SRM INSTITUTE OF SCIENCE & TECHNOLOGY

DEPARTMENT OF NETWORKING & COMMUNICATIONS

18CSC305J-ARTIFICIAL INTELLIGENCE

SEMESTER – 6 BATCH-2

REGISTRATION NUMBER	**RA1911003010734**
NAME	**Kunal Singhal**

B. Tech- CSE, Third Year (Section: C2)

Year 2021-2022 / Even Semester

INDEX

EXPERIMENT -1

TOWER OF HANOI

Aim: To implement a toy problem (Tower of Hanoi) in Python using Google Colab.

Problem Title: Tower of Hanoi

Problem Statement: Tower of Hanoi is a mathematical puzzle where we have three rods and n disks. The objective of the puzzle is to move all the disks from one pole (say 'source pole') to another pole (say 'destination pole') with the help of the third pole (say auxiliary pole), obeying the following simple rules:

1. Only one disk can be moved at a time.
2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e., a disk can only be moved if it is the uppermost disk on a stack.
3. No disk may be placed on top of a smaller disk.

Algorithm:

1. Calculate the total number of moves required i.e., "pow (2, n) - 1" here n is number of disks.
2. If number of disks (i.e., n) is even then interchange destination pole and auxiliary pole.
3. for i = 1 to total number of moves:

    if i%3 == 1:

        legal movement of top disk between source pole and destination pole

    if i%3 == 2:

        legal movement top disk between source pole and auxiliary pole

    if i%3 == 0:

        legal movement top disk between auxiliary pole and destination pole

Tools: Google Colab

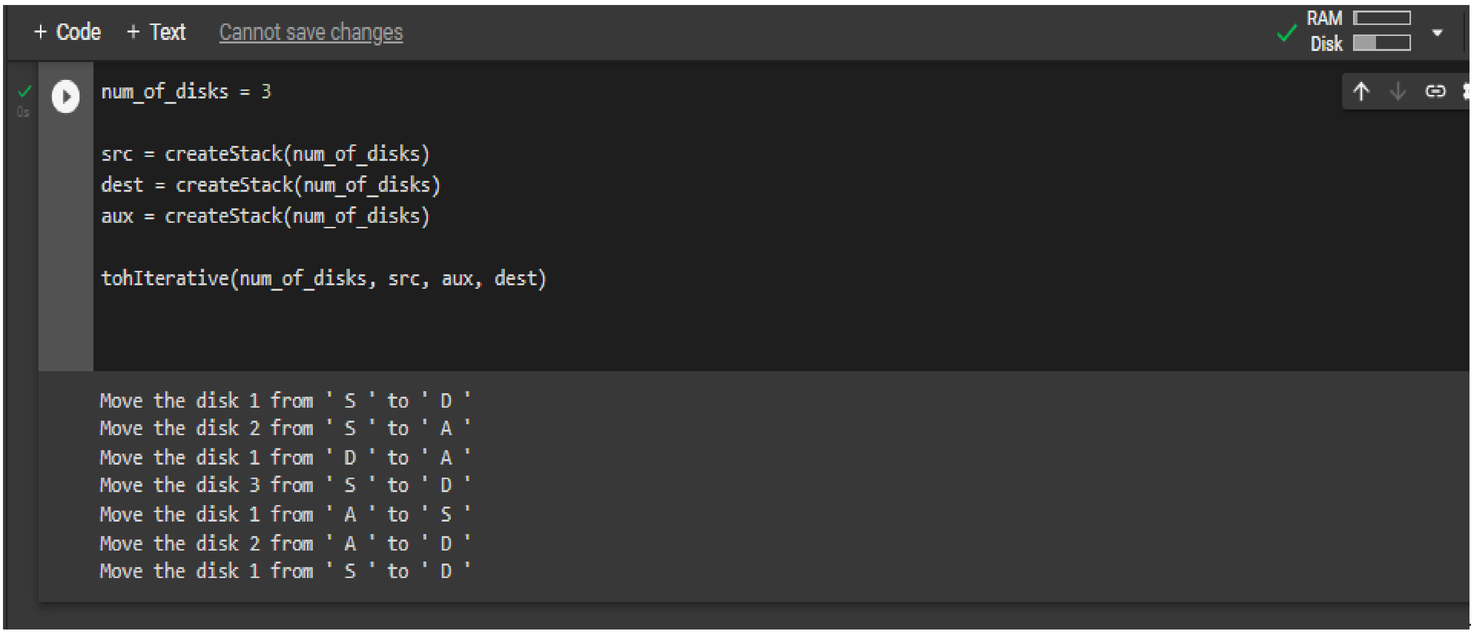
Code:

```
import sys
class Stack:

  def __init__(self, capacity):
```

```python
    self.capacity = capacity
    self.top = -1
    self.array = [0]*capacity

def createStack(capacity):
  stack = Stack(capacity)
  return stack

def isFull(stack):
  return (stack.top == (stack.capacity - 1))

def isEmpty(stack):
  return (stack.top == -1)

def push(stack, item):
  if(isFull(stack)):
    return
  stack.top+=1
  stack.array[stack.top] = item

def Pop(stack):
  if(isEmpty(stack)):
    return -sys.maxsize
  Top = stack.top
  stack.top-=1
  return stack.array[Top]

def moveDisksBetweenTwoPoles(src, dest, s, d):
  pole1TopDisk = Pop(src)
  pole2TopDisk = Pop(dest)

  if (pole1TopDisk == -sys.maxsize):
    push(src, pole2TopDisk)
    moveDisk(d, s, pole2TopDisk)

  elif (pole2TopDisk == -sys.maxsize):
    push(dest, pole1TopDisk)
    moveDisk(s, d, pole1TopDisk)

  elif (pole1TopDisk > pole2TopDisk):
    push(src, pole1TopDisk)
    push(src, pole2TopDisk)
    moveDisk(d, s, pole2TopDisk)

  else:
    push(dest, pole2TopDisk)
    push(dest, pole1TopDisk)
    moveDisk(s, d, pole1TopDisk)

def moveDisk(fromPeg, toPeg, disk):
  print("Move the disk", disk, "from '", fromPeg, "' to '", toPeg, "'")
```

```python
def tohIterative(num_of_disks, src, aux, dest):
  s, d, a = 'S', 'D', 'A'

  if (num_of_disks % 2 == 0):
    temp = d
    d = a
    a = temp
  total_num_of_moves = int(pow(2, num_of_disks) - 1)

  for i in range(num_of_disks, 0, -1):
    push(src, i)

  for i in range(1, total_num_of_moves + 1):
    if (i % 3 == 1):
      moveDisksBetweenTwoPoles(src, dest, s, d)

    elif (i % 3 == 2):
      moveDisksBetweenTwoPoles(src, aux, s, a)

    elif (i % 3 == 0):
      moveDisksBetweenTwoPoles(aux, dest, a, d)

num_of_disks = 3

src = createStack(num_of_disks)
dest = createStack(num_of_disks)
aux = createStack(num_of_disks)

tohIterative(num_of_disks, src, aux, dest)
```

<u>Output:</u>



Result: Successfully implemented the Tower of Hanoi problem and moved the entire stacks from source tower to destination tower using an auxiliary tower.

SRM INSTITUTE OF SCIENCE & TECHNOLOGY

DEPARTMENT OF NETWORKING & COMMUNICATIONS

18CSC305J-ARTIFICIAL INTELLIGENCE

SEMESTER – 6

BATCH-2

REGISTRATION NUMBER	**RA1911003010734**
NAME	**Kunal Singhal**

Year 2021-2022 / Even Semester

Ex No	DATE	Title	Page No	Marks
1	07/02/2022	Developing agent programs for real-world problems (Graph Coloring Problem)		

Exercise: 2

Date: 07-02-2022

GRAPH COLOURING PROBLEM

Problem Statement: graph coloring is a special case of graph labeling; it is an assignment of labels traditionally called "colors" to elements of a graph subject to certain constraints. The problem is, given m colors, find a way of coloring the vertices of a graph such that no two adjacent vertices are colored using the same color. The other graph coloring problem is ***Edge Coloring*** (No vertex is incident to two edges of the same color).

Algorithm :

1.Color first vertex with first color.

2. Do the following for remaining V-1 vertices.

a) Consider the currently picked vertex and color it with the
lowest numbered color that has not been used on any previously colored vertices
adjacent to it.

b) If all previously used colors appear on vertices adjacent to v, assign a new color to it.

Optimization technique: The idea is to assign colors one by one to different vertices, starting from the vertex 0. Before assigning a color, check for safety by considering already assigned colors to the adjacent vertices i.e check if the adjacent vertices have the same color or not. If there is any color assignment that does not violate the conditions, mark the color assignment as part of the solution. If no assignment of color is possible then backtrack and return false.

Algorithm:

1. Create a recursive function that takes the graph, current index, number of vertices, and output color array.
2. If the current index is equal to the number of vertices. Print the color configuration in output array.
3. Assign a color to a vertex (1 to m).
4. For every assigned color, check if the configuration is safe, (i.e. check if the adjacent vertices do not have the same color) recursively call the function with next index and number of vertices
5. If any recursive function returns true, break the loop and return true.
6. If no recursive function returns true then return false.

Tool : Cloud9 ide and Python 3.9.0

Programming code :

```
class Graph:
    def _init_ (self, edges, N):
      self.adj = [[] for _ in range(N)]
      for (src, dest) in edges:
        self.adj[src].append(dest)
        self.adj[dest].append(src)

def colorGraph(graph):
    result = {}
    for u in range(N):
      assigned = set([result.get(i) for i in graph.adj[u] if i in result])
      color = 1
      for c in assigned:
        if color != c:
            break
        color = color + 1
      result[u] = color
    for v in range(N):
        print("color assigned to vertex", v, "is",colors[result[v]])
    print("\n")
    for v in range(N):
        print("color assigned to Edge", v, "is",colors[result[v]+3])

if __name__ == '__main__':
    colors= ["", "YELLOW", "RED", "BLUE", "ORANGE", "GREEN", "PINK", "BLACK",
"BROWN", "WHITE", "PURPLE", "VIOLET"]
    edges = [(0, 1), (1, 2), (2, 3), (3, 4), (4, 0)]
    N = 5
    graph = Graph(edges, N)
    colorGraph(graph)
```

Output screen shots:

```
('color assigned to vertex', 0, 'is', 'YELLOW')
('color assigned to vertex', 1, 'is', 'RED')
('color assigned to vertex', 2, 'is', 'YELLOW')
('color assigned to vertex', 3, 'is', 'RED')
('color assigned to vertex', 4, 'is', 'BLUE')


('color assigned to Edge', 0, 'is', 'ORANGE')
('color assigned to Edge', 1, 'is', 'GREEN')
('color assigned to Edge', 2, 'is', 'ORANGE')
('color assigned to Edge', 3, 'is', 'GREEN')
('color assigned to Edge', 4, 'is', 'PINK')
```

Result : A unique color was successfully assigned to each vertex and edge of the graph.

SRM INSTITUTE OF SCIENCE & TECHNOLOGY

DEPARTMENT OF NETWORKING & COMMUNICATIONS

18CSC305J-ARTIFICIAL INTELLIGENCE

SEMESTER – 6

BATCH-2

REGISTRATION NUMBER	**RA1911003010734**
NAME	**Kunal Singhal**

Year 2021-2022 / Even Semester

INDEX

Ex No	DATE	Title	Page No	Marks
1	14/02/22	**Implementation of constraint satisfaction problems** (Cryptarithmetic problem)		

Exercise: 3

Date : 14/02/22

Implementation of constraint satisfaction problems

Cryptarithmetic Problem

Problem Statement : The goal here is to assign each letter a digit from 0 to 9 so that the arithmetic works out correctly. The rules are that all occurrences of a letter must be assigned the same digit, and no digit can be assigned to more than one letter.

Algorithm :

• First, create a list of all the characters that need assigning to pass to Solve •
If all characters are assigned, return true if puzzle is solved, false otherwise •
Otherwise, consider the first unassigned character
• for (every possible choice among the digits not in use)
make that choice and then recursively try to assign the rest of the characters if recursion successful, return true
if !successful, unmake assignment and try another digit

Optimization technique : The algorithm above actually has a lot in common with the permutations algorithm, it pretty much just creates all arrangements of the mapping from characters to digits and tries each until one works or all have been successfully tried. For a large puzzle, this could take a while.

A smarter algorithm could take into account the structure of the puzzle and avoid going down dead-end paths. For example, if we assign the characters starting from the one's place and moving to the left, at each stage, we can verify the correctness of what we have so far before we continue onwards. This definitely complicates the code but leads to a tremendous improvement in efficiency, making it much more feasible to solve large puzzles.

• Start by examining the rightmost digit of the topmost row, with a carry of 0 • If we are beyond the leftmost digit of the puzzle, return true if no carry, false otherwise

- If we are currently trying to assign a char in one of the addends
  If char already assigned, just recur on the row beneath this one, adding value into the sum If not assigned, then
  - for (every possible choice among the digits not in use)
    make that choice and then on row beneath this one, if successful, return true if !successful, unmake assignment and try another digit
  - return false if no assignment worked to trigger backtracking

• Else if trying to assign a char in the sum
• If char assigned & matches correct,
  recur on next column to the left with carry, if success return true,
• If char assigned & doesn't match, return false
• If char unassigned & correct digit already used, return false
• If char unassigned & correct digit unused,
  assign it and recur on next column to left with carry, if success return true • return false to trigger backtracking.

Tool : aws cloud9 and Python 3.9.0

Programming code :

```
import itertools

import pdb

def get_val(word, substitution):

 s = 0

 factor = 1

 for let in reversed(word):
```

```
s += factor * substitution[let]

factor *= 10

return s

def solve(equation):

l, r = equation.lower().replace(' ', '').split('=')

print(l,r)

l = l.split('+')

print(l)

lets = set(r)

print(lets)

for word in l:

for let in word:

lets.add(let)

lets = list(lets)

print(lets)

digits = range(20)

for perm in itertools.permutations(digits, len(lets)):

sol = dict(zip(lets, perm))  if sum(get_val(word, sol) for word in l) == get_val(r, sol):  print(' + '.join(str(get_val(word, sol)) for word in l) + " = ",get_val(r, sol))
equation = input("Enter:")

solve(equation)
```


Output screen shots :

```
Enter:two+two=four
two+two four
['two', 'two']
{'f', 'r', 'o', 'u'}
['f', 'u', 't', 'w', 'o', 'r']
357 + 357 =  714
357 + 357 =  714
408 + 408 =  816
459 + 459 =  918
459 + 459 =  918
561 + 561 =  1122
612 + 612 =  1224
663 + 663 =  1326
663 + 663 =  1326
714 + 714 =  1428
806 + 806 =  1612
806 + 806 =  1612
857 + 857 =  1714
```

Result : Successfully solved the given constraint satisfaction problem.

Name : Kunal Singhal
Reg. No.: RA1911003010734

Experiment 4

AIM:

To implement Breadth-First Search and Depth-First Search and find the shortest path for an unweighted graph and compare both algorithms.

a) Breadth-First Search

Algorithm :

1. Start by putting any one of the graph’s vertices at the back of the queue.
2. Now take the front item of the queue and add it to the visited list.
3. Create a list of that vertex's adjacent nodes. Add those which are not within the visited list to the rear of the queue.
4. Keep continuing steps two and three till the queue is empty.

CODE:

```
from collections import defaultdict

# This class represents a directed graph
# using adjacency list representation
class Graph:

    # Constructor
    def __init__(self):
```

```
        # default dictionary to store graph
        self.graph = defaultdict(list)

    # function to add an edge to graph
    def addEdge(self,u,v):
        self.graph[u].append(v)

    # Function to print a BFS of graph
    def BFS(self, s):

        # Mark all the vertices as not visited
        visited = [False] * (max(self.graph) + 1)

        # Create a queue for BFS
        queue = []

        # Mark the source node as
        # visited and enqueue it
        queue.append(s)
        visited[s] = True

        while queue:

            # Dequeue a vertex from
            # queue and print it
            s = queue.pop(0)
            print (s, end = " ")

            # Get all adjacent vertices of the
            # dequeued vertex s. If a adjacent
            # has not been visited, then mark it
            # visited and enqueue it
            for i in self.graph[s]:
```

```
            if visited[i] == False:
                queue.append(i)
                visited[i] = True



# Create a graph given in
# the above diagram
g = Graph()
n = int(input("Enter the number of edges:"))
for i in range(1,n+1):
    g.addEdge(int(input("x:")),int(input("y:")))
    print("next edge!!!")


print ("Following is Breadth First Search Path:")
g.BFS(int(input("Enter vertex to start")))
```

Code screenshots:

```python
from collections import defaultdict

# This class represents a directed graph
# using adjacency list representation
class Graph:

    # Constructor
    def __init__(self):

        # default dictionary to store graph
        self.graph = defaultdict(list)

    # function to add an edge to graph
    def addEdge(self,u,v):
        self.graph[u].append(v)

    # Function to print a BFS of graph
    def BFS(self, s):

        # Mark all the vertices as not visited
        visited = [False] * (max(self.graph) + 1)

        # Create a queue for BFS
        queue = []

        # Mark the source node as
        # visited and enqueue it
        queue.append(s)
        visited[s] = True

        while queue:

            # Dequeue a vertex from
            # queue and print it
            s = queue.pop(0)
```

```python
            print (s, end = " ")

            # Get all adjacent vertices of the
            # dequeued vertex s. If a adjacent
            # has not been visited, then mark it
            # visited and enqueue it
            for i in self.graph[s]:
                if visited[i] == False:
                    queue.append(i)
                    visited[i] = True



# Create a graph given in
# the above diagram
g = Graph()
n = int(input("Enter the number of edges:"))
for i in range(1,n+1):
    g.addEdge(int(input("x:")),int(input("y:")))
    print("next edge!!!")


print ("Following is Breadth First Search Path:")
g.BFS(int(input("Enter vertex to start")))
```

Output screenshots:

```
Enter the number of edges:6
x:0
y:1
next edge!!!
x:0
y:2
next edge!!!
x:1
y:2
next edge!!!
x:2
y:0
next edge!!!
x:2
y:3
next edge!!!
x:3
y:3
next edge!!!
Following is Breadth First Traversal
Enter vertex to start2
2 0 3 1 [redacted]
```

b) **<u>Depth-First Search:</u>**

Algorithm :

1. We will start by putting any one of the graph's vertex on top of the stack.
2. After that take the top item of the stack and add it to the visited list of the vertex.

3. Next, create a list of that adjacent node of the vertex. Add the ones which aren't in the visited list of vertices to the top of the stack.
4. Lastly, keep repeating steps 2 and 3 until the stack is empty.

CODE:

```
from collections import defaultdict

# This class represents a directed graph using
# adjacency list representation


class Graph:

    # Constructor
    def __init__(self):

        # default dictionary to store graph
        self.graph = defaultdict(list)

    # function to add an edge to graph
    def 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 vertices
        visited = set()

        # Call the recursive helper function
        # to print DFS traversal
        self.DFSUtil(v, visited)

# Driver code


# Create a graph given
# in the above diagram
g = Graph()
n = int(input("Enter the number of edges:"))
for i in range(1,n+1):
    g.addEdge(int(input("x:")),int(input("y:")))
    print("next edge!!!")

print("Following is Depth First Search Path: ")
g.DFS(int(input("Enter vertex to start")))
```

Code screenshots:

```
from collections import defaultdict

# This class represents a directed graph using
# adjacency list representation


class Graph:

    # Constructor
    def __init__(self):

        # default dictionary to store graph
        self.graph = defaultdict(list)

    # function to add an edge to graph
    def 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 vertices
        visited = set()

        # Call the recursive helper function
        # to print DFS traversal
        self.DFSUtil(v, visited)

# Driver code


# Create a graph given
# in the above diagram
g = Graph()
n = int(input("Enter the number of edges:"))
for i in range(1,n+1):
    g.addEdge(int(input("x:")),int(input("y:")))
    print("next edge!!!")

print("Following is Depth First Search Path: ")
g.DFS(int(input("Enter vertex to start")))
```

Output screenshots:

```
Enter the number of edges:6
x:0
y:1
next edge!!!
x:0
y:2
next edge!!!
x:1
y:2
next edge!!!
x:2
y:0
next edge!!!
x:2
y:3
next edge!!!
x:3
y:3
next edge!!!
Following is Depth First Search Path:
Enter vertex to start1
1 2 0 3 [redacted]
```

Name : Kunal Singhal
Reg. No.: RA1911003010734

Experiment 5

AIM:

To implement Best First Search algorithm and A* algorithm using Python programming language.

a) Best First Search:

Code:

```
from queue import PriorityQueue
import matplotlib.pyplot as plt
import networkx as nx

# for implementing BFS | returns path having lowest cost
def best_first_search(source, target, n):
    visited = [0] * n
    visited[source] = True
    pq = PriorityQueue()
    pq.put((0, source))
    while pq.empty() == False:
        u = pq.get()[1]
        print(u, end=" ") # the path having lowest cost
        if u == target:
            break

        for v, c in graph[u]:
            if visited[v] == False:
                visited[v] = True
                pq.put((c, v))
    print()

# for adding edges to graph
def addedge(x, y, cost):
    graph[x].append((y, cost))
    graph[y].append((x, cost))
```

```
G = nx.Graph()
v = int(input("Enter the number of nodes: "))
graph = [[] for i in range(v)] # undirected Graph
e = int(input("Enter the number of edges: "))
print("Enter the edges along with their weights:")
for i in range(e):
    x, y, z = list(map(int, input().split()))
    addedge(x, y, z)
    G.add_edge(x, y, weight = z)

source = int(input("Enter the Source Node: "))
target = int(input("Enter the Target/Destination Node: "))
print("\nPath: ", end = "")
best_first_search(source, target, v)


print("Graph:\n")
pos = nx.spring_layout(G, seed=7)  # positions for all nodes - seed for reproducibility

# nodes
nx.draw_networkx_nodes(G, pos, node_size=350)

# edges
nx.draw_networkx_edges(G, pos)
nx.draw_networkx_edges(G, pos, alpha=0.5, edge_color="r")

# labels
nx.draw_networkx_labels(G, pos, font_size=20)

ax = plt.gca()
ax.margins(0.08)
plt.axis("off")
plt.tight_layout()
plt.show()
```

Code Screenshots:

```python
from queue import PriorityQueue
import matplotlib.pyplot as plt
import networkx as nx

# for implementing BFS | returns path having lowest cost
def best_first_search(source, target, n):
    visited = [0] * n
    visited[source] = True
    pq = PriorityQueue()
    pq.put((0, source))
    while pq.empty() == False:
        u = pq.get()[1]
        print(u, end=" ") # the path having lowest cost
        if u == target:
            break

        for v, c in graph[u]:
            if visited[v] == False:
                visited[v] = True
                pq.put((c, v))
    print()

# for adding edges to graph
def addedge(x, y, cost):
    graph[x].append((y, cost))
    graph[y].append((x, cost))

G = nx.Graph()
v = int(input("Enter the number of nodes: "))
graph = [[] for i in range(v)] # undirected Graph
e = int(input("Enter the number of edges: "))
print("Enter the edges along with their weights:")
for i in range(e):
    x, y, z = list(map(int, input().split()))
    addedge(x, y, z)
    G.add_edge(x, y, weight = z)
```

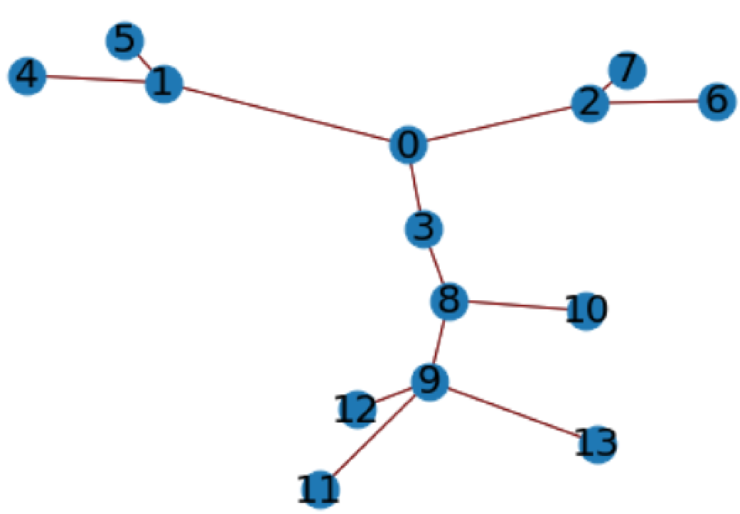
```python
source = int(input("Enter the Source Node: "))
target = int(input("Enter the Target/Destination Node: "))
print("\nPath: ", end = "")
best_first_search(source, target, v)
```

```python
print("Graph:\n")
pos = nx.spring_layout(G, seed=7)  # positions for all nodes - seed for reproducibility

# nodes
nx.draw_networkx_nodes(G, pos, node_size=350)

# edges
nx.draw_networkx_edges(G, pos)
nx.draw_networkx_edges(G, pos, alpha=0.5, edge_color="r")

# labels
nx.draw_networkx_labels(G, pos, font_size=20)

ax = plt.gca()
ax.margins(0.08)
plt.axis("off")
plt.tight_layout()
plt.show()
```

Output Screenshots:

```
Enter the number of nodes: 14
Enter the number of edges: 13
Enter the edges along with their weights:
0 1 3
0 2 6
0 3 5
1 4 9
1 5 8
2 6 12
2 7 14
3 8 7
8 9 5
8 10 6
9 11 1
9 12 10
9 13 2
Enter the Source Node: 0
Enter the Target/Destination Node: 9

Path: 0 1 3 2 8 9
```

Graph:



b) **A* Algorithm:**

Code:

```
from collections import deque
class Graph:
    def __init__(self, adjacency_list):
        self.adjacency_list = adjacency_list

    def get_neighbors(self, v):
        return self.adjacency_list[v]

    def h(self, n):
        H = {
            'A': 1,
            'B': 1,
            'C': 1,
            'D': 1,
            }
        return H[n]
    def a_star_algorithm(self, start_node, stop_node):
        open_list = set([start_node])
        closed_list = set([])
        g = {}
        g[start_node] = 0
        parents = {}
        parents[start_node] = start_node
        while len(open_list) > 0:
            n = None
            for v in open_list:
                if n == None or g[v] + self.h(v) < g[n] + self.h(n):
                    n = v;
            if n == None:
                print('Path does not exist!')
                return None
```

```
        if n == stop_node:
            reconst_path = []
            while parents[n] != n:
                reconst_path.append(n)
                n = parents[n]
            reconst_path.append(start_node)
            reconst_path.reverse()
            print('Path found: {}'.format(reconst_path))
            return reconst_path
        for (m, weight) in self.get_neighbors(n):
            if m not in open_list and m not in closed_list:
                open_list.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_list:
                        closed_list.remove(m)
                        open_list.add(m)
        open_list.remove(n)
        closed_list.add(n)
    print('Path does not exist!')
    return None

adjacency_list = {
'A': [('B', 1), ('C', 3), ('D', 7)],
'B': [('D', 5)],
'C': [('D', 12)]
}

graph1 = Graph(adjacency_list)
graph1.a_star_algorithm('A', 'D')
```

Code Screenshots:

```python
from collections import deque
class Graph:
    def __init__(self, adjacency_list):
        self.adjacency_list = adjacency_list
    def get_neighbors(self, v):
        return self.adjacency_list[v]
    def h(self, n):
        H = {
            'A': 1,
            'B': 1,
            'C': 1,
            'D': 1,
        }
        return H[n]
    def a_star_algorithm(self, start_node, stop_node):
        open_list = set([start_node])
        closed_list = set([])
        g = {}
        g[start_node] = 0
        parents = {}
        parents[start_node] = start_node
        while len(open_list) > 0:
            n = None
            for v in open_list:
                if n == None or g[v] + self.h(v) < g[n] + self.h(n):
                    n = v;
            if n == None:
                print('Path does not exist!')
                return None
            if n == stop_node:
                reconst_path = []
```

```
                reconst_path = []
                while parents[n] != n:
                    reconst_path.append(n)
                    n = parents[n]
                reconst_path.append(start_node)
                reconst_path.reverse()
                print('Path found: {}'.format(reconst_path))
                return reconst_path
            for (m, weight) in self.get_neighbors(n):
                if m not in open_list and m not in closed_list:
                    open_list.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_list:
                            closed_list.remove(m)
                            open_list.add(m)
            open_list.remove(n)
            closed_list.add(n)
        print('Path does not exist!')
        return None
adjacency_list = {
    'A': [('B', 1), ('C', 3), ('D', 7)],
    'B': [('D', 5)],
    'C': [('D', 12)]
}
graph1 = Graph(adjacency_list)
graph1.a_star_algorithm('A', 'D')
```

Output Screenshots:

```
Path found: ['A', 'B', 'D']
```

SRM INSTITUTE OF SCIENCE & TECHNOLOGY

DEPARTMENT OF NETWORKING & COMMUNICATIONS

18CSC305J-ARTIFICIAL INTELLIGENCE

SEMESTER –

6 BATCH-2

REGISTRATION NUMBER	**RA1911003010734**
NAME	**Kunal Singhal**

INDEX

Ex No	DATE	Title	Page No	Marks
6		**Implementation of unification and resolution for real world problems.**		

Experiment No: 6

IMPLEMENTATION OF UNIFICATION AND RESOLUTION

PROBLEM STATEMENT : Developing an optimized technique using an appropriate artificial intelligence algorithm to solve the Unification and Resolution.

ALGORITHM :

1. function PL-RESOLUTION (KB, Q) returns true or false inputs: KB,
2. the knowledge base, group of sentences/facts in propositional logic
3. Q, the query, a sentence in propositional logic
4. clauses → the set of clauses in the CNF representation of KB ^ Q new → { }
5. loop do for each Ci, Cj in clauses do
6. resolvents → PL-RESOLVE (Ci, Cj)
7. if resolvents contains the empty clause the return true
8. new → new union resolvents
9. if new is a subset of clauses then return false
10. clauses → clauses union true

OPTIMIZATION TECHNIQUE:

Resolution basically works by using the principle of proof by contradiction. To find the conclusion we should negate the conclusion. Then the resolution rule is applied to the resulting clauses. Each clause that contains complementary literals is resolved to produce a2. new clause, which can be added to the set of facts (if it is not already present). This process continues until one of the two things happen:•There are no new clauses that can be added. An application of the resolution rule derives the empty clauseAn empty clause shows that the negation of the conclusion is a complete contradiction,hence the negation of the conclusion is invalid or false or the assertion is completely valid or true.

1. Convert the given statements in Predicate/Propositional Logic
2. Convert these statements into Conjunctive Normal Form
3. Negate the Conclusion (Proof by Contradiction)
4. Resolve using a Resolution Tree (Unification)

CODE UNIFICATION :

```
def
    get_index_comma(string):
    index_list =  list()
    par_count = 0

    for i in range(len(string)):
        if string[i] == ',' and par_count == 0:
            index_list.append(i)
             elif string[i] ==
        '(': par_count += 1
             elif string[i] ==
                              ')':
            par_count -= 1

    return index_list



def is_variable(expr):
    for i in expr:
        if i == '(' or i == ')':
            return False

    return True
```

```
def process_expression(expr):
    expr = expr.replace(' ', '')
    index = None
    for i in range(len(expr)):
        if expr[i] == '(':
            index = i
            break
    predicate_symbol = expr[:index]
    expr = expr.replace(predicate_symbol, '')
    expr = expr[1:len(expr) - 1]
    arg_list = list()
    indices = get_index_comma(expr)

    if len(indices) == 0:
        arg_list.append(expr)
    else:
        arg_list.append(expr[:indices[0]])
                for i, j in zip(indices,
                        indices[1:]):
            arg_list.append(expr[i + 1:j])
        arg_list.append(expr[indices[len(indices) - 1] + 1:])

    return predicate_symbol, arg_list



def get_arg_list(expr):
    _, arg_list = process_expression(expr)

    flag = True
    while flag:
```

```
        flag = False

        for i in arg_list:
            if not is_variable(i):
                flag = True
                _, tmp = process_expression(i)
                for j in tmp:
                    if j not in arg_list:
                        arg_list.append(j)
                arg_list.remove(i)

    return arg_list


  def check_occurs(var, expr):
    arg_list = get_arg_list(expr)
    if var in arg_list:
        return True

    return False


def unify(expr1, expr2):

    if is_variable(expr1) and is_variable(expr2):
        if expr1 == expr2:
            return 'Null'
        else:
            return False
    elif is_variable(expr1) and not is_variable(expr2):
        if check_occurs(expr1, expr2):
```

```
    return False
else:
```

```
        tmp = str(expr2) + '/' + str(expr1)
        return tmp
elif not is_variable(expr1) and is_variable(expr2):
if check_occurs(expr2, expr1):
        return False
    else:
        tmp = str(expr1) + '/' + str(expr2)
        return tmp
else:
    predicate_symbol_1, arg_list_1 = process_expression(expr1)
    predicate_symbol_2, arg_list_2 = process_expression(expr2)

    # Step 2
    if predicate_symbol_1 != predicate_symbol_2:
        return False
    # Step 3
    elif len(arg_list_1) != len(arg_list_2):
        return False
    else:
        # Step 4: Create substitution list
        sub_list = list()

        # Step 5:
        for i in range(len(arg_list_1)):
            tmp = unify(arg_list_1[i], arg_list_2[i])

            if not tmp:
                return False
            elif tmp == 'Null':
                pass
```

```
else:
            if type(tmp) == list:
                for j in tmp:
                    sub_list.append(j)
            else:
                sub_list.append(tmp)

        # Step 6
        return sub_list

if __name__== '__main__':
    # f1 = 'Q(a, g(x, a), f(y))'
    # f2 = 'Q(a, g(f(b), a), x)'
    f1 = input('f1 : ')
    f2 = input('f2 : ')

    result = unify(f1, f2)
    if not result:
        print('The process of Unification failed!')
    else:
        print('The process of Unification successful!')
        print(result)
```

OUTPUT UNIFICATION :

```
Vaishnavimoorthy:~/environment/RA1811028010049 $ python unification.py
f1 : 'Q(a, g(x, a), f(y))'
f2 : 'Q(a, g(f(b), a), x)'
The process of Unification successful!
['f(b)/x', 'f(y)/x']
Vaishnavimoorthy:~/environment/RA1811028010049 $
```

SRM INSTITUTE OF SCIENCE & TECHNOLOGY

DEPARTMENT OF NETWORKING &

COMMUNICATIONS **18CSC305J-ARTIFICIAL**

INTELLIGENCE

SEMESTER – 6 BATCH-2

REGISTRATION NUMBER	**RA1911003010734**
NAME	**Kunal Singhal**

INDEX

Ex No	DATE	Title	Page No	Marks
7	21/03/22	**Implementation of uncertain methods for an application (Fuzzy logic/ Dempster Shafer Theory)**		

Experiment No : 7

IMPLEMENTATION OF UNCERTAIN METHODS OF AN APPLICATION

Problem Statement:

To implement Fuzzy logic using matplotlib in python and find the graph of temperature, humidity and speed in different conditions.

Algorithm:

1. Locate the input, output, and state variables of the plane under consideration.
2. Split the complete universe of discourse spanned by each variable into a number of fuzzy subsets, assigning each with a linguistic label. The subsets include all the elements in the universe.
3. Obtain the membership function for each fuzzy subset.
4. Assign the fuzzy relationships between the inputs or states of fuzzy subsets on one side and the output of fuzzy subsets on the other side, thereby forming the rule base.
5. Choose appropriate scaling factors for the input and output variables for normalizing the variables between [0, 1] and [-1, I] interval.
6. Carry out the fuzzification process.
7. Identify the output contributed from each rule using fuzzy approximate reasoning.
8. Combine the fuzzy outputs obtained from each rule.
9. Finally, apply defuzzification to form a crisp output.

Optimization Technique:

1. Decomposing the large-scale system into a collection of various subsystems.
2. Varying the plant dynamics slowly and linearizing the nonlinear plane dynamics about a set of operating points.
3. Organizing a set of state variables, control variables, or output features for the system under consideration.
4. Designing simple P, PD, PID controllers for the subsystems. Optimal controllers can also

be designed.

Uncertainty In this problem : Fuzzy Logic - Temperature, Humidity and Speed.

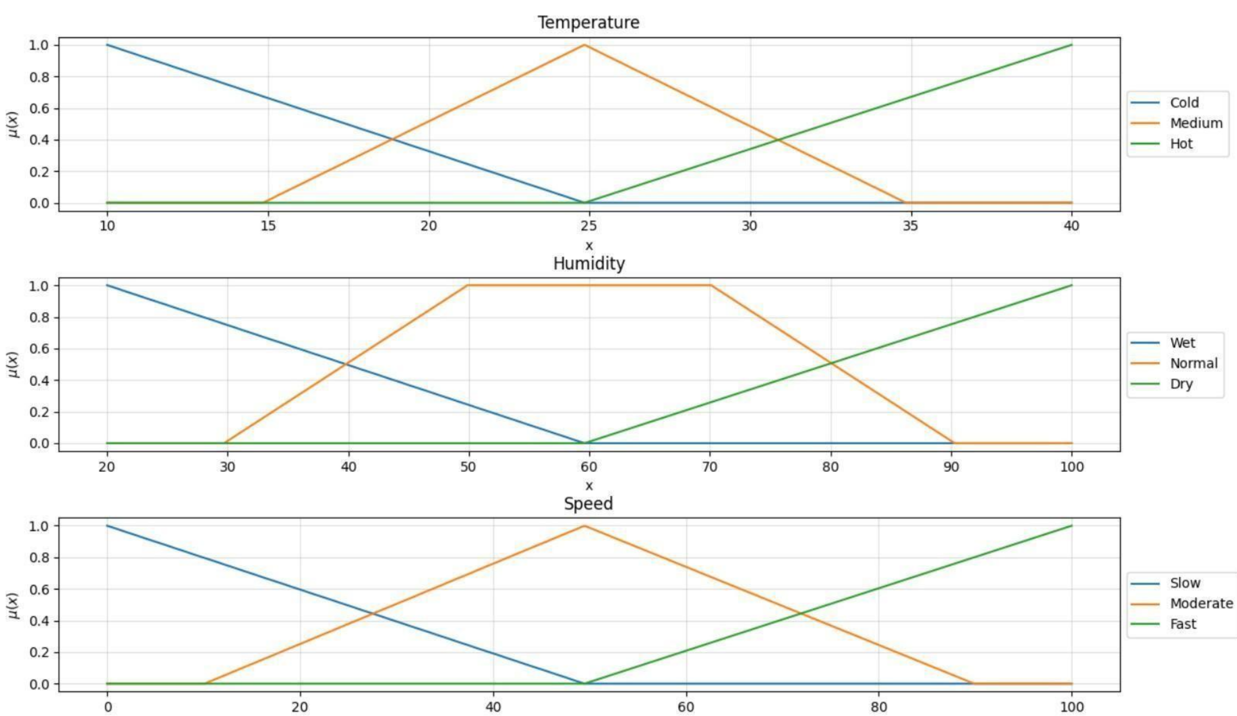
CODE :

```
from fuzzy_system.fuzzy_variable_output import FuzzyOutputVariable
from fuzzy_system.fuzzy_variable_input import FuzzyInputVariable
# from fuzzy_system.fuzzy_variable import FuzzyVariable
from fuzzy_system.fuzzy_system import FuzzySystem
temp = FuzzyInputVariable('Temperature', 10, 40, 100)
temp.add_triangular('Cold', 10, 10, 25)
temp.add_triangular('Medium', 15, 25, 35)
temp.add_triangular('Hot', 25, 40, 40)
humidity = FuzzyInputVariable('Humidity', 20, 100, 100)
humidity.add_triangular('Wet', 20, 20, 60)
humidity.add_trapezoidal('Normal', 30, 50, 70, 90)
humidity.add_triangular('Dry', 60, 100, 100)
motor_speed = FuzzyOutputVariable('Speed', 0, 100, 100)
motor_speed.add_triangular('Slow', 0, 0, 50)
motor_speed.add_triangular('Moderate', 10, 50, 90)
motor_speed.add_triangular('Fast', 50, 100, 100)

system = FuzzySystem()
system.add_input_variable(temp)
system.add_input_variable(humidity)
system.add_output_variable(motor_speed)

system.add_rule(
                { 'Temperature':'Cold',
                        'Humidity':'Wet' },
                { 'Speed':'Slow'})

system.add_rule(
```

```
                { 'Temperature':'Cold',
                        ‘'Humidity':'Normal' },
                { 'Speed':'Slow'})

system.add_rule(
                { 'Temperature':'Medium',
                        'Humidity':'Wet' },
                { 'Speed':'Slow'})

system.add_rule(
                { 'Temperature':'Medium',
                        'Humidity':'Normal' },
                { 'Speed':'Moderate'})

system.add_rule(
                { 'Temperature':'Cold',
                        'Humidity':'Dry' },
                { 'Speed':'Moderate'})

system.add_rule(
                { 'Temperature':'Hot',
                        'Humidity':'Wet' },
                { 'Speed':'Moderate'})

system.add_rule(
                { 'Temperature':'Hot',
                        'Humidity':'Normal' },
                { 'Speed':'Fast'})

system.add_rule(
                { 'Temperature':'Hot',
                        'Humidity':'Dry' },
```

```
                { 'Speed':'Fast'})

system.add_rule(
                { 'Temperature':'Medium',
                        'Humidity':'Dry' },
                { 'Speed':'Fast'})


output = system.evaluate_output({
                                'Temperature':18,
                                'Humidity':60
                })


print(output)
system.plot_system()
```

OUTPUT :



Result: We have successfully implemented fuzzy uncertainty problem using matplotlib and output is received.

Name : Kunal Singhal
Reg. No.: RA1911003010734

Implementation of learning algorithms for an application

Aim:

a) Implementation of Linear Regression algorithm to predict students' score using the given dataset.
b) Implementation of Support Vector Classification algorithm to classify the cases of breast cancer
using the given dataset.
c) Implementation of K-means clustering algorithm to group the customers based on their demographic detail using the given dataset.

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
%matplotlib inline
Import required modules and packages
dataset = pd.read_csv('….\student_scores.csv')
dataset.head()
Import data set
Choose the right path for the dataset
dataset.describe() Descriptive statistics of the attributes
available in the dataset
dataset.plot(x='Hours', y='Scores', style='o')
plt.title('Hours vs Percentage')
plt.xlabel('Hours Studied')
plt.ylabel('Percentage Score')
plt.show()
Visualize the data.
X = dataset.iloc[:, :-1].values
```

y = dataset.iloc[:, 1].values
Identify the independent (X) and
dependent variables (y) in the data set
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=0)
print('X train shape: ', X_train.shape)
print('Y train shape: ', Y_train.shape)
print('X test shape: ', X_test.shape)
Splitting the given data in to training set
(80%) and testing set (20%)
Beginners Level
Lab 11 - Implementation of Learning Algorithms for an Application
18CSC305J - ARTIFICIAL INTELLIGENCE Page 4
print('Y test shape: ', Y_test.shape)
regressor = LinearRegression()
Model instantiation
regressor.fit(X_train, y_train) Model Training
print(regressor.intercept_)
print(regressor.coef_)
Finding out the coefficient (a) and
intercept (b) value of linear model
(y=aX+b)
y_pred = regressor.predict(X_test)
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
print(df)
Testing the model
print('Mean Absolute Error:',
metrics.mean_absolute_error (y_test, y_pred))
print('Mean Squared Error:',
metrics.mean_squared_error (y_test, y_pred))
print('Root Mean Squared Error:',
np.sqrt(metrics.mean_squared_error (y_test, y_pred)))

MAE, MSE, RMSE – Evaluation metrics
of Model
Discussion:

```
[ ] dataset = pd.read_csv('C:\\Users\DELL\Desktop\student_scores.csv')
    dataset.shape()
```

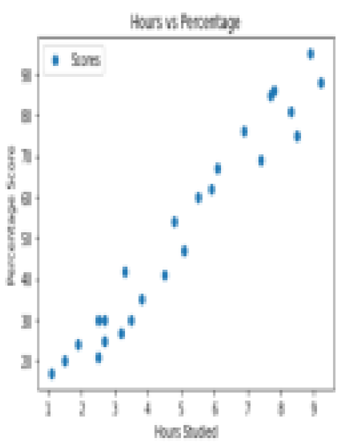
	Hours	Scores
0	2.5	21
1	5.1	47
2	3.2	27
3	8.5	75
4	3.5	30

```
dataset.shape
```

```
(25, 2)
```

```
dataset.describe()
```

	Hours	Scores
count	25.000000	25.000000
mean	5.012000	51.480000
std	2.525094	25.286887
min	1.100000	17.000000
25%	2.700000	30.000000
50%	4.800000	47.000000
75%	7.400000	75.000000
max	9.200000	95.000000



```
[ ] print(regressor.coef_)

    [9.91065648]

[ ] y_pred = regressor.predict(X_test)
    df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
    df
```

	Actual	Predicted
0	20	16.884145
1	27	33.732261
2	69	75.357018
3	30	26.794801
4	62	60.491033

```
[ ] print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
    print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
    print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

    Mean Absolute Error: 4.183859899002975
    Mean Squared Error: 21.5987693072174
    Root Mean Squared Error: 4.6474476121003665
```

B) Code:

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix,
classification_report
Import required modules and packages
dataset = pd.read_csv('….\diabetes data.csv')
print(dataset.head())
Import data set
Choose the right path for the dataset
def diagnosis(x):
if x=='M' :
return 1
if x=='B' :
return 0
dataset['diagnosis'] = dataset['diagnosis'].apply(diagnosis)
print(dataset)
Data cleaning process. Converting
categorical value in to numerical value.
M = malignant, B = benign
print("Any missing sample in data set:",
dataset.isnull().values.any(), "\n")
Check for any missing values in the data
set
dataset = dataset.replace([np.inf, -np.inf], np.nan)
dataset= dataset.fillna(dataset.mean())
dataset
Replace the missing value with its mean

value of the respective attribute
dataset= dataset.drop(columns=["Unnamed: 32"]) drop this column because it's not necessary (null)
Y = dataset['diagnosis']
X = dataset.drop(columns=['diagnosis'])
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=9)
print('X train shape: ', X_train.shape)
print('Y train shape: ', Y_train.shape)
print('X test shape: ', X_test.shape)
print('Y test shape: ', Y_test.shape)
Splitting the given data in to training set (80%) and testing set (20%)
svc_classifier= SVC(kernel='poly') Model instantiation. Apply SVM with different kernels 'linear', 'poly', 'rbf', 'sigmoid' and verify the accuracy of the model
svc_classifier.fit(X_train,Y_train) Model Training
y_pred=svc_classifier.predict(X_test) Testing the model
print(confusion_matrix(Y_test,y_pred))
print(classification_report(Y_test,y_pred))
Evaluation metrics to measure the performance of the model

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix,classification_report
%matplotlib inline
```

```
dataset = pd.read_csv('C:\\Users\DELL\Desktop\data.csv')
dataset.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	texture_worst	perimeter_
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	17.33	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	23.41	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	25.53	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	26.50	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	16.67	

5 rows × 33 columns

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	texture_worst	perimete
0	842302	1	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	...	17.33	
1	842517	1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	...	23.41	
2	84300903	1	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	...	25.53	
3	84348301	1	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	...	26.50	
4	84358402	1	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	...	16.67	
...	...	...	...	...	...	...	...	...	...	...	...	...	
564	926424	1	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	...	26.40	
565	926682	1	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	...	38.25	
566	926954	1	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	...	34.12	
567	927241	1	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	...	39.42	
568	92751	0	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	...	30.37	

569 rows × 33 columns

```
] svc_classifier= SVC(kernel='rbf')
  svc_classifier
```

```
SVC()
```

```
] svc_classifier=svc_classifier.fit(X_train,Y_train)
```

```
] y_pred=svc_classifier.predict(X_test)
```

```
] print(confusion_matrix(Y_test,y_pred))
```

```
[[74  0]
 [40  0]]
```

```
] print(classification_report(Y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.65      1.00      0.79        74
           1       0.00      0.00      0.00        40

    accuracy                           0.65       114
   macro avg       0.32      0.50      0.39       114
weighted avg       0.42      0.65      0.51       114
```

(c) Implementation of K-means clustering algorithm to group the customers based on their demographic detail using the given dataset.

Problem: Client is owing a supermarket mall and through membership cards, client have some basic data

about your customers like Customer ID, age, gender, annual income and spending score. Help the client

to understand the customers like who are the target customers so that the sense can be given to marketing

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
%matplotlib inline
```

```
data=pd.read_csv('C:\\Users\DELL\Desktop\mall_customers.csv')
print(data.head())
```

```
   CustomerID   Genre  Age  Annual Income (k$)  Spending Score (1-100)
0           1    Male   19                  15                      39
1           2    Male   21                  15                      81
2           3  Female   20                  16                       6
3           4  Female   23                  16                      77
4           5  Female   31                  17                      40
```

```
inVsout=data.iloc[:,[3,4]]
inVsout
```

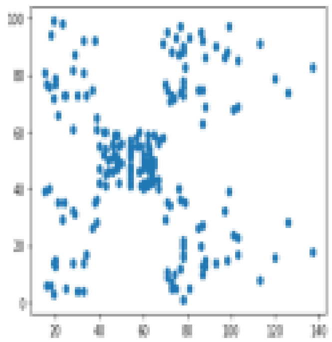
```
inVsout=data.iloc[:,[3,4]]
inVsout
```

	Annual Income (k$)	Spending Score (1-100)
0	15	39
1	15	81
2	16	6
3	16	77
4	17	40
...	...	...
195	120	79
196	126	28
197	126	74
198	137	18
199	137	83

200 rows × 2 columns

```
plt.scatter(inVsout.iloc[:,0],inVsout.iloc[:,1])
```

<matplotlib.collections.PathCollection at 0x1fa26e7ca90>

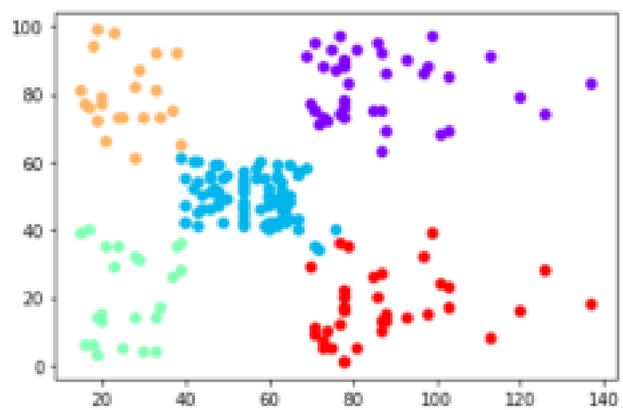


```
kmeans=KMeans(n_clusters=5)
kmeans.fit(inVsout)
```

KMeans(n_clusters=5)

```
plt.scatter(inVsout.iloc[:,0],inVsout.iloc[:,1], c=kmeans.labels_, cmap='rainbow')
plt.show()
```


```
plt.scatter(inVsout.iloc[:,0],inVsout.iloc[:,1], c=kmeans.labels_, cmap='rainbow')
plt.show()
```

	Annual Income (k$)	Spending Score (1-100)
0	15	39
2	16	6
4	17	40
6	18	6
8	19	3
10	19	14
12	20	15
14	20	13
16	21	35
18	23	29
20	24	35
22	25	5
24	28	14
26	28	32
28	29	31
30	30	4
32	33	4
34	33	14

```
silhouette_score(inVsout,kmeans.labels_)
```

```
0.553931997444648
```

Name : Kunal Singhal
Reg. No.: RA1911003010734

Aim: To Implement NLP programs

NLP stands for **Natural Language Processing**, which is a part of **Computer Science, Human language,** and **Artificial Intelligence**. It is the technology that is used by machines to understand, analyse, manipulate, and interpret human's languages. It helps developers to organize knowledge for performing tasks such as **translation, automatic summarization, Named Entity Recognition (NER), speech recognition, relationship extraction,** and **topic segmentation**.

code:-

```
!pip install -q wordcloud
import wordcloud

import nltk
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')

import pandas as pd
import matplotlib.pyplot as plt
import io
import unicodedata
import numpy as np
import re
import string
# Constants
# POS (Parts Of Speech) for: nouns, adjectives, verbs and adverbs
DI_POS_TYPES = {'NN':'n', 'JJ':'a', 'VB':'v', 'RB':'r'}
POS_TYPES = list(DI_POS_TYPES.keys())

# Constraints on tokens
MIN_STR_LEN = 3
RE_VALID = '[a-zA-Z]'
# Upload from google drive
from google.colab import files
uploaded = files.upload()
print("len(uploaded.keys():", len(uploaded.keys()))
```

```
for fn in uploaded.keys():
  print('User uploaded file "{name}" with length {length}
bytes'.format(name=fn, length=len(uploaded[fn])))

# Get list of quotes
df_quotes = pd.read_csv(io.StringIO(uploaded['quotes.txt'].decode('utf-8')), sep='\t')

# Display
print("df_quotes:")
print(df_quotes.head().to_string())
print(df_quotes.describe())

# Convert quotes to list
li_quotes = df_quotes['Quote'].tolist()
print()
print("len(li_quotes):", len(li_quotes)
# Get stopwords, stemmer and lemmatizer
stopwords = nltk.corpus.stopwords.words('english')
stemmer = nltk.stem.PorterStemmer()
lemmatizer = nltk.stem.WordNetLemmatizer()

# Remove accents function
def remove_accents(data):
    return ''.join(x for x in unicodedata.normalize('NFKD', data) if x in string.ascii_letters or x == " ")

# Process all quotes
li_tokens = []
li_token_lists = []
li_lem_strings = []

for i,text in enumerate(li_quotes):
    # Tokenize by sentence, then by lowercase word
    tokens = [word.lower() for sent in nltk.sent_tokenize(text) for word in nltk.word_tokenize(sent)]

    # Process all tokens per quote

    li_tokens_quote = []
    li_tokens_quote_lem = []
    for token in tokens:
        # Remove accents
        t = remove_accents(token)
```

```
        # Remove punctuation
        t = str(t).translate(string.punctuation)
        li_tokens_quote.append(t)

        # Add token that represents "no lemmatization match"
        li_tokens_quote_lem.append("-") # this token will be removed if a lemmatization match is found
below

        # Process each token
        if t not in stopwords:
            if re.search(RE_VALID, t):
                if len(t) >= MIN_STR_LEN:
                    # Note that the POS (Part Of Speech) is necessary as input to the lemmatizer
                    # (otherwise it assumes the word is a noun)
                    pos = nltk.pos_tag([t])[0][1][:2]
                    pos2 = 'n' # set default to noun
                    if pos in DI_POS_TYPES:
                      pos2 = DI_POS_TYPES[pos]

                    stem = stemmer.stem(t)
                    lem = lemmatizer.lemmatize(t, pos=pos2) # lemmatize with the correct POS

                    if pos in POS_TYPES:
                        li_tokens.append((t, stem, lem, pos))

                        # Remove the "-" token and append the lemmatization match
                        li_tokens_quote_lem = li_tokens_quote_lem[:-1]
                        li_tokens_quote_lem.append(lem)

    # Build list of token lists from lemmatized tokens
    li_token_lists.append(li_tokens_quote)

    # Build list of strings from lemmatized tokens
    str_li_tokens_quote_lem = ' '.join(li_tokens_quote_lem)
    li_lem_strings.append(str_li_tokens_quote_lem)

# Build resulting dataframes from lists
df_token_lists = pd.DataFrame(li_token_lists)

print("df_token_lists.head(5):")
print(df_token_lists.head(5).to_string())
```

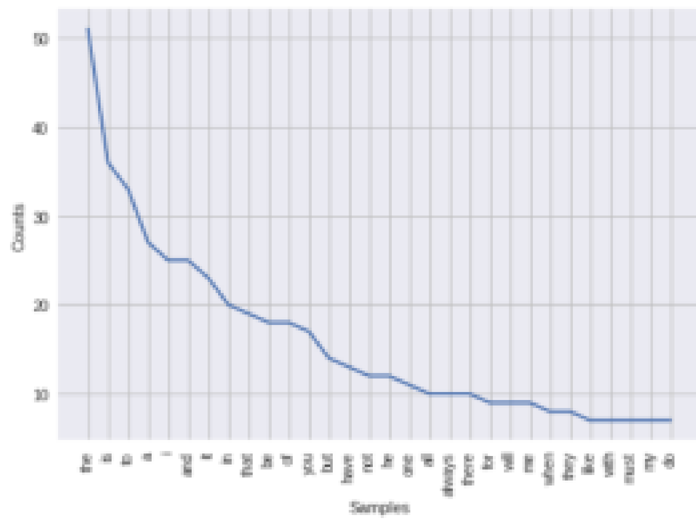
```
# Replace None with empty string
for c in df_token_lists:
    if str(df_token_lists[c].dtype) in ('object', 'string_', 'unicode_'):
        df_token_lists[c].fillna(value='', inplace=True)

df_lem_strings = pd.DataFrame(li_lem_strings, columns=['lem quote'])

print()
print("")
print("df_lem_strings.head():")
print(df_lem_strings.head().to_string())
# Add counts
print("Group by lemmatized words, add count and sort:")
df_all_words = pd.DataFrame(li_tokens, columns=['token', 'stem', 'lem', 'pos']) df_all_words['counts']
= df_all_words.groupby(['lem'])['lem'].transform('count') df_all_words =
df_all_words.sort_values(by=['counts', 'lem'], ascending=[False, True]).reset_index()

print("Get just the first row in each lemmatized group")
df_words = df_all_words.groupby('lem').first().sort_values(by='counts',
ascending=False).reset_index()
print("df_words.head(10):")
print(df_words.head(10))
df_words = df_words[['lem', 'pos', 'counts']].head(200)
for v in POS_TYPES:
    df_pos = df_words[df_words['pos'] == v]
    print()
    print("POS_TYPE:", v)
    print(df_pos.head(10).to_string())
li_token_lists_flat = [y for x in li_token_lists for y in x] # flatten the list of token lists to a single list
print("li_token_lists_flat[:10]:", li_token_lists_flat[:10])
di_freq = nltk.FreqDist(li_token_lists_flat)
del di_freq['']
li_freq_sorted = sorted(di_freq.items(), key=lambda x: x[1], reverse=True) # sorted list
print(li_freq_sorted)

di_freq.plot(30, cumulative=False)
li_lem_words = df_all_words['lem'].tolist()
di_freq2 = nltk.FreqDist(li_lem_words)
li_freq_sorted2 = sorted(di_freq2.items(), key=lambda x: x[1], reverse=True) # sorted list
print(li_freq_sorted2)

di_freq2.plot(30, cumulative=False)
```

Output:-

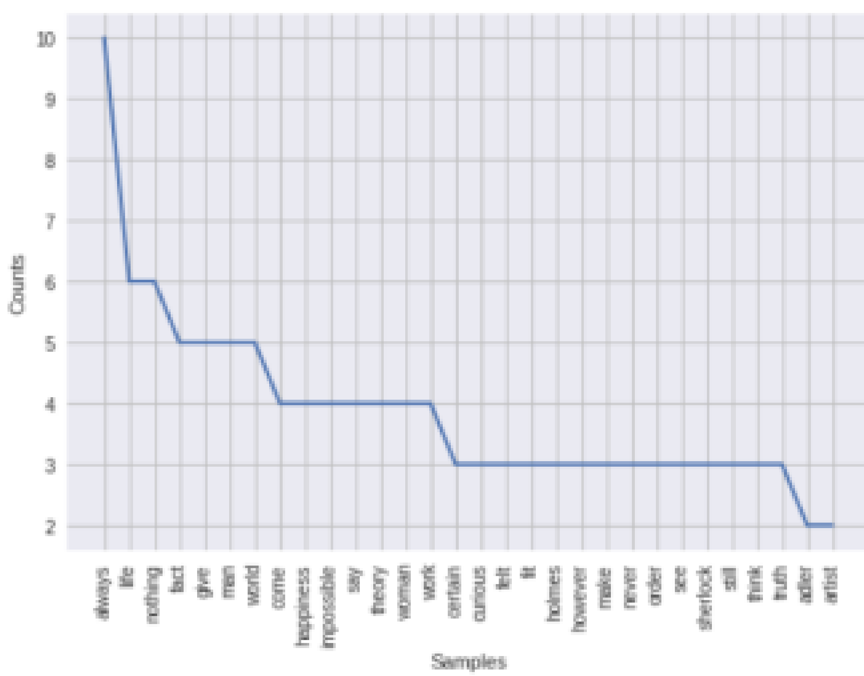
```
Group by lemmatized words, add count and sort:
Get just the first row in each lemmatized group
df_words.head(10):
         lem  index      token    stem pos  counts
0     always     50     always   alway  RB      10
1    nothing    116    nothing    noth  NN       6
2       life     54       life    life  NN       6
3        man     74        man     man  NN       5
4       give     39       gave    gave  VB       5
5       fact    106       fact    fact  NN       5
6      world    121      world   world  NN       5
7  happiness    119  happiness   happi  NN       4
8       work    297       work    work  NN       4
9     theory    101     theory  theori  NN       4
```

```
POS_TYPE: NN
          lem pos  counts
1     nothing  NN       6
2        life  NN       6
3         man  NN       5
5        fact  NN       5
6       world  NN       5
7   happiness  NN       4
8        work  NN       4
9      theory  NN       4
10      woman  NN       4
17     holmes  NN       3

POS_TYPE: JJ
              lem pos  counts
11     impossible  JJ       4
15        certain  JJ       3
18        curious  JJ       3
34           nice  JJ       2
43         little  JJ       2
48           good  JJ       2
61     improbable  JJ       2
62           best  JJ       2
72  philosophical  JJ       1
81       possible  JJ       1

POS_TYPE: VB
          lem pos  counts
4        give  VB       5
12        say  VB       4
13       come  VB       4
22        see  VB       3
23       make  VB       3
26      think  VB       3
```

```
li_token_lists_flat[:10]: ['i', 'like', 'living', '', 'i', 'have', 'sometimes', 'been', 'wildly', '']
[('the', 51), ('is', 36), ('to', 33), ('a', 27), ('i', 25), ('and', 25), ('it', 23), ('in', 20), ('that
```




```
[('always', 10), ('life', 6), ('nothing', 6), ('fact', 5), ('give', 5), ('man', 5),
```



Result:-Thus the NPL program was implemented

SRM INSTITUTE OF SCIENCE & TECHNOLOGY

DEPARTMENT OF NETWORKING & COMMUNICATIONS

18CSC305J-ARTIFICIAL INTELLIGENCE

SEMESTER – 6

BATCH-2

REGISTRATION NUMBER	**RA1911003010734**
NAME	**Kunal Singhal**

INDEX

Ex No	DATE	Title	Page No	Marks
11	05-04-2022	Applying any deep learning methods to solve an application		

Exercise: 11

Date: 05-04-2022

APPLYING DEEP LEARNING METHODS TO SOLVE AN APPLICATION

Problem Statement:

To develop a Deep Neural Network to predict cancer as malignant or benign.

Deep Learning

Deep Learning is a sub-field of machine learning in Artificial intelligence (A.I.) that deals with algorithms inspired from the biological structure and functioning of a brain to aid machines with intelligence.

Tool Used: JUPYTER NOTEBOOK

Deep Neural Network to predict cancer as malignant or benign

Importing Dataset

In [ ]:

```
from sklearn.datasets import load_breast_cancer
dataset = load_breast_cancer()
```

In [ ]:

```
print(dataset.DESCR)
```

```
.. _breast_cancer_dataset:

Breast cancer wisconsin (diagnostic) dataset
--------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 569

    :Number of Attributes: 30 numeric, predictive attributes and the class

    :Attribute Information:
        - radius (mean of distances from center to points on the perimeter)
        - texture (standard deviation of gray-scale values)
        - perimeter
        - area
        - smoothness (local variation in radius lengths)
        - compactness (perimeter^2 / area - 1.0)
        - concavity (severity of concave portions of the contour)
        - concave points (number of concave portions of the contour)
        - symmetry
        - fractal dimension ("coastline approximation" - 1)

        The mean, standard error, and "worst" or largest (mean of the three
        largest values) of these features were computed for each image,
        resulting in 30 features.  For instance, field 3 is Mean Radius, field
        13 is Radius SE, field 23 is Worst Radius.

        - class:
                - WDBC-Malignant
                - WDBC-Benign

    :Summary Statistics:

    ===================================== ====== ======
                                           Min    Max
    ===================================== ====== ======
    radius (mean):                        6.981  28.11
    texture (mean):                       9.71   39.28
    perimeter (mean):                     43.79  188.5
    area (mean):                          143.5  2501.0
    smoothness (mean):                    0.053  0.163
    compactness (mean):                   0.019  0.345
    concavity (mean):                     0.0    0.427
    concave points (mean):                0.0    0.201
    symmetry (mean):                      0.106  0.304
    fractal dimension (mean):             0.05   0.097
    radius (standard error):              0.112  2.873
    texture (standard error):             0.36   4.885
    perimeter (standard error):           0.757  21.98
    area (standard error):                6.802  542.2
    smoothness (standard error):          0.002  0.031
    compactness (standard error):         0.002  0.135
    concavity (standard error):           0.0    0.396
    concave points (standard error):      0.0    0.053
```

```
    concave points (standard error):     0.0    0.053
    symmetry (standard error):           0.008  0.079
    fractal dimension (standard error):  0.001  0.03
    radius (worst):                      7.93   36.04
    texture (worst):                     12.02  49.54
    perimeter (worst):                   50.41  251.2
    area (worst):                        185.2  4254.0
    smoothness (worst):                  0.071  0.223
    compactness (worst):                 0.027  1.058
    concavity (worst):                   0.0    1.252
    concave points (worst):              0.0    0.291
    symmetry (worst):                    0.156  0.664
    fractal dimension (worst):           0.055  0.208
    ===================================== ====== ======

    :Missing Attribute Values: None

    :Class Distribution: 212 - Malignant, 357 - Benign

    :Creator:  Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian

    :Donor: Nick Street

    :Date: November, 1995

This is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.
https://goo.gl/U2Uwz2

Features are computed from a digitized image of a fine needle
aspirate (FNA) of a breast mass.  They describe
characteristics of the cell nuclei present in the image.

Separating plane described above was obtained using
Multisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree
Construction Via Linear Programming." Proceedings of the 4th
Midwest Artificial Intelligence and Cognitive Science Society,
pp. 97-101, 1992], a classification method which uses linear
programming to construct a decision tree.  Relevant features
were selected using an exhaustive search in the space of 1-4
features and 1-3 separating planes.

The actual linear program used to obtain the separating plane
in the 3-dimensional space is that described in:
[K. P. Bennett and O. L. Mangasarian: "Robust Linear
Programming Discrimination of Two Linearly Inseparable Sets",
Optimization Methods and Software 1, 1992, 23-34].

This database is also available through the UW CS ftp server:

ftp ftp.cs.wisc.edu
cd math-prog/cpo-dataset/machine-learn/WDBC/

.. topic:: References

   - W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extraction
     for breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on
     Electronic Imaging: Science and Technology, volume 1905, pages 861-870,
     San Jose, CA, 1993.
   - O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis and
     prognosis via linear programming. Operations Research, 43(4), pages 570-577,
     July-August 1995.
   - W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning techniques
     to diagnose breast cancer from fine-needle aspirates. Cancer Letters 77 (1994)
     163-171.
```

In [ ]:

```
features = dataset.data
target = dataset.target
```

In [ ]:

```
print(features.shape)
```

(569, 30)

In [ ]:

```
print(target.shape)
```

(569,)

Splitting dataset into training set and test set

In [ ]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size = 0.2)
```

In [ ]:

```
print(X_train.shape)
```

(455, 30)

In [ ]:

```
print(y_train.shape)
```

(455,)

In [ ]:

```
print(X_test.shape)
```

(114, 30)

In [ ]:

```
print(y_test.shape)
```

(114,)

In [ ]:

```
from keras.models import Sequential
from keras.layers import Dense
```

In [ ]:

```
model = Sequential()
model.add(Dense(32, input_dim = 30, activation = 'relu')) ## hidden layer 1
model.add(Dense(64, activation = 'relu')) ## hidden layer 2
model.add(Dense(1, activation = 'sigmoid'))
```

In [ ]:

```
model.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
```

In [ ]:

```
model.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_24 (Dense)             (None, 32)                992
_________________________________________________________________
dense_25 (Dense)             (None, 64)                2112
_________________________________________________________________
dense_26 (Dense)             (None, 1)                 65
=================================================================
```

```
Total params: 3,169
Trainable params: 3,169
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
model.fit(X_train, y_train, epochs = 10)
```

```
Epoch 1/10
15/15 [==============================] - 0s 1ms/step - loss: 0.0297 - accuracy: 0.9890
Epoch 2/10
15/15 [==============================] - 0s 1ms/step - loss: 0.0180 - accuracy: 0.9934
Epoch 3/10
15/15 [==============================] - 0s 1ms/step - loss: 0.0173 - accuracy: 0.9956
Epoch 4/10
15/15 [==============================] - 0s 1ms/step - loss: 0.0263 - accuracy: 0.9890
Epoch 5/10
15/15 [==============================] - 0s 1ms/step - loss: 0.0257 - accuracy: 0.9890
Epoch 6/10
15/15 [==============================] - 0s 1ms/step - loss: 0.0161 - accuracy: 0.9978
Epoch 7/10
15/15 [==============================] - 0s 2ms/step - loss: 0.0172 - accuracy: 0.9934
Epoch 8/10
15/15 [==============================] - 0s 1ms/step - loss: 0.0148 - accuracy: 0.9956
Epoch 9/10
15/15 [==============================] - 0s 1ms/step - loss: 0.0145 - accuracy: 0.9956
Epoch 10/10
15/15 [==============================] - 0s 2ms/step - loss: 0.0144 - accuracy: 0.9934
```

Out[ ]:

```
<keras.callbacks.History at 0x7fd5019442d0>
```

In [ ]:

```
scores = model.evaluate(X_test, y_test)
print(scores) ## returns loss and accuracy
```

```
4/4 [==============================] - 0s 2ms/step - loss: 0.4843 - accuracy: 0.9386
[0.48427480459213257, 0.9385964870452881]
```

In [ ]:

```
predictions = model.predict(X_test)
label = []
for pred in predictions:
  if pred>=0.5:
    print("Malignant")
  else:
    print("Benign")
```

```
Malignant
Malignant
Malignant
Malignant
Malignant
Malignant
Benign
Malignant
Malignant
Benign
Benign
Malignant
Malignant
Malignant
Benign
Malignant
Malignant
Malignant
Malignant
Benign
Benign
```

Benign
Malignant
Benign
Benign
Malignant
Malignant
Benign
Benign
Malignant
Malignant
Benign
Malignant
Benign
Malignant
Benign
Malignant
Benign
Malignant
Benign
Benign
Malignant
Malignant
Malignant
Malignant
Malignant
Benign
Malignant
Benign
Benign
Benign
Benign
Malignant
Malignant
Benign
Malignant
Benign
Malignant
Benign
Malignant
Benign
Benign
Malignant
Malignant
Benign
Malignant
Malignant
Malignant
Malignant
Malignant
Benign
Benign
Malignant
Benign
Malignant
Malignant
Malignant
Malignant
Malignant
Malignant
Malignant
Malignant
Malignant
Malignant
Malignant
Malignant
Malignant
Malignant
Benign
Malignant
Malignant
Benign
Benign
Malignant

```
Malignant
Benign
Malignant
Malignant
Benign
Benign
Benign
Malignant
Benign
Malignant
Benign
Malignant
Malignant
Malignant
Benign
Malignant
Benign
Malignant
Benign
Benign
Benign
Malignant
```

RESULT

A Deep Neural Network was successfully developed to determine cancer as malignant or benign.

In [ ]:

```

```