_i = Actual value

$(a_1 x_i + a_0)$ = 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 be high. If the scatter points are close to the regression line, then the residual will be small and hence the cost function.

Gradient Descent:

- Gradient descent is used to minimize the MSE by calculating the gradient of the cost function.
- A regression model uses gradient descent to update the coefficients of the line by reducing the cost function.
- It is done by a random selection of values of coefficient and then iteratively update the values to reach the minimum cost function.

Model Performance:

The Goodness of fit determines how the line of regression fits the set of observations. The process of finding the best model out of various models is called **optimization**. It can be achieved by below method:

1. R-squared method:

- R-squared is a statistical method that determines the goodness of fit.
- It measures the strength of the relationship between the dependent and independent variables on a scale of 0-100%.
- The high value of R-square determines the less difference between the predicted values and actual values and hence represents a good model.
- It is also called a **coefficient of determination**, or **coefficient of multiple determination** for multiple regression.
- It can be calculated from the below formula:

$$R\text{-squared} = \frac{\text{Explained variation}}{\text{Total Variation}}$$

Assumptions of Linear Regression

Below are some important assumptions of Linear Regression. These are some formal checks while building a Linear Regression model, which ensures to get the best possible result from the given dataset.

- **Linear relationship between the features and target:**

Linear regression assumes the linear relationship between the dependent and independent variables.

- **Small or no multicollinearity between the features:**

Multicollinearity means high-correlation between the independent variables. Due to multicollinearity, it may be difficult to find the true relationship between the predictors and target variables. Or we can say, it is difficult to determine which predictor variable is affecting the target variable and which is not. So, the model assumes either little or no multicollinearity between the features or independent variables.

- **Homoscedasticity Assumption:**

Homoscedasticity is a situation when the error term is the same for all the values of independent variables. With homoscedasticity, there should be no clear pattern distribution of data in the scatter plot.

- **Normal distribution of error terms:**

Linear regression assumes that the error term should follow the normal distribution pattern. If error terms are not normally distributed, then confidence intervals will become either too wide or too narrow, which may cause difficulties in finding coefficients.

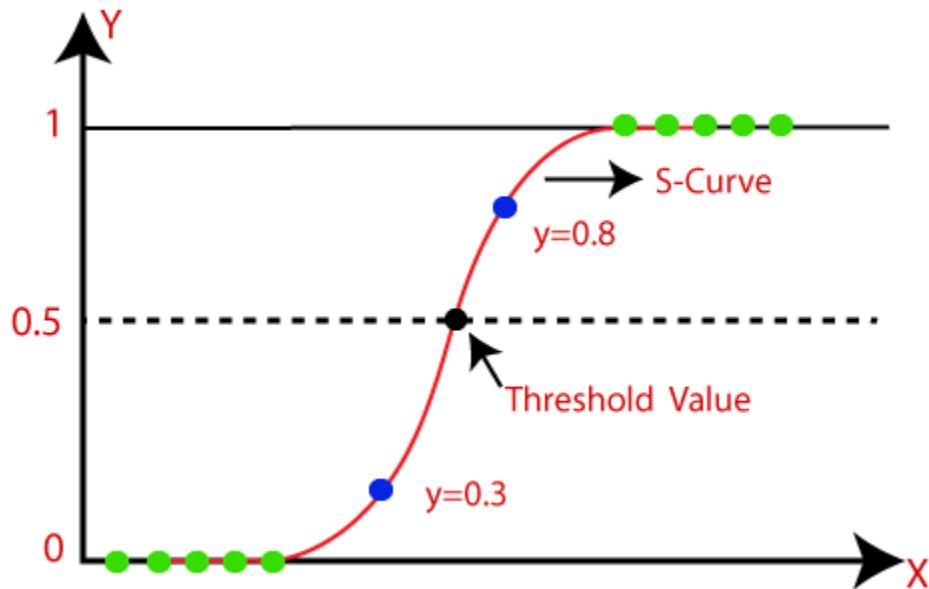
It can be checked using the **q-q plot**. If the plot shows a straight line without any deviation, which means the error is normally distributed.

- **No autocorrelations:**

The linear regression model assumes no autocorrelation in error terms. If there will be any correlation in the error term, then it will drastically reduce the accuracy of the model. Autocorrelation usually occurs if there is a dependency between residual errors.

Logistic Regression in Machine Learning

- Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.
- Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, **it gives the probabilistic values which lie between 0 and 1.**
- Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas **Logistic regression is used for solving the classification problems.**
- In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).
- The curve from the logistic function indicates the likelihood of something such as whether the cells are cancerous or not, a mouse is obese or not based on its weight, etc.
- Logistic Regression is a significant machine learning algorithm because it has the ability to provide probabilities and classify new data using continuous and discrete datasets.
- Logistic Regression can be used to classify the observations using different types of data and can easily determine the most effective variables used for the classification. The below image is showing the logistic function:



Note: Logistic regression uses the concept of predictive modeling as regression; therefore, it is called logistic regression, but is used to classify samples; Therefore, it falls under the classification algorithm.

Logistic Function (Sigmoid Function):

- The sigmoid function is a mathematical function used to map the predicted values to probabilities.
- It maps any real value into another value within a range of 0 and 1.
- The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the "S" form. The S-form curve is called the Sigmoid function or the logistic function.
- In logistic regression, we use the concept of the threshold value, which defines the probability of either 0 or 1. Such as values above the threshold value tends to 1, and a value below the threshold values tends to 0.

Assumptions for Logistic Regression:

- The dependent variable must be categorical in nature.
- The independent variable should not have multi-collinearity.

Logistic Regression Equation:

The Logistic regression equation can be obtained from the Linear Regression equation. The mathematical steps to get Logistic Regression equations are given below:

- We know the equation of the straight line can be written as:

$$y = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \dots + b_nx_n$$

- In Logistic Regression y can be between 0 and 1 only, so for this let's divide the above equation by $(1-y)$:

$$\frac{y}{1-y}; 0 \text{ for } y=0, \text{ and infinity for } y=1$$

- But we need range between $-\infty$ to $+\infty$, then take logarithm of the equation it will become:

$$\log \left[\frac{y}{1-y} \right] = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \dots + b_nx_n$$

The above equation is the final equation for Logistic Regression.

Type of Logistic Regression:

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

- **Binomial:** In binomial Logistic regression, there can be only two possible types of the dependent variables, such as 0 or 1, Pass or Fail, etc.
- **Multinomial:** In multinomial Logistic regression, there can be 3 or more possible unordered types of the dependent variable, such as "cat", "dogs", or "sheep"
- **Ordinal:** In ordinal Logistic regression, there can be 3 or more possible ordered types of dependent variables, such as "low", "Medium", or "High".

Python Implementation of Logistic Regression (Binomial)

To understand the implementation of Logistic Regression in Python, we will use the below example:

Example: There is a dataset given which contains the information of various users obtained from the social networking sites. There is a car making company that has recently launched a new SUV car. So the company wanted to check how many users from the dataset, wants to purchase the car.

For this problem, we will build a Machine Learning model using the Logistic regression algorithm. The dataset is shown in the below image. In this problem, we will predict the **purchased variable (Dependent Variable)** by using **age and salary (Independent variables)**.

User ID	Gender	Age	EstimatedSalary	Purchased
15624510	Male	19	19000	0
15810944	Male	35	20000	0
15668575	Female	26	43000	0
15603246	Female	27	57000	0
15804002	Male	19	76000	0
15728773	Male	27	58000	0
15598044	Female	27	84000	0
15694829	Female	32	150000	1
15600575	Male	25	33000	0
15727311	Female	35	65000	0
15570769	Female	26	80000	0
15606274	Female	26	52000	0
15746139	Male	20	86000	0
15704987	Male	32	18000	0
15628972	Male	18	82000	0
15697686	Male	29	80000	0
15733883	Male	47	25000	1
15617482	Male	45	26000	1
15704583	Male	46	28000	1
15621083	Female	48	29000	1
15649487	Male	45	22000	1
15736760	Female	47	49000	1

Steps in Logistic Regression: To implement the Logistic Regression using Python, we will use the same steps as we have done in previous topics of Regression. Below are the steps:

- Data Pre-processing step
- Fitting Logistic Regression to the Training set
- Predicting the test result
- Test accuracy of the result(Creation of Confusion matrix)
- Visualizing the test set result.

1. Data Pre-processing step: In this step, we will pre-process/prepare the data so that we can use it in our code efficiently. It will be the same as we have done in Data pre-processing topic. The code for this is given below:

1. `#Data Pre-procesing Step`
2. `# importing libraries`
3. `import numpy as nm`
4. `import matplotlib.pyplot as mtp`

```
5. import pandas as pd
6.
7. #importing datasets
8. data_set= pd.read_csv('user_data.csv')
```

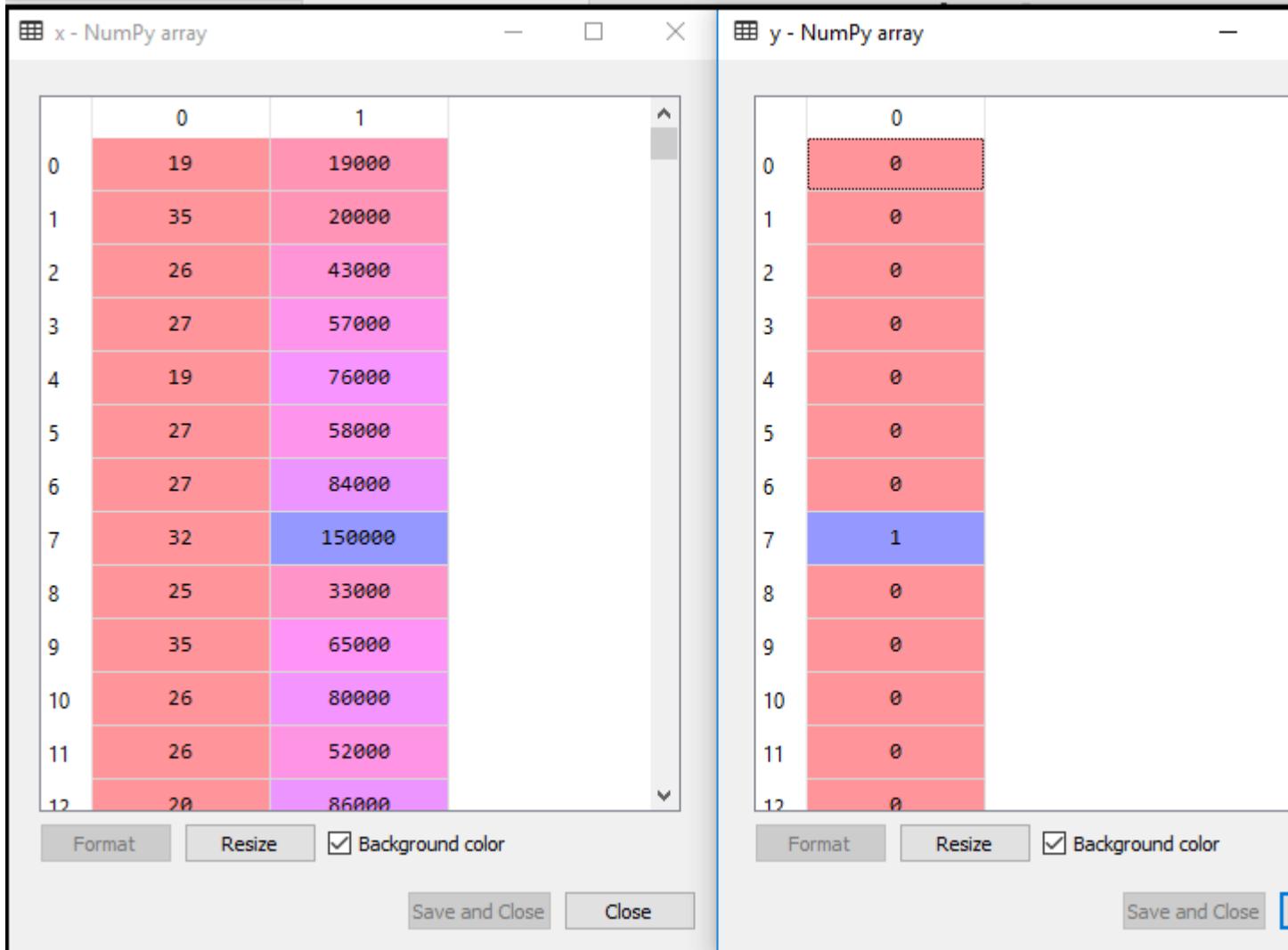
By executing the above lines of code, we will get the dataset as the output. Consider the given image:

Index	User ID	Gender	Age	EstimatedSalary	Purchased
92	15809823	Male	26	15000	0
150	15679651	Female	26	15000	0
43	15792008	Male	30	15000	0
155	15610140	Female	31	15000	0
32	15573452	Female	21	16000	0
180	15685576	Male	26	16000	0
79	15655123	Female	26	17000	0
40	15764419	Female	27	17000	0
128	15722758	Male	30	17000	0
58	15642885	Male	22	18000	0
29	15669656	Male	31	18000	0
13	15704987	Male	32	18000	0
74	15592877	Male	32	18000	0
0	15624510	Male	19	19000	0

Now, we will extract the dependent and independent variables from the given dataset. Below is the code for it:

```
1. #Extracting Independent and dependent Variable
2. x= data_set.iloc[:, [2,3]].values
3. y= data_set.iloc[:, 4].values
```

In the above code, we have taken [2, 3] for x because our independent variables are age and salary, which are at index 2, 3. And we have taken 4 for y variable because our dependent variable is at index 4. The output will be:



The image shows two side-by-side data visualization windows. The left window is titled 'x - NumPy array' and displays a 13x2 table. The first column is labeled '0' and the second column is labeled '1'. The data consists of 13 rows, each with two numerical values. The right window is titled 'y - NumPy array' and displays a 13x1 table. The data consists of 13 rows, each with a single numerical value. Both windows have a 'Format' and 'Resize' button at the bottom left, and a 'Save and Close' button at the bottom right. A 'Background color' checkbox is checked in both windows.

	0	1
0	19	19000
1	35	20000
2	26	43000
3	27	57000
4	19	76000
5	27	58000
6	27	84000
7	32	150000
8	25	33000
9	35	65000
10	26	80000
11	26	52000
12	20	86000

	0
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0
9	0
10	0
11	0
12	0

Now we will split the dataset into a training set and test set. Below is the code for it:

1.

```
# Splitting the dataset into training and test set.
```
2.

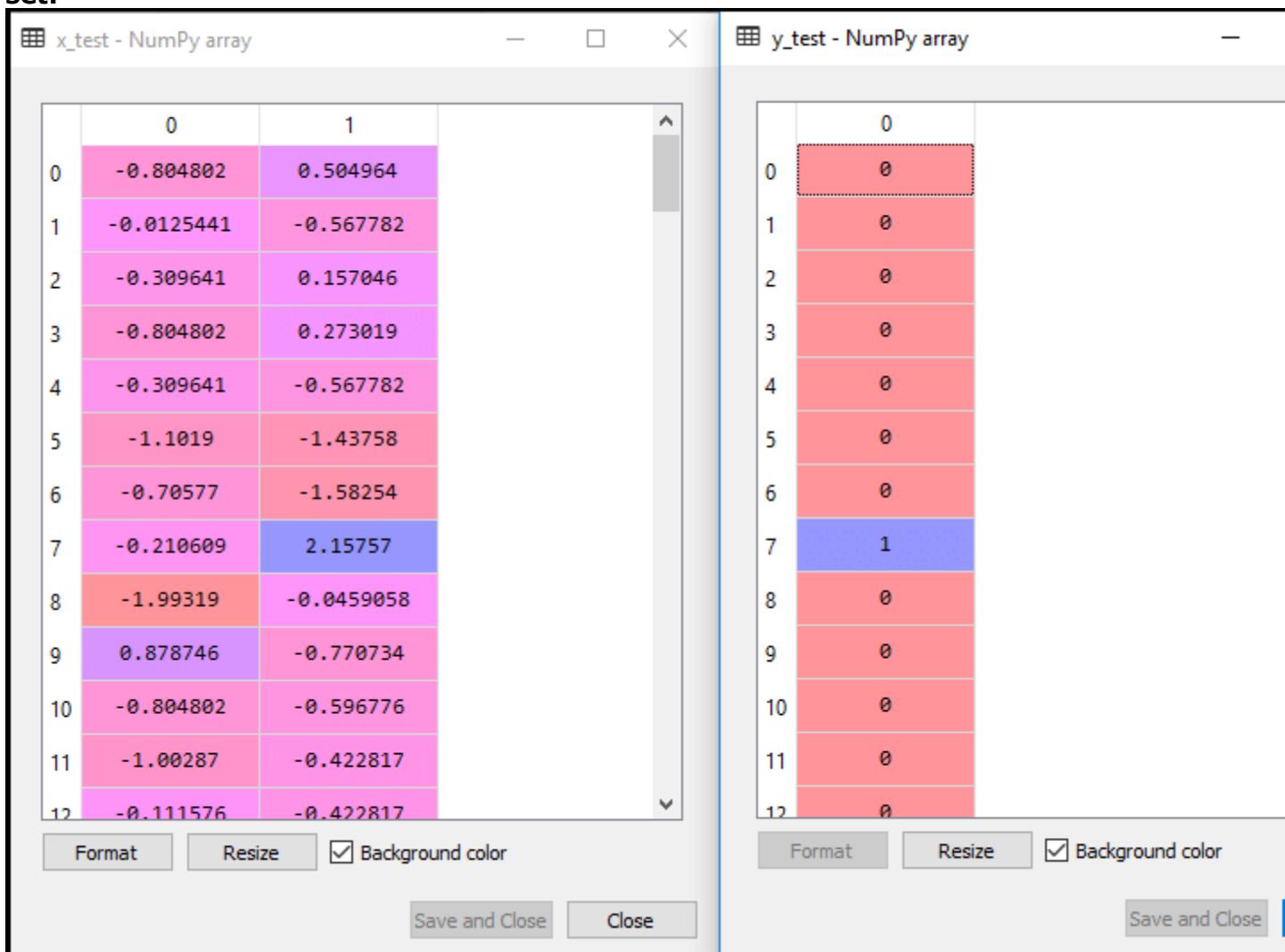
```
from sklearn.model_selection import train_test_split
```
3.

```
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)
```

The output for this is given below:

For test

set:



The image shows two side-by-side data visualization windows. The left window is titled 'x_test - NumPy array' and displays a 13x2 table of numerical values. The right window is titled 'y_test - NumPy array' and displays a 13x1 table of numerical values. Both windows have a 'Format' button, a 'Resize' button, and a checked 'Background color' checkbox. The 'Save and Close' button is highlighted in blue on the right window.

	0	1
0	-0.804802	0.504964
1	-0.0125441	-0.567782
2	-0.309641	0.157046
3	-0.804802	0.273019
4	-0.309641	-0.567782
5	-1.1019	-1.43758
6	-0.70577	-1.58254
7	-0.210609	2.15757
8	-1.99319	-0.0459058
9	0.878746	-0.770734
10	-0.804802	-0.596776
11	-1.00287	-0.422817
12	-0.111576	-0.422817

	0
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0
9	0
10	0
11	0
12	0

For training set:

x_test - NumPy array

	0	1
0	-0.804802	0.504964
1	-0.0125441	-0.567782
2	-0.309641	0.157046
3	-0.804802	0.273019
4	-0.309641	-0.567782
5	-1.1019	-1.43758
6	-0.70577	-1.58254
7	-0.210609	2.15757
8	-1.99319	-0.0459058
9	0.878746	-0.770734
10	-0.804802	-0.596776
11	-1.00287	-0.422817
12	-0.111576	-0.422817

Format Resize Background color

Save and Close Close

y_test - NumPy array

	0
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0
9	0
10	0
11	0
12	0

Format Resize Background color

Save and Close

In logistic regression, we will do feature scaling because we want accurate result of predictions. Here we will only scale the independent variable because dependent variable have only 0 and 1 values. Below is the code for it:

```

1.      #feature Scaling
2.      from sklearn.preprocessing import StandardScaler
3.      st_x= StandardScaler()
4.      x_train= st_x.fit_transform(x_train)
5.      x_test= st_x.transform(x_test)

```

The scaled output is given below:

x_test - NumPy array

	0	1
0	-0.804802	0.504964
1	-0.0125441	-0.567782
2	-0.309641	0.157046
3	-0.804802	0.273019
4	-0.309641	-0.567782
5	-1.1019	-1.43758
6	-0.70577	-1.58254
7	-0.210609	2.15757
8	-1.99319	-0.0459058
9	0.878746	-0.770734
10	-0.804802	-0.596776
11	-1.00287	-0.422817
12	-0.111576	-0.422817

Background color

x_train - NumPy array

	0	1
0	0.581649	-0.886707
1	-0.606738	1.46174
2	-0.0125441	-0.567782
3	-0.606738	1.89663
4	1.37391	-1.40858
5	1.47294	0.997847
6	0.0864882	-0.799728
7	-0.0125441	-0.248858
8	-0.210609	-0.567782
9	-0.210609	-0.190872
10	-0.309641	-1.29261
11	-0.309641	-0.567782
12	0.383585	0.0990599

Background color

2. Fitting Logistic Regression to the Training set:

We have well prepared our dataset, and now we will train the dataset using the training set. For providing training or fitting the model to the training set, we will import the **LogisticRegression** class of the **sklearn** library.

After importing the class, we will create a classifier object and use it to fit the model to the logistic regression. Below is the code for it:

1. #Fitting Logistic Regression to the training set
2. from sklearn.linear_model **import** LogisticRegression
3. classifier= LogisticRegression(random_state=0)
4. classifier.fit(x_train, y_train)

Output: By executing the above code, we will get the below output:

Out[5]:

```
1. LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
2.                     intercept_scaling=1, l1_ratio=None, max_iter=100,
3.                     multi_class='warn', n_jobs=None, penalty='l2',
4.                     random_state=0, solver='warn', tol=0.0001, verbose=0,
5.                     warm_start=False)
```

Hence our model is well fitted to the training set.

3. Predicting the Test Result

Our model is well trained on the training set, so we will now predict the result by using test set data. Below is the code for it:

```
1. #Predicting the test set result
2. y_pred= classifier.predict(x_test)
```

In the above code, we have created a `y_pred` vector to predict the test set result.

Output: By executing the above code, a new vector (`y_pred`) will be created under the variable explorer option. It can be seen as:

y_pred - NumPy array	
0	0
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0
9	1
10	0
11	0

The above output image shows the corresponding predicted users who want to purchase or not purchase the car.

4. Test Accuracy of the result

Now we will create the confusion matrix here to check the accuracy of the classification. To create it, we need to import the **confusion_matrix** function of the **sklearn** library. After importing the function, we will call it using a new variable **cm**. The function takes two parameters, mainly **y_true** (the actual values) and **y_pred** (the targeted value return by the classifier). Below is the code for it:

1. #Creating the Confusion matrix
2. from sklearn.metrics **import** confusion_matrix
3. cm= confusion_matrix()

Output:

By executing the above code, a new confusion matrix will be created. Consider the below image:



We can find the accuracy of the predicted result by interpreting the confusion matrix. By above output, we can interpret that $65+24= 89$ (Correct Output) and $8+3= 11$ (Incorrect Output).

5. Visualizing the training set result

Finally, we will visualize the training set result. To visualize the result, we will use **ListedColormap** class of matplotlib library. Below is the code for it:

```
1.      #Visualizing the training set result
2.      from matplotlib.colors import ListedColormap
3.      x_set, y_set = x_train, y_train
4.      x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() -
1, stop = x_set[:, 0].max() + 1, step = 0.01),
5.      nm.arange(start = x_set[:, 1].min() -
1, stop = x_set[:, 1].max() + 1, step = 0.01))
6.      mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
7.      alpha = 0.75, cmap = ListedColormap(('purple','green' )))
8.      mtp.xlim(x1.min(), x1.max())
9.      mtp.ylim(x2.min(), x2.max())
10.     for i, j in enumerate(nm.unique(y_set)):
```

```

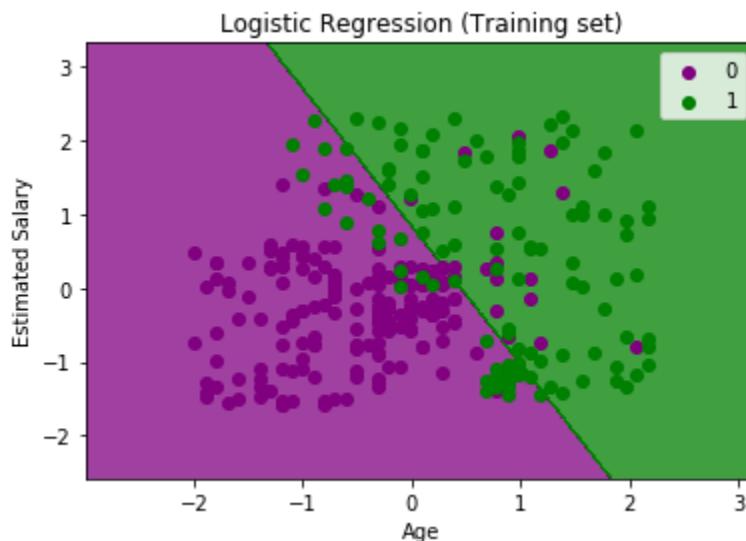
11.     mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
12.                 c = ListedColormap(['purple', 'green'])(i), label = j)
13.     mtp.title('Logistic Regression (Training set)')
14.     mtp.xlabel('Age')
15.     mtp.ylabel('Estimated Salary')
16.     mtp.legend()
17.     mtp.show()

```

In the above code, we have imported the **ListedColormap** class of Matplotlib library to create the colormap for visualizing the result. We have created two new variables **x_set** and **y_set** to replace **x_train** and **y_train**. After that, we have used the **nm.meshgrid** command to create a rectangular grid, which has a range of -1(minimum) to 1 (maximum). The pixel points we have taken are of 0.01 resolution.

To create a filled contour, we have used **mtp.contourf** command, it will create regions of provided colors (purple and green). In this function, we have passed the **classifier.predict** to show the predicted data points predicted by the classifier.

Output: By executing the above code, we will get the below output:



The graph can be explained in the below points:

- In the above graph, we can see that there are some **Green points** within the green region and **Purple points** within the purple region.
- All these data points are the observation points from the training set, which shows the result for purchased variables.

- This graph is made by using two independent variables i.e., **Age on the x-axis** and **Estimated salary on the y-axis**.
- The **purple point observations** are for which purchased (dependent variable) is probably 0, i.e., users who did not purchase the SUV car.
- The **green point observations** are for which purchased (dependent variable) is probably 1 means user who purchased the SUV car.
- We can also estimate from the graph that the users who are younger with low salary, did not purchase the car, whereas older users with high estimated salary purchased the car.
- But there are some purple points in the green region (Buying the car) and some green points in the purple region(Not buying the car). So we can say that younger users with a high estimated salary purchased the car, whereas an older user with a low estimated salary did not purchase the car.

The goal of the classifier:

We have successfully visualized the training set result for the logistic regression, and our goal for this classification is to divide the users who purchased the SUV car and who did not purchase the car. So from the output graph, we can clearly see the two regions (Purple and Green) with the observation points. The Purple region is for those users who didn't buy the car, and Green Region is for those users who purchased the car.

Linear Classifier:

As we can see from the graph, the classifier is a Straight line or linear in nature as we have used the Linear model for Logistic Regression. In further topics, we will learn for non-linear Classifiers.

Visualizing the test set result:

Our model is well trained using the training dataset. Now, we will visualize the result for new observations (Test set). The code for the test set will remain same as above except that here we will use **x_test and y_test** instead of **x_train and y_train**. Below is the code for it:

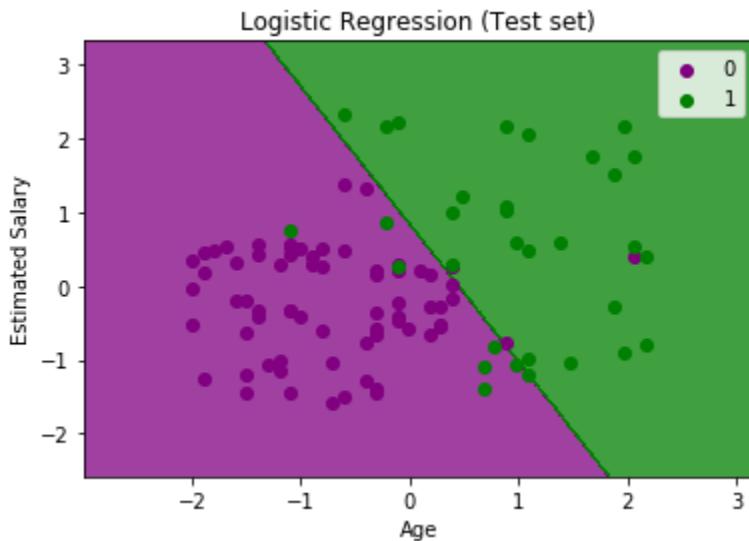
1. #Visulaizing the test set result
2. from matplotlib.colors **import** ListedColormap
3. x_set, y_set = x_test, y_test
4. x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step = 0.01),

```

5.         nm.arange(start = x_set[:, 1].min() -
6.             1, stop = x_set[:, 1].max() + 1, step = 0.01))
7.         mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
8.             alpha = 0.75, cmap = ListedColormap(('purple', 'green' )))
9.         mtp.xlim(x1.min(), x1.max())
10.        mtp.ylim(x2.min(), x2.max())
11.        for i, j in enumerate(nm.unique(y_set)):
12.            mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
13.                c = ListedColormap(('purple', 'green'))(i), label = j)
14.        mtp.title('Logistic Regression (Test set)')
15.        mtp.xlabel('Age')
16.        mtp.ylabel('Estimated Salary')
17.        mtp.legend()
18.        mtp.show()

```

Output:



The above graph shows the test set result. As we can see, the graph is divided into two regions (Purple and Green). And Green observations are in the green region, and Purple observations are in the purple region. So we can say it is a good prediction and model. Some of the green and purple data points are in different regions, which can be ignored as we have already calculated this error using the confusion matrix (11 Incorrect output).

Hence our model is pretty good and ready to make new predictions for this classification problem

K-Nearest Neighbor(KNN) Algorithm for Machine Learning

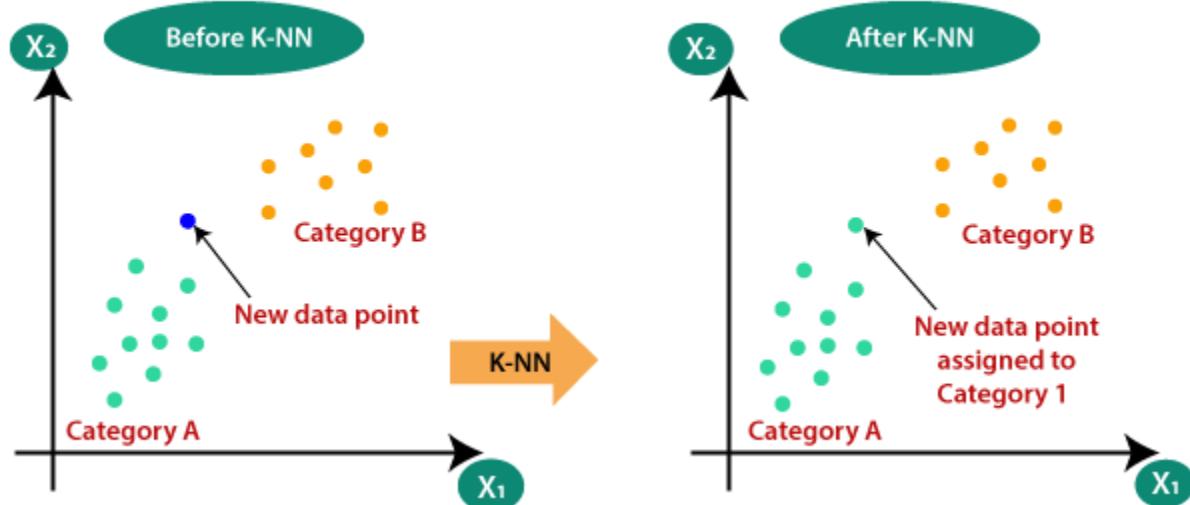
- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm 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 suited 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:



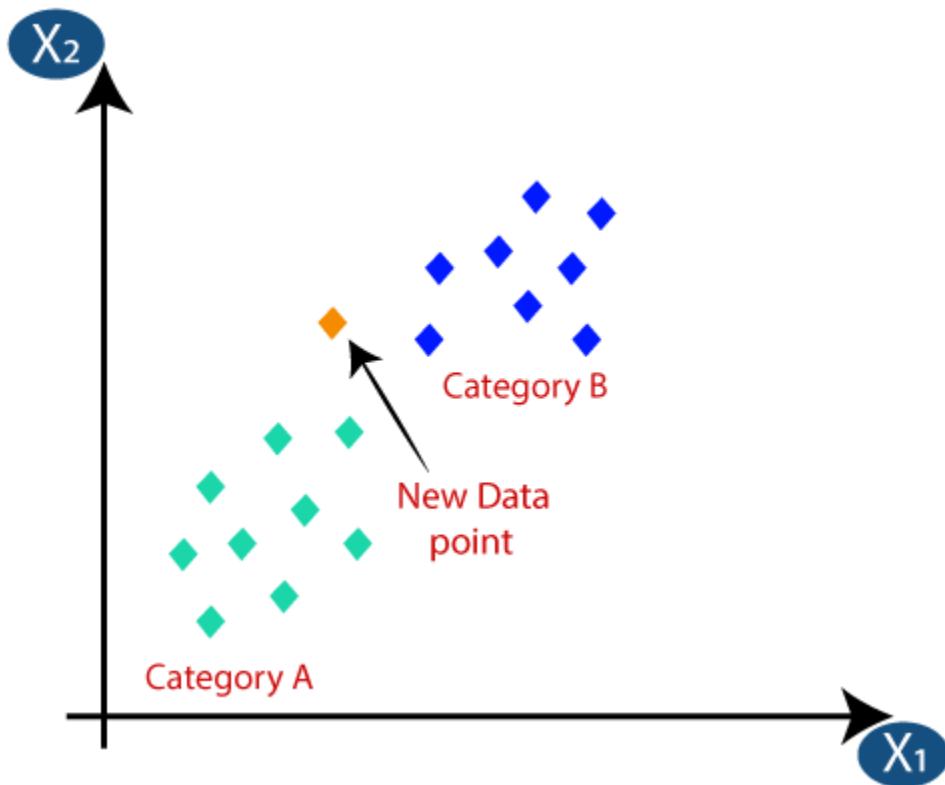
How does K-NN work?

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

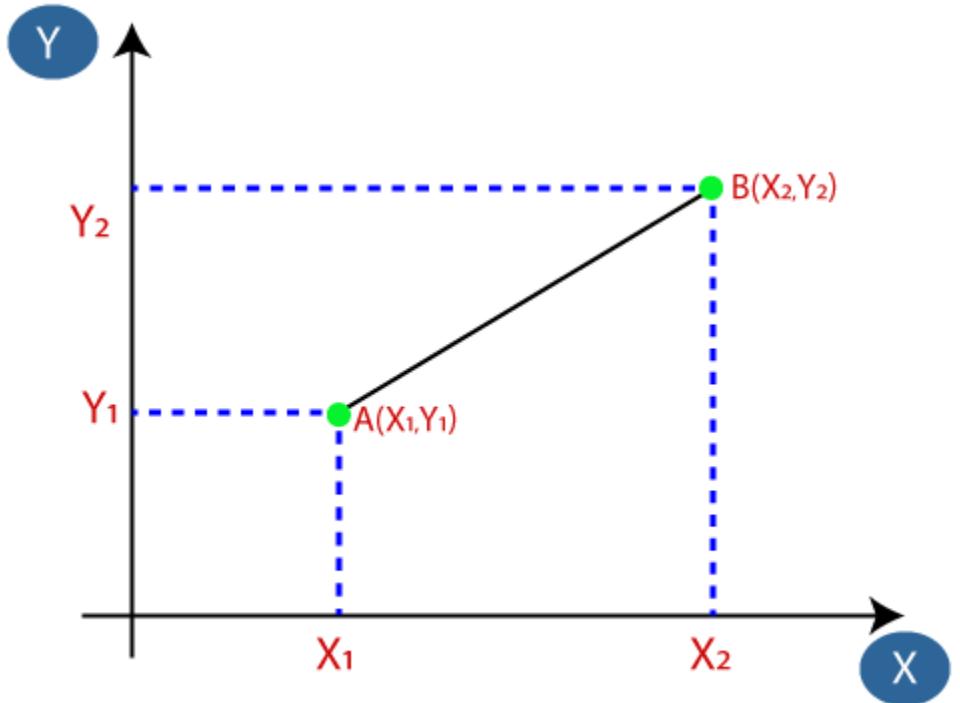
- **Step-1:** Select the number K of the neighbors
- **Step-2:** Calculate the Euclidean distance of **K number of neighbors**

- **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:

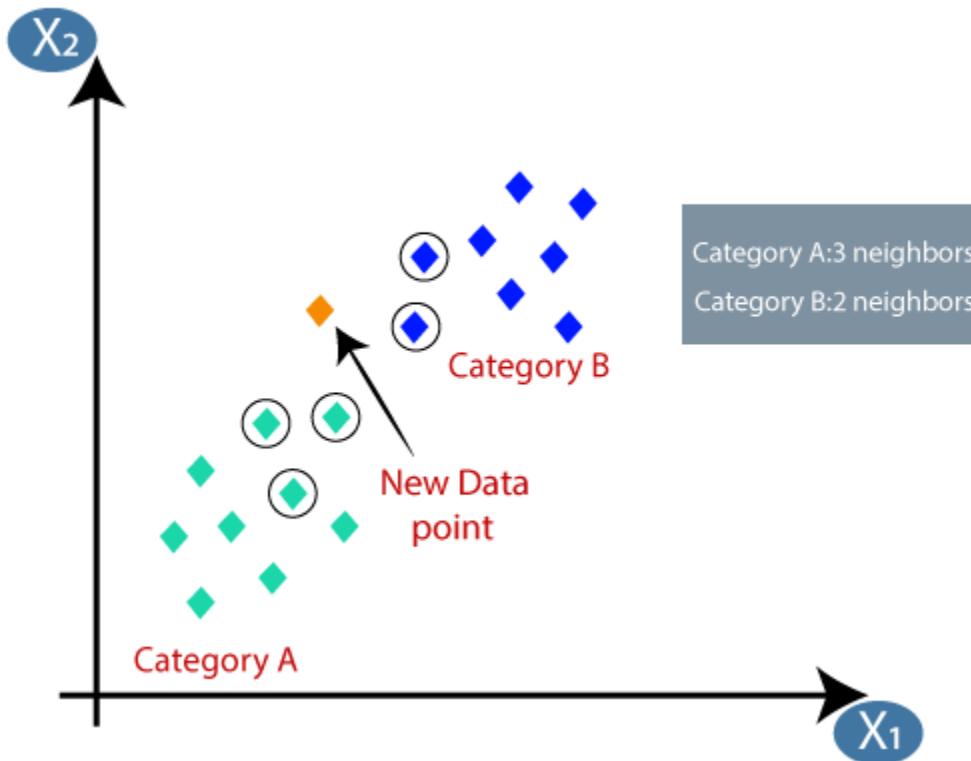


- Firstly, we will choose the number of neighbors, so we will choose the k=5.
- Next, we will calculate the **Euclidean distance** between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:



Euclidean Distance between A_1 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:



- As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A.

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:

- 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.

Python implementation of the KNN algorithm

To do the Python implementation of the K-NN algorithm, we will use the same problem and dataset which we have used in Logistic Regression. But here we will improve the performance of the model. Below is the problem description:

Problem for K-NN Algorithm: There is a Car manufacturer company that has manufactured a new SUV car. The company wants to give the ads to the users who are interested in buying that SUV. So for this problem, we have a dataset that contains multiple user's information through the social network. The dataset contains lots of information but the **Estimated Salary** and **Age** we will consider for the independent variable and the **Purchased variable** is for the dependent variable. Below is the dataset:

User ID	Gender	Age	EstimatedSalary	Purchased
15624510	Male	19	19000	0
15810944	Male	35	20000	0
15668575	Female	26	43000	0
15603246	Female	27	57000	0
15804002	Male	19	76000	0
15728773	Male	27	58000	0
15598044	Female	27	84000	0
15694829	Female	32	150000	1
15600575	Male	25	33000	0
15727311	Female	35	65000	0
15570769	Female	26	80000	0
15606274	Female	26	52000	0
15746139	Male	20	86000	0
15704987	Male	32	18000	0
15628972	Male	18	82000	0
15697686	Male	29	80000	0
15733883	Male	47	25000	1
15617482	Male	45	26000	1
15704583	Male	46	28000	1
15621083	Female	48	29000	1
15649487	Male	45	22000	1
15736760	Female	47	49000	1

Steps to implement the K-NN algorithm:

- Data Pre-processing step
- Fitting the K-NN algorithm to the Training set
- Predicting the test result
- Test accuracy of the result(Creation of Confusion matrix)
- Visualizing the test set result.

Data Pre-Processing Step:

The Data Pre-processing step will remain exactly the same as Logistic Regression. Below is the code for it:

```
1.      # importing libraries
2.      import numpy as nm
3.      import matplotlib.pyplot as mtp
4.      import pandas as pd
5.
6.      #importing datasets
7.      data_set= pd.read_csv('user_data.csv')
8.
9.      #Extracting Independent and dependent Variable
10.     x= data_set.iloc[:, [2,3]].values
11.     y= data_set.iloc[:, 4].values
12.
13.     # Splitting the dataset into training and test set.
14.     from sklearn.model_selection import train_test_split
15.     x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_s
tate=0)
16.
17.     #feature Scaling
18.     from sklearn.preprocessing import StandardScaler
19.     st_x= StandardScaler()
20.     x_train= st_x.fit_transform(x_train)
21.     x_test= st_x.transform(x_test)
```

By executing the above code, our dataset is imported to our program and well pre-processed. After feature scaling our test dataset will look like:

x_test - NumPy array		y_test - NumPy array	
0	1	0	0
-0.804802	0.504964	0	0
-0.0125441	-0.567782	1	0
-0.309641	0.157046	2	0
-0.804802	0.273019	3	0
-0.309641	-0.567782	4	0
-1.1019	-1.43758	5	0
-0.70577	-1.58254	6	0
-0.210609	2.15757	7	1
-1.99319	-0.0459058	8	0
0.878746	-0.770734	9	0
-0.804802	-0.596776	10	0
-1.00287	-0.422817	11	0
-0.111576	-0.422817	12	0

From the above output image, we can see that our data is successfully scaled.

- **Fitting K-NN classifier to the Training data:**

Now we will fit the K-NN classifier to the training data. To do this we will import the **KNeighborsClassifier** class of **Sklearn Neighbors** library. After importing the class, we will create the **Classifier** object of the class. The Parameter of this class will be

- **n_neighbors:** To define the required neighbors of the algorithm. Usually, it takes 5.
- **metric='minkowski':** This is the default parameter and it decides the distance between the points.
- **p=2:** It is equivalent to the standard Euclidean metric.

And then we will fit the classifier to the training data. Below is the code for it:

```
1.      #Fitting K-NN classifier to the training set
2.      from sklearn.neighbors import KNeighborsClassifier
3.      classifier= KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2 )
4.      classifier.fit(x_train, y_train)
```

Output: By executing the above code, we will get the output as:

```
Out[10]:
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

- **Predicting the Test Result:** To predict the test set result, we will create a **y_pred** vector as we did in Logistic Regression. Below is the code for it:

```
1.      #Predicting the test set result
2.      y_pred= classifier.predict(x_test)
```

Output:

The output for the above code will be:

y_pred - NumPy array	
0	0
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0
9	1
10	0
11	0
12	0

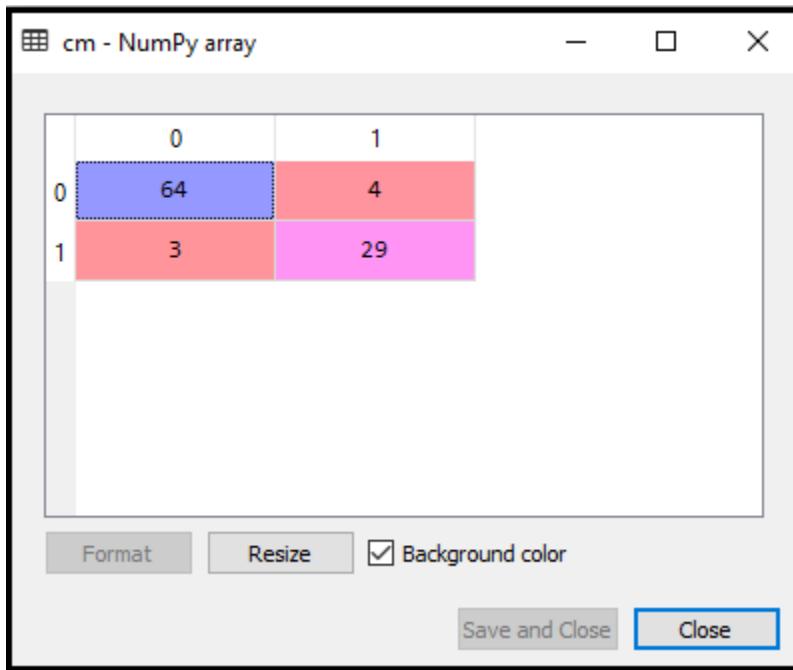
- o **Creating the Confusion Matrix:**

Now we will create the Confusion Matrix for our K-NN model to see the accuracy of the classifier. Below is the code for it:

1. #Creating the Confusion matrix
2. from sklearn.metrics **import** confusion_matrix
3. cm= confusion_matrix(y_test, y_pred)

In above code, we have imported the confusion_matrix function and called it using the variable cm.

Output: By executing the above code, we will get the matrix as below:



In the above image, we can see there are $64+29= 93$ correct predictions and $3+4= 7$ incorrect predictions, whereas, in Logistic Regression, there were 11 incorrect predictions. So we can say that the performance of the model is improved by using the K-NN algorithm.

- **Visualizing the Training set result:**

Now, we will visualize the training set result for K-NN model. The code will remain same as we did in Logistic Regression, except the name of the graph.

Below is the code for it:

1. #Visulaizing the trianing set result
2. from matplotlib.colors **import** ListedColormap
3. x_set, y_set = x_train, y_train
4. x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step = 0.01), nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
5. mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape), alpha = 0.75, cmap = ListedColormap(('red', 'green')))
6. mtp.xlim(x1.min(), x1.max())
7. mtp.ylim(x2.min(), x2.max())
8. **for** i, j in enumerate(nm.unique(y_set)):

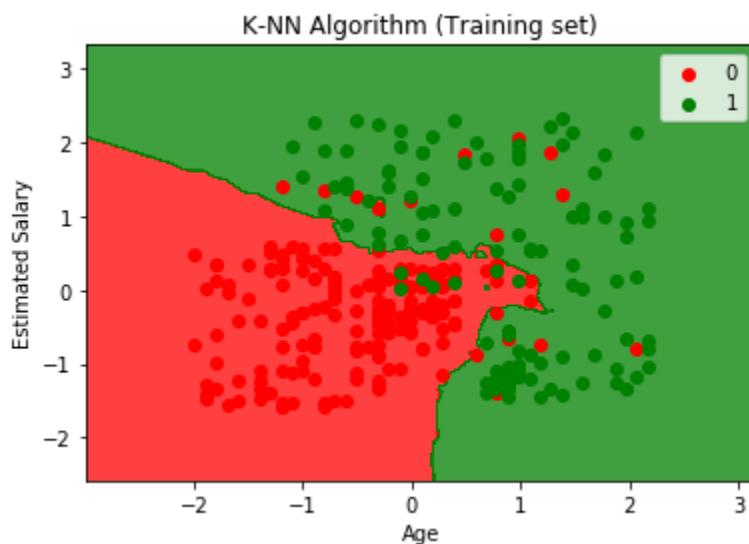
```

11.     mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
12.                   c = ListedColormap(['red', 'green'])(i), label = j)
13.     mtp.title('K-NN Algorithm (Training set)')
14.     mtp.xlabel('Age')
15.     mtp.ylabel('Estimated Salary')
16.     mtp.legend()
17.     mtp.show()

```

Output:

By executing the above code, we will get the below graph:



The output graph is different from the graph which we have occurred in Logistic Regression. It can be understood in the below points:

- As we can see the graph is showing the red point and green points. The green points are for Purchased(1) and Red Points for not Purchased(0) variable.
- The graph is showing an irregular boundary instead of showing any straight line or any curve because it is a K-NN algorithm, i.e., finding the nearest neighbor.
- The graph has classified users in the correct categories as most of the users who didn't buy the SUV are in the red region and users who bought the SUV are in the green region.

- The graph is showing good result but still, there are some green points in the red region and red points in the green region. But this is no big issue as by doing this model is prevented from overfitting issues.
- Hence our model is well trained.

- **Visualizing the Test set result:**

After the training of the model, we will now test the result by putting a new dataset, i.e., Test dataset. Code remains the same except some minor changes: such as **x_train and y_train** will be replaced by **x_test and y_test**.

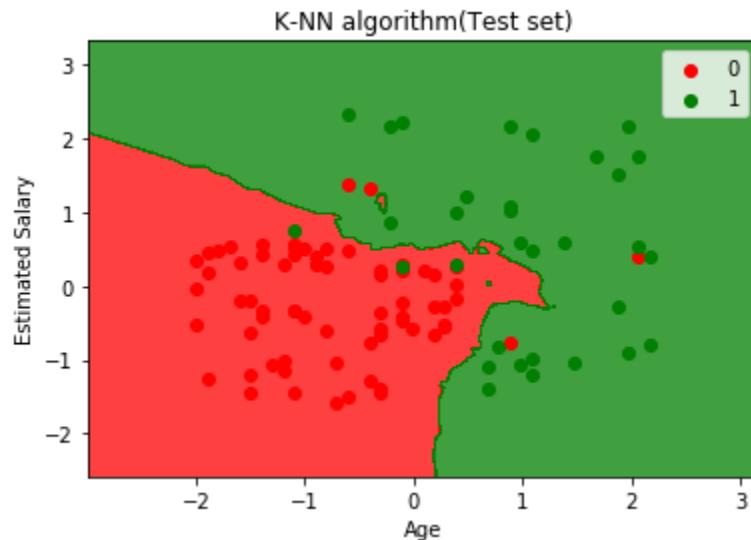
Below is the code for it:

```

1. #Visualizing the test set result
2. from matplotlib.colors import ListedColormap
3. x_set, y_set = x_test, y_test
4. x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() -
1, stop = x_set[:, 0].max() + 1, step = 0.01),
5. nm.arange(start = x_set[:, 1].min() -
1, stop = x_set[:, 1].max() + 1, step = 0.01))
6. mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
7. alpha = 0.75, cmap = ListedColormap(('red','green' )))
8. mtp.xlim(x1.min(), x1.max())
9. mtp.ylim(x2.min(), x2.max())
10. for i, j in enumerate(nm.unique(y_set)):
11.     mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
12.                 c = ListedColormap(('red', 'green'))(i), label = j)
13. mtp.title('K-NN algorithm(Test set)')
14. mtp.xlabel('Age')
15. mtp.ylabel('Estimated Salary')
16. mtp.legend()
17. mtp.show()

```

Output:



The above graph is showing the output for the test data set. As we can see in the graph, the predicted output is well good as most of the red points are in the red region and most of the green points are in the green region.

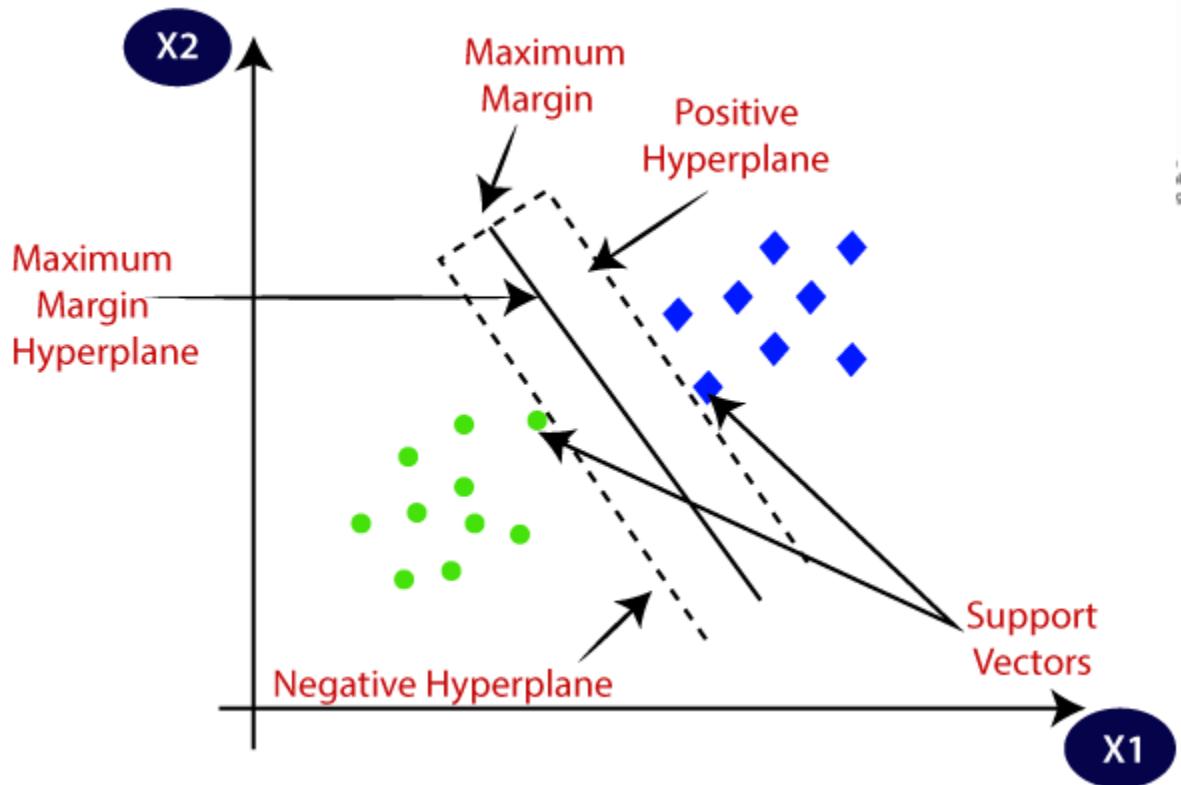
However, there are few green points in the red region and a few red points in the green region. So these are the incorrect observations that we have observed in the confusion matrix(7 Incorrect output).

Support Vector Machine Algorithm

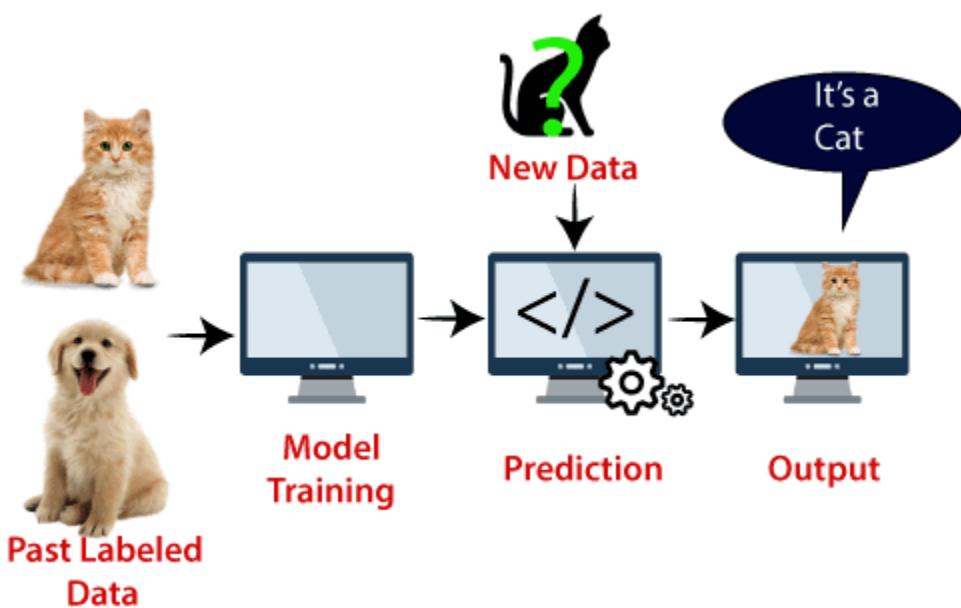
Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

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

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:



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



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

Types of SVM

SVM can be of two types:

- **Linear SVM:** Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.
- **Non-linear SVM:** Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

Hyperplane and Support Vectors in the SVM algorithm:

Hyperplane: There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the hyperplane of SVM.

The dimensions of the hyperplane depend on the features present in the dataset, which means if there are 2 features (as shown in image), then hyperplane will be a straight line. And if there are 3 features, then hyperplane will be a 2-dimension plane.

We always create a hyperplane that has a maximum margin, which means the maximum distance between the data points.

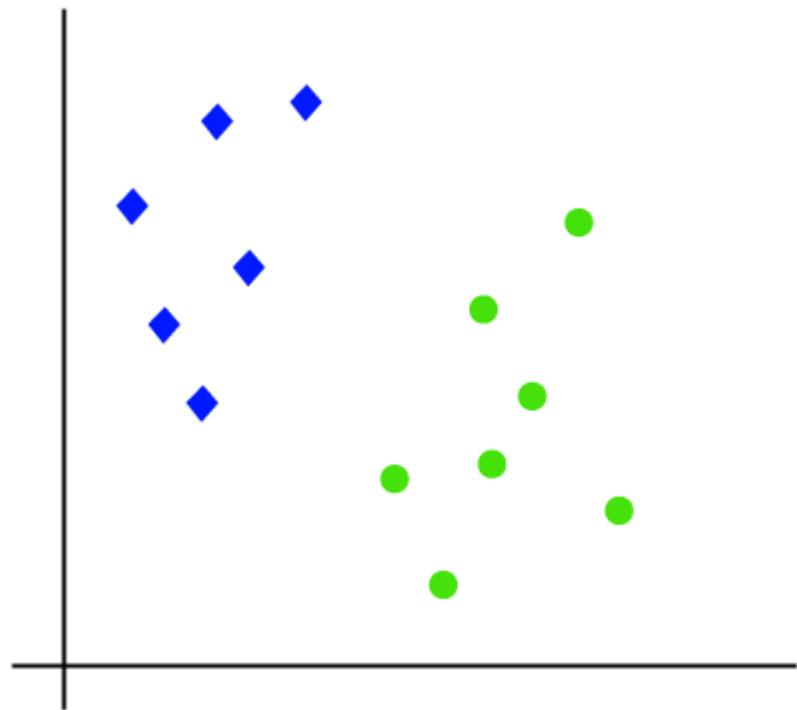
Support Vectors:

The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. Since these vectors support the hyperplane, hence called a Support vector.

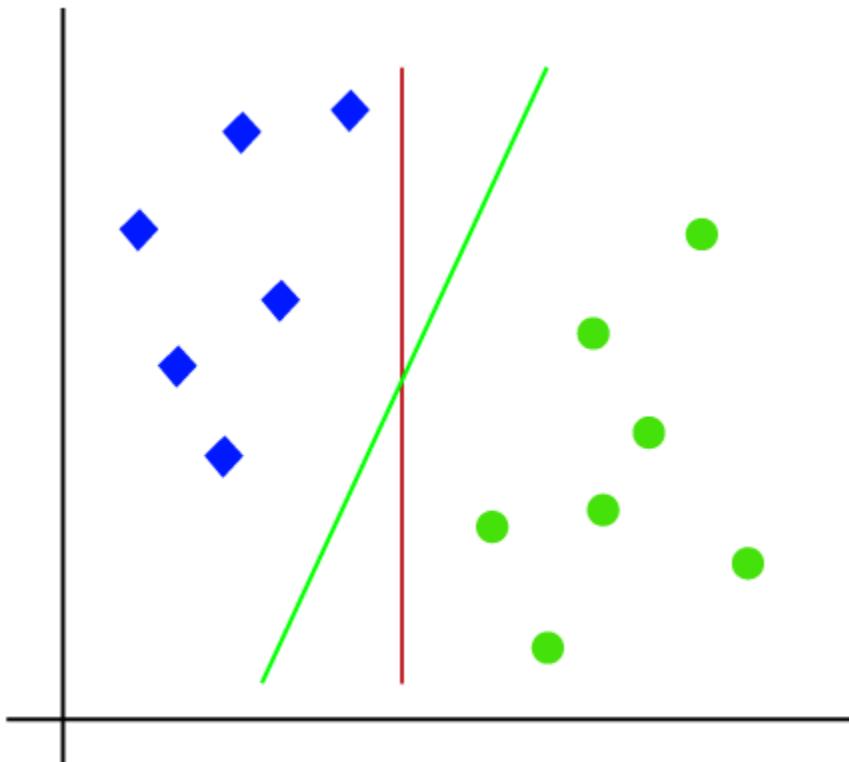
How does SVM works?

Linear SVM:

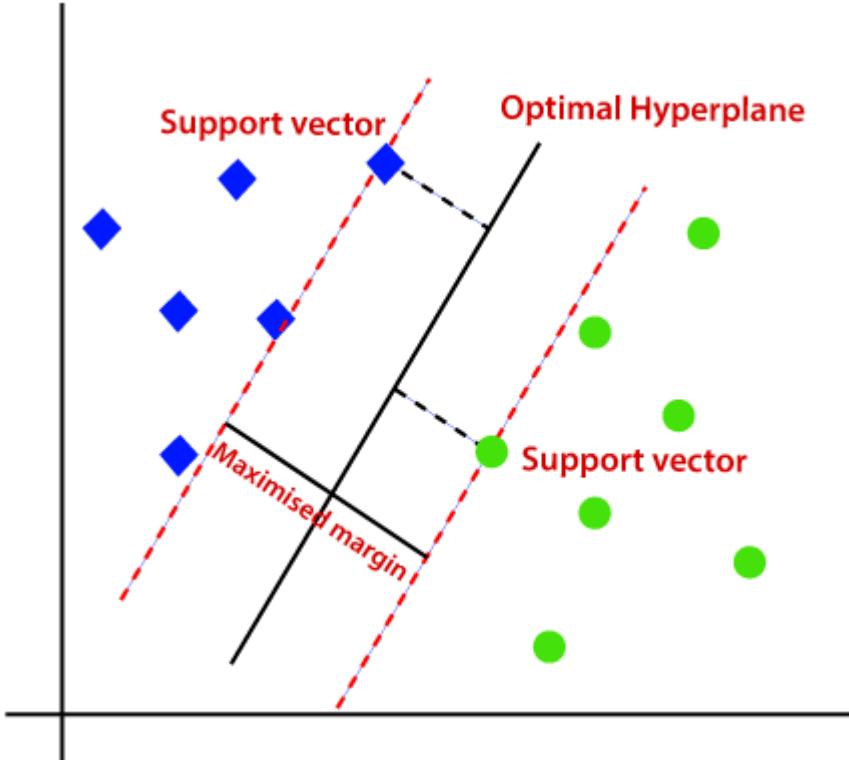
The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x_1 and x_2 . We want a classifier that can classify the pair (x_1, x_2) of coordinates in either green or blue. Consider the below image:



So as it is 2-d space so by just using a straight line, we can easily separate these two classes. But there can be multiple lines that can separate these classes. Consider the below image:

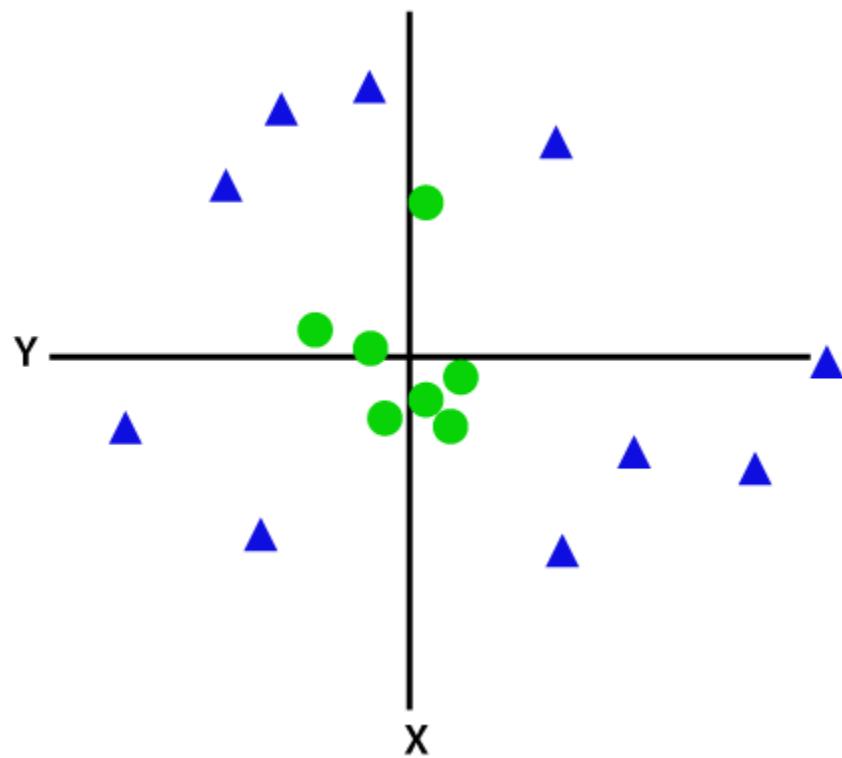


Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a **hyperplane**. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the hyperplane is called as **margin**. And the goal of SVM is to maximize this margin. The **hyperplane** with maximum margin is called the **optimal hyperplane**.



Non-Linear SVM:

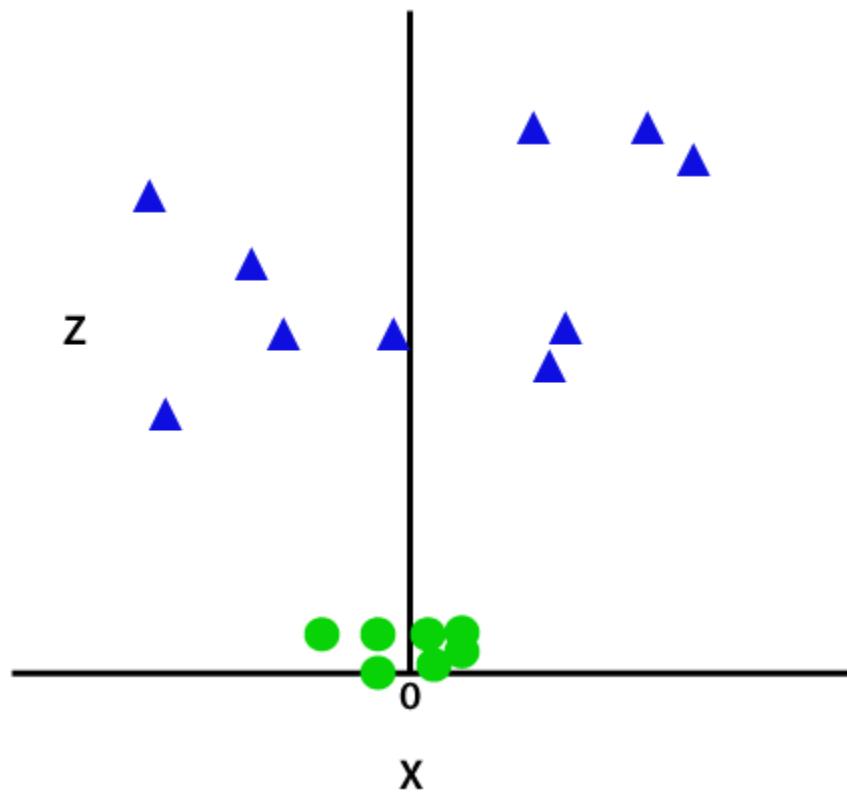
If data is linearly arranged, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line. Consider the below image:



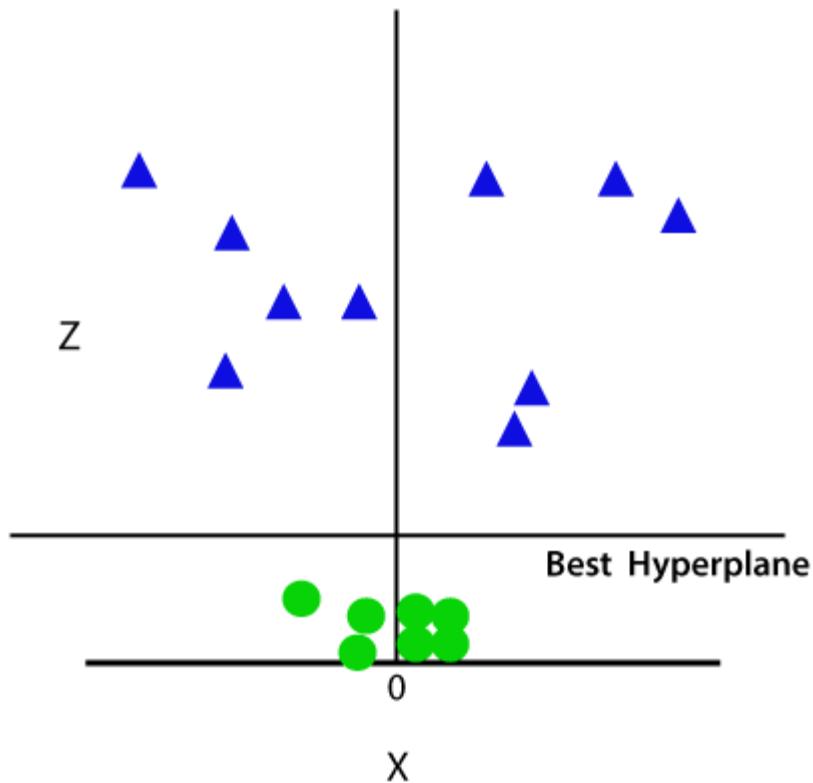
So to separate these data points, we need to add one more dimension. For linear data, we have used two dimensions x and y, so for non-linear data, we will add a third dimension z. It can be calculated as:

$$z = x^2 + y^2$$

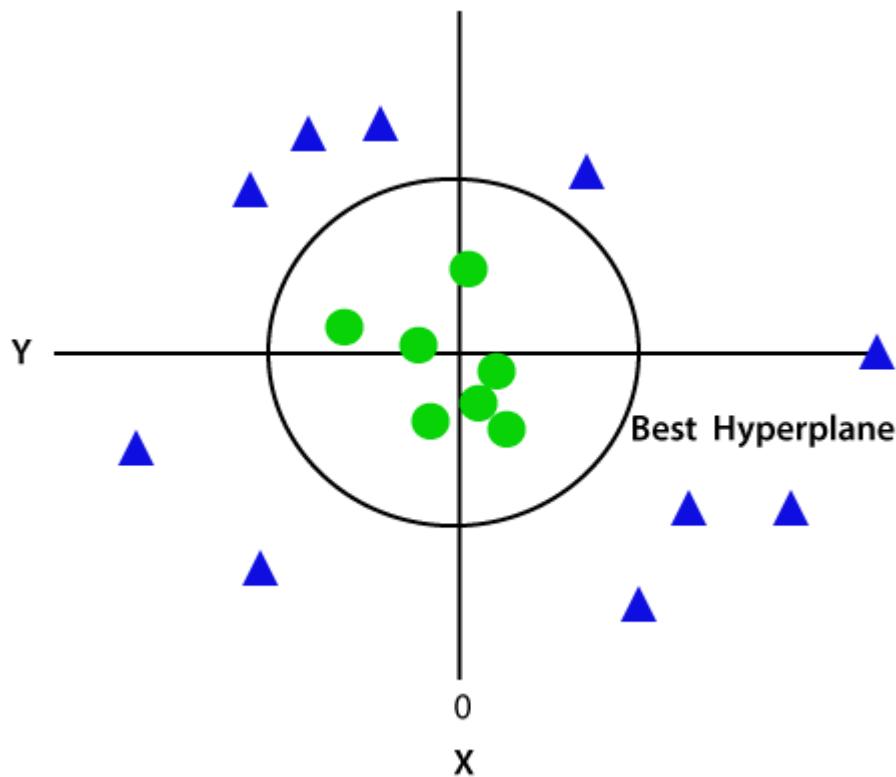
By adding the third dimension, the sample space will become as below image:



So now, SVM will divide the datasets into classes in the following way. Consider the below image:



Since we are in 3-d Space, hence it is looking like a plane parallel to the x-axis. If we convert it in 2d space with $z=1$, then it will become as:



Hence we get a circumference of radius 1 in case of non-linear data.

Python Implementation of Support Vector Machine

Now we will implement the SVM algorithm using Python. Here we will use the same dataset **user_data**, which we have used in Logistic regression and KNN classification.

- **Data Pre-processing step**

Till the Data pre-processing step, the code will remain the same. Below is the code:

```
1.      #Data Pre-processing Step
2.      # importing libraries
3.      import numpy as nm
4.      import matplotlib.pyplot as mtp
5.      import pandas as pd
6.
7.      #importing datasets
8.      data_set= pd.read_csv('user_data.csv')
```

```
9.  
10.     #Extracting Independent and dependent Variable  
11.     x= data_set.iloc[:, [2,3]].values  
12.     y= data_set.iloc[:, 4].values  
13.  
14.     # Splitting the dataset into training and test set.  
15.     from sklearn.model_selection import train_test_split  
16.     x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_s  
tate=0)  
17.     #feature Scaling  
18.     from sklearn.preprocessing import StandardScaler  
19.     st_x= StandardScaler()  
20.     x_train= st_x.fit_transform(x_train)  
21.     x_test= st_x.transform(x_test)
```

After executing the above code, we will pre-process the data. The code will give the dataset as:

data_set - DataFrame

Index	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1
8	15600575	Male	25	33000	0
9	15727311	Female	35	65000	0
10	15570769	Female	26	80000	0
11	15606274	Female	26	52000	0
12	15746139	Male	20	86000	0
13	15704987	Male	32	18000	0
14	15628972	Male	18	82000	0

Format Resize Background color Column min/max Save and Close Close

The scaled output for the test set will be:

x_test - NumPy array

	0	1	
0	-0.804802	0.504964	
1	-0.0125441	-0.567782	
2	-0.309641	0.157046	
3	-0.804802	0.273019	
4	-0.309641	-0.567782	
5	-1.1019	-1.43758	
6	-0.70577	-1.58254	
7	-0.210609	2.15757	
8	-1.99319	-0.0459058	
9	0.878746	-0.770734	
10	-0.804802	-0.596776	
11	-1.00287	-0.422817	
12	-0.111576	-0.422817	

Format Resize Background color

Save and Close **Close**

y_test - NumPy array

	0
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0
9	0
10	0
11	0
12	0

Format Resize Background color

Save and Close

Fitting the SVM classifier to the training set:

Now the training set will be fitted to the SVM classifier. To create the SVM classifier, we will import **SVC** class from **Sklearn.svm** library. Below is the code for it:

1. `from sklearn.svm import SVC # "Support vector classifier"`
2. `classifier = SVC(kernel='linear', random_state=0)`
3. `classifier.fit(x_train, y_train)`

In the above code, we have used **kernel='linear'**, as here we are creating SVM for linearly separable data. However, we can change it for non-linear data. And then we fitted the classifier to the training dataset(`x_train, y_train`)

Output:

```
Out[8]:
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
     decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
```

```
kernel='linear', max_iter=-1, probability=False, random_state=0,  
shrinking=True, tol=0.001, verbose=False)
```

The model performance can be altered by changing the value of **C(Regularization factor), gamma, and kernel**.

- **Predicting the test set result:**

Now, we will predict the output for test set. For this, we will create a new vector `y_pred`. Below is the code for it:

1. `#Predicting the test set result`
2. `y_pred= classifier.predict(x_test)`

After getting the `y_pred` vector, we can compare the result of **y_pred** and **y_test** to check the difference between the actual value and predicted value.

Output: Below is the output for the prediction of the test set:

y_pred - NumPy array	
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0
9	0
10	0
11	0
12	0

- o **Creating the confusion matrix:**

Now we will see the performance of the SVM classifier that how many incorrect predictions are there as compared to the Logistic regression classifier. To create the confusion matrix, we need to import the **confusion_matrix** function of the **sklearn** library. After importing the function, we will call it using a new variable **cm**. The function takes two parameters, mainly **y_true** (the actual values) and **y_pred** (the targeted value return by the classifier). Below is the code for it:

1. `#Creating the Confusion matrix`
2. `from sklearn.metrics import confusion_matrix`
3. `cm= confusion_matrix(y_test, y_pred)`

Output:



As we can see in the above output image, there are $66+24= 90$ correct predictions and $8+2= 10$ correct predictions. Therefore we can say that our SVM model improved as compared to the Logistic regression model.

- o **Visualizing the training set result:**

Now we will visualize the training set result, below is the code for it:

1. `from matplotlib.colors import ListedColormap`
2. `x_set, y_set = x_train, y_train`

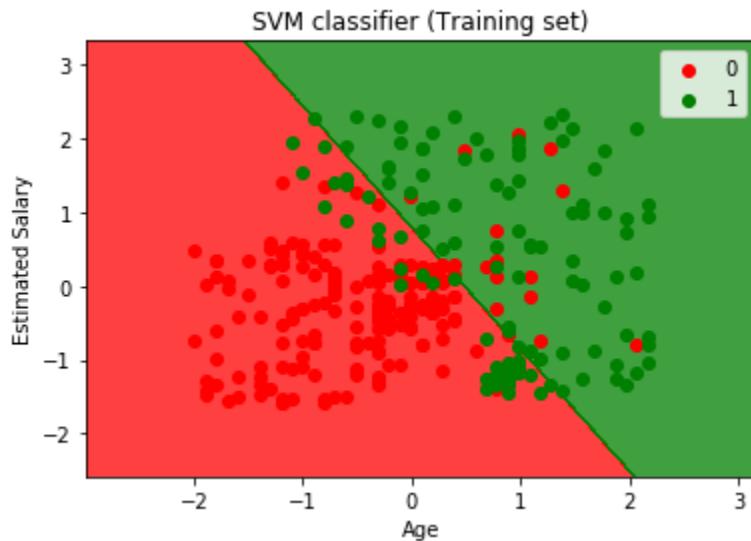
```

3.     x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() -
1, stop = x_set[:, 0].max() + 1, step = 0.01),
4.     nm.arange(start = x_set[:, 1].min() -
1, stop = x_set[:, 1].max() + 1, step = 0.01))
5.     mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
6.     alpha = 0.75, cmap = ListedColormap(('red', 'green')))
7.     mtp.xlim(x1.min(), x1.max())
8.     mtp.ylim(x2.min(), x2.max())
9.     for i, j in enumerate(nm.unique(y_set)):
10.         mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
11.                     c = ListedColormap(('red', 'green'))(i), label = j)
12.     mtp.title('SVM classifier (Training set)')
13.     mtp.xlabel('Age')
14.     mtp.ylabel('Estimated Salary')
15.     mtp.legend()
16.     mtp.show()

```

Output:

By executing the above code, we will get the output as:



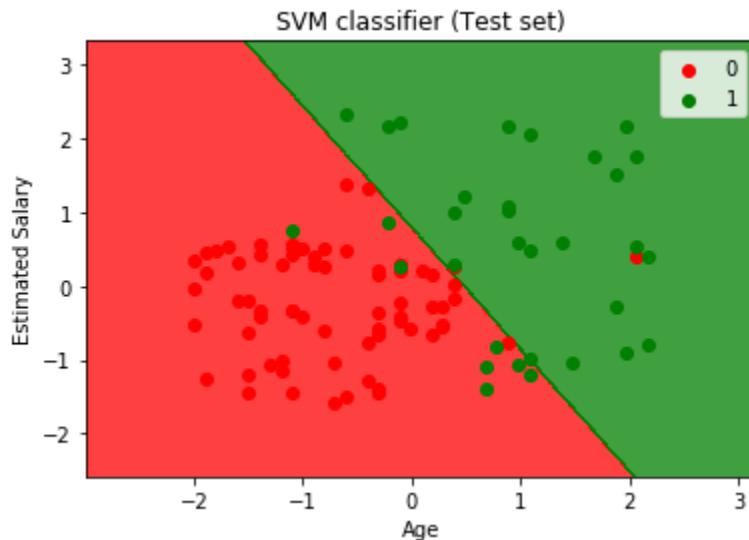
As we can see, the above output is appearing similar to the Logistic regression output. In the output, we got the straight line as hyperplane because we have **used a linear kernel in the classifier**. And we have also discussed above that for the 2d space, the hyperplane in SVM is a straight line.

- o **Visualizing the test set result:**

```
1.      #Visulaizing the test set result
2.      from matplotlib.colors import ListedColormap
3.      x_set, y_set = x_test, y_test
4.      x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() -
1, stop = x_set[:, 0].max() + 1, step = 0.01),
5.      nm.arange(start = x_set[:, 1].min() -
1, stop = x_set[:, 1].max() + 1, step = 0.01))
6.      mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
7.      alpha = 0.75, cmap = ListedColormap(('red','green' )))
8.      mtp.xlim(x1.min(), x1.max())
9.      mtp.ylim(x2.min(), x2.max())
10.     for i, j in enumerate(nm.unique(y_set)):
11.         mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
12.                     c = ListedColormap(('red', 'green'))(i), label = j)
13.     mtp.title('SVM classifier (Test set)')
14.     mtp.xlabel('Age')
15.     mtp.ylabel('Estimated Salary')
16.     mtp.legend()
17.     mtp.show()
```

Output:

By executing the above code, we will get the output as:



As we can see in the above output image, the SVM classifier has divided the users into two regions (Purchased or Not purchased). Users who purchased the SUV are in the red region with the red scatter points. And users who did not purchase the SUV are in the green region with green scatter points. The hyperplane has divided the two classes into Purchased and not purchased variable.

Naïve Bayes Classifier Algorithm

- Naïve Bayes algorithm is a supervised learning algorithm, which is based on **Bayes theorem** and used for solving classification problems.
- It is mainly used in *text classification* that includes a high-dimensional training dataset.
- Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.
- **It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.**
- Some popular examples of Naïve Bayes Algorithm are **spam filtration, Sentimental analysis, and classifying articles.**

Why is it called Naïve Bayes?

The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, Which can be described as:

- **Naïve:** It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the bases of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.
- **Bayes:** It is called Bayes because it depends on the principle of [Bayes' Theorem](#).

Bayes' Theorem:

- Bayes' theorem is also known as **Bayes' Rule** or **Bayes' law**, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.
- The formula for Bayes' theorem is given as:

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

Where,

P(A|B) is Posterior probability: Probability of hypothesis A on the observed event B.

P(B|A) is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true.

P(A) is Prior Probability: Probability of hypothesis before observing the evidence.

P(B) is Marginal Probability: Probability of Evidence.

Working of Naïve Bayes' Classifier:

Working of Naïve Bayes' Classifier can be understood with the help of the below example:

Suppose we have a dataset of **weather conditions** and corresponding target variable "**Play**". So using this dataset we need to decide that whether we should play or not on a particular day according to the weather conditions. So to solve this problem, we need to follow the below steps:

1. Convert the given dataset into frequency tables.

2. Generate Likelihood table by finding the probabilities of given features.
3. Now, use Bayes theorem to calculate the posterior probability.

Problem: If the weather is sunny, then the Player should play or not?

Solution: To solve this, first consider the below dataset:

Outlook		Play
0	Rainy	Yes
1	Sunny	Yes
2	Overcast	Yes
3	Overcast	Yes
4	Sunny	No
5	Rainy	Yes
6	Sunny	Yes
7	Overcast	Yes
8	Rainy	No
9	Sunny	No
10	Sunny	Yes
11	Rainy	No

12	Overcast	Yes
13	Overcast	Yes

Frequency table for the Weather Conditions:

Weather	Yes	No
Overcast	5	0
Rainy	2	2
Sunny	3	2
Total	10	5

Likelihood table weather condition:

Weather	No	Yes	
Overcast	0	5	5/14
Rainy	2	2	4/14
Sunny	2	3	5/14
All	4/14=0.29	10/14=0.71	

Applying Bayes' theorem:

$$P(\text{Yes}|\text{Sunny}) = P(\text{Sunny}|\text{Yes}) * P(\text{Yes}) / P(\text{Sunny})$$

$$P(\text{Sunny}|\text{Yes}) = 3/10 = 0.3$$

$P(\text{Sunny}) = 0.35$

$P(\text{Yes}) = 0.71$

So $P(\text{Yes}|\text{Sunny}) = 0.3 * 0.71 / 0.35 = \mathbf{0.60}$

$P(\text{No}|\text{Sunny}) = P(\text{Sunny}|\text{No}) * P(\text{No}) / P(\text{Sunny})$

$P(\text{Sunny}|\text{No}) = 2/4 = 0.5$

$P(\text{No}) = 0.29$

$P(\text{Sunny}) = 0.35$

So $P(\text{No}|\text{Sunny}) = 0.5 * 0.29 / 0.35 = \mathbf{0.41}$

So as we can see from the above calculation that **$P(\text{Yes}|\text{Sunny}) > P(\text{No}|\text{Sunny})$**

Hence on a Sunny day, Player can play the game.

Advantages of Naïve Bayes Classifier:

- Naïve Bayes is one of the fast and easy ML algorithms to predict a class of datasets.
- It can be used for Binary as well as Multi-class Classifications.
- It performs well in Multi-class predictions as compared to the other Algorithms.
- It is the most popular choice for **text classification problems**.

Disadvantages of Naïve Bayes Classifier:

- Naive Bayes assumes that all features are independent or unrelated, so it cannot learn the relationship between features.

Applications of Naïve Bayes Classifier:

- It is used for **Credit Scoring**.
- It is used in **medical data classification**.
- It can be used in **real-time predictions** because Naïve Bayes Classifier is an eager learner.
- It is used in Text classification such as **Spam filtering** and **Sentiment analysis**.

Types of Naïve Bayes Model:

There are three types of Naive Bayes Model, which are given below:

- **Gaussian:** The Gaussian model assumes that features follow a normal distribution. This means if predictors take continuous values instead of discrete, then the model assumes that these values are sampled from the Gaussian distribution.
- **Multinomial:** The Multinomial Naïve Bayes classifier is used when the data is multinomial distributed. It is primarily used for document classification problems, it means a particular document belongs to which category such as Sports, Politics, education, etc.
The classifier uses the frequency of words for the predictors.
- **Bernoulli:** The Bernoulli classifier works similar to the Multinomial classifier, but the predictor variables are the independent Booleans variables. Such as if a particular word is present or not in a document. This model is also famous for document classification tasks.

Python Implementation of the Naïve Bayes algorithm:

Now we will implement a Naive Bayes Algorithm using Python. So for this, we will use the "**user_data**" **dataset**, which we have used in our other classification model. Therefore we can easily compare the Naive Bayes model with the other models.

Steps to implement:

- Data Pre-processing step
- Fitting Naive Bayes to the Training set
- Predicting the test result
- Test accuracy of the result(Creation of Confusion matrix)
- Visualizing the test set result.

1) Data Pre-processing step:

In this step, we will pre-process/prepare the data so that we can use it efficiently in our code. It is similar as we did in [data-pre-processing](#). The code for this is given below:

1. Importing the libraries
2. **import** numpy as nm
3. **import** matplotlib.pyplot as mtp
4. **import** pandas as pd

```
5.
6.     # Importing the dataset
7.     dataset = pd.read_csv('user_data.csv')
8.     x = dataset.iloc[:, [2, 3]].values
9.     y = dataset.iloc[:, 4].values
10.
11.    # Splitting the dataset into the Training set and Test set
12.    from sklearn.model_selection import train_test_split
13.    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.25, random
14.    _state = 0)
15.
16.    # Feature Scaling
17.    from sklearn.preprocessing import StandardScaler
18.    sc = StandardScaler()
19.    x_train = sc.fit_transform(x_train)
20.    x_test = sc.transform(x_test)
```

In the above code, we have loaded the dataset into our program using "**dataset = pd.read_csv('user_data.csv')**". The loaded dataset is divided into training and test set, and then we have scaled the feature variable.

The output for the dataset is given as:

dataset - DataFrame

Index	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1
8	15600575	Male	25	33000	0
9	15727311	Female	35	65000	0
10	15570769	Female	26	80000	0
11	15606274	Female	26	52000	0
12	15746139	Male	20	86000	0
13	15704987	Male	32	18000	0
14	15628972	Male	18	82000	0
15	15697686	Male	29	80000	0
16	15733883	Male	47	25000	1
17	15617482	Male	45	26000	1
18	15704583	Male	46	28000	1
19	15621083	Female	48	29000	1

2) Fitting Naive Bayes to the Training Set:

After the pre-processing step, now we will fit the Naive Bayes model to the Training set. Below is the code for it:

```
1. # Fitting Naive Bayes to the Training set
2. from sklearn.naive_bayes import GaussianNB
3. classifier = GaussianNB()
4. classifier.fit(x_train, y_train)
```

In the above code, we have used the **GaussianNB classifier** to fit it to the training dataset. We can also use other classifiers as per our requirement.

Output:

```
Out[6]: GaussianNB(priors=None, var_smoothing=1e-09)
```

3) Prediction of the test set result:

Now we will predict the test set result. For this, we will create a new predictor variable **y_pred**, and will use the predict function to make the predictions.

```
1. # Predicting the Test set results
2. y_pred = classifier.predict(x_test)
```

Output:

y_pred - NumPy array

	0
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0
9	1
10	0
11	0
12	0
13	0

Format
Resize
 Background color

Save and Close
Close

y_test - NumPy array

	0
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0
9	0
10	0
11	0
12	0
13	0

Format
Resize
 Background color

Save and Close

The above output shows the result for prediction vector **y_pred** and real vector **y_test**. We can see that some predictions are different from the real values, which are the incorrect predictions.

4) Creating Confusion Matrix:

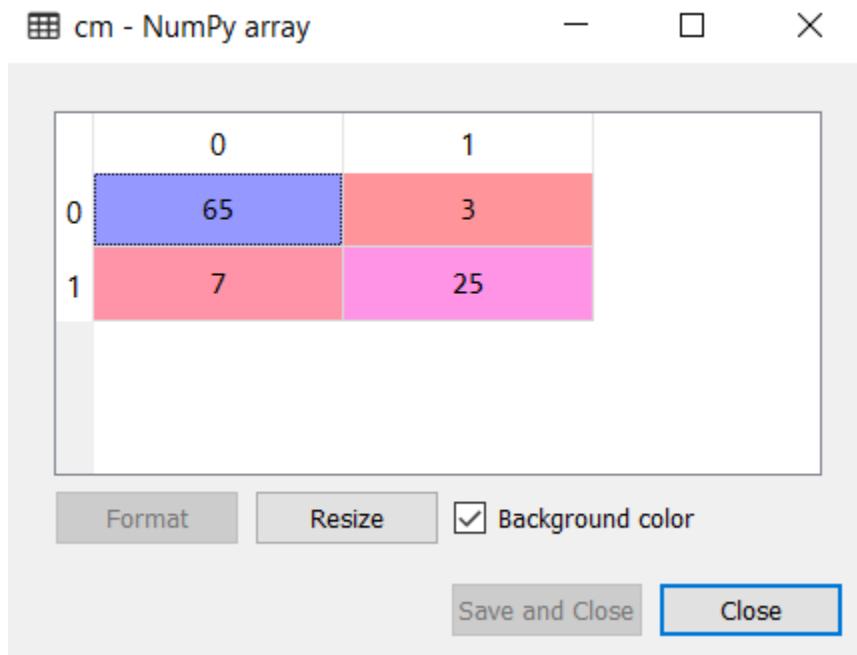
Now we will check the accuracy of the Naive Bayes classifier using the Confusion matrix. Below is the code for it:

```

1.      # Making the Confusion Matrix
2.      from sklearn.metrics import confusion_matrix
3.      cm = confusion_matrix(y_test, y_pred)

```

Output:



As we can see in the above confusion matrix output, there are $7+3= 10$ incorrect predictions, and $65+25=90$ correct predictions.

5) Visualizing the training set result:

Next we will visualize the training set result using Naïve Bayes Classifier. Below is the code for it:

```

1.      # Visualising the Training set results
2.      from matplotlib.colors import ListedColormap
3.      x_set, y_set = x_train, y_train
4.      X1, X2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() -
1, stop = x_set[:, 0].max() + 1, step = 0.01),
5.                            nm.arange(start = x_set[:, 1].min() -
1, stop = x_set[:, 1].max() + 1, step = 0.01))
6.      mtp.contourf(X1, X2, classifier.predict(nm.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
7.                    alpha = 0.75, cmap = ListedColormap(('purple', 'green')))

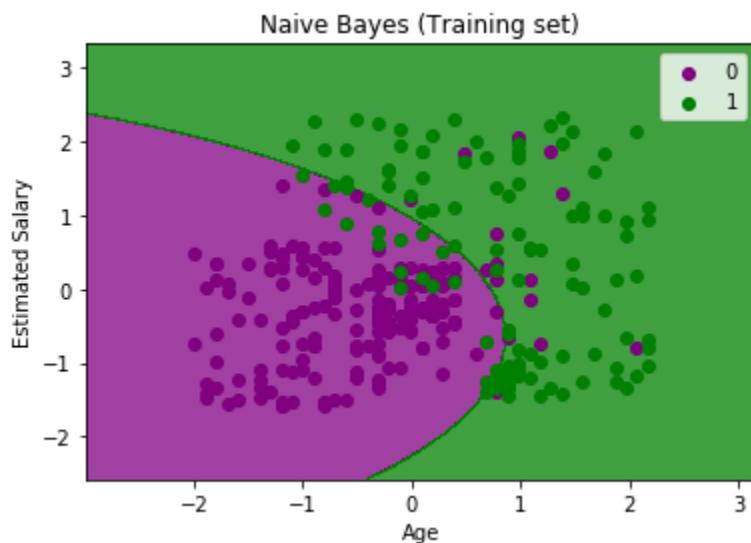
```

```

8.     mtp.xlim(X1.min(), X1.max())
9.     mtp.ylim(X2.min(), X2.max())
10.    for i, j in enumerate(nm.unique(y_set)):
11.        mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
12.                     c = ListedColormap(['purple', 'green'])(i), label = j)
13.    mtp.title('Naive Bayes (Training set)')
14.    mtp.xlabel('Age')
15.    mtp.ylabel('Estimated Salary')
16.    mtp.legend()
17.    mtp.show()

```

Output:



In the above output we can see that the Naïve Bayes classifier has segregated the data points with the fine boundary. It is Gaussian curve as we have used **GaussianNB** classifier in our code.

6) Visualizing the Test set result:

```

1.     # Visualising the Test set results
2.     from matplotlib.colors import ListedColormap
3.     x_set, y_set = x_test, y_test
4.     X1, X2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() -
1, stop = x_set[:, 0].max() + 1, step = 0.01),
5.                           nm.arange(start = x_set[:, 1].min() -
1, stop = x_set[:, 1].max() + 1, step = 0.01))

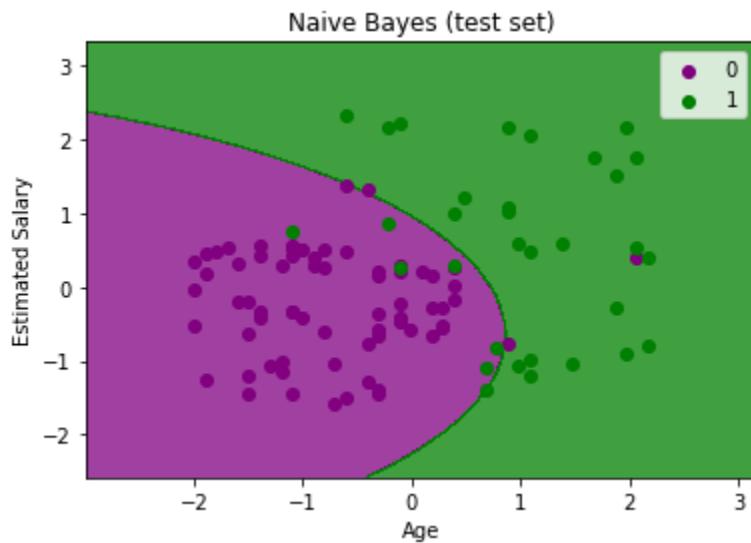
```

```

6.         mtp.contourf(X1, X2, classifier.predict(nm.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
7.                           alpha = 0.75, cmap = ListedColormap(['purple', 'green']))
8.         mtp.xlim(X1.min(), X1.max())
9.         mtp.ylim(X2.min(), X2.max())
10.        for i, j in enumerate(nm.unique(y_set)):
11.            mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
12.                        c = ListedColormap(['purple', 'green'])(i), label = j)
13.        mtp.title('Naive Bayes (test set)')
14.        mtp.xlabel('Age')
15.        mtp.ylabel('Estimated Salary')
16.        mtp.legend()
17.        mtp.show()

```

Output:



The above output is final output for test set data. As we can see the classifier has created a Gaussian curve to divide the "purchased" and "not purchased" variables. There are some wrong predictions which we have calculated in Confusion matrix. But still it is pretty good classifier.

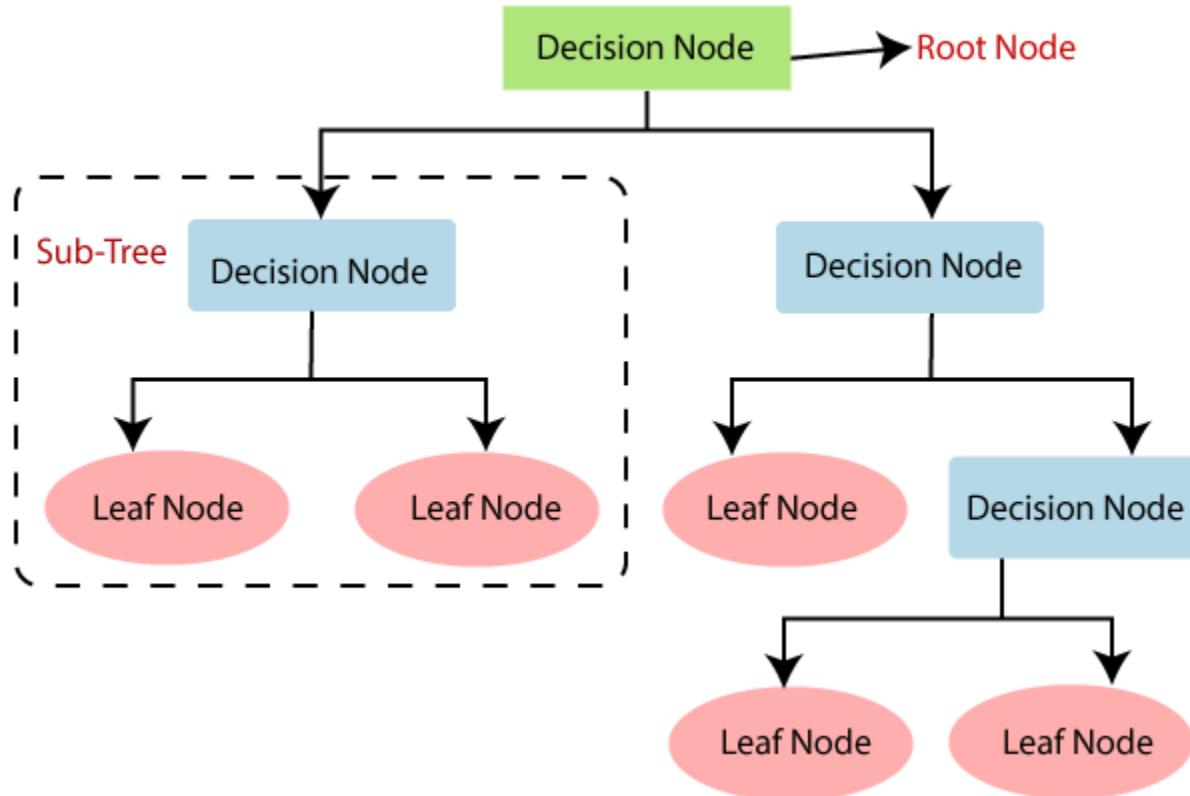
Decision Tree Classification Algorithm

- Decision Tree is a **Supervised learning technique** that can be used for both classification and Regression problems, but mostly it is preferred for solving

Classification problems. It is a tree-structured classifier, where **internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.**

- In a Decision tree, there are two nodes, which are the **Decision Node** and **Leaf Node**. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.
- The decisions or the test are performed on the basis of features of the given dataset.
- ***It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.***
- It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.
- In order to build a tree, we use the **CART algorithm**, which stands for **Classification and Regression Tree algorithm**.
- A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees.
- Below diagram explains the general structure of a decision tree:

Note: A decision tree can contain categorical data (YES/NO) as well as numeric data.



Why use Decision Trees?

There are various algorithms in Machine learning, so choosing the best algorithm for the given dataset and problem is the main point to remember while creating a machine learning model. Below are the two reasons for using the Decision tree:

- Decision Trees usually mimic human thinking ability while making a decision, so it is easy to understand.
- The logic behind the decision tree can be easily understood because it shows a tree-like structure.

Decision Tree Terminologies

- **Root Node:** Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets.
- **Leaf Node:** Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.
- **Splitting:** Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.

- **Branch/Sub Tree:** A tree formed by splitting the tree.
- **Pruning:** Pruning is the process of removing the unwanted branches from the tree.
- **Parent/Child node:** The root node of the tree is called the parent node, and other nodes are called the child nodes.

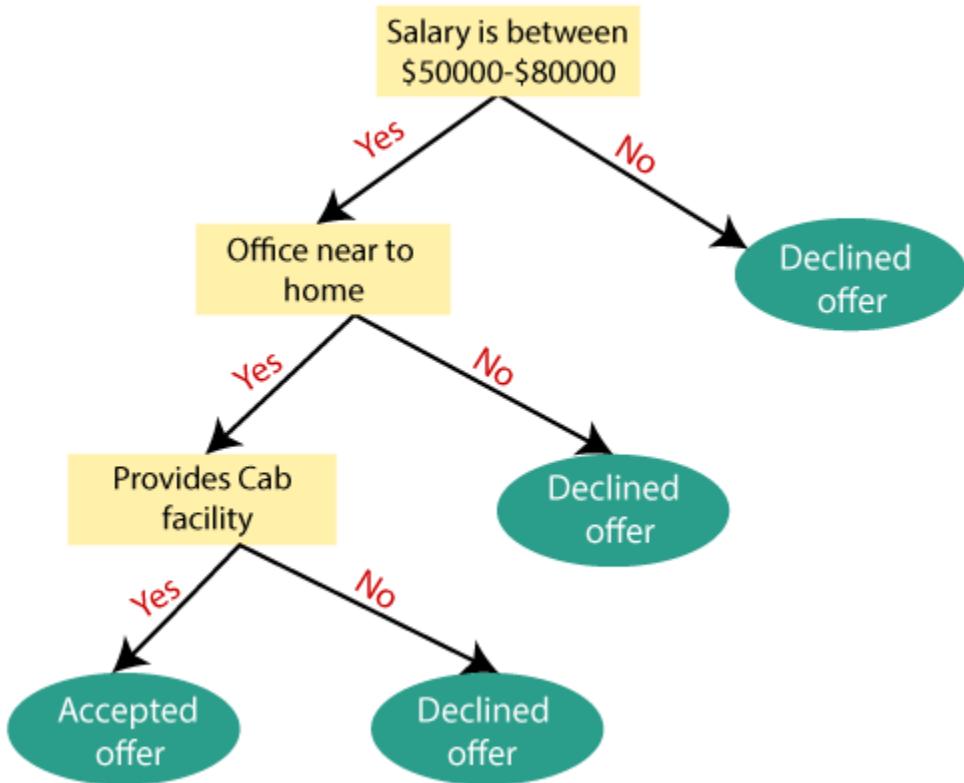
How does the Decision Tree algorithm Work?

In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node.

For the next node, the algorithm again compares the attribute value with the other sub-nodes and move further. It continues the process until it reaches the leaf node of the tree. The complete process can be better understood using the below algorithm:

- **Step-1:** Begin the tree with the root node, says S, which contains the complete dataset.
- **Step-2:** Find the best attribute in the dataset using **Attribute Selection Measure (ASM)**.
- **Step-3:** Divide the S into subsets that contains possible values for the best attributes.
- **Step-4:** Generate the decision tree node, which contains the best attribute.
- **Step-5:** Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

Example: Suppose there is a candidate who has a job offer and wants to decide whether he should accept the offer or Not. So, to solve this problem, the decision tree starts with the root node (Salary attribute by ASM). The root node splits further into the next decision node (distance from the office) and one leaf node based on the corresponding labels. The next decision node further gets split into one decision node (Cab facility) and one leaf node. Finally, the decision node splits into two leaf nodes (Accepted offers and Declined offer). Consider the below diagram:



Attribute Selection Measures

While implementing a Decision tree, the main issue arises that how to select the best attribute for the root node and for sub-nodes. So, to solve such problems there is a technique which is called as **Attribute selection measure or ASM**. By this measurement, we can easily select the best attribute for the nodes of the tree. There are two popular techniques for ASM, which are:

- **Information Gain**
- **Gini Index**

1. Information Gain:

- Information gain is the measurement of changes in entropy after the segmentation of a dataset based on an attribute.
- It calculates how much information a feature provides us about a class.
- According to the value of information gain, we split the node and build the decision tree.
- A decision tree algorithm always tries to maximize the value of information gain, and a node/attribute having the highest information gain is split first. It can be calculated using the below formula:

1. Information Gain= Entropy(S)- [(Weighted Avg) *Entropy(each feature)]

Entropy: Entropy is a metric to measure the impurity in a given attribute. It specifies randomness in data. Entropy can be calculated as:

$$\text{Entropy}(S) = -P(\text{yes}) \log_2 P(\text{yes}) - P(\text{no}) \log_2 P(\text{no})$$

Where,

- o **S= Total number of samples**
- o **P(yes)= probability of yes**
- o **P(no)= probability of no**

2. Gini Index:

- o Gini index is a measure of impurity or purity used while creating a decision tree in the CART(Classification and Regression Tree) algorithm.
- o An attribute with the low Gini index should be preferred as compared to the high Gini index.
- o It only creates binary splits, and the CART algorithm uses the Gini index to create binary splits.
- o Gini index can be calculated using the below formula:

$$\text{Gini Index} = 1 - \sum_j P_j^2$$

Pruning: Getting an Optimal Decision tree

Pruning is a process of deleting the unnecessary nodes from a tree in order to get the optimal decision tree.

A too-large tree increases the risk of overfitting, and a small tree may not capture all the important features of the dataset. Therefore, a technique that decreases the size of the learning tree without reducing accuracy is known as Pruning. There are mainly two types of tree **pruning** technology used:

- o **Cost Complexity Pruning**
- o **Reduced Error Pruning.**

Advantages of the Decision Tree

- o It is simple to understand as it follows the same process which a human follow while making any decision in real-life.

- It can be very useful for solving decision-related problems.
- It helps to think about all the possible outcomes for a problem.
- There is less requirement of data cleaning compared to other algorithms.

Disadvantages of the Decision Tree

- The decision tree contains lots of layers, which makes it complex.
- It may have an overfitting issue, which can be resolved using the **Random Forest algorithm.**
- For more class labels, the computational complexity of the decision tree may increase.

Python Implementation of Decision Tree

Now we will implement the Decision tree using Python. For this, we will use the dataset "**user_data.csv**," which we have used in previous classification models. By using the same dataset, we can compare the Decision tree classifier with other classification models such as [KNN](#) [SVM](#), [LogisticRegression](#), etc.

Steps will also remain the same, which are given below:

- **Data Pre-processing step**
- **Fitting a Decision-Tree algorithm to the Training set**
- **Predicting the test result**
- **Test accuracy of the result(Creation of Confusion matrix)**
- **Visualizing the test set result.**

1. Data Pre-Processing Step:

Below is the code for the pre-processing step:

1. # importing libraries
2. **import** numpy as nm
3. **import** matplotlib.pyplot as mtp
4. **import** pandas as pd
- 5.
6. #importing datasets
7. data_set= pd.read_csv('user_data.csv')

```
8.
9.     #Extracting Independent and dependent Variable
10.    x= data_set.iloc[:, [2,3]].values
11.    y= data_set.iloc[:, 4].values
12.
13.    # Splitting the dataset into training and test set.
14.    from sklearn.model_selection import train_test_split
15.    x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_s
tate=0)
16.
17.    #feature Scaling
18.    from sklearn.preprocessing import StandardScaler
19.    st_x= StandardScaler()
20.    x_train= st_x.fit_transform(x_train)
21.    x_test= st_x.transform(x_test)
```

In the above code, we have pre-processed the data. Where we have loaded the dataset, which is given as:

data_set - DataFrame

Index	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1
8	15600575	Male	25	33000	0
9	15727311	Female	35	65000	0
10	15570769	Female	26	80000	0
11	15606274	Female	26	52000	0
12	15746139	Male	20	86000	0
13	15704987	Male	32	18000	0
14	15628972	Male	18	82000	0
15	15697686	Male	29	80000	0

Format Resize Background color Column min/max Save and Close Close

2. Fitting a Decision-Tree algorithm to the Training set

Now we will fit the model to the training set. For this, we will import the **DecisionTreeClassifier** class from **sklearn.tree** library. Below is the code for it:

1. #Fitting Decision Tree classifier to the training set
2. From sklearn.tree **import** DecisionTreeClassifier
3. classifier= DecisionTreeClassifier(criterion='entropy', random_state=0)
4. classifier.fit(x_train, y_train)

In the above code, we have created a classifier object, in which we have passed two main parameters;

- **"criterion='entropy":** Criterion is used to measure the quality of split, which is calculated by information gain given by entropy.
- **random_state=0":** For generating the random states.

Below is the output for this:

```
Out[8]:  
DecisionTreeClassifier(class_weight=None, criterion='entropy',  
max_depth=None,  
max_features=None, max_leaf_nodes=None,  
min_impurity_decrease=0.0, min_impurity_split=None,  
min_samples_leaf=1, min_samples_split=2,  
min_weight_fraction_leaf=0.0, presort=False,  
random_state=0, splitter='best')
```

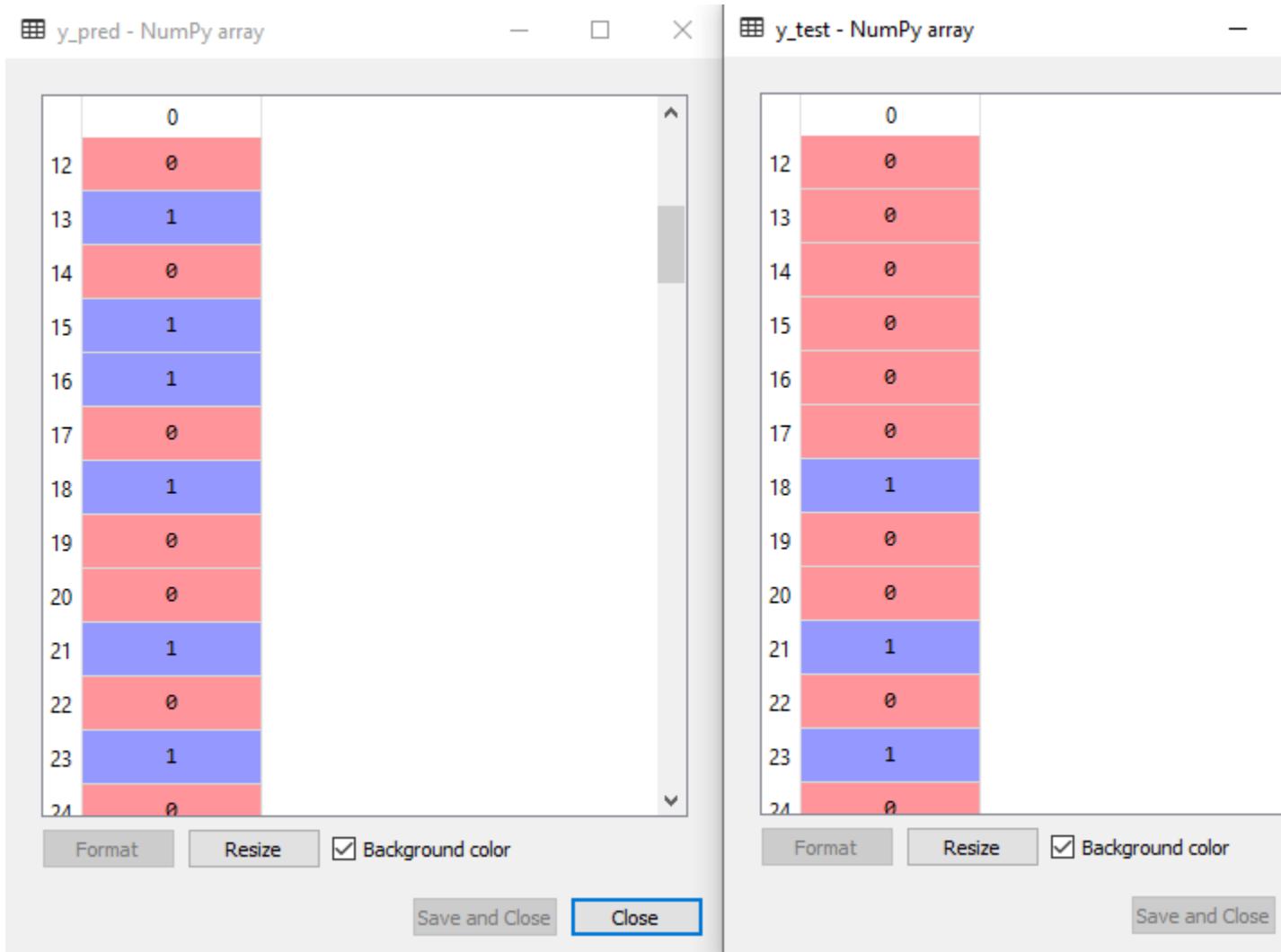
3. Predicting the test result

Now we will predict the test set result. We will create a new prediction vector **y_pred**. Below is the code for it:

1. #Predicting the test set result
2. y_pred= classifier.predict(x_test)

Output:

In the below output image, the predicted output and real test output are given. We can clearly see that there are some values in the prediction vector, which are different from the real vector values. These are prediction errors.

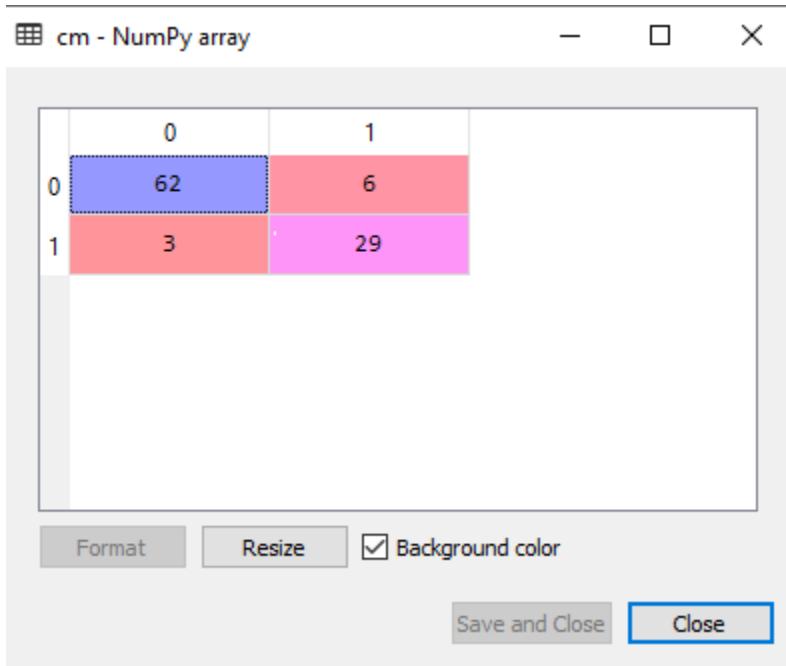


4. Test accuracy of the result (Creation of Confusion matrix)

In the above output, we have seen that there were some incorrect predictions, so if we want to know the number of correct and incorrect predictions, we need to use the confusion matrix. Below is the code for it:

1. #Creating the Confusion matrix
2. from sklearn.metrics **import** confusion_matrix
3. cm= confusion_matrix(y_test, y_pred)

Output:



In the above output image, we can see the confusion matrix, which has **6+3= 9 incorrect predictions** and **62+29=91 correct predictions**. Therefore, we can say that compared to other classification models, the Decision Tree classifier made a good prediction.

5. Visualizing the training set result:

Here we will visualize the training set result. To visualize the training set result we will plot a graph for the decision tree classifier. The classifier will predict yes or No for the users who have either Purchased or Not purchased the SUV car as we did in [Logistic Regression](#). Below is the code for it:

```

1.      #Visualizing the trianing set result
2.      from matplotlib.colors import ListedColormap
3.      x_set, y_set = x_train, y_train
4.      x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() -
1, stop = x_set[:, 0].max() + 1, step = 0.01),
5.      nm.arange(start = x_set[:, 1].min() -
1, stop = x_set[:, 1].max() + 1, step = 0.01))
6.      mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
7.      alpha = 0.75, cmap = ListedColormap(('purple','green' )))
8.      mtp.xlim(x1.min(), x1.max())
9.      mtp.ylim(x2.min(), x2.max())
10.     for i, j in enumerate(nm.unique(y_set)):
```

```

11.     mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
12.                     c = ListedColormap(['purple', 'green'])(i), label = j)
13.     mtp.title('Decision Tree Algorithm (Training set)')
14.     mtp.xlabel('Age')
15.     mtp.ylabel('Estimated Salary')
16.     mtp.legend()
17.     mtp.show()

```

Output:



The above output is completely different from the rest classification models. It has both vertical and horizontal lines that are splitting the dataset according to the age and estimated salary variable.

As we can see, the tree is trying to capture each dataset, which is the case of overfitting.

6. Visualizing the test set result:

Visualization of test set result will be similar to the visualization of the training set except that the training set will be replaced with the test set.

```

1.     #Visualizing the test set result
2.     from matplotlib.colors import ListedColormap
3.     x_set, y_set = x_test, y_test
4.     x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() -
1, stop = x_set[:, 0].max() + 1, step = 0.01),

```

```

5.         nm.arange(start = x_set[:, 1].min() -
6.             1, stop = x_set[:, 1].max() + 1, step = 0.01))
7.         mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
8.             alpha = 0.75, cmap = ListedColormap(('purple','green' )))
9.         mtp.xlim(x1.min(), x1.max())
10.        mtp.ylim(x2.min(), x2.max())
11.        fori, j in enumerate(nm.unique(y_set)):
12.            mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
13.                         c = ListedColormap(('purple', 'green'))(i), label = j)
14.        mtp.title('Decision Tree Algorithm(Test set)')
15.        mtp.xlabel('Age')
16.        mtp.ylabel('Estimated Salary')
17.        mtp.legend()
18.        mtp.show()

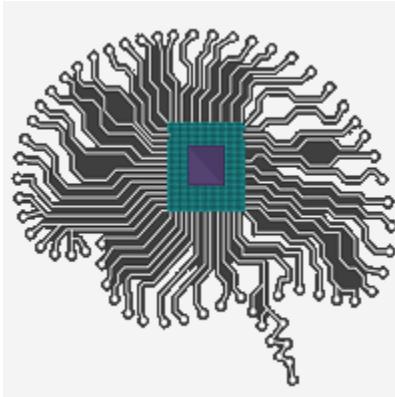
```

Output:



As we can see in the above image that there are some green data points within the purple region and vice versa. So, these are the incorrect predictions which we have discussed in the confusion matrix.

Artificial Neural Network



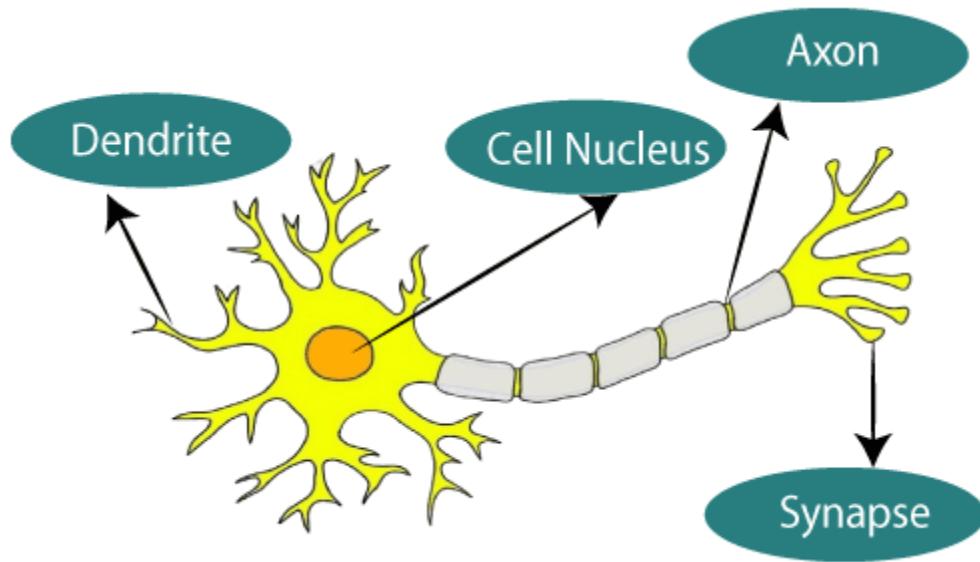
Artificial Neural Network Tutorial provides basic and advanced concepts of ANNs. Our Artificial Neural Network tutorial is developed for beginners as well as professionals.

The term "Artificial neural network" refers to a biologically inspired sub-field of artificial intelligence modeled after the brain. An Artificial neural network is usually a computational network based on biological neural networks that construct the structure of the human brain. Similar to a human brain has neurons interconnected to each other, artificial neural networks also have neurons that are linked to each other in various layers of the networks. These neurons are known as nodes.

Artificial neural network tutorial covers all the aspects related to the artificial neural network. In this tutorial, we will discuss ANNs, Adaptive resonance theory, Kohonen self-organizing map, Building blocks, unsupervised learning, Genetic algorithm, etc.

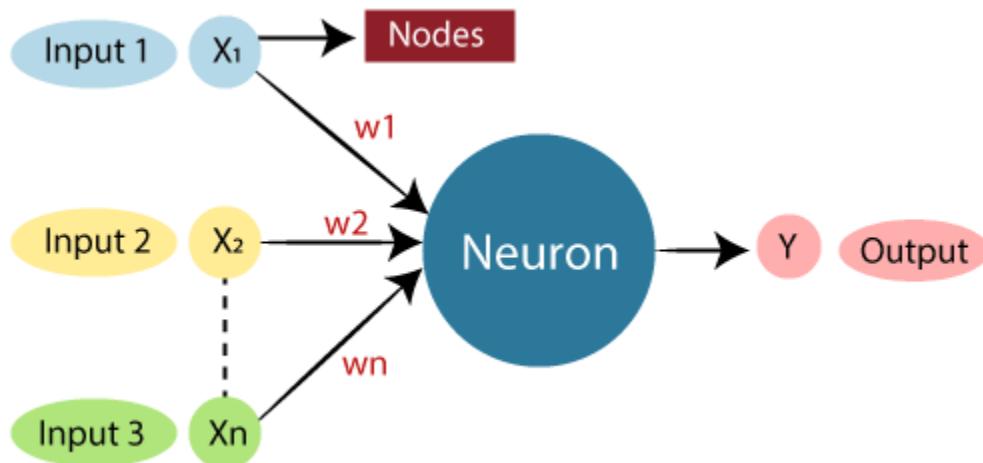
What is Artificial Neural Network?

The term "**Artificial Neural Network**" is derived from Biological neural networks that develop the structure of a human brain. Similar to the human brain that has neurons interconnected to one another, artificial neural networks also have neurons that are interconnected to one another in various layers of the networks. These neurons are known as nodes.



The given figure illustrates the typical diagram of Biological Neural Network.

The typical Artificial Neural Network looks something like the given figure.



Dendrites from Biological Neural Network represent inputs in Artificial Neural Networks, cell nucleus represents Nodes, synapse represents Weights, and Axon represents Output.

Relationship between Biological neural network and artificial neural network:

Biological Neural Network	Artificial Neural Network
Dendrites	Inputs

Cell nucleus	Nodes
Synapse	Weights
Axon	Output

An **Artificial Neural Network** in the field of **Artificial intelligence** where it attempts to mimic the network of neurons makes up a human brain so that computers will have an option to understand things and make decisions in a human-like manner. The artificial neural network is designed by programming computers to behave simply like interconnected brain cells.

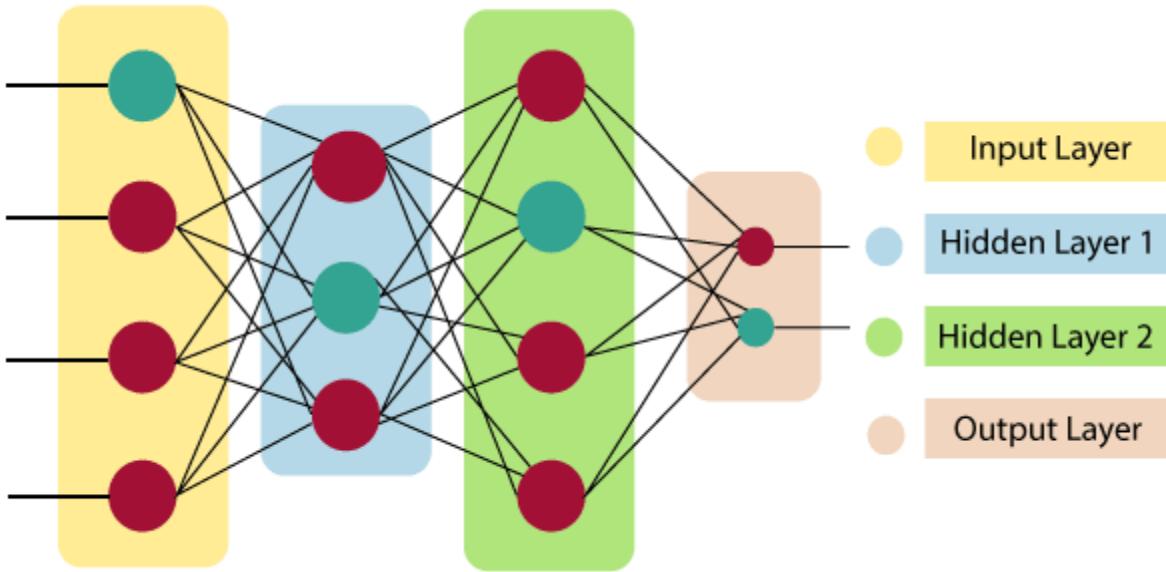
There are around 1000 billion neurons in the human brain. Each neuron has an association point somewhere in the range of 1,000 and 100,000. In the human brain, data is stored in such a manner as to be distributed, and we can extract more than one piece of this data when necessary from our memory parallelly. We can say that the human brain is made up of incredibly amazing parallel processors.

We can understand the artificial neural network with an example, consider an example of a digital logic gate that takes an input and gives an output. "OR" gate, which takes two inputs. If one or both the inputs are "On," then we get "On" in output. If both the inputs are "Off," then we get "Off" in output. Here the output depends upon input. Our brain does not perform the same task. The outputs to inputs relationship keep changing because of the neurons in our brain, which are "learning."

The architecture of an artificial neural network:

To understand the concept of the architecture of an artificial neural network, we have to understand what a neural network consists of. In order to define a neural network that consists of a large number of artificial neurons, which are termed units arranged in a sequence of layers. Lets us look at various types of layers available in an artificial neural network.

Artificial Neural Network primarily consists of three layers:



Input Layer:

As the name suggests, it accepts inputs in several different formats provided by the programmer.

Hidden Layer:

The hidden layer presents in-between input and output layers. It performs all the calculations to find hidden features and patterns.

Output Layer:

The input goes through a series of transformations using the hidden layer, which finally results in output that is conveyed using this layer.

The artificial neural network takes input and computes the weighted sum of the inputs and includes a bias. This computation is represented in the form of a transfer function.

$$\sum_{i=1}^n w_i * x_i + b$$

It determines weighted total is passed as an input to an activation function to produce the output. Activation functions choose whether a node should fire or not. Only those who are fired make it to the output layer. There are distinctive activation functions available that can be applied upon the sort of task we are performing.

Advantages of Artificial Neural Network (ANN)

Parallel processing capability:

Artificial neural networks have a numerical value that can perform more than one task simultaneously.

Storing data on the entire network:

Data that is used in traditional programming is stored on the whole network, not on a database. The disappearance of a couple of pieces of data in one place doesn't prevent the network from working.

Capability to work with incomplete knowledge:

After ANN training, the information may produce output even with inadequate data. The loss of performance here relies upon the significance of missing data.

Having a memory distribution:

For ANN to be able to adapt, it is important to determine the examples and to encourage the network according to the desired output by demonstrating these examples to the network. The succession of the network is directly proportional to the chosen instances, and if the event can't appear to the network in all its aspects, it can produce false output.

Having fault tolerance:

Extortion of one or more cells of ANN does not prohibit it from generating output, and this feature makes the network fault-tolerance.

Disadvantages of Artificial Neural Network:

Assurance of proper network structure:

There is no particular guideline for determining the structure of artificial neural networks. The appropriate network structure is accomplished through experience, trial, and error.

Unrecognized behavior of the network:

It is the most significant issue of ANN. When ANN produces a testing solution, it does not provide insight concerning why and how. It decreases trust in the network.

Hardware dependence:

Artificial neural networks need processors with parallel processing power, as per their structure. Therefore, the realization of the equipment is dependent.

Difficulty of showing the issue to the network:

ANNs can work with numerical data. Problems must be converted into numerical values before being introduced to ANN. The presentation mechanism to be resolved here will directly impact the performance of the network. It relies on the user's abilities.

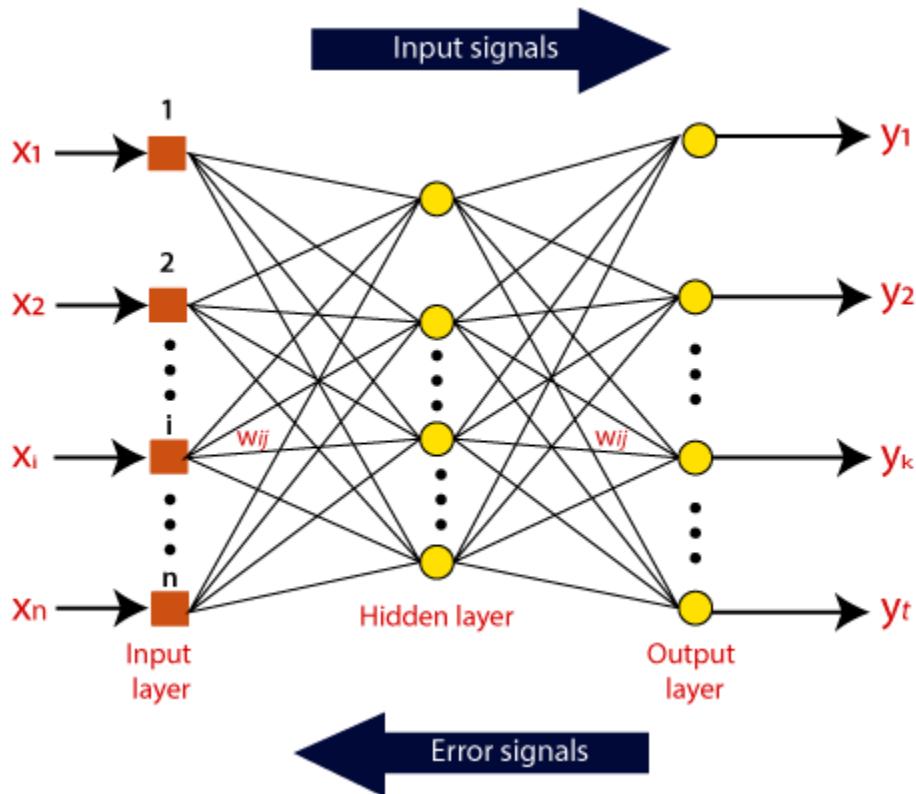
The duration of the network is unknown:

The network is reduced to a specific value of the error, and this value does not give us optimum results.

Science artificial neural networks that have steeped into the world in the mid-20th century are exponentially developing. In the present time, we have investigated the pros of artificial neural networks and the issues encountered in the course of their utilization. It should not be overlooked that the cons of ANN networks, which are a flourishing science branch, are eliminated individually, and their pros are increasing day by day. It means that artificial neural networks will turn into an irreplaceable part of our lives progressively important.

How do artificial neural networks work?

Artificial Neural Network can be best represented as a weighted directed graph, where the artificial neurons form the nodes. The association between the neurons outputs and neuron inputs can be viewed as the directed edges with weights. The Artificial Neural Network receives the input signal from the external source in the form of a pattern and image in the form of a vector. These inputs are then mathematically assigned by the notations $x(n)$ for every n number of inputs.



Afterward, each of the input is multiplied by its corresponding weights (these weights are the details utilized by the artificial neural networks to solve a specific problem). In general terms, these weights normally represent the strength of the interconnection between neurons inside the artificial neural network. All the weighted inputs are summarized inside the computing unit.

If the weighted sum is equal to zero, then bias is added to make the output non-zero or something else to scale up to the system's response. Bias has the same input, and weight equals to 1. Here the total of weighted inputs can be in the range of 0 to positive infinity. Here, to keep the response in the limits of the desired value, a certain maximum value is benchmarked, and the total of weighted inputs is passed through the activation function.

The activation function refers to the set of transfer functions used to achieve the desired output. There is a different kind of the activation function, but primarily either linear or non-linear sets of functions. Some of the commonly used sets of activation functions are the Binary, linear, and Tan hyperbolic sigmoidal activation functions. Let us take a look at each of them in details:

Binary:

In binary activation function, the output is either a one or a 0. Here, to accomplish this, there is a threshold value set up. If the net weighted input of neurons is more than 1, then the final output of the activation function is returned as one or else the output is returned as 0.

Sigmoidal Hyperbolic:

The Sigmoidal Hyperbola function is generally seen as an "S" shaped curve. Here the tan hyperbolic function is used to approximate output from the actual net input. The function is defined as:

$$F(x) = (1/(1 + \exp(-????x)))$$

Where **????** is considered the Steepness parameter.

Types of Artificial Neural Network:

There are various types of Artificial Neural Networks (ANN) depending upon the human brain neuron and network functions, an artificial neural network similarly performs tasks. The majority of the artificial neural networks will have some similarities with a more complex biological partner and are very effective at their expected tasks. For example, segmentation or classification.

Feedback ANN:

In this type of ANN, the output returns into the network to accomplish the best-evolved results internally. As per the **University of Massachusetts**, Lowell Centre for Atmospheric Research. The feedback networks feed information back into itself and are well suited to solve optimization issues. The Internal system error corrections utilize feedback ANNs.

Feed-Forward ANN:

A feed-forward network is a basic neural network comprising of an input layer, an output layer, and at least one layer of a neuron. Through assessment of its output by reviewing its input, the intensity of the network can be noticed based on group behavior of the associated neurons, and the output is decided. The primary advantage of this network is that it figures out how to evaluate and recognize input patterns.