GOVERNMENT S K S J TECHNOLOGICAL INSITITUTE

[AFFILIATED TO VISVESWARAIAH TECHNOLOGICAL

UNIVERSITY] K R CIRCLE.BENGALURU 560001

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

VII Semester

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LABORATORY

Subject Code: 18CSL76

FACULTY INCHARGE:
Dr. ShyleshChandra
G ASSOCIATE
PROFESSOR
DEPT OF CSE

PROGRAM LIST

Expt. No.	Program Name
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 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.

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 *(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
  - ❖ Part A – Procedure + Execution + Viva = 6 + 28 + 6 = 40 Marks
  - ❖ Part B – Procedure + Execution + Viva = 9 + 42 + 9 = 60 Marks

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

LAB - 1 Implement A* Search algorithm

```
def aStarAlgo(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] == None:
            pass
        else:
            for (m, weight) in get_neighbors(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 == 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
```

```
def get_neighbors(v):
   if v in Graph_nodes:
       return Graph_nodes[v]
   else:
       return None

def heuristic(n):
   H_dist = {
       "A": 11,
       "B": 6,
       "C": 99,
       "D": 1,
       "E": 7,
       "G": 0,
   }

   return H_dist[n]

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']
['A', 'E', 'D', 'G']
```

LAB - 2 Implement AO* Search algorithm.

```
class Graph:
   def __init__(self, graph, heuristicNodeList, startNode):
       self.graph = graph
       self.H=heuristicNodeList
       self.start=startNode
       self.parent={}
       self.status={}
```

```
        self.solutionGraph={}

    def applyAOStar(self):
        self.aoStar(self.start, False)

    def getNeighbors(self, v):
        return self.graph.get(v,'')

    def getStatus(self,v):
        return self.status.get(v,0)

    def setStatus(self,v, val):
        self.status[v]=val

    def getHeuristicNodeValue(self, n):
        return self.H.get(n,0)

    def setHeuristicNodeValue(self, n, value):
        self.H[n]=value

    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):
        minimumCost=0
        costToChildNodeListDict={}
        costToChildNodeListDict[minimumCost]=[]
        flag=True
        for nodeInfoTupleList in self.getNeighbors(v):
            cost=0
            nodeList=[]
            for c, weight in nodeInfoTupleList:
                cost=cost+self.getHeuristicNodeValue(c)+weight
                nodeList.append(c)
```

```
            if flag==True:
                minimumCost=cost
                costToChildNodeListDict[minimumCost]=nodeList
                flag=False
            else:
                if minimumCost>cost:
                    minimumCost=cost
                    costToChildNodeListDict[minimumCost]=nodeList
        return minimumCost, costToChildNodeListDict[minimumCost]

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

print("--------------------------------------------------------------------------
---------------------")
        if self.getStatus(v) >= 0:
            minimumCost, childNodeList =
self.computeMinimumCostChildNodes(v)
            print(minimumCost, childNodeList)
            self.setHeuristicNodeValue(v, minimumCost)
            self.setStatus(v,len(childNodeList))
            solved=True
            for childNode in childNodeList:
                self.parent[childNode]=v
                if self.getStatus(childNode)!=-1:
                    solved=solved & False
            if solved==True:
                self.setStatus(v,-1)
                self.solutionGraph[v]=childNodeList
            if v!=self.start:
                self.aoStar(self.parent[v], True)
            if backTracking==False:
                for childNode in childNodeList:
                    self.setStatus(childNode,0)
                    self.aoStar(childNode, False)

print ("Graph")
```

```
h1 = {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I':
7, 'J': 1}

graph = {
   '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(graph, h1, 'A')
G1.applyAOStar()
G1.printSolution()
```

Output

```
Graph
HEURISTIC VALUES : {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G':
5, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {}
PROCESSING NODE : A
-----------------------------------------------------------------------
10 ['B', 'C']
HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G':
5, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {}
PROCESSING NODE : B
-----------------------------------------------------------------------
6 ['G']
HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G':
5, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {}
PROCESSING NODE : A
-----------------------------------------------------------------------
10 ['B', 'C']
HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G':
5, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {}
PROCESSING NODE : G
-----------------------------------------------------------------------
8 ['I']
HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G':
8, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {}
PROCESSING NODE : B
-----------------------------------------------------------------------
8 ['H']
```

```
HEURISTIC VALUES : {'A': 10, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G':
8, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {}
PROCESSING NODE : A
-----------------------------------------------------------------------
12 ['B', 'C']
HEURISTIC VALUES : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G':
8, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {}
PROCESSING NODE : I
-----------------------------------------------------------------------
0 []
HEURISTIC VALUES : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G':
8, 'H': 7, 'I': 0, 'J': 1}
SOLUTION GRAPH : {'I': []}
PROCESSING NODE : G
-----------------------------------------------------------------------
1 ['I']
HEURISTIC VALUES : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G':
1, 'H': 7, 'I': 0, 'J': 1}
SOLUTION GRAPH : {'I': [], 'G': ['I']}
PROCESSING NODE : B
-----------------------------------------------------------------------
2 ['G']
HEURISTIC VALUES : {'A': 12, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G':
1, 'H': 7, 'I': 0, 'J': 1}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}
PROCESSING NODE : A
-----------------------------------------------------------------------
6 ['B', 'C']
HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G':
1, 'H': 7, 'I': 0, 'J': 1}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}
PROCESSING NODE : C
-----------------------------------------------------------------------
2 ['J']
HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G':
1, 'H': 7, 'I': 0, 'J': 1}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}
PROCESSING NODE : A
-----------------------------------------------------------------------
6 ['B', 'C']
HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G':
1, 'H': 7, 'I': 0, 'J': 1}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}
PROCESSING NODE : J
-----------------------------------------------------------------------
0 []
HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G':
1, 'H': 7, 'I': 0, 'J': 0}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G'], 'J': []}
PROCESSING NODE : C
```

```
-----------------------------------------------------------------------
1 ['J']
HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 1, 'D': 12, 'E': 2, 'F': 1, 'G':
1, 'H': 7, 'I': 0, 'J': 0}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G'], 'J': [], 'C': ['J']}
PROCESSING NODE : A
-----------------------------------------------------------------------
5 ['B', 'C']
FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE START NODE: A
------------------------------------------------------------
{'I': [], 'G': ['I'], 'B': ['G'], 'J': [], 'C': ['J'], 'A': ['B', 'C']}
------------------------------------------------------------
```

LAB - 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

file = open("lab3ds.csv")
data = list(csv.reader(file))[1:]
concepts = []
target = []

for i in data:
    concepts.append(i[:-1])
    target.append(i[-1])

specific_h = ["0"] * len(concepts[0])
general_h = [["?" for i in range(len(specific_h))] for i in
range(len(specific_h))]

for i, instance in enumerate(concepts):
    if target[i] == "Yes":
        for x in range(len(specific_h)):
            if specific_h[x] == "0":
                specific_h[x] = instance[x]
            elif instance[x] != specific_h[x]:
                specific_h[x] = "?"
                general_h[x][x] = "?"
    if target[i] == "No":
        for x in range(len(specific_h)):
            if instance[x] != specific_h[x]:
                general_h[x][x] = specific_h[x]
            else:
                general_h[x][x] = "?"
```

```
indices = [i for i, val in enumerate(general_h) if val == ["?", "?", "?",
"?", "?", "?"]]

for i in indices:
    general_h.remove(["?", "?", "?", "?", "?", "?"])

print("Final Specific:", specific_h, sep="\n")
print("Final General:", general_h, sep="\n")
```

Output

```
Final Specific:
['Sunny', 'Warm', '?', 'Strong', '?', '?']
Final General:
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]
```

Dataset

Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Cloudy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

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

```
def find_entropy(df):
    Class = df.keys()[-1]
    entropy = 0
    values = df[Class].unique()
    for value in values:
        fraction = df[Class].value_counts()[value] / len(df[Class])
        entropy += -fraction * np.log2(fraction)
    return entropy


def find_entropy_attribute(df, attribute):
    Class = df.keys()[-1]
    target_variables = df[Class].unique()
    variables = df[attribute].unique()
    entropy2 = 0
    for variable in variables:
        entropy = 0
        for target_variable in target_variables:
            num = len(
                df[attribute][df[attribute] == variable][df[Class] ==
target_variable]
```

```
            )
            den = len(df[attribute][df[attribute] == variable])
            fraction = num / (den + eps)
            entropy += -fraction * log(fraction + eps)
        fraction2 = den / len(df)
        entropy2 += -fraction2 * entropy
    return abs(entropy2)


def find_winner(df):
    IG = []
    for key in df.keys()[:-1]:
        IG.append(find_entropy(df) - find_entropy_attribute(df, key))
    return df.keys()[:-1][np.argmax(IG)]

def get_subtable(df, node, value):
    return df[df[node] == value].reset_index(drop=True)

def buildTree(df, tree=None):
    node = find_winner(df)
    attValue = np.unique(df[node])
    if tree is None:
        tree = {}
        tree[node] = {}
    for value in attValue:
        subtable = get_subtable(df, node, value)
        clValue, counts = np.unique(subtable["play"], return_counts=True)
        if len(counts) == 1:
           tree[node][value] = clValue[0]
        else:
            tree[node][value] = buildTree(subtable)
    return tree

import pandas as pd
import numpy as np

eps = np.finfo(float).eps
from numpy import log2 as log

df = pd.read_csv("tennis.csv")
print("\n Given Play Tennis Data Set:\n\n", df)
tree = buildTree(df)
import pprint

pprint.pprint(tree)

test = {"Outlook": "Sunny", "Temperature": "Hot", "Humidity": "High",
"Wind": "Weak"}

def func(test, tree, default=None):
    attribute = next(iter(tree))
    print(attribute)
```

```
    if test[attribute] in tree[attribute].keys():
        print(tree[attribute].keys())
        print(test[attribute])
        result = tree[attribute][test[attribute]]
        if isinstance(result, dict):
            return func(test, result)
        else:
            return result
    else:
        return default

ans = func(test, tree)
print(ans)
```

Dataset

Outlook	Temperature	Humidity	Wind	play
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

Output

```
Given Play Tennis Data Set:

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

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

Outlook
dict_keys(['Overcast', 'Rain', 'Sunny'])
Sunny
Humidity
dict_keys(['High', 'Normal'])
High
No
```

LAB - 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)

x = x/np.amax(x, axis=0)
y = y/100

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

def derivatives_sigmoid(x):
```

```
    return x*(1-x)

epoch = 5
lr = 0.1

inputlayer_neurons = 2
hiddenlayer_neurons = 3
outputlayer_neurons = 1

wh = np.random.uniform(size=(inputlayer_neurons, hiddenlayer_neurons))
bh = np.random.uniform(size=(1, hiddenlayer_neurons))
wout = np.random.uniform(size=(hiddenlayer_neurons, outputlayer_neurons))
bout = np.random.uniform(size=(1, outputlayer_neurons))

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)

    EO = y - output
    outgrad = derivatives_sigmoid(output)
    d_output = EO * outgrad
    EH = d_output.dot(wout.T)
    hiddengrad = derivatives_sigmoid(hlayer_act)
    d_hiddenlayer = EH * hiddengrad
    wout += hlayer_act.T.dot(d_output) * lr
    wh += x.T.dot(d_hiddenlayer) * lr

    print("--Epoch-",i+1,"--Starts--")
    print("Input :\n"+str(x))
    print("Actual Output : \n"+str(y))
    print("Predicted Output : \n", output)
    print("--Epoch-",i+1,"--Ends--")

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

Output

```
--Epoch- 1 --Starts--
Input :
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output :
[[0.92]
 [0.86]
 [0.89]]
Predicted Output :
 [[0.81504223]
 [0.8014937 ]
 [0.81597075]]
--Epoch- 1 --Ends--
--Epoch- 2 --Starts--
Input :
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output :
[[0.92]
 [0.86]
 [0.89]]
Predicted Output :
 [[0.81604173]
 [0.80245966]
 [0.8169656 ]]
--Epoch- 2 --Ends--
--Epoch- 3 --Starts--
Input :
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output :
[[0.92]
 [0.86]
 [0.89]]
Predicted Output :
```

```
 [[0.81702096]
 [0.80340646]
 [0.81794026]]
--Epoch- 3 --Ends--
--Epoch- 4 --Starts--
Input :
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output :
[[0.92]
 [0.86]
 [0.89]]
Predicted Output :
 [[0.81798054]
 [0.80433467]
 [0.81889534]]
--Epoch- 4 --Ends--
--Epoch- 5 --Starts--
Input :
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output :
[[0.92]
 [0.86]
 [0.89]]
Predicted Output :
 [[0.81892105]
 [0.80524483]
 [0.81983142]]
--Epoch- 5 --Ends--
Input :
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output :
[[0.92]
 [0.86]
 [0.89]]
```

```
Predicted Output :
 [[0.81892105]
 [0.80524483]
 [0.81983142]]
```

LAB - 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 pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

data = pd.read_csv('tennis.csv')
print("The first 5 Values of data is :\n", data.head())

X = data.iloc[:, :-1]
print("\nThe First 5 values of the train attributes is\n", X.head())

Y = data.iloc[:, -1]
print("\nThe First 5 values of target values is\n", Y.head())

obj1= LabelEncoder()
X.Outlook = obj1.fit_transform(X.Outlook)
print("\n The Encoded and Transformed Data in Outlook \n",X.Outlook)

obj2 = LabelEncoder()
X.Temperature = obj2.fit_transform(X.Temperature)

obj3 = LabelEncoder()
X.Humidity = obj3.fit_transform(X.Humidity)

obj4 = LabelEncoder()
X.Wind = obj4.fit_transform(X.Wind)
print("\n The Encoded and Transformed Training Examples \n", X.head())

obj5 = LabelEncoder()
```

```
Y = obj5.fit_transform(Y)
print("The class Label encoded in numerical form is",Y)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20)

from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, Y_train)
from sklearn.metrics import accuracy_score
print("Accuracy is: ", accuracy_score(classifier.predict(X_test), Y_test))
```

Output

```
The first 5 Values of data is :
     Outlook Temperature Humidity    Wind Play
0      Sunny         Hot     High    Weak   No
1      Sunny         Hot     High  Strong   No
2   Overcast         Hot     High    Weak  Yes
3       Rain        Mild     High    Weak  Yes
4       Rain        Cool   Normal    Weak  Yes

The First 5 values of the train attributes is
     Outlook Temperature Humidity    Wind
0      Sunny         Hot     High    Weak
1      Sunny         Hot     High  Strong
2   Overcast         Hot     High    Weak
3       Rain        Mild     High    Weak
4       Rain        Cool   Normal    Weak

The First 5 values of target values is
0     No
1     No
2    Yes
3    Yes
4    Yes
Name: Play, dtype: object

The Encoded and Transformed Data in Outlook
0     2
1     2
```

```
2      0
3      1
4      1
5      1
6      0
7      2
8      2
9      1
10     2
11     0
12     0
13     1
Name: Outlook, dtype: int64

The Encoded and Transformed Training Examples
    Outlook  Temperature  Humidity  Wind
0         2            1         0     1
1         2            1         0     0
2         0            1         0     1
3         1            2         0     1
4         1            0         1     1

The class Label encoded in numerical form is [0 0 1 1 1 0 1 0 1 1 1 1 1 0]

Accuracy is: 1.0
```

Dataset

Outlook	Temperature	Humidity	Wind	play
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

LAB - 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.

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

dataset = load_iris()
print("\n IRIS Dataset:\n", dataset.data)
print("\n IRIS Features:\n", dataset.feature_names)
print("\n IRIS Target:\n", dataset.target)
print("\n IRIS Target:\n", dataset.target_names)
```

```
X = pd.DataFrame(dataset.data)
X.columns=['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']

y=pd.DataFrame(dataset.target)
y.columns=['Targets']

print(y)

plt.figure(figsize=(8,5))
colormap=np.array(['red','lime','blue'])

plt.subplot(1,3,1)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y.Targets],s=20)
plt.title('Before Clustering')

plt.subplot(1,3,2)
model = KMeans(n_clusters=3)
model.fit(X)
predY = np.choose(model.labels_,[0,1,2]).astype(np.int64)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[predY],s=20)
plt.title('KMeans')

scaler=preprocessing.StandardScaler()
scaler.fit(X)
xsa=scaler.transform(X)
xs=pd.DataFrame(xsa,columns=X.columns)
gmm=GaussianMixture(n_components=3)
gmm.fit(xs)

y_cluster_gmm=gmm.predict(xs)
plt.subplot(1,3,3)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y_cluster_gmm],s=20)
plt.title('GMM Clustering')
```

Output

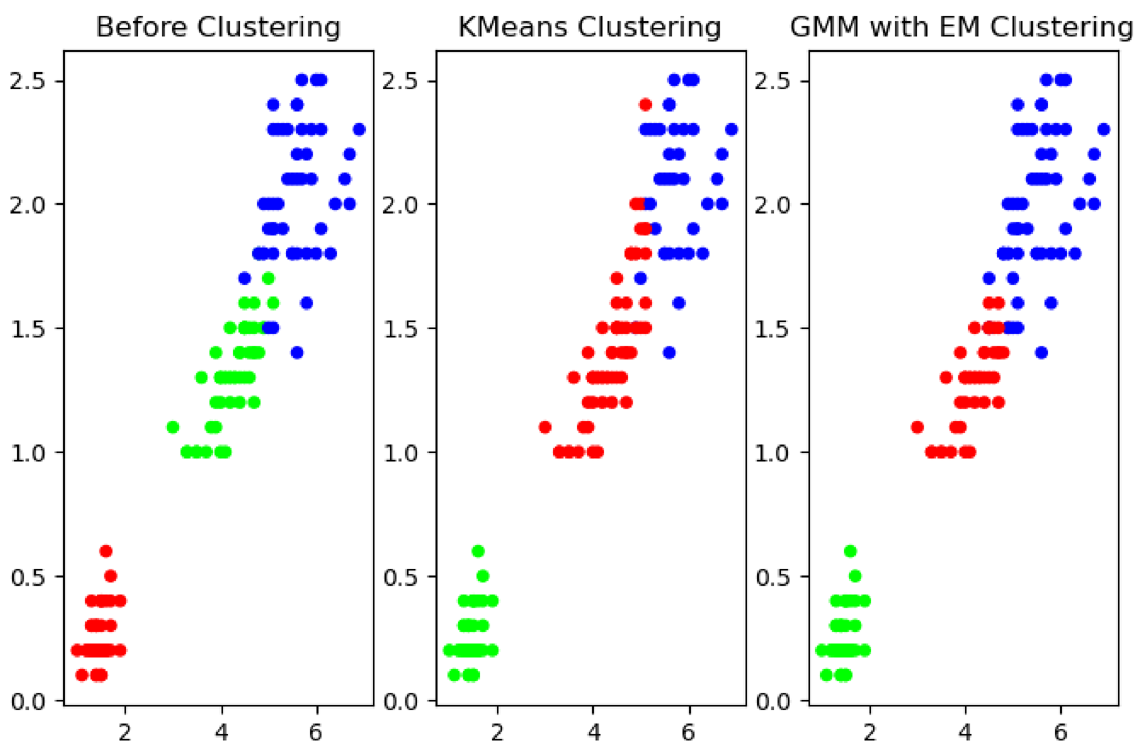
```
IRIS Dataset:
 [[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
```

```
[5.4 3.9 1.7 0.4]
[4.6 3.4 1.4 0.3]
[5.  3.4 1.5 0.2]
[4.4 2.9 1.4 0.2]
[4.9 3.1 1.5 0.1]
[5.4 3.7 1.5 0.2]
[4.8 3.4 1.6 0.2]
[4.8 3.  1.4 0.1]
[4.3 3.  1.1 0.1]
[5.8 4.  1.2 0.2]
[5.7 4.4 1.5 0.4]
[5.4 3.9 1.3 0.4]
[5.1 3.5 1.4 0.3]
[5.7 3.8 1.7 0.3]
[5.1 3.8 1.5 0.3]
[5.4 3.4 1.7 0.2]
[5.1 3.7 1.5 0.4]
[4.6 3.6 1.  0.2]
[5.1 3.3 1.7 0.5]
[4.8 3.4 1.9 0.2]
[5.  3.  1.6 0.2]
[5.  3.4 1.6 0.4]
[5.2 3.5 1.5 0.2]
[5.2 3.4 1.4 0.2]
[4.7 3.2 1.6 0.2]
[4.8 3.1 1.6 0.2]
[5.4 3.4 1.5 0.4]
[5.2 4.1 1.5 0.1]
[5.5 4.2 1.4 0.2]
[4.9 3.1 1.5 0.2]
[5.  3.2 1.2 0.2]
[5.5 3.5 1.3 0.2]
[4.9 3.6 1.4 0.1]
[4.4 3.  1.3 0.2]
[5.1 3.4 1.5 0.2]
[5.  3.5 1.3 0.3]
[4.5 2.3 1.3 0.3]
[4.4 3.2 1.3 0.2]
[5.  3.5 1.6 0.6]
[5.1 3.8 1.9 0.4]
[4.8 3.  1.4 0.3]
[5.1 3.8 1.6 0.2]
[4.6 3.2 1.4 0.2]
[5.3 3.7 1.5 0.2]
[5.  3.3 1.4 0.2]
[7.  3.2 4.7 1.4]
[6.4 3.2 4.5 1.5]
[6.9 3.1 4.9 1.5]
[5.5 2.3 4.  1.3]
[6.5 2.8 4.6 1.5]
[5.7 2.8 4.5 1.3]
[6.3 3.3 4.7 1.6]
```

```
[4.9 2.4 3.3 1. ]
[6.6 2.9 4.6 1.3]
[5.2 2.7 3.9 1.4]
[5.  2.  3.5 1. ]
[5.9 3.  4.2 1.5]
[6.  2.2 4.  1. ]
[6.1 2.9 4.7 1.4]
[5.6 2.9 3.6 1.3]
[6.7 3.1 4.4 1.4]
[5.6 3.  4.5 1.5]
[5.8 2.7 4.1 1. ]
[6.2 2.2 4.5 1.5]
[5.6 2.5 3.9 1.1]
[5.9 3.2 4.8 1.8]
[6.1 2.8 4.  1.3]
[6.3 2.5 4.9 1.5]
[6.1 2.8 4.7 1.2]
[6.4 2.9 4.3 1.3]
[6.6 3.  4.4 1.4]
[6.8 2.8 4.8 1.4]
[6.7 3.  5.  1.7]
[6.  2.9 4.5 1.5]
[5.7 2.6 3.5 1. ]
[5.5 2.4 3.8 1.1]
[5.5 2.4 3.7 1. ]
[5.8 2.7 3.9 1.2]
[6.  2.7 5.1 1.6]
[5.4 3.  4.5 1.5]
[6.  3.4 4.5 1.6]
[6.7 3.1 4.7 1.5]
[6.3 2.3 4.4 1.3]
[5.6 3.  4.1 1.3]
[5.5 2.5 4.  1.3]
[5.5 2.6 4.4 1.2]
[6.1 3.  4.6 1.4]
[5.8 2.6 4.  1.2]
[5.  2.3 3.3 1. ]
[5.6 2.7 4.2 1.3]
[5.7 3.  4.2 1.2]
[5.7 2.9 4.2 1.3]
[6.2 2.9 4.3 1.3]
[5.1 2.5 3.  1.1]
[5.7 2.8 4.1 1.3]
[6.3 3.3 6.  2.5]
[5.8 2.7 5.1 1.9]
[7.1 3.  5.9 2.1]
[6.3 2.9 5.6 1.8]
[6.5 3.  5.8 2.2]
[7.6 3.  6.6 2.1]
[4.9 2.5 4.5 1.7]
[7.3 2.9 6.3 1.8]
[6.7 2.5 5.8 1.8]
```

```
 [7.2 3.6 6.1 2.5]
 [6.5 3.2 5.1 2. ]
 [6.4 2.7 5.3 1.9]
 [6.8 3.  5.5 2.1]
 [5.7 2.5 5.  2. ]
 [5.8 2.8 5.1 2.4]
 [6.4 3.2 5.3 2.3]
 [6.5 3.  5.5 1.8]
 [7.7 3.8 6.7 2.2]
 [7.7 2.6 6.9 2.3]
 [6.  2.2 5.  1.5]
 [6.9 3.2 5.7 2.3]
 [5.6 2.8 4.9 2. ]
 [7.7 2.8 6.7 2. ]
 [6.3 2.7 4.9 1.8]
 [6.7 3.3 5.7 2.1]
 [7.2 3.2 6.  1.8]
 [6.2 2.8 4.8 1.8]
 [6.1 3.  4.9 1.8]
 [6.4 2.8 5.6 2.1]
 [7.2 3.  5.8 1.6]
 [7.4 2.8 6.1 1.9]
 [7.9 3.8 6.4 2. ]
 [6.4 2.8 5.6 2.2]
 [6.3 2.8 5.1 1.5]
 [6.1 2.6 5.6 1.4]
 [7.7 3.  6.1 2.3]
 [6.3 3.4 5.6 2.4]
 [6.4 3.1 5.5 1.8]
 [6.  3.  4.8 1.8]
 [6.9 3.1 5.4 2.1]
 [6.7 3.1 5.6 2.4]
 [6.9 3.1 5.1 2.3]
 [5.8 2.7 5.1 1.9]
 [6.8 3.2 5.9 2.3]
 [6.7 3.3 5.7 2.5]
 [6.7 3.  5.2 2.3]
 [6.3 2.5 5.  1.9]
 [6.5 3.  5.2 2. ]
 [6.2 3.4 5.4 2.3]
 [5.9 3.  5.1 1.8]]

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

 IRIS Target:
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

```
 IRIS Target:
 ['setosa' 'versicolor' 'virginica']
     Targets
0          0
1          0
2          0
3          0
4          0
..       ...
145        2
146        2
147        2
148        2
149        2
```



LAB - 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
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
```

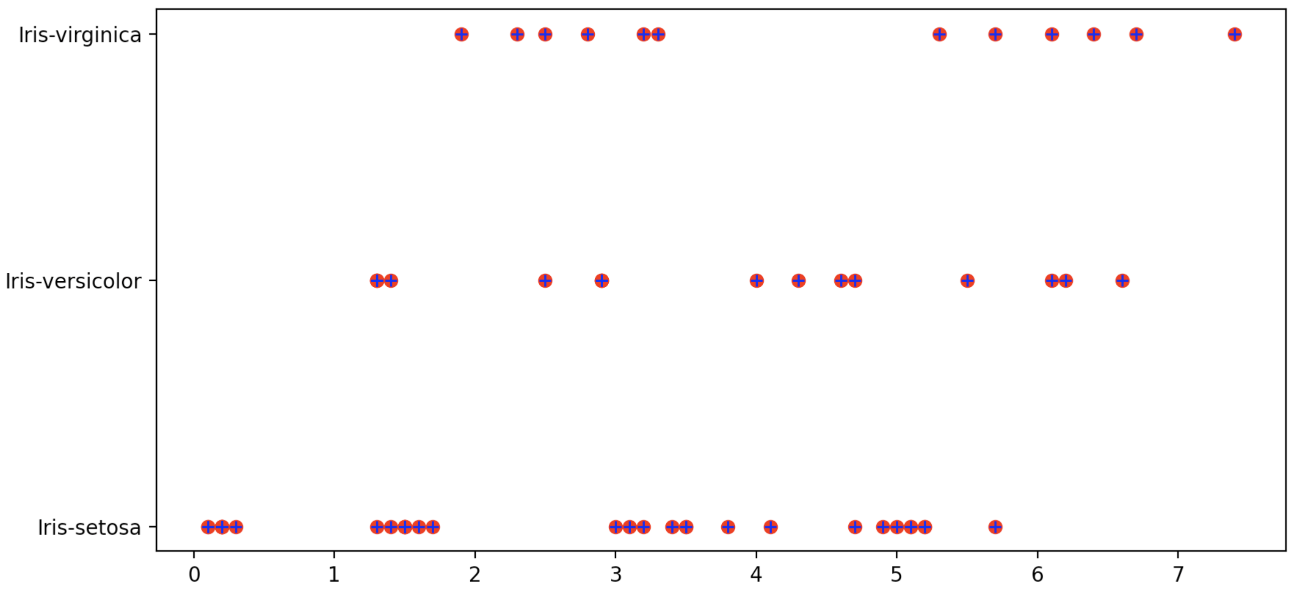
```
from sklearn import metrics
import matplotlib.pyplot as plt

assigned_names = ['sepal-length', 'sepal-width', 'petal-length',
'petal-width', 'Class']

dataset = pd.read_csv("iris2.csv", names=assigned_names)
X = dataset.iloc[:, :-1]
y = dataset.iloc[:, -1]
print(X.head())

Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.10)

classifier = KNeighborsClassifier(n_neighbors=5).fit(Xtrain, ytrain)

ypred = classifier.predict(Xtest)

i = 0
print
("\n---------------------------------------------------------------------")
print ('%-25s %-25s %-25s' % ('Original Label', 'Predicted Label',
'Correct/Wrong'))
print
("-----------------------------------------------------------------------")
for label in ytest:
   print ('%-25s %-25s' % (label, ypred[i]), end="")
   if (label == ypred[i]):
       print (' %-25s' % ('Correct'))
   else:
       print (' %-25s' % ('Wrong'))
   i = i + 1
print
("-----------------------------------------------------------------------")
print("\nConfusion Matrix:\n",metrics.confusion_matrix(ytest, ypred))
print
("-----------------------------------------------------------------------")
print("\nClassification Report:\n",metrics.classification_report(ytest,
ypred))
print
("-----------------------------------------------------------------------")
```

```
print('Accuracy of the classifer is %0.2f' %
metrics.accuracy_score(ytest,ypred))
print
("-------------------------------------------------------------------------")
plt.plot(Xtest,ytest,'ro')
plt.plot(Xtest,ytest,'b+')
```

Output

```
   sepal-length  sepal-width  petal-length  petal-width
0           5.1          3.5           1.4          0.2
1           4.9          3.0           1.4          0.2
2           4.7          3.2           1.3          0.2
3           4.6          3.1           1.5          0.2
4           5.0          3.6           1.4          0.2

-------------------------------------------------------------------------
Original Label             Predicted Label            Correct/Wrong
-------------------------------------------------------------------------
Iris-versicolor            Iris-versicolor            Correct
Iris-setosa                Iris-setosa                Correct
Iris-setosa                Iris-setosa                Correct
Iris-setosa                Iris-setosa                Correct
Iris-virginica             Iris-virginica             Correct
Iris-virginica             Iris-virginica             Correct
Iris-setosa                Iris-setosa                Correct
Iris-setosa                Iris-setosa                Correct
Iris-virginica             Iris-virginica             Correct
Iris-virginica             Iris-virginica             Correct
Iris-versicolor            Iris-versicolor            Correct
Iris-virginica             Iris-virginica             Correct
Iris-versicolor            Iris-versicolor            Correct
Iris-versicolor            Iris-versicolor            Correct
Iris-virginica             Iris-virginica             Correct
-------------------------------------------------------------------------

Confusion Matrix:
 [[5 0 0]
 [0 4 0]
 [0 0 6]]
-------------------------------------------------------------------------

Classification Report:
                  precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         5
Iris-versicolor       1.00      1.00      1.00         4
 Iris-virginica       1.00      1.00      1.00         6
```

```
      accuracy                           1.00        15
     macro avg       1.00      1.00      1.00        15
  weighted avg       1.00      1.00      1.00        15

-------------------------------------------------------------------------
Accuracy of the classifer is 1.00
-------------------------------------------------------------------------
```

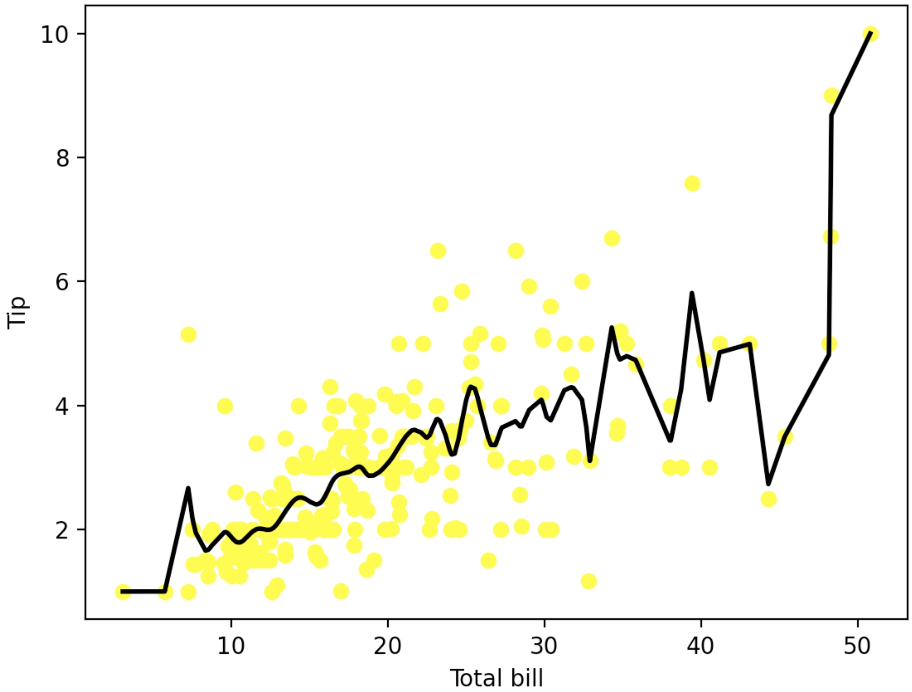


LAB 9 - Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs

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

def kernel(point, xmat, k):
    m,n = np.shape(xmat)
    weights = np.mat(np.eye((m)))
    for j in range(m):
        diff = point - X[j]
        weights[j,j] = np.exp(diff*diff.T/(-2.0*k**2))
    return weights

def localWeight(point, xmat, ymat, k):
    wei = kernel(point,xmat,k)
    W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
    return W
```

```
def localWeightRegression(xmat, ymat, k):
    m,n = np.shape(xmat)
    ypred = np.zeros(m)
    for i in range(m):
        ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
    return ypred

# load data points
data = pd.read_csv('tips.csv')
bill = np.array(data.total_bill)
tip = np.array(data.tip)

mbill = np.mat(bill)
mtip = np.mat(tip)

m= np.shape(mbill)[1]
one = np.mat(np.ones(m))
X = np.hstack((one.T,mbill.T))

#set k here
ypred = localWeightRegression(X,mtip,0.5)
SortIndex = X[:,1].argsort(0)
xsort = X[SortIndex][:,0]

fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.scatter(bill,tip, color='yellow')
ax.plot(xsort[:,1],ypred[SortIndex], color = 'black', linewidth=2)
plt.xlabel('Total bill')
plt.ylabel('Tip')
plt.show();
```

Output



10
8
6
4
2
Tip
10
20
30
40
50
Total bill

1. What is Machine learning?

Machine learning is a field of artificial intelligence that focuses on the development of algorithms and models that can learn from and make predictions or decisions based on data. Machine learning algorithms are able to learn and improve their performance over time by analysing and adapting to new data, without the need for explicit programming or human intervention.

2. Types of ML algorithms?

There are various types of machine learning algorithms, including supervised learning algorithms, which are trained on labelled data and make predictions based on that training; unsupervised learning algorithms, which learn from unlabeled data and can discover patterns and relationships in the data; and reinforcement learning algorithms, which learn from the consequences of their actions and aim to maximise a reward.

3. Applications of ML

There are many applications of machine learning in various fields, some of which include:

1. Image and speech recognition: Machine learning algorithms are used to identify and classify objects, people, and words in images and audio recordings.
2. Natural language processing: Machine learning algorithms are used to understand and interpret human language, such as for language translation or voice-to-text applications.
3. Fraud detection: Machine learning algorithms can analyse patterns in data to detect fraudulent activities, such as credit card fraud or insurance claims fraud.
4. Personalised recommendations: Machine learning algorithms can analyse user data and make personalised recommendations, such as product or content recommendations on e-commerce websites or streaming platforms.
5. Predictive maintenance: Machine learning algorithms can predict when equipment is likely to fail, allowing maintenance to be scheduled before a failure occurs.
6. Self-driving cars: Machine learning algorithms are used to enable autonomous vehicles to make decisions based on data from sensors and cameras.
7. Healthcare: Machine learning algorithms can analyse medical data to predict diseases, suggest treatments, and improve patient outcomes.

These are just a few examples of the many applications of machine learning. As the field continues to advance, machine learning is likely to have an increasing impact on a wide range of industries and applications.

4. Artificial intelligence (AI), machine learning (ML), and deep learning

- These are all related fields that involve the development of algorithms and models that can learn from and make decisions based on data. However, they are not the same thing, and there are some important differences between them:
- Artificial intelligence (AI): AI is a broad field that encompasses the development of intelligent systems that can perceive, reason, and act. AI can be divided into narrow or weak AI, which is designed to perform a specific task, and general or strong AI, which has the ability to exhibit human-like intelligence and perform any intellectual task that a human can.
- Machine learning (ML): ML is a subfield of AI that focuses on the development of algorithms and models that can learn from data and improve their performance over time. ML algorithms are able to learn and adapt to new data without the need for explicit programming, and they can be used for a wide range of applications, such as image and speech recognition, natural language processing, and fraud detection.
- Deep learning: Deep learning is a type of ML that involves the use of artificial neural networks with many layers of interconnected nodes. These networks are able to learn and recognize patterns in data by analysing large amounts of data and adjusting the weights and biases of the nodes in the network. Deep learning is particularly effective for tasks such as image and speech recognition, and it has been used to achieve state-of-the-art results in many areas.
- In summary, AI is a broad field that includes the development of intelligent systems, while ML is a subfield of AI that focuses on the development of algorithms and models that can learn from data. Deep learning is a type of ML that involves the use of artificial neural networks with many layers.

5. Types of Learning

- Learning in machine learning refers to the process of improving a model's performance on a task through experience. A machine learning model is trained on a dataset, and the goal of the training process is to learn patterns and relationships in the data that allow the model to make accurate predictions or decisions.
- There are different types of learning in machine learning, including supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning.
- In supervised learning, the model is trained on labelled data, where the correct output is provided for each input in the training dataset. The model uses this labelled data to learn the relationship between the input and the output, and is then able to make predictions on new, unseen data.
- In unsupervised learning, the model is not provided with labelled training data. Instead, it must learn patterns and relationships in the data by itself. Unsupervised learning is often used for tasks such as clustering and dimensionality reduction.
- In semi-supervised learning, the model is trained on a dataset that is partially labelled. This can be useful in situations where it is expensive or time-consuming to label the entire dataset, but a small amount of labelled data is still available.

- In reinforcement learning, the model learns by interacting with its environment and receiving rewards or punishments based on its actions. This type of learning is often used in tasks such as robot control and game playing.
- Overall, learning in machine learning refers to the process of improving a model's performance on a task through experience and training on a dataset. The specific type of learning depends on the nature of the task and the available data.

6. What are training examples in machine learning?

Training examples are data used to train a machine learning model. They consist of input data (also known as features) and the corresponding desired output (also known as the label or target). Training examples are used to teach the model to make predictions on new, unseen data by adjusting the model's parameters based on the input-output pairs in the training data.

7. What is prediction in machine learning?

Prediction in machine learning refers to the process of using a trained model to make predictions on new, unseen data. A prediction is an output produced by a machine learning model based on a set of input data (also known as features).

8. What are hyperparameters?

- In machine learning, a hyperparameter is a parameter that is not learned from data but is set prior to training. Hyperparameters are used to control the behaviour of a machine learning model and are often chosen through a process called hyperparameter optimization or hyperparameter tuning.
- Some examples of hyperparameters include the learning rate, the regularisation coefficient, the number of hidden units in a neural network, and the type of kernel in a support vector machine.
- Hyperparameters play a crucial role in the performance of a machine learning model and can significantly affect the model's ability to generalise to unseen data. Therefore, it is important to choose appropriate hyperparameters for a given problem.

9. What is convergence in machine learning algorithms?

- In machine learning, convergence refers to the point at which an algorithm has reached a satisfactory solution to a problem. For example, in the case of training a neural network, convergence refers to the point at which the error of the model on the training data is minimised.
- There are several ways in which an algorithm can be said to have converged, including:
  - The algorithm has reached a predefined stopping criterion, such as a maximum number of iterations or a threshold on the error.
  - The error or loss function of the algorithm has reached a minimum or has stopped improving.

    - The parameters of the algorithm have stopped changing significantly or have reached a stable state.
- It is important for an algorithm to converge in order to find a satisfactory solution to a problem. If an algorithm does not converge, it may continue to make changes to the model without improving the model's performance, leading to poor results.