RASHTRASANT TUKADOJI MAHARAJ NAGPUR UNIVERSITY, NAGPUR

FOUR YEAR BACHELOR OF TECHNOLOGY (B.Tech.) DEGREE COURSE

SEMESTER: SIXTH (C.B.C.S)

BRANCH: INFORMATION TECHNOLOGY

Subject Name: Artificial Intelligence and Machine Learning Lab **Subject Code: BTIT602P**

Load	Practical	Credits	College Assessment Marks	University Evaluation	Total Marks
2Hrs (Practical)	2	1	25	25	50

Aim: 1) To apply AI and ML algorithms for real life problems

Prerequisite(s): Basic knowledge of computers, mathematics and programming languages

Course Objectives

1	**To understand the fundamental concepts of AI and ML**
2	**To develop programs based on AI and ML algorithms**
3	**To understand applications of AI and ML in real life**

Course Outcome:

At the end of this course students are able to:

CO1	**Understand the concept of Artificial Intelligence**
CO2	**Familiarize with Knowledge based AI systems and approaches**
CO3	**Apply the aspect of Probabilistic approach to AI**
CO4	**Identify and apply Neural Networks and NLP in designing AI models**
CO5	**Recognize the concepts of Machine Learning and its deterministic tools**

Practical:- Minimum 8 practicals based on the syllabus of AI and ML

K. D. K. COLLEGE OF ENGINEERING, NAGPUR

IMPARTING TECHNICAL EDUCATION SINCE 1984

NBA Accredited - Five programs:(Civil Engg., Mechanical Engg., Electrical Engg. Comp. Sc. & Engg., Electronics & Telecomm. Engg.) Grade A, NAAC Accredited

---

Department of Information Technology

Semester: VI

Name of Subject: Artificial Intelligence and Machine Learning Lab

Subject Code: BTIT602P

Course Outcome:

At the end of this course students are able to:

CO1	Understand the concept of Artificial Intelligence
CO2	Familiarize with Knowledge based AI systems and approaches
CO3	Apply the aspect of Probabilistic approach to AI
CO4	Identify and apply Neural Networks and NLP in designing AI models
CO5	Recognize the concepts of Machine Learning and its deterministic tools

LIST OF PRACTICALS

1	Write a Program to Implement Breadth First Search Algorithm in Python.
2	Write a Program to Implement Depth First Search Algorithm in Python.
3	Write a program in Python to Implement Tic-Tac-Toe game.
4	To Generate a random Lower case and Upper case letter password of any length in python.
5	Write a program in Python to Implement Markov Chain Model.
6	Write a Python Program to Implement shorted path for Directed Acyclic graph in Bayesian network.
7	Implementation of Linear Regression Technique using Decision Tree in Python.
8	Implementation Of Any Game By using Reinforcement learning technique in Python.
9	Create a Neural Network from Scratch in Python.
10	Implementing the Bag Of Words Natural Language Processing Algorithm in Python.

Prof. R. R. Kolte
Subject Teacher

Dr.S.P. Khandait
Head of Department

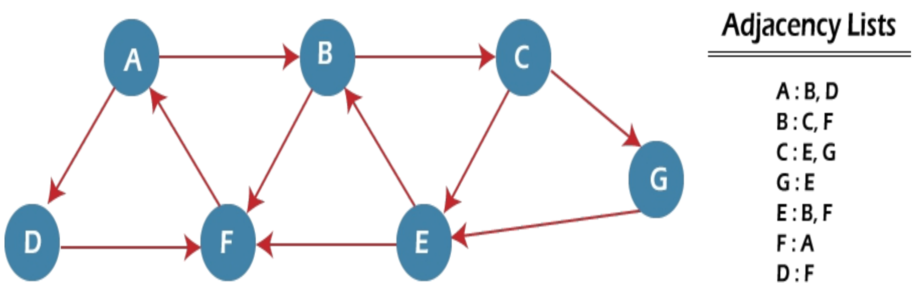
EXPERIMENT NO: 1

AIM: Write A Program To Implement Breadth First Search Algorithm In Python.
THEORY: Breadth First Traversal (or Search) for a graph is similar to Breadth First Traversal of a tree. The only catch here is, unlike trees, graphs may contain cycles, so we may come to the same node again. To avoid processing a node more than once, we use a boolean visited array. For simplicity, it is assumed that all vertices are reachable from the starting vertex.

Breadth-first search (BFS) is an algorithm used for tree traversal on graphs or tree data structures. BFS can be easily implemented using recursion and data structures like dictionaries and lists.

EXAMPLE OF BFS ALGORITHM

Now, let's understand the working of BFS algorithm by using an example. In the example given below, there is a directed graph having 7 vertices.



In the above graph, minimum path 'P' can be found by using the BFS that will start from Node A and end at Node E. The algorithm uses two queues, namely QUEUE1 and QUEUE2. QUEUE1 holds all the nodes that are to be processed, while QUEUE2 holds all the nodes that are processed and deleted from QUEUE1.
Now, let's start examining the graph starting from Node A.
Step 1 - First, add A to queue1 and NULL to queue2.

QUEUE1 = {A} QUEUE2 = {NULL}

Step 2 - Now, delete node A from queue1 and add it into queue2. Insert all neighbors of node A to queue1.

QUEUE1 = {B, D}QUEUE2 = {A}

Step 3 - Now, delete node B from queue1 and add it into queue2. Insert all neighbors of node B to queue1.

QUEUE1 = {D, C, F}QUEUE2 = {A, B}

Step 4 - Now, delete node D from queue1 and add it into queue2. Insert all neighbors of node D to queue1. The only neighbor of Node D is F since it is already inserted, so it will not be inserted again.

QUEUE1 = {C, F} QUEUE2 = {A, B, D}

Step 5 - Delete node C from queue1 and add it into queue2. Insert all neighbors of node C to queue1.

QUEUE1 = {F, E, G} QUEUE2 = {A, B, D, C}

Step 5 - Delete node F from queue1 and add it into queue2. Insert all neighbors of node F to queue1. Since all the neighbors of node F are already present, we will not insert them again.

QUEUE1 = {E, G} QUEUE2 = {A, B, D, C, F}

Step 6 - Delete node E from queue1. Since all of its neighbors have already been added, so we will not insert them again. Now, all the nodes are visited, and the target node E is encountered into queue2.

QUEUE1 = {G}

QUEUE2 = {A, B, D, C, F, E} COMPLEXITY OF BFS ALGORITHM

Time complexity of BFS depends upon the data structure used to represent the graph. The time complexity of BFS algorithm is **O(V+E)**, since in the worst case, BFS algorithm explores every node and edge. In a graph, the number of vertices is O(V), whereas the number of edges is O(E).

PROCEDURE OF EXECUTION:

- Open Anaconda Navigator
- Launch Notebook Jupyter
- Open New File Python Kernel
- Type the Program
- Click On Run Button

PROGRAM CODE:

BFS algorithm in

Pythonimport collections # BFS

algorithm

def bfs(graph, root):

visited, queue = set(),

collections.deque([root])visited.add(root) while

queue:

Dequeue a vertex from queuevertex = queue.popleft() print(str(vertex) +

" ", end="") # If not visited, mark it as visited, and# enqueue it

for neighbour in graph[vertex]:if neighbour not in visited: visited.add(neighbour)

queue.append(neighbour)if_name

__== '________main__':

graph = {0: [1, 2], 1: [2], 2: [3], 3: [1, 2]}

print("Following is Breadth First Traversal: ")bfs(graph, 0)

OUTPUT:

Following is Breadth First Traversal:
0 1 2 3

CONCLUSION: In This Practical, We Have Discussed the Breadth- First Search Technique Along With Its Example, Complexity, And Implementation In Python Programming Language. Hence, the BFS Algorithm executed successfully.

VIVA QUESTIONS:

1) **What is Breadth-First Search (BFS)?**
2) **How does BFS differ from Depth-First Search (DFS)?**
3) **What data structures are commonly used in BFS?**
4) **What is the time and space complexity of BFS?**
5) **Can BFS be used to find the shortest path in a graph?**
6) **How does BFS handle disconnected graphs?**

EXPERIMENT NO: 2

AIM: Write A Program To Implement Depth First Search Algorithmin Python.

THEORY: It is a recursive algorithm to search all the vertices of a tree data structure or a graph. The depth- first search (DFS) algorithm starts with the initial node of graph G and goes deeper until we find the goal node or the node with no children.

Because of the recursive nature, stack data structure can be used to implement the DFS algorithm. The process of implementing the DFS is similar to the BFS algorithm.

THE STEP BY STEP PROCESS TO IMPLEMENT THE DFS TRAVERSAL IS GIVEN AS FOLLOWS –

1. First, create a stack with the total number of vertices in the graph.
2. Now, choose any vertex as the starting point of traversal, and push that vertex into the stack.
3. After that, push a non-visited vertex (adjacent to the vertex on the top of the stack) to thetop of the stack.
4. Now, repeat steps 3 and 4 until no vertices are left to visit from the vertex on the stack's top.
5. If no vertex is left, go back and pop a vertex from the stack.
6. Repeat steps 2, 3, and 4 until the stack is empty.

ALGORITHM

Step 1: SET STATUS = 1 (ready state) for each node in G

Step 2: Push the starting node A on the stack and set its STATUS = 2 (waiting state)

Step 3: Repeat Steps 4 and 5 until STACK is empty

Step 4: Pop the top node N. Process it and set its STATUS = 3 (processed state)

Step 5: Push on the stack all the neighbors of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2 (waiting state)

[END OF LOOP]
Step 6: EXIT

EXAMPLE OF DFS ALGORITHM

Now, let's understand the working of the DFS algorithm by using an example. In the example given below, there is a directed graph having 7 vertices.

Now, let's start examining the graph starting from Node H.

Step 1 - First, push H onto the stack.

STACK: H

Step 2 - POP the top element from the stack, i.e., H, and print it. Now, PUSH all the neighbors of H onto the stack that are in ready state.

Print: H]STACK: A

Step 3 - POP the top element from the stack, i.e., A, and print it. Now, PUSH all the neighbors of A onto the stack that are in ready state.

Print: A STACK: B, D

Step 4 - POP the top element from the stack, i.e., D, and print it. Now, PUSH all the neighbors of D onto the stack that are in ready state.

Print: D STACK: B, F

Step 5 - POP the top element from the stack, i.e., F, and print it. Now, PUSH all the neighbors of F onto the stack that are in ready state.

Print: F STACK: B

Step 6 - POP the top element from the stack, i.e., B, and print it. Now, PUSH all the neighbors of B onto the stack that are in ready state.

Print: B STACK: C

Step 7 - POP the top element from the stack, i.e., C, and print it. Now, PUSH all the neighbors of C onto the stack that are in ready state.

Print: C STACK: E, G

Step 8 - POP the top element from the stack, i.e., G and PUSH all the neighbors of G onto the stack that are in ready state.

Print: G STACK: E

Step 9 - POP the top element from the stack, i.e., E and PUSH all the neighbors of E onto the stack that are in ready state.

Print: ESTACK:

COMPLEXITY OF BFS ALGORITHM

The time complexity of the DFS algorithm is **O(V+E)**, where V is the number of vertices and E is the number of edges in the graph.

The space complexity of the DFS algorithm is O(V).

PROCEDURE OF EXECUTION:

1) Open Anaconda Navigator
2) Launch Notebook Jupyter
3) Open New File Python Kernel
4) Type the Program
5) Click On Run Button

PROGRAM CODE:

Python3 program to print DFS traversal# from a given graph from collections

import default dict # This class represents a directed graph using# adjacency list

representation

class Graph:

Constructor
def_______init___(self):

default dictionary to store graphself.graph = defaultdict(list)

function to add an edge to graphdef addEdge(self,
u, v): self.graph[u].append(v)# A function used by
DFS

def DFSUtil(self, v, visited):

Mark the current node as visited# and
print it visited.add(v) print(v, end=' ')

Recur for all the vertices# adjacent to this vertex
for neighbour in self.graph[v]:if neighbour not in
visited: self.DFSUtil(neighbour, visited)

The function to do DFS traversal. It uses#
recursive DFSUtil() def DFS(self, v):

Create a set to store visited verticesvisited = set()

Call the recursive helper function# to print DFS traversal self.DFSUtil(v, visited)
Driver code
Create a graph given # in the above diagramg = Graph() g.addEdge(0,

```
1) g.addEdge(0, 2)
g.addEdge(1, 2)
g.addEdge(2, 0)
g.addEdge(2, 3)
g.addEdge(3, 3)

print("Following is DFS from (starting from vertex 2)")g.DFS(2)
```

OUTPUT:

Following is Depth First Traversal (starting from vertex 2)2 0 1 3

CONCLUSION : In This Practical, We Have Discussed The Depth-First Search Technique Along With Its Example, Complexity, And Implementation In Python Programming Language. Thus The Dfs Algoritm Executed Successfully.

VIVA QUESTIONS:

1) **What is Depth-First Search (DFS)?**
2) **How does DFS differ from Breadth-First Search (BFS)?**
3) **What data structures are commonly used in DFS?**
4) **What is the time and space complexity of DFS?**
5) **Can DFS be used to find the shortest path in a graph?**

EXPERIMENT NO: 3

AIM: Write A Program In Python To Implement Tic-Tac-Toe Game Using Adversarial Search Algorithm.

THEORY: AI Adversarial search: Adversarial search is a game-playing technique where the agents are surrounded by a competitive environment. A conflicting goal is given to the agents (multiagent). These agents compete with one another and try to defeat one another in order to win the game. Such conflicting goals give rise to the adversarial search. Here, game-playing means discussing those games where human intelligence and logic factor is used, excluding other factors such as luck factor. Tic-tac-toe, chess, checkers, etc., are such type of games where no luck factor works, only mind works.

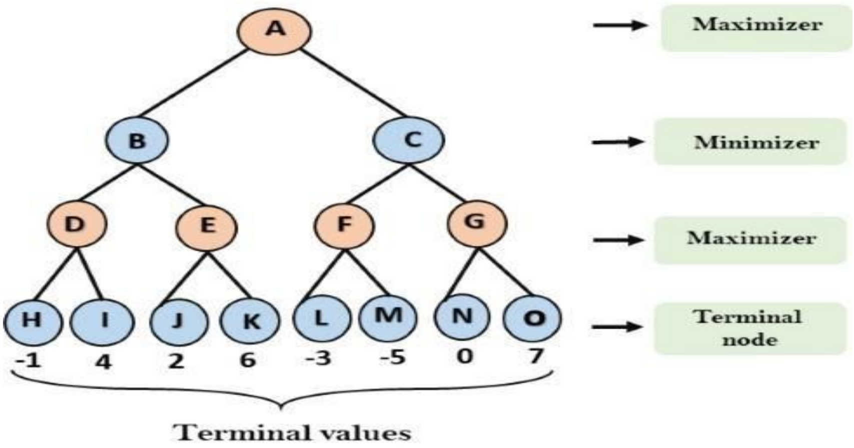
MINI-MAX ALGORITHM IN ARTIFICIAL INTELLIGENCE

- Mini-max algorithm is a recursive or backtracking algorithm which is used in decision- making and game theory. It provides an optimal move for the player assuming that opponent is also playing optimally.
- Mini-Max algorithm uses recursion to search through the game-tree.
- Min-Max algorithm is mostly used for game playing in AI. Such as Chess, Checkers, tic-tac- toe, go, and various tow-players game. This Algorithm computes the minimax decision for the current state.
- In this algorithm two players play the game, one is called MAX and other is called MIN.
- Both the players fight it as the opponent player gets the minimum benefit while they get the maximum benefit.
- Both Players of the game are opponent of each other, where MAX will select the maximized value and MIN will select the minimized value.
- The minimax algorithm performs a depth-first search algorithm for the exploration of the complete game tree.
- The minimax algorithm proceeds all the way down to the terminal node of the tree, then backtrack the tree as the recursion.

- Working of Min-Max Algorithm with Example
- The working of the minimax algorithm can be easily described using a example.Below we have taken an example of game-tree which is representing the two-player game.
- In this example, there are two players one is called Maximizer and other is called Minimizer.
- Maximizer will try to get the Maximum possible score, and Minimizer will try to get the minimum possible score.
- This algorithm applies DFS, so in this game-tree, we have to go all the way through the leaves to reach the terminal nodes.

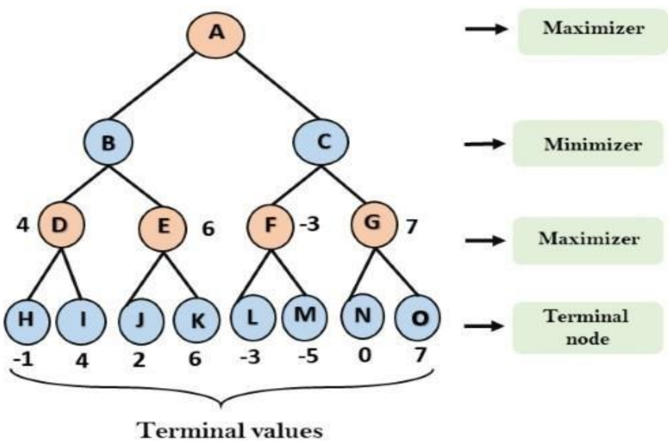
- At the terminal node, the terminal values are given so we will compare those value and backtrack the tree until the initial state occurs. Following are the main steps involved in solving the two-player game tree:

Step-1: In the first step, the algorithm generates the entire game-tree and apply the utility function to get the utility values for the terminal states. In the below tree diagram, let's take A is the initial state of the tree. Suppose maximizer takes first turn which has worst-case initial value =- infinity, and minimizer will take next turn which has worst-case initial value = +infinity.



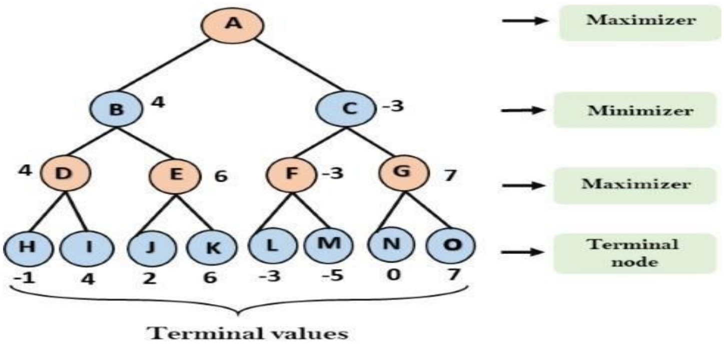
Step 2: Now, first we find the utilities value for the Maximizer, its initial value is -∞, so we will compare each value in terminal state with initial value of Maximizer and determines the higher nodes values. It will find the maximum among the all.

- For node D max(-1,- -∞) => max(-1,4)= 4
- For Node E max(2, -∞) => max(2, 6)= 6
- For Node F max(-3, -∞) => max(-3,-5) = -3
- For node G max(0, -∞) = max(0, 7) = 7



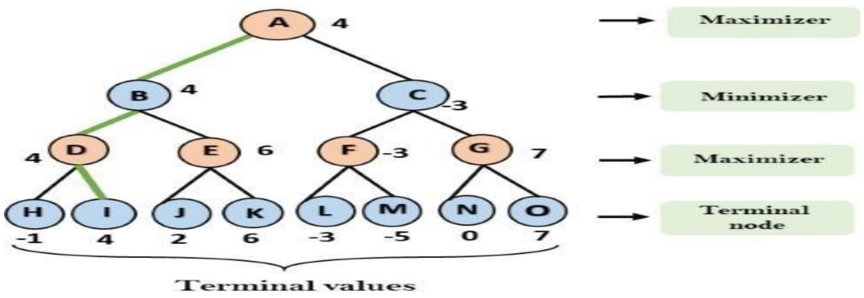
Step 3: In the next step, it's a turn for minimizer, so it will compare all nodes value with +∞, and will find the 3rd layer node values.

- For node B= min(4,6) = 4
- For node C= min (-3, 7) = -3



Step 4: Now it's a turn for Maximizer, and it will again choose the maximum of all nodes value and find the maximum value for the root node. In this game tree, there are only 4 layers, hence we reach immediately to the root node, but in real games, there will be more than 4 layers.

- For node A max(4, -3)= 4



That was the complete workflow of the minimax two player game.

ALGORITHM

We will now discuss the algorithm to write the code. This algorithm will help you to write code in any programming language of your choice. Let‘s see how it‘s done.

Create a board using a 2-dimensional array and initialize each element as empty.

- You can represent empty using any symbol you like. Here, we are going to use ahyphen. '-'.

Write a function to check whether the board is filled or not.

- Iterate over the board and return false if the board contains an empty sign or elsereturn true.

Write a function to check whether a player has won or not.

- We have to check all the possibilities that we discussed in the previous section.
- Check for all the rows, columns, and two diagonals.

Write a function to show the board as we will show the board multiple times to the userswhile they are playing.
Write a function to start the game.

- Select the first turn of the player randomly.
- Write an infinite loop that breaks when the game is over (either win or draw).

- Show the board to the user to select the spot for the next move.
- Ask the user to enter the row and column number.
- Update the spot with the respective player sign.
- Check whether the current player won the game or not.
- If the current player won the game, then print a winning message andbreak the infinite loop.
- Next, check whether the board is filled or not.
- If the board is filled, then print the draw message and break the infiniteloop.

COMPLEXITY OF BFS ALGORITHM

As it performs DFS for the game-tree, so the time complexity of Min-Max algorithm is $O(b^m)$,where b is branching factor of the game-tree, and m is the maximum depth of the tree.

Space complexity of Mini-max algorithm is also similar to DFS which is O(bm).

PROCEDURE OF EXECUTION:

1. Open Anaconda Navigator
2. Launch Notebook Jupyter
3. Open New File Python Kernel
4. Type the Program
5. Click On Run Button

PROGRAM CODE:

```
import random class TicTacToe:

def______init___(self):self.board = []

def create_board(self):for i in range(3):

row = []
```

```
for j in range(3): row.append('-')

self.board.append(row)

def get_random_first_player(self):return

random.randint(0, 1) def fix_spot(self, row, col,

player):self.board[row][col] = player def

is_player_win(self, player):win = None

n = len(self.board)# checking rows for i

in range(n): win = True

for j in range(n):

if self.board[i][j] != player:win

= False breakif win:

return win

# checking columnsfor i

in range(n): win = True
for j in range(n):

if self.board[j][i] != player:win

= False breakif win:

return win

# checking

diagonalswin = True

for i in range(n):

if self.board[i][i] != player:win

= False breakif win:

return

winwin =

True for i

in

range(n):

if self.board[i][n - 1 - i] !=
```

```
player:win = False breakif win:

return winreturn False
for row in self.board:for item in row:

if item == '-':

return False

return True

def is_board_filled(self):for row in self.board:

for item in row:if item == '-':

return Falsereturn True

def swap_player_turn(self, player):return 'X' if player ==

'O' else 'O' def show_board(self): for row in self.board:

for item in row:

print(item, end=" ")

print() def start(self):

self.create_board()

player = 'X' if self.get_random_first_player() == 1 else

'O'while True: print(f"Player {player}

turn")self.show_board()

# taking user inputrow, col = list(

map(int, input("Enter row and column numbers to fix spot:

").split()))print() # fixing the spot

self.fix_spot(row - 1, col - 1, player)

# checking whether current player is won or notif

self.is_player_win(player): print(f"Player {player} wins the game!")break

# checking whether the game is draw or notif

self.is_board_filled(): print("Match Draw!")break

# swapping the turn

player =
self.swap_player_turn(play
er) # showing the final
```

view of boardprint()

self.show_board()# starting the

game tic_tac_toe =

TicTacToe()tic_tac_toe.start()

OUTPUT

Player X turn
- - -
- - -
- - -
Enter row and column numbers to fix spot: 1 1
Player O turnX - -
- - -
- - -
Enter row and column numbers to fix spot: 1 2

Player X turnX O -
- - -
- - -
Enter row and column numbers to fix spot: 2 1

Player
O
turnX
O - X -
-
- - -
Enter row and column numbers to fix spot: 2 2

Player
X
turnX
O - X
O -
- - -
Enter row and column numbers to fix spot: 3 1Player X wins the game!

X O -
X O -X - -

CONCLUSION: Thus We Implement Tic-Tac-Toe Game Using Adversarial Search Algorithm By Using Min Max Technique Successfully

VIVA QUESTION

1) **What is Adversarial Search?**
2) **How would you modify the Minimax Algorithm for a game like Chess?**
3) **What is the significance of heuristic evaluation functions in the Minimax Algorithm?**
4) **What are the limitations of the Minimax Algorithm?**

EXPERIMENT NO: 4

AIM: To Generate A Random Lower Case And Upper Case Letter Password Of Any Length In Python.

THEORY: Probability theory, **a** branch of mathematics concerned with the analysis of random phenomena. The outcome of a random event cannot be determined before it occurs, but it may be any one of several possible outcomes. The actual outcome is considered to be determined by chance.

The issue of generation of random numbers is quite common, but sometimes, we have applications that require us to better that and provide some functionality of generating a random string of digits and alphabets for applications such as passwords. Let‘s discuss certain ways in which this can be performed in Python.

GENERATE A RANDOM STRING USING RANDOM.CHOICES()

This random.choices() function of a random module can help us achieve this task, and providesa one-liner alternative to a whole loop that might be required for this particular task.

String.ascii_uppercase – It returns the string with uppercase.String.digits – It returns the string with digits.
String.punctuation – It returns the string with punctuations String.ascii_letters – It returns the string containing various cases.String.ascii_lowercase – It returns the string with lowercase.

HOW TO CREATE A RANDOM STRING IN PYTHON

We can generate the random string using the random module and string module. Use the below steps to create a random string of any length in Python.

1) Use the string constant ascii_lowercase

The string.ascii_lowercase returns a list of all the lowercase letters from _a‘ to _z‘. This data will be used as a source to generate random characters.

2) Decide the length of a string

Decide how many characters you want in the resultant string.

3) Use a for loop and random choice() function to choose characters from a source

Run a for loop till the decided string length and use the random choice() function in each iteration to pick a single character from the string constant and add it to the string variable using a join() function. print the final string after loop competition

4) Generate a random Password

Use the string.ascii_letters, string.digits, and string.punctuation constants together to create a random password and repeat the first four steps.

PROCEDURE OF EXECUTION:

1. Open Anaconda Navigator
2. Launch Notebook Jupyter
3. Open New File Python Kernel

4. Type the Program

5. Click On Run Button

PROGRAM CODE:

```
import string

def get_random_string(length):

# With combination of lower and upper case

result_str = ''.join(random.choice(string.ascii_letters) for i in range(length))# print random

string print(result_str)# string of length 8

get_random_string(8)get_random_string(8)
```

OUTPUT
XlksUZPScQUjbifq

CONCLUSION: Thus Given Program Run Successfully Which Is Generate The Random Lower Case And Upper Case Password In Python

VIVA QUESTION

1) **What Python modules are commonly used for generating random passwords?**
2) **How can you ensure that a password contains both uppercase and lowercase letters?**
3) **Why is it important to shuffle the characters in the generated password?**
4) **What is the secrets module, and when should it be used over random?**

EXPERIMENT NO: 5

AIM: Write A Program In Python To Implement Markov Chain Model.

THEORY:

A Markov process is a stochastic process that satisfies Markov Property. Markov process is named after the Russian Mathematician Andrey Markov. Markov Chain is a type of Markov process and has many applications in real world. Google's Page Rank algorithm is based on Markov chain. Markov Chain can be applied in speech recognition, statistical mechanics, queueing theory, economics, etc.

In Markov Chain, the next state of the process depends only on the previous state and not on the prior sequence of events. Let us think about a stochastic process {Xn}, n=0,1,2,3,4 .. which has a discrete State Space S and satisfies the Markov Property. This is a Markov chain. Since this stochastic process follows the Markov property, the conditional probability distribution of future states of the process depends only upon the present state. The Markov property can be represented mathematically as below:

Pr {xn+1=j | xn = i, xn–1 = in–1 …., x1 = i, x0 = i0} = Pr {xn+1 = j | xn = i}for any i, j,

i1 , i 2 … i n–1 (elements of S)

A Markov Chain has a set of states and some process that can switch these states to one another based on a transition model. To understand the concept well, let us look at a very simple example — a 2 state Markov Chain. Assume you have 2 shirts — white and blue. Wearing white shirt is represented by W and wearing blue shirt is represented by B.

We can define the State Space S as {W, B}. Let us assume the probability of you

wearing a white shirt and continue wearing the white shirt is 0.7, probability of changing

the white shirt to blue shirt is

0.3. Once you are wearing a blue shirt, the probability of you continue wearing the blue shirt is

0.4 and the probability of changing the blue shirt to white shirt is 0.6. The above data

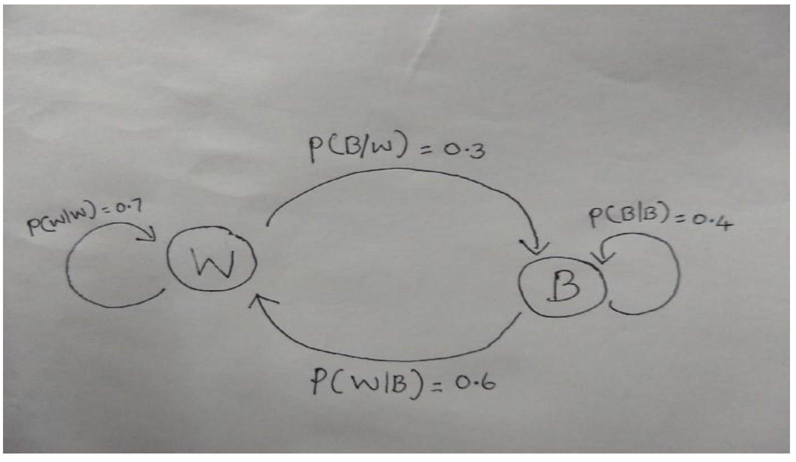
can be represented by the transition matrix T as shown below.

Shirt Color	White(W)	Blue(B)
White(W)	0.7	0.3
Blue(B)	0.6	0.4

We can define these transition probabilities as follows:

P(W|W) = 0.7, P(B|W) = 0.3, P(W|B) = 0.6, P(B|B) = 0.4

This can be diagrammatically represented as:



Suppose you repeat this same process every hour — i.e deciding on which shirt to wear and changing your shirt accordingly. Lets say, at the start you already decided that you will wear a white shirt, so the current state can be defined as

I = [1, 0]
Assuming the transition matrix does not change, we will check the probabilities of you wearing white or blue shirt at the end of 1st, 2nd and 3rd hours.

THE CALCULATIONS FOR FINDING OUT THE PROBABILITIES ARE

(a) At the end of 1 st hour: H1 = I x T

(b) At the end of 2 hours: H2 = H1 x T

(c) At the end of 3 hours: H3 = H2 X T

PROCEDURE OF EXECUTION:

1) Open Anaconda Navigator
2) Launch Notebook Jupyter
3) Open New File Python Kernel
4) Type the Program
5) Click On Run Button

PROGRAM CODE:

```
Markov Chain Exampleimport numpy as np

#Current state I = np.matrix([[1,

0]])#Transition Matrix

T = np.matrix([[.7, 0.3],

[.6, 0.4]])

T1 = I * T

# After 1 hoursprint (T1)

T2 = T1 * T

# After 2 hoursprint (T2)

T3 = T2 * T
# After 3 hoursprint (T3)
```

OUTPUT

[[0.7 0.3]]
[[0.67 0.33]]
[[0.667 0.333]]

CONCLUSION: THUS THE MARKOV CHAIN MODEL IMPLEMENTED IN PYTHON SUCCESSFULLY.

VIVA QUESTION

1) **What is a Markov Chain?**
2) **What is the Markov property?**
3) **Differentiate between deterministic and stochastic processes.**
4) **What is a state space in a Markov Chain?**
5) **What do you mean by transition probability?**
6) **What is a transition probability matrix (TPM)?**

EXPERIMENT NO: 6

AIM:_Write A Python Program To Implement Shortes Path For Directed Acyclic Graph In Bayesian Network.

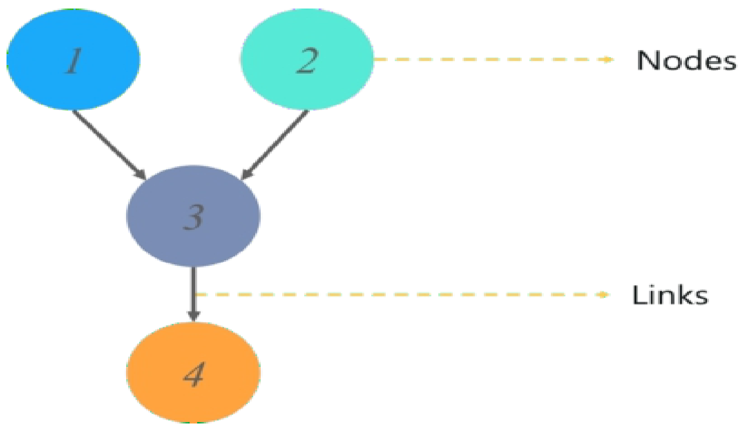
THEORY: Bayesian Networks have given shape to complex problems that provide limited information and resources. It‘s being implemented in the most advancing technologies of the era such as Artificial Intelligence and Machine Learning. Having such a system is a need in today‘s technology-centric world. Keeping this in mind, this article is completely dedicated to the working of Bayesian Networks and how they can be applied to solve convoluted problems.

WHAT IS A BAYESIAN NETWORK?

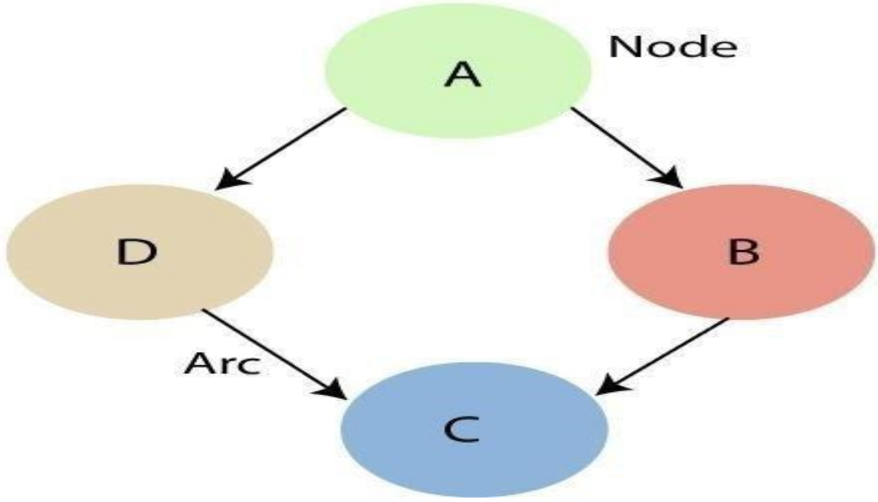
A Bayesian Network falls under the category of Probabilistic Graphical Modelling (PGM) technique that is used to compute uncertainties by using the concept of **probability**. Popularly known as Belief Networks, Bayesian Networks are used to model uncertainties by using **Directed Acyclic Graphs** (DAG).

WHAT IS A DIRECTED ACYCLIC GRAPH?

A Directed Acyclic Graph is used to represent a Bayesian Network and like any other statistical graph, a DAG contains a set of nodes and links, where the links denote the relationship between the nodes.



By definition, Bayesian Networks are a type of Probabilistic Graphical Model that uses the Bayesian inferences for probability computations. It represents a set of variables and its conditional probabilities with a Directed Acyclic Graph (DAG). They are primarily suited for considering an event that has occurred and predicting the likelihood that any one of the several possible known causes is the contributing factor.



As mentioned above, by making use of the relationships which are specified by the Bayesian Network, we can obtain the Joint Probability Distribution (JPF) with the conditional probabilities. Each node in the graph represents a random variable and the arc (or directed arrow) represents the relationship between the nodes. They can be either continuous or discrete in nature.

n the above diagram A, B, C and D are 4 random variables represented by nodes given in the network of the graph. To node B, A is its parent node and C is its child node. Node C is independent of Node A.

LOCAL MARKOV PROPERTY

The Bayesian Networks satisfy the property known as the Local Markov Property. It states that a node is conditionally independent of its non-descendants, given its parents. In the above example, P(D|A, B) is equal to P(D|A) because D is independent of its non-descendent, B. This property aids us in simplifying the Joint Distribution. The Local Markov Property leads us to the concept of a Markov Random Field which is a random field around a variable that is said to follow Markov properties.

CONDITIONAL PROBABILITY

In mathematics, the Conditional Probability of event A is the probability that event A will occur given that another event B has already occurred. In simple terms, p(A | B) is the probability of event A occurring, given that event, B occurs. However, there are two types of event possibilities between A and B. They may be either dependent events or independent events. Depending upon their type, there are two different ways to calculate the conditional probability.

Given A and B are dependent events, the conditional probability is calculated as P (A| B) = P(A and B) / P (B)
If A and B are independent events, then the expression for conditional probability is givenby, P(A| B) = P (A)

JOINT PROBABILITY DISTRIBUTION

Before we get into an example of Bayesian Networks, let us understand the concept of Joint Probability Distribution. Consider 3 variables a1, a2 and a3. By definition, the probabilities of all different possible combinations of a1, a2, and a3 are called its Joint Probability Distribution.

If P[a1,a2, a3,….., an] is the JPD of the following variables from a1 to an, then there are several ways of calculating the Joint Probability Distribution as a combination of various terms such as,

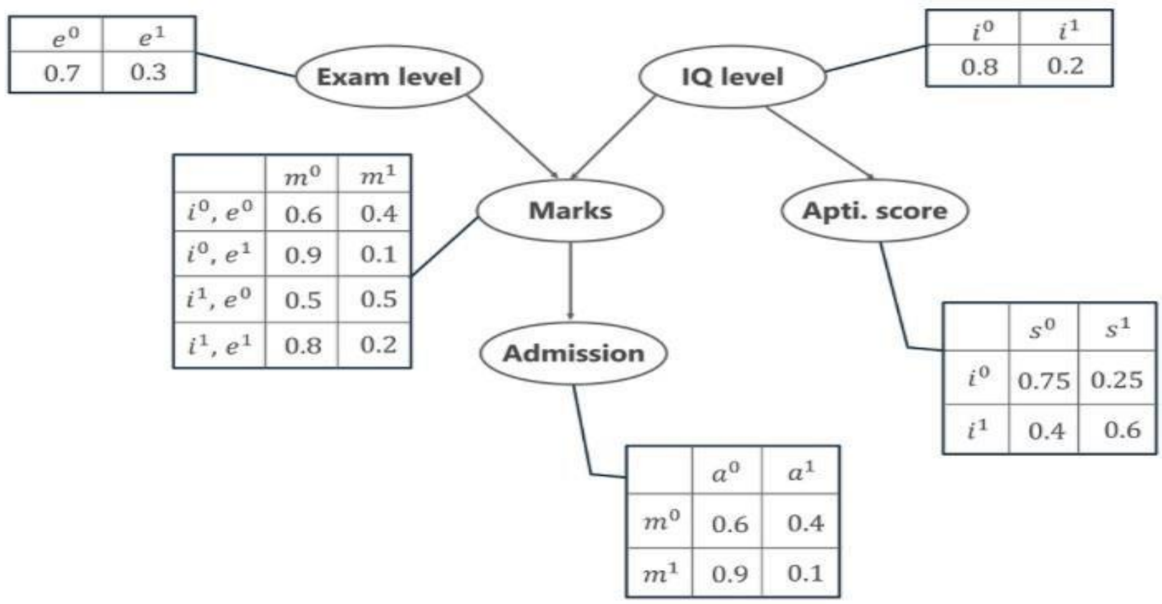
P[a1,a2, a3,….., an] = P[a1 | a2, a3,….., an] * P[a2, a3,….., an]

= P[a1 | a2, a3,….., an] * P[a2 | a3,….., an]….P[an-1|an] * P[an]

EXAMPLE OF BAYESIAN NETWORKS

Let us now understand the mechanism of Bayesian Networks and their advantages with the help of a simple example. In this example, let us imagine that we are given the task of modeling a student‘s marks (***m***) for an exam he has just given. From the given Bayesian Network Graph below, we see that the marks depend upon two other variables. They are,

Exam Level (***e***)– This discrete variable denotes the difficulty of the exam and has two values (0 for easy and 1 for difficult)
IQ Level (***i***) – This represents the Intelligence Quotient level of the student and is also discrete in nature having two values (0 for low and 1 for high)
Additionally, the IQ level of the student also leads us to another variable, which is the Aptitude Score of the student (***s***). Now, with marks the student has scored, he can secure admission to a particular university. The probability distribution for getting admitted (***a***) to a university is also given below.



In the above graph, we see several tables representing the probability distribution values of the given 5 variables. These tables are called the Conditional Probabilities Table or CPT. There are a few properties of the CPT given below –

The sum of the CPT values in each row must be equal to 1 because all the possible cases for a particular variable are exhaustive (representing all possibilities).
If a variable that is Boolean in nature has k Boolean parents, then in the CPT it has 2K probability values.
Coming back to our problem, let us first list all the possible events that are occurring in the above- given table.

1. Exam Level (e)
2. IQ Level (i)
3. Aptitude Score (s)
4. Marks (m)
5. Admission (a)

These five variables are represented in the form of a Directed Acyclic Graph (DAG) in a Bayesian Network format with their Conditional Probability tables. Now, to calculate the Joint Probability Distribution of the 5 variables the formula is given by,

P[a, m, i, e, s]= P(a | m) . P(m | i, e) . P(i) . P(e) . P(s | i)

From the above formula

P(a | m) denotes the conditional probability of the student getting admission based on themarks he has scored in the examination.
P(m | i, e) represents the marks that the student will score given his IQ level and difficultyof the Exam Level.
P(i) and P(e) represent the probability of the IQ Level and the Exam Level.
P(s | i) is the conditional probability of the student's Aptitude Score, given his IQ Level.
With the following probabilities calculated, we can find the Joint Probability Distribution of the entire Bayesian Network.

CALCULATION OF JOINT PROBABILITY DISTRIBUTION

Let us now calculate the JPD for two cases.

Case 1: Calculate the probability that in spite of the exam level being difficult, the student having a low IQ level and a low Aptitude Score, manages to pass the exam and secure admission to the university.

From the above word problem statement, the Joint Probability Distribution can be

written as below, P[a=1, m=1, i=0, e=1, s=0]

From the above Conditional Probability tables, the values for the given conditions are fed to the formula and is calculated as below.

P[a=1, m=1, i=0, e=0, s=0] = P(a=1 | m=1) . P(m=1 | i=0, e=1) . P(i=0) . P(e=1) . P(s=0 | i=0)

= 0.1 * 0.1 * 0.8 * 0.3 * 0.75

= **0.0018**

Case 2: In another case, calculate the probability that the student has a High IQ level and Aptitude Score, the exam being easy yet fails to pass and does not secure admission to the university.

Also Read: Machine Learning Project Ideas & TopicsThe formula for the JPD is

given by P[a=0, m=0, i=1, e=0, s=1]

Thus,

P[a=0, m=0, i=1, e=0, s=1]= P(a=0 | m=0) . P(m=0 | i=1, e=0) . P(i=1) . P(e=0) . P(s=1 | i=1)

= 0.6 * 0.5 * 0.2 * 0.7 * 0.6

= **0.0252**

Hence, in this way, we can make use of Bayesian Networks and Probability tables to calculate the probability for various possible events that occur.

PROCEDURE OF EXECUTION:

1. Open Anaconda Navigator
2. Launch Notebook Jupyter
3. Open New File Python Kernel
4. Type the Program
5. Click On Run Button

PROGRAM CODE:

```
# Python program to find single source shortest paths for Directed Acyclic GraphsComplexity :O(V+E)

#from collections import defaultdict

# Graph is represented using adjacency list. Every node of adjacency list contains

vertex number of# the vertex to which edge connects. It also contains

# weight of the edgeclass Graph:

def__________init___(self,vertices):

self.V = vertices # No. of vertices

# dictionary containing adjacency Listself.graph =
```

defaultdict(list) # function to add an edge to graphdef

addEdge(self,u,v,w): self.graph[u].append((v,w))

A recursive function used by shortestPathdef

topologicalSortUtil(self,v,visited,stack): # Mark the current node as

visited.visited[v] = True

Recur for all the vertices adjacent to this vertexif v in

self.graph.keys(): for node,weight in self.graph[v]:if

visited[node] == False:

self.topologicalSortUtil(node,visited,stack)# Push current vertex to stack which stores

topological sort

stack.append(v)

''' The function to find shortest paths from given vertex.

It uses recursive topologicalSortUtil() to get topologicalsorting of

given graph.''' def shortestPath(self, s):

Mark all the vertices as not visitedvisited =

[False]*self.V stack =[]

Call the recursive helper function to store Topological# Sort starting from source vertices for i in range(self.V):

if visited[i] == False:

self.topologicalSortUtil(s,visited,stack)# Initialize distances to all vertices as

infinite and # distance to source as 0 dist = [float("Inf")] * (self.V)dist[s] =

0

Process vertices in topological orderwhile stack:

Get the next vertex from topological orderi = stack.pop()

Update distances of all adjacent verticesfor node,weight in

self.graph[i]: if dist[node] > dist[i] + weight: dist[node] =

dist[i] + weight

Print the calculated shortest distancesfor i in range(self.V):

print (("%d" %dist[i]) if dist[i] != float("Inf") else

"Inf" ,end=" ") g = Graph(6)
g.addEdge(0, 1, 5)

g.addEdge(0, 2, 3)

g.addEdge(1, 3, 6)

g.addEdge(1, 2, 2)

g.addEdge(2, 4, 4)

g.addEdge(2, 5, 2)

g.addEdge(2, 3, 7)

g.addEdge(3, 4, -1)

g.addEdge(4, 5, -2)# source = 1

s = 1

print ("Following are shortest distances from source %d " % s)g.shortestPath(s)

OUTPUT

Following are shortest distances from source 1Inf 0 2 6 5 3

CONCLUSION: **Thus The Program To Implement Shortes Path For Directed Acyclic Graph In Bayesian Network Is Executed Successfully**

VIVA QUESTION:

a. What is a Directed Acyclic Graph (DAG)?
b. How does a DAG relate to a Bayesian Network?
c. What is the shortest path in a DAG?
d. Why are cycles not allowed in Bayesian Networks?

EXPERIMENT NO: 7

AIM: Implementation Of Linear Regression Technique Using descision Tree In Python.

THEORY:

SIMPLE LINEAR REGRESSION

Simple linear regression is an approach for predicting a response using a single feature. It is assumed that the two variables are linearly related. Hence, we try to find a linear function that predicts the response value(y) as accurately as possible as a function of the feature or independent

variable(x).

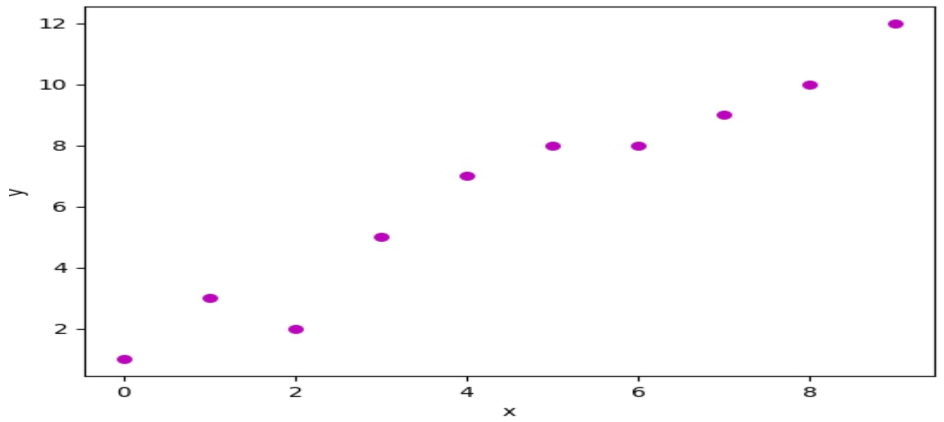
Let us consider a dataset where we have a value of response y for every feature x:

x	0	1	2	3	4	5	6	7	8	9
y	1	3	2	5	7	8	8	9	10	12

For generality, we define:

x as **feature vector**, i.e x = [x_1, x_2, …., x_n], y as **response vector**, i.e y = [y_1, y_2, …., y_n] for **n** observations (in above example, n=10).

A scatter plot of the above dataset looks like:-



Now, the task is to find a line that fits best in the above scatter plot so that we can predict the response for any new feature values. (i.e a value of x not present in a dataset)

This line is called a regression line.

The equation of regression line is represented as:

$$h(x_i) = \beta_0 + \beta_1 x_i$$

Here,

- h(x_i) represents the predicted response value for ith observation.
- b_0 and b_1 are regression coefficients and represent y-intercept and slope of regression line respectively.

PROCEDURE OF EXECUTION:

1. Open Anaconda Navigator
2. Launch Notebook Jupyter
3. Open New File Python Kernel
4. Type the Program
5. Click On Run Button

PROGRAM CODE:

```
import numpy as np

import matplotlib.pyplot as pltdef estimate_coef(x, y):

# number of observations/pointsn

= np.size(x) # mean of x and y

vectorm_x = np.mean(x) m_y =

np.mean(y)

# calculating cross-deviation and deviation about xSS_xy = np.sum(y*x) -

n*m_y*m_x SS_xx = np.sum(x*x) - n*m_x*m_x# calculating regression

coefficientsb_1 = SS_xy / SS_xx b_0 = m_y - b_1*m_xreturn (b_0, b_1)

def plot_regression_line(x, y, b):

# plotting the actual points as scatter
plot plt.scatter(x, y, color = "m",marker
= "o", s = 30)

# predicted response vectory_pred = b[0] + b[1]*x

# plotting the regression line plt.plot(x, y_pred, color = "g")# putting

labels plt.xlabel('x')

plt.ylabel('y')

# function to show plotplt.show()
def main():

# observations / data
```

x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])

estimating coefficientsb =

estimate_coef(x, y) print("Estimated

coefficients:\nb_0 = {} \

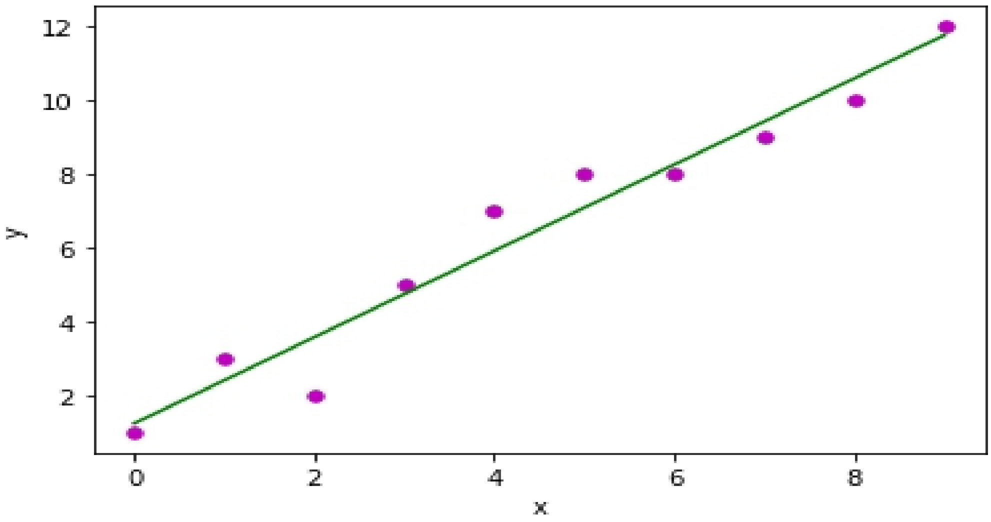
\nb_1 = {}".format(b[0], b[1]))# plotting regression line

plot_regression_line(x, y, b) if__name_== "main

_":main()

OUTPUT

Estimated coefficients:
b_0 = 1.2363636363636363
b_1 = 1.1696969696969697



CONCLUSION: Thus The Implementation Of Linear Regression technique Using Descision Tree In Python Run Successfully.

VIVA QUESTION:
1) **What is linear regression?**
2) **What is a decision tree?**
3) **Can a decision tree be used for regression? How?**
4) **What is the difference between decision tree regression and linear regression?**
5) **Why would you use a decision tree instead of linear regression?**
6) **What are the advantages of using a decision tree for regression tasks?**

EXPERIMENT NO: 8

AIM: Implementation Of Any Game By Using Reinforcement Learning Technique In Python

THEORY: Reinforcement Learning is a feedback-based Machine learning technique in which an agent learns to behave in an environment by performing the actions and seeing the results of actions. For each good action, the agent gets positive feedback, and for each bad action, the agent gets negative feedback or penalty.

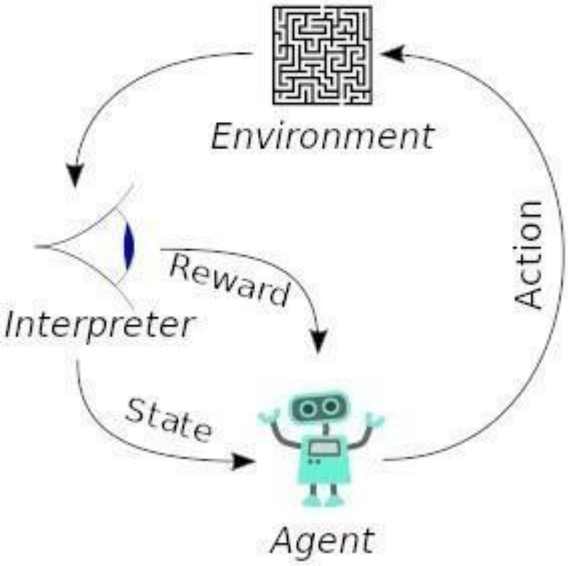
- In Reinforcement Learning, the agent learns automatically using feedbacks without any labeled data, unlike supervised learning.
- Since there is no labeled data, so the agent is bound to learn by its experience only.
- RL solves a specific type of problem where decision making is sequential, and the goal is long- term, such as game-playing, robotics, etc.
- The agent interacts with the environment and explores it by itself. The primary goal of an agent in reinforcement learning is to improve the performance by getting the maximum positive rewards.
- The agent learns with the process of hit and trial, and based on the experience, it learns to perform the task in a better way. Hence, we can say that "Reinforcement learning is a type of machine learning method where an intelligent agent (computer program) interacts with the environment and learns to act within that." How a Robotic dog learns the movement of his arms is an example of Reinforcement learning.
- It is a core part of Artificial intelligence, and all AI agent works on the concept of reinforcement learning. Here we do not need to pre-program the agent, as it learns from its own experience without any human intervention.

TERMS USED IN REINFORCEMENT LEARNING

- Agent(): An entity that can perceive/explore the environment and act upon it.
- Environment(): A situation in which an agent is present or surrounded by. In RL, weassume the stochastic environment, which means it is random in nature.
- State(): State is a situation returned by the environment after each action taken by the agent.
- Reward(): A feedback returned to the agent from the environment to evaluate the action ofthe agent.
- Policy(): Policy is a strategy applied by the agent for the next action based on the current state.
- Value(): It is expected long-term retuned with the discount factor and opposite

to the short-term reward.

- Q-value(): It is mostly similar to the value, but it takes one additional parameter as acurrent action (a).



PROCEDURE OF EXECUTION:

```
from math import inf as infinityfrom random import choice import platform
import time

from os import

systemHUMAN = -1

COMP = +1

board = [ [0, 0, 0],

[0, 0, 0],

[0, 0, 0],

    ]

def evaluate(state):

if wins(state,

COMP):score = +1

elif wins(state,
```

HUMAN):score = -1

else:

score = 0 return score

def wins(state, player):win_state

= [[state[0][0], state[0][1],

state[0][2]],

[state[1][0], state[1][1], state[1][2]],

[state[2][0], state[2][1], state[2][2]],

[state[0][0], state[1][0], state[2][0]],

[state[0][1], state[1][1], state[2][1]],

[state[0][2], state[1][2], state[2][2]],

[state[0][0], state[1][1], state[2][2]],

[state[2][0], state[1][1], state[0][2]],
]

if [player, player, player] in

win_state:return True else:

return False

def game_over(state):

return wins(state, HUMAN) or wins(state, COMP)def empty_cells(state):
cells = []

for x, row in enumerate(state): for y, cell in

enumerate(row): if cell == 0: cells.append([x,

y])

return cells

def valid_move(x, y):

if [x, y] in

```
empty_cells(board):return True

else:

return False

def set_move(x, y, player):if valid_move(x, y):

board[x][y] =

playerreturn True

else:
return False

def mini(state, depth, player):if player == COMP:
best = [-1, -1, -infinity]else:
best = [-1, -1, +infinity]
if depth == 0 or game_over(state):score =
evaluate(state) return [-1, -1, score]
for cell in empty_cells(state):x, y = cell[0], cell[1]
state[x][y] = player score = mini(state, depth - 1, -
player)state[x][y] = 0
score[0], score[1] = x, yif player == COMP:
if score[2] > best[2]:
best = score # max valueelse:
if score[2] < best[2]:
best = score # min
valuereturn best def
clean():
os_name = platform.system().lower()if 'windows' in
os_name: system('cls')else:
system('clear')
def render(state, c_choice, h_choice):chars = {
-1: h_choice,
+1: c_choice,
0: ' '
        }
str_line = '
.......................................................'pr
int('\n' + str_line) for row in
state: for cell in row:
symbol = chars[cell] print(f'| {symbol}
|', end='') print('\n' + str_line)
def ai_turn(c_choice, h_choice): depth =
len(empty_cells(board)) if depth == 0 or
game_over(board):return
clean()
print(f'Computer turn [{c_choice}]')render(board,  c_choice, h_choice) if
depth == 9: x = choice([0, 1, 2])
```

```
y = choice([0, 1, 2])else:
move = mini(board, depth, COMP)x, y = move[0],
move[1] set_move(x, y, COMP)

time.sleep(1)
def human_turn(c_choice, h_choice):"""
The Human plays choosing a valid move.
:param c_choice: computer's choice X or O
:param h_choice: human's choice X or O
:return:
        """
depth = len(empty_cells(board))
if depth == 0 or
game_over(board):return #
Dictionary of valid
movesmove = -1 moves = {
1: [0, 0], 2: [0, 1], 3: [0, 2],
4: [1, 0], 5: [1, 1], 6: [1, 2],
7: [2, 0], 8: [2, 1], 9: [2, 2],
        }

clean()

print(f'Human turn [{h_choice}]')

render(board, c_choice, h_choice)while move < 1
or move > 9: try:
move = int(input('Use numpad (1..9): '))coord = moves[move]

can_move = set_move(coord[0], coord[1], HUMAN)if
not can_move: print('Bad move')move = -1
except (EOFError,
KeyboardInterrupt):print('Bye') exit()
except (KeyError,
ValueError):print('Bad choice') def
main():clean()
h_choice = '' # X or Oc_choice
= ''  # X or O first = ''  # if
human is the first
# Human chooses X or O to play
while h_choice != 'O' and h_choice != 'X':try:
print('')
h_choice = input('Choose X or O\nChosen: ').upper()# Setting
computer's choice if h_choice == 'X':c_choice = 'O'
else:
c_choice = 'X'

except (EOFError,
KeyboardInterrupt):print('Bye') exit()
except (KeyError,
```

```
ValueError):print('Bad choice') #
Human may starts firstclean()
while first != 'Y' and first != 'N':
try:
first = input('First to start?[y/n]: ').upper()except (EOFError,
KeyboardInterrupt): print('Bye')exit()
except (KeyError, ValueError):print('Bad choice')
# Main loop of this game
while len(empty_cells(board)) > 0 and not game_over(board):if first == 'N':

ai_turn(c_choice, h_choice)first
= '' human_turn(c_choice,
h_choice)

ai_turn(c_choice, h_choice)else:
print(len(empty_cells(board))+game_over(board))# Game
over message

if wins(board, HUMAN):clean()

print(f'Human turn [{h_choice}]')render(board, c_choice,

h_choice)print('YOU WIN!') elif wins(board, COMP):clean()

print(f'Computer turn [{c_choice}]')render(board, c_choice, h_choice) print('YOU

LOSE!') else:

clean()

render(board, c_choice,
h_choice) print('DRAW!')
if____________name__== '____main

__':main()
```

OUTPUT :

```
|
X
|
|
|
|
|

|
```

```
|
|
|
|
|
...............................
| ||  || | Human turn [X]

...............................
| X ||  ||  |
...............................
|  || O ||  |
...............................
        |  ||  || |
...............................
Use numpad (1..9): 2Computer turn [O]

...............................
| X || X ||  |
...............................
|  || O || |
...............................
| ||  || | Human turn [X]

...............................
| X || X || O |
...............................
|  || O || |
...............................
        |  ||  ||  |
...............................
Use numpad (1..9): 3Bad move
Use numpad (1..9): 4Computer turn [O]
...............................
| X || X || O |
...............................
| X || O ||  |
...............................
        |  ||  || |
...............................
4
Computer turn [O]

...............................
| X || X || O |
...............................
| X || O ||  |
...............................
| O || || | YOU LOSE!
```

CONCLUSION: Thus We Implementation Tic-Tac-Toe Game By Using Reinforcement Learning Technique In Python In Which Human And Computer Are Invilved

VIVA QUESTION:

1) **What is Reinforcement Learning?**
2) **What are the main components of an RL problem?**
3) **What is the difference between supervised learning and reinforcement learning?**
4) **Define the terms: policy, reward function, value function, and Q-function.**
5) **What is the goal of an RL agent?**
6) **What is an episode in RL?**
7) **What is the exploration vs exploitation dilemma?**
8) **What is a Markov Decision Process (MDP)?**

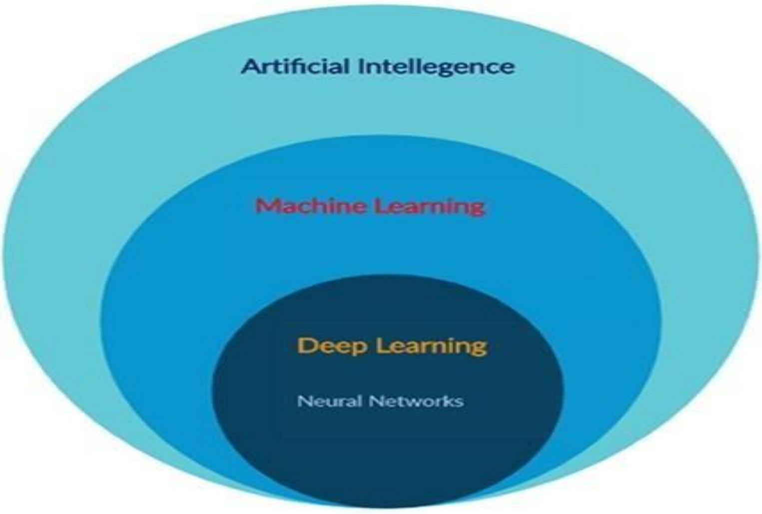
EXPERIMENT NO: 9

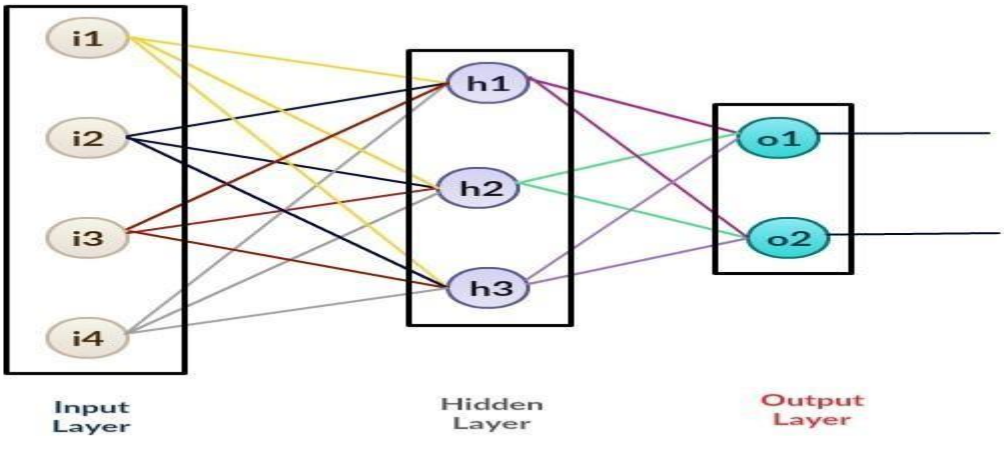
AIM: Create A Neural Network From Scratch In Python

THEORY: Neural networks form the base of deep learning, which is a subfield of machine learning, where the structure of the human brain inspires the algorithms. Neural networks take input data, train themselves to recognize patterns found in the data, and then predict the output for a new set of similar data. Therefore, a neural network can be thought of as the functional unit of deep learning, which mimics the behavior of the human brain to solve complex data-driven problems.

The first thing that comes to our mind when we think of ―neural networks‖ is biology, and indeed, neural nets are inspired by our brains.

Simply put, an ANN represents interconnected input and output units in which each connection has an associated weight. During the learning phase, the network learns by adjusting these weights in order to be able to predict the correct class for input data.





THERE ARE TWO WAYS TO CREATE A NEURAL NETWORK IN PYTHON:

From Scratch – this can be a good learning exercise, as it will teach you how neuralnetworks work from the ground up

Using a Neural Network Library – packages like Keras and TensorFlow simplify the building of neural networks by abstracting away the low-level code. If you‘re already familiar withhow neural networks work, this is the fastest and easiest way to create one.

No matter which method you choose, working with a neural network to make a prediction is essentially the same:

1. **Import the libraries**. For example: **import numpy as np**

2. **Define/create input data**. For example, use **numpy** to create a dataset and an array ofdata values.

3. **Add weights and bias** (if applicable) to input features. These are learnable parameters,meaning that they can be adjusted during training.

Weights = input parameters that influences outputBias = an extra threshold value added to the output

4. **Train the network** against known, good data in order to find the correct values for theweights and biases.

5. **Test the Network** against a set of test data to see how it performs.

6. **Fit the model** with hyperparameters (parameters whose values are used to control thelearning process), calculate accuracy, and make a prediction.

PROCEDURE OF EXECUTION:

1. Open Anaconda Navigator
2. Launch Notebook Jupyter
3. Open New File Python Kernel
4. Type the Program
5. Click On Run Button

PROGRAM CODE:

Import python libraries required in this example:import numpy as np

from scipy.special import expit as activation_functionfrom scipy.stats

import truncnorm # DEFINE THE NETWORK

Generate random numbers within a truncated (bounded)# normal

distribution: def truncated_normal(mean=0, sd=1, low=0,

upp=10):return truncnorm(

(low - mean) / sd, (upp - mean) / sd, loc=mean, scale=sd)# Create the _Nnetwork‘

class and define its arguments:

```
# Set the number of neurons/nodes for each layer# and initialize the
weight matrices: class Nnetwork:
def______init___(self,
no_of_in_nodes, no_of_out_nodes, no_of_hidden_nodes,learning_rate):
self.no_of_in_nodes = no_of_in_nodes self.no_of_out_nodes =
no_of_out_nodes self.no_of_hidden_nodes =
no_of_hidden_nodesself.learning_rate = learning_rate
self.create_weight_matrices()
def create_weight_matrices(self):
""" A method to initialize the weight matrices of the neural network"""rad = 1 /
np.sqrt(self.no_of_in_nodes)
X = truncated_normal(mean=0, sd=1, low=-rad, upp=rad) self.weights_in_hidden =
X.rvs((self.no_of_hidden_nodes,
self.no_of_in_nodes))
rad = 1 / np.sqrt(self.no_of_hidden_nodes)
X = truncated_normal(mean=0, sd=1, low=-rad, upp=rad)self.weights_hidden_out =
X.rvs((self.no_of_out_nodes,
self.no_of_hidden_nodes))def train(self, input_vector, target_vector):
pass # More work is needed to train the networkdef run(self,
input_vector): """
running the network with an input vector 'input_vector'.'input_vector' can be tuple, list
or ndarray """
# Turn the input vector into a column vector:
input_vector = np.array(input_vector, ndmin=2).T
# activation_function() implements the expit function,# which is an implementation of the
sigmoid function:
input_hidden = activation_function(self.weights_in_hidden @ input_vector) output_vector =
```

activation_function(self.weights_hidden_out @ input_hidden)return output_vector

RUN THE NETWORK AND GET A RESULT

Initialize an instance of the class: simple_network = Nnetwork(no_of_in_nodes=2,

no_of_out_nodes=2, no_of_hidden_nodes=4,

learning_rate=0.6)

Run simple_network for arrays, lists and tuples with shape (2):# and get a result:

simple_network.run([(3, 4)])

OUTPUT

array([[0.46131665],[0.53631167]])

CONCLUSION: Thus We Create A Neural Network From Scratch In Python Successfully.

EXPERIMENT NO: 10

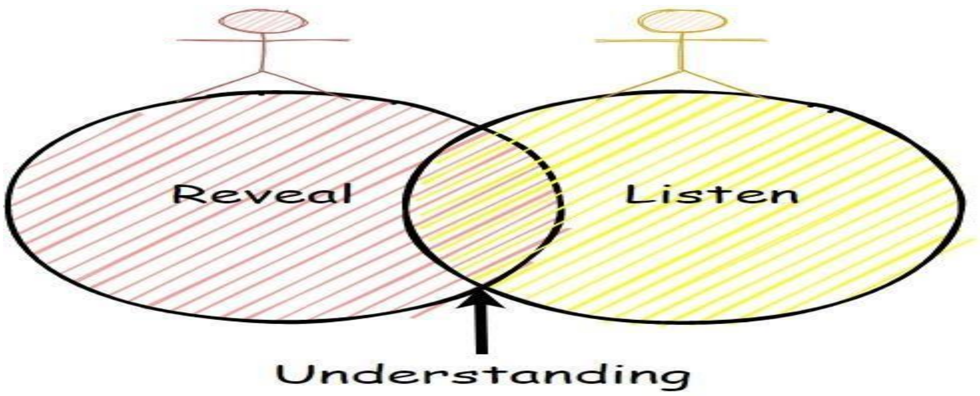
AIM: Implementing The Bag Of Words Natural Language Processing Algorithm In Python.

THEORY: Computers and machines are great at working with tabular data or spreadsheets. However, as human beings generally communicate in words and sentences, not in the form of tables. Much information that humans speak or write is unstructured. So it is not very clear for computers to interpret such. In natural language processing (NLP), the goal is to make computers understand the unstructured text and retrieve meaningful pieces of information from it. Natural language Processing (NLP) is a subfield of artificial intelligence, in which its depth involves the interactions between computers and humans.

APPLICATIONS OF NLP:

- Machine Translation.
- Speech Recognition.
- Sentiment Analysis.
- Question Answering.
- Summarization of Text.
- Chatbot.
- Intelligent Systems.
- Text Classifications.
- Character Recognition.
- Spell Checking.
- Spam Detection.
- Autocomplete.
- Named Entity Recognition.
- Predictive Typing.

Understanding Natural Language Processing (NLP):



We, as humans, perform natural language processing (NLP) considerably well, but even then, we are not perfect. We often misunderstand one thing for another, and we often interpret the same sentences or words differently.

For instance, consider the following sentence, we will try to understand its interpretation in many different ways:

EXAMPLE:

```
I saw a man on a hill with a telescope.
```

NLP example sentence with the text: ―I saw a man on a hill with a telescope.‖These are some interpretations of the sentence shown above.

1. There is a man on the hill, and I watched him with my telescope.
2. There is a man on the hill, and he has a telescope.
3. I‗m on a hill, and I saw a man using my telescope.
4. I‗m on a hill, and I saw a man who has a telescope.
5. There is a man on a hill, and I saw him something with my telescope.

WHAT IS A BAG OF WORDS IN NLP?

Bag of words is a Natural Language Processing technique of text modeling. In technical terms, we can say that it is a method of feature extraction with text data. This approach is a simple and flexible way of extracting features from documents.

A bag of words is a representation of text that describes the occurrence of words within a document. We just keep track of word counts and disregard the grammatical details and the word order. It is called a ―bag‖ of words because any information about the order or structure of words in the document is discarded. The model is only concerned with whether known words occur in the document, not where in the document.

UNDERSTANDING BAG OF WORDS WITH AN EXAMPLE

Let us see an example of how the bag of words technique converts text into vectorsExample(1) without preprocessing:

Sentence 1: ‖Welcome to Great Learning, Now start learning‖Sentence 2: ―Learning is a good practice‖

Sentence 1	Sentence 2
Welcome	Learning
to	is
Great	a
Learning	good
,	practice

Step 1: Go through all the words in the above text and make a list of all of the words in our model vocabulary.

1. Welcome
2. To
3. Great
4. Learning
5. ,

6. Now
7. start
8. learning
9. is
10. a
11. good
12. practice

Note that the words ‗Learning' and ‗ learning' are not the same here because of the difference in their cases and hence are repeated. Also, note that a comma ‗ , ' is also taken in the list.

Because we know the vocabulary has 12 words, we can use a fixed-length document-representation of 12, with one position in the vector to score each word.

The scoring method we use here is to count the presence of each word and mark 0 for absence.This scoring method is used more generally.

The scoring of sentence 1 would look as follows:

Word	Frequency
Welcome	1
to	1
Great	1
Learning	1
,	1
Now	1
start	1
learning	1
is	0
a	0
good	0
practice	0

Writing the above frequencies in the vectorSentence 1 → [

1,1,1,1,1,1,1,1,0,0,0] Now for sentence 2, the scoring would like

Word	Frequency
Welcome	0
to	0
Great	0
Learning	1
,	0
Now	0
start	0
learning	0
is	1
a	1
good	1
practice	1

Similarly, writing the above frequencies in the vector formSentence 2 ➝ [0,0,0,0,0,0,0,1,1,1,1,1]

Sentence	Welcome	to	Great	Learning	,	Now	start	learning	is	a	good	practice
Sentence	1	1	1	1	1	1	1	1	0	0	0	0
Sentence	0	0	0	0	0	0	0	1	1	1	1	1

But is this the best way to perform a bag of words. The above example was not the best example of how to use a bag of words. The words Learning and learning, although having the same meaning are taken twice. Also, a comma ‘,‘ which does not convey any information is also included in the vocabulary.

PROCEDURE OF EXECUTION:

1. Open Anaconda Navigator
2. Launch Notebook Jupyter
3. Open New File Python Kernel
4. Type the Program

Click On Run Button

PROGRAM CODE:

```
def vectorize(tokens):

''' This function takes list of words in a sentence as input and returns a vector of size of
filtered_vocab.It puts 0 if the  word is not present in tokens and count of token if
present.'''

vector=[ ]

for w in filtered_vocab:

vector.append(tokens.count(w)) return vector

def unique(sequence):

'''This functions returns a list in which the order remains same and no item
repeats.Using theset() function does not  preserve the original ordering,so i didnt use
that instead'''

seen = set()

return [x for x in sequence if not (x in seen or seen.add(x))] #create a list of

stopwords.You can import stopwords from nltk toostopwords=["to","is","a"]

#list of special characters.You can use regular expressions toospecial_char=[",",":","

",";",".","?"] #Write the sentences in the corpus,in our case, just two string1="Welcome

to Great Learning , Now start learning"string2="Learning is a good practice"

#convert them to lower casestring1=string1.lower() string2=string2.lower()

#split the sentences into tokenstokens1=string1.split() tokens2=string2.split()

print(tokens1) print(tokens2)

#create a vocabulary list

vocab=unique(tokens1+tokens2)print(vocab) #filter the

vocabulary listfiltered_vocab=[]

for w in vocab:

if w not in stopwords and w not in special_char:filtered_vocab.append(w)

print(filtered_vocab)

#convert sentences into vectordsvector1=vectorize(tokens1) print(vector1)
```

vector2=vectorize(tokens2) print(vector2)

OUTPUT

['welcome', 'to', 'great', 'learning', ',', 'now', 'start', 'learning']['learning', 'is', 'a', 'good', 'practice']
['welcome', 'to', 'great', 'learning', ',', 'now', 'start', 'is', 'a', 'good', 'practice']['welcome',
'great', 'learning', 'now', 'start', 'good', 'practice']
[1, 1, 2, 1, 1, 0, 0]
[0, 0, 1, 0, 0, 1, 1]

CONCLUSION: Thus We Implementing The Bag Of Words Usingnatural Language Processing Algorithm In Python Successfully.

VIVA QUESTION:
1) **What is Natural Language Processing (NLP)?**
2) **What are some real-world applications of NLP?**
3) **What are the main steps in NLP preprocessing?**
4) **What is the difference between stemming and lemmatization?**
5) **What is a corpus in NLP?**
6) **What is a token in NLP?**
7) **What is a stopword? Why are stopwords removed?**