THAKUR TRUSTS

Thakur Educational Trust's (Regd.)

THAKUR COLLEGE OF SCIENCE & COMMERCE

AUTONOMOUS COLLEGE, PERMANENTLY AFFILIATED TO UNIVERSITY OF MUMBAI

NAAC Accredited Grade 'A' (3rd Cycle) & ISO 9001: 2015 (Certified)

Azadi Ka Amrit Mahotsav Best College Award by University of Mumbai for the Year 2018-2019

tcsc CELEBRATING 25 YEARS OF GLORY

M.Sc C.S - II
SEM IV
Journal

Roll No.	4733
Name	Neeraj Venkatsai Laxminarayanrao Appari
Subject	Artificial Intelligence

Thakur Educational Trust's (Regd.)

THAKUR COLLEGE OF SCIENCE & COMMERCE

AUTONOMOUS COLLEGE, PERMANENTLY AFFILIATED TO UNIVERSITY OF MUMBAI

NAAC Accredited Grade 'A' (3rd Cycle) & ISO 9001: 2015 (Certified)

Best College Award by University of Mumbai for the Year 2018-2019

CERTIFICATE

This is here to certify that **Ms. <u>Neeraj Venkatsai Laxminarayanrao Appari</u>** Seat Number **<u>4733</u>** of M.Sc-II Computer Science, has satisfactorily completed the required number of Practicals prescribed by Thakur College Of Science And Commerce during the academic year 2023 – 2024.

Date:
Place: Kandivali,Mumbai

Teacher In-Charge **Head of Department**

External Examiner

INDEX

Practical No.: 1

Aim: Implement Breadth first search algorithm for Romanian map problem.

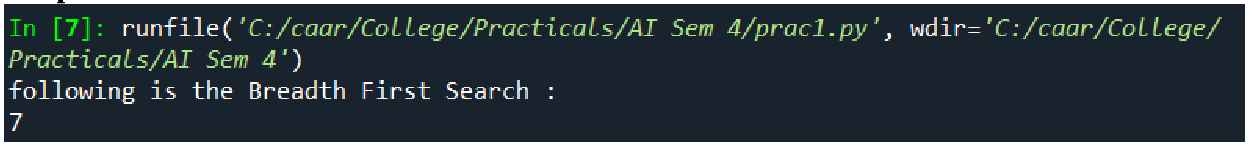
Theory:

Breadth first search is a graph traversal algorithm that starts traversing the graph from root node and explores all the neighboring nodes. Then, it selects the nearest node and explore all the unexplored nodes. The algorithm follows the same process for each of the nearest node until it finds the goal. The algorithm of breadth first search is given below. The algorithm starts with examining the node A and all of its neighbors. In the next step, the neighbors of the nearest node of A are explored and process continues in the further steps. The algorithm explores all neighbors of all the nodes and ensures that each node is visited exactly once and no node is visited twice

Code:

```
graph= {
   '5' : ['3','7'],
   '3' : ['2','4'],
   '7' : ['8'],
   '2' : [],
   '4' : ['8'],
   '8' : []
}
visited = []
queue = []
def bfs(visited, graph, node):
   visited.append(node)
   queue.append(node)
   while queue:
      m = queue.pop(0)
      print(m, end=" ")
      for neighbour in graph[m]:
         if neighbour not in visited:
            visited.append(neighbour)
            queue.append(neighbour)
# drive code
print("following is the Breadth First Search : ")
bfs(visited, graph, '7')
```

Output:



```
In [7]: runfile('C:/caar/College/Practicals/AI Sem 4/prac1.py', wdir='C:/caar/College/Practicals/AI Sem 4')
following is the Breadth First Search :
7
```

Practical No.:2

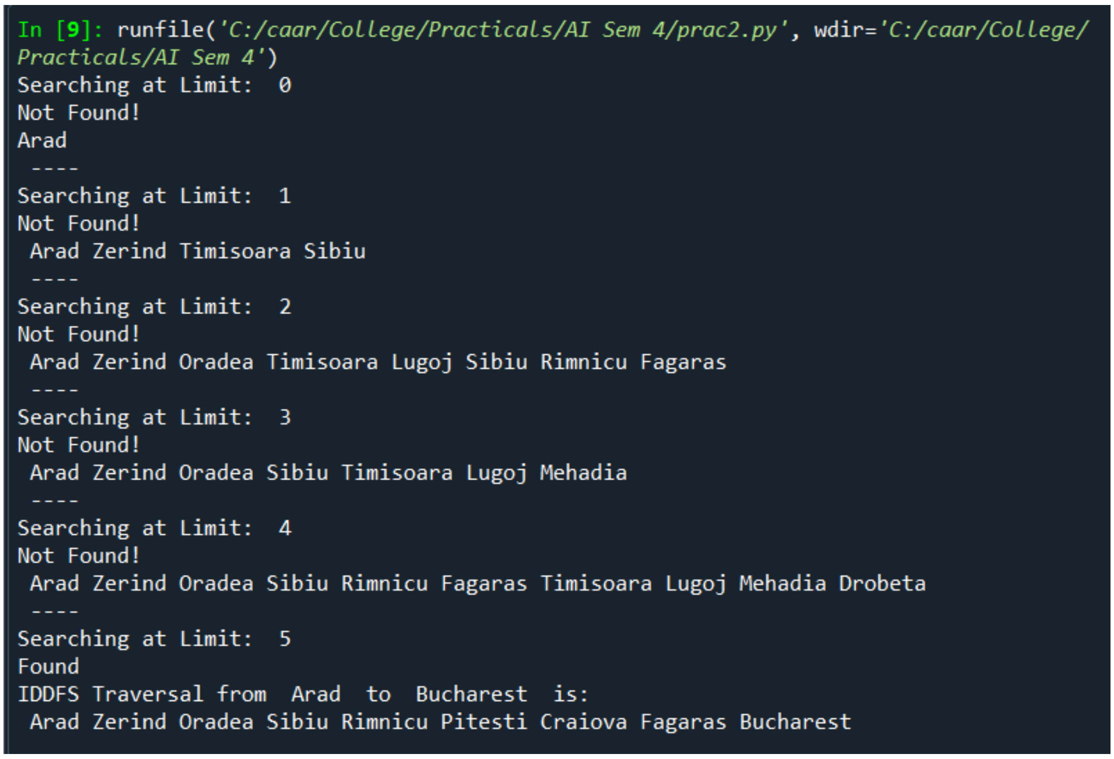
Aim: Implement Iterative deep depth first search for Romanian map problem.

Theory:

Depth-first search is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking. So the basic idea is to start from the root or any arbitrary node and mark the node and move to the adjacent unmarked node and continue this loop until there is no unmarked adjacent node. Then backtrack and check for other unmarked nodes and traverse them. Finally, print the nodes in the path.

Code:

```
dict_hn = {
    'Arad': 336, 'Bucharest': 0, 'Craiova': 160, 'Drobeta': 242, 'Eforie': 161,
    'Fagaras': 176, 'Giurgiu': 77, 'Hirsova': 151, 'Iasi': 226, 'Lugoj': 244,
    'Mehadia': 241, 'Neamt': 234, 'Oradea': 380, 'Pitesti': 100, 'Rimnicu': 193,
    'Sibiu': 253, 'Timisoara': 329, 'Urziceni': 80, 'Vaslui': 199, 'Zerind': 374
}
dict_gn = {
    'Arad': {'Zerind': 75, 'Timisoara': 118, 'Sibiu': 140},
    'Bucharest': {'Urziceni': 85, 'Giurgiu': 90, 'Pitesti': 101, 'Fagaras': 211},
    'Craiova': {'Drobeta': 120, 'Pitesti': 138, 'Rimnicu': 146},
    'Drobeta': {'Mehadia': 75, 'Craiova': 120},
    'Eforie': {'Hirsova': 86},
    'Fagaras': {'Sibiu': 99, 'Bucharest': 211},
    'Giurgiu': {'Bucharest': 90},
    'Hirsova': {'Eforie': 86, 'Urziceni': 98},
    'Iasi': {'Neamt': 87, 'Vaslui': 92},
    'Lugoj': {'Mehadia': 70, 'Timisoara': 111},
    'Mehadia': {'Lugoj': 70, 'Drobeta': 75},
    'Neamt': {'Iasi': 87},
    'Oradea': {'Zerind': 71, 'Sibiu': 151},
    'Pitesti': {'Rimnicu': 97, 'Bucharest': 101, 'Craiova': 138},
    'Rimnicu': {'Sibiu': 80, 'Pitesti': 97, 'Craiova': 146},
    'Sibiu': {'Rimnicu': 80, 'Fagaras': 99, 'Arad': 140, 'Oradea': 151},
    'Timisoara': {'Lugoj': 111, 'Arad': 118},
    'Urziceni': {'Bucharest': 85, 'Hirsova': 98, 'Vaslui': 142},
    'Vaslui': {'Iasi': 92, 'Urziceni': 142},
    'Zerind': {'Oradea': 71, 'Arad': 75}
}
def DLS(city, visitedstack, startlimit, endlimit):
    global result
    found = 0
    result = result + city + ' '
    visitedstack.append(city)
    if city == goal:
        return 1
    if startlimit == endlimit:
            return 0
        for eachcity in dict_gn[city].keys():
            if eachcity not in visitedstack:
                found = DLS(eachcity, visitedstack, startlimit + 1, endlimit)
                if found:
                    return found

def IDDFS(city, visitedstack, endlimit):
    global result
    for i in range(0, endlimit):
        print("Searching at Limit: ", i)
        found = DLS(city, visitedstack, 0, i)
        if found:
            print("Found")
            break
        else:
            print("Not Found! ")
        print(result)
        print(" ---- ")
        result = ' '
        visitedstack = []
start = 'Arad'
goal = 'Bucharest'
result = ''
visitedstack = []
IDDFS(start, visitedstack, 9)
print("IDDFS Traversal from ", start, " to ", goal, " is: ")
print(result)
```
Output:



```
In [9]: runfile('C:/caar/College/Practicals/AI Sem 4/prac2.py', wdir='C:/caar/College/
Practicals/AI Sem 4')
Searching at Limit:  0
Not Found!
Arad
 ----
Searching at Limit:  1
Not Found!
 Arad Zerind Timisoara Sibiu
 ----
Searching at Limit:  2
Not Found!
 Arad Zerind Oradea Timisoara Lugoj Sibiu Rimnicu Fagaras
 ----
Searching at Limit:  3
Not Found!
 Arad Zerind Oradea Sibiu Timisoara Lugoj Mehadia
 ----
Searching at Limit:  4
Not Found!
 Arad Zerind Oradea Sibiu Rimnicu Fagaras Timisoara Lugoj Mehadia Drobeta
 ----
Searching at Limit:  5
Found
IDDFS Traversal from  Arad  to  Bucharest  is:
 Arad Zerind Oradea Sibiu Rimnicu Pitesti Craiova Fagaras Bucharest
```

Practical No.:3

Aim: Implement A* search algorithm for Romanian map problem.

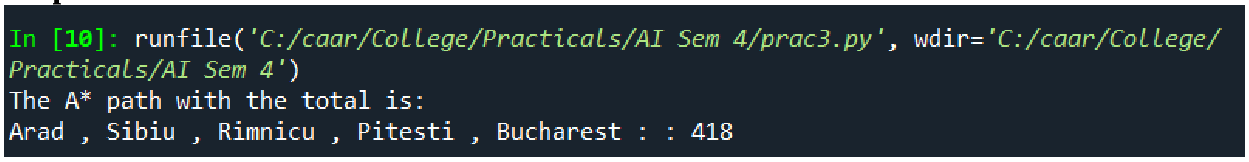
Theory:
A* is formulated with weighted graphs, which means it can find the best path involving the smallest cost in terms of distance and time. This makes A* algorithm in artificial intelligence an informed search algorithm for best-first search. The most important advantage of A* the search algorithm which separates it from other traversal techniques is that it has a brain.

Code:

```
import queue as Q

dict_hn = {
   'Arad': 336, 'Bucharest': 0, 'Craiova': 160, 'Drobeta': 242, 'Eforie': 161,
   'Fagaras': 176, 'Giurgiu': 77, 'Hirsova': 151, 'Iasi': 226, 'Lugoj': 244,
   'Mehadia': 241, 'Neamt': 234, 'Oradea': 380, 'Pitesti': 100, 'Rimnicu': 193,
   'Sibiu': 253, 'Timisoara': 329, 'Urziceni': 80, 'Vaslui': 199, 'Zerind': 374
}

dict_gn = {
   'Arad': {'Zerind': 75, 'Timisoara': 118, 'Sibiu': 140},
   'Bucharest': {'Urziceni': 85, 'Giurgiu': 90, 'Pitesti': 101, 'Fagaras': 211},
   'Craiova': {'Drobeta': 120, 'Pitesti': 138, 'Rimnicu': 146},
   'Drobeta': {'Mehadia': 75, 'Craiova': 120},
   'Eforie': {'Hirsova': 86},
   'Fagaras': {'Sibiu': 99, 'Bucharest': 211},
   'Giurgiu': {'Bucharest': 90},
   'Hirsova': {'Eforie': 86, 'Urziceni': 98},
   'Iasi': {'Neamt': 87, 'Vaslui': 92},
   'Lugoj': {'Mehadia': 70, 'Timisoara': 111},
   'Mehadia': {'Lugoj': 70, 'Drobeta': 75},
   'Neamt': {'Iasi': 87},
   'Oradea': {'Zerind': 71, 'Sibiu': 151},
   'Pitesti': {'Rimnicu': 97, 'Bucharest': 101, 'Craiova': 138},
   'Rimnicu': {'Sibiu': 80, 'Pitesti': 97, 'Craiova': 146},
   'Sibiu': {'Rimnicu': 80, 'Fagaras': 99, 'Arad': 140, 'Oradea': 151},
   'Timisoara': {'Lugoj': 111, 'Arad': 118},
   'Urziceni': {'Bucharest': 85, 'Hirsova': 98, 'Vaslui': 142},
   'Vaslui': {'Iasi': 92, 'Urziceni': 142},
   'Zerind': {'Oradea': 71, 'Arad': 75}
}

start = 'Arad'
goal = 'Bucharest'
result = ''

def get_fn(citystr):
   cities = citystr.split(" , ")
   gn = 0
   hn = dict_hn[cities[-1]]
```

```
    for ctr in range(0, len(cities)-1):
        gn = gn + dict_gn[cities[ctr]][cities[ctr+1]]
    return (hn + gn)

def expand(cityq):
    global result
    while not cityq.empty():
        tot, citystr, thiscity = cityq.get()
        if thiscity == goal:
            result = citystr + " : : " + str(tot)
            break
        for cty in dict_gn[thiscity]:
            cityq.put((get_fn(citystr + " , " + cty), citystr + " , " + cty, cty))

def main():
    cityq = Q.PriorityQueue()
    cityq.put((get_fn(start), start, start))
    expand(cityq)
    print("The A* path with the total is: ")
    print(result)

main()
```

Output:



```
In [10]: runfile('C:/caar/College/Practicals/AI Sem 4/prac3.py', wdir='C:/caar/College/
Practicals/AI Sem 4')
The A* path with the total is:
Arad , Sibiu , Rimnicu , Pitesti , Bucharest : : 418
```

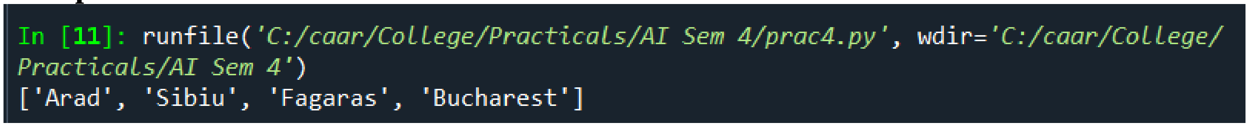
Practical No.:04

Aim: Implement recursive best-first search algorithm for Romanian map problem

Theory:
Recursive Best-First Search (RBFS) is a recursive algorithm for finding the shortest path in a graph. It combines the space-saving technique of the depth-first search with the completeness of the breadth-first search.

Code:
```
from queue import Queue

romaniaMap = {
    'Arad': ['Sibiu', 'Zerind', 'Timisoara'],
    'Zerind': ['Arad', 'Oradea'],
    'Oradea': ['Zerind', 'Sibiu'],
    'Sibiu': ['Arad', 'Oradea', 'Fagaras', 'Rimnicu'],
    'Timisoara': ['Arad', 'Lugoj'],
    'Lugoj': ['Timisoara', 'Mehadia'],
    'Mehadia': ['Lugoj', 'Drobeta'],
    'Drobeta': ['Mehadia', 'Craiova'],
    'Craiova': ['Drobeta', 'Rimnicu', 'Pitesti'],
    'Rimnicu': ['Sibiu', 'Craiova', 'Pitesti'],
    'Fagaras': ['Sibiu', 'Bucharest'],
    'Pitesti': ['Rimnicu', 'Craiova', 'Bucharest'],
    'Bucharest': ['Fagaras', 'Pitesti', 'Giurgiu', 'Urziceni'],
    'Giurgiu': ['Bucharest'],
    'Urziceni': ['Bucharest', 'Vaslui', 'Hirsova'],
    'Hirsova': ['Urziceni', 'Eforie'],
    'Eforie': ['Hirsova'],
    'Vaslui': ['Iasi', 'Urziceni'],
    'Iasi': ['Vaslui', 'Neamt'],
    'Neamt': ['Iasi']
}

def bfs(startingNode, destinationNode):
    # For keeping track of what we have visited
    visited = {}
    # keep track of distance
    distance = {}
    # parent node of specific graph
    parent = {}
    # BFS is queue based so using 'Queue' from python built-in
    queue = Queue()

    # initialise visited, distance, and parent for each city
    for city in romaniaMap.keys():
        visited[city] = False
        parent[city] = None
        distance[city] = -1
    # starting from 'Arad'
    startingCity = startingNode
    visited[startingCity] = True
    distance[startingCity] = 0
    queue.put(startingCity)

    while not queue.empty():
        u = queue.get()
        # explore the adjacent cities to 'u'
        for v in romaniaMap[u]:
            if not visited[v]:
                visited[v] = True
                parent[v] = u
                distance[v] = distance[u] + 1
                queue.put(v)

    # reconstruct path from startingNode to destinationNode
    path = []
    g = destinationNode
    while g is not None:
        path.append(g)
        g = parent[g]
    path.reverse()

    # printing the path to our destination city
    print(path)

# Starting City & Destination City
bfs('Arad', 'Bucharest')
```

Output:



```
In [11]: runfile('C:/caar/College/Practicals/AI Sem 4/prac4.py', wdir='C:/caar/College/Practicals/AI Sem 4')
['Arad', 'Sibiu', 'Fagaras', 'Bucharest']
```

Practical No.:5

Aim: Write a program to implement the general structure and working of the Genetic Algorithm

Theory:

Genetic Algorithm (GA): is a nature-inspired algorithm that has extensively been used to solve optimization problems. It belongs to the branch of approximation algorithms because it does not guarantee to always find the exact optimal solution; however, it may find a near-optimal solution in a limited time.

Chromosome: A chromosome is a candidate/potential solution to the problem being solved. The solution may not necessarily be the optimal solution. A chromosome may be considered as an array of binary/integer variables where each variable may be considered as a gene and the value of a variable is called an allele.

Population: A set of solutions participating in the process of optimization is called population. Unlike trajectory-based algorithms, GA is a population-based algorithm in which several solutions interact with each other to find the global optimum.

Objective function: It is also called a fitness function. Whenever an optimization problem is solved it is first formulated as a mathematical function that evaluates the quality/fitness of the candidate solution.

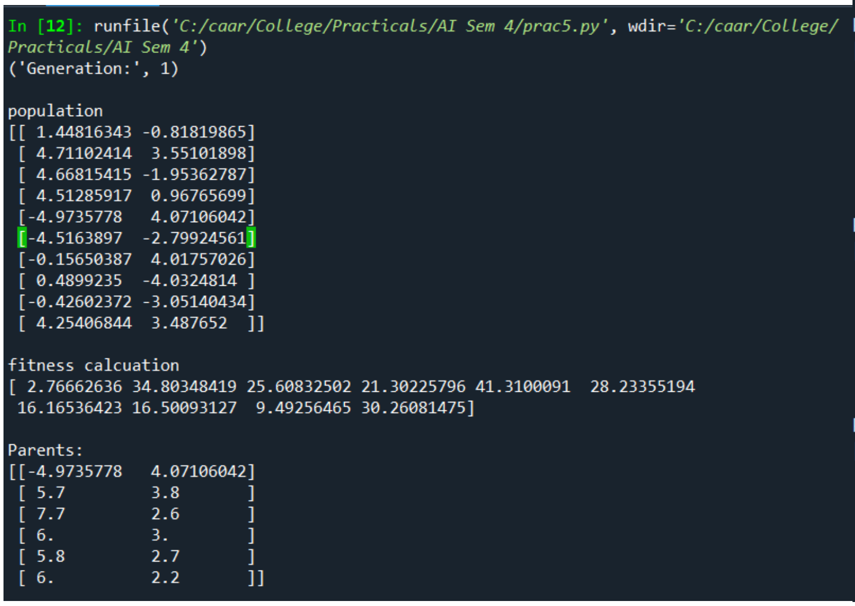
Code:

```
import numpy

# Parameter initialization

genes = 2

chromosomes = 10

mattingPoolSize = 6

offspringSize = chromosomes - mattingPoolSize

lb = -5

ub = 5

populationSize = (chromosomes, genes)

generations = 3

#Population initialization

population = numpy.random.uniform(lb, ub, populationSize)

for generation in range(generations):

    print(("Generation:", generation+1))

    fitness = numpy.sum(population*population, axis=1)
```

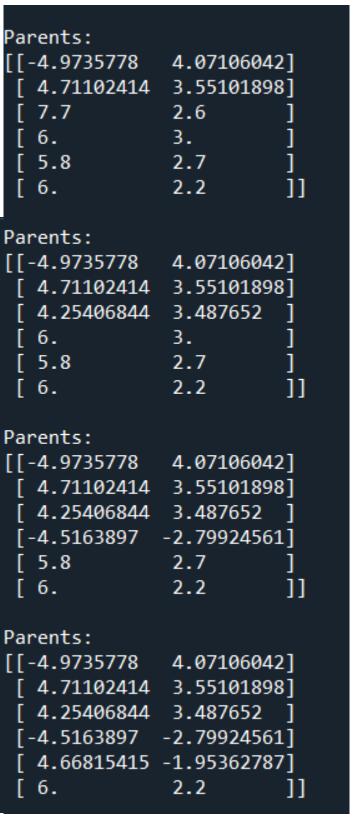
```
print("\npopulation")
print(population)
print("\nfitness calcuation")
print(fitness)
# Following statement will create an empty two dimensional array to store parents
parents = numpy.empty((mattingPoolSize, population.shape[1]))

# A loop to extract one parent in each iteration
for p in range(mattingPoolSize):
    # Finding index of fittest chromosome in the population
    fittestIndex = numpy.where(fitness == numpy.max(fitness))
    # Extracting index of fittest chromosome
    fittestIndex = fittestIndex[0][0]
    # Copying fittest chromosome into parents array
    parents[p, :] = population[fittestIndex, :]
    # Changing fitness of fittest chromosome to avoid reselection of that chromosome
    fitness[fittestIndex] = -1
print("\nParents:")
print(parents)
# Following statement will create an empty two dimensional array to store offspring
offspring = numpy.empty((offspringSize, population.shape[1]))
for k in range(offspringSize):
    #Determining the crossover point
    crossoverPoint = numpy.random.randint(0,genes)
    # Index of the first parent.
    parent1Index = k%parents.shape[0]
    # Index of the second.
    parent2Index = (k+1)%parents.shape[0]
    # Extracting first half of the offspring
    offspring[k, 0: crossoverPoint] = parents[parent1Index, 0: crossoverPoint]
```

```
        # Extracting second half of the offspring
        offspring[k, crossoverPoint:] = parents[parent2Index, crossoverPoint:]
    print("\nOffspring after crossover:")
    print(offspring)
    # Implementation of random initialization mutation.
    for index in range(offspring.shape[0]):
        randomIndex = numpy.random.randint(1,genes)
        randomValue = numpy.random.uniform(lb, ub, 1)
        offspring [index, randomIndex] = offspring [index, randomIndex] + randomValue
    print("\n Offspring after Mutation")
    print(offspring)
    population[0:parents.shape[0], :] = parents
    population[parents.shape[0]:, :] = offspring

    print("\nNew Population for next generation:")
    print(population)
fitness = numpy.sum(population*population, axis=1)
fittestIndex = numpy.where(fitness == numpy.max(fitness))
# Extracting index of fittest chromosome
fittestIndex = fittestIndex[0][0]
# Getting Best chromosome
fittestInd = population[fittestIndex, :]
bestFitness = fitness[fittestIndex]
print("\nBest Individual:")
print(fittestInd)
print("\nBest Individual's Fitness:")
print(bestFitness)
```

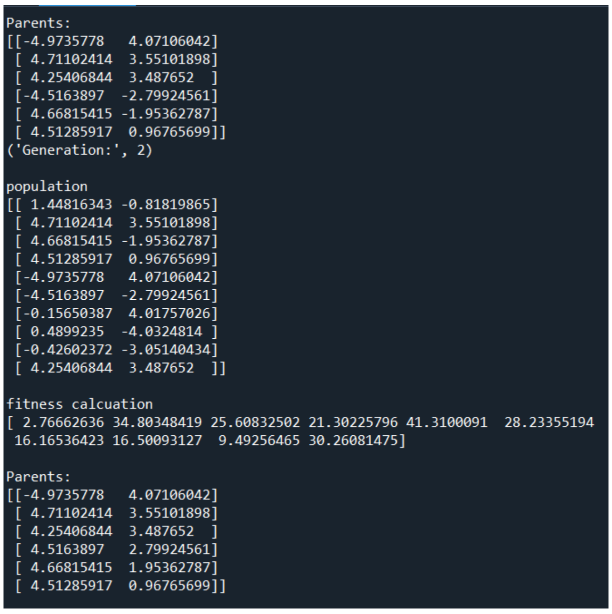
Output:



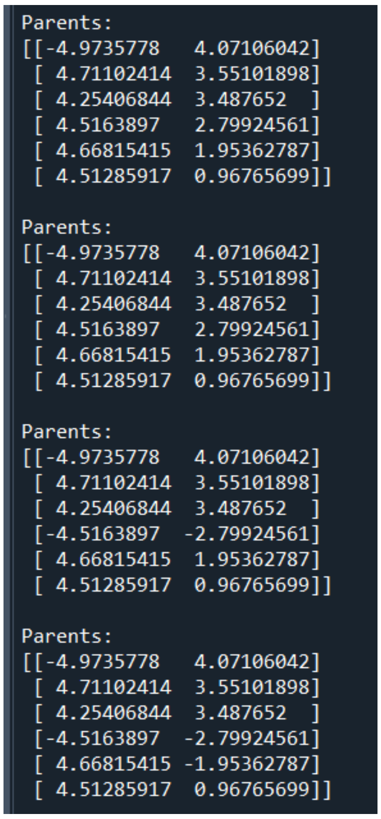
```
In [12]: runfile('C:/caar/College/Practicals/AI Sem 4/prac5.py', wdir='C:/caar/College/
Practicals/AI Sem 4')
('Generation:', 1)

population
[[ 1.44816343 -0.81819865]
 [ 4.71102414  3.55101898]
 [ 4.66815415 -1.95362787]
 [ 4.51285917  0.96765699]
 [-4.9735778   4.07106042]
 [-4.5163897  -2.79924561]
 [-0.15650387  4.01757026]
 [ 0.4899235  -4.0324814 ]
 [-0.42602372 -3.05140434]
 [ 4.25406844  3.487652  ]]

fitness calcuation
[ 2.76662636 34.80348419 25.60832502 21.30225796 41.3100091  28.23355194
 16.16536423 16.50093127  9.49256465 30.26081475]

Parents:
[[-4.9735778   4.07106042]
 [ 5.7         3.8       ]
 [ 7.7         2.6       ]
 [ 6.          3.        ]
 [ 5.8         2.7       ]
 [ 6.          2.2       ]]
```



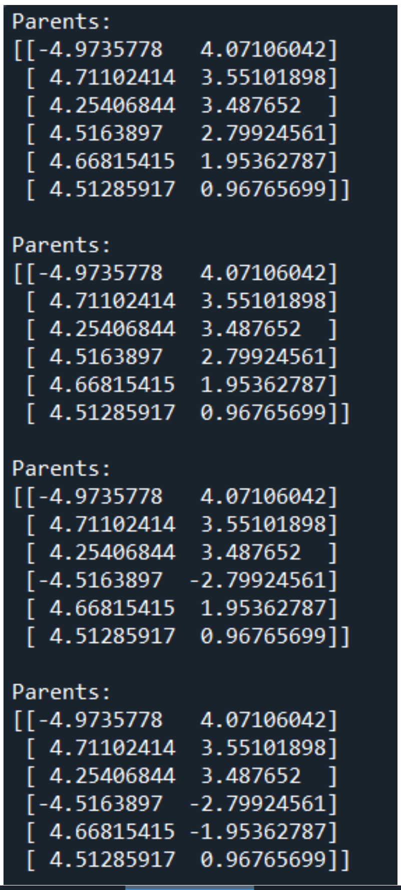
```
Parents:
[[-4.9735778   4.07106042]
 [ 4.71102414  3.55101898]
 [ 7.7         2.6       ]
 [ 6.          3.        ]
 [ 5.8         2.7       ]
 [ 6.          2.2       ]]
Parents:
[[-4.9735778   4.07106042]
 [ 4.71102414  3.55101898]
 [ 4.25406844  3.487652  ]
 [ 6.          3.        ]
 [ 5.8         2.7       ]
 [ 6.          2.2       ]]

Parents:
[[-4.9735778   4.07106042]
 [ 4.71102414  3.55101898]
 [ 4.25406844  3.487652  ]
 [-4.5163897  -2.79924561]
 [ 5.8         2.7       ]
 [ 6.          2.2       ]]

Parents:
[[-4.9735778   4.07106042]
 [ 4.71102414  3.55101898]
 [ 4.25406844  3.487652  ]
 [-4.5163897  -2.79924561]
 [ 4.66815415 -1.95362787]
 [ 6.          2.2       ]]
```



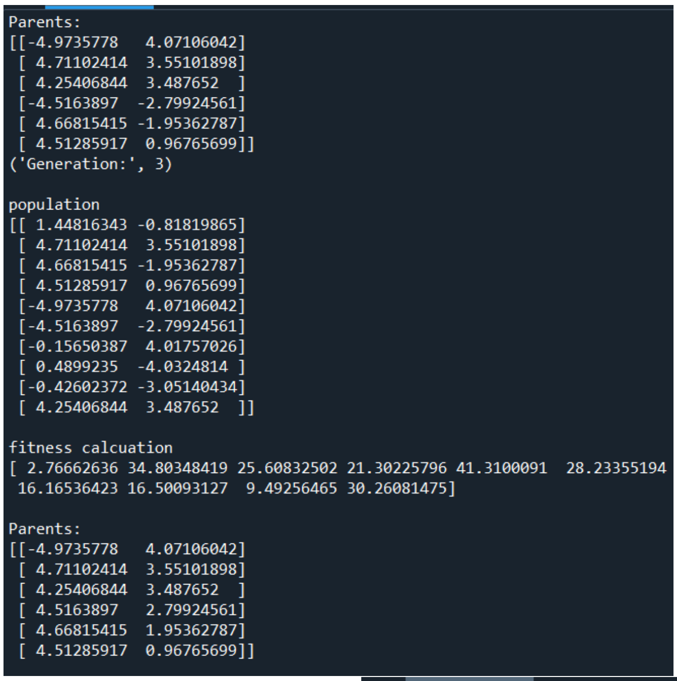
```
Parents:
[[-4.9735778   4.07106042]
 [ 4.71102414  3.55101898]
 [ 4.25406844  3.487652  ]
 [-4.5163897  -2.79924561]
 [ 4.66815415 -1.95362787]
 [ 4.51285917  0.96765699]]
('Generation:', 2)

population
[[ 1.44816343 -0.81819865]
 [ 4.71102414  3.55101898]
 [ 4.66815415 -1.95362787]
 [ 4.51285917  0.96765699]
 [-4.9735778   4.07106042]
 [-4.5163897  -2.79924561]
 [-0.15650387  4.01757026]
 [ 0.4899235  -4.0324814 ]
 [-0.42602372 -3.05140434]
 [ 4.25406844  3.487652  ]]

fitness calcuation
[ 2.76662636 34.80348419 25.60832502 21.30225796 41.3100091  28.23355194
 16.16536423 16.50093127  9.49256465 30.26081475]

Parents:
[[-4.9735778   4.07106042]
 [ 4.71102414  3.55101898]
 [ 4.25406844  3.487652  ]
 [ 4.5163897   2.79924561]
 [ 4.66815415  1.95362787]
 [ 4.51285917  0.96765699]]
```



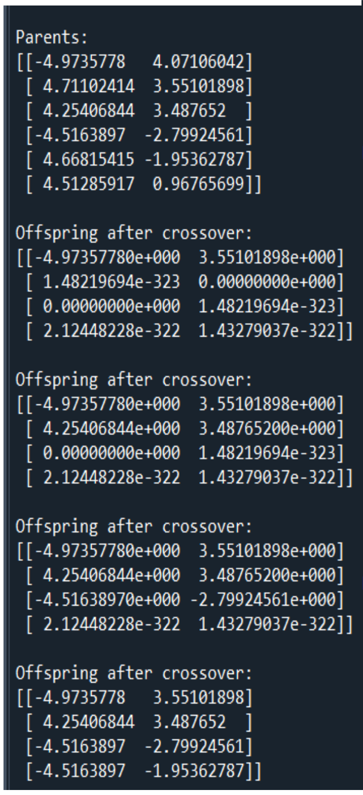
```
Parents:
[[-4.9735778   4.07106042]
 [ 4.71102414  3.55101898]
 [ 4.25406844  3.487652  ]
 [ 4.5163897   2.79924561]
 [ 4.66815415  1.95362787]
 [ 4.51285917  0.96765699]]

Parents:
[[-4.9735778   4.07106042]
 [ 4.71102414  3.55101898]
 [ 4.25406844  3.487652  ]
 [ 4.5163897   2.79924561]
 [ 4.66815415  1.95362787]
 [ 4.51285917  0.96765699]]

Parents:
[[-4.9735778   4.07106042]
 [ 4.71102414  3.55101898]
 [ 4.25406844  3.487652  ]
 [-4.5163897  -2.79924561]
 [ 4.66815415  1.95362787]
 [ 4.51285917  0.96765699]]

Parents:
[[-4.9735778   4.07106042]
 [ 4.71102414  3.55101898]
 [ 4.25406844  3.487652  ]
 [-4.5163897  -2.79924561]
 [ 4.66815415 -1.95362787]
 [ 4.51285917  0.96765699]]
```



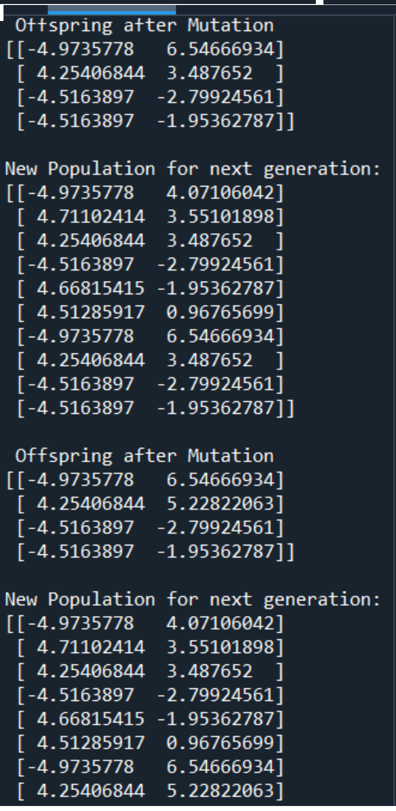
```
Parents:
[[-4.9735778   4.07106042]
 [ 4.71102414  3.55101898]
 [ 4.25406844  3.487652  ]
 [ 4.5163897   2.79924561]
 [ 4.66815415  1.95362787]
 [ 4.51285917  0.96765699]]

Parents:
[[-4.9735778   4.07106042]
 [ 4.71102414  3.55101898]
 [ 4.25406844  3.487652  ]
 [ 4.5163897   2.79924561]
 [ 4.66815415  1.95362787]
 [ 4.51285917  0.96765699]]

Parents:
[[-4.9735778   4.07106042]
 [ 4.71102414  3.55101898]
 [ 4.25406844  3.487652  ]
 [-4.5163897  -2.79924561]
 [ 4.66815415  1.95362787]
 [ 4.51285917  0.96765699]]

Parents:
[[-4.9735778   4.07106042]
 [ 4.71102414  3.55101898]
 [ 4.25406844  3.487652  ]
 [-4.5163897  -2.79924561]
 [ 4.66815415 -1.95362787]
 [ 4.51285917  0.96765699]]
```



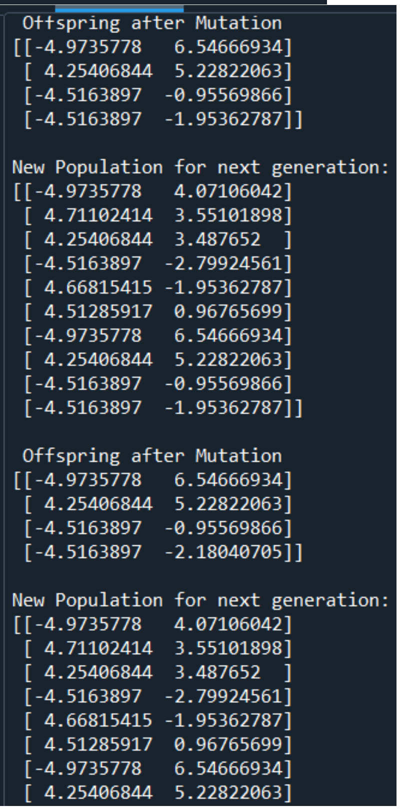
```
Parents:
[[-4.9735778   4.07106042]
 [ 4.71102414  3.55101898]
 [ 4.25406844  3.487652  ]
 [-4.5163897  -2.79924561]
 [ 4.66815415 -1.95362787]
 [ 4.51285917  0.96765699]]
('Generation:', 3)

population
[[ 1.44816343 -0.81819865]
 [ 4.71102414  3.55101898]
 [ 4.66815415 -1.95362787]
 [ 4.51285917  0.96765699]
 [-4.9735778   4.07106042]
 [-4.5163897  -2.79924561]
 [-0.15650387  4.01757026]
 [ 0.4899235  -4.0324814 ]
 [-0.42602372 -3.05140434]
 [ 4.25406844  3.487652  ]]

fitness calcuation
[ 2.76662636 34.80348419 25.60832502 21.30225796 41.3100091  28.23355194
 16.16536423 16.50093127  9.49256465 30.26081475]

Parents:
[[-4.9735778   4.07106042]
 [ 4.71102414  3.55101898]
 [ 4.25406844  3.487652  ]
 [ 4.5163897   2.79924561]
 [ 4.66815415  1.95362787]
 [ 4.51285917  0.96765699]]
```



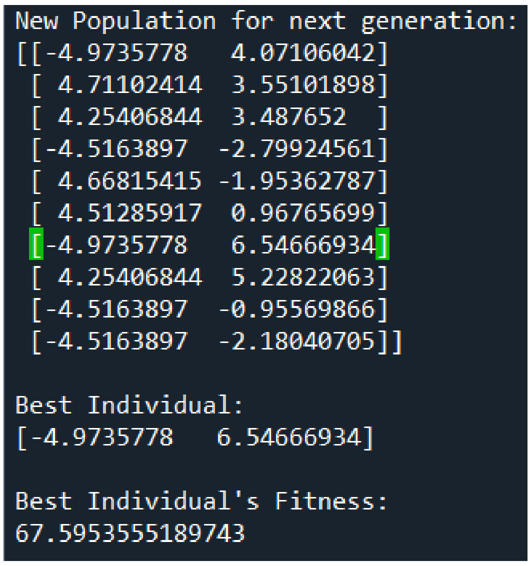
```
Parents:
[[-4.9735778   4.07106042]
 [ 4.71102414  3.55101898]
 [ 4.25406844  3.487652  ]
 [-4.5163897  -2.79924561]
 [ 4.66815415 -1.95362787]
 [ 4.51285917  0.96765699]]

Offspring after crossover:
[[-4.97357780e+000  3.55101898e+000]
 [ 1.48219694e-323  0.00000000e+000]
 [ 0.00000000e+000  1.48219694e-323]
 [ 2.12448228e-322  1.43279037e-322]]

Offspring after crossover:
[[-4.97357780e+000  3.55101898e+000]
 [ 4.25406844e+000  3.48765200e+000]
 [ 0.00000000e+000  1.48219694e-323]
 [ 2.12448228e-322  1.43279037e-322]]

Offspring after crossover:
[[-4.97357780e+000  3.55101898e+000]
 [ 4.25406844e+000  3.48765200e+000]
 [-4.51638970e+000 -2.79924561e+000]
 [ 2.12448228e-322  1.43279037e-322]]

Offspring after crossover:
[[-4.9735778   3.55101898]
 [ 4.25406844  3.487652  ]
 [-4.5163897  -2.79924561]
 [-4.5163897  -1.95362787]]
```



```
 Offspring after Mutation
[[-4.9735778   6.54666934]
 [ 4.25406844  3.487652  ]
 [-4.5163897  -2.79924561]
 [-4.5163897  -1.95362787]]

New Population for next generation:
[[-4.9735778   4.07106042]
 [ 4.71102414  3.55101898]
 [ 4.25406844  3.487652  ]
 [-4.5163897  -2.79924561]
 [ 4.66815415 -1.95362787]
 [ 4.51285917  0.96765699]
 [-4.9735778   6.54666934]
 [ 4.25406844  3.487652  ]
 [-4.5163897  -2.79924561]
 [-4.5163897  -1.95362787]]

 Offspring after Mutation
[[-4.9735778   6.54666934]
 [ 4.25406844  5.22822063]
 [-4.5163897  -2.79924561]
 [-4.5163897  -1.95362787]]

New Population for next generation:
[[-4.9735778   4.07106042]
 [ 4.71102414  3.55101898]
 [ 4.25406844  3.487652  ]
 [-4.5163897  -2.79924561]
 [ 4.66815415 -1.95362787]
 [ 4.51285917  0.96765699]
 [-4.9735778   6.54666934]
 [ 4.25406844  5.22822063]
```



```
 Offspring after Mutation
[[-4.9735778   6.54666934]
 [ 4.25406844  5.22822063]
 [-4.5163897  -0.95569866]
 [-4.5163897  -1.95362787]]

New Population for next generation:
[[-4.9735778   4.07106042]
 [ 4.71102414  3.55101898]
 [ 4.25406844  3.487652  ]
 [-4.5163897  -2.79924561]
 [ 4.66815415 -1.95362787]
 [ 4.51285917  0.96765699]
 [-4.9735778   6.54666934]
 [ 4.25406844  5.22822063]
 [-4.5163897  -0.95569866]
 [-4.5163897  -1.95362787]]

 Offspring after Mutation
[[-4.9735778   6.54666934]
 [ 4.25406844  5.22822063]
 [-4.5163897  -0.95569866]
 [-4.5163897  -2.18040705]]

New Population for next generation:
[[-4.9735778   4.07106042]
 [ 4.71102414  3.55101898]
 [ 4.25406844  3.487652  ]
 [-4.5163897  -2.79924561]
 [ 4.66815415 -1.95362787]
 [ 4.51285917  0.96765699]
 [-4.9735778   6.54666934]
 [ 4.25406844  5.22822063]
```



```
New Population for next generation:
[[-4.9735778   4.07106042]
 [ 4.71102414  3.55101898]
 [ 4.25406844  3.487652  ]
 [-4.5163897  -2.79924561]
 [ 4.66815415 -1.95362787]
 [ 4.51285917  0.96765699]
 [-4.9735778   6.54666934]
 [ 4.25406844  5.22822063]
 [-4.5163897  -0.95569866]
 [-4.5163897  -2.18040705]]

Best Individual:
[-4.9735778   6.54666934]

Best Individual's Fitness:
67.5953555189743
```

Practical No.:6

Aim: Implement the Perceptron Algorithm

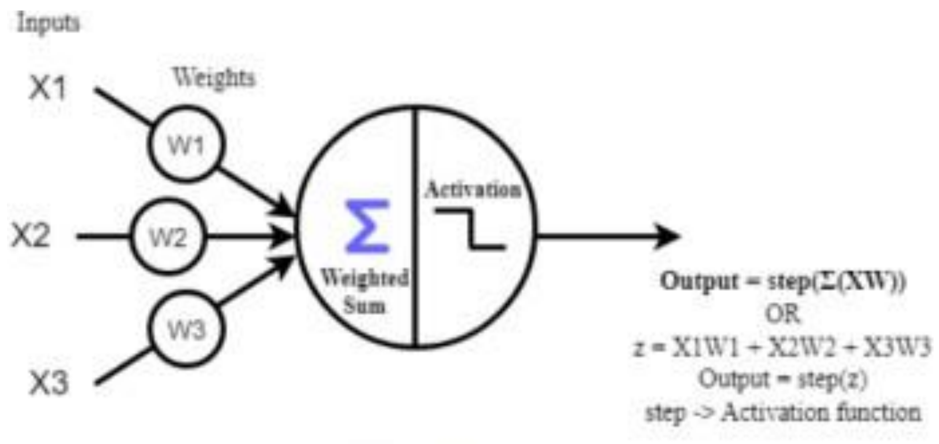
Theory:

The perceptron algorithm is based on the concept of a single neuron in the human brain, A single neuron in the human brain is doing a very simple thing, just receiving some inputs and if the inputs are high, it activates and send the signal to the next neuron. The perceptron was designed to mimic this process, with the input data serving as the input to the neuron and the weights representing the strength of the connections between the input neurons and the output neuron.

The Perceptron is a type of linear classifier(binary classifier), which means it can be used to classify data that is linearly separable.

A Perceptron is somehow similar to Logistic Regression at first glance, but it's different. While Logistic Regression predicts probabilities of a data point falling in a particular class, Perceptron will only tell whether the data point is in a particular class or not, Just like saying "Yes" or "No".

Here is the diagrammatic representation of the perceptron algorithm.



Perceptron(TLU) diagram

A Perceptron is a kind of single Artificial Neuron which is also known as a Threshold Processing Unit(TLU). As you can see in the above diagram, the Perceptron contains some input links X1, X2, and, X3. Each input has its own corresponding weights W1, W2, and W3. These weights are basically the hearts of Perceptrons which determine the strength of each input signal to it.

The Perceptron or TLU computes the weighted sum of the inputs (z = X1W1 + X2W2 + X3W3 .... + XnWn) and then these weighted sums of inputs are passed through an

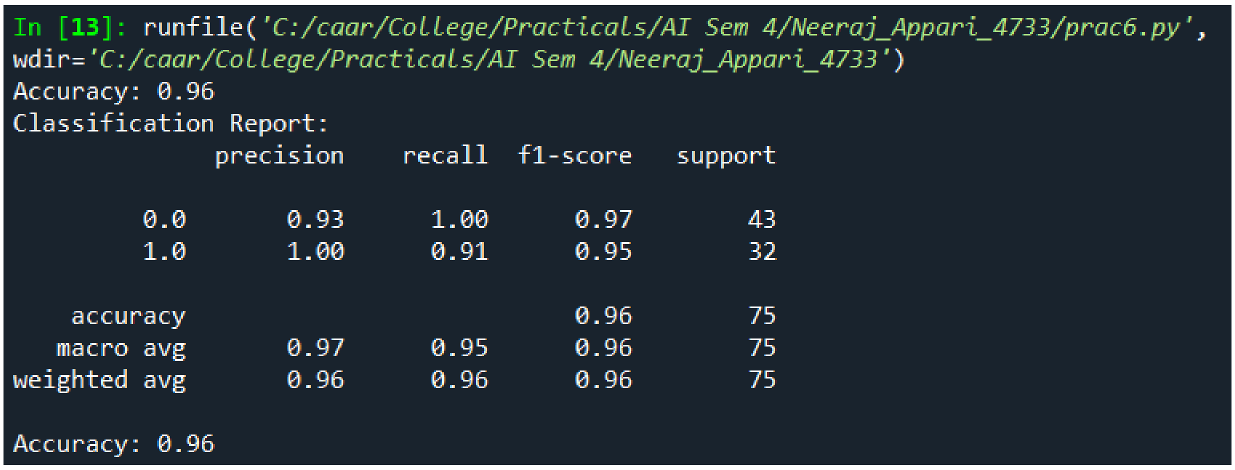
activation function also known as the step function. These activation functions will determine whether the Perceptron needs to be activated or not.

Code:

```
import numpy as np
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris  # Importing Iris dataset

class Perceptron:
    def _init_(self, learning_rate, epochs):
        self.weights = None
        self.bias = None
        self.learning_rate = learning_rate
        self.epochs = epochs

    def activation(self, z):
        return np.heaviside(z, 0)

    def fit(self, X, y):
        n_features = X.shape[1]
        self.weights = np.zeros((n_features))
        self.bias = 0

        for epoch in range(self.epochs):
            for i in range(len(X)):
                z = np.dot(X, self.weights) + self.bias
                y_pred = self.activation(z)

                self.weights = self.weights + self.learning_rate * (y[i] - y_pred[i]) * X[i]
                self.bias = self.bias + self.learning_rate * (y[i] - y_pred[i])

        return self.weights, self.bias
    def predict(self, X):
        z = np.dot(X, self.weights) + self.bias
        return self.activation(z)

# Loading Iris dataset
iris = load_iris()
X = iris.data[:, (0, 1)]  # Considering only petal length and petal width features
y = (iris.target == 0).astype(int)  # Binary classification for class 0



# Splitting the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=42)

# Training and making predictions
perceptron = Perceptron(0.001, 100)
perceptron.fit(X_train, y_train)
pred = perceptron.predict(X_test)

# Accuracy calculation
accuracy = accuracy_score(pred, y_test)
print("Accuracy:", accuracy)

# Classification report
report = classification_report(pred, y_test, digits=2)
print(report)

# Classifying Iris dataset using Scikit-learn Perceptron class
from sklearn.linear_model import Perceptron as SkPerceptron
sk_perceptron = SkPerceptron()
```

```
sk_perceptron.fit(X_train, y_train)
sk_perceptron_pred = sk_perceptron.predict(X_test)

# Accuracy calculation using scikit-learn Perceptron
sk_accuracy = accuracy_score(sk_perceptron_pred, y_test)
print("Scikit-learn Perceptron Accuracy:", sk_accuracy)
```

Output:



```
In [13]: runfile('C:/caar/College/Practicals/AI Sem 4/Neeraj_Appari_4733/prac6.py',
wdir='C:/caar/College/Practicals/AI Sem 4/Neeraj_Appari_4733')
Accuracy: 0.96
Classification Report:
              precision    recall  f1-score   support

         0.0       0.93      1.00      0.97        43
         1.0       1.00      0.91      0.95        32

    accuracy                           0.96        75
   macro avg       0.97      0.95      0.96        75
weighted avg       0.96      0.96      0.96        75

Accuracy: 0.96
```

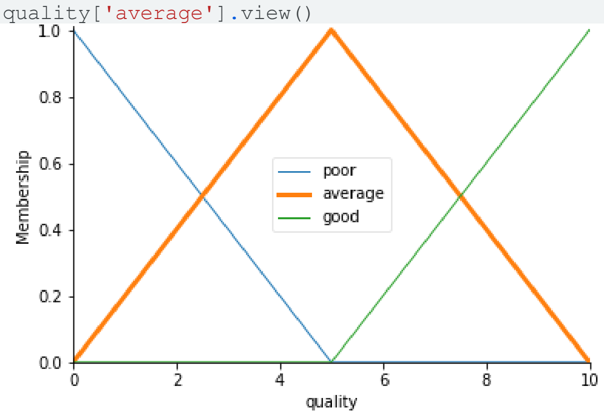
Practical No.: 7

Aim: Fuzzy Interference Systems: Tipping problem easy

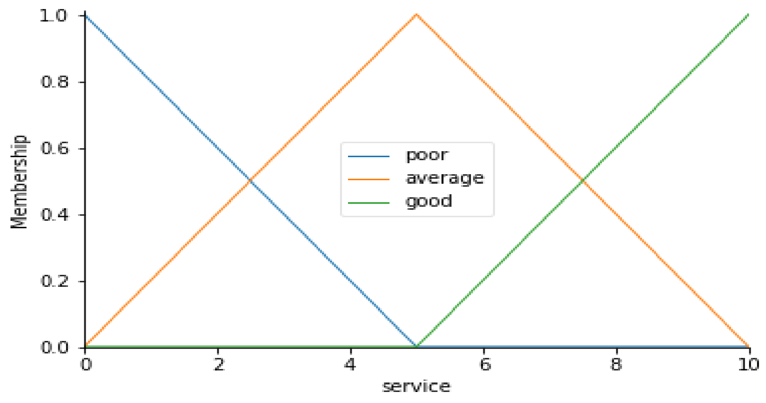
Code:

```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl

# New Antecedent/Consequent objects hold universe variables and membership
# functions
quality = ctrl.Antecedent(np.arange(0, 11, 1), 'quality')
service = ctrl.Antecedentp.arange(0, 11, 1), 'service')
tip = ctrl.Consequent(np.arange(0, 26, 1), 'tip')

# Auto-membership function population is possible with .automf(3, 5, or 7)
quality.automf(3)
service.automf(3)

# Custom membership functions can be built interactively with a familiar,
# Pythonic API
tip['low'] = fuzz.trimf(tip.universe, [0, 0, 13])
tip['medium'] = fuzz.trimf(tip.universe, [0, 13, 25])
tip['high'] = fuzz.trimf(tip.universe, [13, 25, 25])
```

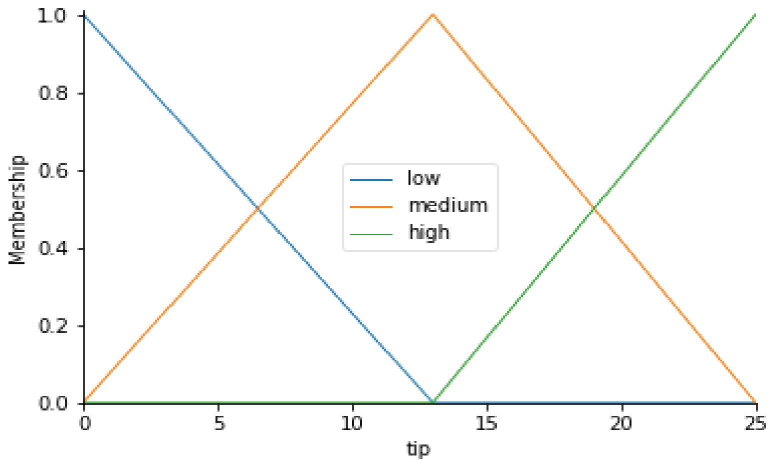
```
# You can see how these look with .view()
quality['average'].view()
```



```
service.view()
```

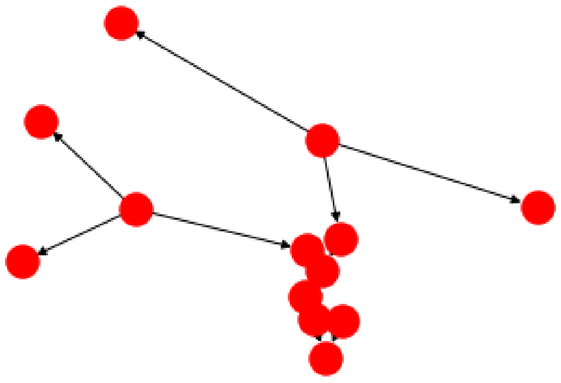


```
tip.view()
```



```
rule1 = ctrl.Rule(quality['poor'] | service['poor'], tip['low'])
rule2 = ctrl.Rule(service['average'], tip['medium'])
rule3 = ctrl.Rule(service['good'] | quality['good'], tip['high'])

rule1.view()
```

```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f939fa37710>)
```



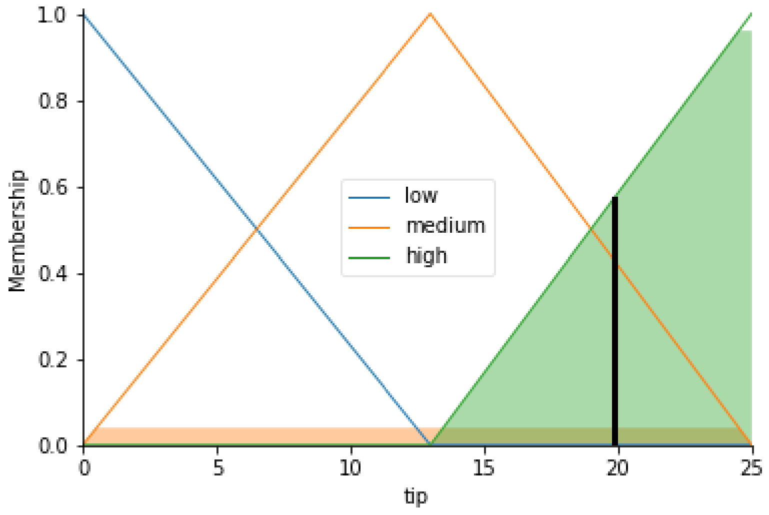
```
tipping_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])
```

```
tipping = ctrl.ControlSystemSimulation(tipping_ctrl)
```

```
# Pass inputs to the ControlSystem using Antecedent labels with Pythonic API
# Note: if you like passing many inputs all at once, use .inputs(dict_of_data)
tipping.input['quality'] = 6.5
tipping.input['service'] = 9.8

# Crunch the numbers
tipping.compute()
```

```
print(tipping.output['tip'])
tip.view(sim=tipping)
```

```
19.847607361963192
```



Practical No.:8

Aim: Fuzzy Interference Systems: Tipping problem hard

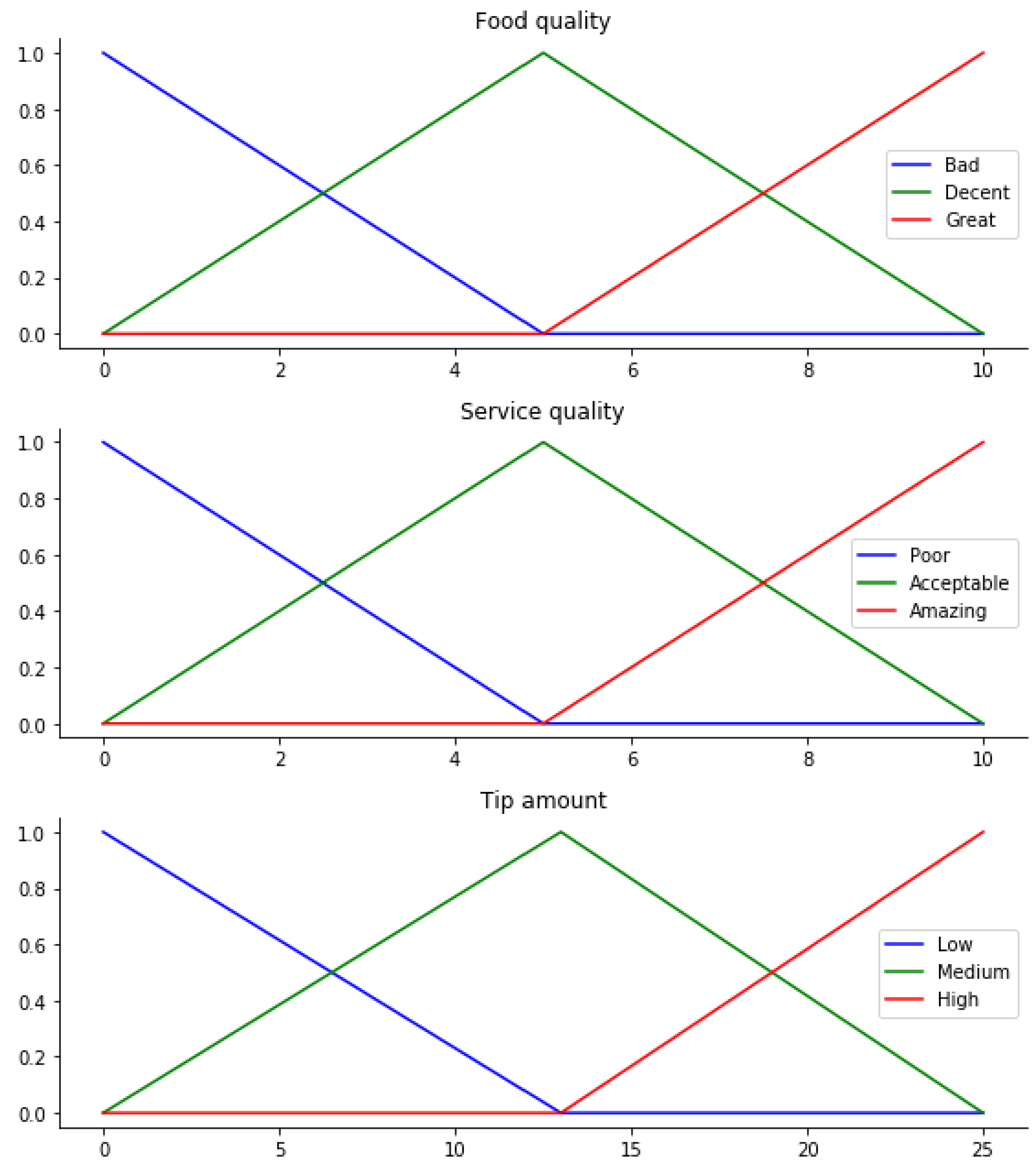
Code:

```
import numpy as np
import skfuzzy as fuzz
import matplotlib.pyplot as plt
```

```
# Generate universe variables
#   * Quality and service on subjective ranges [0, 10]
#   * Tip has a range of [0, 25] in units of percentage points
x_qual = np.arange(0, 11, 1)
x_serv = np.arange(0, 11, 1)
x_tip  = np.arange(0, 26, 1)
```

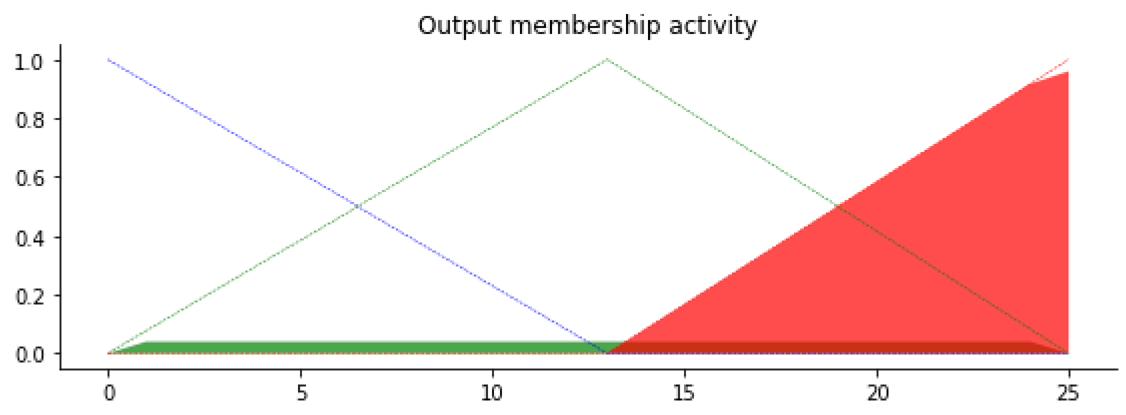
```
# Generate fuzzy membership functions
qual_lo = fuzz.trimf(x_qual, [0, 0, 5])
qual_md = fuzz.trimf(x_qual, [0, 5, 10])
qual_hi = fuzz.trimf(x_qual, [5, 10, 10])
serv_lo = fuzz.trimf(x_serv, [0, 0, 5])
serv_md = fuzz.trimf(x_serv, [0, 5, 10])
serv_hi = fuzz.trimf(x_serv, [5, 10, 10])
tip_lo = fuzz.trimf(x_tip, [0, 0, 13])
tip_md = fuzz.trimf(x_tip, [0, 13, 25])
tip_hi = fuzz.trimf(x_tip, [13, 25, 25])
```

```
# Visualize these universes and membership functions
fig, (ax0, ax1, ax2) = plt.subplots(nrows=3, figsize=(8, 9))

ax0.plot(x_qual, qual_lo, 'b', linewidth=1.5, label='Bad')
ax0.plot(x_qual, qual_md, 'g', linewidth=1.5, label='Decent')
ax0.plot(x_qual, qual_hi, 'r', linewidth=1.5, label='Great')
ax0.set_title('Food quality')
ax0.legend()

ax1.plot(x_serv, serv_lo, 'b', linewidth=1.5, label='Poor')
ax1.plot(x_serv, serv_md, 'g', linewidth=1.5, label='Acceptable')
ax1.plot(x_serv, serv_hi, 'r', linewidth=1.5, label='Amazing')
ax1.set_title('Service quality')
ax1.legend()

ax2.plot(x_tip, tip_lo, 'b', linewidth=1.5, label='Low')
ax2.plot(x_tip, tip_md, 'g', linewidth=1.5, label='Medium')
ax2.plot(x_tip, tip_hi, 'r', linewidth=1.5, label='High')
ax2.set_title('Tip amount')
ax2.legend()

# Turn off top/right axes
for ax in (ax0, ax1, ax2):
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    ax.get_xaxis().tick_bottom()
    ax.get_yaxis().tick_left()

plt.tight_layout()
```

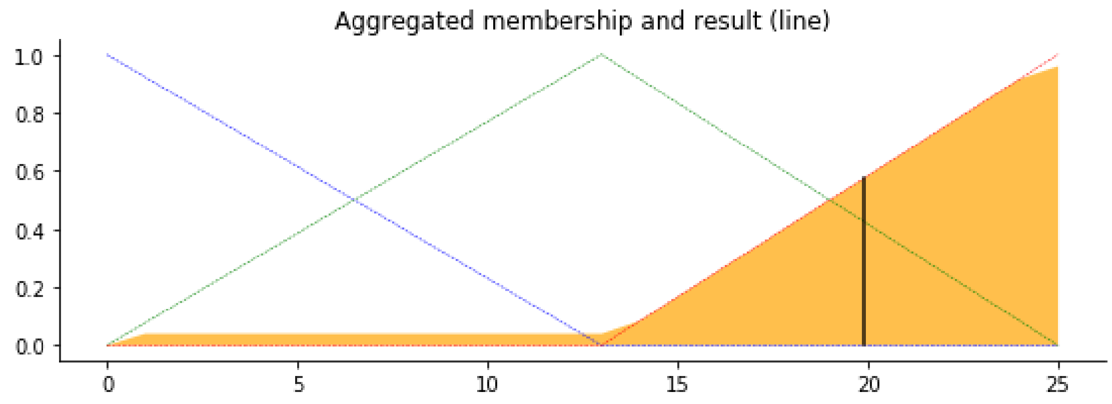


```
# We need the activation of our fuzzy membership functions at these values.
# The exact values 6.5 and 9.8 do not exist on our universes...
# This is what fuzz.interp_membership exists for!
qual_level_lo = fuzz.interp_membership(x_qual, qual_lo, 6.5)
qual_level_md = fuzz.interp_membership(x_qual, qual_md, 6.5)
qual_level_hi = fuzz.interp_membership(x_qual, qual_hi, 6.5)

serv_level_lo = fuzz.interp_membership(x_serv, serv_lo, 9.8)
serv_level_md = fuzz.interp_membership(x_serv, serv_md, 9.8)
serv_level_hi = fuzz.interp_membership(x_serv, serv_hi, 9.8)

# Now we take our rules and apply them. Rule 1 concerns bad food OR service.
# The OR operator means we take the maximum of these two.
active_rule1 = np.fmax(qual_level_lo, serv_level_lo)

# Now we apply this by clipping the top off the corresponding output
# membership function with `np.fmin`
tip_activation_lo = np.fmin(active_rule1, tip_lo)  # removed entirely to 0
```

```python
# For rule 2 we connect acceptable service to medium tipping
tip_activation_md = np.fmin(serv_level_md, tip_md)

# For rule 3 we connect high service OR high food with high tipping
active_rule3 = np.fmax(qual_level_hi, serv_level_hi)
tip_activation_hi = np.fmin(active_rule3, tip_hi)
tip0 = np.zeros_like(x_tip)

# Visualize this
fig, ax0 = plt.subplots(figsize=(8, 3))

ax0.fill_between(x_tip, tip0, tip_activation_lo, facecolor='b', alpha=0.7)
ax0.plot(x_tip, tip_lo, 'b', linewidth=0.5, linestyle='--', )
ax0.fill_between(x_tip, tip0, tip_activation_md, facecolor='g', alpha=0.7)
ax0.plot(x_tip, tip_md, 'g', linewidth=0.5, linestyle='--')
ax0.fill_between(x_tip, tip0, tip_activation_hi, facecolor='r', alpha=0.7)
ax0.plot(x_tip, tip_hi, 'r', linewidth=0.5, linestyle='--')
ax0.set_title('Output membership activity')

# Turn off top/right axes
for ax in (ax0,):
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    ax.get_xaxis().tick_bottom()
    ax.get_yaxis().tick_left()

plt.tight_layout()
```



```python
# Aggregate all three output membership functions together
aggregated = np.fmax(tip_activation_lo,
                     np.fmax(tip_activation_md, tip_activation_hi))

# Calculate defuzzified result
tip = fuzz.defuzz(x_tip, aggregated, 'centroid')
tip_activation = fuzz.interp_membership(x_tip, aggregated, tip)  # for plot

# Visualize this
fig, ax0 = plt.subplots(figsize=(8, 3))

ax0.plot(x_tip, tip_lo, 'b', linewidth=0.5, linestyle='--', )
ax0.plot(x_tip, tip_md, 'g', linewidth=0.5, linestyle='--')
ax0.plot(x_tip, tip_hi, 'r', linewidth=0.5, linestyle='--')
ax0.fill_between(x_tip, tip0, aggregated, facecolor='Orange', alpha=0.7)
ax0.plot([tip, tip], [0, tip_activation], 'k', linewidth=1.5, alpha=0.9)
ax0.set_title('Aggregated membership and result (line)')

# Turn off top/right axes
for ax in (ax0,):
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    ax.get_xaxis().tick_bottom()
    ax.get_yaxis().tick_left()

plt.tight_layout()
```



Neeraj Appari 4733

Practical No.:9

Aim: Naive Bayes' learning algorithm.

Theory:

Naive Bayes is a family of probabilistic machine learning algorithms based on the Bayes Theorem with an assumption of independence among the features.

The Naive Bayes classifier assumes that the presence of a feature in a class is not related to any other feature.

Naive Bayes is a classification algorithm for binary and multi-class classification problems.

Naive Bayes Theorem

Based on prior knowledge of conditions that may be related to an event, Bayes theorem describes the probability of the event conditional probability can be found this way.

Assume we have a Hypothesis(H) and evidence(E), According to Bayes theorem, the relationship between the probability of the Hypothesis before

getting the evidence represented as P(H) and the probability of the hypothesis after getting the evidence represented as P(H|E) is:

P(H|E) = P(E|H)*P(H)/P(E)

Prior probability = P(H) is the probability before getting the evidence

Posterior probability = P(H|E) is the probability after getting evidence

In general,

P(class|data) = (P(data|class) * P(class)) / P(data)

Code/Output:

Naive Bayes Scratch Implementation using Python

Here we are implementing a Naive Bayes Algorithm using Gaussian distributions. It performs all

the necessary steps from data preparation and model training to testing and evaluation.

Importing Libraries

Importing necessary libraries:

1. math: for mathematical operations

2. random: for random number generation

3. pandas: for data manipulation

4. numpy: for scientific computing

Code:

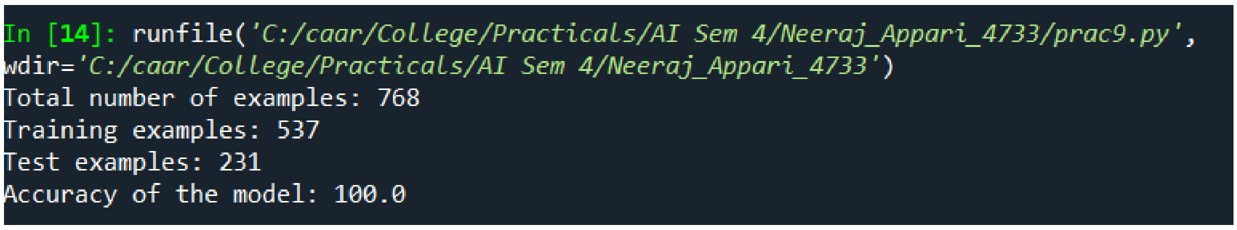
```
import math
import random
import pandas as pd
import numpy as np

# Function to encode class labels into numeric values
def encode_class(mydata):
    classes = []
    for i in range(len(mydata)):
        if mydata[i][-1] not in classes:
            classes.append(mydata[i][-1])

    for i in range(len(classes)):
        for j in range(len(mydata)):
            if mydata[j][-1] == classes[i]:
                mydata[j][-1] = i

    return mydata

# Function to split data into training and testing sets
def splitting(mydata, ratio):
    train_num = int(len(mydata) * ratio)
    train = []
    test = list(mydata)
    while len(train) < train_num:
        index = random.randrange(len(test))
        train.append(test.pop(index))
    return train, test
```

```
# Function to group data by class
def groupUnderClass(mydata):
    data_dict = {}
    for i in range(len(mydata)):
        if mydata[i][-1] not in data_dict:
            data_dict[mydata[i][-1]] = []
        data_dict[mydata[i][-1]].append(mydata[i])
    return data_dict

# Function to calculate mean and standard deviation for a list of numbers
def MeanAndStdDev(numbers):
    avg = np.mean(numbers)
    stddev = np.std(numbers)
    return avg, stddev

# Function to calculate mean and standard deviation for each attribute of each class
def MeanAndStdDevForClass(mydata):
    info = {}
    data_dict = groupUnderClass(mydata)
    for classValue, instances in data_dict.items():
        info[classValue] = [MeanAndStdDev(attribute) for attribute in zip(*instances)]
    return info

# Function to calculate Gaussian probability
def calculateGaussianProbability(x, mean, stdev):
    epsilon = 1e-10
    expo = math.exp(-(math.pow(x - mean, 2) / (2 * math.pow(stdev + epsilon, 2))))
    return (1 / (math.sqrt(2 * math.pi) * (stdev + epsilon))) * expo

# Function to calculate probabilities for each class
```

```
def calculateClassProbabilities(info, test):
    probabilities = {}
    for classValue, classSummaries in info.items():
        probabilities[classValue] = 1
        for i in range(len(classSummaries)):
            mean, std_dev = classSummaries[i]
            x = test[i]
            probabilities[classValue] *= calculateGaussianProbability(x, mean, std_dev)
    return probabilities

# Function to predict class for a test data point
def predict(info, test):
    probabilities = calculateClassProbabilities(info, test)
    bestLabel = max(probabilities, key=probabilities.get)
    return bestLabel

# Function to make predictions for the entire test set
def getPredictions(info, test):
    predictions = [predict(info, instance) for instance in test]
    return predictions

# Function to calculate accuracy rate
def accuracy_rate(test, predictions):
    correct = sum(1 for i in range(len(test)) if test[i][-1] == predictions[i])
    return (correct / float(len(test))) * 100.0

# Load data using pandas
filename = 'D:/ai/diabetes_data.csv' # Add the correct file path
df = pd.read_csv(filename)
mydata = df.values.tolist()
```

```
# Encode classes and convert attributes to float
mydata = encode_class(mydata)
for i in range(len(mydata)):
    for j in range(len(mydata[i]) - 1):
        mydata[i][j] = float(mydata[i][j])

# Split the data into training and testing sets
ratio = 0.7
train_data, test_data = splitting(mydata, ratio)
print('Total number of examples:', len(mydata))
print('Training examples:', len(train_data))
print('Test examples:', len(test_data))

# Train the model
info = MeanAndStdDevForClass(train_data)

# Test the model
predictions = getPredictions(info, test_data)
accuracy = accuracy_rate(test_data, predictions)
print('Accuracy of the model:', accuracy)
```

Output:



```
In [14]: runfile('C:/caar/College/Practicals/AI Sem 4/Neeraj_Appari_4733/prac9.py',
wdir='C:/caar/College/Practicals/AI Sem 4/Neeraj_Appari_4733')
Total number of examples: 768
Training examples: 537
Test examples: 231
Accuracy of the model: 100.0
```