Department of Computer Science & Engineering

Subject: Artificial Intelligence and Machine Learning Laboratory

Subject Code: 18CSL76

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 algorithm to 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 to classify a newsample.
5.	Build an Artificial Neural Network by implementing the Back propagation 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 Regression algorithm 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 fromthe 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 ofthe changed part only.
- Marks Distribution *(Coursed to change in accordance with university regulations)*
  - For laboratories having only one part – Procedure + Execution + Viva-Voce: 15+70+15 = 100 Marks
  - 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

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:
              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', 'E', 'D', 'G']

Out[1]:
['A', 'E', 'D', 'G']
```

2. Implement AO* Search algorithm.

```
# Recursive implementation of AO* 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
                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 csv
a = [ ]
print("\n The Given Training Data Set \n")

with open('ws.csv', 'r') as csvFile:
   reader = csv.reader(csvFile)
   for row in reader:
      a.append (row)
      print(row)
num_attributes = len(a[0])-1 # we don't want last col which is target concet ( yes/no)

print("\n The initial value of hypothesis: ")
S = ['0'] * num_attributes
G = ['?'] * num_attributes
print ("\n The most specific hypothesis S0 : [0,0,0,0,0,0]\n")
print (" \n The most general hypothesis G0 : [?,?,?,?,?,?]\n")

for j in range(0,num_attributes):
      S[j] = a[0][j];

# Comparing with Remaining Training Examples of Given Data Set

print("\n Candidate Elimination algorithm  Hypotheses Version Space Computation\n")
temp=[]

for i in range(0,len(a)):
   if a[i][num_attributes]=='Yes':
      for j in range(0,num_attributes):
         if a[i][j]!=S[j]:
            S[j]='?'

      for j in range(0,num_attributes):
         for k  in range(0,len(temp)):
            if temp[k][j] != '?' and temp[k][j] != S[j]:
               del temp[k] #remove it if it's not matching with the specific hypothesis

      print(" For Training Example No :{0} the hypothesis is S{0}  ".format(i+1),S)

      if (len(temp)==0):
         print(" For Training Example No :{0} the hypothesis is G{0} ".format(i+1),G)
      else:
         print(" For Training Example No :{0} the hypothesis is G{0}".format(i+1),temp)
```

```
    if a[i][num_attributes]=='No':
        for j in range(0,num_attributes):
            if S[j] != a[i][j] and S[j]!= '?':  #if not  matching with the specific Hypothesis take it
seperately and store it
                G[j]=S[j]
                temp.append(G) # this is the version space to store all Hypotheses
                G = ['?'] * num_attributes

        print(" For Training Example No :{0} the hypothesis is S{0} ".format(i+1),S)
        print(" For Training Example No :{0} the hypothesis is G{0}".format(i+1),temp)
```

Dataset:

Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

Output:

```
The Given Training Data Set

['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes']
['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes']
['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'No']
['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Yes']

 The initial value of hypothesis:
 The most specific hypothesis S0 : [0,0,0,0,0,0]
 The most general hypothesis G0 : [?,?,?,?,?,?]
 Candidate Elimination algorithm  Hypotheses Version Space
Computation

 For Training Example No :1 the hypothesis is S1   ['Sunny',
'Warm', 'Normal', 'Strong', 'Warm', 'Same']
 For Training Example No :1 the hypothesis is G1  ['?', '?',
'?', '?', '?', '?']
 For Training Example No :2 the hypothesis is S2   ['Sunny',
'Warm', '?', 'Strong', 'Warm', 'Same']
 For Training Example No :2 the hypothesis is G2  ['?', '?',
'?', '?', '?', '?']
 For Training Example No :3 the hypothesis is S3  ['Sunny',
'Warm', '?', 'Strong', 'Warm', 'Same']
 For Training Example No :3 the hypothesis is G3 [['Sunny',
'?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', 'Same']]
 For Training Example No :4 the hypothesis is S4   ['Sunny',
'Warm', '?', 'Strong', '?', '?']
 For Training Example No :4 the hypothesis is G4 [['Sunny',
'?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]
```

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.

```
import sys
import numpy as np
from numpy import *
import csv

class Node:
    def __init__(self, attribute):
        self.attribute = attribute
        self.children = []
        self.answer = ""

def read_data(filename):
    """ read csv file and return header and data  """
    with open(filename, 'r') as csvfile:
        datareader = csv.reader(csvfile, delimiter=',')
        metadata = next(datareader)
        traindata=[]
        for row in datareader:
            traindata.append(row)

    return (metadata, traindata)

def subtables(data, col, delete):
    dict = {}
    items = np.unique(data[:, col]) # get unique values in a particular column

    count = np.zeros((items.shape[0], 1), dtype=np.int32)   #number of row = number of
values

    for x in range(items.shape[0]):
        for y in range(data.shape[0]):
            if data[y, col] == items[x]:
                count[x] += 1
    #count has the data of number of times each value is present in

    for x in range(items.shape[0]):
        dict[items[x]] = np.empty((int(count[x]), data.shape[1]), dtype="|S32")

        pos = 0
        for y in range(data.shape[0]):
            if data[y, col] == items[x]:
                dict[items[x]][pos] = data[y]
                pos += 1

        if delete:
            dict[items[x]] = np.delete(dict[items[x]], col, 1)
    return items, dict
```

```
def entropy(S):
    """ calculate the entropy """
    items = np.unique(S)
    if items.size == 1:
        return 0

    counts = np.zeros((items.shape[0], 1))
    sums = 0

    for x in range(items.shape[0]):
        counts[x] = sum(S == items[x]) / (S.size)

    for count in counts:
        sums += -1 * count * math.log(count, 2)
    return sums

def gain_ratio(data, col):
    items, dict = subtables(data, col, delete=False)
    #item is the unique value and dict is the data corresponding to it
    total_size = data.shape[0]
    entropies = np.zeros((items.shape[0], 1))

    for x in range(items.shape[0]):
        ratio = dict[items[x]].shape[0]/(total_size)
        entropies[x] = ratio * entropy(dict[items[x]][:, -1])

   total_entropy = entropy(data[:, -1])

   for x in range(entropies.shape[0]):
        total_entropy -= entropies[x]

    return total_entropy

def create_node(data, metadata):

    if (np.unique(data[:, -1])).shape[0] == 1: #to check how many rows in last col(yes,no
column). shape[0] gives no. of rows
        ''' if there is only yes or only no then reutrn a node containing the value '''
        node = Node("")
        node.answer = np.unique(data[:, -1])
        return node

    gains = np.zeros((data.shape[1] - 1, 1))  # data.shape[1] - 1 returns the no of columns in the
dataset, minus one to remove last column
    #size of gains= number of attribute to calculate gain
    #gains is one dim array (size=4) to store the gain of each attribute

    for col in range(data.shape[1] - 1):
        gains[col] = gain_ratio(data, col)

    split = np.argmax(gains) # argmax returns the index of the max value
```

```
    node = Node(metadata[split])
    metadata = np.delete(metadata, split, 0)


    items, dict = subtables(data, split, delete=True)

    for x in range(items.shape[0]):
        child = create_node(dict[items[x]], metadata)
        node.children.append((items[x], child))

    return node

def empty(size):
    """ To generate empty space needed for shaping the tree"""
    s = ""
    for x in range(size):
        s += "   "
    return s

def print_tree(node, level):
    if node.answer != "":

        print(empty(level), node.answer.item(0).decode("utf-8"))
        return

    print(empty(level), node.attribute)

    for value, n in node.children:
        print(empty(level + 1), value.tobytes().decode("utf-8"))
        print_tree(n, level + 2)


metadata, traindata = read_data("tennis.csv")
data = np.array(traindata) # to convert the traindata to numpy array
node = create_node(data, metadata)
print_tree(node, 0)
```

Dataset:

outlook	temp	humidity	windy	play
sunny	hot	high	Weak	no
sunny	hot	high	Strong	no
overcast	hot	high	Weak	yes
rainy	mild	high	Weak	yes
rainy	cool	normal	Weak	yes
rainy	cool	normal	Strong	no
overcast	cool	normal	Strong	yes
sunny	mild	high	Weak	no
sunny	cool	normal	Weak	yes
rainy	mild	normal	Weak	yes
sunny	mild	normal	Strong	yes
overcast	mild	high	Strong	yes
overcast	hot	normal	Weak	yes
rainy	mild	high	Strong	no

Output:

```
outlook
    overcast
       yes
    rainy
       windy
          Strong
             no
          Weak
             yes
    sunny
       humidity
          high
             no
          normal
             yes
```

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

```
import numpy as np
X = np.array(([2, 9], [1, 5], [3, 6])) # Hours Studied, Hours Slept
y = np.array(([92], [86], [89])) # Test Score

y = y/100 # max test score is 100


#Sigmoid Function
def sigmoid(x): #this function maps any value between 0 and 1
    return 1/(1 + np.exp(-x))

#Derivative of Sigmoid Function
def derivatives_sigmoid(x):
    return x * (1 - x)

#Variable initialization
epoch=10000 #Setting training iterations
lr=0.1 #Setting learning rate
inputlayer_neurons = 2 #number of features in data set
hiddenlayer_neurons = 3 #number of hidden layers neurons
output_neurons = 1 #number of neurons of output layer

#weight and bias initialization
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
#bias matrix to the hidden layer
bias_hidden=np.random.uniform(size=(1,hiddenlayer_neurons))
#weight matrix to the output layer
weight_hidden=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bias_output=np.random.uniform(size=(1,output_neurons)) # matrix to the output layer

for i in range(epoch):
    #Forward Propogation
    hinp1=np.dot(X,wh)
    hinp= hinp1 + bias_hidden #bias_hidden GRADIENT DISCENT
    hlayer_activation = sigmoid(hinp)

    outinp1=np.dot(hlayer_activation,weight_hidden)
    outinp= outinp1+ bias_output
    output = sigmoid(outinp)

    #Backpropagation
    EO = y-output
    outgrad = derivatives_sigmoid(output)

#Compute change factor(delta) at output layer, dependent on the gradient of error multiplied by the slope of output layer activation
    d_output = EO * outgrad
```

#At this step, the error will propagate back into the network which means error at hidden layer. we will take the dot product of output layer delta with weight parameters of edges between the hidden and output layer (weight_hidden.T).

```
    EH = d_output.dot(weight_hidden.T)
#how much hidden layer weight contributed to error

    hiddengrad = derivatives_sigmoid(hlayer_activation)
    d_hiddenlayer = EH * hiddengrad

    #update the weights
    weight_hidden += hlayer_activation.T.dot(d_output) *lr
    bias_hidden += np.sum(d_hiddenlayer, axis=0,keepdims=True) *lr

    wh += X.T.dot(d_hiddenlayer) *lr
    bias_output += np.sum(d_output, axis=0,keepdims=True) *lr

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

Output:

```
Input:
[[2 9]
 [1 5]
 [3 6]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.8921829 ]
 [0.88212774]
 [0.89429156]]
```

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 numpy as np
import math
import csv
import pdb
def read_data(filename):

    with open(filename,'r') as csvfile:
        datareader = csv.reader(csvfile)
        metadata = next(datareader)
        traindata=[]
        for row in datareader:
            traindata.append(row)

    return (metadata, traindata)

def splitDataset(dataset, splitRatio):
    trainSize = int(len(dataset) * splitRatio)
    trainSet = []
    testset = list(dataset)
    i=0
    while len(trainSet) < trainSize:
        trainSet.append(testset.pop(i))
    return [trainSet, testset]

def classify(data,test):

    total_size = data.shape[0]
    print("training data size=",total_size)
    print("test data size=",test.shape[0])

    countYes = 0
    countNo = 0
    probYes = 0
    probNo = 0
    print("target    count    probability")

    for x in range(data.shape[0]):
        if data[x,data.shape[1]-1] == 'yes':
            countYes +=1
        if data[x,data.shape[1]-1] == 'no':
            countNo +=1

    probYes=countYes/total_size
    probNo= countNo / total_size

    print('Yes',"\t",countYes,"\t",probYes)
    print('No',"\t",countNo,"\t",probNo)
```

```
    prob0 =np.zeros((test.shape[1]-1))
    prob1 =np.zeros((test.shape[1]-1))
    accuracy=0
    print("instance prediction  target")

    for t in range(test.shape[0]):
        for k in range (test.shape[1]-1):
            count1=count0=0
            for j in range (data.shape[0]):
                #how many times appeared with no
                if test[t,k] == data[j,k] and data[j,data.shape[1]-1]=='no':
                    count0+=1
                #how many times appeared with yes
                if test[t,k]==data[j,k] and data[j,data.shape[1]-1]=='yes':
                    count1+=1
            prob0[k]=count0/countNo
            prob1[k]=count1/countYes

        probno=probNo
        probyes=probYes
        for i in range(test.shape[1]-1):
            probno=probno*prob0[i]
            probyes=probyes*prob1[i]
        if probno>probyes:
            predict='no'
        else:
            predict='yes'

        print(t+1,"\t",predict,"\t    ",test[t,test.shape[1]-1])
        if predict == test[t,test.shape[1]-1]:
            accuracy+=1
    final_accuracy=(accuracy/test.shape[0])*100
    print("accuracy",final_accuracy,"%")
    return

metadata,traindata= read_data("tennis.csv")
splitRatio=0.6
trainingset, testset=splitDataset(traindata, splitRatio)
training=np.array(trainingset)
testing=np.array(testset)

classify(training,testing)
```

Dataset:

outlook	temp	humidity	windy	play
sunny	hot	high	Weak	no
sunny	hot	high	Strong	no
overcast	hot	high	Weak	yes
rainy	mild	high	Weak	yes
rainy	cool	normal	Weak	yes
rainy	cool	normal	Strong	no
overcast	cool	normal	Strong	yes
sunny	mild	high	Weak	no
sunny	cool	normal	Weak	yes
rainy	mild	normal	Weak	yes
sunny	mild	normal	Strong	yes
overcast	mild	high	Strong	yes
overcast	hot	normal	Weak	yes
rainy	mild	high	Strong	no

Output:

```
training data size= 8
test data size= 6
target    count    probability
Yes     4      0.5
No      4      0.5
instance prediction  target
1       no         yes
2       yes        yes
3       no         yes
4       yes        yes
5       yes        yes
6       no         no
accuracy 66.66666666666666 %
```

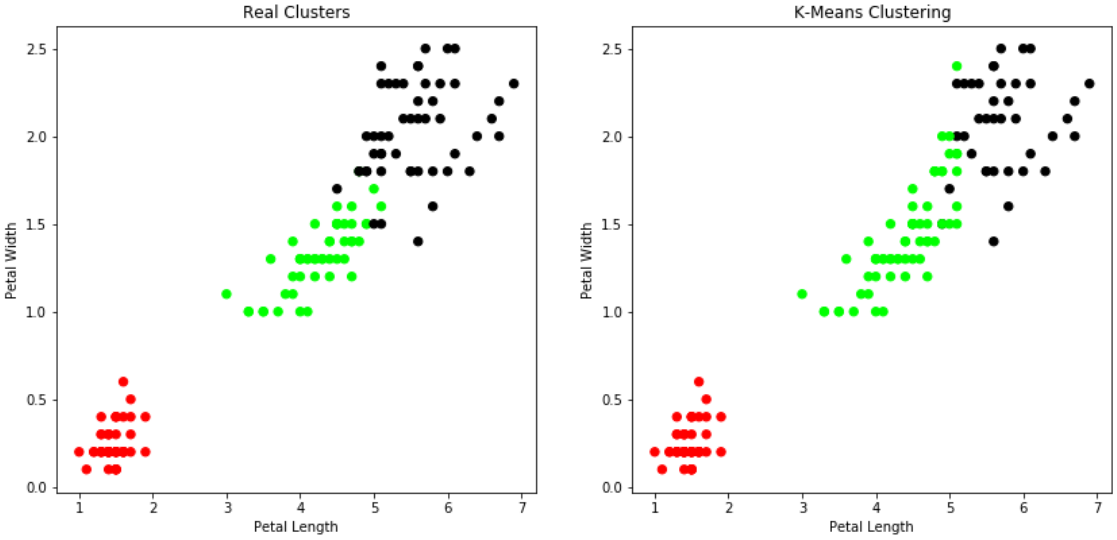
7. Apply **EM algorithm** to cluster a set of data stored in a .CSV file. Use the same data set for clustering using ***k*-Means algorithm**. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

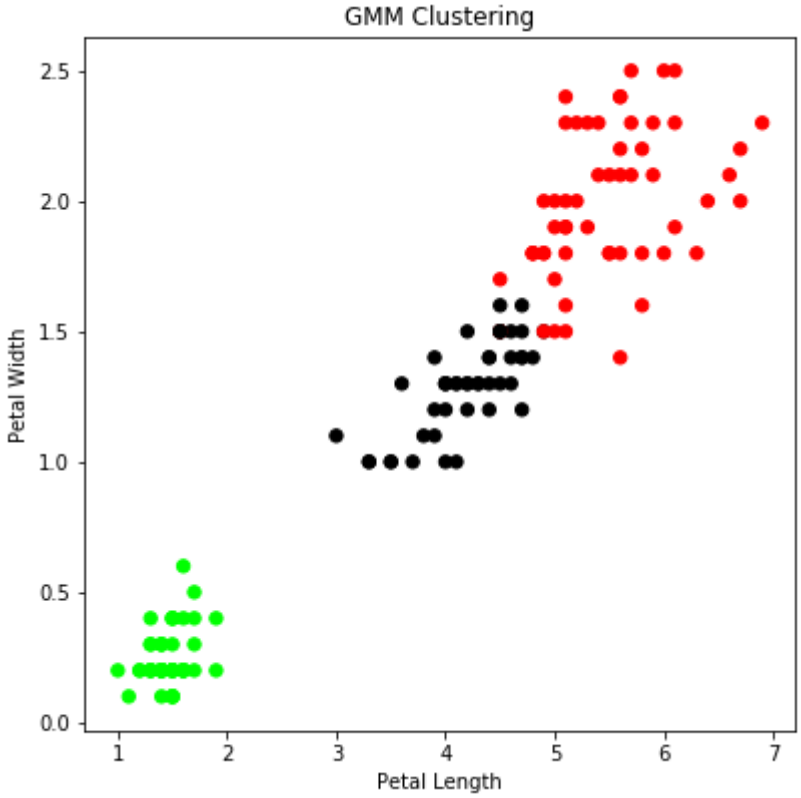
```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import pandas as pd
import numpy as np
# import some data to play with
iris = datasets.load_iris()
X = pd.DataFrame(iris.data)
X.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
y = pd.DataFrame(iris.target)
y.columns = ['Targets']
# Build the K Means Model
model = KMeans(n_clusters=3)
model.fit(X) # model.labels_ : Gives cluster no for which samples belongs to
# # Visualise the clustering results
plt.figure(figsize=(14,14))
colormap = np.array(['red', 'lime', 'black'])
# Plot the Original Classifications using Petal features
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')
# Plot the Models Classifications
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')
# General EM for GMM
from sklearn import preprocessing
```

```
# transform your data such that its distribution will have a
# mean value 0 and standard deviation of 1.
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')
print('Observation: The GMM using EM algorithm based clustering matched the true labels more closely than the Kmeans.')
```

OUTPUT:

Observation: The GMM using EM algorithm based clustering matched the true labels more closely than the Kmeans.





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.

```
from sklearn import datasets
iris=datasets.load_iris()
iris_data=iris.data
iris_labels=iris.target

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(iris_data,iris_labels,test_size=0.30)

from sklearn.neighbors import KNeighborsClassifier
classifier=KNeighborsClassifier(n_neighbors=5)
classifier.fit(x_train,y_train)
y_pred=classifier.predict(x_test)

from sklearn.metrics import classification_report,confusion_matrix
print('Confusion matrix is as follows')
print(confusion_matrix(y_test,y_pred))
print('Accuracy Matrics')
print(classification_report(y_test,y_pred))
```

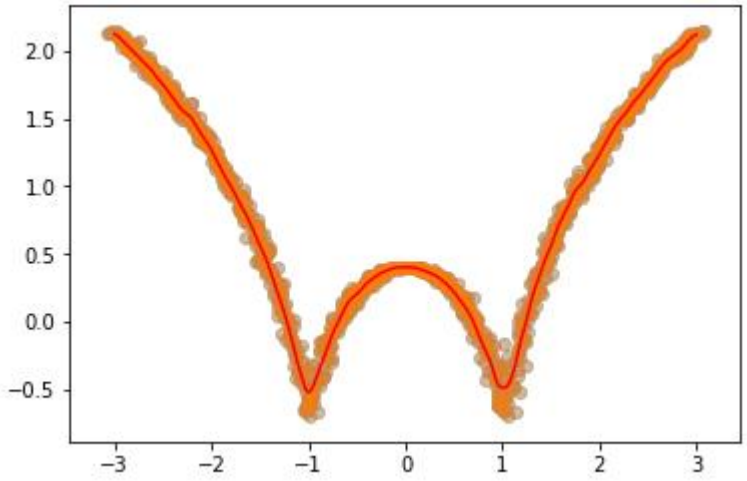
Output:

```
Confusion matrix is as follows
[[10  0  0]
 [ 0 16  1]
 [ 0  1 17]]
Accuracy Matrics
             precision    recall  f1-score   support

          0       1.00      1.00      1.00        10
          1       0.94      0.94      0.94        17
          2       0.94      0.94      0.94        18

avg / total       0.96      0.96      0.96        45
```

9. Implement the non-parametric **Locally Weighted Regressionalgorithm** 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
x=np.linspace(-3,3,1000)
y=np.log(np.abs((x**2)-1)+0.5)
x+=np.random.normal(scale=0.05,size=1000)
plt.scatter(x,y,alpha=0.3)
def local_regression(x0,X,Y,tau):
    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**2))
def plot_lwr(tau):
    domain=np.linspace(-3,3,num=300)
    prediction =[local_regression(x0,x,y,tau) for x0 in domain]
    plt.scatter(x,y,alpha=0.3)
    plt.plot(domain,prediction,color="red")
    return plt
plot_lwr(0.04)
```

Output:



VIVA VOCE QUESTIONS

1. What is machine learning?
2. Define supervised learning
3. Define unsupervised learning
4. Define semi supervised learning
5. Define reinforcement learning
6. What do you mean by hypotheses?
7. What is classification?
8. What is clustering?
9. Define precision, accuracy and recall
10. Define entropy
11. Define regression
12. How Knn is different from k-means clustering
13. What is concept learning?
14. Define specific boundary and general boundary
15. Define target function
16. Define decision tree
17. What is ANN
18. Explain gradient descent approximation
19. State Bayes theorem
20. Define Bayesian belief networks
21. Differentiate hard and soft clustering
22. Define variance
23. What is inductive machine learning?
24. Why K nearest neighbor algorithm is lazy learning algorithm
25. Why naïve Bayes is naïve
26. Mention classification algorithms
27. Define pruning
28. Differentiate Clustering and classification
29. Mention clustering algorithms
30. Define Bias
31. What is learning rate? Why it is need.