Index

Name : P. Boorath Roll No 220701037

Subject : PAI Std : Sec :

School :

S.No.	Date	Title	Page No. marks	Teacher's Sign / Remarks
1.	31/7/24	Sample Python Program	6	
2.	7/8/24	Project report	6	
3	4/9/24	A - queen problem	6	
4.	2/9/24	Depth first Search	6	
5.	11/9/24	A* algorithm	6	
6.	18/9/24	AO* algorithm	8	
7.	25/9/24	Decision tree	8	
8.	9/10/24	k-means	8	
9.	16/10/24	Artificial neural network	8	
10.	23/10/24	minimax	8	
11.	30/10/24	Introduction to Prolog	9	
12.	6/11/24	Prolog - family tree	10	

Completed

Palaniyappa

Ex no: 1
21/7/24

10 Python simple Program

1) Concatenation of 2 lists using '+' operation

Program:-

```
list 1 = [1, 2, 3, 4, 5]
list 2 = [6, 7, 8, 9, 10]
list 3 = list 1 + list 2
print ("Concatenated list using '+' operator : "8+v(list 3))
```

output:

Concatenated list using '+' operator : [1, 2, 3, 4, 5, 6, 7.8, 9, 10

2) Removing Item from a list :-

Program :

```
list 1 = ['apple', 'orange', 'blue berry', 'dragon fruit', 'goa']
print ("orginal list is :", list 1)
list1. remove ('blue berry')
print (" After removing the item :", list 1)
```

output :

orginal list is : ['apple', 'orang', 'blue berry', 'dragon fruid', 'goa']

After removing the item : ['apple', 'orange', dragon fruit 'goa']

3) Finding second smallest element in an array

Program

```
list 1 = [10, 20, 4, 70, 64, 7]
list 2 = list (set (list 1))
list 2. sort ()
print (" second smallest element is :" list 2 [1])
```

Output:

Second smallest element is : 7

4. Finding second largest NO

Program:

```
list 1 = [10, 20, 4, 70, 64, 7]
list 2 = list (set (list 1))
list 2. sort ()
print ("second largest element is :", list 2[-2])
```

output:

second largest element is : 64

5. list of square.

```
def square_list (n)
    return [i**2 for i in range (n1)]
num = int (input ("Enter a number :"))
print ("List of squares :", squares_list (num))
```

output:

Enter a no : 5

list of square : 1, 4, 9, 16, 25

Ex: 4 College recommendation system for admissions

7/8/24

Problem statment:

After completing the school student are unware about the college and which college they should select for their cutoff and cast categery.

abstract:

Students and their families would receive tailored college recommendations under the proposed system, which would significantly alter the account once, based on their academic interest, career objectives, and 10th grade cutoff scores. The system gives you precise, data-driven insights about your prospects of getting into certain colleges by looking at past cutoff scores, admission trends and other relevant data. Complete details about the collaborating colleges are offered, as well as streamlined methods for data analysis and retrieval and user friendly interface. The system uses complex algorithm and a large quantity of data to help kids and their families make decisions that support their academic and career aspirations. The technology integrates advanced analytics with intuitive user interface to empower students to make informed decision about their future educational paths. By giving students acess to information on admission possibilities and

and assisting them in identifying universities that best fit their preferences and objectives.

Objectives

Our Prediction technology provides a comprehensive solution to assist tamil nadu students in making educated judgment regarding their college admissions. our research seamlessly integrates recommendation algorithms and regression modeling to address the challenges of anticipating current year cutoff and offering individualized college options based on a student's subject of interest. our solution uses evaluation, stringent data protection policies and easy to use interface to improve decision makers for both students and eductional institutions.

End users:

our target audience are hsc school students.

Domain: college admit guide

Objectives:

Equitable: Emphasizes fairness and justice in the admission process

Data driven: Highlights the reliance on data for decision-making

intuitive: Refers to the user friendly nature of the system

comprehensive: indicates the thoroughness of the approach

predictive: Describes the model's ability to forecast future cutoffs.

Algorithm: ML processing steps.

1) Data collection & preprocessing
   * Collect: gather data on admission, cutoffs and demographics.
   * clean: Remove duplicates, handle missing values and encode categorical data

2) Feature selection & modeling
   * Select features: choose relevant features using analysis
   * Build models: use regression (eg. linear, Randomforest)

3) Scoring & Ranking
   * Score: calculate weight score for college
   * Rank: sort colleges based on scores

4) user interface
   * Design: create user-friendly web/mobile
   * Ensure Accessibility: include features for

5) evaluation
   * Measure: use metrics like MAE, MSE, RMSE
   * Refine: incorporate feed back and update models.

Exc No:3 N queen Problem

4/9/24

Aim:

To implement N queen problem

Program:

```
N = int(input("Enter the number of queens: "))
board = [[0]*N for _ in range(N)]

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,N), range(col,-1,-1)):
        if board[i][j] == 1:
            return False
    return True

def solve_nqueens(board, col):
    if col >= N:
        return True
    for i in range(N):
        if is_safe(board, i, col):
            board[i][col] = 1
            if solve_nqueens(board, col+1):
                return True
            board[i][col] = 0
    return False

if solve_nqueens(board, 0):
    for row in board:
        print(" ".join("Q" if x==1 else "*" for x in row))
else:
    print("No solution exists")
```

Output:

```
Enter the number of queens: 8
Q * * * * * * *
* * * * * * Q *
* * * * Q * * *
* * * * * * * Q
* Q * * * * * *
* * * Q * * * *
* * * * * Q * *
* * Q * * * * *
```

Result:

Thus the N queen program has been executed successfully.

Ex no: 4
7/9/24

Depth first search

Aim:

To implement Depth first search

Program:

```
def dfs_recursive(graph, start, visited = None):
    if visited is none:
        visited = set()
    visited.add(start)
    print(start)
    for neighbor in graph[start]:
        if neighbor is not in visited:
            dfs_recursive(graph, neighbor, visited)

graph = {
    'A': ['B', 'C'],
    'B': ['A', 'D', 'E'],
    'C': ['A', 'F'],
    'D': ['B'],
    'E': ['B', 'F'],
    'F': ['C', 'E']
}
Print("DFS recursive:")
dfs_recursive(graph, 'A')

def dfs_iterative(graph, start):
    visited = set()
    stack = [start]
    while stack:
        vertex = stack.pop()
        if vertex not in visited:
            print(vertex)
            visited.add(vertex)
            stack.extend(neighbor for neighbor in
            graph[vertex] if neighbor not in visited)

graph = {
    'A': ['B', 'C']
    'B': ['A', 'D', 'E']
    'C': ['A', 'F']
    'D': ['B']
    'E': ['B', 'F']
    F = ['C', 'E']
}
Print("DFS iterative:")
dfs_iterative(graph, 'A')
```

output:

```
DFS iterative
A
C
F
E
B
D
```

Result:

Thus the depth first search has been implement successfully

A* Algorithm

Aim:

To implement A* algorithm

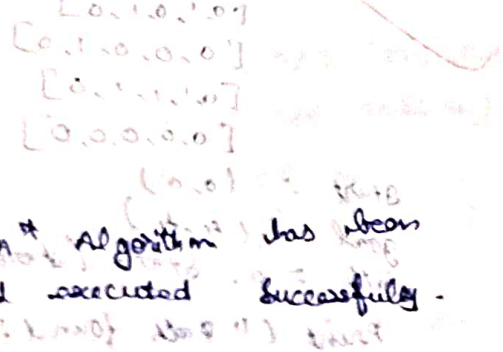
Program:

```
from heap import heappop, heappush

class node:
    def __init__(self, position, parent=None):
        self.position = position
        self.parent = parent
        self.g = 0
        self.h = 0
        self.f = 0
    def __eq__(self, other):
        return self.position == other.position
    def __lt__(self, other):
        return self.f < other.f
def a_star(start, goal, grid):
    start_node = node(start)
    goal_node = node(goal)
    open_list = []
    heappush(open_list, start_node)
while open_list:
    current_node = heappop(open_list)
closed_list.add(current_node.position)
    if current_node == goal_node:
    path = []
while current_node:
        path.append(current_node.position)
        current_node = current_node.
    return path[::-1]
neighbours = [(0,-1),(0,1),(-1,0),(1,0)]
for n in neighbour
    neighbor = position + (current_node.position[0] +
if 0 <= neighbour.position[1] < len(grid[0]) and
 grid[neigbour] position[0]][neighbor.position[1]]==0.
neighbor_node = node(neighbor_position, current node
if neighbor_node = position closed continue.
neighbour_node.g = current_node.g+1
neighbor_node.h = abs(neighbor_node.position[0]
        goal_node.position[0]) + abs(neighbor
        node.position[1] - goal node.position[1],
neighbor_node.f = neighbor_node.g neighbor_node.
    if all(neighbor_node != ona.node for ona in node
                                    open_list)
heappush(open_list, neighbor_node)
return none
grid = [
    [0,1,0,0,0]
    [0,1,0,1,0]
    [0,0,0,1,0]
    [0,1,1,1,0]
    [0,0,0,0,0]]
start = (0,0)
goal = (4,4)
path = a.star([start, goal, grid])
print("Path found:", path)
```

output:

Path found = [(0,0) (1,0) (2,0) (3,0) (4,0) (4,1)
(4,2) (4,3) (4,4)]



Result:

Thus the A* Algorithm has been explained and executed successfully.

Ex: 6
10/9/24

AO* Algorithm

Aim:

To implement AO* Algorithm

Program:

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

def aostar(self, node):
    print("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("Best Path for {node} = {best_path}
                    with cost {min_cost}")
for child in best_path
    self.aostar(child)
    self.aostar(child)
def get_solution(self)
    return self.solution
graph = {
```

```
'n' = [['B','C'],['A']]
'S' = [['E']]
'C' = [['G']]
'D' = [['b']]
'F' = []
'G' = []
'H' = []
}

heuristic = {
    'A' = 0, 'S' = 1, 'C' = 2, 'D' = 4, 'E' = 1, 'G' = 5 }
graph_obj = graph(graph, heuristic)
graph_obj = ao_star('n')
Solution = graph_obj.get_solution()
Print("Solution :", Solution)
```

output:

```
Expanding : A
Best Path for A = ['B','C'] with Best 3
Expanding : B
Best Path for B = ['E']with Best 1
Expanding = E
Expanding = C
Best Pat C = ['G']
Expanding : G
Solution : {A = [B,C], B = [E], C = ['G']}
```

Result:

Thus AO* Algorithm implemented successfully.

Ex:7 Implementation of Decision tree classification

25/9/24

Aim:

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

* Import the tree
* call the function from tree
* Assign value for x & y
* call the function on basics of given random values each give features
* Display the output.

Program:

```
import pandas as pd
from sklearn.tree import decision tree
data = {'Height': [152, 155, 172, 185, 167, 180,
                   164, 177]
'weight': [45, 57, 72, 85, 68, 78, 22, 88]
gender: ['female', 'female', 'male', 'male'
         'female', 'male', 'female', 'male',
         female]
df = pd.dataframe(data)
x = df['height', 'weight']
y = df['gender']
classifier = decision tree classifier
classifier = fit[x,y]
height = float(input("Enter height
                     in kg")for prediction)
weight = float(input("Enter weight (integer) for
                     prediction))
random_value = pd.Data frame([r, height, weight]]
column = ['Height', 'weight'])
```

Ex: 8 Implementation of ANN for an application using python - Regression.

9/10/24

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

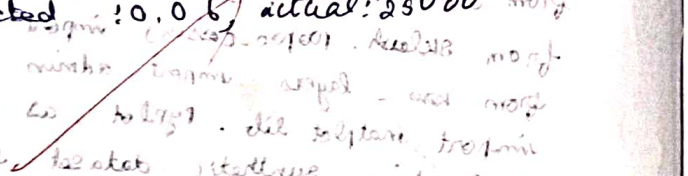
Program:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from keras.layers import Dense
import matplotlib.pyplot as plt
# generating synthetic dataset for demo
X = np.random.rand(1000, 3)
y = 3*X[:,0] + 2*X[:,1] + 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_size=0.2, random_state=42)
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
model = Sequential()
model.add(Dense(12, input_dim=X_train.shape[1], activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='linear'))
model.compile(optimizer='adam', loss='mean_squared_error')
history = model.fit(X_train, y_train, epochs=100, batch_size=32, verbose=0)
y_pred = model.predict(X_test)
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 & validation loss')
plt.xlabel('Epoch')
plt.ylabel('loss')
plt.legend()
plt.show()
```

output:

Predicted : 0.06, actual: 250.00



Result:

Thus the Python Program to implement artificial neural network for an application in regression is executed succesfully.

Ex:9 Implementation of K-means clustering technique

10/10/24

Aim:

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

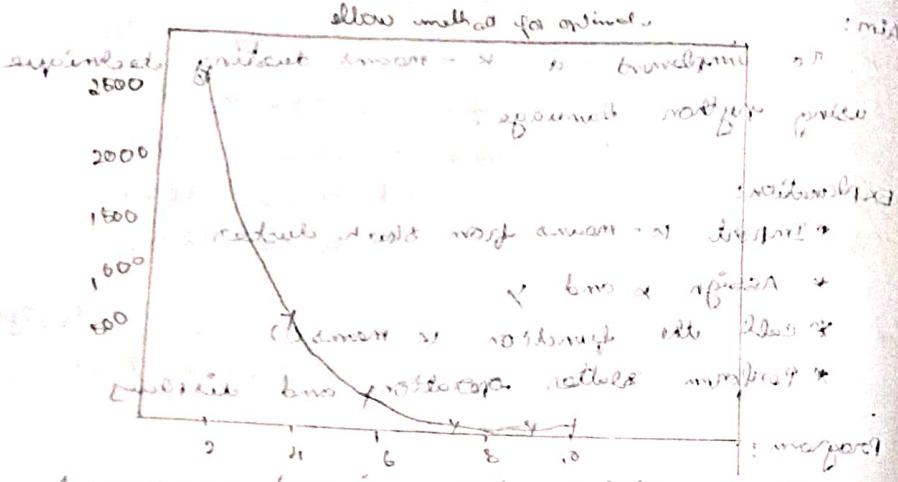
Explanation:

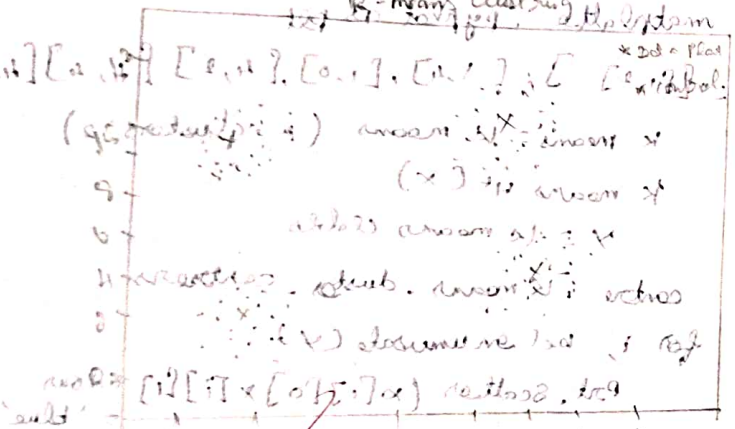
* Import K-means from sklearn.cluster
* Assign x and y
* call the function KMeans()
* Perform scatter operation and display

Program:

```
from sklearn.cluster import K-Means
import matplotlib.pyplot as plt
x = [[1,2] ], [1,4], [1,0], [4,2] [4,4][4,0]]
    k means = K means (n_clusters=2)
    k means fit(x)
      y = k means labels
    centre : K means . cluster_centers
for i, l in enumerate(y):
    plt.scatter (x[i][0] x [i][1] color
      = 'blue'
      if label == 0 else 'green')
plt.scatter (centers[:,0], center[:1];
    color = 'red', mark = 'x', s=100,
        label = 'centroids')
plt.xlabel ('Feature')
plt.Ylabel ('Feat2')
    plt.title ('K-means clustering')
    plt.legend()
    plt.show()
```

output:





Result:

Thus the python program to implement K-means clustering technique is executed successfully.

Ex :10
28/10/24

Minimax

Aim:

To implement minimax algorithm

Program:

```
import math

def minimax (depth, node_index, maximizes,
    scores, height):
    if depth == height:
        return scores[node_index]
    if depth == height:
        return scores [node_index]
    if a_maximizes:
        return max( minimax (depth-1, node_index*2,
            False, scores, height-) minimax)depth-1,nod
            False, scores, height ))
    else
        return min (minimax (depth-1, node_index*2,
            true, slcos, height ) minimax
            (depth+1, node, index*2+1, true, scores
            height))
def calculate_true_height (num_leaves):
    return math.ceil( math.log2 (num_leaves))
    scores = [3, 5, 6, 9, 1, 2, 0, -1]
tree_height = calculate_tree_height (len(scores)
optimal_score = minimax (0,0, true, scores, treeheight)
print ( f" The optimal score is {optimal_score}")
```

The optimal score is : 5

Result :

Thus the decision tree program has been executed successfully

Ex: 11

Aim :

To learn PROLOG terminologies and write basic programs

Terminologies :

1. Atomic terms :-

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

Ex.

dog
ab_c_321

2. Variables :

Variables are string of letters, digits and the underscore, starting with a capital 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, i.e atoms, number, variables, other compound terms) enclosed in parentheses and seperated by commas.

Ex : is bigger (elephant x)
f(g(x,-),7)

4. Facts:

A fact is predicate followed by a dot

Ex:

bigger - animal (whale)
life - is - beautiful.

5. Rules:

A rule consists of an 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:

KB1:

woman (mia)
woman (jody)
woman (yolanda)
plays Air guitar (jody)
Party.

Query 1: ? - woman (mia)
Query 2: ? - Play Air guitar (mia)
Query 3: ? Party
Query 4: concert.

Output:

? - woman (mia)
true
? - Plays Air guitar (mia)
false
? - party
true
! - concert
ERROR: Unknown procedure: concert/0

KB2

happy (yolanda)
listen 2 music (mia)
listen 2 music (yolanda) :- happy (yolanda)
plays guitar (mia) :- listen 2 music (mia)
plays guitar (yolanda) listen 2 music (yolanda)

o/p

? play airguitar (mia)
true
? - play guitar (yolanda)
true

KB3

likes (dan, sally)
likes (sally, dan)
likes (john, britney)
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, britney)
false

KB4

food (burger)
food (sandwich)
food (pizza)
lunch (sandwich)
dinner (pizza)
meal (x) :- food (x)

O/P

? - food (pizza)
true
! - meal (x), lunch (x)
x = sandwich

KB :

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))

O/P

? owns (john, x)
y = car (chevy)
? = owns (john, -)
true

Result:

Thus the basic prolog program has been executed successfully.

Ex. 12
6/11/24

Prolog family tree

Aim:

To develop a family tree program using prolog with all possible facts, rules of queries

Source code:

knowledge_base:
/* facts :: */
male (peter)
male (john)
male (chris)
male (kevin)
female (betty)
female (jeny)
female (jess)
female (helen)
parent of (chris, peter)
parent of (chris, betty)
parent of (kevin, chris)
parent of (jeny, john)
parent of (jeny, jean)
/* Rules ::*/
/* son, parent*/
/ son, grand parent*/
father (x,y) :- male (y), parent (x,y)
mother (x,y) :- female (y), parent (x,y)
grand father (x,y) : male (y), parent of (x,12),
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, z)

sister (x, y) : female (y), father (x, z), father (y, z)

z = 10

o/p male (peter)

true father (chris, peter)

true father (chris, betty)

false

mother (chris, x)

x = betty

brother (chris, betty)

false

Result:

Thus prolog for family tree program has been executed successfully.