2116 220701014

INDEX

NAME: AFRIN FATHIMA T.I SEC.: ROLL NO.: SUB.:

S. No.	Date	Title	~~Page No.~~ marks	Teacher's Sign / Remarks
1	31.7.2024	Basic Python Programmes in Google colab	10	
2	7.8.2024	Email Spam Detection	10	
3	4/9/2024	N-Queens	10	
4	11/9/2024	A* search	10	
5	18/9/2024	Depth First Search	10	
6,	18/9/2024	AO* Algorithm	10	
7,	25/9/2024	Decision Tree	10	
8,	9/10/2024	K-Means	10	
9,	16/10/2024	Artificial Neural Network	10	
10,	23/10/2024	MiniMax	10	
11,	30/10/2024	Introduction to Prolog	10	
12,	6/11/2024	Prolog family Tree	10	

completed

1) Palindrome:

```
word = input()
if word == word[::-1]:
    print(word, " is a palindrome")
else:
    print(word, " is not a palindrome")
```

2) Prime Number:

```
num = int(input("Enter a number:"))
flag = False

if num == 1:
    print(num, " is not a prime number")
elif num > 1:
    for i in range(2, num):
        if (num % i) == 0:
            flag = True
            break
    if flag:
        print(num, " is not a prime number")
    else:
        print(num, "is a prime number")
else:
    print(num, " is not a valid number to check
                for prime")
```

3, Fibonacci Series:

```
def fibonacci(n):
    fib_sequence = [0, 1]
    for i in range(2, n):
        next_number = fib_sequence[-1] + fib_sequence[-2]
        fib_sequence.append(next_number)
    return fib_sequence

n = int(input("Enter the number of fibonacci
                numbers to generate : "))
print(fibonacci(n))
```

4, Factorial of a number:

```
def factorial(n):
    if n==0:
        return 1
    else
        return n * factorial(n-1)
number = int(input("Enter a number :"))
print('Factorial of', n, 'is' number)
```

5, Bubble Sort:

```
def BubbleSort(arr):
    n=len(arr)
    for i in range(n):
        for j in range(0, n-i-1):
            if (arr[j] > arr[j+1]):
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr
numbers = list(map(int, input("Enter numbers by spaces:")
                                .split()))
sorted_array = BubbleSort(numbers)
print(sorted_array)
```

6) Armstrong Number:

```
num = int(input("Enter a number"))
sum = 0
temp = num
while temp > 0:
    digit = temp % 10
    sum += digit ** 3
    temp //= 10
if num == sum:
    print(num, "is an Armstrong number")
else:
    print(num, "is not an Armstrong number")
```

Output:

Enter a number: 153
153 is an Armstrong number.

7) Sum of n natural numbers:

```
num = int(input("Enter a number"))
if (num < 0)
    print("Enter a positive number")
else
    sum = 0
    while (num > 0):
        sum += num
        num = num - 1
    print("The sum is", sum)
```

Output

Enter a number: 10
The sum is 55

8) Largest of 3 nos:

```
num1 = int(input("Enter a number"))
num2 = int(input("Enter a number"))
num3 = int(input("Enter a number"))
if (num1 >= num2) and (num1 >= num3):
    largest = num1
elif (num2 >= num1) and (num2 >= num3):
    largest = num2
else
    largest = num3
print("The largest number is:", largest)
```

Output:

```
Enter a number: 12
Enter a number: 5
Enter a number: 54
The largest number is: 54
```

9) Swap 2 variables:

```
x = 5
y = 10
temp = x
x = y
y = temp
print('The value of x after swapping:', x))
print('The value of y after swaping:', y))
```

Output:

```
The value of x after swapping: 10
The value of y after swapping: 5
```

10) LCM of 2 input numbers:

```
def compute_lcm(x,y):
    if x > y:
        greater = x
    else:
        greater = y
    while (True):
        if ((greater % x == 0) and (greater % y == 0)):
            lcm = greater
            break
        greater += 1
    return lcm
num1 = 54
num2 = 24
print("The LCM is :", compute_lcm(num1, num2))
```

31/01/2024

Output:

The LCM is : 216

02/09/2024

N - QUEENS

PROGRAM:

```
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,
        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:

```
1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0
```

Result:

Thus the program was executed and the output was verified.

11/09/2024

A* Algorithm

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[
                + heuristic(n)):
                n = v
        if (n == stop_node or Graph_nodes[n] == None):
            pass
        else:
            for (m, weight) in get_neighbours(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)

        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)
    print ("Path does not exist!")
    return None
def get_neighbors(v):
    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), ('G', 9)],
                'C' : None,
                'E' : [('D', 6)],
                'D' : [('G', 1)],
              }
astaralgo('A', 'G')
```

OUTPUT:

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

Result :-

Thus the program was executed and output was verified

18/08/2024

DEPTH FIRST SEARCH

PROGRAM:

```
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)

def dfs(adj, s):
    visited = [False] * len(adj)
    dfs_rec(adj, visited, s)

V=5
adj = [[] for _ in range(V)]
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:

```
DFS from source: 1
1 2 0 3 4
```

RESULT:

Thus the program was executed and the output was verified.

IMPLEMENTING ARTIFICIAL NEURAL NETWORKS FOR AN APPLICATION USING PYTHON - REGRESSION

EX NO:

DATE 25/9/2024

AIM:

To implement artificial neural networks for an application in Regression using python.

EXPLANATION:

* Generate synthetic regression data using make_regression with 1000 samples, 100 features and noise of 0.05.

* Split data into training (80%) and testing (20%) sets using train_test_split.

* Initialize the MLPRegressor model with a maximum of 1000 iterations.

* Fit the model to the training data using clf.fit(X, Y)

* Evaluate the model's performance by calculating & printing the R^2 score for both the training and testing datasets.

SOURCE CODE:

```
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_regression
import numpy as np.
import matplotlib.pyplot as plt
import seaborn as sns
% matplotlib inline
X, y = make_regression (n_samples = 1000, noise=0.05, n_features=
100)
```

```
X_train, X_test, y_train, y_test = train_test_split(X,
                                   test_size=0.2, shuffle=
                                   random_state=42)

clf = MLPRegressor(max_iter=1000)
clf.fit(X_train, y_train)
print(f"R2 score for Training Data : {clf.score(X_train,
                                     y_train)}

printf(f"R2 score for Test Data : {clf.score(X_test, y_test,
```

OUTPUT:

```
R2 score for Training Data : 0.9999875908849684
R2 Score for Test Data : 0.9808550368018144
```

RESULT:

The program was executed and output was verified

IMPLEMENTATION OF DECISION TREE CLASSIFICATION TECHNIQUES

EX. NO:

DATE : 9/10/2024

AIM:

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

EXPLANATION:

* Import tree from sklearn
* Call the function DecisionTreeClassifier() from tree
* Assign values for X and Y.
* Call the function predict for predicting on the basis of given random values for each given feature.
* Display the output.

Source code:

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
data = { 'Height' : [152, 172, 185, 167, 180]
         'Weight' : [45, 57, 72, 85, 68]
         'Gender' : ['Female', 'Female', 'Male', 'Male', 'Female']
       }
df = pd.DataFrame(data)
X = df[['Height', 'Weight']]
Y = df['Gender']

Classifier = DecisionTreeClassifier()
Classifier.fit(X, Y)

height = 159
weight = 51
prediction_data = pd.DataFrame([[height, weight]],
                               columns = ['Height', 'weight'])
```

```
predicted_gender = classifier.predict(prediction)
print(f"Predicted gender: {predicted_gender[0]}")
```

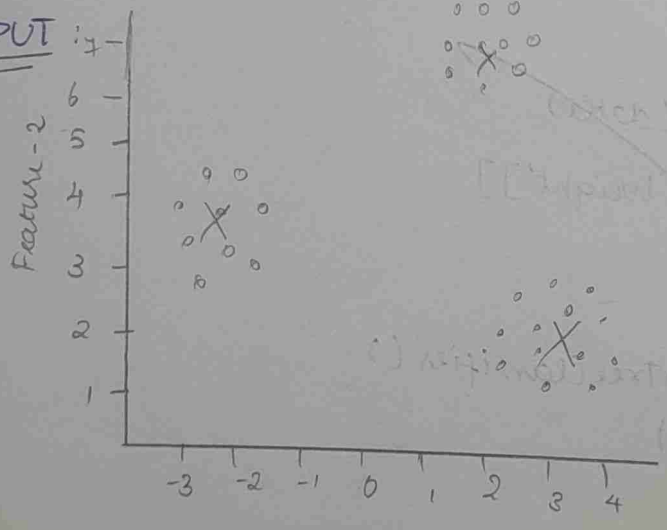
OUTPUT:

Predicted gender: Female

RESULT:

The program was executed and output was verified

OUTPUT:



O clusters
X centroids

IMPLEMENTATION OF CLUSTURING TECHNIQUES —

K-MEANS

EX. NO:

DATE : 16/10/2024

AIM:

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

EXPLANATION:

* Import K-means from sklearn.cluster
* Assign X & Y
* Call the function KMeans()
* Perform scatter operation and display the output.

SOURCE 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=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, marker='x', label='centroids')
plt.title('KMeans Clustering Results')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```

RESULT: Thus the program was executed and output was verified.

AO* ALGORITHM

AIM:

To implement AO* algorithm using python Programming language.

CODE:

```
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.gra
                                        [node]:
            return
        children = self.graph[node]
        best_path = None
        min_cost = float("inf")
        for group in children:
            cos = sum(self.heuristic[child] for chil
                                    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} wit
                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.get_solution()
print("Solution:", solution)
```

OUTPUT :

Expanding : A

Best Path for A : ['B', 'C'] with cost 3

Expanding : B

Best path for B : ['E'] with cost 1

Expanding : E

Expanding : C

Best path for C : ['G'] with cost 3

Expanding : G

Solution : {'A' : ['B', 'C'], 'B' : ['E'], 'C' : ['G']}

RESULT:

Thus the program to implement AO* algorithm was executed and the output was verified.

23/10/2024

MINI MAX ALGORITHM

AIM:

To implement Mini max algorithm using python language.

CODE:

```
import math
def minimax(depth, node_index, is_maximizer, Scores,
                                                height):
    if depth == height:
        return scores[nodes_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,
            True, scores, height),
        minimax(depth+1, node_index*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, 0, -1]
tree_height = calculate_tree_height(len(scores))
Optimal_score = minimax(0, 0, True, Scores, tree_hieght)
print(f"the optimal Score is: {optimal_score}")
```

OUTPUT:

The optimal sore is : 5

RESULT:

Thus the minimax algorithm was executed successfully and the output was verified.

30/10/2024

INTRODUCTION TO PROLOG

AIM:

To learn prolog terminologies and write basic programs.

TERMINOLOGIES:

1) Atomic Terms:

Atomic Term is usually string made up of lower & upper case letters, digits and the underscore, starting with a lowercase letter.

ex: dog abc-321

2, Variables:

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

ex: Dog Apple-420

3, Compound Terms:

Compound terms are used to make up of a PROLOG atom & a no. of arguments (PROLOG terms) and enclosed in paranthesis and seperated by commas.

Ex: is_bigger (elephant, x)

4) Facts:

A fact is a pseudocode followed by a dot.

Ex: bigger_animal (whale)

5) Rules:

A rule of a head (a predicate) and a body (a sequence)

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

SOURCE CODE:

KB1:

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

O/P:

```
?- woman(mia)
true
?- playsAirGuitar(mia)
false
?- party
true
?- concert
procedure concert doesn't exist.
```

SOURCE CODE :

KB2 :

```
happy (yolanda)
listen2music (mia)
listen2music (yolandra) :- happy (yolandra)
playsAirGuitar (mia) :- listen2music(mia)
playsAirGuitar (yolandra) :-listen2music (yolandra).
```

O/P:

```
? - playsAirGuitar (mia)
    true
? - playsAirGuitar (yolandra)
    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)
```

OP:

```
? - likes (dan, X)
    X = sally
? - married (dan, sally)
    true
? - married ( john , brittney)
    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
?- dinner (sandwich)
   false
```

KB5:

```
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)
 x - car (chevy)
?- own (john,-)
   true
?- owns (who, car (chevy))
   who = john
```

```
? - owns(jane, X), sedan(X)
    false
? - owns(jane, X), truck(X)
    X = car(chevy)
```

RESULT:

Thus the program was executed and output was verified.

PROLOG FAMILY TREE

AIM:

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

SOURCE CODE:

KNOWLEDGE BASE:

male (peter)
male (John)
male (chris)
male (kevin)

female (betty)
female (jeny)
female (lisa)
female (helen)

parentOf (chris, peter)
parentOf (chris, betty)
parentOf (helen, peter)
parentOf (helen, betty)
parentOf (kevin, chris)
parentOf (kevin, lisa)
parentOf (jeny, john)
parentOf (jeny, helen)

RULES:

father (X, Y) :- male (Y), parentOf (X, Y)
mother (X, Y) :- female (Y), parentOf (X, Y)
grandfather (X, Y) :- male (Y), parentOf (X, Z), parentO
grandmother (X, Y) :- female (Y), parentOf (X, Z), parentOf (Z

brother(X,Y) :- male(Y), female(X,Z), female(Y,W),
Z == W

sister(X,Y) :- female(Y), female(X,Z), female(Y,W), Z
== W

male(Y), parentOf(X,Y)

X = chris, Y = peter

female(Y), parentOf(X,Y)

X = chris, Y = betty

male(Y), parentOf(X,Z), parentOf(Z,Y)

X = kevin, Y = peter, Z = chris

female(Y), parentOf(X,Z), parentOf(X,Y)

X = kevin, Y = betty, Z = chris

RESULT:

The program was executed and output was verified.