INDEX

NAME: Agishraj STD.: ______ SEC.: ______ ROLL NO.: ______ SUB.: ______

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
1	3/08/2024	Sample python program using google colab	9	
~~2~~		~~Depth first search~~		
~~3~~		~~DF-A* search algorithm~~		
~~4~~				
1.	14/8/24	8 queens problem	9	
2.	14/8/24	Depth first search	10	
3.	1/8/24	A* search algorithm	10	
4.	14/8/24	DFS - water jug problem	10	
5.	16/10/24	ANN regression	10	
6.	28/9/24	Decision tree	10	
7.	9/10/24	K-means	10	
8.	30/10/24	Introduction to prolog	10	
9.	6/11/24	prolog family tree	10	
10.	28/10/24	Min-max algorithm	10	

Completed

4/8/24

Exp: 01

N-Queen's problem

Aim:- To implement N-Queen's solution using python

code:
```
def is_ safe (board, row, coln):
    for i in range (row):
        if board [i] [col ]==1:
            return false
    For i, j in zip(range (row,-1,-1), range (col,-1,-1))
        if board [i][j]==1:
            return false
    return true
def solve_nqueens_until (board, row, n):
    if row >=n:
        return true
    for col in range (n):
        if is_safe (board, row, col, n):
            board [row] [col ] == 1
    if solve_nqueens_until (board, row+1, n):
        return true
        board [row] [col ]=0
    return false
def solve _nqueens (n):
    board = [ [0 for_in_range (n)] for_in_range (n)]
    if not solve_nqueens_util (board, 0, n):
        print ("sol doesn't not exist")
        return none
    return board.
def print_board (board):
    for row in board:
    print (" ".join ("Q" if col else "." for col in row)).
```

```
n= int(input ("Enter the value of N:"))
sol = solve_nqueens (n)
    if solution:
        print_board (solution)
```

output:

Enter the value of N: 4

```
. q . .
. . . q
q . . .
. . q .
```

Result: Thus the above code is executed.

14/8/24

DFS Algorithm

Exp:02

Aim: To implement dfs using python.

code:

```
def dfs_recursive (G, N, visit = none):
    if visit is none
       visit = set()
   visit . add(node)
  print (node, and = " ")
  for neighbour in graph [node]
     if neighbour not in visit.
        dfs-recursive (graph, neighbor, visited)
  return visited.
  def get-graph - from - user ():
        graph = {}
        num = nodes = int (input ("Enter the number
     of nodes in the graph"))
  for-in range (num-nodes).
  node = input ("Enter node name:")
  neighbors = input (f. "enter neighbors of {node}
  (space - seperated ):") . split()
  graph [node] = neighbors.
  return graph.
  graph = get - graph - from - user ()
  start - node = input ("Enter the starting node for
                                           DFS:")
  dfs recursive (graph, start - node)
```

output:

Enter the number of nodes in graph: 4
Enter node name: A
Enter neighbors of A (space - seperated): B C
Enter the node name: B
Enter the starting node for DFS: A

A B D C.

Result: Thus the above code is executed succussfuly.

11/9/24 A* Algorithm

Exp: 03

Aim:- To perform A* Algorithm using Python code.

code:
```
import heapq
def a_star (graph, start, goal):
open_set = []
heapq. heappush (open_set, (0 + heuristic (s, g),
            0, start))
came_from = {}
g_score = {node: float ('inf') for node in graph
g_score [start] = 0
f_score = {node: float ('inf') for node in graph
f_score [start] = heuristic (start, goal)
while open_set:
  current_g, current = heapq. (open_set)
  if current == goal:
    return reconstruct_path (came_from, current
  for neighbor, cost in graph [current]:
    tentative_g_score = g_score [current] + cost
if tentative_g_score < g_score [neighbor]:
   came_from [neighbor] = current
   g_score [neighbor] = tentative_g_score
   f_score [neighbor] = g_score [neighbor] +
    heuristic (neighbor_goal)
if not any (item [1] == neighbor for item
            in open_set):
heapq. heappush (open_set, (f_score [neighbor]
  g_score [neighbor], neighbor))
  return none
heuristic (node, goal):
 return 0
reconstruct_path (came_from, current):
path = [current]
while current in came_from:
```

```
    current = came_from [current]
    path.append (current)
path.reverse ()
return path.
```

output:

path ['A', 'B', 'C', 'D']

Result: Thus the above program is executed successfuly.

Exp:04
14/8/24

Water jug using dfs.

```
def fill-4-gallon (x, y, x-max, y-max):
    return (x-max, y)
def fill-3-gallon (x, y, x-max, y-max):
    return (x, y-max)
def empty-4-gallon (x, y, x-max, y-max):
    return (x, y)
def empty-3-gallon (x, y, x-max, y-max):
    return (x, 0)
def pour-4-to-3 (x, y, x-max, y-max):
    transfer = min (x, y-max-y)
    return (x-transfer, y+transfer)
def pour-3-to-4 (x, y-x-max, y-max):
    transfer = min / y, x-max-x)
    return (x+transfer, y-transfer)
def dfs-water jug (x-max, y-max, goal-x, visited = none, start = (0,0)):
    if visited is none:
        visit = set()
    stack = [start]
    while stack:
        state = stack.pop()
        x, y = state
        if state in visited:
            continue
        visited.add (state)
        print (f" visited state: {state}")
        if x == goal_x:
            print (f" goal reached: {state}")
            return state
```

next states:

```
[fill-4 gallon(x, y, x-max, y-max);
fill-3-gallon(x, y, x-max, y-max);
empty-4-gallon(x, y, x-max, y-max);
empty-3-gallon(x, y, x-max, y-max)
pour-4-to-3(x, y, x, y-max);
pour-3-to-4(x, y, x-max, y-max)]

for new_state in next_states
if new_state not in visited:
stack append(new_state)
return none.

x_max = 4
y-max = 3
goal x = 2
dfs-water_jug(x-max, y-max, goal-x).
```

output:-

visiting state: (0,0)
visiting state: (0,3)
visiting state: (3,0)
visiting state: (3,3)
visiting state: (4,2)
visiting state: (4,0)
visiting state: (1,3)
visiting state: (1,0)
visiting state: (0,1)
visiting state: (4,1)
visiting state: (2,3)
goal reached: (2,3)

(2,3)

Result.

Thus the water jug problem using dfs is executed succesfully.

Exp:05
16/10/24

ANN regression

Aim:- To implementing artificial neural networks for an application in regression using python.

Code:-

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import standard scaler
from keras.models import sequential
from keras.optimizers import adam
import matplotlib.pyplot as plt.

# generating ~~synthetic dataset~~:
np.randoms.seed(42)
X = np.random.rand(1000,3)# 1000 samples, 3 features
y = 3* X[:,0]+2* X[:,1]**2+1.5* np.sin[X[:,2]*
np.pi)+ np.random.normal(0, 0.1, 1000)
X_train, X_test, y_train, y_test = train_test_split(x,y, test_
= 0.2, random_state=42)
scaler = standard scaler()
X_train = scaler.fit_transform(x_train)
X_test = scaler.transform(x_test)
model = sequential()
model.add(Dense(10, input_dim = X_train.shape[1],
activation='relu'))
model.add(Dense(10, activation ='relu'))
model.add(Dense(1, activation ='linear'))
model.compile(optimizer = Adam(learning_rate = 0.01),
loss='mean_squared_error')
history = model.fit(x_train, X_train, epochs=100, batch_size
-32
```

=32, validation_split = 0.2, verbose=1)

Y_pred = model.predict (X_test)

mse = np.mean((Y_test - Y_pred.flatten())**2)

print (f"Mean squared error: {mse:.4f}")

plt.figure (figsize=(12, 6))

plt.plot (history.history ['loss'], label='Training loss')

plt.plot (history.history ['val_loss'], label='validation loss')

plt.title ('Training and validation loss')

plt.xlabel ('Epoch')

plt.ylabel ('Loss')

plt.legend ()

plt.show ()

OUTPUT:

20/20 ———— 0s loss 0.0137 vloss 0.0166

Epoch 86/100

20/20 ———— 0s loss: 0.0128 vloss 0.0166

Epoch 87/100

20/20 ———— 0s loss 0.0132 vloss: 0.0191

Epoch 90/100

20/20 ———— 0s loss 0.0147 vloss: 0.1224

Epoch 91/100

20/20 ———— 0s loss 0.0135 loss 0.0168

Epoch 92/100

20/20 ———— 0s loss 0.0140 loss 0.0198

Epoch 93/100

20/20 ———— 0s loss 0.0159 vloss 0.0198

Epoch 96/100

20/20 ———— 0s loss 0.0123 vloss 0.0170

Epoch 97/100

20/20 ———— 0s loss 0.0123 vloss 0.0170

Epoch 98/100

20/20 ———— 0s loss 0.0123 vloss 0.0170

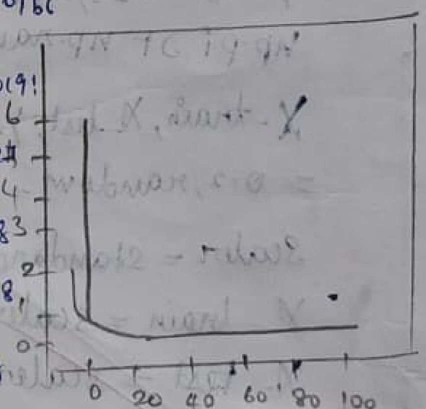
Epoch 99/100

20/20 ———— 0s loss 0.0135 vloss 0.0187

Epoch 100/100

20/20 ———— 0s loss 0.0126 loss 0.0180

7/7 ———— 0s



RESULT:

Thus ANN for an application in regression

Exp: 06
28/9/24

Decision tree

Aim:- To implement a decision tree classification technique for gender classification using python.

code:-

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_split
from sklearn.tree import decision tree classifier
from sklearn.metrics import accuracy_score,
classification_report, confusion_matrix.
import matplotlib.pyplot as plt
from sklearn import tree

data = {'Height': [150, 160, 170, 180, ~~155, 165, 175, 185, 140, 145~~]
 'weight': [50, 60, 70, 80], 'Age': [25, 30, 35, 40],
 'gender': ['male', 'female', 'Male', 'femal']
}

df = pd.dataframe(data)
df ['gender'] = df[' gender'].map({'Femal': 0, 'male':1
x = df[['height', 'weight', 'Age']]
y = df['gender']
x_train, x_test, y_train, y_test = train_test_split
(x, y, test_size = 0.3, random_state = 42, stratify=y)
clf = Decision tree classifier ()
clf.fit(x_train, y_train)
y_pred = clf.predict (x_test)
accuray = accuray_score (y_test, y_pred)
conf_matrix = confusion_matrix (y_test, y_pred)
class report = classification_report(y_test, y
```

zero_division =0)
print (f'Accuracy: {accuracy:.2f}')
print ('confusion matrix:\n', conf_matrix)
Print ('classification report:\n', class_report)
plt.figure(figsize=(12, 8))
tree.plot_tree(clf, feature = X.columns, class_names = ['female', 'Male'], filled = True)
plt.show().

output.

Enter heigh in cm for prediction: 169
Enter weight in cm for prediction: 61
predicted gender for height 169.0cm and weight 61.0 kg: Female

Result: Thus decision tree is executed successfully.

Exp:07
9/10/24

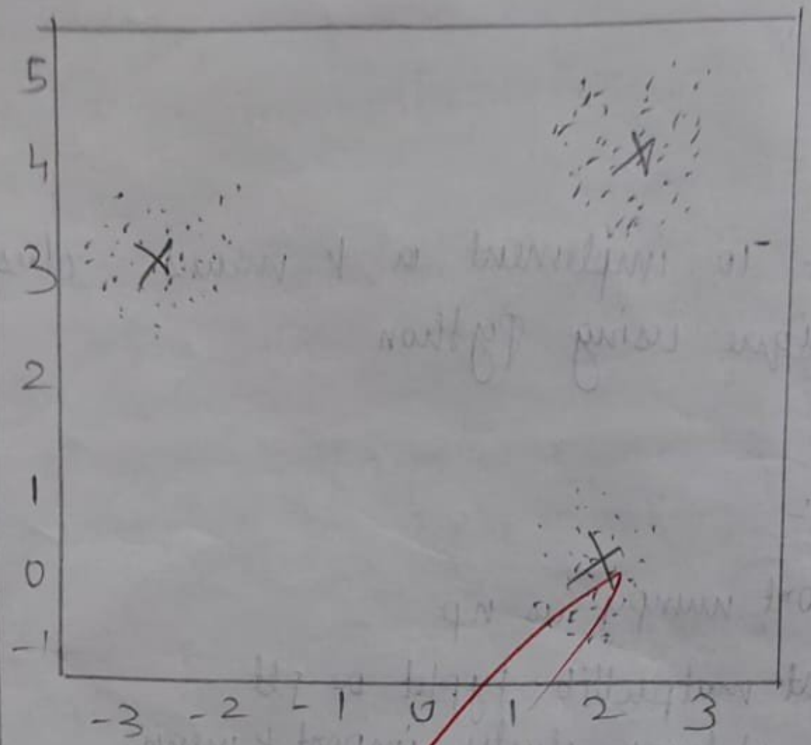
K-means

Aim:- To implement a k-means clustering technique using python

Code:-

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import kmeans
from sklearn.datasets import make_blobs.
x, y_true = make_blobs(n_samples = 309 centres=3,
cluster_std = 0.60, random_state = 0)
k=3
kmeans = Kmeans(n_clusters = k, random_state=0)
y_kmeans = kmeans.fit_predict(x)
plt.figure(figsize = (8, 6))
plt.scatter(x[:, 0], x[:, 1], c=y_kmeans, s=30,
cmap='viridis', label='clusters')
centres = kmeans.cluster_centres_
plt.scatter(centres[:, 0], centres[:, 1], c='red', s=200,
alpha=0.75, marker='x',
label='centroids')
plt.title('K-means clustering results')
plt.xlable('Feature 1')
plt.ylable('feature 2')
plt.legend()
plt.show()
```

Output:



Result:-
Thus k means clustering technique is executed succesfuly.

Introduction to prolog

Exp:08
30/10/24

AIM:- To learn prolog terminologies and write basic programs.

Terminologies:-

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

Eg: dog
ab_c_321

2. variables:- strings of letters, digits and underscore, starting with capital letter or an underscore.

Eg:- Dog
Apple-420

3. compound terms:- Made up of a prolog atom and a number of arrangements enclosed in paranthesis and separated by comas.

Eg:- is_bigger (elephant, X)
f(g(X, _), 7)

4. Facts:- predicate followed by a dot.
Eg:- bigger-animal (in hale)
life_is_beautiful.

5. Rules:- consists of head and a body
Eg:- is_smaller (X, Y) :- is_bigger (Y, X)
aunt (Aunt, Child) :- sister (Aunt, parent), parent (parent, child)

Source code:-

KB1
Woman (mia)
Woman (jody)
Woman (yolanda)
plays Air Guitar as (jody)
parity.

Query 1 : ?- woman (mia).
Query 2 : ?- plays Air Guitar (mia). False
Query 3 : ?- party True
Query 4 : ?- concert procedure concert doesn't exist.

KB2:-

happy (yolanda)
listens 2 music (mia).
listens 2 music (yolanda) :-
plays Air guitar (mia) - listenmusic (mia)
plays Air guitar (yolanda) :- listens music (yolanda)

Query 1 : ?- plays Air guitar (mia) true
Query 2 : ? 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)

Query 1 ? likes (dan, X)
X = sally
Query 2 = ? married (dans, sally). false
Query 3 = ? married (john, brittney). false

KB4:- food (burger)
food (sandwich).
food (pizza).
lunch (sandwich)
dinner (pizza)
meal (X) :- food (X)

Query 1 : ? meal (X), lunch (X)
X = sandwich
Query 2 : ? food (pizza) true
Query 3 : ? dinner (sandwich) false

KB5

Owns (jack, car(bmw))
Owns(john, car (chery)).
Owns (jane, car (chery)).
Sedan (car, (bmw)).
Sedan (car (civic)).
truck (car (chery)).

Query 1: ?

owns(john, X)
X = car (chery)

Query 2: ? owns (john, _)
true

Query 3: ? owns (who, car (chery))
who = john.

Query 4: ? owns (jane, X), sedan (X)
false

Result:-
Thus the prolog programs are executed successfully.

Exp: 9

11/24

Prolog - Family tree

Aim:-

To develop a family tree program using prolog with all possible facts rule and queries.

Source code:-

Knowledge base:

```
/* facts == */
male (peter),
male (john),
male (chris),
male (kevin),
female (betty),
female (jeny),
female (lisa),
female (helen),
parent of (chris, peter),
parent of (chris, betty),
parent of (helen, peter),
parent of (kevin, chris),
parent of (kevin, lisa),
parent of (kevin, john),
parent of (jeny, helen)
/* Rules == */
/* son, parent
/* son, grand parent */
father (X,Y) = male (Y), parent of (X,Y)
mother (X,Y) = female (Y), parent of (X,Y)
grand father (X,Y) :- male (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) Z == W

output:-

male (peter)

true

father (chris, betty)

false

mother (chris, X)

X = betty

brother (chris, helen)

false

Result:- Thus prolog for family tree program has been executed successfully.

MINMAX - ALGORITHM

Exp 10

28/10/24

Aim:- To implement minmax algorithm.

Code:-
```
import math.
def minmax (depth, node_index, is_maximizer,
            scores, height):
    if depth == height:
        return scores [node_index]
    if is_maximizer:
        return max(minmax (depth +1, node
        index *2, false, score, height))
    else:- return min(minmax (depth+1, node_index *2,
              True, scores, height),
           minmax(depth+1, node *2+1, true,
           scores, height))
def calculate_tree_height (num_leaves):
    return math.ceil (math.log2 (num leaves))
scores = [3, 5, 6, 9, 1, 2, 6, -1]
tree_height= calculate_tree_height (len (scores))
optimal_score =minmax (0,0,True, scores, treeheight)
print(f" the optimal score is:{ optimal score}")
```

Output:

The optimal score is :5

Result:- Thus minmax algorithm is executed successfully.