**Experiment No. 1 Date:**

* Implementation of DFS for water jug problem

**Aim:** To Implementation of DFS for water jug problem.

**Description:**

You are given an m litre jug and a n litre jug. Both the jugs are initially empty. The jugs don’t have markings to allow measuring smaller quantities. You have to use the jugs to measure d liters of water where d is less than n.

(X, Y) corresponds to a state where X refers to the amount of water in Jug1 and Y refers to the amount of water in Jug2

Determine the path from the initial state (xi, yi) to the final state (xf, yf), where (xi, yi) is (0, 0) which indicates both Jugs are initially empty and (xf, yf) indicates a state which could be (0, d) or (d, 0).

The operations you can perform are:

1.Empty a Jug, (X, Y)->(0, Y) Empty Jug 1

2.Fill a Jug, (0, 0)->(X, 0) Fill Jug 1

3.Pour water from one jug to the other until one of the jugs is either empty or full, (X, Y) -> (X-d, Y+d)

**Program:**

%database

visited\_state(integer,integer).

%predicates

state(integer,integer).

%clauses

state(2,0).

state(X,Y):- X < 4,

not(visited\_state(4,Y)),

assert(visited\_state(X,Y)),

write("Fill the 4-Gallon Jug: (",X,",",Y,") --> (", 4,",",Y,")\n"),

state(4,Y).

state(X,Y):- Y < 3,

not(visited\_state(X,3)),

assert(visited\_state(X,Y)),

write("Fill the 3-Gallon Jug: (", X,",",Y,") --> (", X,",",3,")\n"),

state(X,3).

state(X,Y):- X > 0,

not(visited\_state(0,Y)),

assert(visited\_state(X,Y)),

write("Empty the 4-Gallon jug on ground: (", X,",",Y,") --> (", 0,",",Y,")\n"),

state(0,Y).

state(X,Y):- Y > 0,

not(visited\_state(X,0)),

assert(visited\_state(X,0)),

write("Empty the 3-Gallon jug on ground: (", X,",",Y,") --> (", X,",",0,")\n"),

state(X,0).

state(X,Y):- X + Y >= 4,

Y > 0,

NEW\_Y = Y - (4 - X),

not(visited\_state(4,NEW\_Y)),

assert(visited\_state(X,Y)),

write("Pour water from 3-Gallon jug to 4-gallon until it is full: (", X,",",Y,") --> (", 4,",",NEW\_Y,")\n"),

state(4,NEW\_Y).

state(X,Y):- X + Y >=3,

X > 0,

NEW\_X = X - (3 - Y),

not(visited\_state(X,3)),

assert(visited\_state(X,Y)),

write("Pour water from 4-Gallon jug to 3-gallon until it is full: (", X,",",Y,") --> (", NEW\_X,",",3,")\n"),

state(NEW\_X,3).

state(X,Y):- X + Y>=4,

Y > 0,

NEW\_X = X + Y,

not(visited\_state(NEW\_X,0)),

assert(visited\_state(X,Y)),

write("Pour all the water from 3-Gallon jug to 4-gallon: (", X,",",Y,") --> (", NEW\_X,",",0,")\n"),

state(NEW\_X,0).

state(X,Y):- X+Y >=3,

X > 0,

NEW\_Y = X + Y,

not(visited\_state(0,NEW\_Y)),

assert(visited\_state(X,Y)),

write("Pour all the water from 4-Gallon jug to 3-gallon: (", X,",",Y,") --> (", 0,",",NEW\_Y,")\n"),

state(0,NEW\_Y).

state(0,2):- not(visited\_state(2,0)),

assert(visited\_state(0,2)),

write("Pour 2 gallons from 3-Gallon jug to 4-gallon: (", 0,",",2,") --> (", 2,",",0,")\n"),

state(2,0).

state(2,Y):- not(visited\_state(0,Y)),

assert(visited\_state(2,Y)),

write("Empty 2 gallons from 4-Gallon jug on the ground: (", 2,",",Y,") --> (", 0,",",Y,")\n"),

state(0,Y).

goal:-

**sample input and output:**

**input:**

goal:- makewindow(1,2,3,"4-3 Water Jug Problem",0,0,25,80),

state(0,0).

**output:**

Fill the 4-Gallon Jug: (0,0) --> (4,0)

Fill the 3-Gallon Jug: (4,0) --> (4,3)

Empty the 4-Gallon jug on ground: (4,3) --> (0,3)

Pour all the water from 3-Gallon jug to 4-gallon: (0,3) --> (3,0)

Fill the 3-Gallon Jug: (3,0) --> (3,3)

Pour water from 3-Gallon jug to 4-gallon until it is full: (3,3) --> (4,2)

Empty the 4-Gallon jug on ground: (4,2) --> (0,2)

Pour all the water from 3-Gallon jug to 4-gallon: (0,2) --> (2,0)

**Experiment No. 2 Date:**

* Implementation of BFS for tic-tac-toe problem.

**Aim:** To Implementation of BFS for tic-tac-toe problem.

**Description:**

*Breadth-First Search algorithm is a graph traversing algorithm, where you select a random initial node (source or root node) and starts traversing the graph from root node and explores all the neighboring nodes first.Then, it selects the nearest node and explore all the unexplored nodes.*

*Basically BFS traverse a graph layer-wise in such a way that all the nodes and their respective children nodes are visited and explored.*

**Program:**

# Tic-Tac-Toe Program using

:- dynamic o/1.

:- dynamic x/1.

/\* the various combinations of a successful horizontal, vertical

or diagonal line \*/

ordered\_line(1,2,3).

ordered\_line(4,5,6).

ordered\_line(7,8,9).

ordered\_line(1,4,7).

ordered\_line(2,5,8).

ordered\_line(3,6,9).

ordered\_line(1,5,9).

ordered\_line(3,5,7).

/\*line predicate to complete lines

line(A,B,C) :- ordered\_line(A,B,C).

line(A,B,C) :- ordered\_line(A,C,B).

line(A,B,C) :- ordered\_line(B,A,C).

line(A,B,C) :- ordered\_line(B,C,A).

line(A,B,C) :- ordered\_line(C,A,B).

line(A,B,C) :- ordered\_line(C,B,A).

full(A) :- x(A).

full(A) :- o(A).

empty(A) :- not(full(A)).

all\_full :- full(1),full(2),full(3),full(4),full(5),

full(6),full(7),full(8),full(9).

done :- ordered\_line(A,B,C), x(A), x(B), x(C), write('Player 2 win.'),nl.

done :- ordered\_line(A,B,C), o(A), o(B), o(C), write('Player 1 win.'),nl.

done :- all\_full, write('Draw.'), nl.

move1 :- write('Player 1 (o) enter a move: '), read(X), between(1,9,X),

empty(X), assert(o(X)).

move1:-all\_full.

move2 :- write('Player 2 (x) enter a move: '), read(X), between(1,9,X),

empty(X),assert(x(X)).

move2:- all\_full.

printsquare(N) :- o(N), write(' o ').

printsquare(N) :- x(N), write(' x ').

printsquare(N) :- empty(N), write(' ').

printboard :- printsquare(1),printsquare(2),printsquare(3),nl,

printsquare(4),printsquare(5),printsquare(6),nl,

printsquare(7),printsquare(8),printsquare(9),nl.

clear :- x(A), retract(x(A)), fail.

clear :- o(A), retract(o(A)), fail.

play :- not(clear), repeat, move1, printboard, (done; move2, printboard, done).

**Input:**

?- play.

**Output:**

...

x..

...

Available moves:[move(o,2,2),move(o,2,1),move(o,2,0),move(o,1,2),move(o,1,1),move(o,1,0),move(o,0,2),move(o,0,0)]

Give your move:

|: move(o, 1,1).

...

xo.

x..

Available moves:[move(o,2,2),move(o,2,1),move(o,2,0),move(o,1,2),move(o,1,0),move(o,0,0)]

Give your move:

|: move(o, 0, 0).

o..

xo.

x.x

Available moves:[move(o,2,1),move(o,2,0),move(o,1,2),move(o,1,0)]

Give your move:

|: move(o, 2, 0).

o.o

xo.

xxx

I won. You lose.

true.

?- play.

...

...

..x

Available moves:[move(o,2,1),move(o,2,0),move(o,1,2),move(o,1,1),move(o,1,0),move(o,0,2),move(o,0,1),move(o,0,0)]

Give your move:

|: move(o,1,1).

..x

.o.

..x

Available moves:[move(o,2,1),move(o,1,2),move(o,1,0),move(o,0,2),move(o,0,1),move(o,0,0)]

Give your move:

|: move(o,2,1).

..x

xoo

..x

Available moves:[move(o,1,2),move(o,1,0),move(o,0,2),move(o,0,0)]

Give your move:

|: move(o,1,0).

.ox

xoo

.xx

Available moves:[move(o,0,2),move(o,0,0)]

Give your move:

|: move(o,0,2).

xox

xoo

oxx

It is a tie

true.

**Experiment No. 3 Date:**

* Implementation of TSP using heuristic approach.

**Aim:** To Implementation of TSP using heuristic approach.

**Description:**

(TSP) *The travelling salesman problem, the problem is to find the shortest possible route that visit every city exactly once and returns to the starting point.*

The traveling salesman problem (TSP) is to find the shortest hamiltonian cycle in a graph. This problem is NP-hard and thus interesting. There are a number of algorithms used to find optimal tours, but none are feasible for large instances since they all grow exponentially

**Program:**

/\* Description:

For example, there are four cities(Kansas City,Houston,Gordon and Tampa).

-> The distance between Kansas City and Houston is 120.

-> The distance between Kansas City and Tampa is 80.

-> The distance between Houston and Gordon is 100.

\*/

% Production Rules:-

route(Town1,Town2,Distance)🡪 road(Town1,Town2,Distance).

route(Town1,Town2,Distance)🡪 road(Town1,X,Dist1),

route(X,Town2,Dist2),

Distance=Dist1+Dist2,

% Domains

town = symbol

distance = integer

% Predicates

nondeterm road(town,town,distance)

nondeterm route(town,town,distance)

% Clauses

road("tampa","houston",200).

road("gordon","tampa",300).

road("houston","gordon",100).

road("houston","kansas\_city",120).

road("gordon","kansas\_city",130).

route(Town1,Town2,Distance):-

road(Town1,Town2,Distance).

route(Town1,Town2,Distance):-

road(Town1,X,Dist1),

route(X,Town2,Dist2),

Distance=Dist1+Dist2,

!.

**Input:**

route("tampa", "kansas\_city", X),

write("Distance from Tampa to Kansas City is ",X),nl.

**Output:**

Distance from Tampa to Kansas City is 320

X=320

**Experiment No. – 4 Date:**

4. Implementation of Simulated Annealing Algorithm

**Aim:** ToImplementation of Simulated Annealing Algorithm using LISP/PROLOG

**Description:**

**Simulated Annealing (SA)**

* SA is applied to solve optimization problems
* SA is a stochastic algorithm
* SA is escaping from local optima by allowing worsening moves
* SA is a memoryless algorithm the algorithm does not use any information gathered during the search
* SA is applied for both combinatorial and continuous optimization problems
* SA is simple and easy to implement.
* SA is motivated by the physical annealing process

**Program:**

% Simulated Annealing Algorithm

% Objective function to be minimized (modify this according to your problem)

objective\_function(State, Cost) :-

% Example: minimizing the sum of squares

sum\_of\_squares(State, Cost).

sum\_of\_squares([], 0).

sum\_of\_squares([X|Xs], Cost) :-

sum\_of\_squares(Xs, RestCost),

Cost is RestCost + X\*X.

% Simulated Annealing Algorithm

simulated\_annealing(CurrentState, BestState) :-

initial\_temperature(InitialTemperature),

cooling\_rate(CoolingRate),

stopping\_temperature(StoppingTemperature),

max\_iterations(MaxIterations),

sa\_loop(CurrentState, 1, InitialTemperature, BestState, StoppingTemperature, CoolingRate, MaxIterations).

% Main loop of Simulated Annealing

sa\_loop(CurrentState, Iteration, \_, CurrentState, \_, \_, MaxIterations) :-

Iteration > MaxIterations,

write('Reached maximum iterations.'),

nl.

sa\_loop(CurrentState, Iteration, Temperature, BestState, StoppingTemperature, CoolingRate, MaxIterations) :-

Iteration =< MaxIterations,

NewTemperature is Temperature \* CoolingRate,

generate\_neighbor(CurrentState, Neighbor),

objective\_function(CurrentState, CurrentCost),

objective\_function(Neighbor, NeighborCost),

DeltaCost is NeighborCost - CurrentCost,

(DeltaCost < 0; probability\_accept(DeltaCost, Temperature)),

!,

sa\_loop(Neighbor, Iteration + 1, NewTemperature, BestState, StoppingTemperature, CoolingRate, MaxIterations).

sa\_loop(CurrentState, Iteration, Temperature, BestState, StoppingTemperature, CoolingRate, MaxIterations) :-

Iteration =< MaxIterations,

NewTemperature is Temperature \* CoolingRate,

sa\_loop(CurrentState, Iteration + 1, NewTemperature, BestState, StoppingTemperature, CoolingRate, MaxIterations).

% Probability to accept a worse solution

probability\_accept(DeltaCost, Temperature) :-

P is exp(-DeltaCost / Temperature),

random(R),

R < P.

% Example: Generate a neighbor by changing one element randomly

generate\_neighbor(State, Neighbor) :-

length(State, Length),

random\_between(1, Length, Index),

select(Index, State, NewElement, RestState),

random(NewElement), % Generate a random value for the new element

insert\_at(Index, RestState, NewElement, Neighbor).

insert\_at(1, List, Element, [Element|List]).

insert\_at(N, [H|T], Element, [H|R]) :-

N > 1,

N1 is N - 1,

insert\_at(N1, T, Element, R).

% Example initial state (modify this according to your problem)

initial\_state([1, 2, 3, 4, 5]).

% Example parameters (modify these according to your problem)

initial\_temperature(100.0).

cooling\_rate(0.95).

stopping\_temperature(0.01).

max\_iterations(1000).

% Example usage:

% ?- initial\_state(InitialState), simulated\_annealing(InitialState, BestState), objective\_function(BestState, Cost).

**Output:**

Reached maximum iterations.

InitialState = [5, 4, 3, 2, 1],

BestState = [5, 4, 3, 2, 1],

Cost = 55

**Experiment No. – 5 Date:**

**5.** Implementation of Hill-climbing to solve 8- Puzzle Problem.

**Aim:** Toimplementation of Hill-climbing to solve 8- Puzzle Problem.

**Description:**

A set of eight numbered tiles are arranged in order on a puzzle slate with an empty tile placed at last. Here, In this problem, need to arrange the tiles which are unorderd , using the hill climbing searching strategy as in the goal state

Hill Climbing is a heuristic search used for mathematical optimization problems in the field of Artificial Intelligence. It is an iterative algorithm that starts with an arbitrary solution to a problem, then attempts to find a better solution by making an incremental change to the solution.

So, given a large set of inputs and a good heuristic function, the algorithm tries to find the best possible solution to the problem in the most reasonable time period.

Solution is not necessary, a optimal solution (Global Optimal Maxima) but it is consider to be good solution according to time period.

Mathematical optimization problems: Implies that hill-climbing solves the problems where we need to maximize or minimize a given real function by choosing values from the given inputs.

**Program:**

/\* This predicate initialises the problem states. The first argument of solve is the initial state, the 2nd the goal state, and the third the plan that will be produced.\*/

test(Plan):-

write('Initial state:'),nl,

Init= [at(tile4,1), at(tile3,2), at(tile8,3), at(empty,4), at(tile2,5), at(tile6,6), at(tile5,7), at(tile1,8), at(tile7,9)],

write\_sol(Init),

Goal= [at(tile1,1), at(tile2,2), at(tile3,3), at(tile4,4), at(empty,5), at(tile5,6), at(tile6,7), at(tile7,8), at(tile8,9)],

nl,write('Goal state:'),nl,

write(Goal),nl,nl,

solve(Init,Goal,Plan).

solve(State, Goal, Plan):-

solve(State, Goal, [], Plan).

% Determines whether Current and Destination tiles are a valid move.

is\_movable(X1,Y1) :- (1 is X1 - Y1) ; (-1 is X1 - Y1) ; (3 is X1 - Y1) ; (-3 is X1 - Y1).

/\* This predicate produces the plan. Once the Goal list is a subset of the current State the plan is complete and it is written to the screen using write\_sol \*/

solve(State, Goal, Plan, Plan):-

is\_subset(Goal, State), nl,

write\_sol(Plan).

solve(State, Goal, Sofar, Plan):-

act(Action, Preconditions, Delete, Add),

is\_subset(Preconditions, State),

\+ member(Action, Sofar),

delete\_list(Delete, State, Remainder),

append(Add, Remainder, NewState),

solve(NewState, Goal, [Action|Sofar], Plan).

/\* The problem has three operators.

1st arg = name

2nd arg = preconditions

3rd arg = delete list

4th arg = add list. \*/

% Tile can move to new position only if the destination tile is empty & Manhattan distance = 1

act(move(X,Y,Z),

[at(X,Y), at(empty,Z), is\_movable(Y,Z)],

[at(X,Y), at(empty,Z)],

[at(X,Z), at(empty,Y)]).

% Utility predicates.

% Check is first list is a subset of the second

is\_subset([H|T], Set):-

member(H, Set),

is\_subset(T, Set).

is\_subset([], \_).

% Remove all elements of 1st list from second to create third.

delete\_list([H|T], Curstate, Newstate):-

remove(H, Curstate, Remainder),

delete\_list(T, Remainder, Newstate).

delete\_list([], Curstate, Curstate).

remove(X, [X|T], T).

remove(X, [H|T], [H|R]):-

remove(X, T, R).

write\_sol([]).

write\_sol([H|T]):-

write\_sol(T),

write(H), nl.

append([H|T], L1, [H|L2]):-

append(T, L1, L2).

append([], L, L).

member(X, [X|\_]).

member(X, [\_|T]):-

member(X, T).

**Input:**

?- test(Plan).

**Output:**

Initial state:

at(tile7,9)

at(tile1,8)

at(tile5,7)

at(tile6,6)

at(tile2,5)

at(empty,4)

at(tile8,3)

at(tile3,2)

at(tile4,1)

Goal state:

[at(tile1,1),at(tile2,2),at(tile3,3),at(tile4,4),at(empty,5),at(tile5,6),at(tile6,7),at(tile7,8),at(tile8,9)]

false.

**Experiment No. – 6. Date:**

**6.** Implementation of Monkey Banana Problem

**Aim:** To Implementation of Monkey Banana Problem

**Description:**

* A hungry monkey is in a room, and he is near the door.
* The monkey is on the floor.
* Bananas have been hung from the center of the ceiling of the room.
* There is a block (or chair) present in the room near the window.
* The monkey wants the banana but cannot reach it.

**Program:**

% Production rules:

can\_reach 🡪 clever,close.

get\_on: 🡪 can\_climb.

under 🡪 in room,in\_room, in\_room,can\_climb.

Close 🡪 get\_on,under | tall

% Clauses:

in\_room(bananas).

in\_room(chair).

in\_room(monkey).

clever(monkey).

can\_climb(monkey, chair).

tall(chair).

can\_move(monkey, chair, bananas).

can\_reach(X, Y):-clever(X),close(X, Y).

get\_on(X,Y):-

can\_climb(X,Y).

under(Y,Z):-

in\_room(X),in\_room(Y),

in\_room(Z),can\_climb(X,Y,Z).

close(X,Z):-

get\_on(X,Y), under(Y,Z);

tall(Y).

**Input:**

?- can\_reach(A, B).

A = monkey.

B = banana.

?- can\_reach(monkey, banana).

**Output:**

Yes.

**Experiment No. – 7 Date:**

7. 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.

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

**Description:**

**T**he find-S algorithm is a basic concept learning algorithm in machine learning. The find-S algorithm finds the most specific hypothesis that fits all the positive examples. We have to note here that the algorithm considers only those positive training example.

**DataSet:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Example | Sky | Air temperature | Humidity | Wind | Water | Forecast | Enjoy-sport |
| 1 | Sunny | Warm | Normal | Strong | Warm | Same | Yes |
| 2 | Sunny | Warm | High | Strong | Warm | Same | Yes |
| 3 | Rainy | Cold | High | Strong | Warm | Change | No |
| 4 | Sunny | Warm | High | Strong | Cool | Change | Yes |

**Program:**

(data entered normally for record you can also take dataset)

import pandas as pd

import numpy as np

x = np.array([['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same'],

['Sunny', 'Warm', 'high', 'Strong', 'Warm', 'Same'],

['Sunny', 'cold', 'high', 'Strong', 'Warm', 'change'],

['Sunny', 'Warm', 'high', 'Strong', 'cool', 'change']

])

y = np.array(['yes','yes','no','yes'])

for i in range(len(y)):

if y[i]=='yes':

h = x[i].copy()

break

for i in range(len(x)):

if(y[i] == 'yes'):

for j in range(len(x[i])):

if(x[i][j] == h[j]):

pass

else:

h[j]='?'

**output:**

**[**'Sunny' 'Warm' '?' 'Strong' '?' '?']

**Experiment No. – 8 Date:**

**8.** For a given set of training data examples stored in a .csv file, implement and demonstrate the candidate elimination algorithm to output a description of the set of all hypotheses consistent with the training examples

**Aim:** For a given set of training data examples stored in a .csv file, implement and demonstrate the candidate elimination algorithm to output a description of the set of all hypotheses consistent with the training examples

**Description:**

Candidate Elimination Algorithm is used to find the set of consistent hypothesis, that is Version space.

The candidate Elimination algorithm finds all hypotheses that match all the given training examples. Unlike in Find-S algorithm and List-then-Eliminate algorithm, it goes through both negative and positive examples, eliminating any inconsistent hypothesis.Candidate generation is the first stage of recommendation. Given a query, the system generates a set of relevant candidates. The following table shows two common candidate generation approaches: Type. Definition.

**DataSet:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Example | Sky | Air temperature | Humidity | Wind | Water | Forecast | Enjoy-sport |
| 1 | Sunny | Warm | Normal | Strong | Warm | Same | Yes |
| 2 | Sunny | Warm | High | Strong | Warm | Same | Yes |
| 3 | Rainy | Cold | High | Strong | Warm | Change | No |
| 4 | Sunny | Warm | High | Strong | Cool | Change | Yes |

**Program:**

import numpy as np

import pandas as pd

x=np.array([['sunny', 'warm' ,'normal', 'strong', 'warm', 'same'],

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

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

['sunny', 'warm', 'high', 'strong' ,'cool' ,'change']])

y=np.array(['yes','yes','no','yes'])

s\_h = x[0].copy()

g\_h = [["?" for i in range(len(s\_h))] for i in range(len(s\_h))]

i=0

for h in x:

if y[i] == "yes":

for x in range(len(s\_h)):

if h[x]!= s\_h[x]:

s\_h[x] ='?'

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

if y[i] == "no":

for x in range(len(s\_h)):

if h[x]!= s\_h[x]:

g\_h[x][x] = s\_h[x]

else:

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

i+=1

f = [i for i, val in enumerate(g\_h) if val == ['?', '?', '?', '?', '?', '?']]

for i in f:

g\_h.remove(['?', '?', '?', '?', '?', '?'])

print()

print( s\_h,'and', g\_h)

**output:**

['sunny' 'warm' '?' 'strong' '?' '?'] and

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

**Experiment No. – 9 Date:**

9. Write a program to demonstrate the working of the decision tree classifier. Use appropriate dataset for building the decision tree and apply this knowledge to classify a new sample.

**Aim:** To Write a program to demonstrate the working of the decision tree classifier. Use appropriate dataset for building the decision tree and apply this knowledge to classify a new sample.

**Description:**

Decision tree is one of the most poweful and popular algorithm decision tree algorithm falls under the company of supersied learning algorithm,it works for both continuous as well as categorical output variable.

**DataSet:**



**Program:**

import pandas as pd

# iris data set is taken for this experiment

data = pd.read\_csv(r"C:\Users\datasets\iris.csv")

#iloc - integer location

#.values - it converts data into array or matrix

#describing the data

data.shape

data.describe()

data.info()

x=data.iloc[:,:-1].values

y=data.iloc[:,-1].values

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

xtrain,xtest,ytrain,ytest=train\_test\_split(x,y,test\_size=0.20,random\_state=4)

from sklearn.tree import DecisionTreeClassifier

model=DecisionTreeClassifier(criterion='entropy')

model.fit(xtrain,ytrain)

ypred=model.predict(xtest)

from sklearn.metrics import accuracy\_score

print(accuracy\_score(ytest,ypred)\*100)

**output:**

96.6666

**Experiment No. – 10 Date:**

10. Write a program to demonstrate the working of Decision tree regressor. Use appropriate dataset for decision tree regressor.

**Aim:** To Write a program to demonstrate the working of Decision tree regressor. Use appropriate dataset for decision tree regressor.

**Description:**

Decision tree regression observes features of an object and trains a model in the structure of a tree to predict data in the future to produce meaningful contains output. continuous output means that the output/result i.e, not discrete ,it is not represented just by a discrete,know set of number or values.

**Data set:**



**Program:**

*# taken a sample data set of a city for predict cost of a house based on given data*

import pandas as pd

a=['crim','zn','indus','chas','nox','rm','age','dis','rad','tax','ptratio','b','lstat','medv']

data=pd.read\_csv(r"D:\dataset\boston .csv",names=a)

data.shape

data.describe()

data.info()

data.dropna(inplace=True)

x=data.iloc[:,:-1].values

y=data.iloc[:,-1].values

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

xtrain,xtest,ytrain,ytest=train\_test\_split(x,y,test\_size=0.20,random\_state=4)

from sklearn.tree import DecisionTreeRegressor

model=DecisionTreeRegressor()

model.fit(xtrain,ytrain)

ypred=model.predict(xtest)

from sklearn.metrics import mean\_squared\_error

import math

print(math.sqrt(mean\_squared\_error(ytest,ypred)))

print(model.predict([[0.00632,18,2.31,0,0.538,6.575,65.2,4.09,1,296,15.3,396.9,4.98]]))

**output:**

5.077217471696674

[24.]

**Experiment No. –11 Date:**

11. Write a program to demonstrate the working of Random Forest classifier. Use appropriate dataset for Random Forest Classifier

**Aim:** To write a program to demonstrate the working of Random Forest classifier. Use appropriate dataset for Random Forest Classifier.

**Description:**

Random is a Supervised learning algorithm. It can be used both for classification and regression It is also the most flexible to use algorithm. A forest is comprised of trees.

it is the Said that the more trees more it has, the more robust a forest is Random forests Creates selects the best solution by means of voting. It also provides a pretty good indicator of the feature importance.

**DataSet:**



**Program:**

import pandas as pd

data = pd.read\_csv(r"D:\datasets\iris.csv")

data.info()

data.shape

data.describe

x=data.iloc[:,:-1].values

y=data.iloc[:,-1].values

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

xtrain,xtest,ytrain,ytest=train\_test\_split(x,y,test\_size=0.20,random\_state=4)

from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier()

model.fit(xtrain,ytrain)

ypred=model.predict(xtest)

from sklearn.metrics import accuracy\_score

print(accuracy\_score(ytest, ypred)\*100)

#predicting a set

print(model.predict([[2.0,3.4,6.32,6.7]]))

**output:**

96.66666666666667

['Iris-virginica']

**Experiment No. –12 Date:**

12. Write a program to demonstrate the working of Logistic Regression classifier. Use appropriate dataset for Logistic Regression.

**Aim:** To write a program to demonstrate the working of Random Forest classifier. Use appropriate dataset for Random Forest Classifier.

**Description:**

Random is a Supervised learning algorithm. It can be used both for classification and regression It is also the most flexible to use algorithm. A forest is comprised of trees.

it is the Said that the more trees more it has, the more robust a forest is Random forests Creates selects the best solution by means of voting. It also provides a pretty good indicator of the feature importance.

**DataSet:**



**Program:**

import pandas as pd

data = pd.read\_csv(r"C:\Users\dataset\iris.csv")

data.info()

data.shape

data.describe

x=data.iloc[:,:-1].values

y=data.iloc[:,-1].values

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

xtrain,xtest,ytrain,ytest=train\_test\_split(x,y,test\_size=0.20,random\_state=4)

from sklearn.linear\_model import LogisticRegression

modellr=LogisticRegression()

modellr.fit(xtrain,ytrain)

ypredlr=modellr.predict(xtest)

from sklearn.metrics import accuracy\_score

print(accuracy\_score(ytest,ypredlr)\*100)

print(modellr.predict([[5.4,3,4.5,1.5]]))

**output:**

96.66666666666667

['Iris-versicolor']