INDEX

2207010007
CSE-A

NAME: Abirami PL STD.: ____ SEC.: ____ ROLL NO.: ____ SUB.: POAI Observation

S. No.	Date	Title	~~Page No.~~ marks	Teacher's Sign / Remarks
1)	31/7/24	Basic Python Programs	9	
2)	7/8/24	Prefive analysis of student Final Performance	9	
3)	4/9/24	n-queens problem		
4)	11/9/24	A*		
5)	18/9/24	DFS	9	
6)	18/9/24	AO* algorithm	10	
7)	25/9/24	Decision tree	10	
8)	9/10/24	K-means	10	
9)	16/10/24	Artificial neural network	10	
10)	23/10/24	Minmax	10	
11)	30/10/24	Introduction to prolog	10	
12)	6/11/24	Prolog - family tree	10	

completed

EX:NO:1 Basic Python Programs

① Palindrome check:

```
def is_palindrome(s):
    return s == s[::-1]

print(is_palindrome("madam"))
print(is_palindrome("hello"))
```

Output :-

```
True
False
```

② Factorial calculation:

```
def factorial(n):
    if n == 0:
        return 1
    else
        return n * factorial(n-1)

print(factorial(5))
print(factorial(7))
```

Output:

```
120
5040
```

③ Fibonacci Sequence:

```
def fibonacci(n):
    sequence = [0, 1]
    while len(sequence) < n:
        sequence.append(sequence[-1] +
                        sequence[-2])
    return sequence
print(fibonacci(10))
```

Output:
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]

④ Prime Number check:

```
def is_prime(n):
    if n<=1:
        return False
    for i in range(2, int(n**0.5)+1):
        if n%i==0
            return False
    return True
print(is_prime(11))
print(is_prime(25))
```

Output:
True
False

⑤ Reverse a string

```
def reverse_string(s):
    return s[::-1]
print(reverse_string("hello"))
print(reverse_string("hi"))
```

Output:
olleh
ih

⑥ Find the largest element in a List :-

```
def find_l(lst):
    return max(lst)
print(find_l([3,5,7,2,8]))
print(find_l([1,2,3,4,5]))
```

Output:
8
5

⑦ Check for Armstrong Number:

```
def is_arm(n):
    n_str = str(n)
    n_len = len(n)
    sum_of_powers = sum(int(digit)**n_len for
                        digit in n_str)
    return sum_of_powers == n
print(is_arm(153))
```

Output:
True.

⑧ Sum of digits

```
def sum(n):
    return sum(int(digit) for digit in str(n))
print(sum(1234))
print(sum(5678))
```

Output:
10

9. Count vowels in a String:

```
def c-v(s):
    v = "aeiouAEIOU"
    return sum(1 for char in s if char in v)

print(c-v("hello"))
print(c-v("world"))
```

Output:

```
2
1
```

10. Find the GCD of two numbers:

```
def gcd(a,b):
    while b:
        a,b = b, a%b
    return a
print(gcd(48, 18))
print(gcd(56,98))
```

Output:

```
6
14
```

Predictive analysis of Student Final Performance

Domain type:

Educational Data Mining and Learning Analytics.

Problem statement:

Predictive modeling ~~for~~ of student's Final Performance.

Algorithm:-

Decision trees:-

- Model decision making processes by splitting data based on feature value.
- USE CASE: Understanding which features (eg. attendance, participation, most influence final grades.

① Criteria for splitting:-

- For classification:
  1) Gini impurity
  2) Entropy
- For Regression.

② Splitting Process:-

① Choose a feature:-

Select the feature that provides the best split according to the criterion.

2) Split Data

Divide the data into subset based on the chosen features.

3) Repeat.

Objective:

Develop and implement a decision tree model to predict students final performance in a specific course based on historical academic data, behavioral metrics & personal characteristics. The aim is to accurately forecast students. final grade or performance outcomes.

People benefit by this project:

1. Students
2. Educators
3. Academic Institutions
4. Parents & Guardians.

n-Queens

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

def solve_n-queens(board, col):
    if col>=len(board):
        return True
    for i in range(len(board)):
        if is_safe(board, i, col):
            board[i][col]=1
            if solve_n-queens(board, col+1):
                return True
            board[i][col]=0
    return False.
```

```
def print_board(board):
    for row in board:
        print(" ".join(str(x) for x in row))

def solve():
    n = 8
    board = [[0 for _ in range(n)]
             for _ in range(n)]
    if not solve_n_queens(board, 0):
        print("solution does not exist")
        return False
    print_board(board)
    return True

solve()
```

Output:

```
0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0
True
```

Result

Thus the program was executed successfully & output is verified.

A* Program

```
def astaralgo(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
                else:
                    if g[m] > g[n] + weight:
                        g[m] = g[n] + weight
                        parents[m] = n
```

```
        if m in closed_set:
            open_set.add(m)
        parents[m] = n

    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 path
    open_set.remove(n)
    closed_set.add(n)

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), ('G', 9)],
    'C': None,
    'D': [('D', 6)],
    'E': [('G', 1)];
}
astaralgo('A', 'G')
```

Output

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

Result: Thus the program was executed successfully & output was verified.

DFS

```
def add_edge(adj, s, t):
    adj[s].append(t)
    adj[t].append(s)

def dfs_rec(adj, visited, s):
    visited[s] = True
    print(s, end=" ")
    for i in adj[s]:
        if not visited[i]:
            dfs_rec(adj, visited, i)

if __name__ == "__main__":
    V = 5
    edges = [[1, 2], [1, 0], [2, 0],
             [2, 3], [2, 4]]

    for e in edges:
        add_edge(adj, e[0], e[1])

    source = 1
    print("DFS from source:", source)
    dfs(adj, source)
```

Output

1 2 0 3 4

Result

Thus the program was executed successfully & output was verified.

18/11/24

AO* algorithm

Aim: To implement AO* algorithm using python Programming language.

```
Class Graph:
    def __init__(self, graph, heuristic):
        self.graph = graph
        self.heuristic = heuristic
        self.solution = {}

    def ao_star(self, node):
        print(f"Expanding: {node}")

        if node not in self.graph or not self.graph[node]:
            return

        children = self.graph[node]

        best_path = None
        min_cost = float('inf')

        for group in children:
            cost = sum(self.heuristic[child] for child in group)

            if cost < min_cost:
                min_cost = cost
                best_path = group

        self.solution[node] = best_path
        print(f"Best path for {node}: {best_path} with
                cost {min_cost}")

        for child in best_path:
            self.ao_star(child)

    def get_solution(self):
        return self.solution.
```

```
graph = {
  'A': [['B', 'C'], ['D']],
  'B': [['E']],
  'C': [['G']],
  'D': [['H']]
  'E': [],
  'G': [],
  'H': []
}

heuristic = {
  'A': 0, 'B': 1, 'C': 2, 'D': 4, 'E': 1, 'G': 3,
                                  'H': 5
}

graph_obj = Graph(graph, heuristic)
graph_obj.ao_star('A')

solution = graph_obj.getSolution()
print("Solution: ", solution)
```

Output: Expanding: A
Best path for A: ['B', 'C'] with cost 3
Expanding: B: ['E'] with cost 1
Expanding: E
Expanding: C
Expanding: G
Solution: {'A': ['B', 'C'], 'B': ['E'], 'C': ['G']}

Result:
Thus the program is executed & output is successfully verified.

25/9/24 Decision tree

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

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier

data = {
    'Height': [152, 155, 172, 185, 167, 180, 157, 180, 164, 177],
    'Weight': [45, 57, 72, 85, 68, 78, 22, 90, 66, 88],
    '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: "))
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]}")
```

Output:

Enter height (in cm) for prediction: 169
Enter weight (in kg) for prediction: 61
Predicted gender for height 169.0cm and weight 61.0kg: Female.

Result:

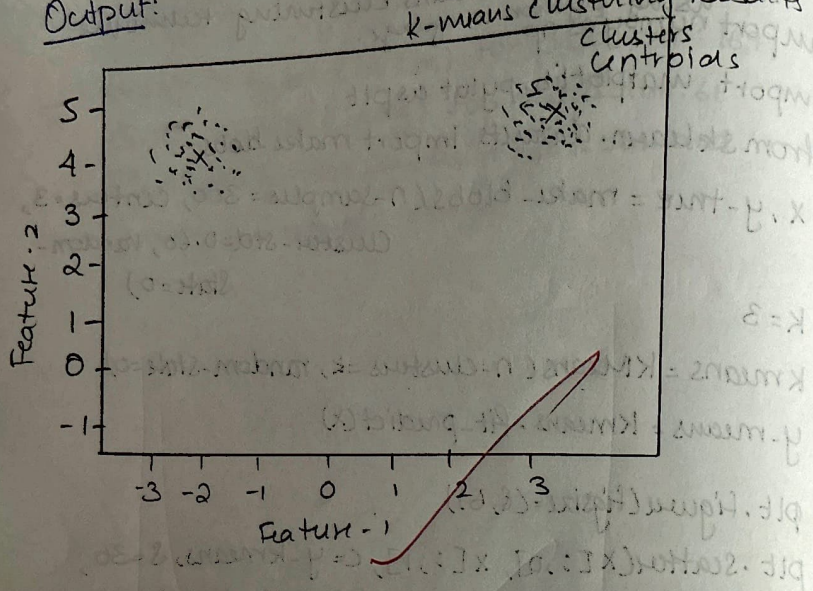
Thus the program was executed and output is verified successfully.

algorithm k-means

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

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
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_means = kmeans.fit_predict(X)
plt.figure(figsize=(8,6))
plt.scatter(X[:,0], X[:,1], c=y_kmeans, s=36,
            cmap='viridis', label='clusters')
centers = kmeans.cluster_centers_
plt.scatter(centers[:,0], centers[:,1], c='red',
            s=200, alpha=0.75, marker='x',
            label='Centroids')
plt.title('K-mean Clustering Results')
plt.xlabel('Feature1')
plt.ylabel('Feature2')
plt.legend()
plt.show()
```

Output:



Result:

Thus the program is executed and successfully verified.

16/10/24 Artificial neural network

Aim: To Implementing artificial networks for an application in classification using python.

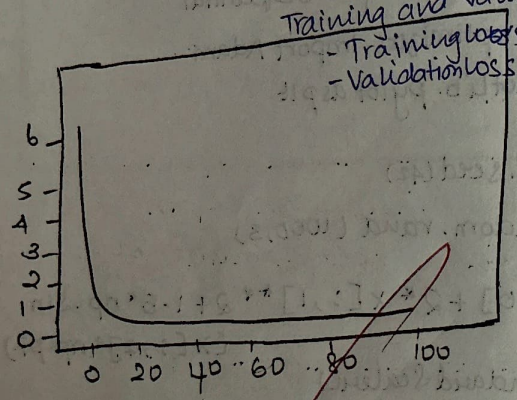
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from keras.models import sequential
from keras.optimizers import Adam
import matplotlib.pylot as plt

np.random.seed(42)
x = np.random.rand(1000,3)
y = 3*x[:,0] + 2*x[:,1]**2 + 1.5*np.sin(x[:,2]*np.pi)
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
model = Sequential()
model.add(Dense(10, activation='relu'))
model.add(Dense(1, activation='linear'))
model.compile(optimizer = Adam(learning_rate=0.01),
              loss = 'mean_squared_error').
y_pred = model.predict(x_test)
mse = np.mean((y_test - y_pred.flatten())**2)
plt.figure(figsize=(12,6))
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.legend()
plt.show()
```

Output:

Epoch 1/100
20/20 ——— 2s 19ms/step - loss: 8.8473 - val_loss: 1.93 73

Epoch 2/100
20/20 ——— 0s 7ms/step - loss: 1.5651 - val_loss: 0.6565



Result:

Thus the program is executed & successfully verified.

23/10/24 Minmax

Aim: To implement minmax algorithm using

Code: Python language.

```
[] 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, scores, height),
                minmax(depth+1, False, scores, height))

    else:
        return min(minmax(depth+1, node_index*2,
                    True, scores, height)

def calculate_tree_height(num_leaves):
    return math.ceil(math.log2(num_leaves))

scores = [3,5,6,9,1,2,0,7]
optimal_score= minmax(0,0, True, scores, tree_height
print(f"The optimal score is :{optimal_score}")
```

Output:

The optimal score is : 5

Result:

Thus the program is executed & output is verified successfully.

30/01/24 Introduction to prolog

Aim:

To learn prolog terminologies and write basic programs.

Terminologies:

1) Atomic terms:

It usually string made up of lower & uppercase letters, digits & underscore.

eg. dog, abc-c-321

2) Variable

They are string of letters, digits and the underscore starting with capital letter of underscore.

eg. dog

Apple-1420

Source code:

KB1:

happy(yolanda).

listen2music(mia)

plays Air Guitar(yolanda): listen2music(yolanda)

O/P:

?- playsAirGuitar(mia)
true

?- playsAirGuitar(yolanda)
true

KB2

likes(dan, sally)

likes(sally, dan)

likes(john, tmitney)

married(x,y) - likes(x,y), likes(y,x)

friends(x,y) - likes(x,y), likes(y,x)

O/P:

?- likes(dan,x)
x=sally

?- married(dan, sally)
true

?- married(john, brittney)
false

KB3

woman(mia)

woman(jody)

plays guitar(jody)

party

O/P:

?- woman(mia)
true

?- playsstir Guitar(mia)
false

?- party
true

?- concert
produce concert doesn't exist.

KB5

owns(jack, car(bmw))
owns(john, car(chevy))
owns(olivia, car(civic))
own(jane, car(chevy))
Sedan(car(bmw))
Sedan(car(civic))
sed truck(car(chevy))

O/P: ?- owns(john, x)
x - car(chevy)
?- own(john, -)
true
?- owns(who, car(chevy))
who = john
?- owns(jane, x), truck(x)
x = car(chevy)

Result:
Thus the program is executed & output is verified succesfully.

6/11/24

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(jeny)
female(lisa)
female(helen)

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

/* RULES:: *
son-parent
* son, grand parent */

* /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) :- female(x), 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(xz), father(y,w)
= w z

Result:

Thus the program is executed & successfully verified.