INDEX

NAME: A. Akilan STD: CSE SEC: A ROLL NO. 2020102022 PO A1

S.No.	Date	Title	Marks	Teacher's Sign/ Remarks
1	31/7	Basic Python programs using google colab	9	
2	7/8	Domain: Sentiment analysis.	10	
3	4/9	Nqueens	10	
4	4/9	Depth first search.	10	
5	11/9	A* algorithm	10	
6	18/9	AO* algorithm	10	
7	25/9	Decision tree	10	
8	9/10	K-means	10	
9	16/10	Artificial neural network	10	
10	23/10	Minimax	10	
11	30/10	Introduction to prolog	10	
12	6/11	Prolog - family tree	10	

completed

Basic Python Programs using Google Colab

1) Factorial

```
Print ("Enter the number")
n = int (input())
f = 1
for i in range (1, n+1):
f = f * i
Print (f)
```

output:
Enter the number
3
6

2) Calculator

```
Print ("Enter the first number")
n1 = int (input())
Print ("Enter the second number")
n2 = int (input())
Print ("Enter the arithmetic operation")
o = input()
if (o == "+"):
  Print (n1+n2)
elif (o == '-'):
  Print (n1 - n2)
elif (o == '*'):
  Print (n1 * n2)
elif (o == '/'):
  Print (n1/n2)
```

Output:
Enter the first number
5
Enter the second number
7
Enter the arithmetic operation
*
35

3) odd or even

```
num = int (input ("Enter a number:"))
if num % 2 == 0:
    print (num, "is even")
else:
    print (num, "is odd")
```

Output

Enter a number: 5

5 is odd

4) Prime or non-prime

```
num = int (input ("Enter a number:"))
if num > 1:
    for i in range(2, num):
        if (num % i) == 0:
            print (num, "is not a prime number")
            break
    else:
        Print (num, "is a prime number")
else:
    print (num, "is not a prime number")
```

Output:

Enter a number: 5

5 is a prime number

5)

```
num_terms = int (input ("How many terms?"))
n1, n2 = 0, 1
count = 0
if num_terms <= 0:
    Print ("please enter a positive integer")
elif num_terms == 1:
    Print ("Fibonacci sequence up to", num_terms, ":")
    Print (n1)
else:
    Print ("Fibonacci sequence:")
    while count < num_terms:
        Print (n1)
        nth = n1 + n2
        n1 = n2
        n2 = nth
        count += 1
```

Output:

How many terms? 4

Fibonacci sequence:

0

1

1

2

6) Palindrome

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

Output

Enter a word: ~~nan~~ nalan

nalan is a Palindrome

7) Variable swapping

```
a = int (input ("Enter first number:"))
b = int (input ("Enter second number:"))
print ("Before swap: a =", a, "b =", b)
a, b = b, a
Print ("After swap: a =", a, "b=", b)
```

Output

Enter first number: 3

Enter second number: 45

Before swap: a = 3 b = 45

After swap: a = 45 b = 3

8) Leap year

```
Year = int (input ("Enter a year:"))
if (year % 4 = =0 and year % 100 . ! =0) or (year % 400 = =0):
    Print (year, "is a leap year")
else:
    print (year, is not a leap year")
```

output:

Enter a year: 2024

2024 is a leap year

9) 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: 1026

1026 is not an armstrong number

10) Sum of natural numbers.

```
num = int (input ("Enter a number: "))
sum = 0
for i in range (1, num+1):
    sum += i
print ("The sum of natural numbers up to", num, "is:", sum)
```

Output:

Enter a number: 5

The sum of natural numbers up to 5 is: 15

31/01/24

Domain: Sentimental analysis.

Sentiment analysis, a key application within Natural Language Processing (NLP), is an advanced computational technique aimed at deciphering the emotional tone embedded in textual data. This domain, a subset of artificial Intelligence (AI), focuses on the automated classification of text into various sentiment categories such as positive, negative and neutral, and extends to more nuanced emotional states like joy, anger or sadness. The primary objective of sentiment analysis is to understand and interpret how individuals feel about specific topics, products, or services based on their written expressions.

Problem Statement:

Design and implement an automated sentiment analysis system that can accurately and efficiently classify feedback into sentiment categories - such as positive, negative and neutral

-and detect nuanced emotions such as anger, happiness, or frustration. This system must be capable of handling large datasets, dealing with linguistic ambiguities, and providing actionable insights in real-time to support improved decision-making and enhance customer experience.

Applications:

1) Business and Companies.
   * To know the feedback and response of their product.
2) Social media analysis.
   * To analyse about their post and advertisements.
3) Political analysts and Campaigns.
   * Public opinion on policies, candidates and political events to shape campaign strategies and messaging.

Algorithm

Support Vector Machine (SVM).

Nqueens Problem

AIM:

To implement Nqueens 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,1), 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 NQueens program has been executed successfully.

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 not in visited:
            dfs_recursive(graph, neighbour, 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 neighbour 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 implemented successfully.

A* algorithm

11/08/2024

Aim:

To implement A* algorithm

Program:

```
from heapq 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 = []
    closed_list = set()
    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.parent
return path[::-1]

neighbours = [(0,-1), (0,1), (-1,0), (1,0)]
for n in neighbours:
    neighbor_position = (current_node.position[0]+n[0],
                         current_node.position[1]+n[1])
    if 0<=neighbor_position[0]<len(grid) and 0<=neighbor
        _position[1]<len(grid[0]) and grid[neighbor
        position[0]][neighbor_position[1]]==0:
        neighbor_node = Node(neighbor_position, current_node)
        if neighbor_node.position in closed_list:
            continue
        neighbor_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.h
        if all(neighbor_node != open_node for open_node in
                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 implemented and executed successfully

18/9/24

AO* algorithm

AIM:
To implement AO* algorithm

PROGRAM:

```
class Graph:
    def_init_(self, graph, heuristic)
    self.graph = graph
    self.heuristic = heuristic
    self.solution = {}

    def 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.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 c : ['G']

expanding : G

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

RESULT:

Thus AO* algorithm implemented successfully.

Implementation of decision tree classification techniques.

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.

Program:

```
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 169.0 cm and weight 61.0 kg: female.
```

Result:

Thus the decision tree classification has been executed successfully.

Implementation of clustering techniques k-means

AIM:

To implement a k-Means clustering technique using Python language.

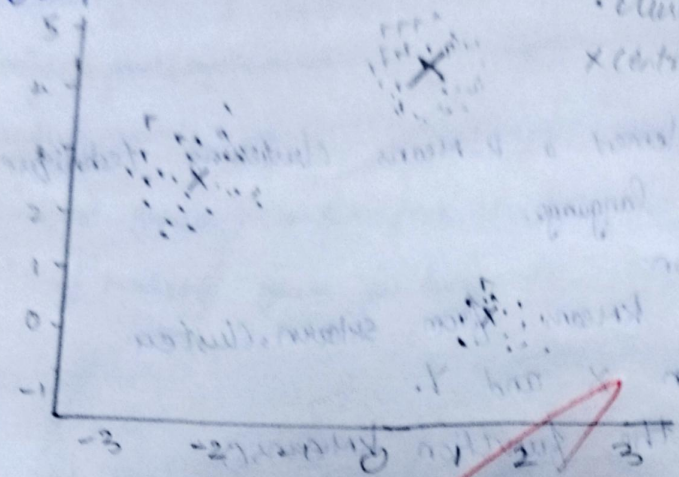
Explanation:

* Import kMeans from sklearn.cluster
* Assign X and Y.
* Call the function kMeans().
* Perform scatter operation and display the output.

Program:

```
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(fig size=(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('k-means. Clustering results')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```

Output



Result

Thus the K-means clustering technique has been implemented successfully

Implementation artificial neural networks for an application using Python - Regression

AIM:
To implementing artifical neural network 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.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)
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_size=
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.dd(Dense(16,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='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

Output.

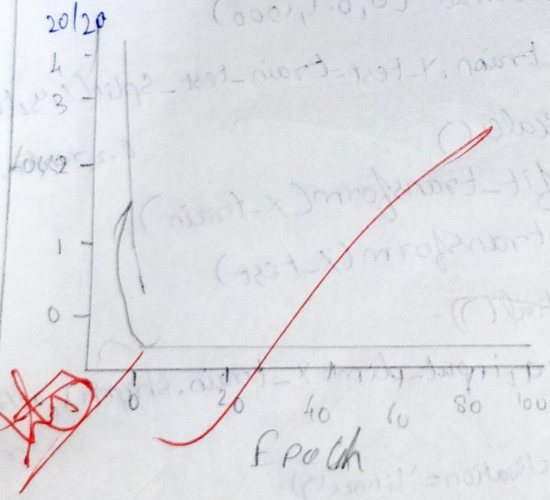
Epoch 1/100

20/20

⋮

Epoch 100/100

20/20



Result

Thus the artificial neural networks for regression has been implemented successfully

MINIMAX ALGORITHM

AIM:

To implement MINIMAX algorithm.

PROGRAM:

```
import math
def minimax(depth, node_index, is_maximizes, scores,
            height):
  if depth == height :
    return scores [node_index]
  if is_maximizes:
    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_height)
print(f"The optimal score is: {optimal_score}")
```

Output-

The optimal score is = 5

Result

Thus the decision tree program has been executed successfully.

Introduction to Prolog.

AIM:

To learn PROLOG terminologies and write basic programs.

TERMINOLOGIES:

1) Atomic terms:

They are usually strings made up of lower- and uppercase letters, digits and the underscore, starting with a lowercase letter.

eg: dog, ab_c_321

2) Variables:

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

Eg: DOG, Apple_420

3) Compound terms:

Compound terms are made up of a PROLOG atom and a number of arguments enclosed in parentheses and separated by commas.

eg: is_bigger(elephant, X). f(g(X, _), 7)

4) Fact

A fact is a predicate followed by a dot.

eg: bigger_animal(whale). life_is_beautiful.

5) Rules

A rule consists of a head and body.

eg: is_smaller(X, Y) :- bigger(Y, X). aunt(Aunt, Child) :- sister(Aunt, Parent), Parent(Parent, Child).

Source code:

KB1

```
woman(mia).
woman(jody).
woman(yolanda).
playsAirGuitar(jody).
party.
Querty 1: ?- woman(mia)
Querty 2: ?- playsAirguitar(mia)
Querty 3: ?- party
Querty 4: ?- concert.
```

output:

```
?- woman(mia)
true
?- playsairguitar(mia)
false
?- concert
Error: unknown procedure: concert/0
```

KB2

```
happy(yolanda).
listens2music(mia).
listens2music(yolanda):- happy(yolanda).
playsairguitar(mia):- listens2music(mia).
playsairguitar(yolanda):- listens2music(yolanda).
```

output:

```
?- playsairguitar(mia)
true.
?- playsairguitar(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).
```

output

?-likes(dan,X)

X = Sally

?-married (dan, sally)

true

?-married (john, brittney)

false.

KB4

food(burgers).

food(sandwich).

food(pizza).

lunch (sandwich).

dinner (pizza).

meal(X) :- food(X)

output.

?- food (pizza)

true

?- meal (X), lunch (X)

X = sandwith.

KB5

owns(jack, car(bmw)).

owns(john, car(chevy)).

own(olivia, car(civic)).

own(jane, car(chevy)).

Sedan (car (bmw)).

Sedan (car (civic)).

truck (car(chevy)).

output.

? owns (john, X)

X = car(chevy)

?- owns(john, _).

true.

Result

Thus the basic prolog programs has been executed successfully.

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 (helen, peter).
Parent of (helen, betty).
Parent of (kevin, chris).
parent of (kevin, lisa).
Parent of (jeny, john).
Parent of (jeny, helen).
/* RULES ::*/
/*son, Parent*/
*son, grandparent*/
```

father (x, y) :- male (y), parentof (x, y).

mother (x, y) :- female (y), parent of (x, y).

grandfather (x, y) :- male (y), parentof (x, z), parent of (z, y)

grandmother (x, y) :- female (y), parentof (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, peter)

true

father (chris, betty)

false.

mothe (chris, x)

x = betty

brother (chris, helen)

false.

Result -

thus prolog for family tree program has been executed successfully.