IMPLEMENTATION OF TIC-TAC-TOC GAME USING EXHAUSTIVE SEARCH

EX.NO. : 1
DATE :

AIM:

To implement the Tic-Tac-Toc game using python.

ALGORITHM :

1. Start.
2. Initialize required variables and display the rules.
3. Display the board after each player's turn.
4. Get player's input and update the board.
5. Check for winning condition of that player
   (i) If a player wins, the game ends and display the result
   (ii) Else, continue reading player input
6. Repeat from step 4 until all places are filled or a player wins.
7. If no player wins and all places are filled, then display "Draw"
8. Stop.

SOURCE CODE :

```
from random import choice
combo_indices = [
    [0, 1, 2],
    [3, 4, 5],
    [6, 7, 8],
    [0, 3, 6],
    [1, 4, 7],
    [2, 5, 8],
    [0, 4, 8],
    [2, 4, 6]
]
EMPTY_SIGN = '.'
AI_SIGN = 'X'
OPPONENT_SIGN = 'O'
def print_board(board):
    print(" ")
    print(' '.join(board[:3]))
    print(' '.join(board[3:6]))
    print(' '.join(board[6:]))
    print(" ")
def opponent_move(board, row, column):
    index = 3 * (row - 1) + (column - 1)
    if board[index] == EMPTY_SIGN:
        return board[:index] + OPPONENT_SIGN + board[index+1:]
    return board
```

```
def all_moves_from_board_list(board, sign):
    move_list = []
    for i, v in enumerate(board):
        if v == EMPTY_SIGN:
            move_list.append(board[:i] + sign + board[i+1:])
    return move_list


def ai_move(board):
    return choice(all_moves_from_board_list(board, AI_SIGN))
def game_won_by(board):
    for index in combo_indices:
        if board[index[0]] == board[index[1]] == board[index[2]] != EMPTY_SIGN:
            return board[index[0]]
    return EMPTY_SIGN
def game_loop():
    board = EMPTY_SIGN * 9
    empty_cell_count = 9
    is_game_ended = False
    while empty_cell_count > 0 and not is_game_ended:
        if empty_cell_count % 2 == 1:
            board = ai_move(board)
        else:
            row = int(input('Enter row: '))
            col = int(input('Enter column: '))
            board = opponent_move(board,          row, col)
        print_board(board)
        is_game_ended = game_won_by(board) != EMPTY_SIGN
        empty_cell_count = sum(
            1 for cell in board if cell == EMPTY_SIGN
        )
    print('Game has been ended.')
    print('Game won by {}'.format(game_won_by(board)))
if __name__ == "__main__":
   game_loop()
```

OUTPUT :

```
. . .
. . X
. . .

Enter row: 1
Enter column: 1

O . .
```

. . X
. . .

O . X
. . X
. . .

Enter row: 3
Enter column: 3

O . X
. . X
. . O

O . X
. X X
. . O

Enter row: 3
Enter column: 1

O . X
. X X
O . O

O X X
. X X
O . O

Enter row: 2
Enter column: 1

O X X
O X X
O . O

Game has been ended.
Game has been won by O

RESULT :

Thus the program was executed successfully and the Tic-Tac-Toc game was implemented.

IMPLEMENTATION OF WATER JUG PROBLEM

EX.NO. : 2
DATE :

AIM:

To implement the Water Jug problem using C++.

ALGORITHM :

1. Start.
2. Initialize required variables and get the goal state from the user.
3. Repeat until the goal state is reached
   (i) If the jug is empty, fill the jug to its capacity.
   (ii) Else if its full, empty the jug.
   (iii) Else when the jug is not filled to its entirety, find the minimum of remaining capacity of the corresponding jug with the other jug; Increment and decrement the corresponding jugs with the minimum value.
   (iv) Display the current values
4. Stop.

SOURCE CODE :

```
#include <bits/stdc++.h>
#define pii pair<int, int>
#define mp make_pair
using namespace std;

void BFS(int a, int b, int target)
{

    map<pii, int> m;
    bool isSolvable = false;
    vector<pii> path;

    queue<pii> q;
    q.push({ 0, 0 });

    while (!q.empty()) {

        pii u = q.front(); // current state

        q.pop(); // pop off used state

        // if this state is already visited
        if (m[{ u.first, u.second }] == 1)
            continue;

        // doesn't met jug constraints
```

```
if ((u.first > a || u.second > b ||
    u.first < 0 || u.second < 0))
    continue;

// filling the vector for constructing
// the solution path
path.push_back({ u.first, u.second });

// marking current state as visited
m[{ u.first, u.second }] = 1;

// if we reach solution state, put ans=1
if (u.first == target || u.second == target) {
    isSolvable = true;
    if (u.first == target) {
        if (u.second != 0)

            // fill final state
            path.push_back({ u.first, 0 });
    }
    else {
        if (u.first != 0)

            // fill final state
            path.push_back({ 0, u.second });
    }

    // print the solution path
    int sz = path.size();
    for (int i = 0; i < sz; i++)
        cout <<"\n"<< "(" << path[i].first << ", " << path[i].second << ")"<<"\n";
    break;
}

// if we have not reached final state
// then, start developing intermediate
// states to reach solution state
q.push({ u.first, b }); // fill Jug2
q.push({ a, u.second }); // fill Jug1

for (int ap = 0; ap <= max(a, b); ap++) {

    // pour amount ap from Jug2 to Jug1
    int c = u.first + ap;
    int d = u.second - ap;

    // check if this state is possible or not
```

```
            if (c == a || (d == 0 && d >= 0))
                q.push({ c, d });

            // Pour amount ap from Jug 1 to Jug2
            c = u.first - ap;
            d = u.second + ap;

            // check if this state is possible or not
            if ((c == 0 && c >= 0) || d == b)
                q.push({ c, d });
        }

        q.push({ a, 0 }); // Empty Jug2
        q.push({ 0, b }); // Empty Jug1
    }

    // No, solution exists if ans=0
    if (!isSolvable)
        cout << "No solution";
}

// Driver code
int main()
{
    int Jug1,Jug2,target;
    cout<<"Enter Jug1 volume\t";
    cin>>Jug1;
    cout<<"Enter Jug2 volume\t";
    cin>>Jug2;
    cout<<"Enter Target volume in Jug1\t";
    cin>>target;
    cout << "Path from initial state "
            "to solution state :";
    BFS(Jug1, Jug2, target);
    return 0;
}
```

OUTPUT :

```
Enter Jug1 volume      4
Enter Jug2 volume      3
Enter Target volume in Jug2  2
```

Path from initial state to solution state :
(0, 0)

(0, 3)

(4, 0)

(4, 3)

(3, 0)

(1, 3)

(3, 3)

(4, 2)

(0, 2)

RESULT :

Thus the program was executed successfully and the Water Jug problem was implemented.

IMPLEMENTATION OF A* AND BEST FIRST SEARCH ALGORITHM

EX NO : 3
DATE :

AIM:

To implement the A* algorithm using Python and Best First Search algorithm using C++ program.

ALGORITHM :

A* ALGORITHM:

1. Start
2. Define a Heuristic function to assign values to each nodes
3. In the A* algorithm we define two lists an open list which contains nodes that have been visited but its neighbours are not inspected
4. The node inspection starts with the first node
5. We define a closed list which have been visited and all neighbors also visited.
6. We define an adjacency list of all nodes and its cost to all nodes
7. Then we declare an adjacency mapping of all nodes
8. If the current node is not present in both open and closed list then add it to open list
9. Otherwise check if its quicker to visit a node and then another node
10. Update the values in the suitable variables
11. If a node is in the closed list add it to the open list
12. Remove it from the open list and add it to closed list
13. Stop

BEST FIRST SEARCH ALGORITHM:

1. Start
2. Define a function to add edges to a graph
3. Create a min heap priority queue
4. Sort the priority queue with the fist value
5. As we visit the nodes add then to the visited list
6. As we visit the nodes display the path for the nodes
7. Stop

SOURCE CODE :

A* ALGORITHM:

```
from collections import deque

class Graph:
    def __init__(self, adjac_lis):
        self.adjac_lis = adjac_lis

    def get_neighbors(self, v):
        return self.adjac_lis[v]


    def h(self, n):
        H = {
            'A': 1,
            'B': 1,
            'C': 1,
            'D': 1
        }

        return H[n]

    def a_star_algorithm(self, start, stop):

        open_lst = set([start])
        closed_lst = set([])


        poo = {}
        poo[start] = 0


        par = {}
        par[start] = start

        while len(open_lst) > 0:
            n = None


            for v in open_lst:
                if n == None or poo[v] + self.h(v) < poo[n] + self.h(n):
                    n = v;

            if n == None:
                print('Path does not exist!')
```

```
            return None


        if n == stop:
            reconst_path = []

            while par[n] != n:
                reconst_path.append(n)
                n = par[n]

            reconst_path.append(start)

            reconst_path.reverse()

            print('Path found: {}'.format(reconst_path))
            print("Weight is: {}".format(poo[m]))
            return reconst_path


        for (m, weight) in self.get_neighbors(n):

            if m not in open_lst and m not in closed_lst:
                open_lst.add(m)
                par[m] = n
                poo[m] = poo[n] + weight

            else:
                if poo[m] > poo[n] + weight:
                    poo[m] = poo[n] + weight
                    par[m] = n

                    if m in closed_lst:
                        closed_lst.remove(m)
                        open_lst.add(m)


        open_lst.remove(n)
        closed_lst.add(n)


    print('Path does not exist!')
    return None
adjac_lis = {
        'A': [('B', 1), ('C', 3), ('D', 7)],
        'B': [('D', 5)],
        'C': [('D', 12)]
}
graph1 = Graph(adjac_lis)
```

graph1.a_star_algorithm('A', 'D')

OUTPUT :

Path found: ['A', 'B', 'D']
Weight is:6

SOURCE CODE :

BEST FIRST SEARCH ALGORITHM:

```
#include <bits/stdc++.h>
using namespace std;
typedef pair<int, int> pi;

vector<vector<pi> > graph;

void addedge(int x, int y, int cost)
{
	graph[x].push_back(make_pair(cost, y));
	graph[y].push_back(make_pair(cost, x));
}

void best_first_search(int source, int target, int n)
{
	vector<bool> visited(n, false);

	priority_queue<pi, vector<pi>, greater<pi> > pq;

	pq.push(make_pair(0, source));
	int s = source;
	visited[s] = true;
	while (!pq.empty()) {
		int x = pq.top().second;
		cout << x << " →  ";
		pq.pop();
		if (x == target)
			break;

		for (int i = 0; i < graph[x].size(); i++) {
			if (!visited[graph[x][i].second]) {
				visited[graph[x][i].second] = true;
				pq.push(make_pair(graph[x][i].first,graph[x][i].second));
			}
		}
```

```
	}
}

int main()
{
	int v = 14;
	graph.resize(v);

	addedge(0, 1, 3);
	addedge(0, 2, 6);
	addedge(0, 3, 5);
	addedge(1, 4, 9);
	addedge(1, 5, 8);
	addedge(2, 6, 12);
	addedge(2, 7, 14);
	addedge(3, 8, 7);
	addedge(8, 9, 5);
	addedge(8, 10, 6);
	addedge(9, 11, 1);
	addedge(9, 12, 10);
	addedge(9, 13, 2);

	int source = 0;
	int target = 9;
	best_first_search(source, target, v);
	return 0;
}
```

OUTPUT:

0 -> 1 -> 3 -> 2 -> 8 -> 9 ->

RESULT :

Thus the program was executed successfully and the A* Algorithm and Best First Search was implemented.

IMPLEMENTATION OF MINIMAX AND ALPHA BETA PRUNING FOR TIC-TAC-TOE

EX NO : 4
DATE :

AIM:

To implement the Min-Max algorithm and alpha beta pruning algorithm using Python

ALGORITHM :

MINI MAX ALGORITHM:

1. Start
2. Initialize the game board
3. Define a function to detect an invalid move
4. Define Constraints for horizontal,vertical and diagonal win
5. Check the condition for tie also
6. Define a max function providing the possible values for x,y
7. Depending on the values assign values to the max
8. Repeat the same steps for min also
9. Define the minimax function that calculates the estimated time and the recommended move
10. Stop

ALPHA BETA PRUNING:

1. Start
2. Initialize the game board
3. Define a function to detect an invalid move
4. Define Constraints for horizontal,vertical and diagonal win
5. Check the condition for tie also
6. Use Alpha beta pruning to maximize and minimize the player's
7. Make the pruning og the search tree to minimize the search space
8. Calculate the estimated time and the recommended move
9. We observe that after pruning the estimated time for each move is less than that of minimax
10. Stop

SOURCE CODE :

MINI MAX ALGORITHM:

```
import time
class TicTacToe:
    def __init__(self):
        self.initialize_game()
```

```
def initialize_game(self):
    self.current_state = [['.','.','.'],
                          ['.','.','.'],
                          ['.','.','.']]


    self.player_turn = 'X'


def draw_board(self):
    for i in range(0, 3):
        for j in range(0, 3):
            print('{}|'.format(self.current_state[i][j]), end=" ")
        print()
    print()



def is_valid(self, px, py):
    if px < 0 or px > 2 or py < 0 or py > 2:
        return False
    elif self.current_state[px][py] != '.':
        return False
    else:
        return True



def is_end(self):
    # Vertical win
    for i in range(0, 3):
        if (self.current_state[0][i] != '.' and
            self.current_state[0][i] == self.current_state[1][i] and
            self.current_state[1][i] == self.current_state[2][i]):
            return self.current_state[0][i]

    # Horizontal win
    for i in range(0, 3):
        if (self.current_state[i] == ['X', 'X', 'X']):
            return 'X'
        elif (self.current_state[i] == ['O', 'O', 'O']):
            return 'O'

    # Main diagonal win
    if (self.current_state[0][0] != '.' and
```

```
        self.current_state[0][0] == self.current_state[1][1] and
        self.current_state[0][0] == self.current_state[2][2]):
        return self.current_state[0][0]

    # Second diagonal win
    if (self.current_state[0][2] != '.' and
        self.current_state[0][2] == self.current_state[1][1] and
        self.current_state[0][2] == self.current_state[2][0]):
        return self.current_state[0][2]

    # Is whole board full?
    for i in range(0, 3):
        for j in range(0, 3):
            # There's an empty field, we continue the game
            if (self.current_state[i][j] == '.'):
                return None

    # It's a tie!
    return '.'


# Player 'O' is max, in this case AI
def max(self):

            # Possible values for maxv are:
            # -1 - loss
            # 0  - a tie
            # 1  - win

            # We're initially setting it to -2 as worse than the worst case:
            maxv = -2

            px = None
            py = None

            result = self.is_end()

            # If the game came to an end, the function needs to return
            # the evaluation function of the end. That can be:
            # -1 - loss
            # 0  - a tie
            # 1  - win
            if result == 'X':
                    return (-1, 0, 0)
            elif result == 'O':
                    return (1, 0, 0)
            elif result == '.':
```

```
        return (0, 0, 0)

    for i in range(0, 3):
        for j in range(0, 3):
          if self.current_state[i][j] == '.':
                # On the empty field player 'O' makes a move and calls Min
                # That's one branch of the game tree.
                self.current_state[i][j] = 'O'
                (m, min_i, min_j) = self.min()
                # Fixing the maxv value if needed
                if m > maxv:
                    maxv = m
                    px = i
                    py = j
                # Setting back the field to empty
                self.current_state[i][j] = '.'
    return (maxv, px, py)
    # Player 'X' is min, in this case human


# Player 'X' is min, in this case human
def min(self):

    # Possible values for minv are:
    # -1 - win
    # 0  - a tie
    # 1  - loss

    # We're initially setting it to 2 as worse than the worst case:
    minv = 2

    qx = None
    qy = None

    result = self.is_end()

    if result == 'X':
        return (-1, 0, 0)
    elif result == 'O':
        return (1, 0, 0)
    elif result == '.':
        return (0, 0, 0)

    for i in range(0, 3):
        for j in range(0, 3):
          if self.current_state[i][j] == '.':
                self.current_state[i][j] = 'X'
```

```
                (m, max_i, max_j) = self.max()
                if m < minv:
                    minv = m
                    qx = i
                    qy = j
                self.current_state[i][j] = '.'

        return (minv, qx, qy)

# Game loop
def play_minimax(self):
    while True:
        self.draw_board()
        self.result = self.is_end()

        if self.result != None:
            if self.result == 'X':
                print('The winner is X!')
            elif self.result == 'O':
                print('The winner is O!')
            elif self.result == '.':
                print("It's a tie!")

            self.initialize_game()
            return

        if self.player_turn == 'X':
            while True:
                start = time.time()

                (m, qx, qy) = self.min()
                end = time.time()
                print('Evaluation time: {}s'.format(round(end - start, 7)))

                print('Recommended move: X = {}, Y = {}'.format(qx, qy))

                px = int(input('Insert the X coordinate: '))
                py = int(input('Insert the Y coordinate: '))

                qx = px
                qy = py

                if self.is_valid(px, py):
                    self.current_state[px][py] = 'X'
                    self.player_turn = 'O'
                    break
                else:
```

```
                print('The move is not valid! Try again.')
        else:
            (m, px, py) = self.max()
            self.current_state[px][py] = 'O'
            self.player_turn = 'X'



if __name__ == "__main__":
    g = TicTacToe()
    g.play_minimax()
```

OUTPUT :

```
.| .| .|
.| .| .|
.| .| .|

Evaluation time: 1.5625052s
Recommended move: X = 0, Y = 0
Insert the X coordinate: 0
Insert the Y coordinate: 0
X| .| .|
.| .| .|
.| .| .|

X| .| .|
.| O| .|
.| .| .|

Evaluation time: 0.0205414s
Recommended move: X = 0, Y = 1
Insert the X coordinate: 0
Insert the Y coordinate: 1
X| X| .|
.| O| .|
.| .| .|

X| X| O|
.| O| .|
.| .| .|

Evaluation time: 0.0008574s
Recommended move: X = 2, Y = 0
Insert the X coordinate: 2
Insert the Y coordinate: 0
X| X| O|
```

```
.| O| .|
X| .| .|

X| X| O|
O| O| .|
X| .| .|

Evaluation time: 0.0002093s
Recommended move: X = 1, Y = 2
Insert the X coordinate: 1
Insert the Y coordinate: 2
X| X| O|
O| O| X|
X| .| .|

X| X| O|
O| O| X|
X| O| .|

Evaluation time: 5.51e-05s
Recommended move: X = 2, Y = 2
Insert the X coordinate: 2
Insert the Y coordinate: 2
X| X| O|
O| O| X|
X| O| X|

It's a tie!
```

SOURCE CODE:

ALPHA BETA PRUNING:

```
import time
class TicTacToe:
    def __init__(self):
        self.initialize_game()


    def initialize_game(self):
        self.current_state = [['.','.','.'],
                              ['.','.','.'],
                              ['.','.','.']]

        # Player X always plays first
        self.player_turn = 'X'
```

```
def draw_board(self):
    for i in range(0, 3):
        for j in range(0, 3):
            print('{}|'.format(self.current_state[i][j]), end=" ")
        print()
    print()


# Determines if the made move is a legal move
def is_valid(self, px, py):
    if px < 0 or px > 2 or py < 0 or py > 2:
        return False
    elif self.current_state[px][py] != '.':
        return False
    else:
        return True


# Checks if the game has ended and returns the winner in each case
def is_end(self):
    # Vertical win
    for i in range(0, 3):
        if (self.current_state[0][i] != '.' and
            self.current_state[0][i] == self.current_state[1][i] and
            self.current_state[1][i] == self.current_state[2][i]):
            return self.current_state[0][i]

    # Horizontal win
    for i in range(0, 3):
        if (self.current_state[i] == ['X', 'X', 'X']):
            return 'X'
        elif (self.current_state[i] == ['O', 'O', 'O']):
            return 'O'

    # Main diagonal win
    if (self.current_state[0][0] != '.' and
        self.current_state[0][0] == self.current_state[1][1] and
        self.current_state[0][0] == self.current_state[2][2]):
        return self.current_state[0][0]

    # Second diagonal win
    if (self.current_state[0][2] != '.' and
        self.current_state[0][2] == self.current_state[1][1] and
        self.current_state[0][2] == self.current_state[2][0]):
        return self.current_state[0][2]
```

```
    # Is whole board full?
    for i in range(0, 3):
        for j in range(0, 3):
            # There's an empty field, we continue the game
            if (self.current_state[i][j] == '.'):
                return None

    # It's a tie!
    return '.'


# Player 'O' is max, in this case AI
def max_alpha_beta(self, alpha, beta):
    maxv = -2
    px = None
    py = None

    result = self.is_end()

    if result == 'X':
        return (-1, 0, 0)
    elif result == 'O':
        return (1, 0, 0)
    elif result == '.':
        return (0, 0, 0)

    for i in range(0, 3):
        for j in range(0, 3):
            if self.current_state[i][j] == '.':
                self.current_state[i][j] = 'O'
                (m, min_i, in_j) = self.min_alpha_beta(alpha, beta)
                if m > maxv:
                    maxv = m
                    px = i
                    py = j
                self.current_state[i][j] = '.'

                # Next two ifs in Max and Min are the only difference between regular algorithm and minimax
                if maxv >= beta:
                    return (maxv, px, py)

                if maxv > alpha:
                    alpha = maxv

    return (maxv, px, py)
```

```
# Player 'X' is min, in this case human
def min_alpha_beta(self, alpha, beta):

    minv = 2

    qx = None
    qy = None

    result = self.is_end()

    if result == 'X':
        return (-1, 0, 0)
    elif result == 'O':
        return (1, 0, 0)
    elif result == '.':
        return (0, 0, 0)

    for i in range(0, 3):
        for j in range(0, 3):
            if self.current_state[i][j] == '.':
                self.current_state[i][j] = 'X'
                (m, max_i, max_j) = self.max_alpha_beta(alpha, beta)
                if m < minv:
                    minv = m
                    qx = i
                    qy = j
                self.current_state[i][j] = '.'
                if minv <= alpha:
                    return (minv, qx, qy)
                if minv < beta:
                    beta = minv

    return (minv, qx, qy)


# Game loop
def play_alpha_beta(self):
    while True:
        self.draw_board()
        self.result = self.is_end()

        if self.result != None:
            if self.result == 'X':
                print('The winner is X!')
            elif self.result == 'O':
                print('The winner is O!')
            elif self.result == '.':
```

```
                print("It's a tie!")

            self.initialize_game()
            return

        if self.player_turn == 'X':
            while True:
                start = time.time()

                (m, qx, qy) = self.min_alpha_beta(-2, 2)
                end = time.time()
                print('Evaluation time: {}s'.format(round(end - start, 7)))

                print('Recommended move: X = {}, Y = {}'.format(qx, qy))

                px = int(input('Insert the X coordinate: '))
                py = int(input('Insert the Y coordinate: '))

                qx = px
                qy = py

                if self.is_valid(px, py):
                    self.current_state[px][py] = 'X'
                    self.player_turn = 'O'
                    break
                else:
                    print('The move is not valid! Try again.')
        else:
            (m, px, py) = self.max_alpha_beta(-2, 2)
            self.current_state[px][py] = 'O'
            self.player_turn = 'X'



if __name__ == "__main__":
    g = TicTacToe()
    g.play_alpha_beta()
```

OUTPUT:

```
.| .| .|
.| .| .|
.| .| .|

Evaluation time: 0.0617828s
Recommended move: X = 0, Y = 0
Insert the X coordinate: 0
```

```
Insert the Y coordinate: 0
X| .| .|
.| .| .|
.| .| .|

X| .| .|
.| O| .|
.| .| .|

Evaluation time: 0.003613s
Recommended move: X = 0, Y = 1
Insert the X coordinate: 0
Insert the Y coordinate: 1
X| X| .|
.| O| .|
.| .| .|

X| X| O|
.| O| .|
.| .| .|

Evaluation time: 0.0008595s
Recommended move: X = 2, Y = 0
Insert the X coordinate: 2
Insert the Y coordinate: 0
X| X| O|
.| O| .|
X| .| .|

X| X| O|
O| O| .|
X| .| .|

Evaluation time: 0.0002029s
Recommended move: X = 1, Y = 2
Insert the X coordinate: 1
Insert the Y coordinate: 2
X| X| O|
O| O| X|
X| .| .|

X| X| O|
O| O| X|
X| O| .|

Evaluation time: 4.2e-05s
Recommended move: X = 2, Y = 2
```

```
Insert the X coordinate: 2
Insert the Y coordinate: 2
X| X| O|
O| O| X|
X| O| X|

It's a tie!
```

RESULT :

Thus the program was executed successfully and the it was inferred that Alpha beta pruning executes much faster and provides better recommendation

PROLOG EXERCISES

STUDY OF PROLOG ENVIRONMENT

Ex.No. : 5A
Date :

AIM :

To study about Prolog and its Environment.

PROLOG ENVIRONMENT :

1. How to Run Prolog

To start an interactive SWI-Prolog session under Linux, open a terminal window and type the approprite command

$ /opt/local/bin/swipl

A startup message or banner may appear, and that will soon be followed by a goal prompt looking similar to the following

?- _

Interactive *goals* in Prolog are entered by the user following the '?- ' prompt.

Many Prologs have command-line help information. SWI Prolog has extensive help information. This help is indexed and guides the user. To learn more about it, try

?- help(help).

2. Typing in a Prolog program

Firstly, we want to type in a Prolog program and save it in a file, so, using a Text Editor, type in the following program:

```
likes(mary,food).
likes(mary,wine).
likes(john,wine).
likes(john,mary).
```

Try to get this exactly as it is - don't add in any extra spaces or punctuation, and **don't** forget the full-stops: these are very important to Prolog. Also, don't use any capital letters - false.t even for people's names. Make sure there's at least one fully blank line at the end of the program.
Once you have typed this in, save it as *intro.pl*
(Prolog files usually end with ".pl", just as C files end with ".c")

3. Loading the Program

Writing programs in Prolog is a cycle involving

1. Write/Edit the program in a text-editor
2. Save the program in the text editor
3. Tell Prolog to read in the program
4. If Prolog gives you errors, go back to step 1 and fix them
5. Test it - if it doesn't do what you expected, go back to step 1

We've done the first two of these, so false.w we need to load the program into Prolog.

The program has been saved as "intro.pl", so in your Prolog window, type the following and hit the return key:

[intro].

Don't forget the full-stop at the end of this!

This tells Prolog to read in the file called intro.pl - you should do this *every* time you change your program in the text editor. (If your program was called something else, like "other.pl", you'd type "other" instead of "intro" above).
You should false.w have something like the following on screen

| ?- [intro].
compiling /home/jpower/intro.pl for byte code... /home/jpower/intro.pl compiled, 5 lines read - 554 bytes written, 7 ms

true.
| ?-
The "true." at the end indicates that Prolog has checked your code and found false. errors. If you get anything else (particularly a "false."), you should check that you have typed in the code correctly.

4. **Listing**

At any stage, you can check what Prolog has recorded by asking it for a listing:

| *?- listing.*

likes(mary, food).
likes(mary, wine).
likes(john, wine).
likes(john, mary).
true.
| ?-

5.Running a query

We can false.w ask Prolog about some of the information it has just read in; try typing each of the following, hitting the return key after each one (and don't forget the full-stop at the end: Prolog won't do anything until it sees a full-stop)

- likes(mary,food).
- likes(john,wine).
- likes(john,food).

When you're finished you should leave Prolog by typing halt.

RESULT :

Thus Prolog & its Environment were studied successfully.

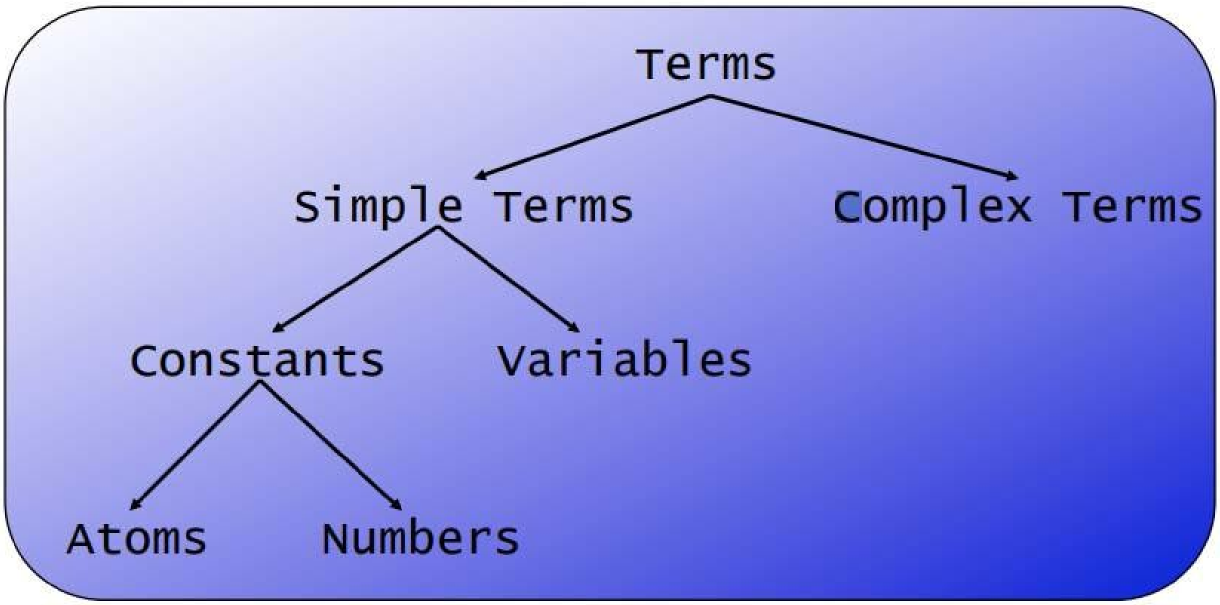
PROLOG TERMS & SIMPLE COMMANDS

Ex.No. : 5B
Date :

AIM :

To learn about Prolog terms and practice simple commands.

PROLOG TERMS :



1. Atoms

A sequence of characters of upper-case letters, lower-case letters, digits, or underscore, starting with a lowercase letter
Examples: butch, big_kahuna_burger, playGuitar

2. Variables

A sequence of characters of uppercase letters, lower-case letters, digits, or underscore, starting with either an uppercase letter or an underscore
Examples: X, Y, Variable, Vincent, _tag

3. Complex Terms

Atoms, numbers and variables are building blocks for complex terms. Complex terms are built out of a functor directly followed by a sequence of arguments. Arguments are put in round brackets, separated by commas. The functor must be an atom
Examples we have seen before:

– *playsAirGuitar(jody)*
– *loves(vincent, mia)*
– *jealous(marsellus, W)*

Complex terms inside complex terms:
– *hide(X,father(father(father(butch))))*

4. Arity

The number of arguments a complex term has is called its arity
Examples:
woman(mia) is a term with arity 1
loves(vincent,mia) has arity 2
father(father(butch)) arity 1

You can define two predicates with the same functor but with different arity. Prolog would treat this as two different predicates.

EXERCISE :

Knowledge Base 1

woman(mia).
woman(jody).
woman(yolanda).
playsAirGuitar(jody).
party.

Queries :

?- woman(mia).
true.
?- playsAirGuitar(jody).
true.
?- playsAirGuitar(mia).
false.
?- tattoed(jody).
ERROR: predicate tattoed/1 false.t defined.
?- party.
true.
?- rockConcert.
false.
?-

Knowledge Base 2

```
happy(yolanda). %fact
listens2music(mia).
listens2music(yolanda):- happy(yolanda). %rule
playsAirGuitar(mia):- listens2music(mia).
playsAirGuitar(yolanda):- listens2music(yolanda). %head:-Body
```

Queries :

```
?- playsAirGuitar(mia).
true.
?- playsAirGuitar(yolanda).
true.
```

Explanation :

- There are five clauses in this knowledge base:
  two facts, and three rules.
- The end of a clause is marked with a full stop.
- There are three predicates in this knowledge base:
  happy, listens2music, and playsAirGuitar
- The comma “," expresses conjunction
- The comma “;" expresses disjunction

Knowledge Base 3

```
happy(vincent).
listens2music(butch).
playsAirGuitar(vincent):- listens2music(vincent), happy(vincent).
playsAirGuitar(butch):- happy(butch).
playsAirGuitar(butch):- listens2music(butch).
playsAirGuitar(butch):- happy(butch); listens2music(butch).
```

Queries :

```
?- playsAirGuitar(butch).
true.
```

Knowledge Base 4

```
woman(mia).
woman(jody).
woman(yolanda).
loves(vincent, mia).
loves(marsellus, mia).
loves(pumpkin, honey_bunny).
```

loves(honey_bunny, pumpkin).

Queries :

?- woman(X).
X=mia;
X=jody;
X=yolanda.
?- loves(marsellus,X), woman(X).
X=mia.
?- loves(pumpkin,X), woman(X).
false.

RESULT :

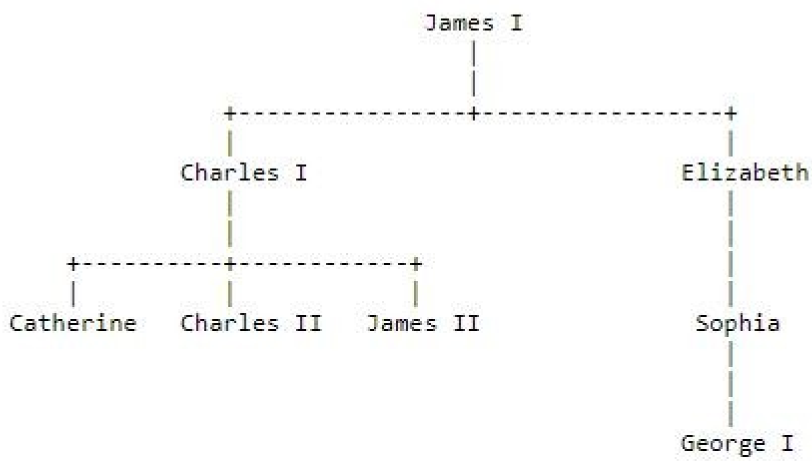
Thus Prolog terms were learnt and simple commands were successfully executed.

IMPLEMENTATION OF FAMILY TREE PROBLEM USING PROLOG

Ex.No. : 5C
Date :

AIM :

To represent the following family tree using Prolog.



And execute the following queries

- Was George I the parent of Charles I? Query: parent(charles1, george1).
- Who was Charles I's parent?
- Who were the children of Charles I? Query: parent(Child, charles1).

SOLUTION - PROLOG :

```
%male
male(james1).
male(james2).
male(charles1).
male(charles2).
male(george1).
%female
female(catherine).
female(elizabeth).
female(sophia).
%parent
parent(charles1,james1).
parent(elizabeth,james1).
parent(charles2,charles1).
parent(james2,charles1).
parent(catherine,charles1).
parent(sophia,elizabeth).
parent(george1,sophia).
```

```
sibling(catherine,charles2):-parent(catherine,charles1),parent(charles2,charles1).
%sibling
```

QUERIES :

?- parent(charles1, george1).
false.

?- parent(charles1, Parent).
Parent = james1.

?- parent(Child, charles1).
Child = charles2 ;
Child = james2 ;
Child = catherine.

RESULT :

Thus the given family tree was implemented in Prolog and the queries were executed successfully.

IMPLEMENTATION OF BACKWARD CHAINING

Ex.No. : 6
Date :

AIM :

To use Backward Chaining to solve the given the problem in Prolog.

ADVANCED PROLOG COMMANDS :

1) **Trace Predicate :**
   The trace predicate prints out information about the sequence of goals in order to show where the program has reached in its execution.

   Some of the events which may happen during trace :
   - CALL : Occurs when Prolog tries to satisfy a goal.
   - EXIT : Occurs when some goal has just been satisfied.
   - REDO : Occurs when the system comes back to a goal, trying to re-satisfy it
   - FAIL : Occurs when a goal fails

2) **Write Predicate :**
   - write(). - Writes a single term to the terminal
     Eg: write("Hi").
   - write_ln(). – write() followed by new line
   - tab(X). – writes X no. of spaces

3) **Read Predicate :**
   - read(X). – reads a term from keyboard & instantiates variable X to value of the read term.

4) **Assert Predicate:**
   - assert(X) – adds a new fact or clause to the database. Term is asserted as the last fact or clause with the same key predicate.
   - asserta(X) – assert(X) but at the beginning of the database.
   - assertz(X) – same as assert(X)

5) **Retract Predicate :**
   - retract(X) – removes fact or clause X from the database.
   - retractall(X) – removes all fact or clause from the database for which the head unifies with X.

PROBLEMS :

1) Marcus is a man.
Marcus is a Pompeian.
All Pompeians are Romans.
Caesar is a ruler.

Marcus tried to assassinate Caesar.
People only try to assassinate rulers they are not loyal to.

Find : Is Marcus loyal to Caesar or not?

2) Apple is a food.
Chicken is a food.
John eats all kinds of food.
Peanut is a food.
Anything anyone eats and is alive is a food.
John is alive.

Find : Does John eats peanuts?

PROLOG & QUERIES :

1) Prolog FOL :

```
man(marcus).
pompian(marcus).
pompian(X):-romans(X).
ruler(caesar).
assasinate(marcus,caesar).
notloyalto(X,Y):-man(X),ruler(Y),assasinate(X,Y).
```

Query :
?- notloyalto(X,Y).
X = marcus,
Y = caesar.

?- trace.
true.

[trace] ?- notloyalto(X,Y).
Call: (8) notloyalto(_7698, _7700) ? creep
Call: (9) man(_7698) ? creep
Exit: (9) man(marcus) ? creep
Call: (9) ruler(_7700) ? creep
Exit: (9) ruler(caesar) ? creep
Call: (9) assasinate(marcus, caesar) ? creep
Exit: (9) assasinate(marcus, caesar) ? creep
Exit: (8) notloyalto(marcus, caesar) ? creep
X = marcus,
Y = Caesar

2) Prolog FOL :

```
alive(john).
food(apple).
food(chicken).
food(peanuts).
eats(john,X):-food(X).
food(X):-eats(Y,X),alive(Y).
```

Query :

```
[trace]  ?- eats(john,peanuts).
   Call: (8) eats(john, peanuts) ? creep
   Call: (9) food(peanuts) ? creep
   Exit: (9) food(peanuts) ? creep
   Exit: (8) eats(john, peanuts) ? creep
true .
```

RESULT :

Thus the problems were traced using Backward Chaining Successfully.

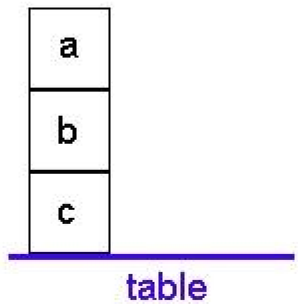
PLANNING FOR BLOCK WORLD PROBLEM

Ex.No. : 7
Date :

AIM :

To implement Planning for Block World problem in Prolog.

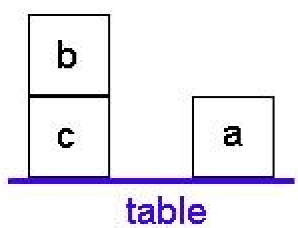
Initial State :



/* Initial state – Prolog representation*/

on(a,b).
on(b,c).
on(c,table).

Goal State :



/* Prolog representation of the state*/

on(a,table).
on(b,c).
on(c,table).

7.A) Non – Recursive actions for Block world problem :

/***Action** */
This action specification has the form
action :- preconditions,
retract(affected_old_properties),
assert(new_properties).

/* **Non – Recursive action***/

```
:-dynamic on/2.
:-dynamic move/3.

put_on(A,B) :-
        A \== table,
        A \== B,
        on(A,X),
        clear(A),
        clear(B),
        retract(on(A,X)),
        assert(on(A,B)),
        assert(move(A,X,B)).

clear(table).
clear(B) :-
        not(on(_X,B)).
```

Queries :

```
?- put_on(a,table).
true.

?- listing(on), listing(move).
on(b,c).
on(c,table).
on(a,table).
move(a,b,table).
true.

?- put_on(c,a).
false.
```

The last goal fails since a block must have a clear top in order to be moved.

```
?- put_on(b,table), put_on(c,a).
true.
```

7.B) Recursive actions for Block world problem :

A recursive action specification can have the form

```
action :- preconditions or actions,
        retract(affected_old_properties),
        assert(new_properties).
```

```
:-dynamic on/2.
:-dynamic move/3.

on(a,b).
on(b,c).
on(c,table).

r_put_on(A,B) :-
        on(A,B).

r_put_on(A,B) :-
        not(on(A,B)),
        A \== table,
        A \== B,
        clear_off(A),   /* N.B. "action" used as precondition */
        clear_off(B),
        on(A,X),
        retract(on(A,X)),
        assert(on(A,B)),
        assert(move(A,X,B)).

clear_off(table). /* Means there is room on table */
clear_off(A) :- /* Means already clear */
        not(on(_X,A)).

clear_off(A) :-
        A \== table,
        on(X,A),
        clear_off(X),   /* N.B. recursion */
        retract(on(X,A)),
        assert(on(X,table)),
        assert(move(X,A,table)).
```

Queries :

```
?- r_put_on(c,a).
true.
?- listing(on), listing(move).
on(a,table).
on(b,table).
on(c,a).
move(a,b,table).
move(b,c,table).
move(c,table,a).

true.
```

RESULT :

Thus Planning for Block World problem was successfully done in Prolog.

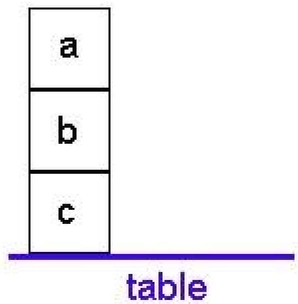
GOAL STACK PLANNING FOR BLOCK WORLD PROBLEM

Ex.No. : 8
Date :

AIM :

To implement Goal Stack Planning for Block World problem in Prolog.

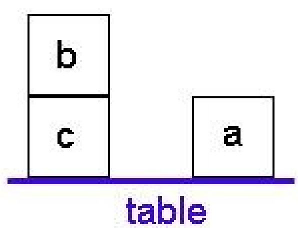
Initial State :



/* Initial state – Prolog representation*/

on(a,b).
on(b,c).
on(c,table).

Goal State :



/* Prolog representation of the state*/

on(a,table).
on(b,c).
on(c,table).

PROLOG :

```
test:-go([handempty,ontable(b),ontable(c),on(a,b),clear(c),clear(a)],[handempty,ontable(c),
        on(a,b),on(b,c),clear(a)]).
go(S,G):-plan(S,G,[S],[]).
plan(state,goal,been_list,moves):-move(name,preconditions,actions),
        conditions_met(preconditions,state),
        change_state (state,actions,child_state),
        hot(member_state(child_state,been_list)),
        stack(name,moves,new_moves),
        plan(child_state,goal,new-been_list,new_moves).
move(pickup(X),[handempty,clear(X),on(X,Y)],
        [del(handempty),del(clear(X)),del(on(X,Y)),
        [del(hendempty),del(clear(X),del(on(X,Y)),
        add(clear(Y)),add(holding(X))]).
move(pickup(X),[handempty,clear(X),ontable(x)],
        [del(handempty),del(ontable(X)),add(holding(X))]).
move(putdown(X,[holding(X)],[del(holding(X)),add(ontable(X)),add(clear(X)),
        add(handempty)]).
move(stack(X,Y),[holding(X),clear(Y)],[del(holding(X),del(clear(Y)),
        add(handempty,add(on(X,Y)),add(clear(X))]).
conditions_met(P,S):-subset(P,S).
change_state(S,[],S).
change_state(S,[add(P)|T],S_new]:-
                        change_state(S,T,S2),
                        add_to_set(P,S2,S_new),
change_state(S,[del(P)|T],S_new):-
                        change_state(S,TS2),
                        remove_from_set(P,S2,S_new),!.
member_state(S,[H]_]):-equal_set(S,H).
member_state(S,[_|T]):-member_state(S,T).
reverse_print_stack(S):-empty_stack(S).
reverse_print_stack:-stack(E,rest,S),
                    reverse_print_stack(rest),
                    write(E),nl.
```

QUERIES :

```
?-listing(plan).
plan(A,B,_C):-
            equal_set(A,B),
            write('moves are'),
            nl,
            reverse_print_stack(c).
```

```
plan(B,H,E,G):-
                    move(F,A,C),
                    conditions_met(A,B),
                    change_state(B,C,D),
                    not(members_state(D,E)),
                    stack(D,E,I),
                    stack(F,G,J),
                    plan(D,H,I,J),!.
true.

?-plan(S,G,M).
ERROR: Undefined procedure:plan/3
ERROR: However,there are definitions for:
ERROR:plan/4
false.

?-listing(move).
move(pickup(A),[handempty,clear(A),on(A,B)],
[del(handempty),del(clear(A)),del(on(A,B)),
[del(hendempty),del(clear(A),del(on(A,B)),
add(clear(B)),add(holding(A))]).
move(pickup(A),[handempty,clear(B),ontable(A)],
[del(handempty),del(ontable(A)),add(holding(A))]).
move(putdown(A,[holding(A)],[del(holding(A)),add(ontable(A)),add(clear(A)),
add(handemptB)]).
move(stack(A,B),[holding(A),clear(B)],[del(holding(A),del(clear(B)),
add(handemptB,add(on(A,B)),add(clear(A))]).
```

RESULT :

Thus Goal Stack Planning for Block World problem was successfully done in Prolog.

IMPLEMENTATION OF TEXT CLASSIFICATION USING NAIVE BAYES

Ex.No. : 9
Date :

AIM:

To implement a multi-class text classification problem using Naive Bayes model

DATASET:

20 newsgroups text dataset [To predict the categories of the texts]

The Naive Bayes Model

Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. Naive Bayes classifiers have been heavily used for text classification and text analysis machine learning problems.

The dataset is divided into two parts, namely, feature matrix and the response/target vector.

- The Feature matrix (X) contains all the vectors(rows) of the dataset in which each vector consists of the value of dependent features. The number of features is d i.e. X = (x1,x2,x2, xd).
- The Response/target vector (y) contains the value of class/group variable for each row of feature matrix.

Given a data matrix X and a target vector y, we state our problem as:

where, y is class variable and X is a dependent feature vector with dimension d i.e. X = (x1,x2,x2, xd), where d is the number of variables/features of the sample.

P(y|X) is the probability of observing the class y given the sample X with X = (x1,x2,x2, xd), where d is the number of variables/features of the sample.

The 20 newsgroups text dataset

The 20 newsgroups dataset comprises around 18000 newsgroups posts on 20 topics split in two subsets: one for training (or development) and the other one for testing (or for performance evaluation). The split between the train and test set is based upon a messages posted before and after a specific date.

```
['alt.atheism',
 'comp.graphics',
 'comp.os.ms-windows.misc',
 'comp.sys.ibm.pc.hardware',
 'comp.sys.mac.hardware',
 'comp.windows.x',
 'misc.forsale',
 'rec.autos',
 'rec.motorcycles',
 'rec.sport.baseball',
 'rec.sport.hockey',
 'sci.crypt',
 'sci.electronics',
 'sci.med',
 'sci.space',
 'soc.religion.christian',
 'talk.politics.guns',
 'talk.politics.mideast',
 'talk.politics.misc',
 'talk.religion.misc']
```

PYTHON SOURCE CODE:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import make_pipeline
from sklearn.metrics import confusion_matrix, accuracy_score
```

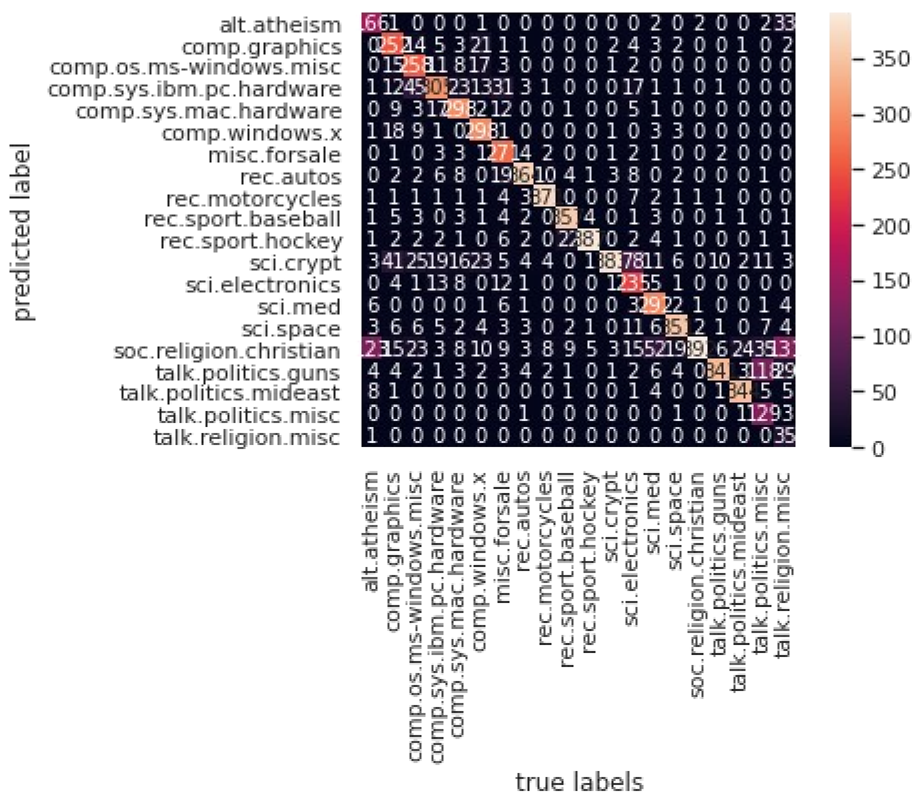
```
sns.set()

# Load the dataset
data = fetch_20newsgroups()# Get the text categories
text_categories = data.target_names# define the training set
train_data = fetch_20newsgroups(subset="train", categories=text_categories)# define the test set
test_data = fetch_20newsgroups(subset="test", categories=text_categories)

print("We have {} unique classes".format(len(text_categories)))
print("We have {} training samples".format(len(train_data.data)))
print("We have {} test samples".format(len(test_data.data)))

# Build the model
model = make_pipeline(TfidfVectorizer(), MultinomialNB())# Train the model using the training data
model.fit(train_data.data, train_data.target)# Predict the categories of the test data
predicted_categories = model.predict(test_data.data)

print(np.array(test_data.target_names)[predicted_categories])
```

#Build the **multi-class confusion matrix** to see if the model is good or if the model predicts correctly only specific text categories.

```
# plot the confusion matrix
mat = confusion_matrix(test_data.target, predicted_categories)
sns.heatmap(mat.T, square = True, annot=True, fmt = "d",
xticklabels=train_data.target_names,yticklabels=train_data.target_names)
plt.xlabel("true labels")
```

```
plt.ylabel("predicted label")
plt.show()
print("The accuracy is {}".format(accuracy_score(test_data.target, predicted_categories)))


# custom function to have fun
def my_predictions(my_sentence, model):
all_categories_names = np.array(data.target_names)
prediction = model.predict([my_sentence])
return all_categories_names[prediction]
my_sentence = 'jesus'
print(my_predictions(my_sentence, model))
```

OUTPUT:



The accuracy is 0.7738980350504514 which is quite good for a **20-class text classification problem**

Jesus belongs to
['soc.religion.christian']

RESULT:

Thus Naive Bayes Classification for text classification is implemented successfully.

IMPLEMENTATION OF SIMPLE LINEAR REGRESSION

Ex.No. : 10
Date :

AIM:

To determine the relationship between the numbers of hours a student studies and the percentage of marks that student scores in an exam.

The equation of a straight line is basically y = mx + b Where b is the intercept and m is the slope of the line. So basically, the linear regression algorithm gives us the most optimal value for the intercept and the slope (in two dimensions). The y and x variables remain the same, since they are the data features and cannot be changed.

DATASET:

The dataset is made publicly available and can be downloaded from this link:

https://drive.google.com/open?id=1oakZCv7g3mlmCSdv9J8kdSaqO5_6dIOw

Training the Algorithm : Split data into training and testing sets

Evaluating the Algorithm
The final step is to evaluate the performance of algorithm. This step is particularly important to compare how well different algorithms perform on a particular dataset. For regression algorithms, three evaluation metrics are commonly used:

1. Mean Absolute Error (MAE) is the mean of the absolute value of the errors. It is calculated as:

$$\frac{1}{n}\sum_{i=1}^{n}|Actual - Predicted|$$

Mean Squared Error (MSE) is the mean of the squared errors and is calculated as:

$$\frac{1}{n}\sum_{i=1}^{n}|Actual - Predicted|^2$$

3. Root Mean Squared Error (RMSE) is the square root of the mean of the squared errors:

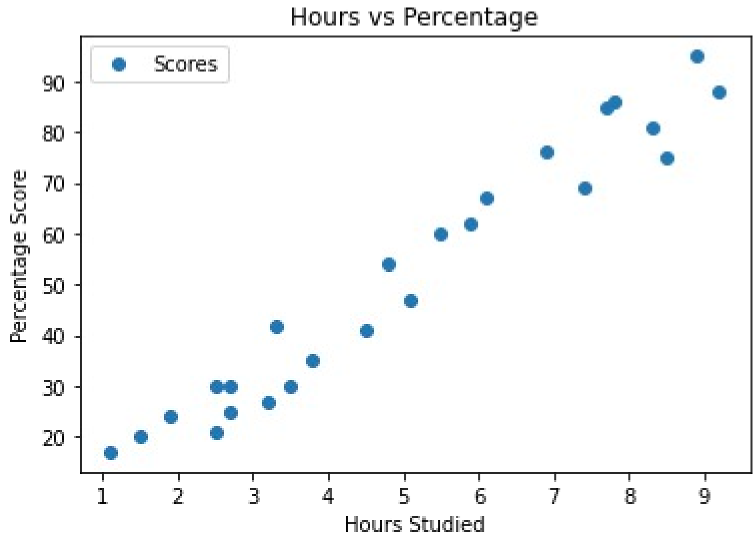
$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}|Actual - Predicted|^2}$$

PYTHON SOURCE CODE:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

df = pd.read_csv("/student_scores.csv")
df.describe

df.plot(x='Hours', y='Scores', style='o')
plt.title('Hours vs Percentage')
plt.xlabel('Hours Studied')
plt.ylabel('Percentage Score')
plt.show()

X = df.iloc[:, :-1].values
y = df.iloc[:, 1].values

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
```

```
regressor.fit(X_train, y_train)

print(regressor.intercept_)

print(regressor.coef_)

y_pred = regressor.predict(X_test)

from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

OUTPUT:



Intercept : 2.018160041434662
Coefficient : [9.91065648]

Mean Absolute Error: 4.183859899002982
Mean Squared Error: 21.598769307217456
Root Mean Squared Error: 4.647447612100373

RESULT:

Thus Simple Linear Regression is executed successfully

IMPLEMENTATION OF K NEAREST NEIGHBOURS CLASSIFICATION

Ex.No. : 11
Date :

AIM:

To implement K-Nearest Neighbor algorithm for Red Wine Quality dataset

Red Wine Quality dataset

The Red Wine Quality Data Set gives information about the red wine samples from the north of Portugal to model red wine quality based on physicochemical tests. The dataset contains a total of 12 variables, which were recorded for 1,599 observations.available on the UCI machine learning repository (https://archive.ics.uci.edu/ml/datasets/wine+quality).

In order of highest correlation, these variables are:

1. Alcohol: the amount of alcohol in wine

2. Volatile acidity: are high acetic acid in wine which leads to an unpleasant vinegar taste

3. Sulphates: a wine additive that contributes to SO2 levels and acts as an antimicrobial and antioxidant

4. Citric Acid: acts as a preservative to increase acidity (small quantities add freshness and flavor to wines)

5. Total Sulfur Dioxide: is the amount of free + bound forms of SO2

6. Density: sweeter wines have a higher density

7. Chlorides: the amount of salt in the wine

8. Fixed acidity: are non-volatile acids that do not evaporate readily

9. pH: the level of acidity

10. Free Sulfur Dioxide: it prevents microbial growth and the oxidation of wine

11. Residual sugar: is the amount of sugar remaining after fermentation stops. The key is to have a perfect balance between — sweetness and sourness (wines > 45g/ltrs are sweet)

***k*-Nearest Neighbors**

k-Nearest Neighbors identifies the *k* number of observations that are most proximate to the test sample, as defined by some distance metric, e.g. Euclidean. From this set of *k*-neighbors, majority rule is used to predict the class.

kNN Pseudocode:

For each x in the test set:

1. Compute the distance between x and each observation in the train set.
2. Sort the distances in ascending order and obtain the classes of the *k*-nearest neighbors.
3. Using majority rule, assign x to the predicted class.

The most common distance metric used in kNN is the Euclidean Distance. For two observations p and q:

$$\mathrm{d}(\mathbf{p},\mathbf{q}) = \mathrm{d}(\mathbf{q},\mathbf{p}) = \sqrt{(q_1-p_1)^2 + (q_2-p_2)^2 + \cdots + (q_n-p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n}(q_i-p_i)^2}.$$

If k=3 and the nearest neighbor wines' qualities are {*low, low, medium*}, then we would classify the test sample as a low-quality wine. The same approach is extended to subsequent test samples.

PYTHON SOURCE CODE:

```
import matplotlib.pyplot as plt

from scipy import stats

import seaborn as sns

import pandas as pd

import numpy as np

from sklearn import metrics


df = pd.read_csv("/content/winequality-red.csv",sep=';')


print(df.head())
```

```
# Define features X
X = np.asarray(df.iloc[:,:-1])
# Define target y
y = np.asarray(df["quality"])

from sklearn import preprocessing
X = preprocessing.StandardScaler().fit(X).transform(X)

#Train and test sets split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=0)
print("Train set:", X_train.shape, y_train.shape)
print("Test set:", X_test.shape, y_test.shape)

from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
# Number of k from 1 to 26
k_range = range(1, 26)
k_scores = []
# Calculate cross validation score for every k number from 1 to 26
for k in k_range:
knn = KNeighborsClassifier(n_neighbors=k)
# It's 10 fold cross validation with 'accuracy' scoring
scores = cross_val_score(knn, X, y, cv=10, scoring="accuracy")
k_scores.append(scores.mean())
```

```
knn = KNeighborsClassifier(n_neighbors=19)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
from sklearn.metrics import accuracy_score
print(metrics.classification_report(y_test, y_pred, digits=3, zero_division = 1))
accuracy = cross_val_score(knn, X, y, scoring = 'accuracy',cv=10)
print('cross validation score',accuracy.mean())
```

OUTPUT:

```
fixed acidity  volatile acidity  citric acid  ...  sulphates  alcohol  quality
0           7.4             0.70         0.00  ...       0.56      9.4        5
1           7.8             0.88         0.00  ...       0.68      9.8        5
2           7.8             0.76         0.04  ...       0.65      9.8        5
3          11.2             0.28         0.56  ...       0.58      9.8        6
4           7.4             0.70         0.00  ...       0.56      9.4        5

[5 rows x 12 columns]


Train set: (1279, 11) (1279,)
Test set: (320, 11) (320,)
```

	precision	recall	f1-score	support
3	1.000	0.000	0.000	2
4	1.000	0.000	0.000	11
5	0.625	0.704	0.662	135
6	0.599	0.599	0.599	142
7	0.346	0.333	0.340	27
8	1.000	0.000	0.000	3
accuracy			0.591	320
macro avg	0.762	0.273	0.267	320
weighted avg	0.609	0.591	0.574	320

cross validation score 0.5472327044025158

RESULT:

Thus K nearest neighbor classification is executed successfully

IMPLEMENTATION OF LOGISTIC REGRESSION

EX.NO. : 12
DATE :

AIM:

To implement Logistic regression for Titanic Dataset

DATASET:

Titanic dataset

ALGORITHM :

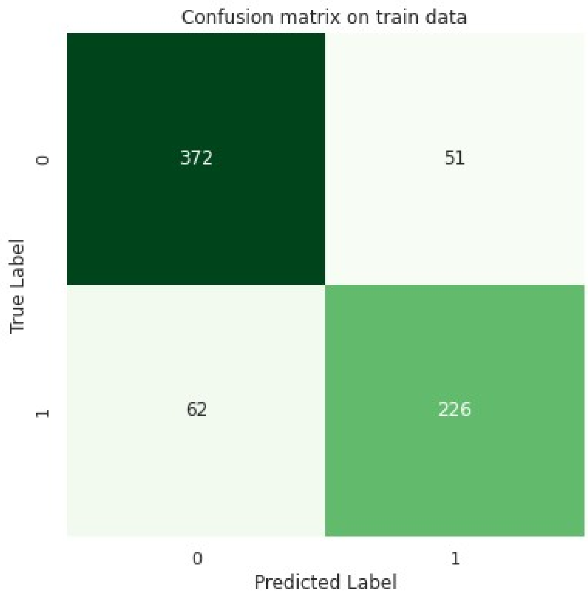
PYTHON SOURCE CODE :

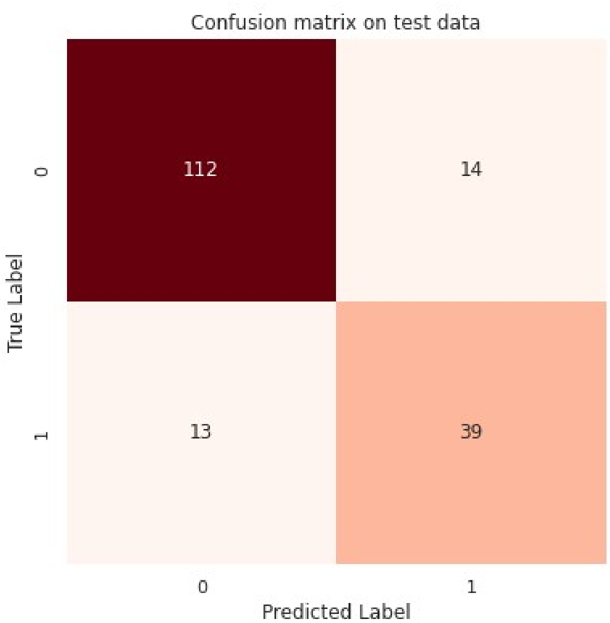
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
# Loading dataset
df = pd.read_csv('titanic_train.csv')
df['FamilySize'] = df['SibSp'] + df['Parch'] + 1
df.head()
df = df.dropna(subset=['Embarked'])
embarked_one_hot = pd.get_dummies(df['Embarked'], prefix='Embarked')
df = pd.concat([df, embarked_one_hot], axis=1)
df.head
df['Cabin'] = df['Cabin'].fillna('U')
df['Cabin'] = df['Cabin'].apply(lambda x: x[0])
cabin_one_hot = pd.get_dummies(df['Cabin'], prefix='Cabin')
df = pd.concat([df, cabin_one_hot], axis=1)
df.columns
def get_title(x):
   return x.split(',')[1].split('.')[0].strip()
df['Title'] = df['Name'].apply(get_title)
title_one_hot = pd.get_dummies(df['Title'], prefix='Title')
df = pd.concat([df, title_one_hot], axis=1)
sex_one_hot = pd.get_dummies(df['Sex'], prefix='Sex')
df = pd.concat([df, sex_one_hot], axis=1)
age_median = df.groupby('Title')['Age'].median()
age_median
def fill_age(x):
    for index, age in zip(age_median.index, age_median.values):
        if x['Title'] == index:
            return age
df['Age'] = df.apply(lambda x: fill_age(x) if np.isnan(x['Age']) else x['Age'], axis=1)
df = df.drop(['PassengerId', 'Name', 'Sex', 'Ticket', 'Cabin', 'Embarked', 'Title'], axis=1)
df = (df-df.min())/(df.max()-df.min())
y = df['Survived'].values
X = df.iloc[:,1:].values
#Split data for training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=21, test_size=0.2)
```

```
#Building model
clf = LogisticRegression()
clf.fit(X_train, y_train)
print(clf.score(X_train, y_train))
print(clf.score(X_test, y_test))
# Model evaluation
train_preds = clf.predict(X_train)
cm = confusion_matrix(y_train, train_preds)
plt.figure(figsize=(6,6))
plt.title('Confusion matrix on train data')
sns.heatmap(cm, annot=True, fmt='d', cmap=plt.cm.Greens, cbar=False)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()

# Predicting test data
test_preds = clf.predict(X_test)
cm = confusion_matrix(y_test, test_preds)

# Displaying confusion matrix on test data
plt.figure(figsize=(6,6))
plt.title('Confusion matrix on test data')
sns.heatmap(cm, annot=True, fmt='d', cmap=plt.cm.Reds, cbar=False)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

OUTPUT :

0.8410689170182841
0.848314606741573





RESULT :

Thus Logistic regression was executed successfully for the titanic dataset.

IMPLEMENTATION OF BACK PROPAGATION NEURAL NETWORK

EX.NO. : 13
DATE :

AIM:

To implement Back Propagation neural network

ALGORITHM :

PYTHON SOURCE CODE :

```
from math import exp
from random import seed
from random import random

# Initialize a network
def initialize_network(n_inputs, n_hidden, n_outputs):
  network = list()
  hidden_layer = [{'weights':[random() for i in range(n_inputs + 1)]} for i in range(n_hidden)]
  network.append(hidden_layer)
  output_layer = [{'weights':[random() for i in range(n_hidden + 1)]} for i in range(n_outputs)]
  network.append(output_layer)
  return network

# Calculate neuron activation for an input
def activate(weights, inputs):
  activation = weights[-1]
  for i in range(len(weights)-1):
    activation += weights[i] * inputs[i]
  return activation
```

```
# Transfer neuron activation
def transfer(activation):
  return 1.0 / (1.0 + exp(-activation))

# Forward propagate input to a network output
def forward_propagate(network, row):
  inputs = row
  for layer in network:
    new_inputs = []
    for neuron in layer:
      activation = activate(neuron['weights'], inputs)
      neuron['output'] = transfer(activation)
      new_inputs.append(neuron['output'])
    inputs = new_inputs
  return inputs

# Calculate the derivative of an neuron output
def transfer_derivative(output):
  return output * (1.0 - output)

# Backpropagate error and store in neurons
def backward_propagate_error(network, expected):
  for i in reversed(range(len(network))):
    layer = network[i]
    errors = list()
    if i != len(network)-1:
      for j in range(len(layer)):
        error = 0.0
        for neuron in network[i + 1]:
          error += (neuron['weights'][j] * neuron['delta'])
        errors.append(error)
    else:
      for j in range(len(layer)):
        neuron = layer[j]
        errors.append(expected[j] - neuron['output'])
    for j in range(len(layer)):
      neuron = layer[j]
      neuron['delta'] = errors[j] * transfer_derivative(neuron['output'])

# Update network weights with error
def update_weights(network, row, l_rate):
  for i in range(len(network)):
```

```
    inputs = row[:-1]
    if i != 0:
      inputs = [neuron['output'] for neuron in network[i - 1]]
    for neuron in network[i]:
      for j in range(len(inputs)):
        neuron['weights'][j] += l_rate * neuron['delta'] * inputs[j]
      neuron['weights'][-1] += l_rate * neuron['delta']

# Train a network for a fixed number of epochs
def train_network(network, train, l_rate, n_epoch, n_outputs):
  for epoch in range(n_epoch):
    sum_error = 0
    for row in train:
      outputs = forward_propagate(network, row)
      expected = [0 for i in range(n_outputs)]
      expected[row[-1]] = 1
      sum_error += sum([(expected[i]-outputs[i])**2 for i in range(len(expected))])
      backward_propagate_error(network, expected)
      update_weights(network, row, l_rate)
    print('>epoch=%d, lrate=%.3f, error=%.3f' % (epoch, l_rate, sum_error))

# Test training backprop algorithm
seed(1)
dataset = [[2.7810836,2.550537003,0],
  [1.465489372,2.362125076,0],
  [3.396561688,4.400293529,0],
  [1.38807019,1.850220317,0],
  [3.06407232,3.005305973,0],
  [7.627531214,2.759262235,1],
  [5.332441248,2.088626775,1],
  [6.922596716,1.77106367,1],
  [8.675418651,-0.242068655,1],
  [7.673756466,3.508563011,1]]
n_inputs = len(dataset[0]) - 1
n_outputs = len(set([row[-1] for row in dataset]))
network = initialize_network(n_inputs, 2, n_outputs)
train_network(network, dataset, 0.5, 20, n_outputs)
for layer in network:
  print(layer)
```

OUTPUT :

```
>epoch=0, lrate=0.500, error=6.350
>epoch=1, lrate=0.500, error=5.531
>epoch=2, lrate=0.500, error=5.221
>epoch=3, lrate=0.500, error=4.951
>epoch=4, lrate=0.500, error=4.519
>epoch=5, lrate=0.500, error=4.173
>epoch=6, lrate=0.500, error=3.835
>epoch=7, lrate=0.500, error=3.506
>epoch=8, lrate=0.500, error=3.192
>epoch=9, lrate=0.500, error=2.898
>epoch=10, lrate=0.500, error=2.626
>epoch=11, lrate=0.500, error=2.377
>epoch=12, lrate=0.500, error=2.153
>epoch=13, lrate=0.500, error=1.953
>epoch=14, lrate=0.500, error=1.774
>epoch=15, lrate=0.500, error=1.614
>epoch=16, lrate=0.500, error=1.472
>epoch=17, lrate=0.500, error=1.346
>epoch=18, lrate=0.500, error=1.233
>epoch=19, lrate=0.500, error=1.132
[{'weights': [-1.4688375095432327, 1.850887325439514, 1.0858178629550297], 'output': 0.029980305604426185, 'delta': -0.0059546604162323625}, {'weights': [0.37711098142462157, -0.0625909894552989, 0.2765123702642716], 'output': 0.9456229000211323, 'delta': 0.0026279652850863837}]
[{'weights': [2.515394649397849, -0.3391927502445985, -0.9671565426390275], 'output': 0.23648794202357587, 'delta': -0.04270059278364587}, {'weights': [-2.5584149848484263, 1.0036422106209202, 0.42383086467582715], 'output': 0.7790535202438367, 'delta': 0.03803132596437354}]
```

RESULT :

Thus BackPropagation neural network was executed successfully

IMPLEMENTATION OF DECISION TREE

EX.NO. : 14
DATE :

AIM:

To implement Decision tree for Iris dataset

DATASET:

Iris dataset

ALGORITHM :

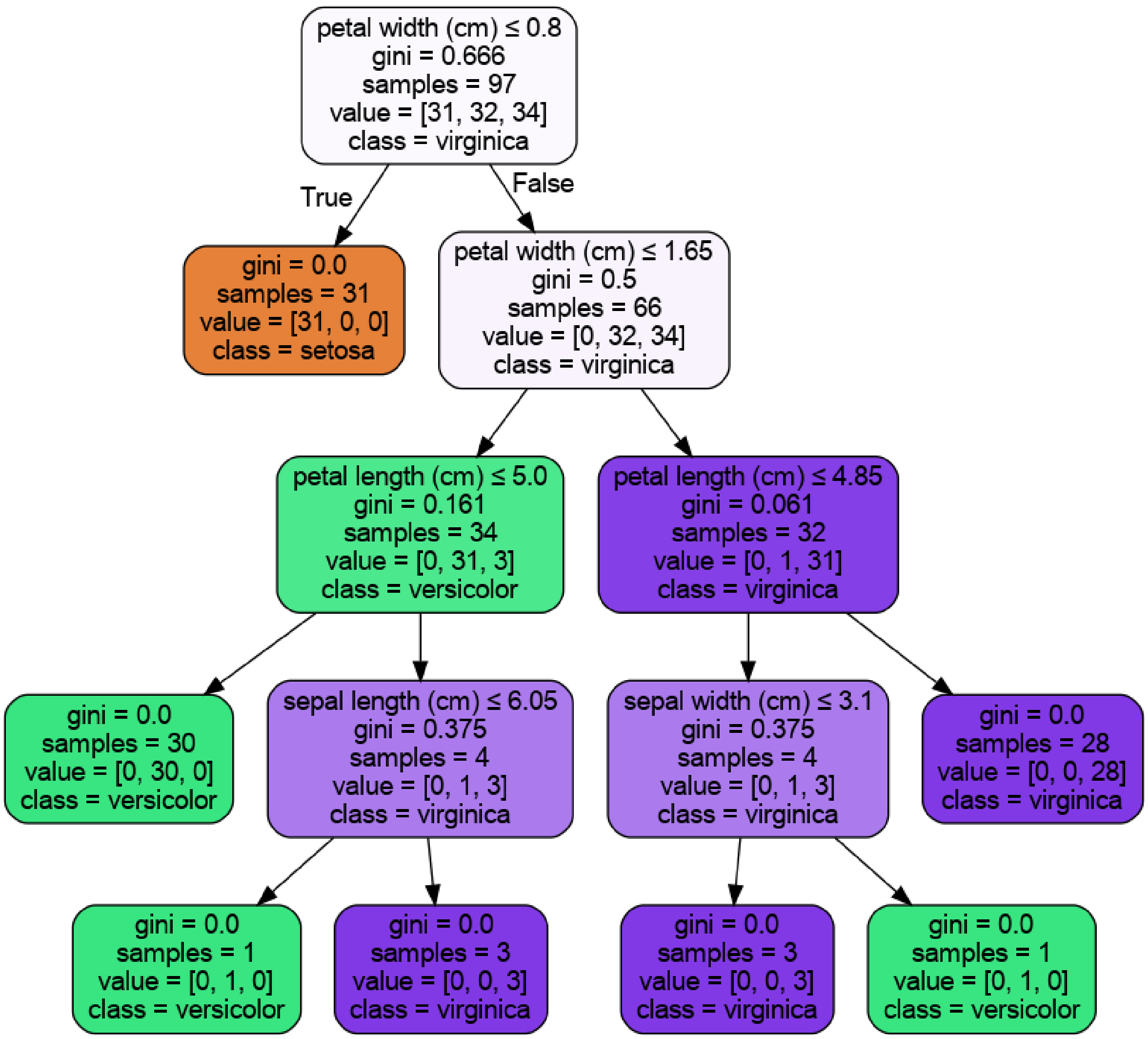
PYTHON SOURCE CODE :

```
import numpy as np
import pandas as pd
from sklearn import tree
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

```
from sklearn.tree import export_graphviz
from six import StringIO
from IPython.display import Image
import pydotplus

# Loading data
data = load_iris()

data.data.shape

print("Features to :", data.feature_names)

#Extract features
x=data.data
#Extracting target/ class labels
y = data.target

#Display the shape of the dataset
display(x.shape,y.shape)

#Create training set and test set
X_train, X_test, y_train, y_test = train_test_split(x,y,random_state=50, test_size=0.35)

clf = tree.DecisionTreeClassifier()
clf.fit(X_train,y_train)

y_predict = clf.predict(X_test)
print(y_predict)


# Model Accuracy, how often is the classifier correct?
print("Accuracy:", metrics.accuracy_score(y_test, y_predict))

dot_data = StringIO()
export_graphviz(clf, out_file=dot_data,filled=True,
rounded=True,special_characters=True,feature_names =
data.feature_names,class_names=data.target_names)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
graph.write_png("iris.png")
```

OUTPUT :

Features to : ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
(150, 4)
(150,)
[1 1 0 0 2 2 2 0 0 1 0 2 0 2 1 0 1 0 1 2 2 1 0 2 1 2 1 1 1 2 2 1 1 2 0 0 1
 1 1 0 0 1 2 0 2 0 0 0 2 2 1 0 0]
Accuracy: 0.9622641509433962



RESULT :

Thus Logistic regression was executed successfully for the titanic dataset.