INDEX

NAME: Baarthraj V.I STD: CSE SEC: A ROLL NO: 038 SUB: PAI

S. No.	Date	Title	Page No. / Marks	Teacher's Sign / Remarks
1.	22/7/24	Basic Python Program using Google Colab	8	
2.	7/8/24	Project - Details	8	
3.	4/9/24	code for DFS Search N - Queens Problems	9	
4.	4/9/24	N - Queens Problems	9	
5.	11/9/24	A* Algorithm	9	
6.	21/9/24	Water Jug Problem	10	
7.	26/9/24	Implementation of decision tree classification	10	
8.	5/10/24	Artificial neural network	10	
9.	16/10/24	k - means clustering	10	
10.	23/10/24	Introduction Prolog	10	
11.	30/10/24	Prolog family tree	10	
12.	6/11/24	unification and resolution	10	

completed

Project Name: Plant diseases prediction

Domain: ~~Tourism~~ Agricultural

Abstraction:

Plant diseases can significantly affect agricultural productivity, leading to food shortages and economic loss. Traditional methods of detecting plant diseases often rely on manual inspection, which is time-consuming, labor-intensive and prone to errors. ML offers a more efficient and accurate approach to diagnosing plant diseases by analyzing leaf images to identify patterns that correspond to specific diseases. In this context, ML models can automate the disease detecting process.

Problem statement:

Early detection and identification of plant diseases are critical to prevent crop loss and ensure food security. However, conventional method for identifying plant diseases are limited by human expertise, which can result in delayed or inaccurate diagnoses.

Solution: the proposed solution is a machine learning-based system that automatically detects plant diseases by analyzing images of leaves. the system uses a combination of image processing techniques and machine learning algorithms such as (CNNs).

Once trained, the model can be used to accurately identify diseases in new images, providing farmers with real-time insights to take preventive actions.

Target audience:

- Farmers and agricultural workers:
  To help them detect diseases in crops early and accurately.
- Agricultural scientists and researchers: To aid in understanding and improving plant disease management
- Agriculture extension service: For providing advisory tools to support farmers
- Agri-tech companies: To integrate disease detection technology into their solutions.

Objective:

The main objective of the project is to develop a machine learning model that can accurately identify plant diseases from images of leaves. The systems aims to:

- Provide early detection of plant diseases.
- Reduce dependency on manual inspection
- Improve crop yield and minimize loss

Algorithm used:

The primary algorithm used for plant disease detection in this project is Convolutional Neural Networks (CNNs). CNNs are well-suited for image classification tasks due to their ability to automatically extract features from images and learn hierarchical patterns.

N-Queen's Problem

Ex. No: 2

Aim:

To write the python code for the N-Queen's problem.

CODE:-

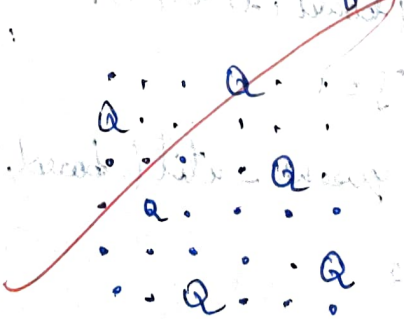
```
def is_safe (board, row, col, n):
    for i in range (col):
        if board [row] [i] == 1:
            return false.
    for i, j in zip (range (row, n) range (
col, -1, -1) ):
        if board [i] [j] == 1 :
            return False.
    return True

def solve_n_queens_util ( board, col, n):
    if (col >= n):
        return True
    for i in range(n) :
        if is_safe (board, i, col, n):
            board [i] [col] = 1
            if solve_n_queens_util ( board, col+n, n)
                return True
            board [i] [col] = 0
    return False.
def Print_board (board, n) :
```

```
print ("\n Solution :")
for row in board:
    for col in row:
if col == 1:
    print ('Q', end=' ')
else:
    Print ('.', end=' ')
print ()
def Solve - n - queen (n):
    board [[0 for - in range (n)] for - in rang
if not Solve-n- queens - until (board, 0, n):
    Print ("No Solution exists")
    return False
Print- board (board, n)
return True
# Driver code:
    n = int (input ("enter the value of N"))
    Solve-n- queen (n)
```

output:

enter the value of N: 6

Solution:



Result:

Thus the code for n-Queen's problem is done Successfully

EX.NO:3

DFS ALGORITHM.

AIM:

To write the Python code for dfs algorithm.

```
class Node:
    def __init__(self, value):
        self.value = value
        self.neighbors = []
    def add_neighbor(self, neighbor):
        self.neighbors.append(neighbor)
class Graph:
    def __init__(self):
        self.nodes = {}
    def add_node(self, value):
        if value not in self.nodes:
            self.nodes[value] = Node(value)
def add_edge(self, from_value, to_value):
    if from_value in self.nodes and to_value
in self.nodes:
        self.nodes[from_value].add_neighbor(
                                    self.nodes[to_value])
        self.nodes[to_value].add_neighbor(self.nodes
                                    [from_value])
def dfs(self, start_value):
    visited = set()
    stack = [self.nodes.get(start_value)]
```

while stack:

node = stack.Pop()

if node and node.value not in visited

Print(node.value)

visited.add(node.value)

for neighbor in reversed(node.neighbors)

if neighbor.value not in visited:

stack.append(neighbor)

graph = Graph()

graph.add-node('x')
graph.add-node('y')
graph.add-node('z')
graph.add-node('u')
graph.add-node('v')
graph.add-node('w')

graph.add-edge('x','y')
graph.add-edge('x','z')
graph.add-edge('y','u')
graph.add-edge('y','v')
graph.add-edge('z','w')
graph.add-edge('v','w')

Print("DFS starting from 'x':")

result = graph.dfs('x')

Print(result)

OUTPUT:

DFS starting from 'x'

x
y
u
v
w
z

Result:

Thus the depth first search Program done and executed Successfully.

A* search Algorithm

EX.NO:4

Aim:

To implement A* search Algorithm using Python.

Code:

```
def astaralgorithm ( start_node, stop_node):
    open_set = set (start_node)
    closed_set = set []
    g = {}
    Parents = {}
g [start_node] = 0
    Parents [start_node] = start_node
while len (open_set) > 0:
    n = None
for v in open_set:
    if n == none or g[v] + heuristic (v) < g[n]
                + heuristic [n]:
    n = v
if n == stop_node or graph_node [n] == none:
    Pass
else:
    for (m, weight) in get_neighbors (n):
        if m not in open_set and m not
                in close_set:
```

```
open - set add(m)
Parents.[m] = n
    g[m] = g[n] + weight

else:
    if g[m] > g[n] + weight
    g[m] = g[n] + weight
    Parents[m] = n

    if m in closed_set:
        closed_set.remove(m)
        open_set.add(m)
if n == none:
    Print("Path does not exist!")
    return none
    if n == stop_node:
    Path = []
while Parents[n] != n:
        Path.append(n)
        n = Parents[n]
Path.append(start_node)
    Path.reverse()
    Print("Path found: {}".format(Path))
    return
    open_set.remove(n)
    closed_set.add(n)
Print("Path does not exist!!")
    return none
def get_neighbors(n):
```

```
if V in graph-nodes:
        return graph-nodes[V]
else:
    return none
def heuristic(n)
        H.dist = {
'A' :11
'B' :6
'C' :99
'D' : 1
'E' : 7
'G' : 0
}       return H_dist[n]
graph_nodes={
    'A' : [('B',2), ('E',3)]
    'B' : [('C',1) ,('',9)]
    'C' : none
     'E' : [('D',6)]
     'D' : [('G',1)]
     }
        astar algorithm('A','G')
```

output:

Path found ['A','E','D','G']

EX.P.NO:5 Water jug Problem using DFS

Aim: To implement water jug problem using Python code.

code:

```
def water_Jug_Problem_dfs(Jug1_cap, Jug2_cap, target_amount):
    J1 = 0
    J2 = 0
    actions = [("fill", 1), ("fill", 2), ("empty", 1),
    ("empty", 2), ("Pour", 1, 2), ("Pour", 2, 1)]
    visited = set()
    stack = [(J1, J2, [])]
while stack:
    j1, j2, seq = stack.pop()
    if (j1, j2) not in visited:
        visited.add((j1, j2))
    if (j1 == target_amount or j2 == target_amount):
        return seq
for action in actions:
    if action[0] == "fill":
        if action[1] == 1:
            next_state = (jug1_cap, j2)
    else:
        next_state = (j1, jug2_cap)
    elsif action[0] == "empty":
        if action[1] == 1:
            next_state = (0, j2)
    else: next_state = (j1, 0)
    else: if action[1] == 1:
        amount = min(j1, jug2_cap - j2)
        next_state = (j1 - amount, j2 + amount)
```

```
else:
    amount = min (j0, jug2cap - j1)
    next_state = (j0 + amount, j1 - amount)
if next_state not in visited
    next_seq = seq + [action]
    stack.append ((next_state[0], next_state[1], next_seq))
return none
result = water_jug_Problem_dfs ([4, 3, 2])
print (result)
```

Output:

```
[('fill', 2), ('Pour', 2, 1),
('fill', 2), ('Pour', 2, 1)]
```

Result:

Thus the water jug program is successfully executed.

Ex.NO: 6 Implementation of decision Tree

date: classification techniques

Aim:

To implement a decision tree classification technique for gender classification techniques using Python

Explanation:

- Import tree from sklearn
- call the function decisionTree classifier() from tree
- Assign values for x and y
- call the function predict for predicting on basis of given random values fo each given feature
- Display the output.

```
import pandas as pd
from sklearn.tree import decisionTree classifier
data = {
    Height : [152, 155, 172, 185, 167, 180, 157, 180, 164, 177]
    'weight' : [45, 57, 72, 85, 68, 78, 22, 90, 66, 86],
    'Gender' : ['Female', 'Female', 'male', 'male', 'Female',
    'male', 'Female', 'male', 'Female', 'male']
}
df = pd.DataFrame(data)
x = df [['Height', 'weight']]
y = df ['Gender']
```

```
classifier = decisionTreeClassifier()
classifier.fit(x, y)
height = float(input("enter height (in cm) for prediction:"))
weight = float(input("enter weight (in kg) for prediction:"))
weight = float(input("
random_values = pd.DataFrame([[height, weight]],
                columns=['height', 'weight'])
predicted_gender = classifier.predict(random_values)
print(f"Predicted gender for height {height} cm and weight {weight} kg: {predicted_gender[0]}")
```

Result:-

Thus the program to implement decision tree is executed successfully.

EXP.NO: 7
date :

k- Means clustering technique

Aim:

To implement a k-means clustering technique using Python language.

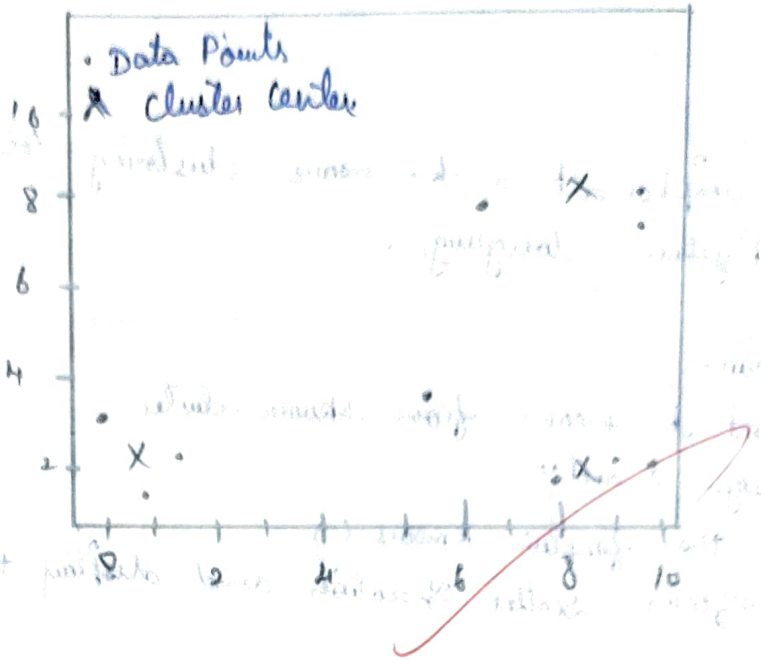
Explanation:

→ import k means from sklearn.cluster
→ Assign x and y
→ call the function kmeans()
→ Perform scatter operation and display the output.

Program:

```
from sklearn.cluster import k-means
import numpy as np
import matplotlib.pyplot as plt
x = np.array([[1,2],[1.5,1.8],[5,8],[8,8],[1,0.6],
[9,11],[8,2],[10,2],[9,3],[8,9],[0,3],
[6,4]]).
k means = k means (n-clusters = 3)
k means: fit-(x)
centers = k means.cluster-centers.
labels = k means.labels
Plt.Scatter (x[:0],x[:1], c= labels, cmap = 'viridis'
marker = 'o', label = 'Data Points')
Plt.Scatter (centers[:,0], cents[:,1], S=200,
c='red', marker='x', label = 'cluster Centers')
Plt.xlabel ("x-axis")
Plt.Y label ('y-axis")
Plt.legend()
Plt.show()
```

output :



Result :-

EX.P.No: 8 Artifical Neural Networks
Date :

Aim:
To implementing artificial Neural Networks for an Application in Regression using Python.

Program:

```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error.

np.random.seed(42)
x = np.random.rand(1000,3) * 10
y = x[:,0]*300 + x[:,1]*500 + x[:,2]*100
    + 5000 + np.random.randn(1000)*100
x_train, x_test, x_train, y_test = train_test_
split(x, y, test_size=0.2, random_state=42)
Scaler = StandardScaler()
x_train = Scaler.fit_transform(x_train)
x_test = Scaler.transform(x_test)
model = Sequential()
model.add(Dense(64, input_dim = x.shape[1],
          activation='relu'))
model.add(Dense(64, input_dim='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse',
metrics=['mae'])
```

history = model.fit(x_train, y_train, epochs=
batch_size=32, validation_split=0.2, verbose=1)

loss, mae = model.evaluate(x_test, y_test, verbose=0)
print("mean absolute error on test data:", mae)

y_pred = model.predict(x_test)
mse = mean_squared_error(y_test, y_pred)
print("mean squared error on test data:", mse)

output:

mean Absolute error on test data: 7.17.3853...
4.38

mean Squared error on test data: 7.29649.3500...
2.55.

Result:-

thus Artificial neural network was successfully implemented.

EX NO: 9 Introduction to Prolog

Date:

Aim: To learn PROLOG terminologies and write basic programs.

Terminologies:-

1. Atomic Terms:-

Atomic terms are usually strings made up of lower- and uppercase letters, digits, and the underscore, starting with a lowercase letter.

Ex: dog

ab_c_321

2. Variables:-

Variables are strings of letters, digits, and the underscore, starting with a capital letter or an underscore.

Ex: Dog

Apple-42=0

3. Compound terms:-

Compound terms are made up of a PROLOG atom and a number of arguments (PROLOG terms, i.e., atoms, numbers, variables, or other compound terms) enclosed in parentheses and separated by commas.

Ex: is_bigger(elephant, X)

f(g(X,-),7)

4. Facts:-

A fact is a predicate followed by a dot

Ex:

bigger_animal(whale).
life_is_beautiful

5. Rules :-

A rule consists of a head (a predicate) and a body (a sequence of predicates separated by commas).

Ex:

is_smaller(X,Y) :- is_bigger(Y,X).

aunt(Aunt, child) :- Sister(Aunt, Parent),
Parent(Parent, child).

SOURCE Code :-

```
woman(mia).
woman(jody).
woman(yolanda).
playsAirGuitar(jody).
Party.
```

Query 1: ?- woman(mia)

true

Query 2: ?- playsAirGuitar(mia).

false

Query 3: ?- Party.

true

Query 4: ?- concat

error Procedure ! concat ! doesn't exist

KB2:

happy (yolanda).

listens2music (mia).

Listens music (yolanda) :- happy (yolanda).

Plays Air Guitar (mia) :- listens2music (mia).

Plays Air Guitar (yolanda) :- listens2music (yolanda).

Output :-

?- Plays Air Guitar (mia)

true.

?- Plays Air Guitar (yolanda).

true.

KB3:

likes (dan, sally).

likes (sally, dan).

likes (john, brittney).

married (X, Y) :- likes (X, Y), likes (Y, X).

friends (X, Y) :- likes (X, Y); likes (Y, X).

Output :-

?- likes (dan, X)

X = sally

?- married (John, brittney)

KBH:

```
food (burger)
food (sandwich)
food (Pizza)
lunch (Sandwich)
Dinner (Pizza)
meal( 1 :- food(x)
```

output:

```
? - food (Pizza )
  True
? - meal (x), lunch(x)
  x = Sandwich
? dinner (Sandwich)
  fales.
```

Result:-

Thus the basic problem to learn prolog Technologies has been executed successfully.

ex.No:10 Prolog - Family Tree

Aim:
To develop a family tree program using Prolog with all possible facts, rules and queries.

Source Code:
Knowledge Base:

```
/* FACTS: */
male(Peter)
male(John)
male(chris)
male(kevin).
female(Betty)
female(rosa)
female(chelsea)
Parent of (christ, Peter)
Parent of (Chris, Betty)
Parent of (Helen, Peter)
Parent of (kevin, Chris)
Parent of (Jony, John)

/* Rules :: */
/* son, Parent
* son, grandparent */
father(x,y) :- male(y) Parent of (x,y)
mother(x,y) :- female(y). Parent of (x,y)
grandmother(x,y) :- male(y), Parent of (y,z)
                    Parent of (z,y)
grand mother(x,y) :- female(y). Parent of (x,z),
                     Parent of (z,y)
brother(x,y) :- male(y), father(x,z), father(y,w), z=w
Sister(x,y) :- female(y). father(x,z). father(y,w)
               x=:=w
```

Result :-

Thus the Prolog Problems to implement and execute family tree was successfully completed.

aPROG:2 Unification and Resolution

Aim:
To execute programs based on unification and resolution

Ex:
Let us see for below the Prolog Program - how unification take place after querying ..

FACTS: likes (john, jane)
likes (jane, john)

Query: ?- likes (john, x)
x = jane

Here upon asking the query first prolog to search matching terms in Facts in Top-down manner for like predicate with arguments and it can match likes (john, ..)

x = jane

Ex: 2 at the prolog query prompt, where you write below query.

?- owns (x, car (bmw)) and owns (y, car (c))

unifies - because i) Predicate names over are same on both side.

(iii) 2nd argument with car predicate inside the brackets are same both side a {x, y} and c in instead both side a (x, y) and c in visible tab mw, -- we write as {bmw/cy} and this is called fact will

(i) select two clauses that contain conflicting term

(ii) can those two clauses

(iii) Come out this conflicting terms.

For example we have following statements

1) If it is a pleasant day you will strawberry Picking..

2) If you are doing strawberry Picking you are happy.

(i) strawberry - Picking ← Pleasant

(ii) happy ← strawberry - Picking and again these statement can be written in CNF like this

(i) (strawberry - Picking ∨ Pleasant)

(ii) (happy ∨ ¬ strawberry - Picking)

By resolving these two clauses and cancelling out the conflicting terms strawberry - Picking and strawberry Picking we can have one new clause

3) Pleasant ∨ happy

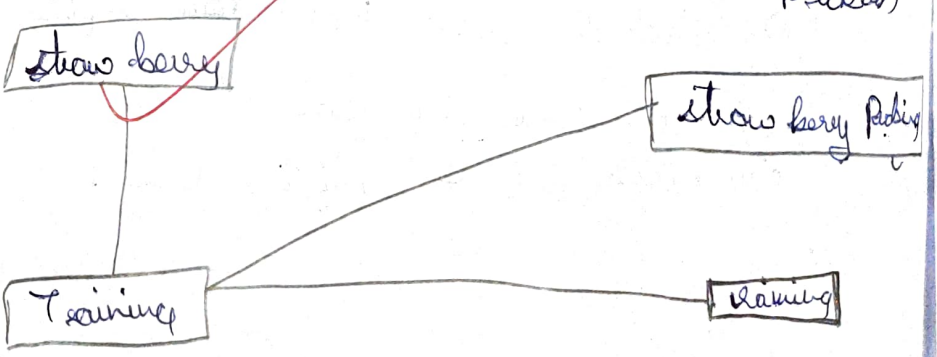
Have > see the figure on right

when we write above new clause in wfa or implies form, we have pleasant happy or happy ← Pleasant

i.e If it is a Pleasant day you are happy

1) (strawberry - Picking ∨ Pleasant)

2) (happy ∨ ¬ strawberry Picking)



Part (i): English sentences

1) If it is sunny and warm day you will enjoy
2) If it is warm and pleasant day you will do strawberry picking.
3) If it is raining then no strawberry picking
4) If it is raining then you will get wet
5) It is warm day
6) It is sunny.

Part (ii): Propositional statements

1) enjoy ← sunny ∧ warm
2) strawberry - Picking ← warm ∧ pleasant
3) warm
4) raining
5) Sunny
6) raining.

Part (iii): CNF

1) (enjoy ∨ ¬ sunny ∨ ¬ warm)
2) (strawberry - Picking ∨ warm)
3) (warm) ∧
4) (raining) ∧
5) (sunny)

Part (iv): other statement we want to Prove by refutation

(Goal 1): you are not doing strawberry picking

(Goal 2): you will enjoy

Source code:

enjoy :- sunny, warm.
strawberry - Picking - warm, Pleasant

wet :- raining

warm

raining

sunny

output:

? not strawberry Picking

fun

? - enjoy

true

? - wet

true

Result:-

This the prolog for family tree program has been executed successfully.