UNIT – 1 (8 Marks and 16 Marks)

1.Is AI is a science or is it engineering? Or neither or both? Explain. (AU-MAY 2012).

Human vs. Machine

Everyone knows that humans and machines are different. Machines are the creation of humans, and they were created to make their work easier.

- Humans depend more and more on machines for their day-to-day things. Machines have created a revolution, and no human can think of a life without machines.
- A machine is only a device consisting of different parts, and is used for performing different functions. They do not have life, as they are mechanical. On the other hand, humans are made of flesh and blood; life is just not mechanical for humans.
- Humans have feelings and emotions, and they can express these emotions; happiness and sorrow are part of one's life. On the other hand, machines have no feelings and emotions. They just work as per the details fed into their mechanical brain.
- Humans have the capability to understand situations, and behave accordingly. On the contrary, Machines do not have this capability.
- While humans behave according to their consciousness, machines perform as they are taught. Humans perform activities as per their own intelligence. On the contrary, machines only have an artificial intelligence.
- It is a man-made intelligence that the machines have. The brilliance of the intelligence of a machine depends on the intelligence of the humans that created it.
- Another striking difference that can be seen is that humans can do anything original, and machines cannot. Machines have limitations to their performance because they need humans to guide them.

- Humans can do anything original, and machines cannot.

- Humans have the capability to understand situations, and behave accordingly. On the contrary, machines do not have this capability.

- While humans behave as per their consciousness, machines just perform as they are taught.

2. Explain the schematic of AI'S agent performing action. (Dec-09, 12, 14, May-12).

3. Explain the role of an agent program. (Dec-09,16)

Following diagram illustrates the agent's action process, as specified by architecture. This can also be termed as agent's structure.

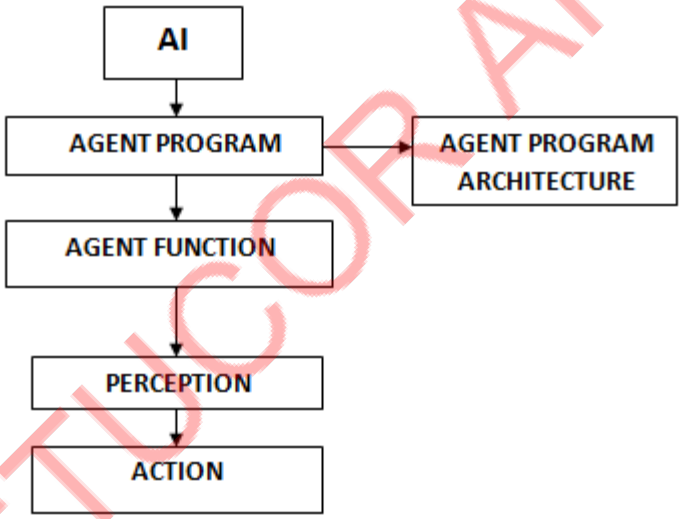


Fig: agent's action process

- An agent function program is internally implemented as agent function.
- An agent program takes input as the current percept from the sensor and returns an action to the effectors (actuators).

4. Discuss any 2 uninformed search methods with examples. (Dec-2009), explain the following uninformed search strategies. I) IDDFS AND 2) Bidirectional search. (May 2010); what is uninformed search and explain depth first search with example. (May-2013, Dec 2013 , may 2014; May 2015, Dec 2016)

An Uninformed search is a group of wide range usage algorithms of the era. These algorithms are brute force operations, and they don't have extra information about the search space; the only information they have is on how to traverse or visit the nodes in the tree. Thus uninformed

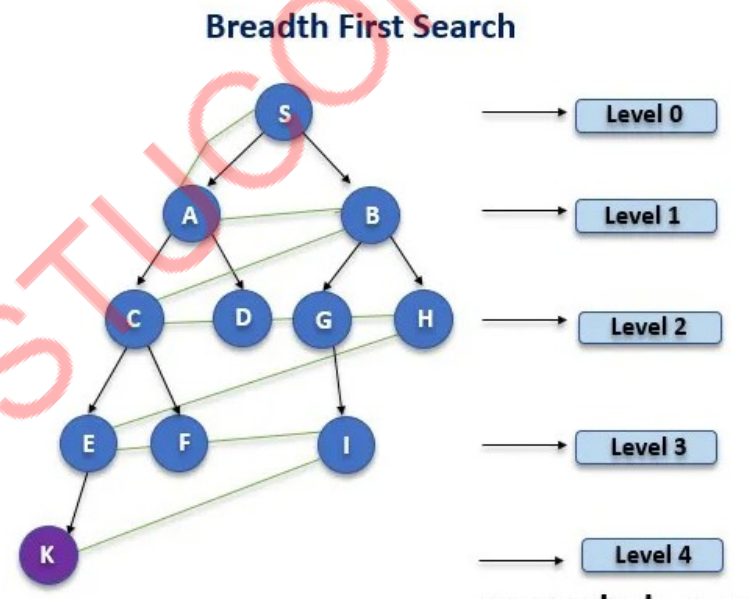
search algorithms are also called blind search algorithms. The search algorithm produces the search tree without using any domain knowledge, which is the brute force in nature. They are different from informed search algorithms in a way that you check for a goal when a node is generated or expanded, and they don't have any background information on how to approach the goal.

TYPES OF UNINFORMED SEARCH ALGORITHMS

➢ Breadth-First Search Algorithms

BFS is a search operation for finding the nodes in a tree. The algorithm works breadth wise and traverses to find the desired node in a tree. It starts searching operation from the root nodes and expands the successor nodes at that level before moving ahead and then moves along breadth wise for further expansion.

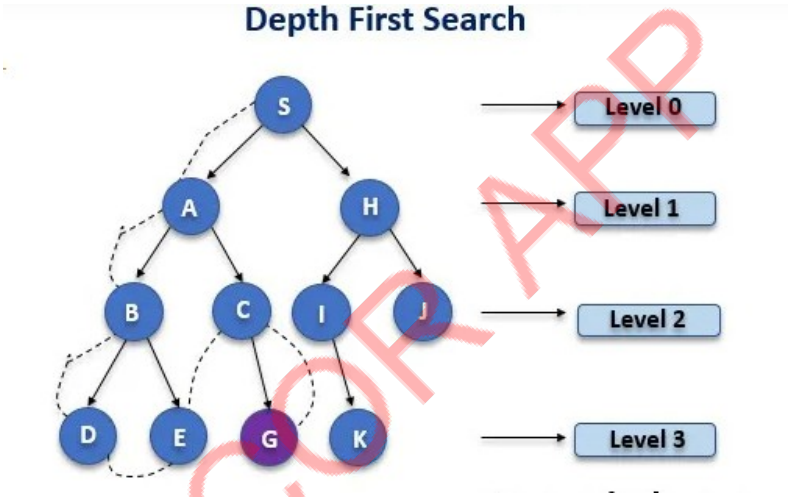
- It occupies a lot of memory space, and time to execute when the solution is at the bottom or end of the tree and uses the FIFO queue.
- Time Complexity of BFS is expressed as T (n) = $1+n^2+n^3+\ldots\ldots+ n^d= O (n^d)$ and;
- Space Complexity of BFS is O (n^d).
- The breadth-first search algorithm is complete.
- The optimal solution is possible to obtain from BFS.



➢ Depth First Search Algorithms

DFS is one of the recursive algorithms we know. It traverses the graph or a tree depth-wise. Thus it is known to be a depth-first search algorithm as it derives its name from the way it functions. The DFS uses the stack for its implementation. The process of search is similar to BFS. The only difference lies in the expansion of nodes which is depth-wise in this case.

- Unlike the BFS, the DFS requires very less space in the memory because of the way it stores the nodes stack only on the path it explores depth-wise.
- In comparison to BFS, the execution time is also less if the expansion of nodes is correct. If the path is not correct, then the recursion continues, and there is no guarantee that one may find the solution. This may result in an infinite loop formation.
- The DFS is complete only with finite state space.
- Time Complexity is expressed as $T(n) = 1+ n^2+ n^3+\ldots\ldots\ldots+ n^m=O(n^m)$.
- The Space Complexity is expressed as O (bm).
- The DFS search algorithm is not optimal, and it may generate large steps and possibly high cost to find the solution.

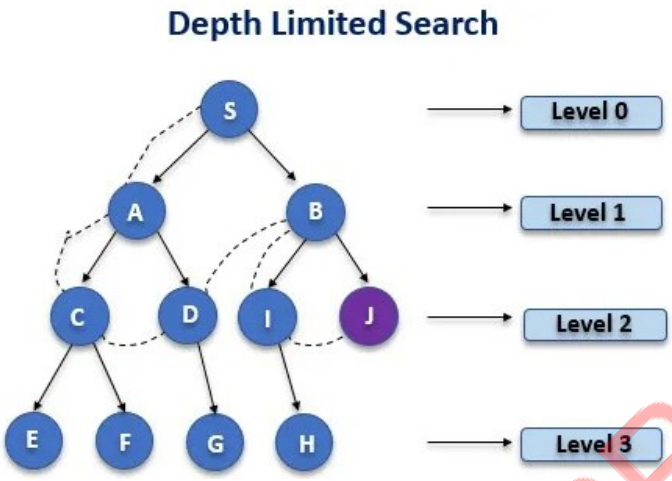


➢ **Depth Limited Search Algorithm**

The DLS algorithm is one of the uninformed strategies. A depth limited search is close to DFS to some extent. It can find the solution to the demerit of DFS. The nodes at the depth may behave as if no successor exists at the depth. Depth-limited search can be halted in two cases:

  - SFV: The Standard failure value which tells that there is no solution to the problem.
  - CFV: The Cutoff failure value tells that there is no solution within the given depth.

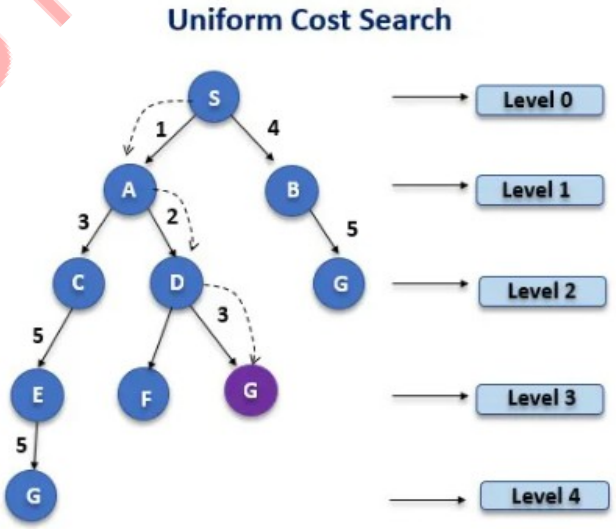
- The DLS is efficient in memory space utilization.
- Time Complexity is expressed as $O(b^{\ell})$.
- Space Complexity is expressed as $O(b\times\ell)$.
- It has the demerit of incompleteness. It is complete only if the solution is above the depth limit.



➢ **Uniform-cost Search Algorithm**

The UCS algorithm is used for visiting the weighted tree. The main goal of the uniform cost search is to fetch a goal node and find the true path, including the cumulative cost. The following are the properties of the UCS algorithm:

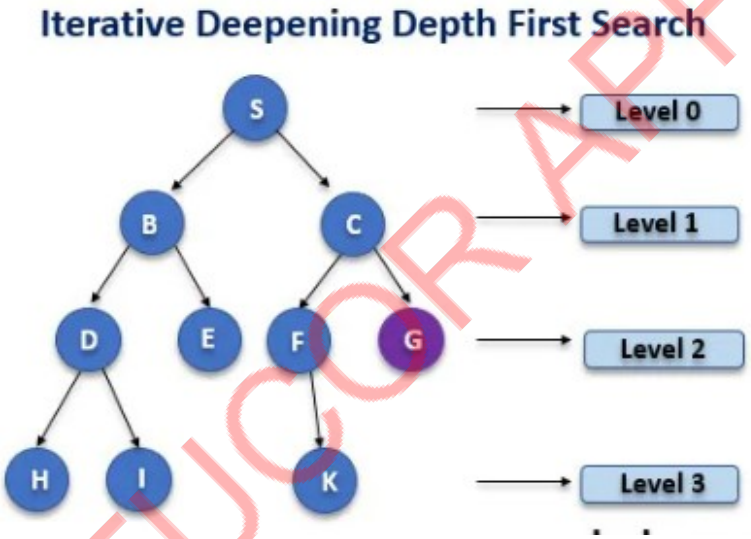
- The expansion takes place on the basis of cost from the root. The UCS is implemented using a priority queue.
- The UCS does not care for the number of steps, and so it may end up an infinite loop.
- The uniform-cost search algorithm is known to be complete.
- Time Complexity can be expressed as $O(b^{1 + [C^*/\varepsilon]})$/
- Space Complexity is expressed as $O(b^{1 + [C^*/\varepsilon]})$.
- We can say that UCs is the optimal algorithm as it chooses the path with the lowest cost only.



- **Iterative deepening depth-first Search**

This algorithm is a combination of BFS and DFS searching techniques. It is iterative in nature. The best depth is found using it. The algorithm is set to search only at a certain depth. The depth keeps increasing at each recursive step until it finds the goal node.

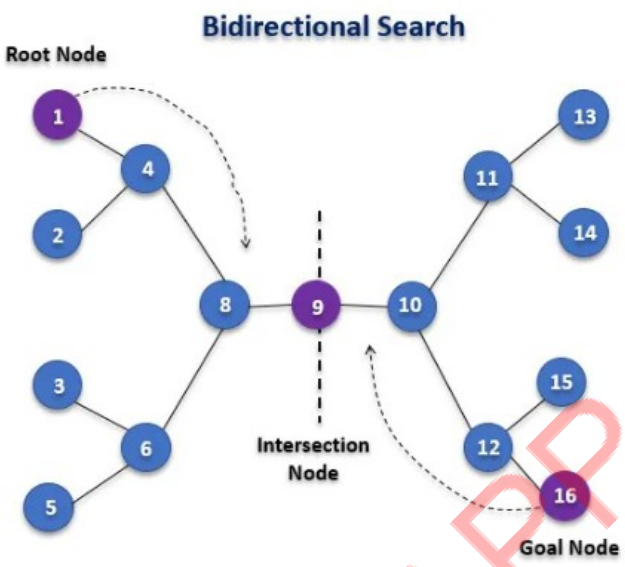
- The power of BFS and DFS combination is observed in this algorithm.
- When the search space is large, it proves itself, and the depth is not known.
- This algorithm has one demerit, and it is that it iterates all the previous steps.
- The algorithm is known to be complete only if the branching factor is known r finite.
- Time Complexity is expressed as $O(b^d)$.
- Space Complexity is expressed as O(bd).
- This algorithm is optimal.



- **Bidirectional Search Algorithm**

The Two way or Bidirectional search algorithm executes in a way that t has to run two searches simultaneously one in a forward direction and the other in the backward direction. The search will stop when the two simultaneous searches intersect each other to find the goal node. It is free to use any search algorithm discussed above, like BFS, DFS, etc.

- Bidirectional search is quick and occupies less memory.
- The implementation is difficult, and the goal node should be known in advance to execute it.
- The Bidirectional Search algorithm is found to be complete and optimal.
- Time Complexity is expressed as $O(b^d)$.
- Space Complexity is expressed as $O(b^d)$.



5. What is game and applications of game theory? (May-03)

Game:

The term game means a sort of conflict in which n individuals or groups participate

Game theory denotes games of strategy. Games are integral attribute of human beings. Games engage the intellectual faculties of humans. If computers are to mimic people they should be able to play games.

Applications of Game Theory

The following are just a few examples of game theory applications:

- Stock trades and the investors' reactions and decisions against stock market developments and the behaviors and decisions of other investors
- OPEC member countries' decision to change the amount of oil extraction and sale and their compliance or non-compliance with quota arrangements
- Corporate behavior regarding product pricing in monopoly or multilateral competition markets
- Animal interaction with one another in social life (hunting or sharing achievements or supporting each other)

6. Explain the Mini-Max algorithm and how it is work for game tic-tac-toe. (dec-03,04, May -09,10, 17 , 19)

➢ 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.

Pseudo-code for MinMax Algorithm:

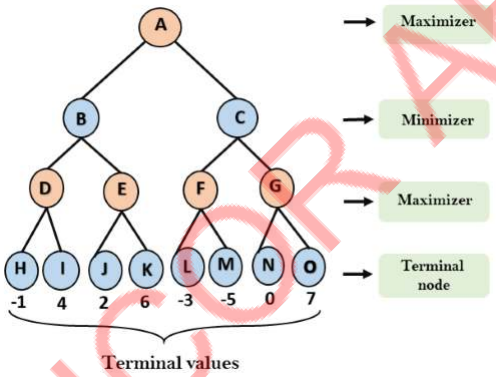
```
function minimax(node, depth, maximizingPlayer) is
if depth ==0 or node is a terminal node then
return static evaluation of node

if MaximizingPlayer then      // for Maximizer Player
maxEva= -infinity
 for each child of node do
 eva= minimax(child, depth-1, false)
maxEva= max(maxEva,eva)        //gives Maximum of the values
return maxEva

else                          // for Minimizer player
 minEva= +infinity
 for each child of node do
 eva= minimax(child, depth-1, true)
 minEva= min(minEva, eva)         //gives minimum of the values
 return minEva
```

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.

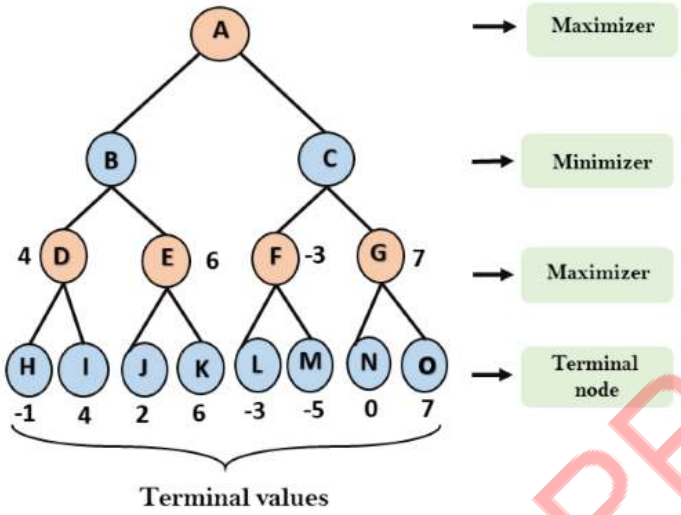
- In this example, there are two players one is called Maximizer and other is called Minimizer.
- Maximizer will try to get the Maximum possible score, and Minimizer will try to get the minimum possible score.
- This algorithm applies DFS, so in this game-tree, we have to go all the way through the leaves 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 applies 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.



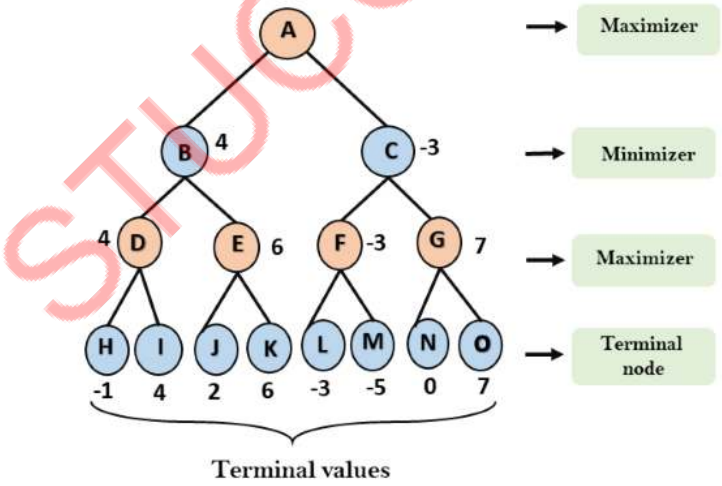
Step 2: Now, first we find the utilities value for the Maximizer, its initial value is -∞, so we will compare each value in terminal state with initial value of Maximizer and determines the higher nodes values. It will find the maximum among the all.

- For node D max(-1,- -∞) => max(-1,4)= 4
- For Node E max(2, -∞) => max(2, 6)= 6
- For Node F max(-3, -∞) => max(-3,-5) = -3
- For node G max(0, -∞) = max(0, 7) = 7



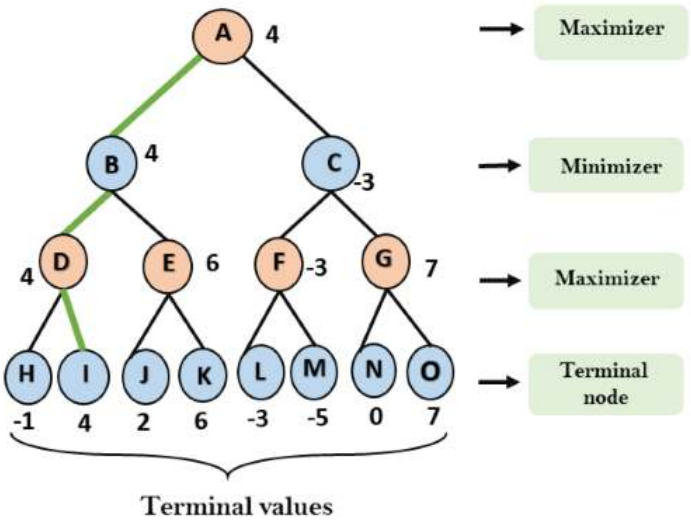
Step 3: In the next step, it's a turn for minimizer, so it will compare all nodes value with +∞, and will find the 3rd layer node values.

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



Step 4: 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



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.**

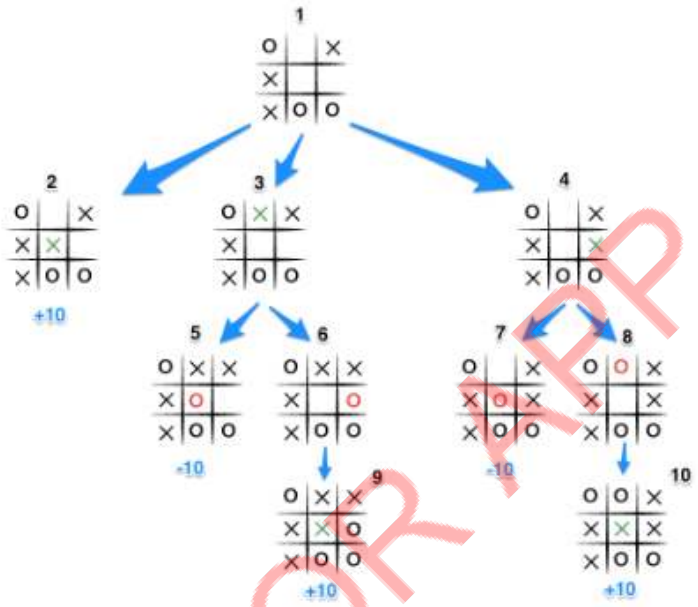
Describing Minimax

The key to the Minimax algorithm is a back and forth between the two players, where the player whose "turn it is" desires to pick the move with the maximum score. In turn, the scores for each of the available moves are determined by the opposing player deciding which of its available moves has the minimum score. And the scores for the opposing players moves are again determined by the turn-taking player trying to maximize its score and so on all the way down the move tree to an end state.

A description for the algorithm, assuming X is the "turn taking player," would look something like:

- If the game is over, return the score from X's perspective.
- Otherwise get a list of new game states for every possible move
- Create a scores list
- For each of these states add the minimax result of that state to the scores list
- If it's X's turn, return the maximum score from the scores list
- If it's O's turn, return the minimum score from the scores list

You'll notice that this algorithm is recursive; it flips back and forth between the players until a final score is found. Let's walk through the algorithm's execution with the full move tree, and show why, algorithmically, the instant winning move will be picked:



- It's X's turn in state 1. X generates the states 2, 3, and 4 and calls minimax on those states.
- State 2 pushes the score of +10 to state 1's score list, because the game is in an end state.
- State 3 and 4 are not in end states, so 3 generates states 5 and 6 and calls minimax on them, while state 4 generates states 7 and 8 and calls minimax on them.
- State 5 pushes a score of -10 onto state 3's score list, while the same happens for state 7 which pushes a score of -10 onto state 4's score list.
- State 6 and 8 generate the only available moves, which are end states, and so both of them add the score of +10 to the move lists of states 3 and 4.
- Because it is O's turn in both state 3 and 4, O will seek to find the minimum score, and given the choice between -10 and +10, both states 3 and 4 will yield -10.
- Finally the score list for states 2, 3, and 4 are populated with +10, -10 and -10 respectively, and state 1 seeking to maximize the score will chose the winning move with score +10, state 2.

That is certainly a lot to take in. And that is why we have a computer execute this algorithm.

7. Explain Alpha-Beta Pruning using example. (Dec—04,10, May-10,17)

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 -∞.
    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 +∞.

- 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.

Condition for Alpha-beta pruning:

The main condition which required for alpha-beta pruning is: **α>=β**

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 α= -∞ and β= +∞, these value of alpha and beta passed down to node B where again α= -∞ and β= +∞, 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 algorithms backtrack to node B, where the value of β will change as this is a turn of Min, Now β= +∞ will compare with the available subsequent nodes value, i.e. min (∞, 3) = 3, hence at node B now α= -∞, and β= 3. In the next step, algorithm traverse the next successor of Node B which is node E, and the values of α= -∞, and β= 3 will also be passed.

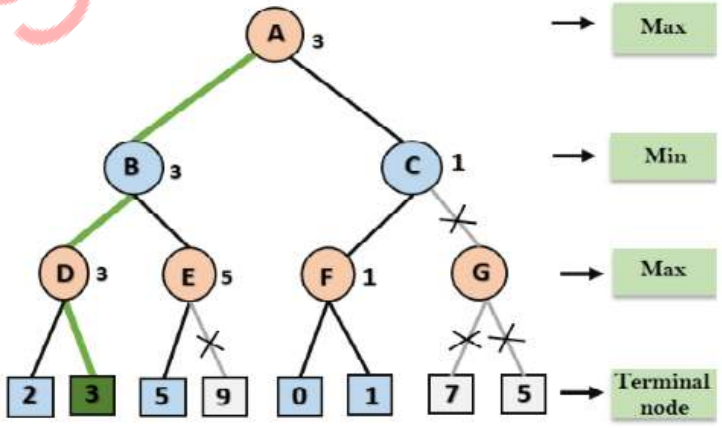
Step 4: At node E, Max will take its turn, and the value of alpha will change. The current value of alpha will be compared with 5, so max (-∞, 5) = 5, hence at node E α= 5 and β= 3, where α>=β, so the right successor of E will be pruned, and algorithm will not traverse it, and the value at node E will 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 (-∞, 3)= 3, and β= +∞, these two values now passes to right successor of A which is Node C.
At node C, α=3 and β= +∞, 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 α= 3 and β= +∞, here the value of beta will be changed, it will compare with 1 so min (∞, 1) = 1. Now at C, α=3 and β= 1, and again it satisfies the condition α>=β, 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 the showing the nodes which are computed and nodes which has never computed. Hence the optimal value for the maximizer is 3 for this example.



Move Ordering in Alpha-Beta pruning:

The effectiveness of alpha-beta pruning is highly dependent on the order in which each node is examined. Move order is an important aspect of alpha-beta pruning. It can be of two types:

- **Worst ordering:** In some cases, alpha-beta pruning algorithm does not prune any of the leaves of the tree, and works exactly as minimax algorithm. In this case, it also consumes more time because of alpha-beta factors, such a move of pruning is called worst ordering. In this case, the best move occurs on the right side of the tree. The time complexity for such an order is $O(b^m)$.
- **Ideal ordering:** The ideal ordering for alpha-beta pruning occurs when lots of pruning happens in the tree, and best moves occur at the left side of the tree. We apply DFS hence it first search left of the tree and go deep twice as minimax algorithm in the same amount of time. Complexity in ideal ordering is $O(b^{m/2})$.

8. Explain heuristic search with an example. Explain A* search and give the proof of optimality of A*.

Informed Search Algorithms

- 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:

$$h(n) <= h^*(n)$$

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 unit 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**

✓ **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.

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:

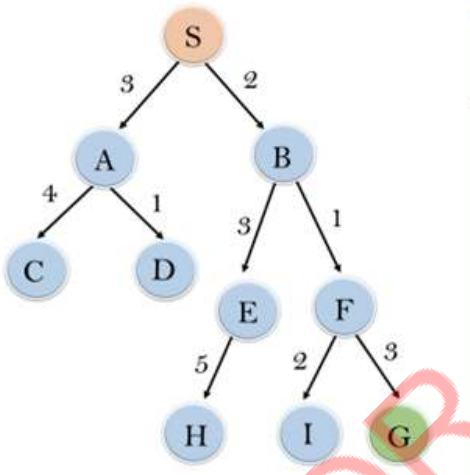
- Best first search can switch between BFS and DFS by gaining the advantages of both the algorithms.
- This algorithm is more efficient than BFS and DFS algorithms.

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.

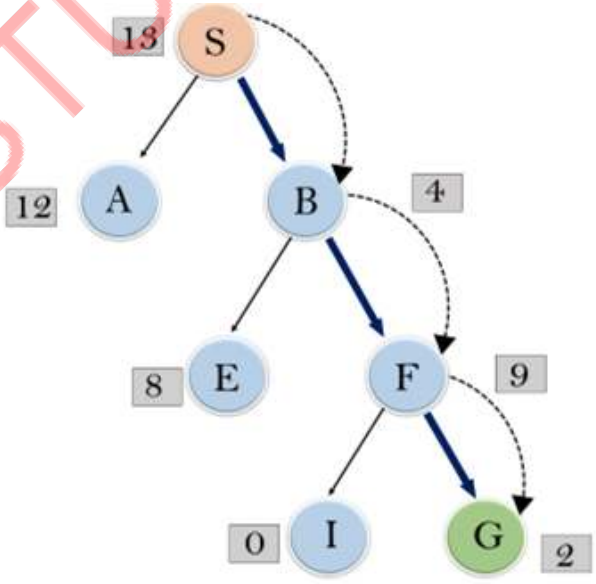
Example:

- 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]

Iteration2: 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]

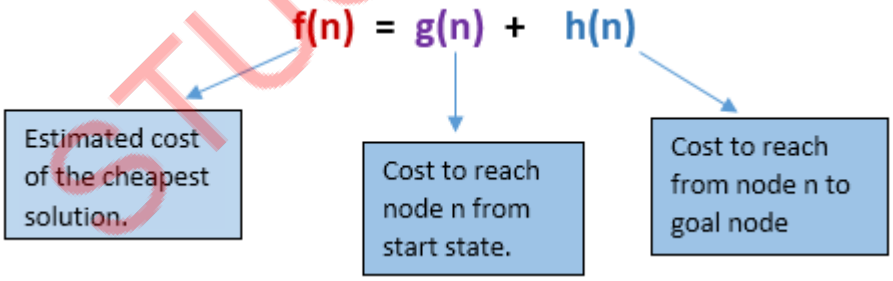
Hence the final solution path will be: **S----> B----->F----> G**

Time 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**.



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

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.

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)<="" li="">

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)**

9. Explain the approaches for solving tree structured constraints satisfaction problem with suitable examples.

Constraint Satisfaction Problems in Artificial Intelligence

This section examines the constraint optimization methodology, another form or real concern method. By its name, constraints fulfillment implies that such an issue must be solved while adhering to a set of restrictions or guidelines. Whenever a problem is actually variables comply with stringent conditions of principles, it is said to have been addressed using the solving multi - objective method. Wow what a method results in a study sought to achieve of the intricacy and organization of both the issue.

Three factors affect restriction compliance, particularly regarding

- It refers to a group of parameters, or X.
- D: The variables are contained within a collection several domain. Every variables has a distinct scope.
- C: It is a set of restrictions that the collection of parameters must abide by.

In constraint satisfaction, domains are the areas wherein parameters were located after the restrictions that are particular to the task. Those three components make up a constraint satisfaction technique in its entirety. The pair "scope, rel" makes up the number of something like the requirement. The scope is a tuple of variables that contribute to the restriction, as well as rel is indeed a relationship that contains a list of possible solutions for the parameters should assume in order to meet the restrictions of something like the issue.

Issues with Contains a certain amount Solved

For a constraint satisfaction problem (CSP), the following conditions must be met:

- States area
- Fundamental idea while behind remedy.

The definition of a state in phase space involves giving values to any or all of the parameters, like as X1 = v1, X2 = v2, etc.

There are 3 methods to economically beneficial to something like a parameter:

1. Consistent or Legal Assignment: A task is referred to as consistent or legal if it complies with all laws and regulations.
2. Complete Assignment: An assignment in which each variable has a number associated to it and that the CSP solution is continuous. One such task is referred to as a completed task.
3. A partial assignment is one that just gives some of the variables values. Projects of this nature are referred to as incomplete assignment.

Domain Categories within CSP

The parameters utilize one of the two types of domains listed below:

- Discrete Domain: This limitless area allows for the existence of a single state with numerous variables. For instance, every parameter may receive a endless number of beginning states.
- It is a finite domain with continuous phases that really can describe just one area for just one particular variable. Another name for it is constant area.

Types of Constraints in CSP

Basically, there are three different categories of limitations in regard towards the parameters:

- Unary restrictions are the easiest kind of restrictions because they only limit the value of one variable.
- Binary resource limits: These restrictions connect two parameters. A value between x1 and x3 can be found in a variable named x2.
- Global Resource limits: This kind of restriction includes a unrestricted amount of variables.

The main kinds of restrictions are resolved using certain kinds of resolution methodologies:

- In linear programming, when every parameter carrying an integer value only occurs in linear equation, linear constraints are frequently utilized.
- Non-linear Constraints: With non-linear programming, when each variable (an integer value) exists in a non-linear form, several types of restrictions were utilized.

UNIT – 2 (8 Marks and 16 Marks)

1. How to handle uncertain knowledge with example? And How to represent knowledge in an uncertain domain? (Dec- 2013) and Define uncertain knowledge, prior probability and conditional probability. State the Baye's theorem. How it is useful for decision making under uncertainty? (May- 2014)

PROBABILISTIC REASONING

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→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**

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.

$0 \leq P(A) \leq 1$, where P(A) is the probability of an event A.

P(A) = 0, indicates total uncertainty in an event A.

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***/* **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⋀*B*) by P(B)**.

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. Baye's theorem was named after the British mathematician **Thomas Bayes**. The **Bayesian inference** is an application of Baye's 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:

$$P(A \wedge B) = P(A|B)\, P(B) \text{ or}$$

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

$$P(A \wedge B) = P(B|A)\, P(A)$$

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

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

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 evidence.

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

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

Where A_1, A_2, A_3,....., 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:

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

Application 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.

2. Discuss about Bayesian theory and Bayesian network. (Dec 2017)

Bayesian network

- "A Bayesian network is a probabilistic graphical model which represents a set of variables and their conditional dependencies using a directed acyclic graph."
- It is also called a **Bayes network, belief network, decision network**, or **Bayesian model**.
- Bayesian networks are probabilistic, because these networks are built from a **probability distribution**, and also use probability theory for prediction and anomaly detection.

Real world applications are probabilistic in nature, and to represent the relationship between multiple events, we need a Bayesian network. It can also be used in various tasks including **prediction, anomaly detection, diagnostics, automated insight, reasoning, time series prediction**, and **decision making under uncertainty**.

➢ Bayesian Network can be used for building models from data and experts opinions, and it consists of two parts:

- **Directed Acyclic Graph**
- **Table of conditional probabilities.**

The generalized form of Bayesian network that represents and solve decision problems under uncertain knowledge is known as an Influence diagram.

A Bayesian network graph is made up of nodes and Arcs (directed links), where:

- Each node corresponds to the random variables, and a variable can be continuous or discrete.
- Arc or directed arrows represent the causal relationship or conditional probabilities between random variables. These directed links or arrows connect the pair of nodes in the graph. These links represent that one node directly influence the other node, and if there is no directed link that means that nodes are independent with each other
  - In the above diagram, A, B, C, and D are random variables represented by the nodes of the network graph.
  - If we are considering node B, which is connected with node A by a directed arrow, then node A is called the parent of Node B.
  - Node C is independent of node A.

The Bayesian network has mainly two components:

- **Causal Component**
- **Actual numbers**

Each node in the Bayesian network has condition probability distribution $\mathbf{P(X_i \mid Parent(X_i))}$, which determines the effect of the parent on that node. Bayesian network is based on Joint probability distribution and conditional probability. So let's first understand the joint probability distribution:

Joint probability distribution:

If we have variables x1, x2, x3,....., xn, then the probabilities of a different combination of x1, x2, x3.. xn, are known as Joint probability distribution.

$\mathbf{P[x_1, x_2, x_3,....., x_n]}$, it can be written as the following way in terms of the joint probability distribution.

$\mathbf{= P[x_1 \mid x_2, x_3,....., x_n]P[x_2, x_3,....., x_n]}$

$\mathbf{= P[x_1 \mid x_2, x_3,....., x_n]P[x_2 \mid x_3,....., x_n]....P[x_{n-1} \mid x_n]P[x_n].}$

In general for each variable Xi, we can write the equation as:

$$\mathbf{P(X_i \mid X_{i-1},....., X_1) = P(X_i \mid Parents(X_i))}$$

3. Explain about Dempster shafer theory. (May- 2017)

Dempster – Shafer Theory (DST)

- DST is a mathematical **theory of evidence** based on belief functions and plausible reasoning. It is used to combine separate pieces of information (evidence) to calculate the probability of an event.

- DST offers an alternative to traditional probabilistic theory for the mathematical representation of uncertainty.
- DST can be regarded as, a more general approach to represent uncertainty than the Bayesian approach. Bayesian methods are sometimes inappropriate

Example:

Let **A** represent the proposition **"Moore is attractive"**. Then the axioms of probability insist that **P(A) + P(¬A) = 1**. Now suppose that Andrew does not even know who **"Moore"** is, then Also, it is not fair to say that he disbelieves the proposition. It would therefore be meaningful to denote Andrew's belief **B** of **B(A)** and **B(¬A)** as both being **0**.

Dempster-Shafer Model

The idea is to allocate a number between 0 and 1 to indicate a degree of belief on a proposal as in the probability framework. However, it is not considered a probability but a belief mass. The distribution of masses is called basic belief assignment.

In other words, in this formalism a degree of belief (referred as mass) is represented as a belief function rather than a Bayesian probability distribution.

Example: Belief assignment

Suppose a system has five members, say five independent states, and exactly one of which is actual. If the original set is called **S, | *S* | = 5**, then the set of all subsets (the power set) is called $\mathbf{2^S}$. If each possible subset as a binary vector (describing any member is present or not by writing **1** or **0**), then $\mathbf{2^5}$ subsets are possible, ranging from the empty subset **(0, 0, 0, 0, 0)** to the "everything" subset **(1, 1, 1, 1, 1)**.

The "empty" subset represents a "contradiction", which is not true in any state, and is thus assigned a mass of **one**; The remaining masses are normalized so that their total is **1**. The "everything" subset is labeled as "unknown"; it represents the state where all elements are present **one** , in the sense that you cannot tell which is actual.

Belief and Plausibility

Shafer's framework allows for belief about propositions to be represented as intervals, bounded by two values, belief (or support) and plausibility:

belief ≤ plausibility

Belief in a hypothesis is constituted by the sum of the masses of all sets enclosed by it (i.e. the sum of the masses of all subsets of the hypothesis). It is the amount of belief that directly supports a given hypothesis at least in part, forming a lower bound.

Plausibility is 1 minus the sum of the masses of all sets whose intersection with the hypothesis is empty. It is an upper bound on the possibility that the hypothesis could possibly happen, up to that value, because there is only so much evidence that contradicts that hypothesis.

Example:

A proposition say **"the cat in the box is dead."** Suppose we have **belief of 0.5** and **plausibility of 0.8** for the proposition.

For example,

Suppose we have a belief of 0.5 for a proposition, say "the cat in the box is dead." This means that we have evidence that allows us to state strongly that the proposition is true with a confidence of 0.5. However, the evidence contrary to that hypothesis (i.e. "the cat is alive") only has a confidence of 0.2. The remaining mass of 0.3 (the gap between the 0.5 supporting evidence on the one hand, and the 0.2 contrary evidence on the other) is "indeterminate," meaning that the cat could either be dead or alive. This interval represents the level of uncertainty based on the evidence in the system.

Hypothesis	Mass	Belief	Plausibility
Neither (alive nor dead)	0	0	0
Alive	0.2	0.2	0.5
Dead	0.5	0.5	0.8
Either (alive or dead)	0.3	1.0	1.0

- The "neither" hypothesis is set to zero by definition (it corresponds to "no solution"). The orthogonal hypotheses "Alive" and "Dead" have probabilities of 0.2 and 0.5, respectively. This could correspond to "Live/Dead Cat Detector" signals, which have respective reliabilities of 0.2 and 0.5.

- Finally, the all-encompassing "Either" hypothesis (which simply acknowledges there is a cat in the box) picks up the slack so that the sum of the masses is 1. The belief for the "Alive" and "Dead" hypotheses matches their corresponding masses because they have no subsets; belief for "Either" consists of the sum of all three masses (Either, Alive, and Dead) because "Alive" and "Dead" are each subsets of "Either".

- The "Alive" plausibility is $1 - m$ (Dead): 0.5 and the "Dead" plausibility is $1 - m$ (Alive): 0.8. In other way, the "Alive" plausibility is m(Alive) + m(Either) and the "Dead" plausibility is m(Dead) + m(Either).

- Finally, the "Either" plausibility sums m(Alive) + m(Dead) + m(Either). The universal hypothesis ("Either") will always have 100% belief and plausibility—it acts as a checksum of sorts.

Plausibility in K: It is the sum of masses of set that intersects with K. i.e; Pl(K) = m(a) + m(b) + m(c) + m(a, b) + m(b, c) + m(a, c) + m(a, b, c)

Characteristics of Dempster Shafer Theory:

- It will ignorance part such that probability of all events aggregate to 1.
- Ignorance is reduced in this theory by adding more and more evidences.
- Combination rule is used to combine various types of possibilities.

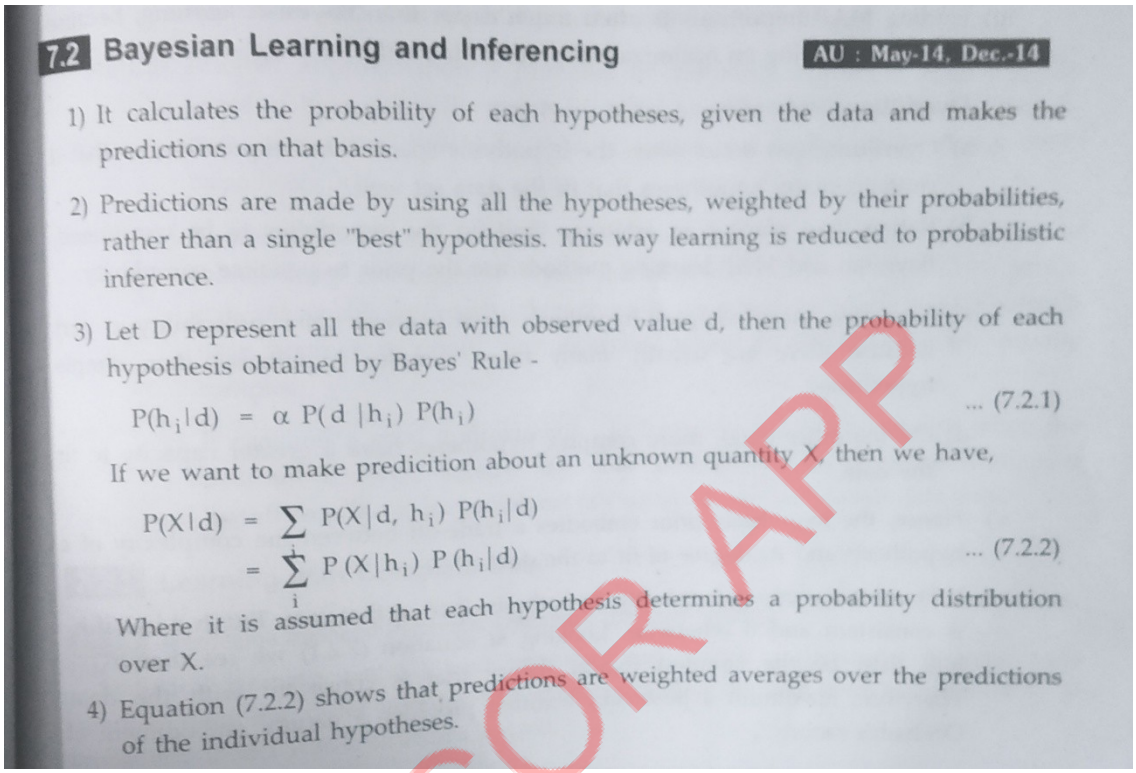
Advantages:

- As we add more information, uncertainty interval reduces.
- DST has much lower level of ignorance.
- Diagnose hierarchies can be represented using this.
- Person dealing with such problems is free to think about evidences.

Disadvantages:

- In this, computation effort is high, as we have to deal with 2^n of sets.

4. Explain about the exact inference in Bayesian networks. (May- 2015)



7.2 Bayesian Learning and Inferencing AU : May-14, Dec.-14

1) It calculates the probability of each hypotheses, given the data and makes the predictions on that basis.

2) Predictions are made by using all the hypotheses, weighted by their probabilities, rather than a single "best" hypothesis. This way learning is reduced to probabilistic inference.

3) Let D represent all the data with observed value d, then the probability of each hypothesis obtained by Bayes' Rule -

$$P(h_i|d) = \alpha\, P(d|h_i)\, P(h_i) \quad \ldots (7.2.1)$$

If we want to make predicition about an unknown quantity X, then we have,

$$P(X|d) = \sum_i P(X|d, h_i)\, P(h_i|d)$$

$$= \sum_i P(X|h_i)\, P(h_i|d) \quad \ldots (7.2.2)$$

Where it is assumed that each hypothesis determines a probability distribution over X.

4) Equation (7.2.2) shows that predictions are weighted averages over the predictions of the individual hypotheses.

5) The important quantities in Bayesian learning are the hypothesis prior - $P(h_i)$ and the likelihood of the data under each hypothesis - $P(d|h_i)$.

6) The basic characteristic of Baysian learning is that "True hypothesis dominates the Bayesian prediction".

7) For any fixed prior that does not rule out the true hypothesis, the posterior probability of any false hypothesis will eventually vanish, simply because the probability of generating "uncharacteristic" data indefinitely is vanishingly small.

8) The Bayesian prediction is optimal whether the data set be small or large.

9) For real learning problems, the hypothesis space is usually very large or infinite.

10) Approximation in Bayesian learning : -

i) A prediction can be made on the basis of single most probable hypothesis, h_i, that maximizes $P(h_i|d)$. This is called as maximum a posteriori or MAP hypothesis.

ii) Predictions made according to an MAP hypothesis h_{map} are approximately Bayesian to the extent that $P(X|d) \approx P(X|h_{map})$.

iii) Finding MAP hypothesis is often much easier than Bayesian learning, because it requires solving an optimization problem instead of a large summation.

iv) Overfitting Trade-offs :

a) Overfitting can occur when the hypothesis space is too expressive, so that it contains many hypotheses that fit the data set well.

b) Rather than placing an arbitrary limit on the hypotheses to be considered, Bayesian and MAP learning methods use the prior to penalize complexity.

c) Typically, more complex hypothesis have a lower prior probability-in part because there are usually many more complex hypotheses than simple hypotheses.

d) On the other hand, more complex hypotheses have a greater capacity to fit the data.

v) Hence, the hypothesis prior embodies a trade-off between the complexity of a hypothesis and its degree of fit to the data.

vi) If H contains only deterministic hypothesis, then in that case, $P(d|h_i)$ is 1 if h_i is consistent and 0 otherwise. Looking at equation (7.2.1) we see that h_{MAP} will then be the simplest logical theory that is consistent with the data. Therefore, maximum a posteriori learning provides a natural embodiment of Ockham's razor.

vii) a) Another trade-off between complexity and degree of fit is obtained by taking the logarithm of equation (7.2.1).

b) Choosing h_{MAP} to maximize $P(d|h_i)\ P(h_i)$ is equivalent to minimizing

$$-\log_2 P(d|h_i) - \log_2 P(h_i)$$

c) Using the

vii) a) Another trade-off between complexity and degree of fit is obtained by taking the logarithm of equation (7.2.1).

b) Choosing h_{MAP} to maximize $P(d|h_i)\ P(h_i)$ is equivalent to minimizing

$$-\log_2 P(d|h_i) - \log_2 P(h_i)$$

c) Using the connection between information encoding and probability we see that the $-\log_2 P(h_i)$ term equals the number of bits required to specify the hypothesis h_i.

d) $\log_2 P(d/h_i)$ is the additional number of bits required to specify the data given the hypothesis.

e) To see this, consider that no bits are required if the hypothesis predicts the data exactly as with h_5 and the string of lime candies and $\log_2 1 = 0$.

f) MAP learning is choosing the hypothesis that provides maximum compression of the data.

g) The same task is addressed more directly by the minimum description length or MDL, learning method, which attempts to minimize the size of hypothesis and data encodings rather than work with probabilities.

viii) a) Another approximation is provided by assuming a uniform prior over the space of hypotheses. In that case, MAP learning reduces to choosing an h_i that maximizes $P(d|H_i)$. This is called a **maximum-likelihood (ML)** hypothesis, h_{ML}.

b) Maximum-likelihood learning is very common in statistics, a discipline in which many researchers destruct the subjective nature of hypothesis priors. It is reasonable approach when there is no reason to prefer one hypothesis over another a priori. For example, when all hypotheses are equally complex.

c) It provides a good approximation to Bayesian and MAP learning when the data set is large, because the data swamps the prior distribution over hypotheses, but it has problems (all we shall see) with small data sets.

7.2.1 Learning with Complete Data

7.2.1.1 Maximum-Likelihood Parameter Learning : (Discrete Models)

Statistical learning methods have important task which is parameter learning with complete data. Parameter learning task involves finding the numerical parameters for a

Data is said to be complete when each data points contains values for every variable in the mode.

Consider candy-bag example : -

New manufacturer : Then the lime/Cherry proportions is completely unknown.

Parameter : $\theta \in [0, 1]$ (proportion of cherry)

Hypothesis : h_θ

Assumption : All proportions equally likely a priori.

BN's variables : Flavour $\in$ {Cherry, Lime}

N unwrapped candies : C cherries and $l = N - C$ limes.

Likelihood of this particular data set :

$$P(d \mid h_\theta) = \prod_{j=1}^{N} P(d_j \mid h_\theta) = \theta^c (1 - \theta)^l$$

- Finding the maximum-likelihood hypothesis h_{ML} is then equivalent to maximising the log-likelihood :

$$L(d \mid h_\theta) = \log P(d \mid h_\theta)$$

$$= \sum_{j=1}^{N} \log P(d_j \mid h_\theta) = c \log\theta + l \log(1 - \theta)$$

To that end : 1) differentiate L with respect to θ and

2) set the resulting expression to 0.

$$\frac{dL(d \mid h_\theta)}{d\theta} = \frac{c}{\theta} - \frac{l}{1 - \theta} = 0 \Rightarrow \theta = \frac{c}{c + l} = \frac{c}{N}$$

- Previous result is obvious, but the process is important : -
  1) Write down the expression for the likelihood of the data as a function of the parameter(s).
  2) Write down the derivative of the log-likelihood with respect to each parameter.
  3) Find the parameter values such that the derivatives are zero.

The last step can be tricky, using iterative solution algorithms or numerical optimisation.

- **Problem with ML learning :**

If some events have never been observed, h_{ML} assigns them 0 probability.

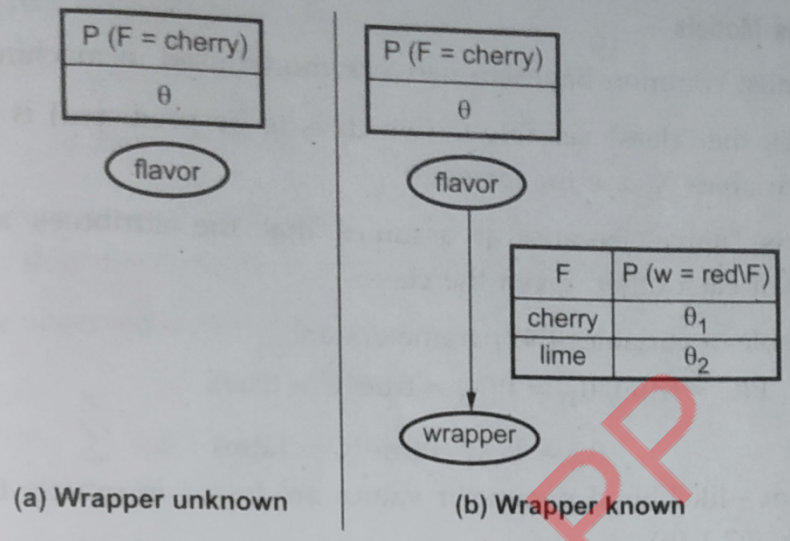


Fig. 7.2.1 Problem in ML Learning

New pb(b) : Wrappers red or blue. Colors selected probabilistically (see new BN)

$P(\text{Flavor} = \text{Cherry}, \text{Wrapper} = \text{green} \mid h_{\theta,\theta_1,\theta_2})$

$= P(\text{Flavor} = \text{cherry} \mid h_{\theta,\theta_1,\theta_2}) \; P(\text{Wrapper} = \text{green} \mid \text{Flavor} = \text{Cherry}, h_{\theta,\theta_1,\theta_2})$

$= \theta \cdot (1 - \theta_1)$

Experiment :

$$N = c + l = (r_c + g_c) + (r_l + g_l)$$

Likelihood :

$$P(d \mid h_{\theta,\theta_1,\theta_2}) = \theta^c (1-\theta)^l \cdot \theta_1^{r_c} (1-\theta_1)^{g_c} \cdot \theta_2^{r_l} (1-\theta_2)^{g_l}$$

log-likelihood :

$$L = [c \log\theta + l \log(1-\theta)] + [r_c \log\theta_1 + g_c \log(1-\theta_1)] + [r_l \log\theta_2 + g_l \log(1-\theta_2)]$$

Derivatives :

$$\frac{dL}{d\theta} = \frac{c}{\theta} - \frac{l}{1-\theta} = 0 \qquad \Rightarrow \theta = \frac{c}{c+l}$$

$$\frac{dL}{d\theta_1} = \frac{r_c}{\theta_1} - \frac{g_c}{1-\theta_1} = 0 \qquad \Rightarrow \theta = \frac{r_c}{r_c + g_c}$$

$$\frac{dL}{d\theta_2} = \frac{r_1}{\theta_2} - \frac{g_l}{1-\theta_2} = 0 \qquad \Rightarrow \theta = \frac{r_l}{r_l + g_l}$$

With complete data, ML parameter learning for a BN decomposes into separate

7.2.1.2 Naive Bayes Models

1) This is the most common Bayesian network model used in machine learning.
2) In this model, the "class" variable C (which is to be predicted) is the root and the "attribute" variables X_i are the leaves.
3) The model is "naive" because it assumes that the attributes are conditionally independent of each other, given the class.
4) Assuming Boolean variables the parameters are,

$$\theta = P(C = \text{true}),\ \theta_{i1} = P(X_i = \text{true} \mid C = \text{true}),$$

$$\theta_{i2} = P(X_i = \text{true} \mid C = \text{false})$$

The maximum - likelihood parameter values are found in exactly the same way as shown in Fig. 7.2.1 (b).

5) Once the model has been trained in this way, it can be used to classify new examples for which the class variable C is unobserved. With observed attribute values, $x_1, x_2, \ldots x_n$, the probability of each class is given by,

$$P(C \mid x_1, x_2, \ldots x_n) = \alpha\ P(C) \prod_i P(x_i \mid C)$$

6) A deterministic prediction can be obtained by choosing the most likely class.
7) The method learns fairly well but not as well as decision-tree learning; this is presumably because the true hypothesis - which is a decision tree - is not representable exactly using a naive Bayes model.
8) Naive Bayes learning do well in a wide range of applications; the boosted version is one of the most effective general-purpose learning algorithm.
9) Naive Bayes learning scales well to very large problems : with n Boolean attributes, there are just 2n+1 parameters, and no search is required to find h_{ML}, the maximum - likelihood naive Bayes hypothesis.
10) Naive Bayes learning has no difficulty with noisy data and can give probabilistic predictions when appropriate.

7.2.1.3 Maximum-Likelihood Parameter Learning : (Continuous Models)

1) Continuous probability model such as the linear-Gaussian model is used for maximum - likelihood parameter learning.
2) Because continuous variables are ubiquitous in real-world applications, it is important to know how to learn continuous models from data.
3) The principles for maximum likelihood learning are identical to those of the discrete case.
4) a) Let us begin with a very simple case : Learning the parameters of a Gaussian

b) That is, the data are generated as follows : -

$$P(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

The parameters of this model are the mean μ and the standard deviation σ. (Notice that the normalizing "constant" depends on σ, so we cannot ignore it.)

c) Let the observed values be $x_1, \ldots, x_N$. Then the log likelihood is,

$$L = \sum_{j=1}^{N} \log \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(x_j-\mu)^2}{2\sigma^2}}$$

$$= N(-\log\sqrt{2\pi} - \log\sigma) - \sum_{j=1}^{N} \frac{(x_j-\mu)^2}{2\sigma^2}$$

d) Setting the derivatives to zero as usual, we obtain,

$$\frac{\delta L}{\delta \mu} = -\frac{1}{\sigma^2} \sum_{j=1}^{N} (x_j - \mu) = 0 \qquad \Rightarrow \mu = \frac{\sum_j x_j}{N}$$

$$\frac{\delta L}{\delta \sigma} = -\frac{N}{\sigma} + \frac{1}{\sigma^3} \sum_{j=1}^{N} (x_j - \mu)^2 = 0 \qquad \Rightarrow \sigma = \sqrt{\frac{\sum_j (x_j-\mu)^2}{N}}$$

5) The maximum-likelihood value of the mean is the sample average and the maximum-likelihood value of the standard deviation is the square root of the sample variance. Again, these are comforting results that confirm "commonsense" practice.

6) a) Now consider a linear Gaussian model with one continuous parent X and a continuous child Y.

b) Y has a Gaussian distribution whose mean depends linearly on the value of X and whose standard deviation is fixed.

c) To learn the conditional distribution P(Y|X), we can maximize the conditional likelihood.

$$P(y|x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(y-(\theta_1 x+\theta_2))^2}{2\sigma^2}}$$

d) Here, the parameters are θ_1, θ_2 and σ. The data are a collection of (x_j, y_j) pairs, as shown in Fig. 7.2.4.

find the maximum-likelihood values of the

7.2.1.4 Bayesian Parameter Learning

ML learning is simple, but not appropriate for small data sets.

Example -

If only cherries have been observed, $h_{ML} \rightarrow \theta = 1.0$

- Bayesian Approach :
  - It uses hypothesis prior over possible values of the parameters.
  - Update of this distribution is used as the data arrives.
- Candy example with Bayesian view : -
  - θ : unknown value of a variable Θ.
  - Hypothesis prior : $P(\Theta)$. (Continuous over [0, 1] and integrating to 1).

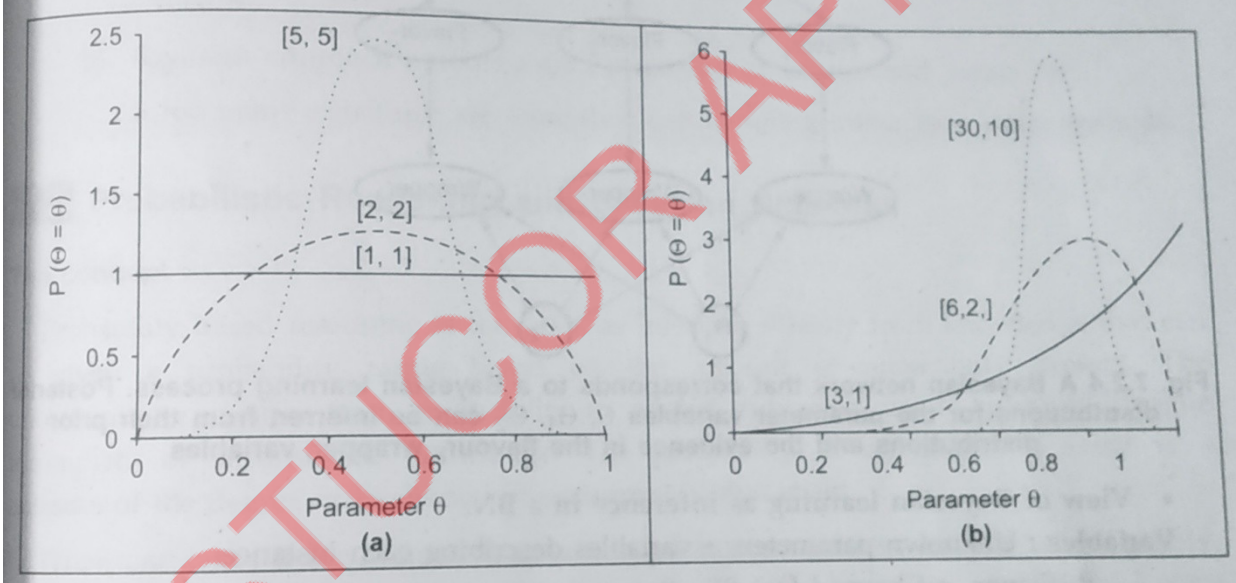


Fig. 7.2.3 (a) and (b) Examples of the beta [a, b] distribution for different values of [a, b]

- Candidates :
  beta distributions, defined by 2 hyperparameters a and b

  such that :

  $$\text{beta}[a, b](\theta) = \alpha \theta^{a-1} (1-\theta)^{b-1}$$

- Nice property of the beta family :

If Θ has prior beta [a, b] and a data point is observed, then the posterior for Θ is also a beta distribution.

Beta family :

A beta family is called as the **conjugate prior** for the family of distributions for a Boolean variable.

$$P(\theta \mid D_1 = \text{Cherry}) = \alpha\, P(D_1 = \text{Cherry} \mid \theta)\, P(\theta)$$
$$= \alpha' \theta \cdot \text{beta}\,[a, b]\,(\theta) = \alpha'\, \theta \cdot \theta^{a-1} (1-\theta)^{b-1}$$
$$= \alpha'\, \theta^{a} (1-\theta)^{b-1} = \text{beta}\,[a+1, b]\,(\theta)$$

Note : a and b are virtual counts (starting with beta [1, 1]).

With wrappers : 3 parameters. It need to specify $P(\theta, \theta_1, \theta_2)$.

Assuming parameter independence : $P(\theta, \theta_1, \theta_2) = P(\theta)\, P(\theta_1)\, P(\theta_2)$

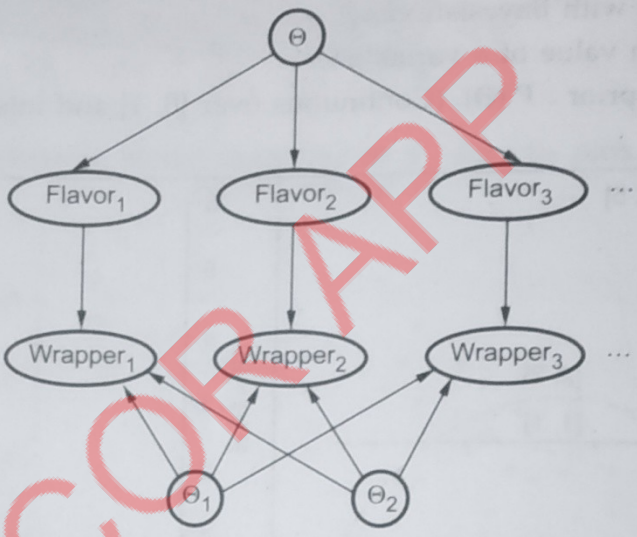


Fig. 7.2.4 A Bayesian network that corresponds to a Bayesian learning process. Posterior distributions for the parameter variables Θ, Θ_1, Θ_2 can be inferred from their prior distributions and the evidence in the flavour_i, wrapper_i variables

- **View of Bayesian learning as inference in a BN.**

Variables : Unknown parameters + variables describing each instance.

$P(\text{Flavor}_i = \text{Cherry} \mid \Theta = \theta) = \theta$

$P(\text{Wrapper}_i = \text{red} \mid \text{Flavor}_i = \text{Cherry}, \Theta_1 = \theta_1) = \theta_1$

7.2.1.5 Learning Bayes Net Structures

Often the Bayes net structures are easy to get from expert knowledge. But sometimes the causality relationships are debatable.

For example :

Smoking ⇒ Cancer ?

too Much TV ⇒ Bad at school ?

- To search for a good model :
  - Start with a linkless model and add parents to each node, then learn parameters and measure accuracy of the resulting model.
  - Start with an initial guess of the structure, and use hill-climbing or simulated

UNIT – 3 (8 Marks and 16 Marks)

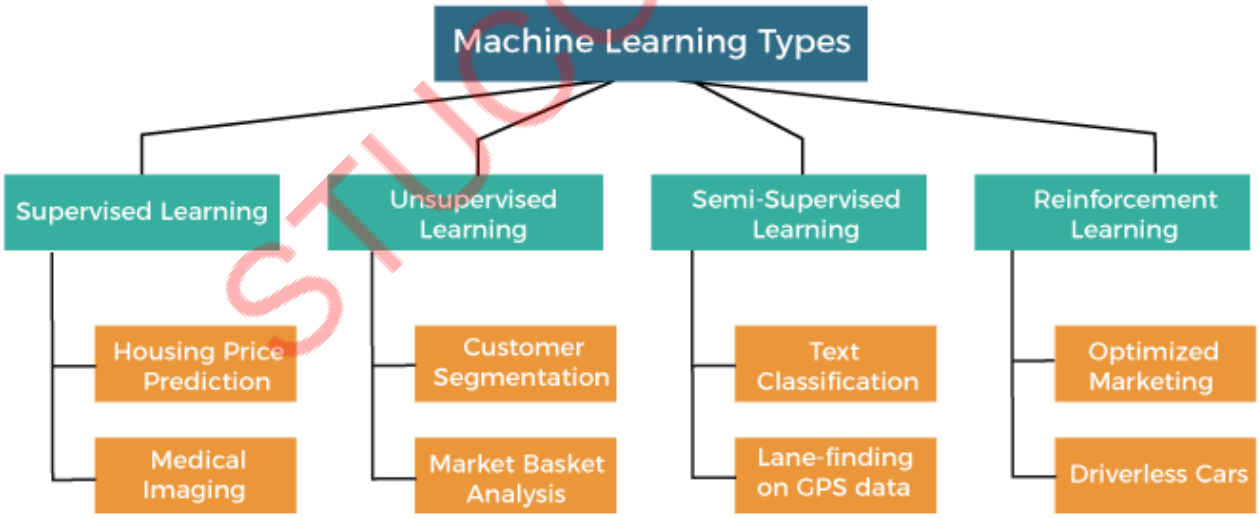
1. Explain the types of learning in machine learning.

Types of Machine Learning

Machine learning is a subset of AI, which enables the machine to automatically learn from data, improve performance from past experiences, and make predictions**.** Machine learning contains a set of algorithms that work on a huge amount of data. Data is fed to these algorithms to train them, and on the basis of training, they build the model & perform a specific task.

These ML algorithms help to solve different business problems like Regression, Classification, Forecasting, Clustering, and Associations, etc.
Based on the methods and way of learning, machine learning is divided into mainly four types, which are:

1. Supervised Machine Learning
2. Unsupervised Machine Learning
3. Semi-Supervised Machine Learning
4. Reinforcement Learning



Supervised Machine Learning

Supervised machine learning is based on supervision. It means in the supervised learning technique, we train the machines using the "labelled" dataset, and based on the training, the machine predicts the output. Here, the labelled data specifies that some of the inputs are already mapped to the output. More preciously, we can say; first, we train the machine with the input and corresponding output, and then we ask the machine to predict the output using the test dataset.

Example: Suppose we have an input dataset of cats and dog images. So, first, we will provide the training to the machine to understand the images, such as the shape & size of the tail of cat and dog, Shape of eyes, colour, height (dogs are taller, cats are smaller), etc. After completion of training, we input the picture of a cat and ask the machine to identify the object and predict the output. Now, the machine is well trained, so it will check all the features of the object, such as height, shape, color, eyes, ears, tail, etc., and find that it's a cat. So, it will put it in the Cat category. This is the process of how the machine identifies the objects in Supervised Learning.

The main goal of the supervised learning technique is to map the input variable(x) with the output variable(y). Some real-world applications of supervised learning are Risk Assessment, Fraud Detection, Spam filtering, etc.

Supervised machine learning can be classified into two types of problems, which are given below:

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

a) Classification

Classification algorithms are used to solve the classification problems in which the output variable is categorical, such as "Yes" or No, Male or Female, Red or Blue, etc. The classification algorithms predict the categories present in the dataset. Some real-world examples of classification algorithms are Spam Detection, Email filtering, etc.

Some popular classification algorithms are given below:

- Random Forest Algorithm
- Decision Tree Algorithm
- Logistic Regression Algorithm
- Support Vector Machine Algorithm

b) Regression

Regression algorithms are used to solve regression problems in which there is a linear relationship between input and output variables. These are used to predict continuous output variables, such as market trends, weather prediction, etc.

Some popular Regression algorithms are given below:

- **Simple Linear Regression Algorithm**
- **Multivariate Regression Algorithm**
- **Decision Tree Algorithm**
- **Lasso Regression**

Advantages:

- Since supervised learning work with the labelled dataset so we can have an exact idea about the classes of objects.
- These algorithms are helpful in predicting the output on the basis of prior experience.

Disadvantages:

- These algorithms are not able to solve complex tasks.
- It may predict the wrong output if the test data is different from the training data.
- It requires lots of computational time to train the algorithm.

Applications of Supervised Learning

Some common applications of Supervised Learning are given below:

- **ImageSegmentation**
- **MedicalDiagnosis**
- **Fraud Detection**
- **Spam detection Speech Recognition**

2. Unsupervised Machine Learning

Unsupervised learning is different from the Supervised learning technique; as its name suggests, there is no need for supervision. It means, in unsupervised machine learning, the machine is trained using the unlabeled dataset, and the machine predicts the output without any supervision. In unsupervised learning, the models are trained with the data that is neither classified nor labelled, and the model acts on that data without any supervision.

Unsupervised Learning can be further classified into two types, which are given below:

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

Advantages:

- These algorithms can be used for complicated tasks compared to the supervised ones because these algorithms work on the unlabeled dataset.
- Unsupervised algorithms are preferable for various tasks as getting the unlabeled dataset is easier as compared to the labelled dataset.

Disadvantages:

- The output of an unsupervised algorithm can be less accurate as the dataset is not labelled, and algorithms are not trained with the exact output in prior.
- Working with Unsupervised learning is more difficult as it works with the unlabelled dataset that does not map with the output.

Applications of Unsupervised Learning

Network Analysis
Recommendation Systems

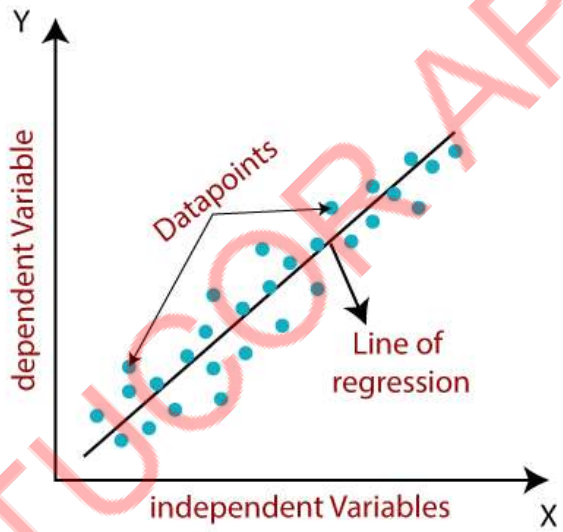
2. Explain in details about regression models- linear regression models?

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 (y) variables, hence called as linear regression. Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.

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



$$y= a_0+a_1x+ \varepsilon$$

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

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

Types of Linear Regression

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

- **SimpleLinearRegression:**

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.

- **MultipleLinearregression:**

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:

- **PositiveLinearRelationship:**
  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.

- **NegativeLinearRelationship:**
  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.

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:

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

Where,
N=Total number of observation
Yi = Actual value
$(a1x_i+a_0)$= Predicted value.

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

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**.

3. Explain in details about regression models- linear classification models.

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:

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:

$$y= b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

- 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 for y= 0, and infinity for y=1

- But we need range between -[infinity] to +[infinity], then take logarithm of the equation it will become:

$$log\left[\frac{y}{1-y}\right] = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

The above equation is the final equation for Logistic Regression.

Type of Logistic Regression:

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".

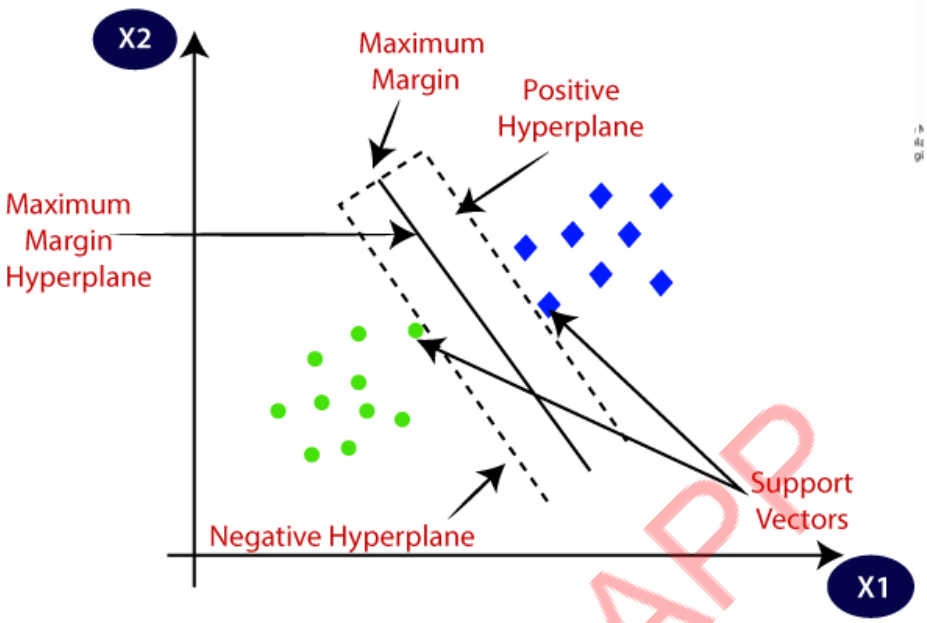
4. Explain in detail about support vector machine?

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.

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.

Support Vectors:

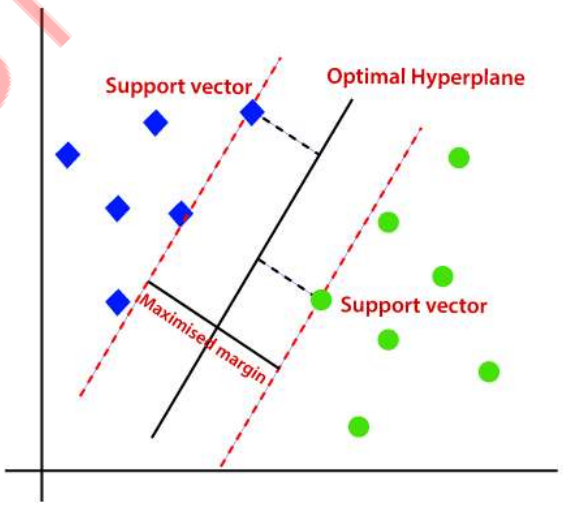
The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. Since these vectors support the hyperplane, hence called a Support vector.

How does SVM works?

Linear SVM:

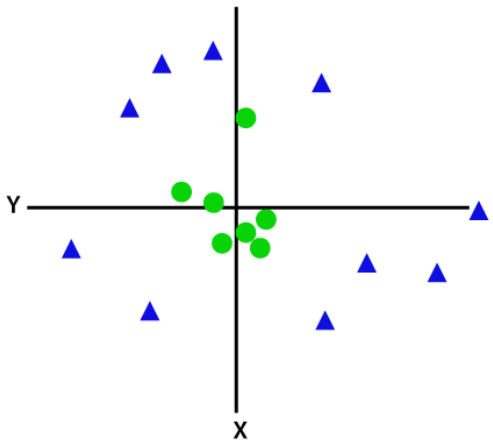
The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x1 and x2. We want a classifier that can classify the pair(x1, x2) of coordinates in either green or blue. Consider the below image:

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:

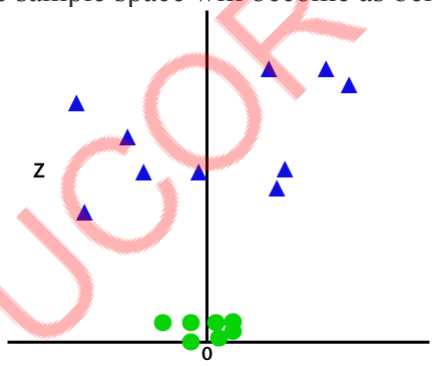
If data is linearly arranged, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line. Consider the below image:



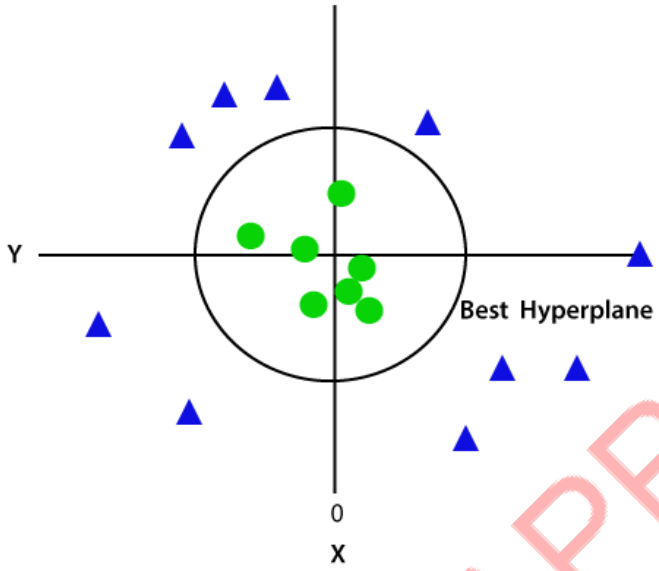
So to separate these data points, we need to add one more dimension. For linear data, we have used two dimensions x and y, so for non-linear data, we will add a third dimension z. It can be calculated as:

$$z=x^2+y^2$$

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. 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:



5. Explain in detail about decision tree and random forest?

Decision Tree Classification Algorithm

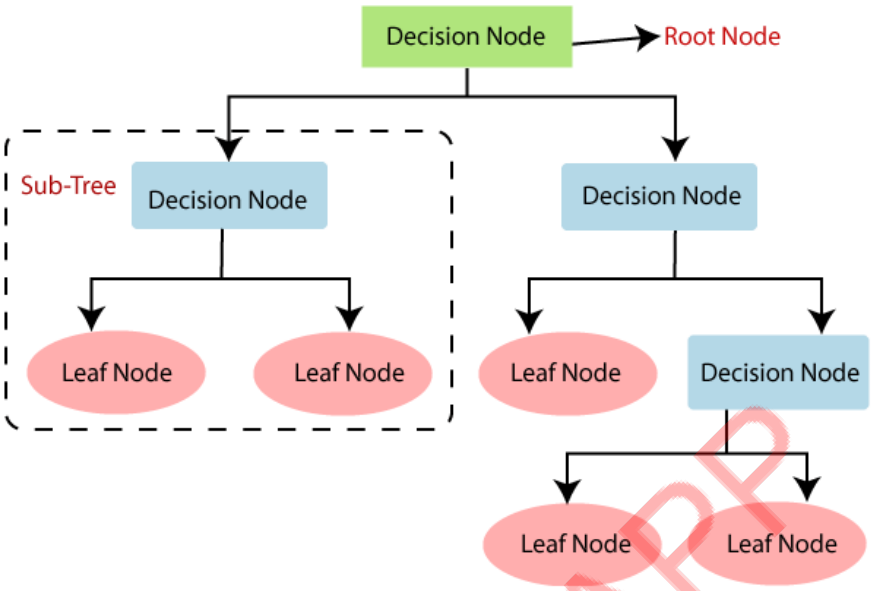
- Decision Tree is a **Supervised learning technique** that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules **and** each leaf node represents the outcome.
- In a Decision tree, there are two nodes, which are

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:



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:

Advantages of the Decision Tree

- 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.

UNIT – 4 (8 Marks and 16 Marks)

1.Explain the various ensemble learning techniques?

Ensemble methods are techniques that aim at improving the accuracy of results in models by combining multiple models instead of using a single model. The combined models increase the accuracy of the results significantly. This has boosted the popularity of ensemble methods in machine learning.

Categories of Ensemble Methods

Ensemble methods fall into two broad categories, i.e., sequential ensemble techniques and parallel ensemble techniques. **Sequential ensemble techniques** generate base learners in a sequence, e.g., Adaptive Boosting (AdaBoost). The sequential generation of base learners promotes the dependence between the base learners. The performance of the model is then improved by assigning higher weights to previously misrepresented learners.

- In **parallel ensemble techniques**, base learners are generated in a parallel format, e.g., random forest. Parallel methods utilize the parallel generation of base learners to encourage independence between the base learners. The independence of base learners significantly reduces the error due to the application of averages.

- The majority of ensemble techniques apply a single algorithm in base learning, which results in homogeneity in all base learners. Homogenous base learners refer to base learners of the same type, with similar qualities. Other methods apply heterogeneous base learners, giving rise to heterogeneous ensembles. Heterogeneous base learners are learners of distinct types.

Main Types of Ensemble Methods

1. Bagging

- Bagging, the short form for bootstrap aggregating, is mainly applied in classification and regression. It increases the accuracy of models through decision trees, which reduces variance to a large extent. The reduction of variance increases accuracy, eliminating overfitting, which is a challenge to many predictive models.
- Bagging is classified into two types, i.e., bootstrapping and aggregation. **Bootstrapping** is a sampling technique where samples are derived from the whole population (set) using the replacement procedure. The sampling with replacement method helps make the selection procedure randomized. The base learning algorithm is run on the samples to complete the procedure.
- **Aggregation** in bagging is done to incorporate all possible outcomes of the prediction and randomize the outcome. Without aggregation, predictions will not be accurate because all outcomes are not put into consideration. Therefore, the aggregation is based on the probability bootstrapping procedures or on the basis of all outcomes of the predictive models.

Bagging is advantageous since weak base learners are combined to form a single strong learner that is more stable than single learners. It also eliminates any variance, thereby reducing the overfitting of models. One limitation of bagging is that it is computationally expensive. Thus, it can lead to more bias in models when the proper procedure of bagging is ignored.

2. Boosting

- Boosting is an ensemble technique that learns from previous predictor mistakes to make better predictions in the future. The technique combines several weak base learners to form one strong learner, thus significantly improving the predictability of models. Boosting works by arranging weak learners in a sequence, such that weak learners learn from the next learner in the sequence to create better predictive models.
- Boosting takes many forms, including gradient boosting, Adaptive Boosting (AdaBoost), and XGBoost (Extreme Gradient Boosting). AdaBoost uses weak learners in the form of decision trees, which mostly include one split that is popularly known as decision stumps. AdaBoost's main decision stump comprises observations carrying similar weights.

- Gradient boosting adds predictors sequentially to the ensemble, where preceding predictors correct their successors, thereby increasing the model's accuracy. New predictors are fit to

counter the effects of errors in the previous predictors. The gradient of descent helps the gradient booster identify problems in learners' predictions and counter them accordingly.

3. Stacking

Stacking, another ensemble method is often referred to as stacked generalization. This technique works by allowing a training algorithm to ensemble several other similar learning algorithm predictions. Stacking has been successfully implemented in regression, density estimations, distance learning, and classifications. It can also be used to measure the error rate involved during bagging.

Variance Reduction

Ensemble methods are ideal for reducing the variance in models, thereby increasing the accuracy of predictions. The variance is eliminated when multiple models are combined to form a single prediction that is chosen from all other possible predictions from the combined models. An ensemble of models combines various models to ensure that the resulting prediction is the best possible, based on the consideration of all predictions.

Simple Ensemble Techniques

In this section, we will look at a few simple but powerful techniques, namely:

1. Max Voting
2. Averaging
3. Weighted Averaging

2. Explain in detail about k-means algorithm?

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.

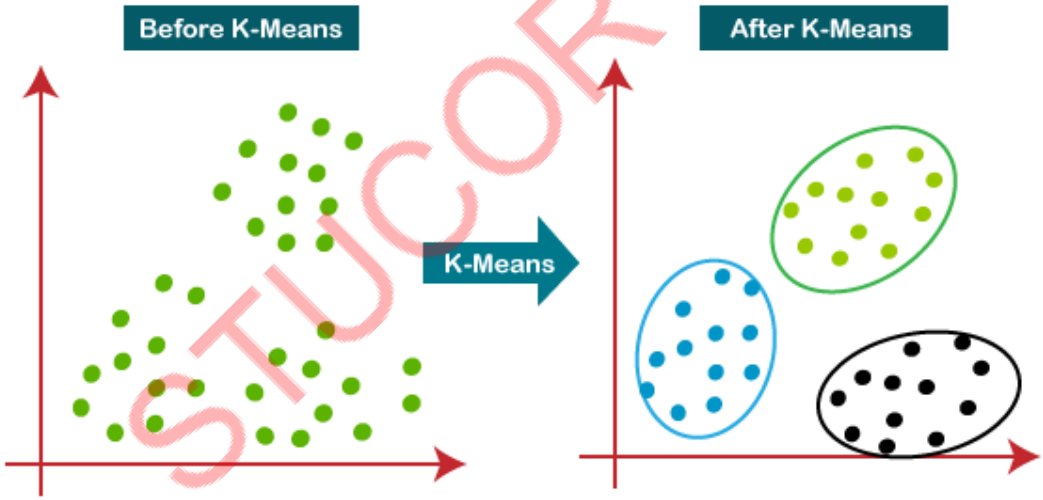
- It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

- 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.

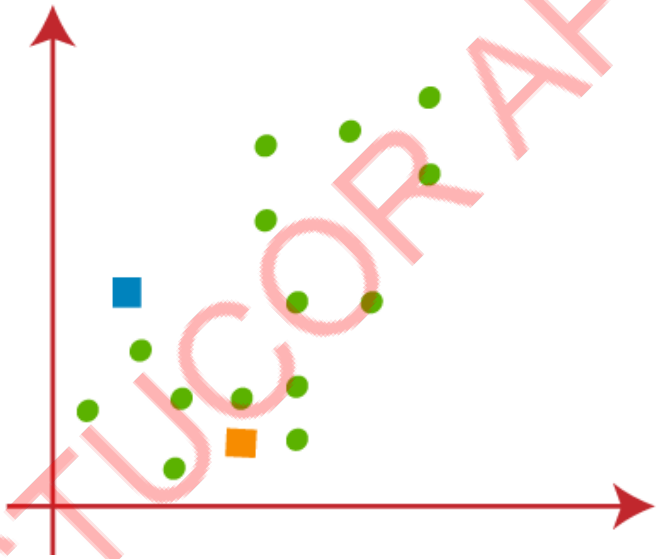
Step-5: Repeat the third steps, which mean reassign each datapoint to the new closest centroid of each cluster.

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

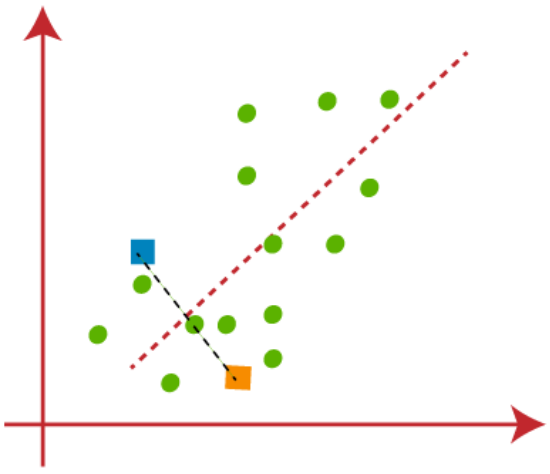
Step-7: The model is ready.

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.

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:

3. Explain details about KNN algorithm?

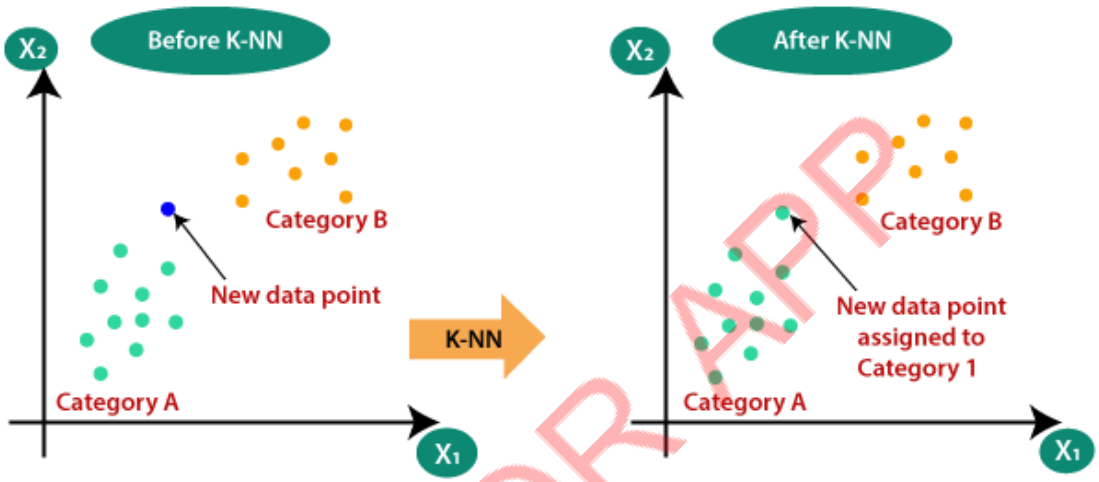
K-Nearest Neighbor(KNN) Algorithm for Machine Learning

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.
- It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

- **Example:** Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.

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 x1, 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:

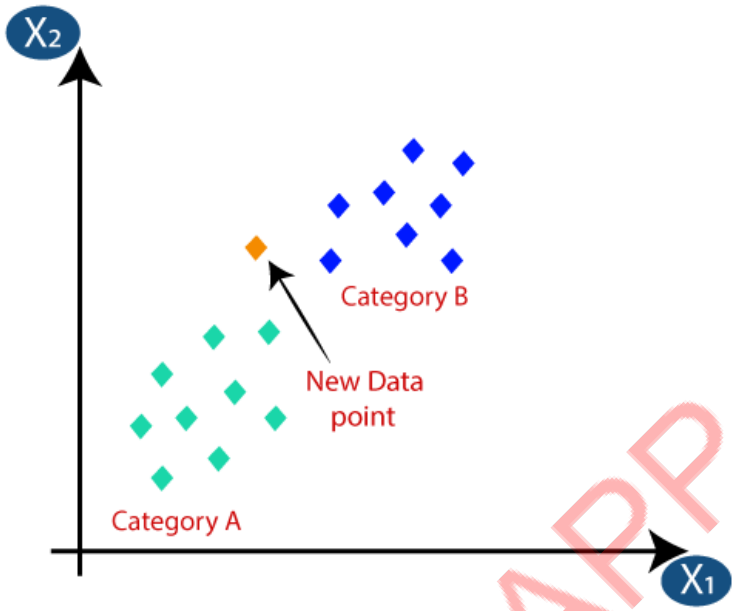


How does K-NN work?

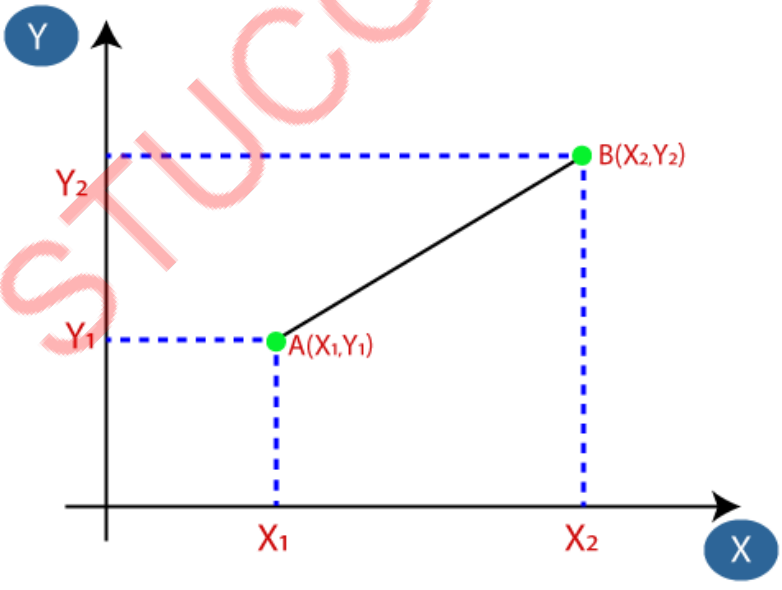
The K-NN working can be explained on the basis of the below algorithm:

- **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:

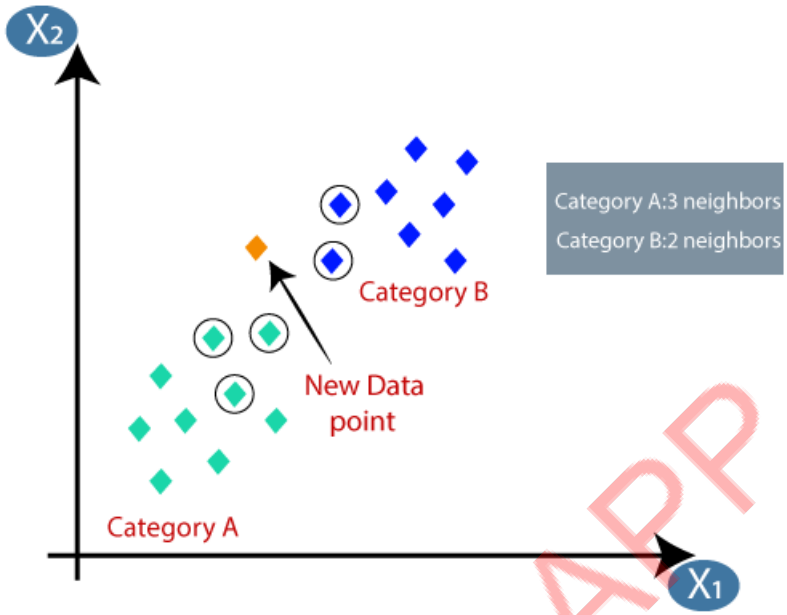


- Firstly, we will choose the number of neighbors, so we will choose the k=5.
- Next, we will calculate the **Euclidean distance** between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:



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.

4. Explain in detail about Gaussian mixture models and expectation maximization?

EM algorithm in GMM

In statistics, EM (expectation maximization) algorithm handles latent variables, while GMM is the Gaussian mixture model.

- ✓ Gaussian mixture models (GMMs) are a type of machine learning algorithm. They are used to classify data into different categories based on the probability distribution. Gaussian mixture models can be used in many different areas, including finance, marketing and so much more.

- ✓ Gaussian Mixture Models (GMMs) give us more flexibility than K-Means. With GMMs we assume that the data points are Gaussian distributed; this is a less restrictive assumption than saying they are circular by using the mean. That way, we have two parameters to describe the shape of the clusters: the mean and the standard deviation!

- ✓ Taking an example in two dimensions, this means that the clusters can take any kind of elliptical shape (since we have standard deviation in both the x and y directions). Thus, each Gaussian distribution is assigned to a single cluster. In order to find the parameters of the Gaussian for each cluster (e.g the mean and standard deviation) we will use an optimization algorithm called Expectation–Maximization (EM). Take a look at the graphic below as an illustration of the Gaussians being fitted to the clusters. Then we can proceed on to the process of Expectation–Maximization clustering using GMMs.

- ✓ Gaussian mixture models (GMM) are a probabilistic concept used to model real-world data sets. GMMs are a generalization of Gaussian distributions and can be used to represent any data set that can be clustered into multiple Gaussian distributions. The Gaussian mixture model is a probabilistic model that assumes all the data points are generated from a mix of Gaussian distributions with unknown parameters.

- ✓ A Gaussian mixture model can be used for clustering, which is the task of grouping a set of data points into clusters. GMMs can be used to find clusters in data sets where the clusters may not be clearly defined. Additionally, GMMs can be used to estimate the probability that a new data point belongs to each cluster. Gaussian mixture models are also relatively robust to outliers, meaning that they can still yield accurate results even if there are some data points that do not fit neatly into any of the clusters. This makes GMMs a flexible and powerful tool for clustering data.

- ✓ It can be understood as a probabilistic model where Gaussian distributions are assumed for each group and they have means and co variances which define their parameters. GMM consists of two parts – mean vectors (μ) & covariance matrices (Σ). A Gaussian distribution is defined as a continuous probability distribution that takes on a bell-shaped curve. Another

name for Gaussian distribution is the normal distribution. Here is a picture of Gaussian mixture models:

- ✓ GMM has many applications, such as density estimation, clustering, and image segmentation. For density estimation, GMM can be used to estimate the probability density function of a set of data points. For clustering, GMM can be used to group together data points that come from the same Gaussian distribution. And for image segmentation, GMM can be used to partition an image into different regions.

- ✓ Gaussian mixture models can be used for a variety of use cases, including identifying customer segments, detecting fraudulent activity, and clustering images. In each of these examples, the Gaussian mixture model is able to identify clusters in the data that may not be immediately obvious. As a result, Gaussian mixture models are a powerful tool for data analysis and should be considered for any clustering task.

Expectation-maximization (EM) method in relation to GMM

In Gaussian mixture models, an expectation-maximization method is a powerful tool for estimating the parameters of a Gaussian mixture model (GMM). The expectation is termed E and maximization is termed M. Expectation is used to find the Gaussian parameters which are used to represent each component of gaussian mixture models. Maximization is termed M and it is involved in determining whether new data points can be added or not.

- ✓ The expectation-maximization method is a two-step iterative algorithm that alternates between performing an expectation step, in which we compute expectations for each data point using current parameter estimates and then maximize these to produce a new gaussian, followed by a maximization step where we update our gaussian means based on the maximum likelihood estimate.

- ✓ The EM method works by first initializing the parameters of the GMM, then iteratively improving these estimates. At each iteration, the expectation step calculates the expectation of the log-likelihood function with respect to the current parameters. This expectation is then used to maximize the likelihood in the maximization step. The process is then repeated until convergence. Here is a picture representing the two-step iterative aspect of the algorithm

The EM algorithm consists of two steps: the E- step and the M-step. Firstly, the model parameters and the can be randomly initialized. In the E-step, the algorithm tries to guess the value of based on the parameters, while in the M-step, the algorithm updates the value of the model parameters based on the guess of the E-step. These two steps are repeated until convergence is reached. The algorithm in GMM is repeat until convergence.

Optimization uses the Expectation Maximization algorithm, which alternates between two steps:

1. E-step: Compute the posterior probability over z given our current model - i.e. how much do we think each Gaussian generates each datapoint.

2. M-step: Assuming that the data really was generated this way, change the parameters of each Gaussian to maximize the probability that it would generate the data it is currently responsible for.

The K-Means Algorithm:

1. Assignment step: Assign each data point to the closest cluster
2. Refitting step: Move each cluster center to the center of gravity of the data assigned to it

The EM Algorithm:

1. E-step: Compute the posterior probability over z given our current model
2. M-step: Maximize the probability that it would generate the data it is currently responsible for.

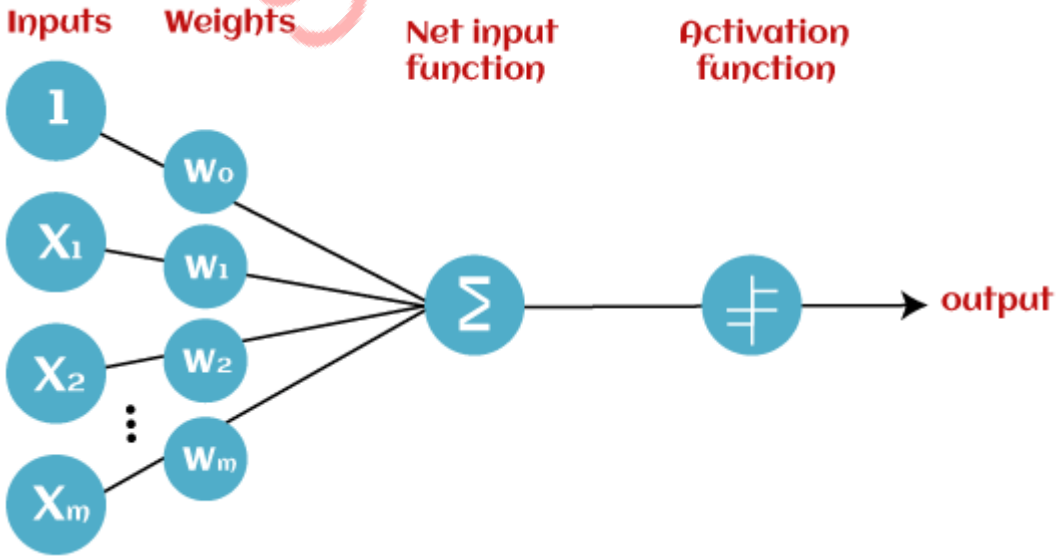
UNIT – 5 (8 Marks and 16 Marks)

1. Explain in detail about Perceptrons and its types?

Perceptron is Machine Learning algorithm for supervised learning of various binary classification tasks. Further, Perceptron is also understood as an Artificial Neuron or neural network unit that helps to detect certain input data computations in business intelligence. Perceptron model is also treated as one of the best and simplest types of Artificial Neural networks. However, it is a supervised learning algorithm of binary classifiers. Hence, we can consider it as a single-layer neural network with four main parameters, i.e., input values, weights and Bias, net sum, and an activation function.

Basic Components of Perceptron

Mr. Frank Rosenblatt invented the perceptron model as a binary classifier which contains three main components. These are as follows:



- **Input Nodes or Input Layer:**

This is the primary component of Perceptron which accepts the initial data into the system for further processing. Each input node contains a real numerical value.

- **Weight and Bias:**

Weight parameter represents the strength of the connection between units. This is another most important parameter of Perceptron components. Weight is directly proportional to the strength of the associated input neuron in deciding the output. Further, Bias can be considered as the line of intercept in a linear equation.

- **Activation Function:**

These are the final and important components that help to determine whether the neuron will fire or not. Activation Function can be considered primarily as a step function.

Types of Activation functions:

- Sign function
- Step function, and
- Sigmoid function

Types of Perceptron Models

Based on the layers, Perceptron models are divided into two types. These are as follows:

1. Single-layer Perceptron Model
2. Multi-layer Perceptron model

• Single Layer Perceptron Model:

➢ This is one of the easiest Artificial neural networks (ANN) types. A single-layered perceptron model consists feed-forward network and also includes a threshold transfer function inside the model. The main objective of the single-layer perceptron model is to analyze the linearly separable objects with binary outcomes.

➢ In a single layer perceptron model, its algorithms do not contain recorded data, so it begins with inconstantly allocated input for weight parameters. Further, it sums up all inputs (weight). After adding all inputs, if the total sum of all inputs is more than a pre-determined value, the model gets activated and shows the output value as +1.

➢ If the outcome is same as pre-determined or threshold value, then the performance of this model is stated as satisfied, and weight demand does not change. However, this model consists of a few discrepancies triggered when multiple weight inputs values are fed into the model. Hence, to find desired output and minimize errors, some changes should be necessary for the weights input.

• Multi-Layered Perceptron Model:

A multi-layer perceptron model also has the same model structure but has a greater number of hidden layers.

The multi-layer perceptron model is also known as the Backpropagation algorithm, which executes in two stages as follows:

- Forward Stage: Activation functions start from the input layer in the forward stage and terminate on the output layer.
- Backward Stage: In the backward stage, weight and bias values are modified as per the model's requirement. In this stage, the error between actual output and demanded originated backward on the output layer and ended on the input layer.

Instead of linear, activation function can be executed as sigmoid, TanH, ReLU, etc., for deployment.
A multi-layer perceptron model has greater processing power and can process linear and non-linear patterns. Further, it can also implement logic gates such as AND, OR, XOR, NAND, NOT, XNOR, NOR.

Advantages of Multi-Layer Perceptron:

- A multi-layered perceptron model can be used to solve complex non-linear problems.
- It works well with both small and large input data.
- It helps us to obtain quick predictions after the training.
- It helps to obtain the same accuracy ratio with large as well as small data.

Disadvantages of Multi-Layer Perceptron:

- In Multi-layer perceptron, computations are difficult and time-consuming.
- In multi-layer Perceptron, it is difficult to predict how much the dependent variable affects each independent variable.
- The model functioning depends on the quality of the training.

Perceptron Function

Perceptron function "f(x)" can be achieved as output by multiplying the input 'x' with the learned weight coefficient 'w'.

Mathematically, we can express it as follows:

f(x)=1; if w.x+b>0 ; otherwise, f(x)=0

'w' represents real-valued weights vector
'b' represents the bias
'x' represents a vector of input x values.

Characteristics of Perceptron

The perceptron model has the following characteristics.

1. Perceptron is a machine learning algorithm for supervised learning of binary classifiers.
2. In Perceptron, the weight coefficient is automatically learned.
3. Initially, weights are multiplied with input features, and the decision is made whether the neuron is fired or not.
4. The activation function applies a step rule to check whether the weight function is greater than zero.
5. The linear decision boundary is drawn, enabling the distinction between the two linearly separable classes +1 and -1.
6. If the added sum of all input values is more than the threshold value, it must have an output signal; otherwise, no output will be shown.

2.Explain about ReLu, Hyperparameter tuning, Normalization, Regularization, Dropout?

i) Hyperparameter Tuning in Deep Learning

The first hyperparameter to tune is the number of neurons in each hidden layer. In this case, the number of neurons in every layer is set to be the same. It also can be made different. The number of neurons should be adjusted to the solution complexity. The task with a more complex level to predict needs more neurons. The number of neurons range is set to be from 10 to 100.

- An activation function is a parameter in each layer. Input data are fed to the input layer, followed by hidden layers, and the final output layer. The output layer contains the output value. The input values moving from a layer to another layer keep changing according to the activation function.

- The activation function decides how to compute the input values of a layer into output values. The output values of a layer are then passed to the next layer as input values again. The next layer then computes the values into output values for another layer again. There are 9 activation functions to tune in to this demonstration. Each activation function has its own formula (and graph) to compute the input values. It will not be discussed in this article.

- The layers of a neural network are compiled and an optimizer is assigned. The optimizer is responsible to change the learning rate and weights of neurons in the neural network to reach the minimum loss function. Optimizer is very important to achieve the possible highest accuracy or minimum loss. There are 7 optimizers to choose from. Each has a different concept behind it.

- One of the hyperparameters in the optimizer is the learning rate. We will also tune the learning rate. Learning rate controls the step size for a model to reach the minimum loss function. A higher learning rate makes the model learn faster, but it may miss the minimum loss function and only reach the surrounding of it. A lower learning rate gives a better chance to find a minimum loss function. As a tradeoff lower learning rate needs higher epochs, or more time and memory capacity resources.

ii) ReLu - Rectified Linear Unit

A Rectified Linear Unit is a form of activation function used commonly in deep learning models. In essence, the function returns 0 if it receives a negative input, and if it receives a positive value, the function will return back the same positive value. The function is understood as:

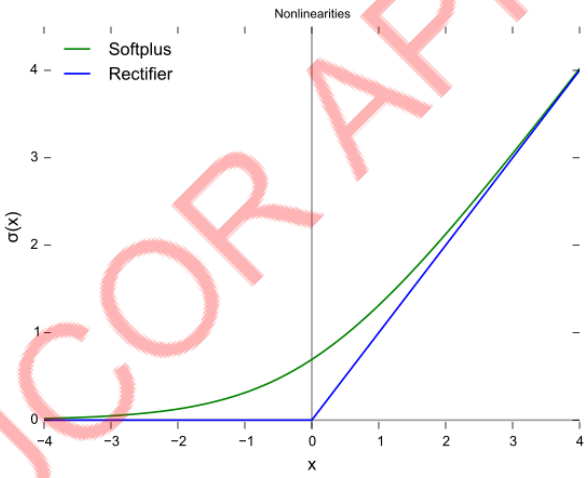
$$f(x)=\max(0,x)$$

The rectified linear unit, or ReLU, allows for the deep learning model to account for non-linearities and specific interaction effects. The image above displays the graphic representation of the ReLU

function. Note that the values for any negative X input result in an output of 0, and only once positive values are entered does the function begin to slope upward.

How does a Rectified Linear Unit work?

To understand how a ReLU works, it is important to understand the effects it has on variable interaction effects. An interaction effect is when a variable affects a prediction depending on the value of associated variables. For example, comparing IQ scores of two different schools may have interaction effects of IQ and age. The IQ of a student in high school is better than the IQ of an elementary school student, as age and IQ interact with each other regardless of the school. This is known as an interaction effect and ReLUs can be applied to minimize interaction effects. For example, if A=1 and B=2, and both have the respective associated weights of 2 and 3, the function would be, f(2A+3B). If A increases, the output will increase as well. However, if B is a large negative value, the output will be 0.



Benefits of using the ReLU function

Its simplicity leads it to be a relatively cheap function to compute. As there is no complicated math, the model can be trained and run in a relatively short time. Similarly, it converges faster, meaning the slope doesn't plateau as the value for X gets larger. This vanishing gradient problem is avoided in ReLU, unlike alternative functions such as sigmoid or tanh. Lastly, ReLU is sparsely activated because for all negative inputs, the output is zero. Sparsity is the principle that specific functions only are activated in concise situations. This is a desirable feature for modern neural networks, as in a sparse network it is more likely that neurons are appropriately processing valuable parts of a problem. For example, a model that is processing images of fish may contain a neuron that is specialized to identity fish eyes. That specific neuron would not be activated if the model was processing images of airplanes instead. This specified use of neuron functions accounts for network sparsity.

iii) Regularization :

- Regularization is one of the most important concepts of machine learning. It is a technique to prevent the model from overfitting by adding extra information to it. Sometimes the machine learning model performs well with the training data but does not perform well with the test data. It means the model is not able to predict the output when deals with unseen data by introducing noise in the output, and hence the model is called overfitted. This problem can be deal with the help of a regularization technique.

- This technique can be used in such a way that it will allow to maintain all variables or features in the model by reducing the magnitude of the variables. Hence, it maintains accuracy as well as a generalization of the model. It mainly regularizes or reduces the coefficient of features toward zero. In simple words, "In regularization technique, we reduce the magnitude of the features by keeping the same number of features."

How does Regularization Work?

Regularization works by adding a penalty or complexity term to the complex model. Let's consider the simple linear regression equation:

$$y = \beta 0+\beta 1x1+\beta 2x2+\beta 3x3+\cdots+\beta_n x_n + b$$

In the above equation, Y represents the value to be predicted X1, X2, …Xn are the features for Y.β0,β1,…..βn are the weights or magnitude attached to the features, respectively. Here represents the bias of the model, and b represents the intercept.. Linear regression models try to optimize the β0 and b to minimize the cost function. The equation for the cost function for the linear model is given below:

Now, we will add a loss function and optimize parameter to make the model that can predict the accurate value of Y. The loss function for the linear regression is called as **RSS or Residual sum of squares.**

Techniques of Regularization

There are mainly two types of regularization techniques, which are given below:

- **Ridge Regression**
- **Lasso Regression**

Ridge Regression

- Ridge regression is one of the types of linear regression in which a small amount of bias is introduced so that we can get better long-term predictions.
- Ridge regression is a regularization technique, which is used to reduce the complexity of the model. It is also called as **L2 regularization**.
- In this technique, the cost function is altered by adding the penalty term to it. The amount of bias added to the model is called **Ridge Regression penalty**. We can calculate it by multiplying with the lambda to the squared weight of each individual feature.
- The equation for the cost function in ridge regression will be:

$$\sum_{i=1}^{M}(y_i - y'_i)^2 = \sum_{i=1}^{M}\left(y_i - \sum_{j=0}^{n}\beta_j * x_{ij}\right)^2 + \lambda\sum_{j=0}^{n}{\beta_j}^2$$

- In the above equation, the penalty term regularizes the coefficients of the model, and hence ridge regression reduces the amplitudes of the coefficients that decreases the complexity of the model.
- As we can see from the above equation, if the values of **λ tend to zero, the equation becomes the cost function of the linear regression model.** Hence, for the minimum value of λ, the model will resemble the linear regression model.
- A general linear or polynomial regression will fail if there is high collinearity between the independent variables, so to solve such problems, Ridge regression can be used.
- It helps to solve the problems if we have more parameters than samples.

Lasso Regression:

- Lasso regression is another regularization technique to reduce the complexity of the model. It stands for **Least Absolute and Selection Operator.**
- It is similar to the Ridge Regression except that the penalty term contains only the absolute weights instead of a square of weights.
- Since it takes absolute values, hence, it can shrink the slope to 0, whereas Ridge Regression can only shrink it near to 0.
- It is also called as **L1 regularization.** The equation for the cost function of Lasso regression will be:

$$\sum_{i=1}^{M}(y_i - y'_i)^2 = \sum_{i=1}^{M}\left(y_i - \sum_{j=0}^{n}\beta_j * x_{ij}\right)^2 + \lambda\sum_{j=0}^{n}|\beta_j|$$

- Some of the features in this technique are completely neglected for model evaluation.
- Hence, the Lasso regression can help us to reduce the overfitting in the model as well as the feature selection.

Key Difference between Ridge Regression and Lasso Regression

- **Ridge regression** is mostly used to reduce the overfitting in the model, and it includes all the features present in the model. It reduces the complexity of the model by shrinking the coefficients.
- **Lasso regression** helps to reduce the overfitting in the model as well as feature selection.

3. Explain about Error Backpropagation?

Backpropagation, or backward propagation of errors, is an algorithm that is designed to test for errors working back from output nodes to input nodes. It is an important mathematical tool for improving the accuracy of predictions in data mining and machine learning. Essentially, backpropagation is an algorithm used to calculate derivatives quickly.

There are two leading types of backpropagation networks:

1. **Static backpropagation:**
   Static backpropagation is a network developed to map static inputs for static outputs. Static backpropagation networks can solve static classification problems, such as optical character recognition (OCR).

2. **Recurrent backpropagation.**
   The recurrent backpropagation network is used for fixed-point learning. Recurrent backpropagation activation feeds forward until it reaches a fixed value.

What is a backpropagation algorithm in a neural network?

Artificial neural networks use backpropagation as a learning algorithm to compute a gradient descent with respect to weight values for the various inputs. By comparing desired outputs to achieved system outputs, the systems are tuned by adjusting connection weights to narrow the difference between the two as much as possible. The algorithm gets its name because the weights are updated backward, from output to input.

Advantages

- It does not have any parameters to tune except for the number of inputs.
- It is highly adaptable and efficient and does not require any prior knowledge about the network.
- It is a standard process that usually works well.
- It is user-friendly, fast and easy to program.
- Users do not need to learn any special functions.

DISADVANTAGES

It prefers a matrix-based approach over a mini-batch approach.

- Data mining is sensitive to noise and irregularities.
- Performance is highly dependent on input data.
- Training is time- and resource-intensive.

Features of Backpropagation:

1. it is the gradient descent method as used in the case of simple perceptron network with the differentiable unit.
2. it is different from other networks in respect to the process by which the weights are calculated during the learning period of the network.
3. training is done in the three stages :
   - the feed-forward of input training pattern
   - the calculation and backpropagation of the error
   - updation of the weight

Working of Backpropagation:

Neural networks use supervised learning to generate output vectors from input vectors that the network operates on. It Compares generated output to the desired output and generates an error report if the result does not match the generated output vector. Then it adjusts the weights according to the bug report to get your desired output.

Backpropagation Algorithm:

Step 1: Inputs X, arrive through the preconnected path.
Step 2: The input is modeled using true weights W. Weights are usually chosen randomly.
Step 3: Calculate the output of each neuron from the input layer to the hidden layer to the output layer.
Step 4: Calculate the error in the outputs

Backpropagation Error= Actual Output – Desired Output

Step 5: From the output layer, go back to the hidden layer to adjust the weights to reduce the error.
Step 6: Repeat the process until the desired output is achieved.

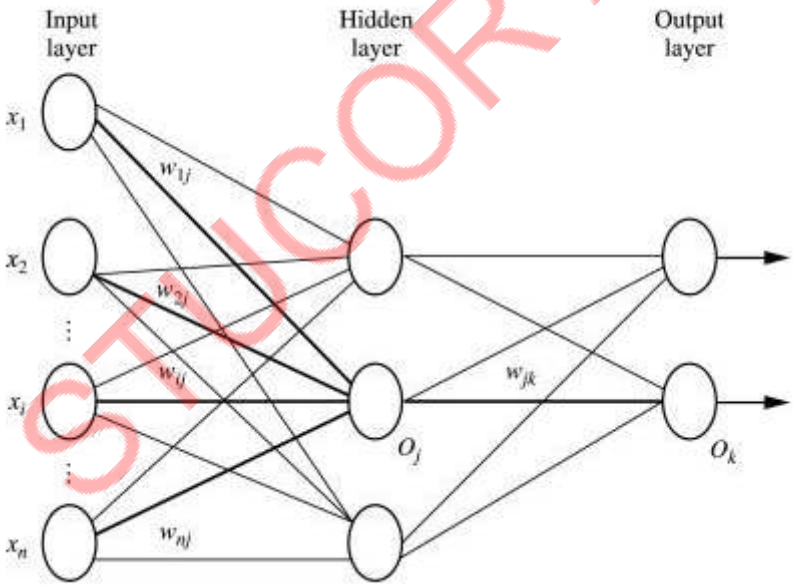


Fig: Error backpropagation

- x = inputs training vector $x=(x_1,x_2,\ldots\ldots\ldots\ldots x_n)$.
- t = target vector $t=(t_1,t_2\ldots\ldots\ldots\ldots t_{n)}$.
- δ_k = error at output unit.
- δ_j = error at hidden layer.
- α = learning rate.
- V_{0j} = bias of hidden unit j.

Training Algorithm :

Step 1: Initialize weight to small random values.

Step 2: While the steps stopping condition is to be false do step 3 to 10.
Step 3: For each training pair do step 4 to 9 (Feed-Forward).
Step 4: Each input unit receives the signal unit and transmits the signal x_i signal to all the units.
Step 5: Each hidden unit Zj (z=1 to a) sums its weighted input signal to calculate its net input

$$z_{inj} = v_{0j} + \Sigma x_i v_{ij} \quad (i=1 \text{ to } n)$$

Applying activation function $z_j = f(z_{inj})$ and sends this signals to all units in the layer about i.e output units

For each output l=unit y_k = (k=1 to m) sums its weighted input signals.

$$y_{ink} = w_{0k} + \Sigma z_i w_{jk} \quad (j=1 \text{ to } a)$$

and applies its activation function to calculate the output signals.

$$y_k = f(y_{ink})$$

Backpropagation Error :
Step 6: Each output unit y_k (k=1 to n) receives a target pattern corresponding to an input pattern then error is calculated as:

$$\delta_k = (t_k - y_k) + y_{ink}$$

Step 7: Each hidden unit Z_j (j=1 to a) sums its input from all units in the layer above

$$\delta_{inj} = \Sigma \delta_j w_{jk}$$

The error information term is calculated as :

$$\delta_j = \delta_{inj} + z_{inj}$$

Updation of weight and bias :
Step 8: Each output unit y_k (k=1 to m) updates its bias and weight (j=1 to a). The weight correction term is given by :

$$\Delta w_{jk} = \alpha \delta_k z_j$$

and the bias correction term is given by $\Delta w_k = \alpha \delta_k$.

therefore $w_{jk(new)} = w_{jk(old)} + \Delta w_{jk}$

$$w_{0k(new)} = w_{ok(old)} + \Delta w_{ok}$$

for each hidden unit z_j (j=1 to a) update its bias and weights (i=0 to n) the weight connection term

$$\Delta v_{ij} = \alpha \delta_j x_i$$

and the bias connection on term

$$\Delta v_{0j} = \alpha \delta_j$$

Therefore $v_{ij(new)} = v_{ij(old)} + \Delta v_{ij}$

$$v_{0j(new)} = v_{0j(old)} + \Delta v_{0j}$$

Step 9: Test the stopping condition. The stopping condition can be the minimization of error, number of epochs.

4. Explain detail about activation functions?

ACTIVATION FUNCTIONS

The activation function decides whether a neuron should be activated or not by calculating the weighted sum and further adding bias to it. The purpose of the activation function

is to introduce non-linearity into the output of a neuron.

✓ In a neural network, we would update the weights and biases of the neurons on the basis of the error at the output. This process is known as back-propagation**.** Activation functions make the back-propagation possible since the gradients are supplied along with the error to update the weights and biases.

✓ A neural network without an activation function is essentially just a linear regression model. The activation function does the non-linear transformation to the input making it capable to learn and perform more complex tasks.

Calculation at Output layer

$$z(2) = (W(2) * [W(1)X + b(1)]) + b(2)$$

$$z(2) = [W(2) * W(1)] * X + [W(2)*b(1) + b(2)]$$

Let,

$$[W(2) * W(1)] = W$$

$$[W(2)*b(1) + b(2)] = b$$

Final output : $z(2) = W*X + b$

which is again a linear function

Variants of Activation Function

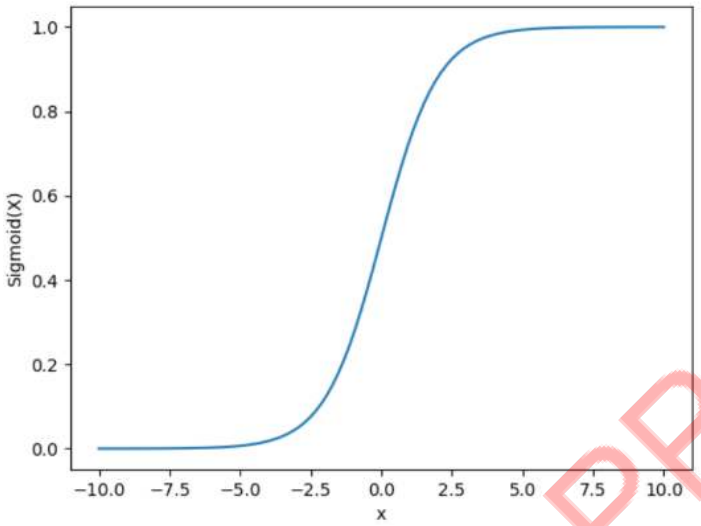
Linear Function

- **Equation :** Linear function has the equation similar to as of a straight line i.e. **y = x**
- No matter how many layers we have, if all are linear in nature, the final activation function of last layer is nothing but just a linear function of the input of first layer.
- **Range :** -inf to +inf
- **Uses:** **Linear activation function** is used at just one place i.e. output layer.
- **Issues:** If we will differentiate linear function to bring non-linearity, result will no more depend on *input "x"* and function will become constant, it won't introduce any ground-breaking behavior to our algorithm.

For example: Calculation of price of a house is a regression problem. House price may have any big/small value, so we can apply linear activation at output layer. Even in this case neural net must have any non-linear function at hidden layers.

Sigmoid Function

- It is a function which is plotted as **'S'** shaped graph.
  - **Equation :** $A = 1/(1 + e^{-x})$



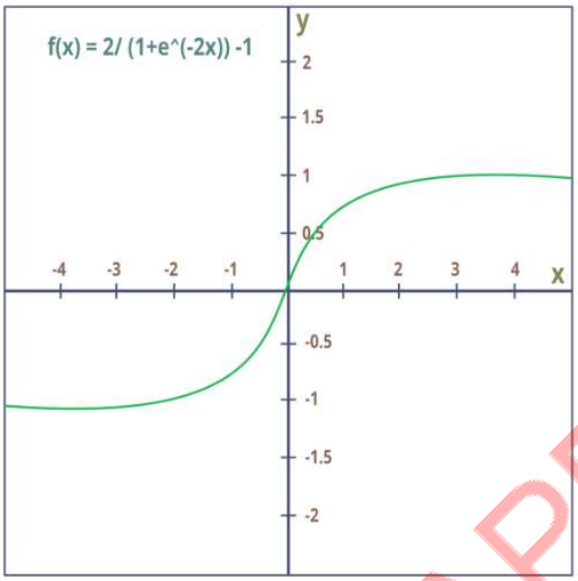
- **Nature:** Non-linear. Notice that X values lies between -2 to 2, Y values are very steep. This means, small changes in x would also bring about large changes in the value of Y.
- **Value Range :** 0 to 1
- **Uses:** Usually used in output layer of a binary classification, where result is either 0 or 1, as value for sigmoid function lies between 0 and 1 only so, result can be predicted easily to be ***1*** if value is greater than **0.5** and ***0*** otherwise.

Tanh Function

- The activation that works almost always better than sigmoid function is Tanh function also knows as **Tangent Hyperbolic function**. It's actually mathematically shifted version of the sigmoid function. Both are similar and can be derived from each other.
- **Equation :-**

$$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

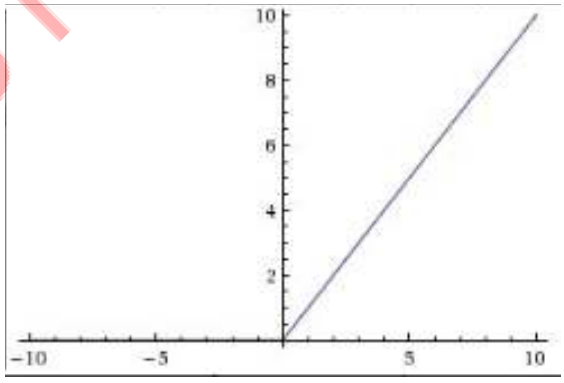
- **Value Range :-** -1 to +1
- **Nature :-** non-linear
- **Uses: -** Usually used in hidden layers of a neural network as its values lies between **-1 to 1** hence the mean for the hidden layer comes out be 0 or very close to it, hence helps in centering the data by bringing mean close to 0. This makes learning for the next layer much easier.



RELU Function

- It Stands for Rectified linear unit. It is the most widely used activation function. Chiefly implemented in hidden layers of neural network.
- **Equation: -** ***A(x) = max (0, x)***. It gives an output x if x is positive and 0 otherwise.
- **Value Range :-** [0, inf)
- **Nature: -** non-linear, which means we can easily backpropagate the errors and have multiple layers of neurons being activated by the ReLU function.
- **Uses: -** ReLu is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. At a time only a few neurons are activated making the network sparse making it efficient and easy for computation.

In simple words, RELU learns *much faster* than sigmoid and Tanh function.



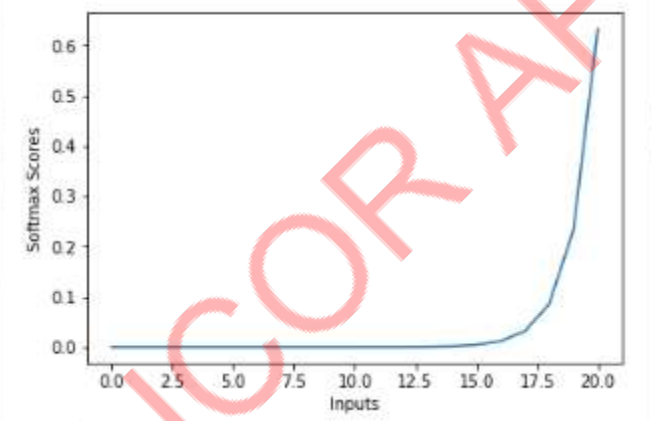
Softmax Function

The softmax function is also a type of sigmoid function but is handy when we are trying to handle multi- class classification problems.

- **Nature :-** non-linear

- **Uses: -** Usually used when trying to handle multiple classes. The softmax function was commonly found in the output layer of image classification problems. The softmax function would squeeze the outputs for each class between 0 and 1 and would also divide by the sum of the outputs.

- **Output: -** The softmax function is ideally used in the output layer of the classifier where we are actually trying to attain the probabilities to define the class of each input.
- The basic rule of thumb is if you really don't know what activation function to use, then simply use RELU as it is a general activation function in hidden layers and is used in most cases these days.
- If your output is for binary classification then, sigmoid function is very natural choice for output layer.
- If your output is for multi-class classification then, Softmax is very useful to predict the probabilities of each classes.



5. Explain in detail about gradient descent optimization?

Gradient descent optimization

✓ Gradient descent is an optimization algorithm in gadget mastering used to limit a feature with the aid of iteratively moving towards the minimal fee of characteristic.

✓ We essentially use this algorithm when we have to locate the least possible values which could fulfill a given free function. In gadget getting to know, greater regularly that not we try to limit loss features (like mean squared error). By minimizing the loss characteristic, we will improve our model and gradient descent is one of the most popular algorithms used for this cause.

✓ The graph above shows how exactly a gradient descent set of rules works.

✓ We first take a factor in the value function and begin shifting in steps in the direction of the minimum factor. The size of the step, or how quickly we ought to converge to the minimum factor is defined by learning rate.

✓ We can cowl more location with better learning fee but at the risk of overshooting the minima. On the opposite hand, small steps/ smaller gaining knowledge of charges will eat a number of times to attain the lowest point.

✓ Now, the direction where in algorithm has to transport is also important. We calculate this by way of using derivatives. You need to be familiar with derivatives from calculus. A spinoff is largely calculated because the slope of the graph at any specific factor. We get that with the aid of finding the tangent line to the graph at that point. The extra sleep the tangent, would suggest that more steps would be needed to reach minimum point; much less steep might suggest lesser steps are required to reach the minimum factor.

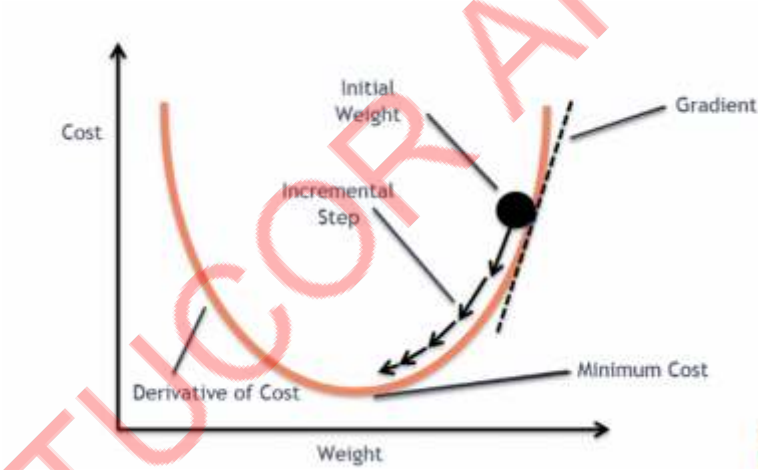


Fig: Gradient descent optimization

Stochastic gradient descent

The word stochastic means a system or a process that is linked with a random probability. Hence, in stochastic gradient descent, a few samples are selected randomly instead of the whole data set for each iteration.

- ✓ Stochastic gradient descent is a type of gradient descent that runs one training example per iteration. It processes a training epoch for each example within a dataset and updates each training example's parameters one at a time.
- ✓ As it requires only one training example at a time, hence it is easier to store in allocated memory. However, it shows some computational efficiency losses in comparison to batch gradient systems as it shows frequent updates that require more detail and speed.

- ✓ Further, due to frequent updates, it is also treated as a noisy gradient. However, sometimes it can be helpful in finding the global minimum and also escaping the local minimum.

Advantages of stochastic gradient descent:

It is easier to allocate in desired memory.
It is relatively fast to compute than batch gradient descent.
It is more efficient for large dataset.

Disadvantages of stochastic gradient descent:

SGD require a number of hyperparameters such as the regularization parameter and the number of iterations.

SGD is sensitive to feature scaling.