RAO BAHADUR Y MAHABALESWARAPPA ENGINEERING COLLEGE, BELLARY

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

LAB MANNUAL

ARTIFICIAL AND MACHINE LEARNING LABORATORY

SUB CODE- 18CSL76

VII SEM

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LABORATORY (Effective from the academic year 2018 -2019) SEMESTER – VII			
Course Code	**18CSL76**	**CIE Marks**	40
Number of Contact Hours/Week	0:0:2	**SEE Marks**	60
Total Number of Lab Contact Hours	36	**Exam Hours**	03
Credits – 2			

Course Learning Objectives: This course (18CSL76) will enable students to:

- Implement and evaluate AI and ML algorithms in and Python programming language.

Descriptions (if any):

Installation procedure of the required software must be demonstrated, carried out in groups and documented in the journal.

Programs List:

1.	Implement A* Search algorithm.
2.	Implement AO* Search algorithm.
3.	For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithmto output a description of the set of all hypotheses consistent with the training examples.
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 toclassify a new sample.
5.	Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.
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.
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.
8.	Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.
9.	Implement the non-parametric Locally Weighted Regressionalgorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs

Laboratory Outcomes: The student should be able to:

- Implement and demonstrate AI and ML algorithms.
- Evaluate different algorithms.

Conduct of Practical Examination:

- Experiment distribution
  - For laboratories having only one part: Students are allowed to pick one experiment from the lot with equal opportunity.
  - For laboratories having PART A and PART B: Students are allowed to pick one experiment from PART A and one experiment from PART B, with equal opportunity.
- Change of experiment is allowed only once and marks allotted for procedure to be made zero of the changed part only.
- Marks Distribution *(Courseed to change in accoradance with university regulations)*
  - q) For laboratories having only one part – Procedure + Execution + Viva-Voce: 15+70+15 = 100 Marks
  - r) For laboratories having PART A and PART B
    - i. Part A – Procedure + Execution + Viva = 6 + 28 + 6 = 40 Marks
    - ii. Part B – Procedure + Execution + Viva = 9 + 42 + 9 = 60 Marks

Program 1.A* SEARCH ALGORITHM

```
"""Implement A* search algorithm"""

defaStarAlgo(start_node, stop_node):
open_set = set(start_node)
closed_set = set()
    g = {}
    parents = {}

    g[start_node] = 0
    parents[start_node] = start_node
    while len(open_set) > 0:
        n = None
        for v in open_set:
            if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):
                n = v
        if n == stop_node or Graph_nodes[n] is None:
            pass
        else:
            for (m, weight) in get_neighbours(n):
                if m not in open_set and m not in closed_set:
open_set.add(m)
                    parents[m] = n
                    g[m] = g[n] + weight
                else:
                    if g[m] > g[n] + weight:
                        g[m] = g[n] + weight
                        parents[m] = n
                    if m in closed_set:
closed_set.remove(m)
open_set.add(m)
        if n is None:
print('path does not exist!')
            return None
        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
open_set.remove(n)
closed_set.add(n)
print('path does not exist!')
    return None


defget_neighbours(v):
    if v in Graph_nodes:
```

```
        return Graph_nodes[v]
    else:
        return None


def heuristic(n):
H_dist = {
        'A': 10,
        'B': 8,
        'C': 5,
        'D': 7,
        'E': 3,
        'F': 6,
        'G': 5,
        'H': 3,
        'I': 1,
        'J': 0
    }
    return H_dist[n]


Graph_nodes = {
    'A': [('B', 6), ('F', 3)],
    'B': [('C', 3), ('D', 2)],
    'C': [('D', 1), ('E', 5)],
    'D': [('C', 1), ('E', 8)],
    'E': [('I', 5), ('J', 5)],
    'F': [('G', 1), ('H', 7)],
    'G': [('I', 3)],
    'H': [('I', 2)],
    'I': [('E', 5), ('J', 3)]
}

aStarAlgo('A', 'J')
```

Out put

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

PROGRAM -2

A+O_* SEARCHALGORITHM

```
"""Recursive implementation of AO* algorithm"""

class Graph:
def __init__(self, graph, heuristicNodeList, startNode):

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

defapplyAOStar(self):
self.aoStar(self.start, False)

defgetNeighbors(self, v):
        return self.graph.get(v, '')

defgetStatus(self, v):
        return self.status.get(v, 0)

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

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

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

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

defcomputeMinimumCostChildNodes(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
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

defaoStar(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 not backTracking:  # 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 respective 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']}
------------------------------------------------------------
```

Program 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.

Algorithm:

G ← maximally general hypotheses in H
S ← maximally specific hypotheses in H
For each training example d=<x,c(x)>
Case 1 : If d is a positive example

Remove from G any hypothesis that is inconsistent with d
For each hypothesis s in S that is not consistent with d

- *Remove s from S.*
- *Add to S all minimal generalizations h of s such that*
    - *h consistent with d*
    - *Some member of G is more general than h*
- *Remove from S any hypothesis that is more general than another hypothesis in S*

Case 2: If d is a negative example

Remove from S any hypothesis that is inconsistent with d
For each hypothesis g in G that is not consistent with d

- *Remove g from G.*
- *Add to G all minimal specializations h of g such that*
    - *h consistent with d*
    - *Some member of S is more specific than h*
- *Remove from G any hypothesis that is less general than another hypothesis in G*

Program Code:

```
import random
import csv
```

```
def g_0(n):
    return ("?",)*n
def s_0(n):
    return ('0',)*n
```

```
ef more_general(h1, h2):
    more_general_parts = []
    for x, y in zip(h1, h2):
        mg = x == "?" or (x != "0" and (x == y or y == "0"))
        more_general_parts.append(mg)
    return all(more_general_parts)
l1 = [1, 2, 3]
l2 = [3, 4, 5]
list(zip(l1, l2))
```

```
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] != '0' else x[i]
    return [tuple(h_new)]
```

```
min_generalizations(h=('0', '0'  , 'sunny'),
                    x=('rainy', 'windy', 'cloudy'))
```

```
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] != "0":
            h_new = h[:i] + ('0',) + h[i+1:]
            results.append(h_new)
    return results
```

```
min_specializations(h=('?', 'x',),
                    domains=[['a', 'b', 'c'], ['x', 'y']],
                    x=('b', 'x'))
```

```
with open('C:\\Users\\Desktop\\c1.csv')  as csvFile:
        examples = [tuple(line) for line in
csv.reader(csvFile)]
examples
```

```
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]
get_domains(examples)
```

```
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]
        if cx=='Y':
            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)
            S.update([h for h in Splus if
any([more_general(g,h)
                                                        for g in G])])
            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)
            G.update([h for h in Gminus if
any([more_general(h, s)
                                                 for s in S])])
            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)
```

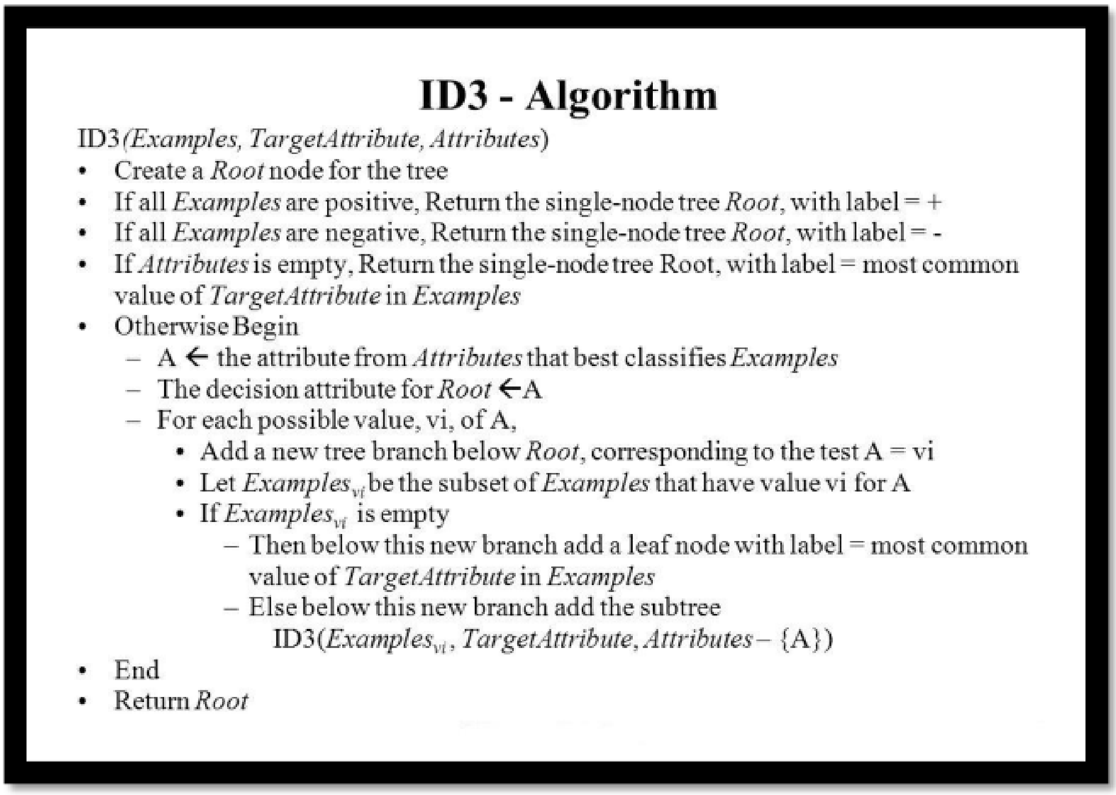
OUTPUT

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

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

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

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

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

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

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

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

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

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

Note: save the file c1.csv on desktop in your folder and change the path of file name in open() function in the program code

Program4: 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 classifya new sample.

Algorithm :



ID3 - Algorithm

ID3(*Examples*, *TargetAttribute*, *Attributes*)

- Create a *Root* node for the tree
- If all *Examples* are positive, Return the single-node tree *Root*, with label = +
- If all *Examples* are negative, Return the single-node tree *Root*, with label = -
- If *Attributes* is empty, Return the single-node tree Root, with label = most common value of *TargetAttribute* in *Examples*
- Otherwise Begin
  - A ← the attribute from *Attributes* that best classifies *Examples*
  - The decision attribute for *Root* ←A
  - For each possible value, vi, of A,
    - Add a new tree branch below *Root*, corresponding to the test A = vi
    - Let $Examples_{vi}$ be the subset of *Examples* that have value vi for A
    - If $Examples_{vi}$ is empty
      - Then below this new branch add a leaf node with label = most common value of *TargetAttribute* in *Examples*
      - Else below this new branch add the subtree ID3($Examples_{vi}$, *TargetAttribute*, *Attributes* – {A})
- End
- Return *Root*

Program Code:

```
import pandas as pd
from pandas import DataFrame
df_tennis = DataFrame.from_csv('C:\\Users\\ISE\\Desktop\\Python-
Decision-Tree-Using-ID3-master\\PlayTennis.csv')
print("\n Given Play Tennis Data Set:\n\n", df_tennis)
df_tennis.keys()[0]
```

Entropy of the Training Data Set

```
def entropy(probs):
    import math
    return sum( [-prob*math.log(prob, 2) for prob in probs] )


def entropy_of_list(a_list):

    from collections import Counter
    cnt = Counter(x for x in a_list)
```

```
    num_instances = len(a_list)*1.0
    print("\n Number of Instances of the Current Sub Class
is{0}:".format(num_instances ))
    probs = [x / num_instances for x in cnt.values()]
    print("\n Classes:",min(cnt),max(cnt))
    print(" \n Probabilities of Class {0} is
{1}:".format(min(cnt),min(probs)))
    print(" \n Probabilities of Class {0} is
{1}:".format(max(cnt),max(probs)))
    return entropy(probs) # Call Entropy :


print("\n  INPUT DATA SET FOR ENTROPY CALCULATION:\n",
df_tennis['PlayTennis'])

total_entropy = entropy_of_list(df_tennis['PlayTennis'])

print("\n Total Entropy of PlayTennis Data Set:",total_entropy)
```

Information Gain of Attributes

```
def information_gain(df, split_attribute_name, target_attribute_name, trace=0):
    print("Information Gain Calculation of ",split_attribute_name)
    df_split = df.groupby(split_attribute_name)
    nobs = len(df.index) * 1.0
    df_agg_ent = df_split.agg({target_attribute_name : [entropy_of_list, lambda x: len(x)/nobs]
})[target_attribute_name]
    df_agg_ent.columns = ['Entropy', 'PropObservations']
    new_entropy = sum( df_agg_ent['Entropy'] * df_agg_ent['PropObservations'] )
    old_entropy = entropy_of_list(df[target_attribute_name])
    return old_entropy - new_entropy
print('Info-gain for Outlook is :'+str( information_gain(df_tennis, 'Outlook', 'PlayTennis')),"\n")
print('\n Info-gain for Humidity is: ' + str( information_gain(df_tennis, 'Humidity', 'PlayTennis')),"\n")
print('\n Info-gain for Wind is:' + str( information_gain(df_tennis, 'Wind', 'PlayTennis')),"\n")
print('\n Info-gain for Temperature is:' + str( information_gain(df_tennis, 'Temperature','PlayTennis')),"\n")
```

```
def id3(df, target_attribute_name, attribute_names, default_class=None):

    from collections import Counter
    cnt = Counter(x for x in df[target_attribute_name])

    if len(cnt) == 1:
        return next(iter(cnt))

    elif df.empty or (not attribute_names):
        return default_class
```

```
    else:

        default_class = max(cnt.keys())

        gainz = [information_gain(df, attr, target_attribute_name) for
attr in attribute_names] #
        index_of_max = gainz.index(max(gainz)) # Index of Best Attribute

        best_attr = attribute_names[index_of_max]


        tree = {best_attr:{}}
        remaining_attribute_names = [i for i in attribute_names if i !=
best_attr]

        for attr_val, data_subset in df.groupby(best_attr):
            subtree = id3(data_subset,
                        target_attribute_name,
                        remaining_attribute_names,
                        default_class)
            tree[best_attr][attr_val] = subtree
        return tree
```

ID3 Algorithm

Predicting Attributes

```
attribute_names = list(df_tennis.columns)
print("List of Attributes:", attribute_names)
attribute_names.remove('PlayTennis')
print("Predicting Attributes:", attribute_names)
```

Tree Construction

```
from pprint import pprint
tree = id3(df_tennis,'PlayTennis',attribute_names)
print("\n\nThe Resultant Decision Tree is :\n")
pprint(tree)
attribute = next(iter(tree))
print("Best Attribute :\n",attribute)
print("Tree Keys:\n",tree[attribute].keys())
```

Classification Accuracy

```
def classify(instance, tree, default=None): # Instance of Play Tennis with
Predicted

    attribute = next(iter(tree))
    print("Key:",tree.keys())
    print("Attribute:",attribute)

    if instance[attribute] in tree[attribute].keys():
        result = tree[attribute][instance[attribute]]
```

```
        print("Instance 
Attribute:",instance[attribute],"TreeKeys:",tree[attribute].keys())
        if isinstance(result, dict):
            return classify(instance, result)
        else:
            return result
    else:
        return default
```

```
df_tennis['predicted'] = df_tennis.apply(classify, axis=1, args=(tree,'No')
)
    # classify func allows for a default arg: when tree doesn't have answer 
for a particular
    # combitation of attribute-values, we can use 'no' as the default guess

print(df_tennis['predicted'])

print('\n Accuracy is:\n' + str(
sum(df_tennis['PlayTennis']==df_tennis['predicted'] ) /
(1.0*len(df_tennis.index)) ))

df_tennis[['PlayTennis', 'predicted']]
```

OUTPUT:

```
Information Gain Calculation of  Outlook
```

```
Overcast
  PlayTennis   Outlook Temperature Humidity    Wind predicted
2          Yes  Overcast          Hot      High    Weak         Yes
6          Yes  Overcast         Cool    Normal  Strong         Yes
```

```
Rain
  PlayTennis Outlook Temperature Humidity    Wind predicted
3          Yes    Rain        Mild     High    Weak         Yes
4          Yes    Rain        Cool   Normal    Weak         Yes
5           No    Rain        Cool   Normal  Strong          No
9          Yes    Rain        Mild   Normal    Weak         Yes
Sunny
  PlayTennis Outlook Temperature Humidity    Wind predicted
1           No   Sunny         Hot     High  Strong          No
7           No   Sunny        Mild     High    Weak          No
8          Yes   Sunny        Cool   Normal    Weak         Yes
No and Yes Classes: PlayTennis Counter({'Yes': 2})
No and Yes Classes: PlayTennis Counter({'Yes': 3, 'No': 1})
No and Yes Classes: PlayTennis Counter({'No': 2, 'Yes': 1})
No and Yes Classes: PlayTennis Counter({'Yes': 6, 'No': 3})
Information Gain Calculation of  Temperature
```

```
Cool
  PlayTennis   Outlook Temperature Humidity    Wind predicted
4          Yes      Rain        Cool   Normal    Weak         Yes
5           No      Rain        Cool   Normal  Strong          No
6          Yes  Overcast        Cool   Normal  Strong         Yes
8          Yes     Sunny        Cool   Normal    Weak         Yes
Hot
  PlayTennis   Outlook Temperature Humidity    Wind predicted
1           No     Sunny         Hot     High  Strong          No
2          Yes  Overcast         Hot     High    Weak         Yes
```

```
Mild
  PlayTennis Outlook Temperature Humidity  Wind predicted
3        Yes     Rain        Mild     High  Weak        Yes
7         No    Sunny        Mild     High  Weak         No
9        Yes     Rain        Mild   Normal  Weak        Yes
No and Yes Classes: PlayTennis Counter({'Yes': 3, 'No': 1})
No and Yes Classes: PlayTennis Counter({'No': 1, 'Yes': 1})
No and Yes Classes: PlayTennis Counter({'Yes': 2, 'No': 1})
No and Yes Classes: PlayTennis Counter({'Yes': 6, 'No': 3})
Information Gain Calculation of  Humidity
High
  PlayTennis   Outlook Temperature Humidity    Wind predicted
1         No     Sunny         Hot     High  Strong         No
2        Yes  Overcast         Hot     High    Weak        Yes
3        Yes      Rain        Mild     High    Weak        Yes
7         No     Sunny        Mild     High    Weak         No
Normal
  PlayTennis   Outlook Temperature Humidity    Wind predicted
4        Yes      Rain        Cool   Normal    Weak        Yes
5         No      Rain        Cool   Normal  Strong         No
6        Yes  Overcast        Cool   Normal  Strong        Yes
8        Yes     Sunny        Cool   Normal    Weak        Yes
9        Yes      Rain        Mild   Normal    Weak        Yes
No and Yes Classes: PlayTennis Counter({'No': 2, 'Yes': 2})
No and Yes Classes: PlayTennis Counter({'Yes': 4, 'No': 1})
No and Yes Classes: PlayTennis Counter({'Yes': 6, 'No': 3})
Information Gain Calculation of  Wind
Strong
  PlayTennis   Outlook Temperature Humidity    Wind predicted
1         No     Sunny         Hot     High  Strong         No
5         No      Rain        Cool   Normal  Strong         No
6        Yes  Overcast        Cool   Normal  Strong        Yes
Weak
  PlayTennis   Outlook Temperature Humidity  Wind predicted
2        Yes  Overcast         Hot     High  Weak        Yes
3        Yes      Rain        Mild     High  Weak        Yes
4        Yes      Rain        Cool   Normal  Weak        Yes
7         No     Sunny        Mild     High  Weak         No
8        Yes     Sunny        Cool   Normal  Weak        Yes
9        Yes      Rain        Mild   Normal  Weak        Yes
No and Yes Classes: PlayTennis Counter({'No': 2, 'Yes': 1})
No and Yes Classes: PlayTennis Counter({'Yes': 5, 'No': 1})
No and Yes Classes: PlayTennis Counter({'Yes': 6, 'No': 3})
Information Gain Calculation of  Temperature
Cool
```

```
  PlayTennis Outlook Temperature Humidity    Wind predicted
4          Yes     Rain          Cool    Normal    Weak          Yes
5           No     Rain          Cool    Normal  Strong           No
Mild
  PlayTennis Outlook Temperature Humidity  Wind predicted
3          Yes     Rain          Mild      High  Weak          Yes
9          Yes     Rain          Mild    Normal  Weak          Yes
No and Yes Classes: PlayTennis Counter({'Yes': 1, 'No': 1})
No and Yes Classes: PlayTennis Counter({'Yes': 2})
No and Yes Classes: PlayTennis Counter({'Yes': 3, 'No': 1})
Information Gain Calculation of  Humidity
High
  PlayTennis Outlook Temperature Humidity  Wind predicted
3          Yes     Rain          Mild      High  Weak          Yes
Normal
```

```
  PlayTennis Outlook Temperature Humidity    Wind predicted
4          Yes     Rain          Cool    Normal    Weak          Yes
```

```
5           No     Rain          Cool    Normal  Strong           No
9          Yes     Rain          Mild    Normal    Weak          Yes
No and Yes Classes: PlayTennis Counter({'Yes': 1})
No and Yes Classes: PlayTennis Counter({'Yes': 2, 'No': 1})
No and Yes Classes: PlayTennis Counter({'Yes': 3, 'No': 1})
Information Gain Calculation of  Wind
Strong
  PlayTennis Outlook Temperature Humidity    Wind predicted
5           No     Rain          Cool    Normal  Strong           No
Weak
  PlayTennis Outlook Temperature Humidity  Wind predicted
3          Yes     Rain          Mild      High  Weak          Yes
4          Yes     Rain          Cool    Normal  Weak          Yes
9          Yes     Rain          Mild    Normal  Weak          Yes
No and Yes Classes: PlayTennis Counter({'No': 1})
No and Yes Classes: PlayTennis Counter({'Yes': 3})
No and Yes Classes: PlayTennis Counter({'Yes': 3, 'No': 1})
Information Gain Calculation of  Temperature
```

```
  PlayTennis Outlook Temperature Humidity  Wind predicted
3        Yes    Rain        Mild     High  Weak       Yes
4        Yes    Rain        Cool   Normal  Weak       Yes
9        Yes    Rain        Mild   Normal  Weak       Yes
No and Yes Classes: PlayTennis Counter({'No': 1})
No and Yes Classes: PlayTennis Counter({'Yes': 3})
No and Yes Classes: PlayTennis Counter({'Yes': 3, 'No': 1})
Information Gain Calculation of  Temperature
Cool
  PlayTennis Outlook Temperature Humidity  Wind predicted
8        Yes   Sunny        Cool   Normal  Weak       Yes
Hot
  PlayTennis Outlook Temperature Humidity    Wind predicted
1         No   Sunny         Hot     High  Strong         No
Mild
  PlayTennis Outlook Temperature Humidity  Wind predicted
```

```
  PlayTennis Outlook Temperature Humidity  Wind predicted
7         No   Sunny        Mild     High  Weak        No
No and Yes Classes: PlayTennis Counter({'Yes': 1})
No and Yes Classes: PlayTennis Counter({'No': 1})
No and Yes Classes: PlayTennis Counter({'No': 1})
No and Yes Classes: PlayTennis Counter({'No': 2, 'Yes': 1})
Information Gain Calculation of  Humidity
High
  PlayTennis Outlook Temperature Humidity    Wind predicted
1         No   Sunny         Hot     High  Strong         No
7         No   Sunny        Mild     High    Weak         No
Normal
  PlayTennis Outlook Temperature Humidity  Wind predicted
8        Yes   Sunny        Cool   Normal  Weak       Yes
No and Yes Classes: PlayTennis Counter({'No': 2})
No and Yes Classes: PlayTennis Counter({'Yes': 1})
No and Yes Classes: PlayTennis Counter({'No': 2, 'Yes': 1})
Information Gain Calculation of  Wind
```

```
Strong
  PlayTennis Outlook Temperature Humidity    Wind predicted
1          No    Sunny          Hot      High  Strong         No
Weak
  PlayTennis Outlook Temperature Humidity  Wind predicted
7          No    Sunny         Mild      High  Weak         No
8         Yes    Sunny         Cool    Normal  Weak        Yes
No and Yes Classes: PlayTennis Counter({'No': 1})
No and Yes Classes: PlayTennis Counter({'No': 1, 'Yes': 1})
No and Yes Classes: PlayTennis Counter({'No': 2, 'Yes': 1})
```

```
Accuracy is : 0.75
```

Program5: Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets

Algorithm:

function BackProp $(D, \eta, n_{in}, n_{hidden}, n_{out})$

- D is the training set consists of m pairs: $\{(x_i, y_i)^m\}$
- η is the learning rate as an example (0.1)
- n_{in}, n_{hidden} e n_{out} are the numbero of imput hidden and output unit of neural network

Make a feed-forward network with n_{in}, n_{hidden} e n_{out} units
Initialize all the weight to short randomly number (es. [-0.05 0.05])
Repeat until termination condition are verifyed:
For any sample in D:
Forward propagate the network computing the output o_u of every unit u of the network
Back propagate the errors onto the network:
– For every output unit k, compute the error δ_k: $\delta_k = o_k(1-o_k)(t_k - o_k)$
– For every hidden unit h compute the error δ_h: $\delta_h = o_h(1-o_h) \sum_{k \in outputs} w_{kh}\delta_k$
– Update the network weight w_{ji}: $w_{ji} = w_{ji} + \Delta w_{ji}$, $where \ \Delta w_{ji} = \eta \delta_j x_{ji}$

(x_{ji} is the input of unit j from coming from unit i)

The Backpropagation Algorithm for a feed-forward 2-layer network of sigmoid units, the stochastic version

Program Code:

```
import numpy as np

# X = (hours studying, hours sleeping), y = score on test,
xPredicted = 4 hours studying & 8 hours sleeping (input data for
prediction)
X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)
xPredicted = np.array(([4,8]), dtype=float)

X = X/np.amax(X, axis=0)
xPredicted = xPredicted/np.amax(xPredicted, axis=0)
y = y/100 # max test score is 100

class Neural_Network(object):
    def __init__(self):
        self.inputSize = 2
        self.outputSize = 1
        self.hiddenSize = 3
        self.W1 = np.random.randn(self.inputSize, self.hiddenSize)
# (3x2) weight matrix from input to hidden layer
        self.W2 = np.random.randn(self.hiddenSize, self.outputSize)
# (3x1) weight matrix from hidden to output layer

    def forward(self, X):
        self.z = np.dot(X, self.W1)
        self.z2 = self.sigmoid(self.z) # activation function
```

```
        self.z3 = np.dot(self.z2, self.W2) # dot product of hidden
layer (z2)andsecondsetof 3x1 weights
        o = self.sigmoid(self.z3) # final activation function
        return o
    def sigmoid(self, s):
        return 1/(1+np.exp(-s))

    def sigmoidPrime(self, s):
        return s * (1 - s)

    def backward(self, X, y, o):
        self.o_error = y - o # error in output
        self.o_delta = self.o_error*self.sigmoidPrime(o) # applying
derivative of sigmoid to error

        self.z2_error = self.o_delta.dot(self.W2.T) # z2 error: how
much our hidden layer weights contributed to output error
        self.z2_delta = self.z2_error*self.sigmoidPrime(self.z2) #
applying derivative of sigmoid to z2 error

        self.W1 += X.T.dot(self.z2_delta) # adjusting first set
(input --> hidden) weights
        self.W2 += self.z2.T.dot(self.o_delta) # adjusting second
set (hidden --> output) weights

    def train(self, X, y):
        o = self.forward(X)
        self.backward(X, y, o)

    def saveWeights(self):
        np.savetxt("w1.txt", self.W1, fmt="%s")
        np.savetxt("w2.txt", self.W2, fmt="%s")

    def predict(self):
        print("Predicted data based on trained weights: ")
        print("Input (scaled): \n" + str(xPredicted))
        print("Output: \n" + str(self.forward(xPredicted)))

NN = Neural_Network()
for i in range(1000): # trains the NN 1,000 times
    print("# " + str(i) + "\n")
    print("Input (scaled): \n" + str(X))
    print("Actual Output: \n" + str(y))
    print("Predicted Output: \n" + str(NN.forward(X)))
    print("Loss: \n" + str(np.mean(np.square(y - NN.forward(X)))))
# mean sum squared loss
    print("\n")

    NN.train(X, y)
    NN.saveWeights()
    NN.predict()
```

```
 [0.89]]
Predicted Output:
[[0.90427003]
 [0.87150692]
 [0.89246923]]
Loss:
0.0001286460894599845


Predicted data based on trained weights:
Input (scaled):
[0.5 1. ]
Output:
[0.90052261]
# 999

Input (scaled):
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
[[0.9042714 ]
 [0.87150719]
 [0.8924675 ]]
Loss:
0.0001286309334927327


Predicted data based on trained weights:
Input (scaled):
[0.5 1. ]
Output:
[0.90052454]
```

Program5: 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.

Bayesian Theorem:

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

- $P(h)$ = prior probability of hypothesis h
- $P(D)$ = prior probability of training data D
- $P(h|D)$ = probability of h given D
- $P(D|h)$ = probability of D given h

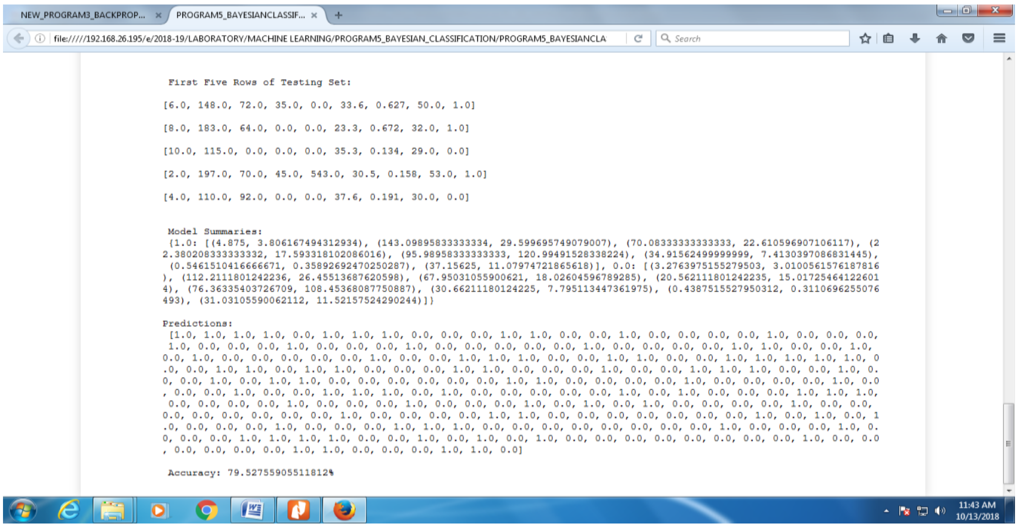
Program Code:

```
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 = []
    copy = list(dataset)
    while len(trainSet) < trainSize:
        index = random.randrange(len(copy)) # random index
        trainSet.append(copy.pop(index))
    return [trainSet, copy]
def separateByClass(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)
```

```
    return separated
 def mean(numbers):
    return sum(numbers)/float(len(numbers))

def stdev(numbers):
    avg = mean(numbers)
    variance = sum([pow(x-avg,2) for x in numbers])/float(len(numbers)-1)
    return math.sqrt(variance)

def summarize(dataset):
    summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*dataset)]
    del summaries[-1]
    return summaries

def summarizeByClass(dataset):
    separated = separateByClass(dataset)
    summaries = {}
    for classValue, instances in separated.items():
        summaries[classValue] = summarize(instances)
    return summaries

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

def calculateClassProbabilities(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)
    return probabilities

def predict(summaries, inputVector):
    probabilities = calculateClassProbabilities(summaries, inputVector)
    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/float(len(testSet))) * 100.0
 def main():
    filename     =     'C:\\Users\\ISE\\Desktop\\Python-naive-
Bayesian-Classifier1-master\\pima-indians-diabetes.csv'
    splitRatio = 0.67
    dataset = loadcsv(filename)
    print("\n The length of the Data Set : ",len(dataset))

    print("\n The Data Set Splitting into Training and Testing
\n")
    trainingSet, testSet = splitDataset(dataset, splitRatio)

    print('\n   Number   of   Rows   in   Training   Set:{0}
rows'.format(len(trainingSet)))
    print('\n   Number   of   Rows   in   Testing   Set:{0}
rows'.format(len(testSet)))

    print("\n First Five Rows of Training Set:\n")
    for i in range(0,5):
        print(trainingSet[i],"\n")

    print("\n First Five Rows of Testing Set:\n")
    for i in range(0,5):
        print(testSet[i],"\n")
       summaries = summarizeByClass(trainingSet)
    print("\n Model Summaries:\n",summaries)
       predictions = getPredictions(summaries, testSet)
    print("\nPredictions:\n",predictions)

    accuracy = getAccuracy(testSet, predictions)
    print('\n Accuracy: {0}%'.format(accuracy))
main()
```

OUTPUT


NEW_PROGRAM3_BACKPROP...
PROGRAM5_BAYESIANCLASSIF...
file://///192.168.26.195/e/2018-19/LABORATORY/MACHINE LEARNING/PROGRAM5_BAYESIAN_CLASSIFICATION/PROGRAM5_BAYESIANCLA
Search
First Five Rows of Testing Set:
[6.0, 148.0, 72.0, 35.0, 0.0, 33.6, 0.627, 50.0, 1.0]
[8.0, 183.0, 64.0, 0.0, 0.0, 23.3, 0.672, 32.0, 1.0]
[10.0, 115.0, 0.0, 0.0, 0.0, 35.3, 0.134, 29.0, 0.0]
[2.0, 197.0, 70.0, 45.0, 543.0, 30.5, 0.158, 53.0, 1.0]
[4.0, 110.0, 92.0, 0.0, 0.0, 37.6, 0.191, 30.0, 0.0]
Model Summaries:
{1.0: [(4.875, 3.806167494312934), (143.09895833333334, 29.599695749079007), (70.08333333333333, 22.610596907106117), (2
2.380208333333332, 17.593318102086016), (95.98958333333333, 120.99491528338224), (34.91562499999999, 7.4130397086831445),
(0.5461510416666671, 0.35892692470250287), (37.15625, 11.07974721865618)], 0.0: [(3.2763975155279503, 3.0100561576187816
), (112.2111801242236, 26.45513687620598), (67.95031055900621, 18.02604596789285), (20.562111801242235, 15.01725464122601
4), (76.36335403726709, 108.45368087750887), (30.66211180124225, 7.795113447361975), (0.4387515527950312, 0.3110696255076
493), (31.03105590062112, 11.52157524290244)]}
Predictions:
[1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0,
1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0,
0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0
.0, 0.0, 1.0, 1.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.
0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0
, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0,
0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1
.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.
0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0
, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0]
Accuracy: 79.52755905511812%
11:43 AM
10/13/2018

Program 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.

Algorithm :

Expectation Maximization (EM) Algorithm

- When to use:
  - Data is only partially observable
  - Unsupervised clustering (target value unobservable)
  - Supervised learning (some instance attributes unobservable)
- Some uses:
  - Train Bayesian Belief Networks
  - Unsupervised clustering (AUTOCLASS)
  - Learning Hidden Markov Models

EM for Estimating *k* Means

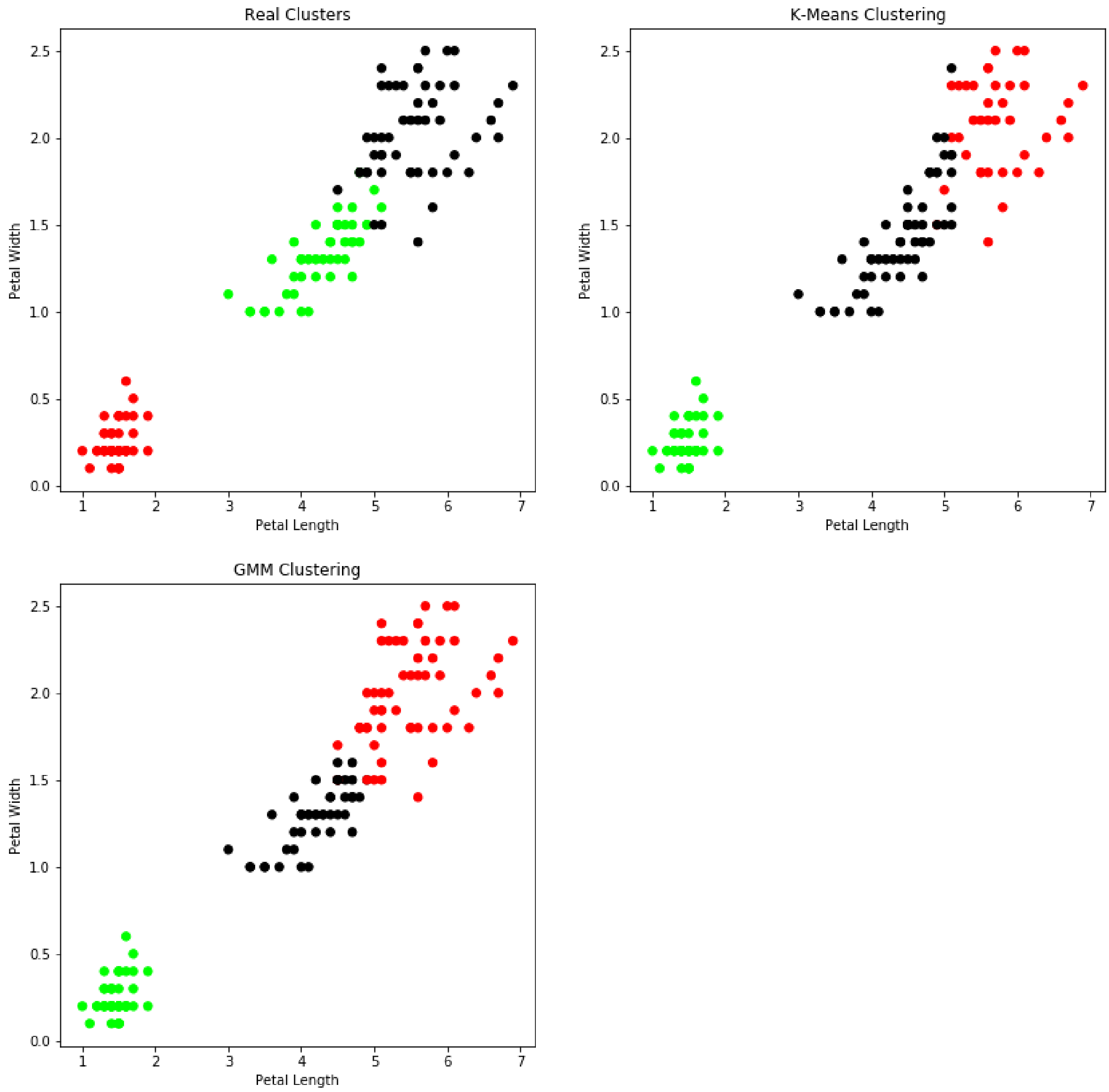
- Given:
  - Instances from X generated by mixture of k Gaussian distributions
  - Unknown means $<\mu_1,\ldots,\mu_k>$ of the k Gaussians
  - Don't know which instance x_i was generated by which Gaussian
- Determine:
  - Maximum likelihood estimates of $<\mu_1,\ldots,\mu_k>$
- Think of full description of each instance as

  $y_i = < x_i, z_{i1}, z_{i2}>$ where
  - z_{ij} is 1 if x_i generated by jth Gaussian
  - x_i observable
  - z_{ij} unobservable

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import pandas as pd
import numpy as np

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

```
model = KMeans(n_clusters=3)
model.fit(X)
plt.figure(figsize=(14, 14))
colormap = np.array(['red', 'lime', 'black'])
plt.subplot(2, 2, 1)
plt.scatter(X.Petal_Length,                         X.Petal_Width,
c=colormap[y.Targets], s=40)
plt.title('Real Clusters')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

plt.subplot(2, 2, 2)
plt.scatter(X.Petal_Length,                         X.Petal_Width,
c=colormap[model.labels_], s=40)
plt.title('K-Means Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

from sklearn import preprocessing
scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns=X.columns)
from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=3)
gmm.fit(xs)
gmm_y = gmm.predict(xs)
plt.subplot(2, 2, 3)
plt.scatter(X.Petal_Length,  X.Petal_Width,  c=colormap[gmm_y],
s=40)
plt.title('GMM Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.show()
print('Observation:  The  GMM  using  EM  algorithm  based
clustering  matched  the  true  labels  more  closely  than  the
Kmeans.')
```

OUTPUT:


Real Clusters
Petal Width
Petal Length
K-Means Clustering
Petal Width
Petal Length
GMM Clustering
Petal Width
Petal Length

Program8 : Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

KNN ALGORITHM

Training algorithm:

- For each training example $\langle x, f(x)\rangle$, add the example to the list *training_examples*

Classification algorithm:

- Given a query instance x_q to be classified,
  - Let $x_1 \ldots x_k$ denote the k instances from *training_examples* that are nearest to x_q
  - Return

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} \delta(v, f(x_i))$$

where $\delta(a, b) = 1$ if $a = b$ and where $\delta(a, b) = 0$ otherwise.

The k-NEAREST NEIGHBOR algorithm for approximating a discrete-valued function $f : \Re^n \rightarrow V$.

PROGRAM CODE:

```
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()

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"])

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)
kn = KNeighborsClassifier(n_neighbors=1)
kn.fit(X_train, y_train)

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 :

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

TEST SCORE[ACCURACY]: 0.97
```

Program9 : Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

Algorithm

1. **Read the Given data Sample to X and the curve (linear or non linear) to Y**
2. **Set the value for Smoothening parameter or Free parameter say τ**
3. **Set the bias /Point of interest set X0 which is a subset of X**
4. **Determine the weight matrix using :**

$$w(x, x_o) = e^{-\frac{(x-x_o)^2}{2\tau^2}}$$

5. **Determine the value of model term parameter β using :**

$$\hat{\beta}(x_o) = (X^T W X)^{-1} X^T W y$$

6. **Prediction = x0*β**

Program code :

```
import numpy as np
from bokeh.plotting import figure, show, output_notebook
from bokeh.layouts import gridplot
from bokeh.io import push_notebook

output_notebook()
```

```
import numpy as np

def local_regression(x0, X, Y, tau):
    # add bias term
    x0 = np.r_[1, x0]
    X = np.c_[np.ones(len(X)), X]


    xw = X.T * radial_kernel(x0, X, tau)

    beta = np.linalg.pinv(xw @ X) @ xw @ Y


    return x0 @ beta
```

```
def radial_kernel(x0, X, tau):
    return np.exp(np.sum((X - x0) ** 2, axis=1) / (-2 * tau *
tau))
```

```
n = 1000
# generate dataset
X = np.linspace(-3, 3, num=n)
print("The Data Set ( 10 Samples) X :\n",X[1:10])
Y = np.log(np.abs(X ** 2 - 1) + .5)
print("The Fitting Curve Data Set (10 Samples) Y
:\n",Y[1:10])
# jitter X
X += np.random.normal(scale=.1, size=n)
print("Normalised (10 Samples) X :\n",X[1:10])
```

```
domain = np.linspace(-3, 3, num=300)
print(" Xo Domain Space(10 Samples)  :\n",domain[1:10])

def plot_lwr(tau):
    prediction = [local_regression(x0, X, Y, tau) for x0
in domain]
    plot = figure(plot_width=400, plot_height=400)
    plot.title.text='tau=%g' % tau
    plot.scatter(X, Y, alpha=.3)
    plot.line(domain, prediction, line_width=2,
color='red')
    return plot
```

 # Plotting the curves with different tau

```
# Plotting the curves with different tau
show(gridplot([
    [plot_lwr(10.), plot_lwr(1.)],
    [plot_lwr(0.1), plot_lwr(0.01)]
]))
```

Output :