RNS INSTITUTE OF TECHNOLOGY

Dr. VISHNUVARDHAN ROAD, CHANNASANDRA, BENGALURU-560 098

Department of Information Science & Engineering

ARTIFICIAL INTELLIGENCE

&

MACHINE LEARNING LABORATORY

18CSL76

LABORATORY MANUAL

VII Semester

Faculty-in-charge

Mrs. Kusuma R

Mrs. Poonam Kumari

RNS INSTITUTE OF TECHNOLOGY

Dr. VISHNUVARDHAN ROAD, CHANNASANDRA, BENGALURU -560 098

Department of Information Science and Engineering

VISION of the College

Building RNSIT into a World - Class Institution

MISSION of the College

To impart high quality education in Engineering, Technology and Management with a difference, enabling students to excel in their career by

1. Attracting quality Students and preparing them with a strong foundation in fundamentals so as *to achieve distinctions in various walks of life* leading to outstanding contributions.
2. Imparting value based, need based, and choice based and skill based professional education to the aspiring youth and *carving them into disciplined, World class Professionals* with *social responsibility.*
3. Promoting excellence in Teaching, Research and Consultancy that galvanizes academic consciousness among Faculty and Students.
4. Exposing Students to emerging frontiers of knowledge in various domains and make them suitable for Industry, Entrepreneurship, Higher studies, and Research & Development.
5. Providing freedom of action and choice for all the Stake holders with better visibility.

VISION of the Department

Fostering winning professionals of Strong Informative Potentials.

MISSION of the Department

Imparting high quality education in the area of Information Science so as to graduate the students with **good fundamentals**, "**Information System Integration", "Software Creation"** capability & suitably train them to thrive in **Industries, higher schools of learning** and **R & D centers** with a comprehensive perspective.

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

ISE Graduates within three-four years of graduation will have

- **PEO1**: Acquired the fundamentals of computers and applied knowledge of Information Science & Engineering and continue to develop their technical competencies by problem solving using programming.
- **PEO2**: Ability to formulate problems, attained the Proficiency to develop system/application software in a scalable and robust manner with various platforms, tools and frameworks to provide cost effective solutions.
- **PEO3**: Obtained the capacity to investigate the necessities of the software Product, adapt to technological advancement, promote collaboration and interdisciplinary activities, Protecting Environment and developing Comprehensive leadership.
- **PEO4**: Enabled to be employed and provide innovative solutions to real-world problems across different domains.
- **PEO5**: Possessed communication skills, ability to work in teams, professional ethics, social responsibility, entrepreneur and management, to achieve higher career goals, and pursue higher studies.

PROGRAM OUTCOMES(POs)

Engineering Graduates will be ableto:

- **PO1: Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization for the solution of complex engineering problems
- **PO2: Problem analysis:** Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences.
- **PO3: Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for public health and safety, and cultural, societal, and environmental considerations.
- **PO4: Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
- **PO5: Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools, including prediction and modeling to complex engineering activities, with an understanding of the limitations.
- **PO6: The engineer and society:** Apply reasoning informed by the contextual knowledge to assess Societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

- **PO7: Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
- **PO8: Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
- **PO9: Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
- **PO10: Communication:** Communicate effectively on complex engineering activities with the engineering community and with the society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
- **PO11: Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
- **PO12: Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES (PSOs)

ISE Graduateswill have

- **PSO1 – Problem Solving Abilities:** Ability to demonstrate the fundamental and theoretical concepts, analyze the real time problems and develop customized software solutions by applying the knowledge of mathematics and algorithmic techniques.
- **PSO2 – Applied Engineering Skills:** Enable creative thinking, Ability to apply standard practices and strategies, technical skills in software design, development, integration of systems and management for improving the security, reliability and survivability of the infrastructure.
- **PSO3 – General Expertise and Higher Learning:** Ability to exchange knowledge effectively demonstrate the ability of team work, documentation skills, professional ethics, entrepreneurial skills and continuing higher education in the field of Information technology.

RNS INSTITUTE OF TECHNOLOGY

Dr. VISHNUVARDHAN ROAD, CHANNASANDRA, BENGALURU -560 098

Department of Information Science and Engineering

Artificial Intelligence & Machine Learning Laboratory

Subject Code: 18CSL76 **Total Hours: 40**
Hours/Week: 01I + 2P **Exam Hours: 03**

Scheme	18CSL76
I.A. Marks	40
Exam Marks	60

Course objectives

This course will enable students to

- Make use of data sets in implementing the machine learning algorithms.
- Implement the machine learning concepts and algorithms in any suitable language of choice.

Course Outcomes

After studying this course, students will be able to:

CO1	Implement and demonstrate the searching and concept learning algorithms (A* and AO*, Candidate-Elimination)
CO2	Demonstrate the working of the decision tree and apply this knowledge to classify a new sample. (ID3)
CO3	Build an Artificial Neural Network by implementing the Back propagation algorithm.
CO4	Apply the naïve Bayesian classification methods and build Bayesian network.
CO5	Implement EM algorithm, k-means and Compare their results.
CO6	Investigate instance based and regression algorithms.

CO mapping to PO/PSOs

CO / PO & PSO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
18CSL76.1	2	2	2									2	2	1	2
18CSL76.2	2	2	2		1							2	2	1	2
18CSL76.3	2	2	2		1							2	2	1	2
18CSL76.4	2	2	2	2	1							2	2	1	2
18CSL76.5	2	2	2	2	1							2	2	1	2
18CSL76.6	2	2	2		1							2	2	1	2

Artificial Intelligence & Machine Learning Laboratory

Subject Code: 18CSL76 **Total Hours: 40**
Hours/Week: 01I + 2P **Exam Hours: 03**

Scheme	18CSL76
I.A. Marks	40
Exam Marks	60

List of Programs

Sl. No	Name of Experiment	CO
1	**Implement and evaluate AI and ML algorithms in Python programming language** Implement A* Search algorithm.	CO1
2	Implement AO* Search algorithm	CO1
3	For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.	CO1
4	Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.	CO2
5	Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.	CO3
6	Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.	CO4
7	Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.	CO5
8	Write a program to implement k-Nearest Neighbor algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.	CO6
9	Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs	CO6

Artificial Intelligence & Machine Learning Laboratory

Evaluation Rubrics

Subject Code: 18CSL76 **I.A. Marks : 40**
Hours/Week: 01I+2P **Exam Hours: 03**
Total Hours: 40 **Exam Marks: 60**

Lab Write-up and Execution Rubrics (Max: 24 marks)

		Above Average	Average	Below Average
a.	**Understanding of problem and approach to solve. (8 Marks)**	Able to analyze the given problem and efficiently implement using suitable high-level language instructions.(8-6)	Able to analyze the problem and moderate understanding of high-level language instructions. (5-3)	Poor understanding of high-level language instructions or No program write-up. (2-0)
b.	**Execution and Viva (5 questions) (8 Marks)**	Program executed for varied inputs with valid results and able to answer allfive questions appropriately.(8-6)	Program is executed for some inputs and able to answer three-two questions. (5-3)	Program has compilation errors or no Execution and not answered any questions. (2-0)
c.	**Results and Documentation (8 Marks)**	Program and results obtained arelegibly written / documented. (8-6)	Program and results obtained is acceptably documented. (5-3)	No Proper results and poor documentation. (2-0)

LAB Internal Assessment rubrics (Max: 16 marks)

		Above Average	Average	Below Average
a.	**Write-up (3 Marks)**	Able to write the complete code.(3)	Able to write the code with few errors. (2-1)	Unable to write. (0)
b.	**Execution (10 Marks)**	Executed successfully for all the inputs given. (10-7)	Obtained partially correct results. (6-3)	Program has compilation errors or No Execution. (2-0)
c.	**Viva (5 questions) (3 Marks)**	Able to answer all five questions correctly. (3)	Able to answer three-two questions. (2-1)	Not answered any.(0)

Artificial Intelligence & Machine Learning Laboratory

Subject Code: 18CSL76 **Total Hours: 40**
Hours/Week: 01I + 2P **Exam Hours: 03**

Scheme	18CSL76
I.A. Marks	40
Exam Marks	60

Lesson Planning / Schedule of Experiments

Week No.	Name of Experiment	Page No.
1	Demonstration of Installation procedure of required software's – python idle and jupyter notebook and Sample Programs in Python.	
2	Implement A* Search algorithm.	1-3
3	Implement AO* Search algorithm.	4-8
4	For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.	9-12
5	Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.	13-15
6	Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.	16-17
7	**Lab Internal-1**	
8	Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.	18-21
9	Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.	22-28
10	Write a program to implement k-Nearest Neighbor algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.	29-38
11	Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.	39
12	**Lab Internal-2**	

1. Implement A* Search algorithm.

```
def aStarAlgo(start_node, stop_node):

    open_set = set(start_node)
    closed_set = set()
    g = {} #store distance from starting node
    parents = {}# parents contains an adjacency map of all nodes

    #ditance of starting node from itself is zero
    g[start_node] = 0
    #start_node is root node i.e it has no parent nodes
    #so start_node is set to its own parent node
    parents[start_node] = start_node


    while len(open_set) > 0:
       n = None

       #node with lowest f() is found
       for v in open_set:                                          small value selection
          if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):
             n = v


       if n == stop_node or Graph_nodes[n] == None:
          pass
       else:
          for (m, weight) in get_neighbors(n):
             #nodes 'm' not in first and last set are added to first
             #n is set its parent
             if m not in open_set and m not in closed_set:
                open_set.add(m)
                parents[m] = n
                g[m] = g[n] + weight


             #for each node m,compare its distance from start i.e g(m) to the
             #from start through n node
             else:
                if g[m] > g[n] + weight:
                   #update g(m)
                   g[m] = g[n] + weight
                   #change parent of m to n
                   parents[m] = n

                   #if m in closed set,remove and add to open
                   if m in closed_set:
                      closed_set.remove(m)
                      open_set.add(m)
```

```
            if n == None:
                print('Path does not exist!')
                return None

            # if the current node is the stop_node
            # then we begin reconstructin the path from it to the start_node
            if n == stop_node:
                path = []

                while parents[n] != n:
                    path.append(n)
                    n = parents[n]

                path.append(start_node)

                path.reverse()

                print('Path found: {}'.format(path))
                return path


            # remove n from the open_list, and add it to closed_list
            # because all of his neighbors were inspected
            open_set.remove(n)
            closed_set.add(n)

        print('Path does not exist!')
        return None

#define fuction to return neighbor and its distance
#from the passed node
def get_neighbors(v):
    if v in Graph_nodes:
        return Graph_nodes[v]
    else:
        return None
#for simplicity we ll consider heuristic distances given
#and this function returns heuristic distance for all nodes
def heuristic(n):
        H_dist = {
            'A': 11,
            'B': 6,
            'C': 99,
            'D': 1,
            'E': 7,
            'G': 0,

        }

        return H_dist[n]
```

```
#Describe your graph here
Graph_nodes = {
    'A': [('B', 2), ('E', 3)],
    'B': [('C', 1),('G', 9)],
    'C': None,
    'E': [('D', 6)],
    'D': [('G', 1)],

}
aStarAlgo('A', 'G')
```

OUTPUT:

```
Path found: ['A', 'F', 'G', 'I', 'J']
['A', 'F', 'G', 'I', 'J']
```

2. Implement AO* Search algorithm.

```
class Graph:
   def __init__(self, graph, heuristicNodeList, startNode):  #instantiate graph object with graph topology, heuristic values, start node

      self.graph = graph
      self.H=heuristicNodeList
      self.start=startNode
      self.parent={}
      self.status={}
      self.solutionGraph={}

   def applyAOStar(self):        # starts a recursive AO* algorithm
      self.aoStar(self.start, False)

   def getNeighbors(self, v):     # gets the Neighbors of a given node
      return self.graph.get(v,'')

   def getStatus(self,v):         # return the status of a given node
      return self.status.get(v,0)

   def setStatus(self,v, val):    # set the status of a given node
      self.status[v]=val

   def getHeuristicNodeValue(self, n):
      return self.H.get(n,0)     # always return the heuristic value of a given node

   def setHeuristicNodeValue(self, n, value):
      self.H[n]=value            # set the revised heuristic value of a given node


   def printSolution(self):
      print("FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE START NODE:",self.start)
      print("------------------------------------------------------------")
      print(self.solutionGraph)
      print("------------------------------------------------------------")

def computeMinimumCostChildNodes(self, v):  # Computes the Minimum Cost of child nodes of a given node v
      minimumCost=0
      costToChildNodeListDict={}
      costToChildNodeListDict[minimumCost]=[]
      flag=True
      for nodeInfoTupleList in self.getNeighbors(v):  # iterate over all the set of child node/s
         cost=0
         nodeList=[]
         for c, weight in nodeInfoTupleList:
            cost=cost+self.getHeuristicNodeValue(c)+weight
```

empty

```
            nodeList.append(c)

        if flag==True:                  # initialize Minimum Cost with the cost of first set of child node/s
            minimumCost=cost
            costToChildNodeListDict[minimumCost]=nodeList       # set the Minimum Cost child node/s
            flag=False
        else:                           # checking the Minimum Cost nodes with the current Minimum Cost
            if minimumCost>cost:
                minimumCost=cost
                costToChildNodeListDict[minimumCost]=nodeList  # set the Minimum Cost child node/s


    return minimumCost, costToChildNodeListDict[minimumCost]    # return Minimum Cost and Minimum Cost child node/s


  def aoStar(self, v, backTracking):    # AO* algorithm for a start node and backTracking status flag

    print("HEURISTIC VALUES  :", self.H)
    print("SOLUTION GRAPH    :", self.solutionGraph)
    print("PROCESSING NODE   :", v)
    print("-------------------------------------------------------------------------------------------")

    if self.getStatus(v) >= 0:      # if status node v >= 0, compute Minimum Cost nodes of v
        minimumCost, childNodeList = self.computeMinimumCostChildNodes(v)
        self.setHeuristicNodeValue(v, minimumCost)
        self.setStatus(v,len(childNodeList))

        solved=True                 # check the Minimum Cost nodes of v are solved
        for childNode in childNodeList:
            self.parent[childNode]=v
            if self.getStatus(childNode)!=-1:
                solved=solved & False

        if solved==True:            # if the Minimum Cost nodes of v are solved, set the current node status as solved(-1)
            self.setStatus(v,-1)
            self.solutionGraph[v]=childNodeList # update the solution graph with the solved nodes which may be a part of solution


        if v!=self.start:           # check the current node is the start node for backtracking the current node value
            self.aoStar(self.parent[v], True)       # backtracking the current node value with backtracking status set to true
```

```
        if backTracking==False:     # check the current call is not for backtracking
            for childNode in childNodeList:   # for each Minimum Cost child node
                self.setStatus(childNode,0)   # set the status of child node to 0(needs exploration)
                self.aoStar(childNode, False) # Minimum Cost child node is further explored with backtracking status as false

h1 = {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1, 'T': 3}
graph1 = {
   'A': [[('B', 1), ('C', 1)], [('D', 1)]],
   'B': [[('G', 1)], [('H', 1)]],
   'C': [[('J', 1)]],
   'D': [[('E', 1), ('F', 1)]],
   'G': [[('I', 1)]]
}
G1= Graph(graph1, h1, 'A')
G1.applyAOStar()
G1.printSolution()

h2 = {'A': 1, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}  # Heuristic values of Nodes
graph2 = {                                # Graph of Nodes and Edges
   'A': [[('B', 1), ('C', 1)], [('D', 1)]],      # Neighbors of Node 'A', B, C & D with repective weights
   'B': [[('G', 1)], [('H', 1)]],              # Neighbors are included in a list of lists
   'D': [[('E', 1), ('F', 1)]]               # Each sublist indicate a "OR" node or "AND" nodes
}

G2 = Graph(graph2, h2, 'A')                    # Instantiate Graph object with graph, heuristic values and start Node
G2.applyAOStar()                           # Run the AO* algorithm
G2.printSolution()                          # Print the solution graph as output of the AO* algorithm search
```

OUTPUT:

```
HEURISTIC VALUES  : {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1, 'T': 3}
SOLUTION GRAPH    : {}
PROCESSING NODE   : A
-------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1, 'T': 3}
SOLUTION GRAPH    : {}
PROCESSING NODE   : B
-------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1, 'T': 3}
SOLUTION GRAPH    : {}
PROCESSING NODE   : A
-------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1, 'T': 3}
SOLUTION GRAPH    : {}
PROCESSING NODE   : G
-------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1, 'T': 3}
```

```
SOLUTION GRAPH    : {}
PROCESSING NODE   : B
-------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 10, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1, 'T': 3}
SOLUTION GRAPH    : {}
PROCESSING NODE   : A
-------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1, 'T': 3}
SOLUTION GRAPH    : {}
PROCESSING NODE   : I
-------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 0, 'J': 1, 'T': 3}
SOLUTION GRAPH    : {'I': []}
PROCESSING NODE   : G
-------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1, 'T': 3}
SOLUTION GRAPH    : {'I': [], 'G': ['I']}
PROCESSING NODE   : B
-------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 12, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1, 'T': 3}
SOLUTION GRAPH    : {'I': [], 'G': ['I'], 'B': ['G']}
PROCESSING NODE   : A
-------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1, 'T': 3}
SOLUTION GRAPH    : {'I': [], 'G': ['I'], 'B': ['G']}
PROCESSING NODE   : C
-------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1, 'T': 3}
SOLUTION GRAPH    : {'I': [], 'G': ['I'], 'B': ['G']}
PROCESSING NODE   : A
-------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1, 'T': 3}
SOLUTION GRAPH    : {'I': [], 'G': ['I'], 'B': ['G']}
PROCESSING NODE   : J
-------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 0, 'T': 3}
SOLUTION GRAPH    : {'I': [], 'G': ['I'], 'B': ['G'], 'J': []}
PROCESSING NODE   : C
-------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 6, 'B': 2, 'C': 1, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 0, 'T': 3}
SOLUTION GRAPH    : {'I': [], 'G': ['I'], 'B': ['G'], 'J': [], 'C': ['J']}
PROCESSING NODE   : A
-------------------------------------------------------------------------------------
FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE START NODE: A
------------------------------------------------------------
{'I': [], 'G': ['I'], 'B': ['G'], 'J': [], 'C': ['J'], 'A': ['B', 'C']}
------------------------------------------------------------
HEURISTIC VALUES  : {'A': 1, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH    : {}
PROCESSING NODE   : A
-------------------------------------------------------------------------------------
```

```
HEURISTIC VALUES  : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH    : {}
PROCESSING NODE   : D
------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH    : {}
PROCESSING NODE   : A
------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH    : {}
PROCESSING NODE   : E
------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 0, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH    : {'E': []}
PROCESSING NODE   : D
------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 11, 'B': 6, 'C': 12, 'D': 6, 'E': 0, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH    : {'E': []}
PROCESSING NODE   : A
------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 7, 'B': 6, 'C': 12, 'D': 6, 'E': 0, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH    : {'E': []}
PROCESSING NODE   : F
------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 7, 'B': 6, 'C': 12, 'D': 6, 'E': 0, 'F': 0, 'G': 5, 'H': 7}
SOLUTION GRAPH    : {'E': [], 'F': []}
PROCESSING NODE   : D
------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 7, 'B': 6, 'C': 12, 'D': 2, 'E': 0, 'F': 0, 'G': 5, 'H': 7}
SOLUTION GRAPH    : {'E': [], 'F': [], 'D': ['E', 'F']}
PROCESSING NODE   : A
------------------------------------------------------------------------------------
FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE START NODE: A
------------------------------------------------------------
{'E': [], 'F': [], 'D': ['E', 'F'], 'A': ['D']}
```

3. For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.import random

```
import csv
def g_0(n):
    return ("?",)*n

def s_0(n):
    return ('ϕ',)*n

def more_general(h1, h2):
    more_general_parts = []
    for x, y in zip(h1, h2):
        mg = x == "?" or (x != "ϕ" and (x == y or y == "ϕ"))
        more_general_parts.append(mg)
    return all(more_general_parts)

def fulfills(example, hypothesis):
    ### the implementation is the same as for hypotheses:
    return more_general(hypothesis, example)

def min_generalizations(h, x):
    h_new = list(h)
    for i in range(len(h)):
        if not fulfills(x[i:i+1], h[i:i+1]):
            h_new[i] = '?' if h[i] != 'ϕ' else x[i]
    return [tuple(h_new)]

def min_specializations(h, domains, x):
    results = []
    for i in range(len(h)):
        if h[i] == "?":
            for val in domains[i]:
                if x[i] != val:
```

```
                h_new = h[:i] + (val,) + h[i+1:]
                results.append(h_new)
        elif h[i] != "ϕ":
            h_new = h[:i] + ('ϕ',) + h[i+1:]
            results.append(h_new)
    return results

with open('trainingexamples.csv')  as csvFile:
        examples = [tuple(line) for line in csv.reader(csvFile)]

def get_domains(examples):
    d = [set() for i in examples[0]]
    for x in examples:
        for i, xi in enumerate(x):
            d[i].add(xi)
    return [list(sorted(x)) for x in d]

def candidate_elimination(examples):
    domains = get_domains(examples)[:-1]

    G = set([g_0(len(domains))])
    S = set([s_0(len(domains))])
    i = 0
    print("\n G[{0}]:".format(i), G)
    print("\n S[{0}]:".format(i), S)
    for xcx in examples:
        i = i + 1
        x, cx = xcx[:-1], xcx[-1]  # Splitting data into attributes and decisions
        if cx == 'Y':  # x is positive example
            G = {g for g in G if fulfills(x, g)}
            S = generalize_S(x, G, S)
        else:  # x is negative example
            S = {s for s in S if not fulfills(x, s)}
            G = specialize_G(x, domains, G, S)
```

```
        print("\n G[{0}]:".format(i), G)
        print("\n S[{0}]:".format(i), S)
    return


def generalize_S(x, G, S):
    S_prev = list(S)
    for s in S_prev:
        if s not in S:
            continue
        if not fulfills(x, s):
            S.remove(s)
            Splus = min_generalizations(s, x)
            ## keep only generalizations that have a counterpart in G
            S.update([h for h in Splus if any([more_general(g,h)
                                    for g in G])])
            ## remove hypotheses less specific than any other in S
            S.difference_update([h for h in S if
                            any([more_general(h, h1)
                                for h1 in S if h != h1])])
    return S
def specialize_G(x, domains, G, S):
    G_prev = list(G)
    for g in G_prev:
        if g not in G:
            continue
        if fulfills(x, g):
            G.remove(g)
            Gminus = min_specializations(g, domains, x)
            ## keep only specializations that have a conuterpart in S
            G.update([h for h in Gminus if any([more_general(h, s)
                                    for s in S])])
            ## remove hypotheses less general than any other in G
            G.difference_update([h for h in G if
                            any([more_general(g1, h)
```

```
                    for g1 in G if h != g1])])
    return G
candidate_elimination(examples)
```

OUTPUT:

G[0]: {('?', '?', '?', '?', '?', '?')}

S[0]: {('ϕ', 'ϕ', 'ϕ', 'ϕ', 'ϕ', 'ϕ')}

G[1]: {('?', '?', '?', '?', '?', '?')}

S[1]: {('Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same')}

G[2]: {('?', '?', '?', '?', '?', '?')}

S[2]: {('Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same')}

G[3]: {('Sunny', '?', '?', '?', '?', '?'), ('?', '?', '?', '?', '?', 'Same'), ('?', 'Warm', '?', '?', '?', '?')}

S[3]: {('Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same')}

G[4]: {('Sunny', '?', '?', '?', '?', '?'), ('?', 'Warm', '?', '?', '?', '?')}

S[4]: {('Sunny', 'Warm', '?', 'Strong', '?', '?')}

4. Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

```
def infoGain(P, N):
    import math
    return -P / (P + N) * math.log2(P / ( P + N)) - N / (P + N) * math.log2(N / (P + N))


def insertNode(tree, addTo, Node):
    for k, v in tree.items():
        if isinstance(v, dict):
            tree[k] = insertNode(v, addTo, Node)
    if addTo in tree:
        if isinstance(tree[addTo], dict):
            tree[addTo][Node] = 'None'
        else:
            tree[addTo] = {Node:'None'}
    return tree

def insertConcept(tree, addTo, Node):
    for k, v in tree.items():
        if isinstance(v, dict):
            tree[k] = insertConcept(v, addTo, Node)
    if addTo in tree:
        tree[addTo] = Node
    return tree

def getNextNode(data, AttributeList, concept, conceptVals, tree, addTo):
    Total = data.shape[0]
    if Total == 0:
        return tree

    countC = {}
    for cVal in conceptVals:
        dataCC = data[data[concept] == cVal]
        countC[cVal] = dataCC.shape[0]
    if countC[conceptVals[0]] == 0:
        tree = insertConcept(tree, addTo, conceptVals[1])
        return tree

    if countC[conceptVals[1]] == 0:
        tree = insertConcept(tree, addTo, conceptVals[0])
```

```
        return tree
    ClassEntropy = infoGain(countC[conceptVals[1]],countC[conceptVals[0]])

    Attr = {}
    for a in AttributeList:
        Attr[a] = list(set(data[a]))

    AttrCount = {}
    EntropyAttr = {}
    for att in Attr:
        for vals in Attr [att]:
            for c in conceptVals:
                iData = data[data[att] == vals]
                dataAtt = iData[iData[concept] == c]
                AttrCount[c] = dataAtt.shape[0]
            TotalInfo = AttrCount[conceptVals[1]] + AttrCount[conceptVals[0]]
            if AttrCount[conceptVals[1]] == 0 or AttrCount[conceptVals[0]] == 0:
                InfoGain=0
            else:
                InfoGain = infoGain(AttrCount[conceptVals[1]], AttrCount[conceptVals[0]])

            if att not in EntropyAttr:
                EntropyAttr[att] = ( TotalInfo / Total ) * InfoGain
            else:
                EntropyAttr[att] = EntropyAttr[att] + ( TotalInfo / Total ) * InfoGain

    Gain = {}
    for g in EntropyAttr:
        Gain[g] = ClassEntropy - EntropyAttr[g]

    Node = max(Gain, key = Gain.get)

    tree = insertNode(tree, addTo, Node)
    for nD in Attr[Node]:
        tree = insertNode(tree, Node, nD)
        newData = data[data[Node] == nD].drop(Node, axis = 1)
        AttributeList=list(newData)[:-1]
        tree = getNextNode(newData, AttributeList, concept, conceptVals, tree, nD)
    return tree


def main():
    import pandas as pd
    data = pd.read_csv('id3.csv')
```

```
        AttributeList = list(data)[:-1]
        concept = str(list(data)[-1])
        conceptVals = list(set(data[concept]))
        tree = getNextNode(data, AttributeList, concept, conceptVals, {'root':'None'}, 'root')
        print(tree)
        compute(tree)

    main()
```

OUTPUT:

```
The Resultant Decision Tree is :
{'Outlook': {'Overcast': 'Yes',
        'Rain': {'Wind': {'Strong': 'No', 'Weak': 'Yes'}},
        'Sunny': {'Humidity': {'High': 'No', 'Normal': 'Yes'}}}}
Best Attribute :
 Outlook
Tree Keys:
 dict_keys(['Overcast', 'Rain', 'Sunny'])

Accuracy is : 0.75
```

5. Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.

```
import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)
X = X/np.amax(X) # max of array
y = y/100

def sigmoid (x):
    return 1/(1 + np.exp(-x))

def derivatives_sigmoid(x):
    return x * (1 - x)

epoch=5000
lr=0.1

wh = np.random.uniform(size=(2,3))
bh = np.random.uniform(size=(1,3))
wout = np.random.uniform(size=(3,1))
bout = np.random.uniform(size=(1,1))

for i in range(epoch):
    # forward prop
    hinp=np.dot(X,wh) + bh
    hlayer_act = sigmoid(hinp)
    outinp=np.dot(hlayer_act,wout) + bout
    output = sigmoid(outinp)

    hiddengrad = derivatives_sigmoid(hlayer_act)
    outgrad = derivatives_sigmoid(output)

    EO = y-output
    d_output = EO* outgrad

    EH = d_output.dot(wout.T)
    d_hiddenlayer = EH * hiddengrad

    wout += hlayer_act.T.dot(d_output) *lr
    wh += X.T.dot(d_hiddenlayer) *lr
```

```
print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
```

OUTPUT:

```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.69734296]
 [0.68194708]
 [0.69700956]]
```

6. Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

```
import csv
import random
import math

def loadcsv(filename):
    lines = csv.reader(open(filename, "r"))
    dataset = list(lines)
    for i in range(len(dataset)):
        dataset[i] = [float(x) for x in dataset[i]]
    return dataset

def splitDataset(dataset, splitRatio):
    trainSize = int(len(dataset) * splitRatio)
    trainSet = []
    trainSet,testSet = dataset[:trainSize],dataset[trainSize:]
    return [trainSet, testSet]

def mean(numbers):
    return sum(numbers)/(len(numbers))

def stdev(numbers):
    avg = mean(numbers)
    v = 0
    for x in numbers:
        v += (x-avg)**2
    return math.sqrt(v/(len(numbers)-1))

def summarizeByClass(dataset):
    separated = {}
    for i in range(len(dataset)):
        vector = dataset[i]
        if (vector[-1] not in separated):
            separated[vector[-1]] = []
        separated[vector[-1]].append(vector)
    summaries = {}
    for classValue, instances in separated.items():
        summaries[classValue] = [(mean(attribute), stdev(attribute)) for attribute in
zip(*instances)][:-1]
    return summaries
```

```
def calculateProbability(x, mean, stdev):
    exponent = math.exp((-(x-mean)**2)/(2*(stdev**2)))
    return (1 / ((2*math.pi)**(1/2)*stdev)) * exponent

def predict(summaries, inputVector):
    probabilities = {}
    for classValue, classSummaries in summaries.items():
        probabilities[classValue] = 1
        for i in range(len(classSummaries)):
            mean, stdev = classSummaries[i]
            x = inputVector[i]
            probabilities[classValue] *= calculateProbability(x, mean, stdev)
    bestLabel, bestProb = None, -1
    for classValue, probability in probabilities.items():
        if bestLabel is None or probability > bestProb:
            bestProb = probability
            bestLabel = classValue
    return bestLabel

def getPredictions(summaries, testSet):
    predictions = []
    for i in range(len(testSet)):
        result = predict(summaries, testSet[i])
        predictions.append(result)
    return predictions

def getAccuracy(testSet, predictions):
    correct = 0
    for i in range(len(testSet)):
        if testSet[i][-1] == predictions[i]:
            correct += 1
    return (correct/(len(testSet))) * 100.0

filename = 'pima-indians-diabetes.csv'
splitRatio = 0.67
dataset = loadcsv(filename)
trainingSet, testSet = splitDataset(dataset, splitRatio)
summaries = summarizeByClass(trainingSet)
predictions = getPredictions(summaries, testSet)
print("\nPredictions:\n",predictions)
accuracy = getAccuracy(testSet, predictions)
print('Accuracy ',accuracy)
```

OUTPUT:

Naive Bayes Classifier for concept learning problem
Split 14 rows into
Number of Training data: 12
Number of Test Data: 2

The values assumed for the concept learning attributes are

OUTLOOK=> Sunny=1 Overcast=2 Rain=3
TEMPERATURE=> Hot=1 Mild=2 Cool=3
HUMIDITY=> High=1 Normal=2
WIND=> Weak=1 Strong=2
TARGET CONCEPT:PLAY TENNIS=> Yes=10 No=5

The Training set are:
[1.0, 1.0, 1.0, 1.0, 5.0]

The Test data set are:
[1.0, 1.0, 1.0, 2.0, 5.0]

The Test data set are:
[2.0, 1.0, 1.0, 1.0, 10.0]

The Test data set are:
[3.0, 2.0, 1.0, 1.0, 10.0]

The Test data set are:
[3.0, 3.0, 2.0, 1.0, 10.0]

The Test data set are:
[3.0, 3.0, 2.0, 2.0, 5.0]

The Test data set are:
[2.0, 3.0, 2.0, 2.0, 10.0]

The Test data set are:
[1.0, 2.0, 1.0, 1.0, 5.0]

The Test data set are:
[1.0, 3.0, 2.0, 1.0, 10.0]

The Test data set are:
[3.0, 2.0, 2.0, 1.0, 10.0]

The Test data set are:
[1.0, 2.0, 2.0, 2.0, 10.0]

The Test data set are:
[2.0, 2.0, 1.0, 2.0, 10.0]

The Test data set are:

[2.0, 1.0, 2.0, 1.0, 10.0]
[3.0, 2.0, 1.0, 2.0, 5.0]

Summarize Attributes By Class
{5.0: [(1.5, 1.0), (1.75, 0.9574271077563381), (1.25, 0.5), (1.5, 0.5773502691896257)], 10.0: [(2.125, 0.8345229603962802), (2.25, 0.7071067811865476), (1.625, 0.5175491695067657), (1.375, 0.5175491695067657)]}

Actual values: [5.0]%
Predictions: [5.0, 5.0]%
Accuracy: 50.0%

7. Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import sklearn.metrics as sm
import pandas as pd
import numpy as np
from sklearn import preprocessing
from sklearn.mixture import GaussianMixture

l1 = [0,1,2]

def rename(s):
    l2 = []
    for i in s:
        if i not in l2:
            l2.append(i)
    for i in range(len(s)):
        pos = l2.index(s[i])
        s[i] = l1[pos]
    return s

iris = datasets.load_iris()

X = pd.DataFrame(iris.data,columns =['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width'] )
y = pd.DataFrame(iris.target,columns = ['Targets'])

def graph_plot(l,title,s,target):
        plt.subplot(l[0],l[1],l[2])
        if s==1:
                plt.scatter(X.Sepal_Length,X.Sepal_Width, c=colormap[target], s=40)
        else:
                plt.scatter(X.Petal_Length,X.Petal_Width, c=colormap[target], s=40)
        plt.title(title)

plt.figure()
colormap = np.array(['red', 'lime', 'black'])

graph_plot([1, 2, 1],'sepal',1,y.Targets)
```

```
graph_plot([1, 2, 2],'petal',0,y.Targets)
plt.show()

def fit_model(modelName):
    model = modelName(3)
    model.fit(X)

    plt.figure()
    colormap = np.array(['red', 'lime', 'black'])
    graph_plot([1, 2, 1],'Real Classification',0,y.Targets)
    if modelName == KMeans:
        m = 'Kmeans'
    else:
        m = 'Em'
     y1 = model.predict(X)
    graph_plot([1, 2, 2],m,0,y1)
    plt.show()

    km = rename(y1)
    print("\nPredicted: \n", km)
    print("Accuracy ",sm.accuracy_score(y, km))
    print("Confusion Matrix ",sm.confusion_matrix(y, km))


fit_model(KMeans)
fit_model(GaussianMixture)
```

OUTPUT:

```
IRIS DATA : [[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
 [4.3 3.  1.1 0.1]
 [5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
```

[5.1 3.5 1.4 0.3]
[5.7 3.8 1.7 0.3]
[5.1 3.8 1.5 0.3]
[5.4 3.4 1.7 0.2]
[5.1 3.7 1.5 0.4]
[4.6 3.6 1. 0.2]
[5.1 3.3 1.7 0.5]
[4.8 3.4 1.9 0.2]
[5. 3. 1.6 0.2]
[5. 3.4 1.6 0.4]
[5.2 3.5 1.5 0.2]
[5.2 3.4 1.4 0.2]
[4.7 3.2 1.6 0.2]
[4.8 3.1 1.6 0.2]
[5.4 3.4 1.5 0.4]
[5.2 4.1 1.5 0.1]
[5.5 4.2 1.4 0.2]
[4.9 3.1 1.5 0.2]
[5. 3.2 1.2 0.2]
[5.5 3.5 1.3 0.2]
[4.9 3.6 1.4 0.1]
[4.4 3. 1.3 0.2]
[5.1 3.4 1.5 0.2]
[5. 3.5 1.3 0.3]
[4.5 2.3 1.3 0.3]
[4.4 3.2 1.3 0.2]
[5. 3.5 1.6 0.6]
[5.1 3.8 1.9 0.4]
[4.8 3. 1.4 0.3]
[5.1 3.8 1.6 0.2]
[4.6 3.2 1.4 0.2]
[5.3 3.7 1.5 0.2]
[5. 3.3 1.4 0.2]
[7. 3.2 4.7 1.4]
[6.4 3.2 4.5 1.5]
[6.9 3.1 4.9 1.5]
[5.5 2.3 4. 1.3]
[6.5 2.8 4.6 1.5]
[5.7 2.8 4.5 1.3]
[6.3 3.3 4.7 1.6]
[4.9 2.4 3.3 1.]
[6.6 2.9 4.6 1.3]
[5.2 2.7 3.9 1.4]
[5. 2. 3.5 1.]
[5.9 3. 4.2 1.5]
[6. 2.2 4. 1.]
[6.1 2.9 4.7 1.4]
[5.6 2.9 3.6 1.3]
[6.7 3.1 4.4 1.4]
[5.6 3. 4.5 1.5]
[5.8 2.7 4.1 1.]

[6.2 2.2 4.5 1.5]
[5.6 2.5 3.9 1.1]
[5.9 3.2 4.8 1.8]
[6.1 2.8 4. 1.3]
[6.3 2.5 4.9 1.5]
[6.1 2.8 4.7 1.2]
[6.4 2.9 4.3 1.3]
[6.6 3. 4.4 1.4]
[6.8 2.8 4.8 1.4]
[6.7 3. 5. 1.7]
[6. 2.9 4.5 1.5]
[5.7 2.6 3.5 1.]
[5.5 2.4 3.8 1.1]
[5.5 2.4 3.7 1.]
[5.8 2.7 3.9 1.2]
[6. 2.7 5.1 1.6]
[5.4 3. 4.5 1.5]
[6. 3.4 4.5 1.6]
[6.7 3.1 4.7 1.5]
[6.3 2.3 4.4 1.3]
[5.6 3. 4.1 1.3]
[5.5 2.5 4. 1.3]
[5.5 2.6 4.4 1.2]
[6.1 3. 4.6 1.4]
[5.8 2.6 4. 1.2]
[5. 2.3 3.3 1.]
[5.6 2.7 4.2 1.3]
[5.7 3. 4.2 1.2]
[5.7 2.9 4.2 1.3]
[6.2 2.9 4.3 1.3]
[5.1 2.5 3. 1.1]
[5.7 2.8 4.1 1.3]
[6.3 3.3 6. 2.5]
[5.8 2.7 5.1 1.9]
[7.1 3. 5.9 2.1]
[6.3 2.9 5.6 1.8]
[6.5 3. 5.8 2.2]
[7.6 3. 6.6 2.1]
[4.9 2.5 4.5 1.7]
[7.3 2.9 6.3 1.8]
[6.7 2.5 5.8 1.8]
[7.2 3.6 6.1 2.5]
[6.5 3.2 5.1 2.]
[6.4 2.7 5.3 1.9]
[6.8 3. 5.5 2.1]
[5.7 2.5 5. 2.]
[5.8 2.8 5.1 2.4]
[6.4 3.2 5.3 2.3]
[6.5 3. 5.5 1.8]
[7.7 3.8 6.7 2.2]
[7.7 2.6 6.9 2.3]

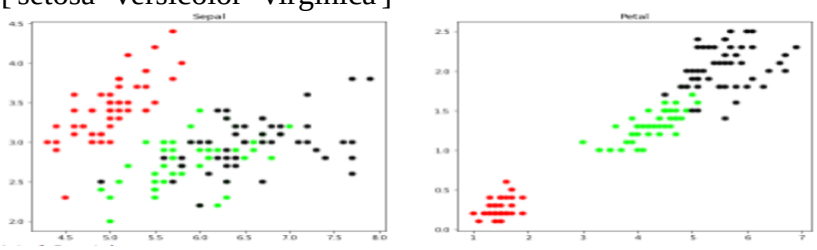
```
[6.  2.2 5.  1.5]
[6.9 3.2 5.7 2.3]
[5.6 2.8 4.9 2. ]
[7.7 2.8 6.7 2. ]
[6.3 2.7 4.9 1.8]
[6.7 3.3 5.7 2.1]
[7.2 3.2 6.  1.8]
[6.2 2.8 4.8 1.8]
[6.1 3.  4.9 1.8]
[6.4 2.8 5.6 2.1]
[7.2 3.  5.8 1.6]
[7.4 2.8 6.1 1.9]
[7.9 3.8 6.4 2. ]
[6.4 2.8 5.6 2.2]
[6.3 2.8 5.1 1.5]
[6.1 2.6 5.6 1.4]
[7.7 3.  6.1 2.3]
[6.3 3.4 5.6 2.4]
[6.4 3.1 5.5 1.8]
[6.  3.  4.8 1.8]
[6.9 3.1 5.4 2.1]
[6.7 3.1 5.6 2.4]
[6.9 3.1 5.1 2.3]
[5.8 2.7 5.1 1.9]
[6.8 3.2 5.9 2.3]
[6.7 3.3 5.7 2.5]
[6.7 3.  5.2 2.3]
[6.3 2.5 5.  1.9]
[6.5 3.  5.2 2. ]
[6.2 3.4 5.4 2.3]
[5.9 3.  5.1 1.8]]
```

IRIS FEATURES :
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']

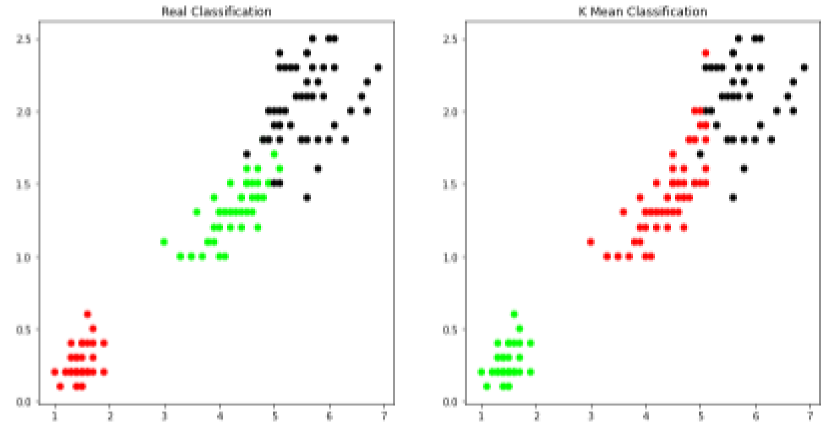
IRIS TARGET :

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

IRIS TARGET NAMES:
['setosa' 'versicolor' 'virginica']

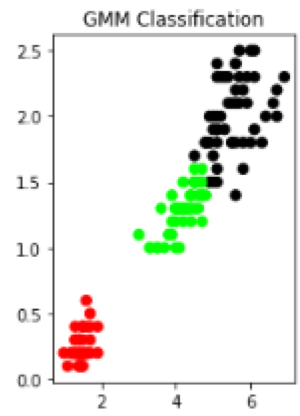


Actual Target is:
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2]



What KMeans thought:
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 2 2 1 2 2 2 2
2 2 1 1 2 2 2 2 1 2 1 2 1 2 2 1 1 2 2 2 2 2 1 2 2 2 2 1 2 2 2 1 2 2 2 1 2
2 1]
Accuracy of KMeans is 0.8933333333333333
Confusion Matrix for KMeans is
[[50 0 0]
[0 48 2]
[0 14 36]]

	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width
4	-1.021849	1.249201	-1.340227	-1.315444
93	-1.021849	-1.743357	-0.260315	-0.262387
32	-0.779513	2.400185	-1.283389	-1.447076
58	0.916837	-0.362176	0.478571	0.132510
88	-0.294842	-0.131979	0.194384	0.132510



What EM thought:
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 1 2 1
 1 1 1 2 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
Accuracy of EM is 0.9666666666666667
Confusion Matrix for EM is
 [[50 0 0]
 [0 45 5]
 [0 0 50]]

8. Write a program to implement k-Nearest Neighbor algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

```
#import the dataset and library files
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
import numpy as np
from sklearn.model_selection import train_test_split

iris_dataset=load_iris()

#display the iris dataset
print("\n IRIS FEATURES \ TARGET NAMES: \n ", iris_dataset.target_names)
for i in range(len(iris_dataset.target_names)):
   print("\n[{0}]:[{1}]".format(i,iris_dataset.target_names[i]))

print("\n IRIS DATA :\n",iris_dataset["data"])

#split the data into training and testing data
X_train, X_test, y_train, y_test = train_test_split(iris_dataset["data"], iris_dataset["target"]
, random_state=0)

print("\n Target :\n",iris_dataset["target"])
print("\n X TRAIN \n", X_train)
print("\n X TEST \n", X_test)
print("\n Y TRAIN \n", y_train)
print("\n Y TEST \n", y_test)

#train and fit the model
kn = KNeighborsClassifier(n_neighbors=5)
kn.fit(X_train, y_train)

#predicting from model
x_new = np.array([[5, 2.9, 1, 0.2]])
print("\n XNEW \n",x_new)
prediction = kn.predict(x_new)
print("\n Predicted target value: {}\n".format(prediction))
print("\n Predicted feature name: {}\n".format(iris_dataset["target_names"][prediction]))

i=1
x= X_test[i]
x_new = np.array([x])
print("\n XNEW \n",x_new)
```

```
for i in range(len(X_test)):
  x = X_test[i]
  x_new = np.array([x])
  prediction = kn.predict(x_new)
  print("\n Actual : {0} {1}, Predicted :{2}{3}".format(y_test[i],iris_dataset["target_names
"][y_test[i]],prediction,iris_dataset["target_names"][ prediction]))
print("\n TEST SCORE[ACCURACY]: {:.2f}\n".format(kn.score(X_test, y_test)))
```

OUTPUT:

```
IRIS FEATURES \ TARGET NAMES:
 ['setosa' 'versicolor' 'virginica']

[0]:[setosa]

[1]:[versicolor]

[2]:[virginica]

 IRIS DATA :
[[5.1 3.5 1.4 0.2]
[4.9 3.  1.4 0.2]
[4.7 3.2 1.3 0.2]
[4.6 3.1 1.5 0.2]
[5.  3.6 1.4 0.2]
[5.4 3.9 1.7 0.4]
[4.6 3.4 1.4 0.3]
[5.  3.4 1.5 0.2]
[4.4 2.9 1.4 0.2]
[4.9 3.1 1.5 0.1]
[5.4 3.7 1.5 0.2]
[4.8 3.4 1.6 0.2]
[4.8 3.  1.4 0.1]
[4.3 3.  1.1 0.1]
[5.8 4.  1.2 0.2]
[5.7 4.4 1.5 0.4]
[5.4 3.9 1.3 0.4]
[5.1 3.5 1.4 0.3]
[5.7 3.8 1.7 0.3]
[5.1 3.8 1.5 0.3]
[5.4 3.4 1.7 0.2]
[5.1 3.7 1.5 0.4]
[4.6 3.6 1.  0.2]
[5.1 3.3 1.7 0.5]
[4.8 3.4 1.9 0.2]
[5.  3.  1.6 0.2]
[5.  3.4 1.6 0.4]
[5.2 3.5 1.5 0.2]
```

[5.2 3.4 1.4 0.2]
[4.7 3.2 1.6 0.2]
[4.8 3.1 1.6 0.2]
[5.4 3.4 1.5 0.4]
[5.2 4.1 1.5 0.1]
[5.5 4.2 1.4 0.2]
[4.9 3.1 1.5 0.2]
[5. 3.2 1.2 0.2]
[5.5 3.5 1.3 0.2]
[4.9 3.6 1.4 0.1]
[4.4 3. 1.3 0.2]
[5.1 3.4 1.5 0.2]
[5. 3.5 1.3 0.3]
[4.5 2.3 1.3 0.3]
[4.4 3.2 1.3 0.2]
[5. 3.5 1.6 0.6]
[5.1 3.8 1.9 0.4]
[4.8 3. 1.4 0.3]
[5.1 3.8 1.6 0.2]
[4.6 3.2 1.4 0.2]
[5.3 3.7 1.5 0.2]
[5. 3.3 1.4 0.2]
[7. 3.2 4.7 1.4]
[6.4 3.2 4.5 1.5]
[6.9 3.1 4.9 1.5]
[5.5 2.3 4. 1.3]
[6.5 2.8 4.6 1.5]
[5.7 2.8 4.5 1.3]
[6.3 3.3 4.7 1.6]
[4.9 2.4 3.3 1.]
[6.6 2.9 4.6 1.3]
[5.2 2.7 3.9 1.4]
[5. 2. 3.5 1.]
[5.9 3. 4.2 1.5]
[6. 2.2 4. 1.]
[6.1 2.9 4.7 1.4]
[5.6 2.9 3.6 1.3]
[6.7 3.1 4.4 1.4]
[5.6 3. 4.5 1.5]
[5.8 2.7 4.1 1.]
[6.2 2.2 4.5 1.5]
[5.6 2.5 3.9 1.1]
[5.9 3.2 4.8 1.8]
[6.1 2.8 4. 1.3]
[6.3 2.5 4.9 1.5]
[6.1 2.8 4.7 1.2]
[6.4 2.9 4.3 1.3]
[6.6 3. 4.4 1.4]
[6.8 2.8 4.8 1.4]
[6.7 3. 5. 1.7]
[6. 2.9 4.5 1.5]

[5.7 2.6 3.5 1.]
[5.5 2.4 3.8 1.1]
[5.5 2.4 3.7 1.]
[5.8 2.7 3.9 1.2]
[6. 2.7 5.1 1.6]
[5.4 3. 4.5 1.5]
[6. 3.4 4.5 1.6]
[6.7 3.1 4.7 1.5]
[6.3 2.3 4.4 1.3]
[5.6 3. 4.1 1.3]
[5.5 2.5 4. 1.3]
[5.5 2.6 4.4 1.2]
[6.1 3. 4.6 1.4]
[5.8 2.6 4. 1.2]
[5. 2.3 3.3 1.]
[5.6 2.7 4.2 1.3]
[5.7 3. 4.2 1.2]
[5.7 2.9 4.2 1.3]
[6.2 2.9 4.3 1.3]
[5.1 2.5 3. 1.1]
[5.7 2.8 4.1 1.3]
[6.3 3.3 6. 2.5]
[5.8 2.7 5.1 1.9]
[7.1 3. 5.9 2.1]
[6.3 2.9 5.6 1.8]
[6.5 3. 5.8 2.2]
[7.6 3. 6.6 2.1]
[4.9 2.5 4.5 1.7]
[7.3 2.9 6.3 1.8]
[6.7 2.5 5.8 1.8]
[7.2 3.6 6.1 2.5]
[6.5 3.2 5.1 2.]
[6.4 2.7 5.3 1.9]
[6.8 3. 5.5 2.1]
[5.7 2.5 5. 2.]
[5.8 2.8 5.1 2.4]
[6.4 3.2 5.3 2.3]
[6.5 3. 5.5 1.8]
[7.7 3.8 6.7 2.2]
[7.7 2.6 6.9 2.3]
[6. 2.2 5. 1.5]
[6.9 3.2 5.7 2.3]
[5.6 2.8 4.9 2.]
[7.7 2.8 6.7 2.]
[6.3 2.7 4.9 1.8]
[6.7 3.3 5.7 2.1]
[7.2 3.2 6. 1.8]
[6.2 2.8 4.8 1.8]
[6.1 3. 4.9 1.8]
[6.4 2.8 5.6 2.1]
[7.2 3. 5.8 1.6]

```
[7.4 2.8 6.1 1.9]
[7.9 3.8 6.4 2. ]
[6.4 2.8 5.6 2.2]
[6.3 2.8 5.1 1.5]
[6.1 2.6 5.6 1.4]
[7.7 3.  6.1 2.3]
[6.3 3.4 5.6 2.4]
[6.4 3.1 5.5 1.8]
[6.  3.  4.8 1.8]
[6.9 3.1 5.4 2.1]
[6.7 3.1 5.6 2.4]
[6.9 3.1 5.1 2.3]
[5.8 2.7 5.1 1.9]
[6.8 3.2 5.9 2.3]
[6.7 3.3 5.7 2.5]
[6.7 3.  5.2 2.3]
[6.3 2.5 5.  1.9]
[6.5 3.  5.2 2. ]
[6.2 3.4 5.4 2.3]
[5.9 3.  5.1 1.8]]

Target :
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]

X TRAIN
[[5.9 3.  4.2 1.5]
[5.8 2.6 4.  1.2]
[6.8 3.  5.5 2.1]
[4.7 3.2 1.3 0.2]
[6.9 3.1 5.1 2.3]
[5.  3.5 1.6 0.6]
[5.4 3.7 1.5 0.2]
[5.  2.  3.5 1. ]
[6.5 3.  5.5 1.8]
[6.7 3.3 5.7 2.5]
[6.  2.2 5.  1.5]
[6.7 2.5 5.8 1.8]
[5.6 2.5 3.9 1.1]
[7.7 3.  6.1 2.3]
[6.3 3.3 4.7 1.6]
[5.5 2.4 3.8 1.1]
[6.3 2.7 4.9 1.8]
[6.3 2.8 5.1 1.5]
[4.9 2.5 4.5 1.7]
[6.3 2.5 5.  1.9]
[7.  3.2 4.7 1.4]
[6.5 3.  5.2 2. ]
```

[6. 3.4 4.5 1.6]
[4.8 3.1 1.6 0.2]
[5.8 2.7 5.1 1.9]
[5.6 2.7 4.2 1.3]
[5.6 2.9 3.6 1.3]
[5.5 2.5 4. 1.3]
[6.1 3. 4.6 1.4]
[7.2 3.2 6. 1.8]
[5.3 3.7 1.5 0.2]
[4.3 3. 1.1 0.1]
[6.4 2.7 5.3 1.9]
[5.7 3. 4.2 1.2]
[5.4 3.4 1.7 0.2]
[5.7 4.4 1.5 0.4]
[6.9 3.1 4.9 1.5]
[4.6 3.1 1.5 0.2]
[5.9 3. 5.1 1.8]
[5.1 2.5 3. 1.1]
[4.6 3.4 1.4 0.3]
[6.2 2.2 4.5 1.5]
[7.2 3.6 6.1 2.5]
[5.7 2.9 4.2 1.3]
[4.8 3. 1.4 0.1]
[7.1 3. 5.9 2.1]
[6.9 3.2 5.7 2.3]
[6.5 3. 5.8 2.2]
[6.4 2.8 5.6 2.1]
[5.1 3.8 1.6 0.2]
[4.8 3.4 1.6 0.2]
[6.5 3.2 5.1 2.]
[6.7 3.3 5.7 2.1]
[4.5 2.3 1.3 0.3]
[6.2 3.4 5.4 2.3]
[4.9 3. 1.4 0.2]
[5.7 2.5 5. 2.]
[6.9 3.1 5.4 2.1]
[4.4 3.2 1.3 0.2]
[5. 3.6 1.4 0.2]
[7.2 3. 5.8 1.6]
[5.1 3.5 1.4 0.3]
[4.4 3. 1.3 0.2]
[5.4 3.9 1.7 0.4]
[5.5 2.3 4. 1.3]
[6.8 3.2 5.9 2.3]
[7.6 3. 6.6 2.1]
[5.1 3.5 1.4 0.2]
[4.9 3.1 1.5 0.2]
[5.2 3.4 1.4 0.2]
[5.7 2.8 4.5 1.3]
[6.6 3. 4.4 1.4]
[5. 3.2 1.2 0.2]

[5.1 3.3 1.7 0.5]
[6.4 2.9 4.3 1.3]
[5.4 3.4 1.5 0.4]
[7.7 2.6 6.9 2.3]
[4.9 2.4 3.3 1.]
[7.9 3.8 6.4 2.]
[6.7 3.1 4.4 1.4]
[5.2 4.1 1.5 0.1]
[6. 3. 4.8 1.8]
[5.8 4. 1.2 0.2]
[7.7 2.8 6.7 2.]
[5.1 3.8 1.5 0.3]
[4.7 3.2 1.6 0.2]
[7.4 2.8 6.1 1.9]
[5. 3.3 1.4 0.2]
[6.3 3.4 5.6 2.4]
[5.7 2.8 4.1 1.3]
[5.8 2.7 3.9 1.2]
[5.7 2.6 3.5 1.]
[6.4 3.2 5.3 2.3]
[6.7 3. 5.2 2.3]
[6.3 2.5 4.9 1.5]
[6.7 3. 5. 1.7]
[5. 3. 1.6 0.2]
[5.5 2.4 3.7 1.]
[6.7 3.1 5.6 2.4]
[5.8 2.7 5.1 1.9]
[5.1 3.4 1.5 0.2]
[6.6 2.9 4.6 1.3]
[5.6 3. 4.1 1.3]
[5.9 3.2 4.8 1.8]
[6.3 2.3 4.4 1.3]
[5.5 3.5 1.3 0.2]
[5.1 3.7 1.5 0.4]
[4.9 3.1 1.5 0.1]
[6.3 2.9 5.6 1.8]
[5.8 2.7 4.1 1.]
[7.7 3.8 6.7 2.2]
[4.6 3.2 1.4 0.2]]

X TEST
[[5.8 2.8 5.1 2.4]
[6. 2.2 4. 1.]
[5.5 4.2 1.4 0.2]
[7.3 2.9 6.3 1.8]
[5. 3.4 1.5 0.2]
[6.3 3.3 6. 2.5]
[5. 3.5 1.3 0.3]
[6.7 3.1 4.7 1.5]
[6.8 2.8 4.8 1.4]
[6.1 2.8 4. 1.3]

[6.1 2.6 5.6 1.4]
[6.4 3.2 4.5 1.5]
[6.1 2.8 4.7 1.2]
[6.5 2.8 4.6 1.5]
[6.1 2.9 4.7 1.4]
[4.9 3.6 1.4 0.1]
[6. 2.9 4.5 1.5]
[5.5 2.6 4.4 1.2]
[4.8 3. 1.4 0.3]
[5.4 3.9 1.3 0.4]
[5.6 2.8 4.9 2.]
[5.6 3. 4.5 1.5]
[4.8 3.4 1.9 0.2]
[4.4 2.9 1.4 0.2]
[6.2 2.8 4.8 1.8]
[4.6 3.6 1. 0.2]
[5.1 3.8 1.9 0.4]
[6.2 2.9 4.3 1.3]
[5. 2.3 3.3 1.]
[5. 3.4 1.6 0.4]
[6.4 3.1 5.5 1.8]
[5.4 3. 4.5 1.5]
[5.2 3.5 1.5 0.2]
[6.1 3. 4.9 1.8]
[6.4 2.8 5.6 2.2]
[5.2 2.7 3.9 1.4]
[5.7 3.8 1.7 0.3]
[6. 2.7 5.1 1.6]]

Y TRAIN
[1 1 2 0 2 0 0 1 2 2 2 2 1 2 1 1 2 2 2 2 1 2 1 0 2 1 1 1 1 2 0 0 2 1 0 0 1
0 2 1 0 1 2 1 0 2 2 2 2 0 0 2 2 0 2 0 2 2 0 0 2 0 0 0 1 2 2 0 0 0 1 1 0 0
1 0 2 1 2 1 0 2 0 2 0 0 2 0 2 1 1 1 2 2 1 1 0 1 2 2 0 1 1 1 1 0 0 0 2 1 2
0]

Y TEST
[2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0
1]

XNEW
[[5. 2.9 1. 0.2]]

Predicted target value: [0]

Predicted feature name: ['setosa']

XNEW
[[6. 2.2 4. 1.]]

Actual : 2 virginica, Predicted :[2]['virginica']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 0 setosa, Predicted :[0]['setosa']

Actual : 2 virginica, Predicted :[2]['virginica']

Actual : 0 setosa, Predicted :[0]['setosa']

Actual : 2 virginica, Predicted :[2]['virginica']

Actual : 0 setosa, Predicted :[0]['setosa']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 2 virginica, Predicted :[2]['virginica']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 0 setosa, Predicted :[0]['setosa']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 0 setosa, Predicted :[0]['setosa']

Actual : 0 setosa, Predicted :[0]['setosa']

Actual : 2 virginica, Predicted :[2]['virginica']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 0 setosa, Predicted :[0]['setosa']

Actual : 0 setosa, Predicted :[0]['setosa']

Actual : 2 virginica, Predicted :[2]['virginica']

Actual : 0 setosa, Predicted :[0]['setosa']

Actual : 0 setosa, Predicted :[0]['setosa']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 0 setosa, Predicted :[0]['setosa']

Actual : 2 virginica, Predicted :[2]['virginica']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 0 setosa, Predicted :[0]['setosa']

Actual : 2 virginica, Predicted :[2]['virginica']

Actual : 2 virginica, Predicted :[2]['virginica']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 0 setosa, Predicted :[0]['setosa']

Actual : 1 versicolor, Predicted :[2]['virginica']

TEST SCORE[ACCURACY]: 0.97

9. Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

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

def local_regression(x0, X, Y, tau):
    x0 = [1, x0]
    X = [[1, i] for i in X]
    X = np.asarray(X)
    xw = (X.T) * np.exp(np.sum((X - x0) ** 2, axis=1) / (-2 * tau))
    beta = np.linalg.pinv(xw @ X) @ xw @ Y @ x0
    return beta

def draw(tau):
    prediction = [local_regression(x0, X, Y, tau) for x0 in domain]
    plt.plot(X, Y, 'o', color='black')
    plt.plot(domain, prediction, color='red')
    plt.show()

X = np.linspace(-3, 3, num=1000)
domain = X
Y = np.log(np.abs(X ** 2 - 1) + .5)

draw(10)
draw(0.1)
```

OUTPUT: