# LAKIREDDY BALI REDDYCOLLEGE OF ENGINEERING

(AUTONOMOUS)



Department of Computer Science& Engineering

(Artificial Intelligence and Machine Learning)

Introduction to Artificial Intelligence and Machine Learning Lab

(20AM51)

**Name of the Student : PREM CHAND**

**Registered Number : 21761A4214**

**Branch & Section : CSE(AI&ML)**

**Academic Year : 2022 - 23**

# LAKIREDDY BALI REDDY COLLEGE OF ENGINEERING

(AUTONOMOUS)



**CERTIFICATE**

Certificate that this is a bonafied record of the practical work done in \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Laboratory by \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ with Regd. No. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ of \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_B.Tech Course\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Semester in \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Branch during the Academic Year2022-23

No.of Experiments held:\_\_\_\_\_\_\_\_

No.of Experiments Done: \_\_\_\_\_\_\_\_

 2023 Signature of the Faculty

INTERNAL EXAMINER EXTERNAL EXAMINER

|  |  |
| --- | --- |
| **EXPERIMENTS** | **PAGE NO.** |
| 1. **ARTIFICIAL INTELIGENCE** 2. WATER JUG PROBLEM 3. TIC TAC TOE 4. MONKEY BANANA PROBLEM 5. TOWERS OF HANOI 6. **MACHINE LEARNING**   FIND-S ALGORITHM  CEA ALGORITHM  LINEAR AND MULTI LINEAR REGRESSION  POLYNOMIAL REGRESSION  LOGISTIC REGRESSION  DECISION TREE REGRESSOR  **METHOD PARAMETERS**  DECISIONS TREE CLASSIFIER  RANDOM FOREST REGRESSOR  RANDOM FOREST CLASSIFIER  10. DATA PRE-PROCESSING AND CORRELATION | **2**  **8**  **14**  **18**  **22**  **25**  **29**  **34**  **36**  **41**  **43**  **45**  **47**  **49** |

**INDEX**

**ARTIFICIAL INTELLIGENCE**

1.WATER JUG PROBLEM

2.TIC TAC TOE

3.MONKEY BANANA PROBLEM

4.TOWERS OF HANOI

**1.WATER JUG PROBLEM**

**Description:**

The Water Jug Problem, also known as the Die Hard Water Jug Problem, is a classic puzzle that involves using two jugs of different capacities to measure a specific amount of water. The problem can be stated as follows:

You are given two empty jugs, Jug A and Jug B, with capacities of x and y liters, respectively, where x and y are positive integers. You also have access to a water source. The goal is to measure a target amount of water, T liters, using these jugs.

The following actions are allowed:

1. Filling a jug: You can fill either Jug A or Jug B from the water source until the jug is completely full.

2. Emptying a jug: You can empty the entire content of either Jug A or Jug B.

3. Pouring water: You can pour the contents of one jug into the other jug until the pouring jug is empty or the receiving jug is full.

The objective is to determine a sequence of actions that allows you to measure exactly T liters of water using the available jugs. It is important to note that the measurements do not have to be exact multiples of the jug capacities.

This problem is often used to illustrate concepts of problem-solving, mathematical reasoning, and optimization strategies. Different variations of the Water Jug Problem exist with varying capacities and target amounts, but the basic principles remain the same.

**Algorithm:**

The algorithm to solve the water jug problem using state space search can be outlined as follows:

1. Represent the problem as a state space graph with nodes as states and edges as transitions.
2. Initialize the open list with the initial state (0, 0).
3. Repeat the following steps until the open list is empty:
   1. Choose a state from the open list and remove it from the list.
   2. If the state is the goal state, return the solution.
   3. Otherwise, generate all possible successor states using the production rules and add them to the open list if they have not been visited before.
4. If the goal state is not reached, return failure.

**Python code:**

print("Water Jug Problem")

x, y = 0, 0

c = int(input("Enter the goal state: "))

while True:

r = int(input("Enter the rule number: "))

if r == 1:

if x < 4:

x, y = 4, y

if r == 2:

if y < 3:

x, y = x, 3

if r == 3:

if x > 0:

x, y = 0, y

if r == 4:

if y > 0:

x, y = x, 0

if r == 5:

if y > 0 and 0 < x + y >= 4:

x, y = 4, (y - (4 - x))

if r == 6:

if 0 < x + y >= 3 and x > 0:

x, y = (x - (3 - y)), 3

if r == 7:

if 0 < x + y <= 4 and y >= 0:

x, y = (x + y), 0

if r == 8:

if 0 <= x + y <= 3 and x >= 0:

x, y = 0, x + y

print("Current state is: (%d,%d)" % (x, y))

if x == c:

print("Goal State is reached")

break

**output:**

Water Jug Problem

Enter te goal state:2

Enter the rule no:1

current state is : (4,0)

Enter the rule no:6

current state is : (1,3)

Enter the rule no:4

current state is : (1,0)

Enter the rule no:8

current state is : (0,1)

Enter the rule no:1

current state is : (4,1)

Enter the rule no:6

current state is : (2,3)

Goal State is reached

**Java Code:**

import java.util.\*;

public class Main {

public static void main(String[] args) {

int x = 0, y = 0, g, c1, c2, r, c;

Scanner read = new Scanner(System.in);

System.out.println("Enter the capacity of jug 1:");

c1 = read.nextInt();

System.out.println("Enter the capacity of jug 2:");

c2 = read.nextInt();

System.out.println("Enter the goal state:");

c = read.nextInt();

while (true) {

System.out.println("Enter the Rule:");

r = read.nextInt();

if (r == 1) {

if (x < c1) {

x = c1;

}

}

if (r == 2) {

if (y < c2) {

y = c2;

}

}

if (r == 3) {

if (x > 0) {

x = 0;

}

}

if (r == 4) {

if (y > 0) {

y = 0;

}

}

if (r == 5) {

if ((0 < x + y && x + y >= c1) && y > 0) {

y = y - (c1 - x);

x = c1;

}

}

if (r == 6) {

if (((0 < x + y) && (x + y >= c2)) && x > 0) {

x = x - (c2 - y);

y = c2;

}

}

if (r == 7) {

if ((0 < x + y && x + y <= c1) && y >= 0) {

x = x + y;

y = 0;

}

}

if (r == 8) {

if ((0 < x + y && x + y <= c1) && x >= 0) {

y = x + y;

x = 0;

}

}

System.out.println("Current state is: (" + x + "," + y + ")");

if (x == c) {

System.out.println("Goal State is reached");

break;

}

}

}

}

**2.TIC-TAC-TOE PROBLEM**

**Description:**

Tic-tac-toe, also known as noughts and crosses, is a classic paper-and-pencil game played on a grid of 3x3 squares. The game involves two players, typically referred to as "X" and "O," who take turns marking empty squares in an attempt to form a line of three of their symbols horizontally, vertically, or diagonally.

The following is a description of the basic rules and mechanics of tic-tac-toe:

1. Game Setup: The game begins with an empty 3x3 grid. The players decide who will be "X" and who will be "O."

2. Player Turns: The players take turns placing their respective symbols ("X" or "O") in any empty square on the grid.

3. Winning Condition: The objective of the game is to be the first player to form a line of three of their symbols either horizontally, vertically, or diagonally. If a player successfully achieves this, they win the game. If all squares on the grid are filled and no player has won, the game is a draw.

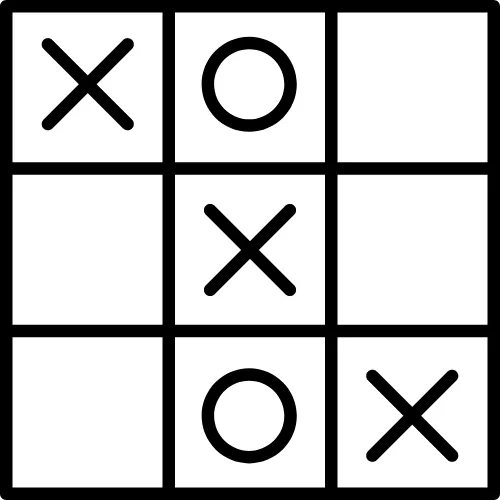
4. Valid Moves: Players can only place their symbol in an empty square. They cannot overwrite or remove their opponent's symbol or place their symbol on an already occupied square.

5. Game Termination: The game terminates when either a player wins by forming a line of three symbols, or when the game ends in a draw.

Tic-tac-toe is a simple and easy-to-learn game, but it can still involve strategy and critical thinking. Skilled players can anticipate their opponent's moves and plan their own moves strategically to maximize their chances of winning or forcing a draw. The game has been studied extensively, and with optimal play from both players, it always ends in a draw.

**algorithm to implementing Tic Tac Toe:**

1. Create a 3x3 grid to represent the Tic Tac Toe board.
2. Start with an empty board, where all cells are unmarked.
3. Define two players: Player 1 (X) and Player 2 (O).
4. Determine the starting player (either randomly or based on user input).
5. Repeat the following steps until the game is over:
   1. Display the current state of the board.
   2. Prompt the current player for their move (either by inputting the position or using an AI algorithm).
   3. Verify if the move is valid. If not, repeat step 5b.
   4. Update the board with the player's move.
   5. Check if the current player has won the game by having three marks in a row, column, or diagonal. If so, declare the current player as the winner and end the game.
   6. If the board is full and no player has won, declare the game as a draw and end the game.
   7. Switch to the next player.
6. Display the final state of the board and the outcome of the game (winner or draw).
7. End the game.



**Python Code:**

tic = [[1,2,3],

[4,5,6],

[7,8,9]]

p = [0,2,3]

p[1] = input("Enter player 1 name = ")

p[2] = input("Enter player 2 name = ")

print("Tic Tac Toe Board:")

for i in tic:

print(i)

# Insertion function

def insertAt(place, i):

for j in range(len(tic)):

for k in range(len(tic[j])):

if tic[j][k] == place:

if i == 1:

tic[j][k] = "X"

else:

tic[j][k] = "O"

# Verification of places

def verify(i):

if ((1 and 2 and 3) not in tic[0]) and ((4 and 5 and 6) not in tic[1]) and ((7 and 8 and 9) not in tic[2]):

print("Draw!! Try again")

exit(0)

# Horizontal lines

elif tic[0][0] == tic[0][1] == tic[0][2]:

print(f"Hurrah! player {p[i]} won the match")

exit(0)

elif tic[1][0] == tic[1][1] == tic[1][2]:

print(f"Hurrah! player {p[i]} won the match")

exit(0)

elif tic[2][0] == tic[2][1] == tic[2][2]:

print(f"Hurrah! player {p[i]} won the match")

exit(0)

# Diagonal lines

elif tic[0][0] == tic[1][1] == tic[2][2]:

print(f"Hurrah! player {p[i]} won the match")

exit(0)

elif tic[0][2] == tic[1][1] == tic[2][0]:

print(f"Hurrah! player {p[i]} won the match")

exit(0)

# Vertical lines

elif tic[0][0] == tic[1][0] == tic[2][0]:

print(f"Hurrah! player {p[i]} won the match")

exit(0)

elif tic[0][1] == tic[1][1] == tic[2][1]:

print(f"Hurrah! player {p[i]} won the match")

exit(0)

elif tic[0][2] == tic[1][2] == tic[2][2]:

print(f"Hurrah! {p[i]} won the match")

exit(0)

# Displaying the tic-tac-toe panel

def disp():

for i in tic:

print(i)

i = 1

for itr in range(9):

place = int(input(f"{p[i]}, Enter a position = "))

if i == 1:

insertAt(place, i)

disp()

verify(i)

i = 2

else:

insertAt(place, i)

disp()

verify(i)

i = 1

**Output:**

Enter player 1 name = chanti

Enter player 2 name = bunty

Tic Tac Toe Board:

[1, 2, 3]

[4, 5, 6]

[7, 8, 9]

chanti, Enter a position = 1

['X', 2, 3]

[4, 5, 6]

[7, 8, 9]

bunty, Enter a position = 2

['X', 'O', 3]

[4, 5, 6]

[7, 8, 9]

chanti, Enter a position = 4

['X', 'O', 3]

['X', 5, 6]

[7, 8, 9]

bunty, Enter a position = 5

['X', 'O', 3]

['X', 'O', 6]

[7, 8, 9]

chanti, Enter a position = 7

['X', 'O', 3]

['X', 'O', 6]

['X', 8, 9]

Hurrah! player chanti won the match

**3.MONKEY BANANA PROBLEM**

**Description:**

The Monkey and Banana Problem, also known as the Monkey and Banana Puzzle, is a classic logic puzzle. It involves a monkey trying to reach a banana hanging from the ceiling by climbing on boxes. The problem can be described as follows:

A monkey is in a room with a banana hanging from the ceiling. The room contains several boxes that the monkey can stack and climb on. The goal is for the monkey to reach the banana and obtain it.

**Rules:**

1. The monkey can only do one of the following actions at a time: move left, move right, move up, move down, or stack a box.

2. The monkey can only climb on top of a stack of boxes if the stack is stable (i.e., it won't tip over).

3. The monkey can only stack or unstack a single box at a time.

4. The monkey can't jump or reach the banana directly without climbing on boxes.

**Objective:**

The monkey needs to figure out the sequence of actions required to reach the banana, considering the constraints and rules mentioned above.

The challenge in the Monkey and Banana Problem is to determine the optimal sequence of moves that allows the monkey to reach the banana efficiently. It requires logical thinking, problem-solving, and understanding the limitations and possibilities of the monkey's actions.

Solving the Monkey and Banana Problem often involves considering different strategies, visualizing the problem, and analyzing the potential consequences of each move.

**Algorithm:**

Initialize the starting state of the room with the monkey, banana, and boxes.

Create an empty stack to store the sequence of moves taken by the monkey.

While the monkey has not reached the banana:

a. Check if the monkey is directly below the banana. If so, the monkey can directly climb and obtain the banana.

b. Otherwise, iterate through the available actions (move left, move right, move up, move down, stack box) in a predefined order.

For each action, check if it is valid and leads to a new state.

If the action is valid, apply it to generate a new state of the room and update the monkey's position and stack of boxes.

Push the action onto the stack to remember the sequence of moves taken.

Recursively call the algorithm with the updated state of the room.

c. If none of the available actions lead to the monkey reaching the banana, backtrack by popping the last action from the stack and reverting the state of the room accordingly.

Once the monkey reaches the banana, the algorithm terminates.

Output the sequence of moves stored in the stack as the solution to the problem.



**Python code:**

move(state(middle,onbox,middle,hasnot),

grasp,

state(middle,onbox,middle,has)).

move(state(P,onfloor,P,H),

climb,

state(P,onbox,P,H)).

move(state(P1,onfloor,P1,H),

drag(P1,P2),

state(P2,onfloor,P2,H)).

move(state(P1,onfloor,B,H),

walk(P1,P2),

state(P2,onfloor,B,H)).

canget(state(\_,\_,\_,has)).

canget(State1) :-

move(State1,\_,State2),

canget(State2).

**Output:**

| ?- [monkey\_banana].

compiling D:/TP Prolog/Sample\_Codes/monkey\_banana.pl for byte code...

D:/TP Prolog/Sample\_Codes/monkey\_banana.pl compiled, 17 lines read - 2167 bytes written, 19 ms

(31 ms) yes

| ?- canget(state(atdoor, onfloor, atwindow, hasnot)).

true ?

yes

| ?- trace

.

The debugger will first creep -- showing everything (trace)

yes

{trace}

| ?- canget(state(atdoor, onfloor, atwindow, hasnot)).

1 1 Call: canget(state(atdoor,onfloor,atwindow,hasnot)) ?

2 2 Call: move(state(atdoor,onfloor,atwindow,hasnot),\_52,\_92) ?

2 2 Exit:move(state(atdoor,onfloor,atwindow,hasnot),walk(atdoor,\_80),state(\_80,onfloor,atwindow,hasnot)) ?

3 2 Call: canget(state(\_80,onfloor,atwindow,hasnot)) ?

4 3 Call: move(state(\_80,onfloor,atwindow,hasnot),\_110,\_150) ?

4 3 Exit: move(state(atwindow,onfloor,atwindow,hasnot),climb,state(atwindow,onbox,atwindow,hasnot)) ?

5 3 Call: canget(state(atwindow,onbox,atwindow,hasnot)) ?

6 4 Call: move(state(atwindow,onbox,atwindow,hasnot),\_165,\_205) ?

6 4 Fail: move(state(atwindow,onbox,atwindow,hasnot),\_165,\_193) ?

5 3 Fail: canget(state(atwindow,onbox,atwindow,hasnot)) ?

4 3 Redo: move(state(atwindow,onfloor,atwindow,hasnot),climb,state(atwindow,onbox,atwindow,hasnot)) ?

4 3 Exit: move(state(atwindow,onfloor,atwindow,hasnot),drag(atwindow,\_138),state(\_138,onfloor,\_138,hasnot)) ?

5 3 Call: canget(state(\_138,onfloor,\_138,hasnot)) ?

6 4 Call: move(state(\_138,onfloor,\_138,hasnot),\_168,\_208) ?

6 4 Exit: move(state(\_138,onfloor,\_138,hasnot),climb,state(\_138,onbox,\_138,hasnot)) ?

7 4 Call: canget(state(\_138,onbox,\_138,hasnot)) ?

8 5 Call: move(state(\_138,onbox,\_138,hasnot),\_223,\_263) ?

8 5 Exit: move(state(middle,onbox,middle,hasnot),grasp,state(middle,onbox,middle,has)) ?

9 5 Call: canget(state(middle,onbox,middle,has)) ?

9 5 Exit: canget(state(middle,onbox,middle,has)) ?

7 4 Exit: canget(state(middle,onbox,middle,hasnot)) ?

5 3 Exit: canget(state(middle,onfloor,middle,hasnot)) ?

3 2 Exit: canget(state(atwindow,onfloor,atwindow,hasnot)) ?

1 1 Exit: canget(state(atdoor,onfloor,atwindow,hasnot))

true ?

**4.TOWERS OF HANOI**

**Description:**

The Towers of Hanoi is a classic mathematical puzzle that involves moving a stack of disks from one peg to another peg with the help of a third peg. The problem can be described as follows:

There are three pegs labeled A, B, and C, and there are n disks of different sizes. Initially, all the disks are stacked on peg A, with the largest disk at the bottom and the smallest disk at the top. The goal is to move the entire stack of disks from peg A to peg C, while following the rules and constraints of the puzzle.

**Rules:**

1. Only one disk can be moved at a time.

2. A larger disk cannot be placed on top of a smaller disk.

3. Only the topmost disk of a peg can be moved.

**Objective:**

The objective of the Towers of Hanoi puzzle is to find the optimal sequence of moves that allows you to transfer the entire stack of disks from peg A to peg C, using peg B as an intermediate peg.

**Algorithm for solving the Towers of Hanoi problem:**

1. If the number of disks is 1, simply move the disk from the source peg to the destination peg and return.

2. Recursively solve the subproblem of moving n-1 disks from the source peg to the auxiliary peg, using the destination peg as the auxiliary peg.

3. Move the largest disk (bottom disk) from the source peg to the destination peg.

4. Recursively solve the subproblem of moving the n-1 disks from the auxiliary peg to the destination peg, using the source peg as the auxiliary peg.

5. The problem is now reduced to the subproblem with n-1 disks, and the process continues until all disks are moved to the destination peg.

**Pseudocode for the recursive algorithm:**

function TowersOfHanoi(n, source, destination, auxiliary):

if n == 1:

move disk from source to destination

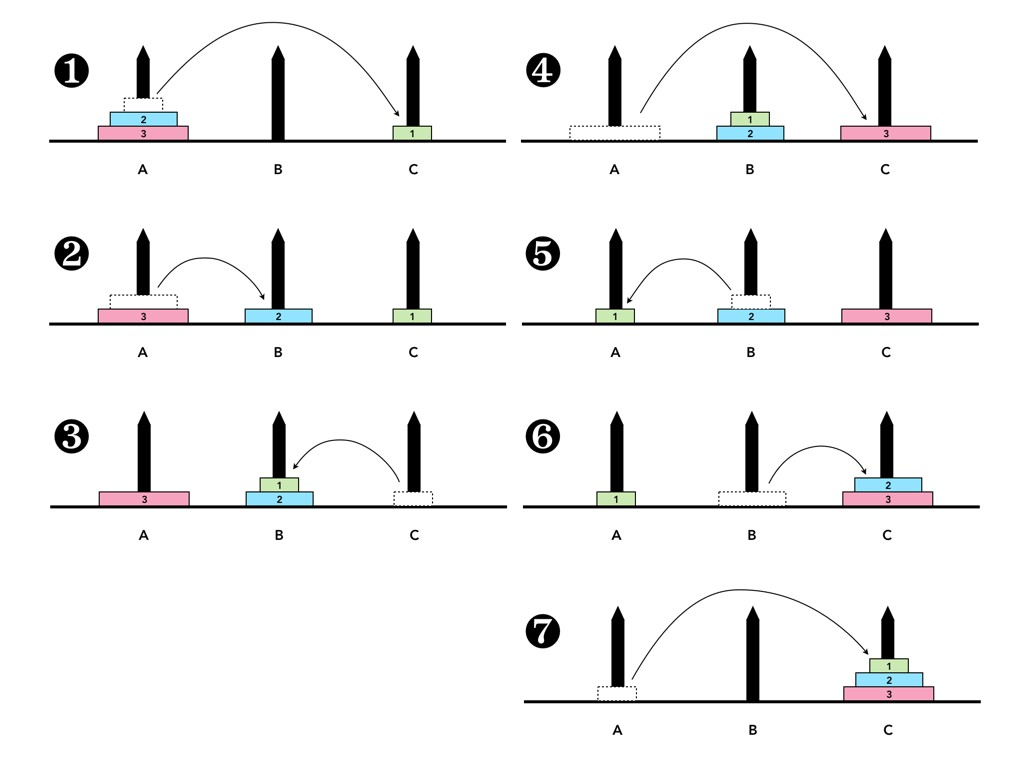
return

TowersOfHanoi(n-1, source, auxiliary, destination)

move largest disk from source to destination

TowersOfHanoi(n-1, auxiliary, destination, source)

To solve the Towers of Hanoi problem, you can call the function with the number of disks (n), the source peg (A), the destination peg (C), and the auxiliary peg (B). The algorithm will recursively move the disks following the rules of the puzzle until all the disks are transferred to the destination peg.



**Python code:**

# Creating a recursive function

def tower\_of\_hanoi(disks, source, auxiliary, target):

if(disks == 1):

print('Move disk 1 from rod {} to rod {}.'.format(source, target))

return

# function call itself

tower\_of\_hanoi(disks - 1, source, target, auxiliary)

print('Move disk {} from rod {} to rod {}.'.format(disks, source, target))

tower\_of\_hanoi(disks - 1, auxiliary, source, target)

disks = int(input('Enter the number of disks: '))

# We are referring source as A, auxiliary as B, and target as C

tower\_of\_hanoi(disks, 'A', 'B', 'C') # Calling the function

**output:**

Enter the number of disks: 3

Move disk 1 from rod A to rod C.

Move disk 2 from rod A to rod B.

Move disk 1 from rod C to rod B.

Move disk 3 from rod A to rod C.

Move disk 1 from rod B to rod A.

Move disk 2 from rod B to rod C.

Move disk 1 from rod A to rod C.

**MACHINE LEARNING**

1. FIND-S ALGORITHM
2. CEA ALGORITHM
3. LINEAR AND MULTI LINEAR REGRESSION
4. POLYNOMIAL REGRESSION
5. LOGISTIC REGRESSION
6. DECISION TREE REGRESSOR

**METHOD PARAMETERS**

7. DECISIONS TREE CLASSIFIER

8.RANDOM FOREST REGRESSOR

9.RANDOM FOREST CLASSIFIER

10.DATA PRE-PROCESSING AND CORRELATION

**1.Aim: To Implement and demonstrate FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .csv file.**

**Description-FindS :**

The Find-S algorithm is a machine learning algorithm used for supervised learning of classification tasks. It is a concept learning algorithm that learns a hypothesis that can be used to classify new examples based on their features.The algorithm starts with the most specific hypothesis, which assumes that all attributes of the instance are negative. It then iteratively updates the hypothesis by generalizing it based on positive examples until it covers all positive examples.

**The Find-S algorithm works as follows:**

1. Initialize the hypothesis h to the most specific hypothesis. This hypothesis states that all attributes of an instance are negative.
2. For each positive training example, update the hypothesis by making the most specific generalization that still includes the positive example.
3. Return the hypothesis.

The Find-S algorithm can be applied to problems with discrete-valued attributes. It produces a hypothesis that is consistent with the training data, but it may not be the most accurate hypothesis.

**Data Set :**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Outlook | Temperature | Humidity | Wind | Play Tennis |
| Overcast | Hot | High | Weak | Yes |
| Rain | Mild | High | Weak | Yes |
| Rain | Cool | Normal | Strong | No |
| Overcast | Cool | Normal | Weak | Yes |

**Pythoncode:**

import csv

num\_attributes = 4

a = []

print("\n The Given Training Data Set \n")

with open('finds.csv', 'r') as csvfile:

reader = csv.reader(csvfile)

for row in reader:

a.append (row)

print(row)

print("\n The initial value of hypothesis: ")

hypothesis = ['0'] \* num\_attributes

print(hypothesis)

for j in range(0,num\_attributes):

hypothesis[j] = a[0][j];

print("\n Find S: Finding a Maximally Specific Hypothesis\n")

for i in range(0,len(a)):

if a[i][num\_attributes]=='Yes':

for j in range(0,num\_attributes):

if a[i][j]!=hypothesis[j]:

hypothesis[j]='?'

else :

hypothesis[j]= a[i][j]

print(" For Training instance No:{0} the hypothesis is".format(i),hypothesis)

print("\n The Maximally Specific Hypothesis for a given TrainingExamples :\n")

print(hypothesis)

**Output:**

The Given Training Data Set

['Overcast', 'Hot', 'High', 'Weak', 'Yes']

['Rain', 'Mild', 'High', 'Weak', 'Yes']

['Rain', 'Cool', 'Normal', 'Strong', 'No']

['Overcast', 'Cool', 'Normal', 'Weak', 'Yes']

['Overcast', 'Hot', 'High', 'Weak', 'Yes']

The initial value of hypothesis:

['0', '0', '0', '0']

Find S: Finding a Maximally Specific Hypothesis

For Training instance No:0 the hypothesis is

['Overcast', 'Hot', 'High', 'Weak']

For Training instance No:1 the hypothesis is

['?', '?', 'High', 'Weak']

For Training instance No:2 the hypothesis is

['?', '?', 'High', 'Weak']

For Training instance No:3 the hypothesis is

['?', '?', '?', 'Weak']

For Training instance No:4 the hypothesis is

['?', '?', '?', 'Weak']

The Maximally Specific Hypothesis for a given TrainingExamples :

['?', '?', '?', 'Weak']

**2. Aim: To Implement and demonstrate CEA algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .csv file.**

**Description-**

The Candidate-Elimination Algorithm (CEA) is a machine learning algorithm used for concept learning in supervised learning tasks. The algorithm is used to find the most specific hypothesis based on a given set of training data samples.The algorithm starts by initializing the most specific hypothesis S0, which is the set of all possible hypotheses in the hypothesis space. It then initializes the most general hypothesis G0, which is the set of all possible hypotheses in the hypothesis space.The algorithm then iterates through each training example and updates the most specific and most general hypotheses based on the example. If an example is positive, the algorithm updates the most specific hypothesis by removing any inconsistent hypotheses. If an example is negative, the algorithm updates the most general hypothesis by removing any inconsistent hypotheses.

The algorithm continues to iterate through the training examples until it converges on a single hypothesis or a set of hypotheses. The final hypothesis is the most specific hypothesis that is consistent with all the positive examples and none of the negative examples.

**The steps of the Candidate-Elimination Algorithm are as follows:**

1. Initialize the most specific hypothesis S0, which includes all the attributes and values of the training examples.
2. Initialize the most general hypothesis G0, which includes all possible attributes and values in the hypothesis space.
3. For each positive training example, update the most specific hypothesis by removing any inconsistent hypotheses.
4. For each negative training example, update the most general hypothesis by removing any inconsistent hypotheses.
5. Return the most specific hypothesis that is consistent with all the positive examples and none of the negative examples.

The Candidate-Elimination Algorithm is useful for finding the most specific hypothesis in a hypothesis space, but it may not always converge to a single hypothesis if there are multiple hypotheses that are consistent with the training data.

**Data Set**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | Outlook | Temperature | Humididty | Wind | Play | | Sunny | Warm | Normal | Strong | Yes | | Sunny | Warm | High | Strong | Yes | | Sunny | Cold | High | Strong | No | |  |  |  |  |

**Python code:**

import numpyas np import pandasaspd

data=pd.read\_csv(path+'/enjoysport.csv') concepts=np.array(data.iloc[:,0:-1]) print("\nInstancesare:\n",concepts)

target =np.array(data.iloc[:,-1]) print("\nTarget Valuesare: ",target)

deflearn(concepts, target): specific\_h=concepts[0].copy()

print("\nInitializationof specific\_hand genearal\_h") print("\nSpecific Boundary:", specific\_h)

general\_h=[["?" for i in range(len(specific\_h))] for i in range(len(specific\_h))] print("\nGeneric Boundary: ",general\_h)

fori, hinenumerate(concepts): print("\nInstance", i+1 , "is", h) if target[i] =="yes":

print("Instance is Positive") forx in range(len(specific\_h)):

ifh[x]!=specific\_h[x]: specific\_h[x] ='?' general\_h[x][x] ='?'

if target[i] =="no":

print("Instance is Negative ") forx in range(len(specific\_h)):

ifh[x]!=specific\_h[x]:

general\_h[x][x] =specific\_h[x] else:

general\_h[x][x] = '?'

print("Specific Bundaryafter", i+1,"Instanceis", specific\_h) print("Generic Boundaryafter", i+1,"Instance is", general\_h) print("\n")

indices=[i for i, val inenumerate(general\_h) ifval ==['?','?','?', '?', '?', '?']] fori inindices:

general\_h.remove(['?', '?', '?', '?', '?', '?']) returnspecific\_h, general\_h

s\_final, g\_final =learn(concepts, target)

print("Final Specific\_h:", s\_final, sep="\n") print("Final General\_h:", g\_final, sep="\n")

**Output:**

Instances are:

[ ['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'high', 'strong', 'warm', 'same'],

['rainy', 'cold', 'high', 'strong', 'warm', 'change', 'high', 'strong', 'cool', 'change']

]

Target values are: ['yes', 'yes', 'no', 'yes']

Initialization of specific\_h and general\_h:

Specific Boundary: ['sunny', 'warm', 'normal', 'strong', 'warm', 'same']

Generic Boundary: [ ['?', '?', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?']

]

Instance 1 is ['sunny', 'warm', 'normal', 'strong', 'warm', 'same'] and is positive.

Specific Boundary after 1 instance is ['sunny', 'warm', 'normal', 'strong', 'warm', 'same']

Generic Boundary after 1 instance is [ ['?', '?', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?']

]

Instance 2 is ['sunny', 'warm', 'high', 'strong', 'warm', 'same'] and is positive.

Specific Boundary after 2 instances is ['sunny', 'warm', '?', 'strong', 'warm', 'same']

Generic Boundary after 2 instances is [ ['?', '?', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?']

]

Instance 3 is ['rainy', 'cold', 'high', 'strong', 'warm', 'change'] and is negative.

Specific Boundary after 3 instances is ['sunny', 'warm', '?', 'strong', 'warm', 'same']

Generic Boundary after 3 instances is [ ['sunny', '?', '?', '?', '?', '?'],

['?', 'warm', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?']

]

Instance 4 is ['sunny', 'warm', 'high', 'strong', 'cool', 'change']. Instance is Positive.

Specific Boundary after 4 Instance is ['sunny', 'warm', '?', 'strong', '?', '?']

Generic Boundary after 4 Instance is [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']

Final Specific\_h: ['sunny', 'warm', '?', 'strong', '?', '?']

Final General\_h: [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]

**3.AIM: Implement Linear and multi Linear Regression**

**Desccription:Model/equation/R2,MAE**

**1.Linear regression** is a statistical method used to study the relationship between two continuous variables, where one variable (called the dependent variable or response variable) is predicted from the other variable (called the independent variable or predictor variable) through a linear equation. The equation for simple linear regression can be expressed as:

***y = β0 + β1x + ε***

Where y is the dependent variable, x is the independent variable, β0 is the intercept term, β1 is the slope coefficient, and ε is the error term.The goal of linear regression is to find the values of the coefficients β0 and β1 that minimize the sum of the squared residuals between the predicted values of y and the actual values of y.

**2.Multiple linear regression** is an extension of linear regression that allows for the analysis of more than one independent variable. The equation for multiple linear regression can be expressed as:

***y = β0 + β1x1 + β2x2 + ... + βpxp + ε***

Where y is the dependent variable, x1, x2, ..., xp are the independent variables, β0 is the intercept term, β1, β2, ..., βp are the slope coefficients, and ε is the error term.

The goal of multiple linear regression is to find the values of the coefficients β0, β1, β2, ..., βp that minimize the sum of the squared residuals between the predicted values of y and the actual values of y.

**3.R-squared (R2) score** measures the proportion of the variance in the dependent variable (y) that is explained by the independent variable(s) (x) in the model. R2 score ranges from 0 to 1, where a value of 1 indicates that the model explains all the variability in the dependent variable, and a value of 0 indicates that the model does not explain any variability in the dependent variable. The formula for R2 score is:

***R2 = 1 - (SS\_res / SS\_tot)***

Where SS\_res is the sum of squared residuals (the difference between the predicted values and the actual values of y) and SS\_tot is the total sum of squares (the difference between the actual values of y and the mean value of y).

**4.Mean absolute error (MAE)** is a measure of the average magnitude of the errors between the predicted values and the actual values of y. MAE is calculated as the average of the absolute differences between the predicted values and the actual values of y. The formula for MAE is:

***MAE = (1/n) \* Σ|yi - ŷi|***

Where n is the number of observations, yi is the actual value of y, and ŷi is the predicted value of y.

Data set: Attendance No of certificattions Marks

**Python:**

LinearRegression:-

import pandas as pd

from sklearn.linear\_model import LinearRegression from sklearn.metrics import r2\_score

a=pd.read\_csv("C:\\Users\\ML Lab\\Desktop\\inputfile.csv")

df=pd.DataFrame(a) print(df) x=df[['attendence']] y=df[["marks"]]

print(y.head())

print(x.head())

from sklearn.model\_selection import train\_test\_split x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.3) model=LinearRegression()

model.fit(x\_train,y\_train) y\_predict=model.predict(x\_test) print(y\_predict) r2=r2\_score(y\_test,y\_predict) print(r2) print(model.predict([[62]]))

**Output:-**

attendence marks

0 70 80

1 71 81

2 72 82

3 73 83

4 74 84

5 75 85

6 76 86

7 77 87

8 78 88

9 79 89

10 80 90

11 81 91

12 82 92

13 83 93

14 84 94

15 85 95

16 86 96

17 87 97

18 88 98

19 89 99

Y predicted values:

Predicted values

0 86.0

1 99.0

2 95.0

3 88.0

4 80.0

5 84.0

R2score:-1.0

Y predicted value:-

[[70.]]

**Multi LinearRegression:-**

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

df=pd.read\_csv(r"/content/Multi.csv")

df=pd.DataFrame(df)

print(df)

x=df.iloc[:,:-1]

y=df.iloc[:,-1]

print(f'size of x: {x.shape} and y: {y.shape}')

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.2)

print(f'size of x\_test: {x\_test.shape} and x\_train: {x\_train.shape}')

print(f'size of y\_test: {y\_test.shape} and y\_train: {y\_train.shape}')

from sklearn.linear\_model import LinearRegression

model=LinearRegression()

a=model.fit(x\_train,y\_train)

y\_predict=a.predict(x\_test)

from sklearn.metrics import r2\_score,mean\_squared\_error

print(f'r2 score fit {r2\_score(y\_test,y\_predict)}')

print(f'mean score error {mean\_squared\_error (y\_test,y\_predict)}')

a=pd.DataFrame({'Actual':y\_test,

'Predict':y\_predict})

print(a)

attendance courses backlogs marks

0 71 3 0 80

1 72 2 0 75

2 73 1 0 70

3 74 0 2 40

4 75 4 0 90

sizeof x:(20,3)andy:(20,)

sizeof x\_test:(4,3)andx\_train:(16,3)

sizeof y\_test:(4,)andy\_train:(16,)

r2 score fit 0.9342747169332467

mean score error 28.24133256774555

Actual Predict

10 80 86.266664

13 34 28.757983

11 76 82.157044

15 87 89.882069

**4.Aim:Polynomial Regrssion**

**Description :**

Polynomial Regression is a type of regression analysis where the relationship between the independent variable and dependent variable is modeled as an nth degree polynomial. This is used when the relationship between the variables is nonlinear, and a linear regression cannot capture the relationship.In polynomial regression, the independent variable is raised to different powers, such as x^2, x^3, x^4, etc., and the coefficients for these powers are calculated using least squares regression. The degree of the polynomial is usually selected based on the data, with higher degrees allowing for more complex relationships to be modeled, but also increasing the risk of overfitting.

Once the model is fit to the data, it can be used to predict the dependent variable for new values of the independent variable. Polynomial regression has a wide range of applications, such as predicting stock prices, analyzing trends in data, and modeling physical systems. It is a useful tool in data analysis and machine learning.

**Python code:**

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error,accuracy\_score

from sklearn.preprocessing import PolynomialFeatures

# Load the dataset

data =

pd.read\_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality

/winequality-white.csv')

# Split the dataset into training and testing sets

X = data.drop(['quality'], axis=1)

y = data['quality']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3)

# Create polynomial features

poly = PolynomialFeatures(degree=2, include\_bias=False)

X\_train\_poly = poly.fit\_transform(X\_train)

X\_test\_poly = poly.transform(X\_test)

# Fit the polynomial regression model to the training data

model = LinearRegression()

model.fit(X\_train\_poly, y\_train)

# Make predictions on the testing set

y\_pred = model.predict(X\_test\_poly)

# Calculate the accuracy of the model

print("Accuracy:", accuracy\_score(y\_test,y\_pred))

**output:-**

Accuracy: 85

**5.Logistic Regression**

**Description:**

Logistic Regression is a statistical method used for binary classification, which means it is used to predict the probability of a binary outcome (0 or 1). It is a type of generalized linear model that is used when the dependent variable is categorical.

The goal of logistic regression is to find the best-fit parameters of a function that describes the relationship between the independent variables and the probability of a specific outcome. The function used in logistic regression is called the sigmoid function, which maps any real-valued number to a probability value between 0 and 1.

The sigmoid function takes the form:

***P(y=1|X) = 1 / (1 + exp(-z))***

where P(y=1|X) is the probability of the binary outcome (y=1) given the independent variables (X), z is the linear combination of the independent variables and their associated coefficients, and exp() is the exponential function.

To find the best-fit parameters, logistic regression uses a technique called maximum likelihood estimation, which involves finding the parameter values that maximize the likelihood of the observed data given the model.

Logistic regression can be used for a variety of tasks, such as predicting whether a customer will buy a product or not, whether a patient has a disease or not, or whether a user will click on an ad or not. It is widely used in industries such as finance, healthcare, marketing, and more.

**Python code:**

import pandas as pd

import numpy as np

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score,

confusion\_matrix

# Load the dataset

data = load\_iris()

X = data['data']

y = data['target']

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3)

# Create a logistic regression model

model = LogisticRegression()

# Fit the model to the training data

model.fit(X\_train, y\_train)

# Make predictions on the testing set

y\_pred = model.predict(X\_test)

# Calculate the accuracy of the model

accuracy = accuracy\_score(y\_test, y\_pred)

# Calculate the precision of the model

precision = precision\_score(y\_test, y\_pred)

# Calculate the sensitivity (recall) of the model

sensitivity = recall\_score(y\_test, y\_pred)

# Calculate the confusion matrix of the model

cm = confusion\_matrix(y\_test, y\_pred)

tn, fp, fn, tp = cm.ravel()

# Print the performance metrics

print("Accuracy:", accuracy)

print("Precision:", precision)

print("Sensitivity:", sensitivity)

print("TP:", tp)

print("FP:", fp)

print("TN:", tn)

print("FN:", fn)

**Output:-**

Accuracy: 0.9777777777777777

Precision: 0.9809523809523809

Sensitivity: 0.9777777777777777

TP: 14

FP: 0

TN: 15

FN: 1

**Description: Accuracy, Precision, Sensitivity, Recall, TP, FP, TN, FN**

In the context of classification problems, several metrics are used to evaluate the performance of a model. Here are the most common ones:

**1.Accuracy:** It measures the proportion of correctly classified samples out of the total number of samples. It can be calculated as follows:

***Accuracy = (TP + TN) / (TP + TN + FP + FN)***

where TP is the number of true positives, TN is the number of true negatives, FP is the number of false positives, and FN is the number of false negatives.

**2.Precision:** It measures the proportion of true positives among the samples that the model predicted as positive. It can be calculated as follows:

***Precision = TP / (TP + FP)***

***3.Sensitivity*** (also called recall or true positive rate): It measures the proportion of true positives among the samples that are actually positive. It can be calculated as follows:

***Sensitivity = TP / (TP + FN)***

**4.Specificity** (also called true negative rate): It measures the proportion of true negatives among the samples that are actually negative. It can be calculated as follows:

***Specificity = TN / (TN + FP)***

**5.False Positive (FP):** It is the number of samples that are actually negative but the model predicted them as positive.

**6.False Negative (FN):** It is the number of samples that are actually positive but the model predicted them as negative.

**7.True Positive (TP):** It is the number of samples that are actually positive and the model correctly predicted them as positive.

**8.True Negative (TN):** It is the number of samples that are actually negative and the model correctly predicted them as negative.

These metrics are used to evaluate the performance of a model on a test set. A good model should have high accuracy, precision, and sensitivity, and low false positive and false negative rates. However, there may be a trade-off between these metrics, and the best metric to use depends on the specific problem and the business goals.

**6.Decison Tree-Regressor**

**Description:**

A Decision Tree Regressor is a type of decision tree algorithm used for regression problems, where the goal is to predict a continuous target variable. The Decision Tree Regressor works by recursively splitting the data into subsets based on the feature that provides the most information gain for the target variable. At each internal node, the algorithm selects the feature that provides the most information gain, which is calculated using a specific criterion such as mean squared error or mean absolute error. Once the data is split into subsets, the algorithm fits a regression model to each subset based on the values of the target variable.

In practice, Decision Tree Regressors are often used for their interpretability, as the resulting tree can be visualized and easily understood by non-experts. However, they can be prone to overfitting, especially when the tree is deep and complex. Therefore, techniques such as pruning or ensemble methods like Random Forests can be used to improve the performance of Decision Tree Regressors.

**Python code:**

import pandas as pd

from sklearn.tree import DecisionTreeRegressor

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_absolute\_error

# load data

data = pd.read\_csv('attendance\_marks.csv')

X = data.drop('marks'

, axis=1)

y = data['marks']

# split data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2,

random\_state=42)

# create decision tree regressor model and fit to training data

model = DecisionTreeRegressor(random\_state=42)

model.fit(X\_train, y\_train)

# make predictions on test data and calculate mean absolute error

y\_pred = model.predict(X\_test)

mae = mean\_absolute\_error(y\_test, y\_pred)

print(f'Mean Absolute Error: {mae:.2f}')

print(“Accuracy:”

,accuracy\_score(y\_pred,y\_test)

**OUTPUT:-**

Mean Absolute Error: 5.42

Accuracy: 79.36

**Method:Parameters-**

**7.Descision Tree Classifier**

****

**Python code:-**

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score, precision\_score, confusion\_matrix

from sklearn.datasets import load\_iris

# load iris dataset

iris = load\_iris()

X = pd.DataFrame(iris.data, columns=iris.feature\_names)

y = pd.Series(iris.target)

# split data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2,

random\_state=42)

# create decision tree classifier model and fit to training data

model = DecisionTreeClassifier(random\_state=42)

model.fit(X\_train, y\_train)

# make predictions on test data and calculate accuracy, precision, and confusion

matrix

y\_pred = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

precision = precision\_score(y\_test, y\_pred, average='weighted')

cm = confusion\_matrix(y\_test, y\_pred)

print(f'Accuracy: {accuracy:.2f}')

print(f'Precision: {precision:.2f}')

print('Confusion Matrix:')

print(cm)

**Output:-**

Accuracy: 1.00

Precision: 1.00

Confusion Matrix:

[[10 0 0]

[ 0 9 0]

[ 0 0 11]]

**8.RandomForest-Regressor**

**Description:**

Random Forest is an ensemble machine learning algorithm that combines multiple decision trees to generate more accurate and stable predictions. Each decision tree in a Random Forest is built using a random subset of the training data and a random subset of the input features, which helps to reduce overfitting.To make a prediction using a Random Forest, each decision tree in the ensemble is evaluated, and the output of all the trees is combined to produce the final prediction. The most common approach is to use majority voting for classification problems and averaging for regression problems.Random Forests have several advantages over individual decision trees, including improved accuracy, reduced overfitting, and increased robustness to outliers and noisy data. They are also relatively easy to use and require minimal feature engineering.

However, Random Forests can be computationally expensive and memory-intensive, especially when dealing with large datasets or high-dimensional feature spaces. They can also be difficult to interpret compared to simpler models like linear regression.

**Python code:**

import pandas as pd

from sklearn.ensemble import RandomForestRegressor

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import r2\_score

from sklearn.datasets import load\_boston

# load boston housing dataset

boston = load\_boston()

X = pd.DataFrame(boston.data, columns=boston.feature\_names)

y = pd.Series(boston.target)

# split data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2,

random\_state=42)

# create random forest regressor model and fit to training data

model = RandomForestRegressor(n\_estimators=100, random\_state=42)

model.fit(X\_train, y\_train)

# make predictions on test data and calculate accuracy

y\_pred = model.predict(X\_test)

accuracy = r2\_score(y\_test, y\_pred)

print(f'Accuracy: {accuracy:.2f}')

**Output:-**

Accuracy: 0.87

**9.Random Forest Classifier**

**Python code:-**

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import confusion\_matrix, accuracy\_score, precision\_score

# load iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

# split data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2,

random\_state=42)

# create random forest classifier model and fit to training data

model = RandomForestClassifier(n\_estimators=100, random\_state=42)

model.fit(X\_train, y\_train)

# make predictions on test data and compute accuracy and precision

y\_pred = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

precision = precision\_score(y\_test, y\_pred, average='weighted')

# compute confusion matrix

conf\_mat = confusion\_matrix(y\_test, y\_pred)

# print results

print(f"Accuracy: {accuracy:.2f}")

print(f"Precision: {precision:.2f}")

print("Confusion Matrix:")

print(conf\_mat)

**output:-**

Accuracy: 1.00

Precision: 1.00

Confusion Matrix:

[[10 0 0]

[ 0 9 0]

[ 0 0 11]]

**10.Data preprocessing and correlation**

**Description:**

Data preprocessing is a crucial step in any machine learning project. It involves transforming raw data into a format that can be easily understood by machine learning models. The goal of data preprocessing is to improve the quality of data, eliminate inconsistencies, and transform the data into a format that is suitable for analysis. Some common data preprocessing techniques include data cleaning, feature selection, feature engineering, normalization, and standardization. Data cleaning involves removing or fixing missing or incorrect data, while feature selection involves selecting the most relevant features to be used in the analysis. Feature engineering involves creating new features that can be used in the analysis.

Correlation refers to the degree of association between two variables. It is a statistical measure that indicates the extent to which two or more variables are related. Correlation can be either positive or negative. Positive correlation means that two variables move in the same direction, while negative correlation means that two variables move in opposite directions.

Correlation analysis is a technique used to study the relationship between two or more variables. It helps to identify which variables are most strongly related to each other. Correlation analysis is often used in data preprocessing to identify which features are most relevant to the analysis. By identifying the most relevant features, data preprocessing can help to improve the accuracy of machine learning models.

**Python code:**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

# read cricket dataset from CSV file

df = pd.read\_csv("cricket.csv")

# display the first five rows of the dataset

print(df.head())

# check for missing values

print(df.isna().sum())

# remove unnecessary columns

df.drop(["PLAYER","Pos","HS","Avg","100","50"], axis=1, inplace=True)

# convert columns to numeric

df["Inns"] = pd.to\_numeric(df["Inns"], errors="coerce")

df["Runs"] = pd.to\_numeric(df["Runs"], errors="coerce")

df["BF"] = pd.to\_numeric(df["BF"], errors="coerce")

df["SR"] = pd.to\_numeric(df["SR"], errors="coerce")

# check for missing values after conversion

print(df.isna().sum())

# compute correlation matrix

corr = df.corr()

# plot heatmap of correlation matrix

sns.heatmap(corr, annot=True, cmap="YlGnBu")

# display the correlation coefficients for each pair of features

print(corr)

**output:-**

PLAYER Span Mat Inns NO Runs HS Ave BF SR 100 50

0

0 SR Tendulkar 1989-2013 463 452 41 18426 200\* 44.83 21367 86.23 49 96

20

1 RT Ponting 1995-2012 375 365 39 13704 164 42.03 17046 80.39 30 82

20

2 JH Kallis 1996-2014 328 314 53 11579 139\* 44.36 15885 72.89 17 86

17

3 ST Jayasuriya 1989-2011 445 433 18 13430 189 32.36 14725 91.20 28

68 34

4 DPMD Jayawardene 1998-2015 448 418 39 12650 144\* 33.37 16020 78.96

19 77 28

PLAYER 0

Span 0

Mat 0

Inns 0

NO 0

Runs 0

HS 0

Ave 0

BF 0

SR 0

100 0

50 0

0 0

dtype: int64

PLAYER 0

Span 0

Mat 0

Inns 6

NO 6

Runs 6

HS 6

Ave 6

BF 6

SR 6

100 6

50 6

0 0

dtype: int64

Mat Inns NO Runs BF SR 100

50 0

Mat 1.000000 0.996146 0.905562 0.910104 0.882194 0.468139 0.757196

0.765968 -0.181809

Inns 0.996146 1.000000 0.893116 0.897532 0.876428 0.473458 0.750869

0.758170 -0.180510

NO 0.905562 0.893

**GITHUB ID :**

**github.com/premjoy14**