Vidyavardhaka Sangha®, Mysore
VIDYAVARDHAKA COLLEGE OF ENGINEERING
Autonomous Institute, Affiliated to Visvesvaraya Technological University, Belagavi
(Approved by AICTE, New Delhi & Government of Karnataka)
Accredited by NBA | NAAC with 'A' Grade
Department of Information Science & Engineering
Phone: +91 821-4276210, Email: hodis@vvce.ac.in
Web: http://www.vvce.ac.in
VVCE
@vvceofficial

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 to classify 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 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.

Course Instructors: N S Prema, R Kasturi Rangan
Department Of ISE, VVCE.

Program 1:

Implement A* Search algorithm

```
class Graph:
    def __init__(self, adjac_lis):
        self.adjac_lis = adjac_lis

    def get_neighbors(self, v):
        return self.adjac_lis[v]
    def h(self, n):
        H = {
            'S': 14,
            'B': 12,
            'C': 11,
            'D': 6,
            'E': 4,
            'F': 11,
            'G': 10
        }
        return H[n]

    def a_star_algorithm(self, start, stop):
        open_lst = set([start])
        closed_lst = set([])

        # poo has present distances from start to all other nodes with heuristics
        # the default value is +infinity
        poo = {}
        poo[start] = 0

        org = {}
        org[start] = 0

        # par contains an adjac mapping of all nodes
        par = {}
        par[start] = start

        while len(open_lst) > 0:
            n = None
            for v in open_lst:
                if n == None or org[v] + self.h(v) < org[n] + self.h(n):
                    n = v    #n=b v=c,D
                    #print(n)

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

            # if the current node is the stop
            # then we start again from start
            if n == stop:
                reconst_path = []

                while par[n] != n:
                    reconst_path.append(n)
                    n = par[n]
                    print(n)

                reconst_path.append(start)
```

```
                reconst_path.reverse()

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

            # for all the neighbors of the current node do
            for (m, weight) in self.get_neighbors(n):
                if m not in open_lst and m not in closed_lst:
                    open_lst.add(m)
                    par[m] = n
                    poo[m] = org[n] + weight + self.h(m)
                    org[m] = org[n] + weight
                # otherwise, check if it's quicker to first visit n, then m
                # and if it is, update par data and poo data
                # and if the node was in the closed_lst, move it to open_lst
                else:
                    if poo[m] > org[n] + weight + self.h(m):
                        poo[m] = org[n] + weight + self.h(m)
                        par[m] = n   #check the path cost if costlier take back m
to n i.e look for alternate path by going to previous node

                        if m in closed_lst:
                            closed_lst.remove(m)
                            open_lst.add(m)
            # remove n from the open_lst, and add it to closed_lst
            # because all of his neighbors were inspected
            open_lst.remove(n)
            closed_lst.add(n)

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

```
adjac_lis = {
    'S': [('B', 4), ('C', 3)],
    'B': [('F', 5), ('E', 12)],
    'C': [('D', 7), ('E', 10)],
    'D': [('E', 2)],
    'E': [('G', 5)],
    'F': [('G', 16)]
}
graph1 = Graph(adjac_lis)
graph1.a_star_algorithm('S', 'G')
```

Output:

```
E
D
C
S
Path found: ['S', 'C', 'D', 'E', 'G']
['S', 'C', 'D', 'E', 'G']
```

Program 2:

Implement AO* Search algorithm.

```
class Graph:
```

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

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

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

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

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

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

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

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


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

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

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']}
```

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.

```
import numpy as np
import pandas as pd
data = pd.DataFrame(data=pd.read_csv(r'candidate.csv'))
concepts = np.array(data.iloc[:,0:-1])
target = np.array(data.iloc[:,-1])
def learn(concepts, target):
    specific_h = concepts[0].copy()
    general_h = [["?" for i in range(len(specific_h))] for i in
range(len(specific_h))]

    for i,h in enumerate(concepts):
        if target[i] == "Yes":
```

```
            for x in range(len(specific_h)):
                if h[x]!=specific_h[x]:
                    specific_h[x] = '?'
                    general_h[x][x] = '?'
        if target[i] == "No":
            for x in range(len(specific_h)):
                if h[x]!=specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'
        print(specific_h)
        print(general_h)
    indices = [i for i,val in enumerate(general_h) if val ==
['?','?','?','?','?','?']]
    for i in indices:
        general_h.remove(['?','?','?','?','?','?'])
    return specific_h, general_h
s_final,g_final = learn(concepts,target)
print("Final S:",s_final, sep="\n")
print("Final G:",g_final, sep="\n")
data.head()
```

Output:

```
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm']
[['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?']]
['Sunny' 'Warm' '?' 'Strong' 'Warm']
[['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?']]
['Sunny' 'Warm' '?' 'Strong' 'Warm']
[['Sunny', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?'], ['?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?']]
['Sunny' 'Warm' '?' 'Strong' '?']
[['Sunny', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?'], ['?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?']]
Final S:
['Sunny' 'Warm' '?' 'Strong' '?']
Final G:
[['Sunny', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?'], ['?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?']]
```

	Sky	Airtemp	Humidity	Wind	Water	WaterSport
0	Sunny	Warm	Normal	Strong	Warm	Yes
1	Sunny	Warm	High	Strong	Warm	Yes
2	Cloudy	Cold	High	Strong	Warm	No
3	Sunny	Warm	High	Strong	Cool	Yes

Program 4:

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

```
# # DECISION TREE IMPLEMENTATION- ENTROPY AND INFO GAIN
import pandas as pd
from pandas import DataFrame
df_tennis = pd.DataFrame(data=pd.read_csv(r"PlayTennis.csv") )
df_tennis
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)
    print("No and Yes Classes:",a_list.name,cnt)
    num_instances = len(a_list)*1.0
    probs = [x / num_instances for x in cnt.values()]
    return entropy(probs) # Call Entropy:
total_entropy = entropy_of_list(df_tennis['PlayTennis'])
print("Entropy of given PlayTennis Data Set:",total_entropy)

def information_gain(df, split_attribute_name, target_attribute_name, trace=0):
 df_split = df.groupby(split_attribute_name)
 for name,group in df_split:
        print(name)
        print(group)
 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:
        gainz = [information_gain(df, attr, target_attribute_name) for attr in
attribute_names]
    index_of_max = gainz.index(max(gainz))
    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
attribute_names = list(df_tennis.columns)
print("List of Attributes:", attribute_names)
attribute_names.remove('PlayTennis')
print("Predicting Attributes:", attribute_names)
from pprint import pprint
tree = id3(df_tennis,'PlayTennis',attribute_names)
print("\n\nThe Resultant Decision Tree is :\n")
pprint(tree)
```

Output:

```
No and Yes Classes: PlayTennis Counter({'Yes': 9, 'No': 5})
Entropy of given PlayTennis Data Set: 0.9402859586706309
Overcast
   PlayTennis   Outlook Temperature Humidity    Wind
2         Yes  Overcast         Hot     High    Weak
6         Yes  Overcast        Cool   Normal  Strong
11        Yes  Overcast        Mild     High  Strong
12        Yes  Overcast         Hot   Normal    Weak
Rain
   PlayTennis Outlook Temperature Humidity    Wind
3         Yes    Rain        Mild     High    Weak
4         Yes    Rain        Cool   Normal    Weak
5          No    Rain        Cool   Normal  Strong
9         Yes    Rain        Mild   Normal    Weak
13         No    Rain        Mild     High  Strong
Sunny
   PlayTennis Outlook Temperature Humidity    Wind
0          No   Sunny         Hot     High    Weak
1          No   Sunny         Hot     High  Strong
7          No   Sunny        Mild     High    Weak
8         Yes   Sunny        Cool   Normal    Weak
10        Yes   Sunny        Mild   Normal  Strong
No and Yes Classes: PlayTennis Counter({'Yes': 4})
No and Yes Classes: PlayTennis Counter({'Yes': 3, 'No': 2})
No and Yes Classes: PlayTennis Counter({'No': 3, 'Yes': 2})
No and Yes Classes: PlayTennis Counter({'Yes': 9, 'No': 5})
Info-gain for Outlook is :0.2467498197744391

High
   PlayTennis   Outlook Temperature Humidity    Wind
0          No     Sunny         Hot     High    Weak
1          No     Sunny         Hot     High  Strong
2         Yes  Overcast         Hot     High    Weak
3         Yes      Rain        Mild     High    Weak
7          No     Sunny        Mild     High    Weak
11        Yes  Overcast        Mild     High  Strong
13         No      Rain        Mild     High  Strong
Normal
   PlayTennis   Outlook Temperature Humidity    Wind
4         Yes      Rain        Cool   Normal    Weak
5          No      Rain        Cool   Normal  Strong
6         Yes  Overcast        Cool   Normal  Strong
8         Yes     Sunny        Cool   Normal    Weak
9         Yes      Rain        Mild   Normal    Weak
```

```
10        Yes     Sunny        Mild   Normal  Strong
12        Yes  Overcast         Hot   Normal    Weak
No and Yes Classes: PlayTennis Counter({'No': 4, 'Yes': 3})
No and Yes Classes: PlayTennis Counter({'Yes': 6, 'No': 1})
No and Yes Classes: PlayTennis Counter({'Yes': 9, 'No': 5})

 Info-gain for Humidity is: 0.15183550136234136

Strong
   PlayTennis   Outlook Temperature Humidity    Wind
1          No     Sunny         Hot     High  Strong
5          No      Rain        Cool   Normal  Strong
6         Yes  Overcast        Cool   Normal  Strong
10        Yes     Sunny        Mild   Normal  Strong
11        Yes  Overcast        Mild     High  Strong
13         No      Rain        Mild     High  Strong
Weak
   PlayTennis   Outlook Temperature Humidity  Wind
0          No     Sunny         Hot     High  Weak
2         Yes  Overcast         Hot     High  Weak
3         Yes      Rain        Mild     High  Weak
4         Yes      Rain        Cool   Normal  Weak
7          No     Sunny        Mild     High  Weak
8         Yes     Sunny        Cool   Normal  Weak
9         Yes      Rain        Mild   Normal  Weak
12        Yes  Overcast         Hot   Normal  Weak
No and Yes Classes: PlayTennis Counter({'No': 3, 'Yes': 3})
No and Yes Classes: PlayTennis Counter({'Yes': 6, 'No': 2})
No and Yes Classes: PlayTennis Counter({'Yes': 9, 'No': 5})

 Info-gain for Wind is:0.04812703040826927

Cool
  PlayTennis   Outlook Temperature Humidity    Wind
4        Yes      Rain        Cool   Normal    Weak
5         No      Rain        Cool   Normal  Strong
6        Yes  Overcast        Cool   Normal  Strong
8        Yes     Sunny        Cool   Normal    Weak
Hot
   PlayTennis   Outlook Temperature Humidity    Wind
0          No     Sunny         Hot     High    Weak
1          No     Sunny         Hot     High  Strong
2         Yes  Overcast         Hot     High    Weak
12        Yes  Overcast         Hot   Normal    Weak
Mild
   PlayTennis   Outlook Temperature Humidity    Wind
3         Yes      Rain        Mild     High    Weak
7          No     Sunny        Mild     High    Weak
9         Yes      Rain        Mild   Normal    Weak
10        Yes     Sunny        Mild   Normal  Strong
11        Yes  Overcast        Mild     High  Strong
13         No      Rain        Mild     High  Strong
No and Yes Classes: PlayTennis Counter({'Yes': 3, 'No': 1})
No and Yes Classes: PlayTennis Counter({'No': 2, 'Yes': 2})
No and Yes Classes: PlayTennis Counter({'Yes': 4, 'No': 2})
No and Yes Classes: PlayTennis Counter({'Yes': 9, 'No': 5})

 Info-gain for Temperature is:0.029222565658954647

List of Attributes: ['PlayTennis', 'Outlook', 'Temperature', 'Humidity', 'Wind']
Predicting Attributes: ['Outlook', 'Temperature', 'Humidity', 'Wind']
Overcast
```

```
   PlayTennis   Outlook Temperature Humidity    Wind
2          Yes  Overcast         Hot     High    Weak
6          Yes  Overcast        Cool   Normal  Strong
11         Yes  Overcast        Mild     High  Strong
12         Yes  Overcast         Hot   Normal    Weak
Rain
   PlayTennis Outlook Temperature Humidity    Wind
3          Yes    Rain        Mild     High    Weak
4          Yes    Rain        Cool   Normal    Weak
5           No    Rain        Cool   Normal  Strong
9          Yes    Rain        Mild   Normal    Weak
13          No    Rain        Mild     High  Strong
Sunny
   PlayTennis Outlook Temperature Humidity    Wind
0           No   Sunny         Hot     High    Weak
1           No   Sunny         Hot     High  Strong
7           No   Sunny        Mild     High    Weak
8          Yes   Sunny        Cool   Normal    Weak
10         Yes   Sunny        Mild   Normal  Strong
No and Yes Classes: PlayTennis Counter({'Yes': 4})
No and Yes Classes: PlayTennis Counter({'Yes': 3, 'No': 2})
No and Yes Classes: PlayTennis Counter({'No': 3, 'Yes': 2})
No and Yes Classes: PlayTennis Counter({'Yes': 9, 'No': 5})
Cool
  PlayTennis   Outlook Temperature Humidity    Wind
4        Yes      Rain        Cool   Normal    Weak
5         No      Rain        Cool   Normal  Strong
6        Yes  Overcast        Cool   Normal  Strong
8        Yes     Sunny        Cool   Normal    Weak
Hot
   PlayTennis   Outlook Temperature Humidity    Wind
0           No     Sunny         Hot     High    Weak
1           No     Sunny         Hot     High  Strong
2          Yes  Overcast         Hot     High    Weak
12         Yes  Overcast         Hot   Normal    Weak
Mild
   PlayTennis   Outlook Temperature Humidity    Wind
3          Yes      Rain        Mild     High    Weak
7           No     Sunny        Mild     High    Weak
9          Yes      Rain        Mild   Normal    Weak
10         Yes     Sunny        Mild   Normal  Strong
11         Yes  Overcast        Mild     High  Strong
13          No      Rain        Mild     High  Strong
No and Yes Classes: PlayTennis Counter({'Yes': 3, 'No': 1})
No and Yes Classes: PlayTennis Counter({'No': 2, 'Yes': 2})
No and Yes Classes: PlayTennis Counter({'Yes': 4, 'No': 2})
No and Yes Classes: PlayTennis Counter({'Yes': 9, 'No': 5})
High
   PlayTennis   Outlook Temperature Humidity    Wind
0           No     Sunny         Hot     High    Weak
1           No     Sunny         Hot     High  Strong
2          Yes  Overcast         Hot     High    Weak
3          Yes      Rain        Mild     High    Weak
7           No     Sunny        Mild     High    Weak
11         Yes  Overcast        Mild     High  Strong
13          No      Rain        Mild     High  Strong
Normal
   PlayTennis   Outlook Temperature Humidity    Wind
4          Yes      Rain        Cool   Normal    Weak
5           No      Rain        Cool   Normal  Strong
6          Yes  Overcast        Cool   Normal  Strong
8          Yes     Sunny        Cool   Normal    Weak
```

```
9         Yes      Rain        Mild   Normal    Weak
10        Yes     Sunny        Mild   Normal  Strong
12        Yes  Overcast         Hot   Normal    Weak
No and Yes Classes: PlayTennis Counter({'No': 4, 'Yes': 3})
No and Yes Classes: PlayTennis Counter({'Yes': 6, 'No': 1})
No and Yes Classes: PlayTennis Counter({'Yes': 9, 'No': 5})
Strong
   PlayTennis   Outlook Temperature Humidity    Wind
1          No     Sunny         Hot     High  Strong
5          No      Rain        Cool   Normal  Strong
6         Yes  Overcast        Cool   Normal  Strong
10        Yes     Sunny        Mild   Normal  Strong
11        Yes  Overcast        Mild     High  Strong
13         No      Rain        Mild     High  Strong
Weak
   PlayTennis   Outlook Temperature Humidity  Wind
0          No     Sunny         Hot     High  Weak
2         Yes  Overcast         Hot     High  Weak
3         Yes      Rain        Mild     High  Weak
4         Yes      Rain        Cool   Normal  Weak
7          No     Sunny        Mild     High  Weak
8         Yes     Sunny        Cool   Normal  Weak
9         Yes      Rain        Mild   Normal  Weak
12        Yes  Overcast         Hot   Normal  Weak
No and Yes Classes: PlayTennis Counter({'No': 3, 'Yes': 3})
No and Yes Classes: PlayTennis Counter({'Yes': 6, 'No': 2})
No and Yes Classes: PlayTennis Counter({'Yes': 9, 'No': 5})
Cool
  PlayTennis Outlook Temperature Humidity    Wind
4        Yes    Rain        Cool   Normal    Weak
5         No    Rain        Cool   Normal  Strong
Mild
   PlayTennis Outlook Temperature Humidity    Wind
3         Yes    Rain        Mild     High    Weak
9         Yes    Rain        Mild   Normal    Weak
13         No    Rain        Mild     High  Strong
No and Yes Classes: PlayTennis Counter({'Yes': 1, 'No': 1})
No and Yes Classes: PlayTennis Counter({'Yes': 2, 'No': 1})
No and Yes Classes: PlayTennis Counter({'Yes': 3, 'No': 2})
High
   PlayTennis Outlook Temperature Humidity    Wind
3         Yes    Rain        Mild     High    Weak
13         No    Rain        Mild     High  Strong
Normal
  PlayTennis Outlook Temperature Humidity    Wind
4        Yes    Rain        Cool   Normal    Weak
5         No    Rain        Cool   Normal  Strong
9        Yes    Rain        Mild   Normal    Weak
No and Yes Classes: PlayTennis Counter({'Yes': 1, 'No': 1})
No and Yes Classes: PlayTennis Counter({'Yes': 2, 'No': 1})
No and Yes Classes: PlayTennis Counter({'Yes': 3, 'No': 2})
Strong
   PlayTennis Outlook Temperature Humidity    Wind
5          No    Rain        Cool   Normal  Strong
13         No    Rain        Mild     High  Strong
Weak
  PlayTennis Outlook Temperature Humidity  Wind
3        Yes    Rain        Mild     High  Weak
4        Yes    Rain        Cool   Normal  Weak
9        Yes    Rain        Mild   Normal  Weak
No and Yes Classes: PlayTennis Counter({'No': 2})
No and Yes Classes: PlayTennis Counter({'Yes': 3})
```

```
No and Yes Classes: PlayTennis Counter({'Yes': 3, 'No': 2})
Cool
  PlayTennis Outlook Temperature Humidity  Wind
8        Yes   Sunny        Cool   Normal  Weak
Hot
  PlayTennis Outlook Temperature Humidity    Wind
0         No   Sunny         Hot     High    Weak
1         No   Sunny         Hot     High  Strong
Mild
   PlayTennis Outlook Temperature Humidity    Wind
7          No   Sunny        Mild     High    Weak
10        Yes   Sunny        Mild   Normal  Strong
No and Yes Classes: PlayTennis Counter({'Yes': 1})
No and Yes Classes: PlayTennis Counter({'No': 2})
No and Yes Classes: PlayTennis Counter({'No': 1, 'Yes': 1})
No and Yes Classes: PlayTennis Counter({'No': 3, 'Yes': 2})
High
  PlayTennis Outlook Temperature Humidity    Wind
0         No   Sunny         Hot     High    Weak
1         No   Sunny         Hot     High  Strong
7         No   Sunny        Mild     High    Weak
Normal
   PlayTennis Outlook Temperature Humidity    Wind
8         Yes   Sunny        Cool   Normal    Weak
10        Yes   Sunny        Mild   Normal  Strong
No and Yes Classes: PlayTennis Counter({'No': 3})
No and Yes Classes: PlayTennis Counter({'Yes': 2})
No and Yes Classes: PlayTennis Counter({'No': 3, 'Yes': 2})
Strong
   PlayTennis Outlook Temperature Humidity    Wind
1          No   Sunny         Hot     High  Strong
10        Yes   Sunny        Mild   Normal  Strong
Weak
  PlayTennis Outlook Temperature Humidity  Wind
0         No   Sunny         Hot     High  Weak
7         No   Sunny        Mild     High  Weak
8        Yes   Sunny        Cool   Normal  Weak
No and Yes Classes: PlayTennis Counter({'No': 1, 'Yes': 1})
No and Yes Classes: PlayTennis Counter({'No': 2, 'Yes': 1})
No and Yes Classes: PlayTennis Counter({'No': 3, 'Yes': 2})


The Resultant Decision Tree is :

{'Outlook': {'Overcast': 'Yes',
             'Rain': {'Wind': {'Strong': 'No', 'Weak': 'Yes'}},
             'Sunny': {'Humidity': {'High': 'No', 'Normal': 'Yes'}}}}
```

Program 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]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)

print(np.amax(X,axis=0))    #running vertically downwards across rows (axis 0)
```

```
print(np.amax(X,axis=1))    # running horizontally across columns (axis 1)

X = X/np.amax(X,axis=0) # maximum of X array longitudinally
y = y/100
#Sigmoid Function
def sigmoid (x):
    return (1/(1 + np.exp(-x)))
#Derivative of Sigmoid Function
def derivatives_sigmoid(x):
    return x * (1 - x)
print(X)

#Variable initialization
epoch=9000
#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 at output layer

#weight and bias initialization
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
#size=(rows,columns)
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))
# draws a random range of numbers uniformly of dim x*y
#Forward Propagation
for i in range(epoch):
    hinp1=np.dot(X,wh)
    hinp=hinp1 + bh
    hlayer_act = sigmoid(hinp)
    outinp1=np.dot(hlayer_act,wout)
    outinp= outinp1+ bout
    output = sigmoid(outinp)
#Backpropagation
    EO = y-output
    outgrad = derivatives_sigmoid(output)
    d_output = EO* outgrad
    EH = d_output.dot(wout.T) #T transpose
    hiddengrad = derivatives_sigmoid(hlayer_act)
#how much hidden layer wts contributed to error
    d_hiddenlayer = EH * hiddengrad
    wout += hlayer_act.T.dot(d_output) *lr
# dotproduct of nextlayererror and currentlayerop
    bout += np.sum(d_output, axis=0,keepdims=True) *lr
    wh += X.T.dot(d_hiddenlayer) *lr
    bh += np.sum(d_hiddenlayer, axis=0,keepdims=True) *lr

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

Output:

```
[3. 9.]
[9. 5. 6.]
[[0.66666667 1.        ]
```

```
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.89362634]
 [0.87864882]
 [0.89697568]]
```

Program 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 datasets.

```
import numpy as np
import pandas as pd
from sklearn import metrics
#Import dataset
from sklearn import datasets
#Import Gaussian Naive Bayes model
from sklearn.naive_bayes import GaussianNB

#Load dataset
iris = datasets.load_iris()
print ("Features: ", iris.feature_names)
print ("Labels: ", iris.target_names)
X=pd.DataFrame(iris['data'])

#print(X.head())
#print(iris.data.shape)
#y=print (iris.target)

# Import train_test_split function
from sklearn.model_selection import train_test_split

# Split dataset into training set and test set

X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target,
test_size=0.30,random_state=109)

#Create a Gaussian Classifier

gnb = GaussianNB()

#Train the model using the training sets

gnb.fit(X_train, y_train)

#Predict the response for test dataset

y_pred = gnb.predict(X_test)
```

```
print(y_pred)
# Model Accuracy

print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

Output:

```
Features:  ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal
width (cm)']
Labels:  ['setosa' 'versicolor' 'virginica']
[2 1 2 0 2 1 0 2 1 2 2 0 1 0 0 0 1 1 0 1 1 0 2 0 0 2 2 1 1 2 1 2 1 2 2 1 0
 2 2 1 1 1 1 2 0]
Accuracy: 0.9555555555555556
```

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.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score, silhouette_score
iris = load_iris()
x = pd.DataFrame(iris.data, columns=iris.feature_names)
y = iris.target
plt.figure(figsize=(20,7))
plt.subplot(1, 2, 1)
plt.scatter(x['sepal length (cm)'], x['sepal width (cm)'], c=y, s=40)
plt.title('Sepal')
plt.subplot(1, 2, 2)
plt.scatter(x['petal length (cm)'], x['petal width (cm)'], c=y, s=40)
plt.title('Petal')

model = KMeans(n_clusters=3)
model.fit(x)
model.labels_
accuracy_score(y, model.labels_)
```

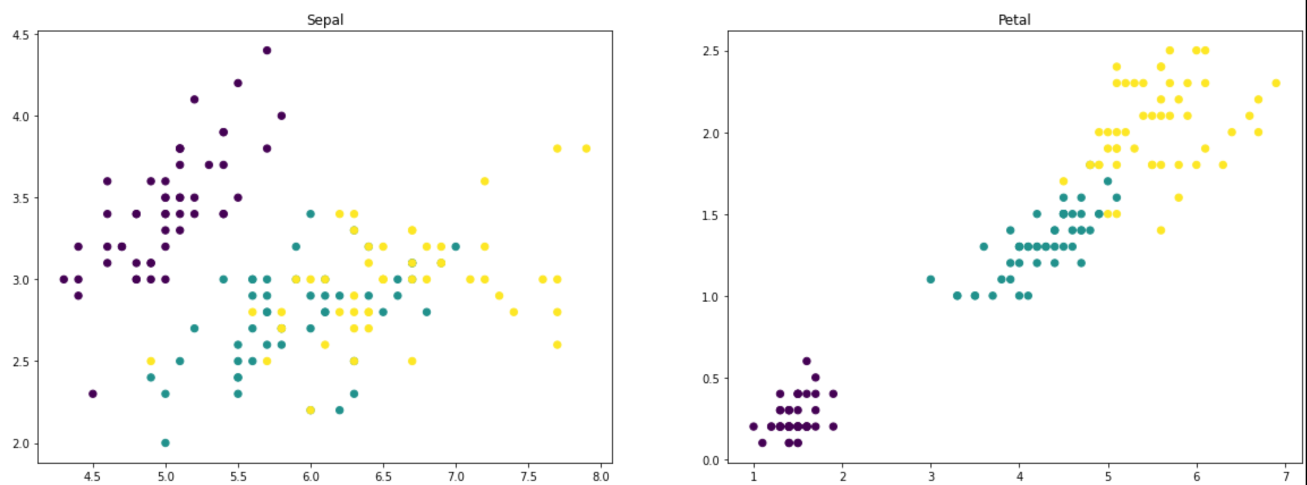
```
from sklearn.mixture import GaussianMixture
GMM = GaussianMixture(n_components=3) # Instantiate and fit the model
GMM.fit(x)
gmm_clusters = GMM.predict(x)
gmm_clusters
accuracy_score(y, gmm_clusters)
```

```
plt.figure(figsize=(17,7))
```

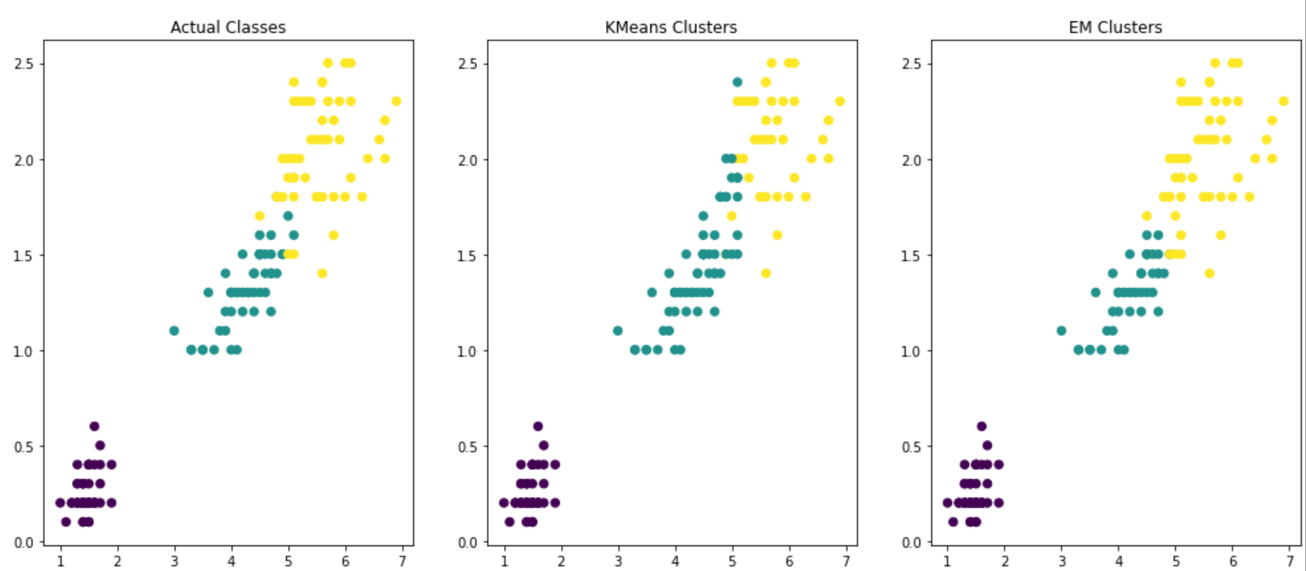
```
plt.subplot(1, 3, 1)
plt.scatter(x['petal length (cm)'], x['petal width (cm)'], c=y, s=40)
plt.title('Actual Classes')
plt.subplot(1, 3, 2)
plt.scatter(x['petal length (cm)'], x['petal width (cm)'], c=model.labels_, s=40)
plt.title('KMeans Clusters')
plt.subplot(1, 3, 3)
plt.scatter(x['petal length (cm)'], x['petal width (cm)'], c=gmm_clusters, s=40)
plt.title('EM Clusters')
```

Output:

```
0.8933333333333333
```




```
0.9666666666666667
```



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.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix

iris = load_iris()
x = iris.data
y = iris.target
print('Size of Iris dataset : ', x.shape)
print('Size of Iris dataset : ', y.shape)

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3)

print('Size of Train dataset : ', x_train.shape)
print('Size of Train targets : ', y_train.shape)
print('Size of Test dataset : ', x_test.shape)
print('Size of Test targets : ', y_test.shape)

knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(x_train, y_train)
predictions = knn_model.predict(x_test)
print(predictions)

print("Training accuracy Score is : ", accuracy_score(y_train,
knn_model.predict(x_train)))
print("Testing accuracy Score is : ", accuracy_score(y_test,
knn_model.predict(x_test)))
print("Training Confusion Matrix is : \n", confusion_matrix(y_train,
knn_model.predict(x_train)))
print("Testing Confusion Matrix is : \n", confusion_matrix(y_test,
knn_model.predict(x_test)))
```

Output:

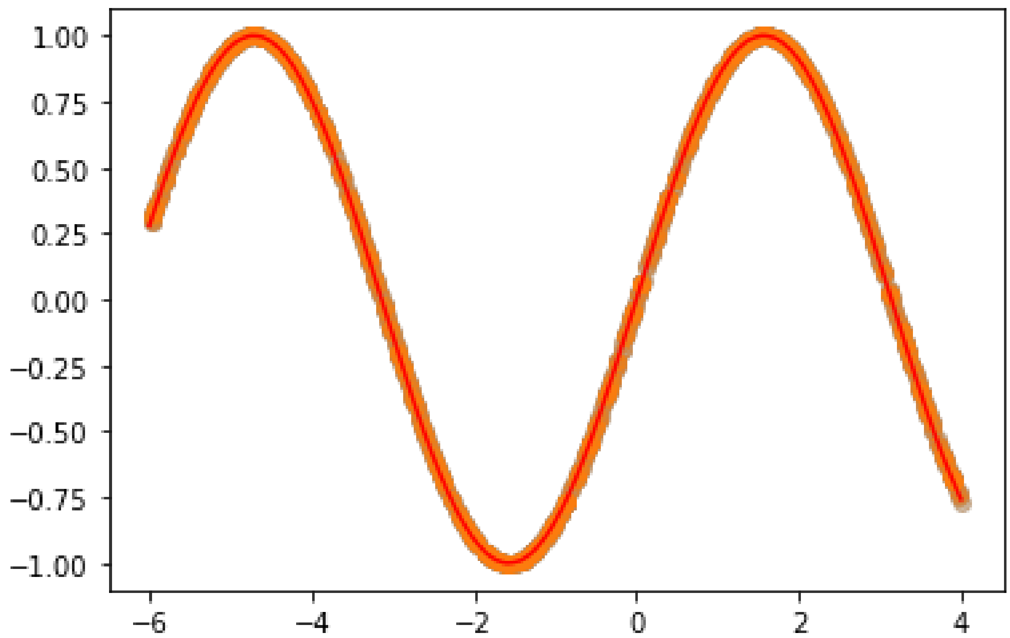
```
Size of Iris dataset :  (150, 4)
Size of Iris dataset :  (150,)
Size of Train dataset :  (105, 4)
Size of Train targets :  (105,)
Size of Test dataset :  (45, 4)
Size of Test targets :  (45,)
[0 0 1 0 1 2 0 2 0 1 2 0 2 0 1 2 0 2 2 1 2 2 1 2 2 1 1 2 1 0 2 1 1 2 1 2 0
 0 0 0 2 1 2 1 2]
Training accuracy Score is :  0.9904761904761905
Testing accuracy Score is :  0.9333333333333333
Training Confusion Matrix is :
 [[37  0  0]
 [ 0 35  0]
 [ 0  1 32]]
Testing Confusion Matrix is :
 [[13  0  0]
 [ 0 13  2]
 [ 0  1 16]]
```

Program 9:

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

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

x = np.linspace(-6, 4, 1000) #data range random values selection
x = x + np.random.normal(scale=0.05, size=1000)   #put to normal probability
distribution representation
#y = np.log(np.abs((x ** 2) - 1) + 0.5)
#y = np.abs((x ** 2) - 1) + 0.5  # generate label values based on any function
f(x)
y = np.sin(x)

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_lr(tau):
    domain = np.linspace(-6, 4, num=300)
    pred = [local_regression(x0, x, y, tau) for x0 in domain]
    plt.scatter(x, y, alpha=0.3)
    plt.plot(domain, pred, color="red")
    return plt


plot_lr(0.05).show()
```

Output:



Program 10: [Open Ended Program]

Implement the program to play a TIC TAC TOE game using artificial intelligence.

TIC TAC TOE game:
Two people play Tic Tac Toe. One player is X and the other player is O. Players take turns placing their X or O. If a player gets three of their marks on the board in a row, column or one of the two diagonals, they win. When the board fills up with neither player winning, the game ends in a draw.

```
# Tic Tac Toe .....Ref: https://inventwithpython.com/chapter10.html

import random

def drawBoard(board):
    # This function prints out the board that it was passed.

    # "board" is a list of 10 strings representing the board (ignore index 0)
    print('   |   |')
    print(' ' + board[7] + ' | ' + board[8] + ' | ' + board[9])
    print('   |   |')
    print('-----------')
    print('   |   |')
    print(' ' + board[4] + ' | ' + board[5] + ' | ' + board[6])
    print('   |   |')
    print('-----------')
    print('   |   |')
    print(' ' + board[1] + ' | ' + board[2] + ' | ' + board[3])
    print('   |   |')

```

```
def inputPlayerLetter():
    # Lets the player type which letter they want to be.
    # Returns a list with the player’s letter as the first item, and the computer's letter as the second.
    letter = ''
    while not (letter == 'X' or letter == 'O'):
        print('Do you want to be X or O?')
        letter = input().upper()

    # the first element in the list is the player’s letter, the second is the computer's letter.
    if letter == 'X':
        return ['X', 'O']
    else:
        return ['O', 'X']

def whoGoesFirst():
    # Randomly choose the player who goes first.
    if random.randint(0, 1) == 0:
        return 'computer'
    else:
        return 'player'

def playAgain():
    # This function returns True if the player wants to play again, otherwise it returns False.
    print('Do you want to play again? (yes or no)')
    return input().lower().startswith('y')

def makeMove(board, letter, move):
    board[move] = letter

def isWinner(bo, le):
    # Given a board and a player’s letter, this function returns True if that player has won.
    # We use bo instead of board and le instead of letter so we don’t have to type as much.
    return ((bo[7] == le and bo[8] == le and bo[9] == le) or # across the top
    (bo[4] == le and bo[5] == le and bo[6] == le) or # across the middle
    (bo[1] == le and bo[2] == le and bo[3] == le) or # across the bottom
    (bo[7] == le and bo[4] == le and bo[1] == le) or # down the left side
    (bo[8] == le and bo[5] == le and bo[2] == le) or # down the middle
    (bo[9] == le and bo[6] == le and bo[3] == le) or # down the right side
    (bo[7] == le and bo[5] == le and bo[3] == le) or # diagonal
    (bo[9] == le and bo[5] == le and bo[1] == le)) # diagonal

def getBoardCopy(board):
    # Make a duplicate of the board list and return it the duplicate.
    dupeBoard = []

    for i in board:
        dupeBoard.append(i)

    return dupeBoard

def isSpaceFree(board, move):
    # Return true if the passed move is free on the passed board.
    return board[move] == ' '

def getPlayerMove(board):
    # Let the player type in their move.
    move = ' '
    while move not in '1 2 3 4 5 6 7 8 9'.split() or not isSpaceFree(board, int(move)):
        print('What is your next move? (1-9)')
        move = input()
    return int(move)

def chooseRandomMoveFromList(board, movesList):
    # Returns a valid move from the passed list on the passed board.
    # Returns None if there is no valid move.
    possibleMoves = []
```

```
    for i in movesList:
        if isSpaceFree(board, i):
            possibleMoves.append(i)

    if len(possibleMoves) != 0:
        return random.choice(possibleMoves)
    else:
        return None

def getComputerMove(board, computerLetter):
    # Given a board and the computer's letter, determine where to move and return that move.
    if computerLetter == 'X':
        playerLetter = 'O'
    else:
        playerLetter = 'X'

    # Here is our algorithm for our Tic Tac Toe AI:
    # First, check if we can win in the next move
    for i in range(1, 10):
        copy = getBoardCopy(board)
        if isSpaceFree(copy, i):
            makeMove(copy, computerLetter, i)
            if isWinner(copy, computerLetter):
                return i

    # Check if the player could win on their next move, and block them.
    for i in range(1, 10):
        copy = getBoardCopy(board)
        if isSpaceFree(copy, i):
            makeMove(copy, playerLetter, i)
            if isWinner(copy, playerLetter):
                return i

    # Try to take one of the corners, if they are free.
    move = chooseRandomMoveFromList(board, [1, 3, 7, 9])
    if move != None:
        return move

    # Try to take the center, if it is free.
    if isSpaceFree(board, 5):
        return 5

    # Move on one of the sides.
    return chooseRandomMoveFromList(board, [2, 4, 6, 8])

def isBoardFull(board):
    # Return True if every space on the board has been taken. Otherwise return False.
    for i in range(1, 10):
        if isSpaceFree(board, i):
            return False
    return True


print('Welcome to Tic Tac Toe!')

while True:
    # Reset the board
    theBoard = [' '] * 10
    playerLetter, computerLetter = inputPlayerLetter()
    turn = whoGoesFirst()
    print('The ' + turn + ' will go first.')
    gameIsPlaying = True

    while gameIsPlaying:
        if turn == 'player':
            # Player’s turn.
```

```
            drawBoard(theBoard)
            move = getPlayerMove(theBoard)
            makeMove(theBoard, playerLetter, move)

            if isWinner(theBoard, playerLetter):
                drawBoard(theBoard)
                print('Hooray! You have won the game!')
                gameIsPlaying = False
            else:
                if isBoardFull(theBoard):
                    drawBoard(theBoard)
                    print('The game is a tie!')
                    break
                else:
                    turn = 'computer'

        else:
            # Computer’s turn.
            move = getComputerMove(theBoard, computerLetter)
            makeMove(theBoard, computerLetter, move)

            if isWinner(theBoard, computerLetter):
                drawBoard(theBoard)
                print('The computer has beaten you! You lose.')
                gameIsPlaying = False
            else:
                if isBoardFull(theBoard):
                    drawBoard(theBoard)
                    print('The game is a tie!')
                    break
                else:
                    turn = 'player'

    if not playAgain():
        break
```

Output:

```
Welcome to Tic Tac Toe!
Do you want to be X or O?
X
The computer will go first.
   |   |
 O |   |
   |   |
-----------
   |   |
   |   |
   |   |
-----------
   |   |
   |   |
   |   |
What is your next move? (1-9)
3
   |   |
 O |   |
   |   |
-----------
   |   |
   |   |
   |   |
-----------
   |   |
 O |   | X
```

```
   |   |
What is your next move? (1-9)
4
   |   |
 O |   | O
   |   |
-----------
   |   |
 X |   |
   |   |
-----------
   |   |
 O |   | X
   |   |
What is your next move? (1-9)
5
   |   |
 O | O | O
   |   |
-----------
   |   |
 X | X |
   |   |
-----------
   |   |
 O |   | X
   |   |
The computer has beaten you! You lose.
Do you want to play again? (yes or no)
no
```

VIVA 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.