POAI - Observation

INDEX

NAME: K. Chithra STD.: CSE SEC.: A ROLL NO.: 220701054 SUB.: POAI

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
1.	09/8/24	8-Queens Problem	10	
2.	16/8/24	Depth First Search	10	
3.	23/8/24	DFS-water jug	10	
4.	30/8/24	A* Search	10	
5.	06/9/24	Implementation of Decision tree classification techniques	10	
6.	27/9/24	Implementation of ANN for using python-Regression	10	
7.	4/10/24	Implementation of clustering techniques K-means.	10	
8.	18/10/24	Minimax Algorithm	10	
9.	25/10/24	Intro to PROLOG	10	
10.	8/11/24	PROLOG - Family Tree	10	

completed

2024/11/21 08:00

Ex.no:1
Date:09/8/24

N-QUEEN'S PROBLEM

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+1, n):
                return True
            board[i][col] = 0
    return False.

def print_board(board, n):
    print("In solution:")
    for row in board:
        for cell in row:
```

2024/11/21 07:49

```
            if cell == 1:
                print('Q', end = ' ')
            else:
                print('.' end = ' ')

        print()

    def solve_n_queens(n):
        board [[0 for _ in range(n)] for _ in range(n)]
        if not solve_n_queens_util(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_queens(n).
```

OUTPUT:

Enter the value of n: 6

```
. . . Q . .
Q . . . . .
. . . . Q .
. Q . . . .
. . . . . Q
. . Q . . .
```

Enter the value of n: 4

```
. . Q .
Q . . .
. . . Q
. Q . .
```

RESULT: Thus the code for n-Queen's problem is done successfully.

2024/11/21 07:50

Ex. NO: 2
Date: 16/8/24

DFS Algorithm

Aim:

To write the python code for DFS algorithm.

CODE:

```
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.node = {}
    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_
```

2024/11/21 07:50

```
        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)


# Example Usage
graph = Graph()
graph.add_node('1')
graph.add_node('2')
graph.add_node('3')
graph.add_node('4')
graph.add_node('5')
graph.add_node('6')

graph.add_edge('1', '2')
graph.add_edge('1', '3')
graph.add_edge('2', '4')
graph.add_edge('2', '5')
graph.add_edge('3', '6')
graph.add_edge('5', '6')
# perform DFS
print("DFS:")
graph.dfs('3')
```

OUTPUT:

```
DFS:
3
1
2
4
5
6
```

2024/11/21 07:51

RESULT:

Thus the code for DFS algorithm is done successfully.

Ex.no: 3
Date: 30/8/24

A* algorithm

Aim:

To write a code for A* algorithm.

CODE:

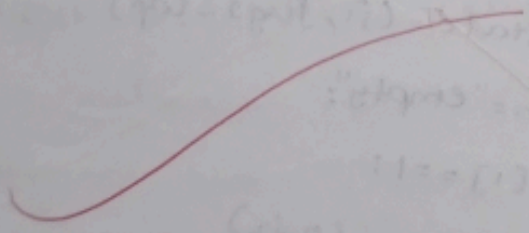
```
def astar (start_node, stop_node):
    open_set = set(start_node)
    closed_set = set()

    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_nodes[n]==None:
        pass

    else:
        for (m, weight) in get_neighbors(n):
        if m not in open_set and m not in closed_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)
```

2024/11/21 07:51

```
Graph_nodes = {
    'A': [('B', 2), ('E', 3)],
    'B': [('C', 1), ('G', 9)],
    'C': None,
    'E': [('D', 6)],
    'D': [('G', 1)],
}
astar ('A', 'G')
```

OUTPUT:

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



RESULT:

2024/11/21 07:51

Thus the code for A* algorithm is done successfully.

Ex.NO: 4
Date: 23/8/24

Water Jug Problem using DFS

Aim:

To write a code for water jug problem using DFS.

CODE:

```
def water_jug_problem_dfs(jug1_cap, jug2_cap, target_amount):
    def dfs(j1, j2, seq, visited):
        if j1 == target_amount or j2 == target_amount:
            return seq
        visited.add((j1,j2))
        actions = [
            ("fill", 1), ("fill", 2), ("empty", 1), ("empty", 2),
                        ("Pour", 1, 2), ("Pour", 2, 1)]

        for action in actions:
            if action[0] == "fill":
                if action[1] == 1:
                    next_state = (jug1_cap, j2)
                else:
                    next_state = (j1, jug2_cap)
            elif 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)
```

2024/11/21 07:52

```
        else:
            amount = min(j2, jug1_cap - j1)
            next_state = (j1 + amount, j2 - amount)

    if next_state not in visited
            next_seq = seq + [action]
            result = dfs(next_state[0], next_state[1],
                         next_seq, visited)

            if result:
                return result

    return None
# initialize the DFS
    visited = set()
    return dfs(0, 0, [], visited)

# Example usage
    result = water_jug_problem_dfs(4, 3, 2)
    print(result)
```

OUTPUT:

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

RESULT:

Thus the code for water jug problem using DFS is executed successfully.

2024/11/21 07:52

Date: 06/9/24

Ex.no:05

Implementation of Decision tree classification techniques

Aim:

To implement a decision tree classification techniques for gender classification using python.

Algorithm:

1. Import tree from Sklearn.
2. Call the function DecisionTreeClassifier() from tree
3. Assign values for x and y.
4. Call the function predict for predicting on the basis of given random values for each given feature.
5. Display the output.

CODE:

```
import pandas as pd
from sklearn.tree import
DecisionTreeClassifier
data = {'Height' : [154, 165, 170, 180, 157, 177, 172, 164, 155, 180],
        'weight' : [50, 45, 57, 70, 85, 68, 78, 22, 66, 85],
        '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)
```

2024/11/21 07:53

```
height = float(input("Enter height (in cm): "))
weight = float(input("Enter weight (in kg): "))
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 is {predicted_gender[0]}")
```

Output:

```
Enter height in cm : 154
Enter weight in kg : 39
predicted-gender for height 154.0cm and weight
    39.0 kg is Female.
```

Result:

Thus the python code for decision tree classification for gender classification is executed successfully.

2024/11/21 07:53

Ex.NO:06 Implementing Artificial Neural
Date: 27/9/24 for AN application using python - Regression.

Aim:

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

Algorithm:

1. Import libraries like Sequential and Dense from keras.
2. Load the dataset, separate features(x) and target(Y).
3. Compile the model using a regression loss function (Mean squared Error)
4. adding Dense layers with RELU activation.
5. Display the output.

CODE:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import
train_test_split
from sklearn.models preprocessing import
StandardScaler
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
import matplotlib.pyplot as plt

np.random.seed(42)
x = np.random.rand(1000,3)
```

2024/11/21 07:54

```
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, xtest, y_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(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, y_train, epochs=100,
          batch_size=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='valiation_loss')
plt.title('Training and validation loss')
plt.xlabel('Epoch')
plt.ylable('Loss')
plt.legend()
plt.show()
```

2024/11/21 07:54

Output:

	Feature	target
0	3.745401	7.846204
1	9.507143	16.343597
2	7.319939	15.400275
3	5.986585	13.194341
4	1.560186	4.239954

Epoch 1/100

20/20 ———— 2s 10ms/step - Loss: 143.6730

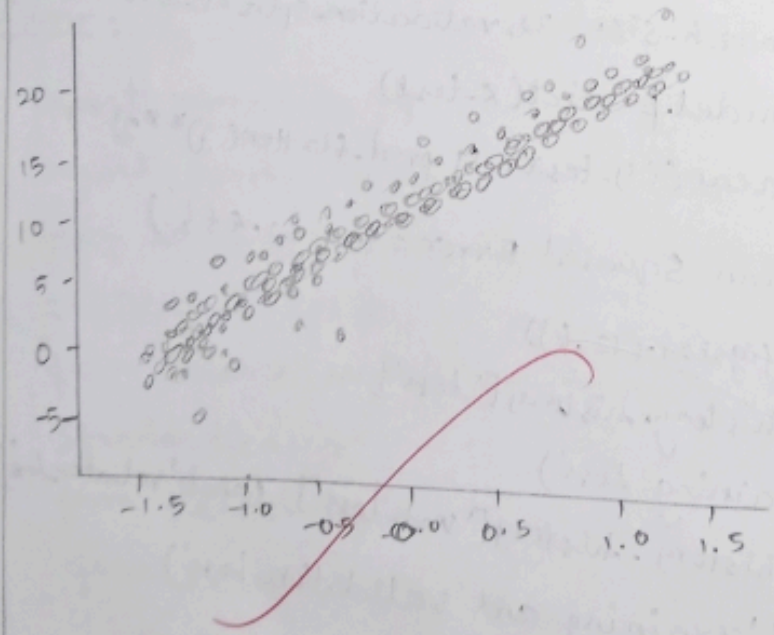
⋮

Epoch 100/100

20/20 ———— 0s 4ms/step - Loss: 4.2248

7/7 ———— 0s 4ms/step - Loss: 3.9068

Test loss (MSE): 3.5400872230529785



Result:

2024/11/21 07:55

Thus the implementation of ANN for an application in regression using python is executed successfully.

Ex.NO: 7 Implementation of clustering techniques

Date: 4/10/24 K - Means

Aim:

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

Algorithm:

1. Import K-Means from sklearn.cluster
2. Assign x and y.
3. Call the function kMeans().
4. Perform scatter operation and .
5. display the output.

CODE:

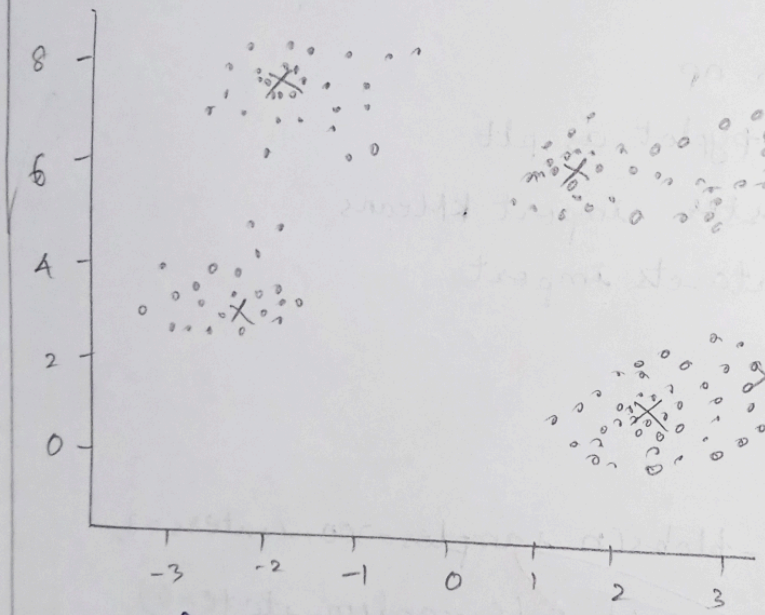
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import
make_blobs

sklearn.cluster
X, y_true = make_blobs(n_samples=300, centers=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')
Centers = kmeans.cluster_centers_ plt.scatter(Centers
[:,0], Centers[:,1], c='red', s=200, alpha=0.75
                    label='Centroids')
```

2024/11/21 07:55

```
plt.title('K-means Clustering Results')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```

Output:

	Feature 1	Feature 2
0	0.836857	2.136359
1	-1.413658	7.409623
2	-1.15213	5.099619
3	1.018616	7.814915
4	1.271351	1.892542



Inertia : 212.00599621083475

Silhouette score : 0.6819938690643478

Result:

Thus the implementation of K-means clustering technique using python is executed successfully.

2024/11/21 07:56

EX.NO:08 Introduction to Prolog Date: 25/10/24

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.

Ex: dog
ab-c-321

2. Variables:-

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

Ex: Dog
Apple-420

3. Compound Terms:-

Compound terms are made up of a prolog atom and a number of arguments (PROLOG terms, atoms, numbers, variables, or other compound terms).

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.

2024/11/21 07:56

5. Rules!:-

A rule consists of a head 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:

KB1:

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

?- woman(mia).
True
?- woman(etw).
False
?- concert.
Procedure concert does not exist.
```

KB2:

```
happy(yolanda).
listen2music(mia).
Listens2music(yolanda):- happy(yolanda).
playsAirGuitar(mia):- listens2music(mia).
playsAirGuitar(Yolanda):- listens2music(yolanda).
```

2024/11/21 07:56

?- happy(yolanda).

True

?- playsAirGuitar(mia).

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).

?- married(dan, sally).

true.

?- married(john, brittney).

false

KB4:

food(burger).

food(sandwich).

food(pizza).

lunch(sandwich).

dinner(pizza).

meal(X) :- food(X)

?- food(pizza).

True

?- dinner(sandwich).

False

2024/11/21 07:57

KB 5:

```
owns(jack, car(bmw)).
owns(john, car(chevy)).
owns(olivia, car(civic)).
owns(jane, car(chevy)).
sedan(car(bmw)).
sedan(car(civic)).
truck(car(chevy)).
```

```
?- owns(john, X).
   X = car(chevy).

?- owns(john, _).
   True

?- owns(jane, X).
   False
```

Result:

PROLOG terminologies and writing basic programs are executed successfully.

2024/11/21 07:58

Ex.NO:09 PROLOG - Family Tree Date: 8/11/24

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

Knowledge Base:

```
/* facts */
male(peter)
male(john)
male(chris)
male(kelvin)
female(betty)
female(jeny)
female(lisa)
female(helen)

parent of (chris, peter)
parent of (chris, betty)
parent of (helen, peter)
parent of (helen, betty)
parent of (kelvin, chris)
parent of (kelvin, lisa)
parent of (jeny, john)
parent of (jeny, helen)

/* rules */
```

a) father(x,y) :- male(x), parent of(x,y)

Y	X
peter	Chris
peter	Helen
John	jeny
Chris	kelvin

2024/11/21 07:58

b) female(Y), parent of (X, Y)

Y	X
betty	chris
betty	Helen
Lisa	Kelvin
Helen	Jeny

c) Male(Y), Parent of (X, Z), parent of (Z, Y)

Y	X	Z
peter	Kelvin	chris
peter	Jeny	Helen

d) Male(Y), parent of (X, Z), parent of (Z, Y)

Y	X	Z
betty	kelvin	chris
betty	Jeny	Helen

e) Male(Y), father(X, Z), father(Y, W) Z == W

Procedure father(A, B)' does not exist

f) female(Y), father(X, Z), father(Y, W) Z == W

procedure father (A, B)' does not exist.

Result:

This the prolog family tree is executed sucessfully.

2024/11/21 07:59

EX.NO:10 Minimax Algorithm Date: 18.10.24

Aim:

To write a code for minimax algorithm.

Implementation:

- A simple examples can be used to explain how the minimax algorithm works we have included in the example.
- There are 2 players in this scenario one named Maximizes and other named Minimized.
- Maximizes will strive for highest possible score and minimizes will strive for lowest score.

Code:

```
import math
def minimax (depth, node_index, is_maximizer,
        scores, height)
if depth == height
    return scores[node_index]
if is_maximizer:
    return max (minimax (depth+1,
node_index*2, false, scores, height),
minimax (depth+1, node_index*2+1, False,
scores, height))
else:
    return min (minimax (depth+1, node_index*2,
```

2024/11/21 07:59

```
        false, scores, height),
        minimax(depth+1, node_index *2+1, false,
            scores, height)).

def calculate_tree_height(num_leaves):
    return math.ceil(math.log2(num_leaves))
scores = list(map(int, input("Enter the score
    seperated by spaces: ").split()))
tree_height = calculate_tree_height(0, 0, True,
    scores, tree_height)
optimal_score = minimax(0, 0, True, scores, tree_
    height)
print("The optimal score is : {optimal_score}")
```

Output:

Enter the scores:

-1 4 2 6 -3 5 0 7

The optimal score: 4

Result:

Thus the minimax algorithm successfully determints the optimal move for players.

2024/11/21 08:00