Accredited by NAAC & NBA*

SVCE BENGALURU

SRI VENKATESHWARA COLLEGE OF ENGINEERING

— Affiliated to VTU, Approved by AICTE, Recognised by UGC u/s 2(f) & 12(B)—

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LABORATORY

LABORATORY OVERVIEW

DEGREE:	BE	DEPARTMENT:	ISE
SEMESTER:	7	ACADEMIC YEAR:	2021-22
LABORATORY TITLE:	ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING	LABORATORY CODE:	18CSL76
LAB MANUAL AUTHORS:	AMRUTHA C MARIA NAVIN J R	ASSISTANT PROFESSORS	DEPARTMENT OF ISE
VERIFIED BY:	Dr. PRATHIMA V R	HOD	DEPT. OF ISE

DESCRIPTION

1. COURSE OUTCOMES:

The student should be able to:

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

2. RESOURCES REQUIRED:

1. Hardware resources:

- Desktop PC
- Windows/Linux operating system

2. Software resources:

- Anaconda IDE
- Python

3. Dataset from standard reporsitaries

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LABORATORY

PROGRAMS LIST:

1.	Implement A* Search algorithm.
2.	Implement AO* Search algorithm.
3.	For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples
4.	Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a 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.

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LABORATORY

INTRODUCTION:

1. Artificial intelligence:

Artificial intelligence is the simulation of human intelligence processes by machines, especially computer systems.

2. Machine learning:

Machine learning is a branch of artificial intelligence (AI) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy.

- **How does machine learning works?**

  Machine learning is a form of artificial intelligence (AI) that teaches computers to think in a similar way to how humans do: Learning and improving upon past experiences. It works by exploring data and identifying patterns, and involves minimal human intervention.

3. Types of Machine Learning:

1. Supervised.
2. Unsupervised.
3. Reinforcement.

➢ **Supervised Learning:**

Supervised learning is one of the most basic types of machine learning. In this type, the machine learning algorithm is trained on labeled data. Even though the data needs to be labeled accurately for this method to work, supervised learning is extremely powerful when used in the right circumstances.

- **Unsupervised Learning:**

Unsupervised machine learning holds the advantage of being able to work with unlabeled data. This means that human labor is not required to make the dataset machine-readable, allowing much larger datasets to be worked on by the program.

- **Reinforcement Learning:**

Reinforcement learning directly takes inspiration from how human beings learn from data in their lives. It features an algorithm that improves upon itself and learns from new situations using a trial-and-error method. Favorable outputs are encouraged or 'reinforced', and non-favorable outputs are discouraged or 'punished'.

Note:

- Classification and Regression are types of supervised learning algorithms
- Clustering is a type of unsupervised algorithm.

4. Classification:

Classification is a type of supervised machine learning algorithm. For any given input, the classification algorithms help in the prediction of the class of the output variable. There can be multiple types of classifications like binary classification, multi-class classification, etc. It depends upon the number of classes in the output variable.

Types of Classification algorithms:

- **Logistic Regression:**

It is one of the linear models which can be used for classification. It uses the sigmoid function to calculate the probability of a certain event occurring. It is an ideal method for the classification of binary variables.

➢ K-Nearest Neighbours (KNN):

It uses distance metrics like Euclidean distance, Manhattan distance, etc. to calculate the distance of one data point from every other data point. To classify the output, it takes a majority vote from k nearest neighbors of each data point.

➢ Decision Trees:

It is a non-linear model that overcomes a few of the drawbacks of linear algorithms like Logistic regression. It builds the classification model in the form of a tree structure that includes nodes and leaves. This algorithm involves multiple if-else statements which help in breaking down the structure into smaller structures and eventually providing the final outcome. It can be used for regression as well as classification problems.

➢ Random Forest:

It is an ensemble learning method that involves multiple decision trees to predict the outcome of the target variable. Each decision tree provides its own outcome. In the case of the classification problem, it takes the majority vote of these multiple decision trees to classify the final outcome. In the case of the regression problem, it takes the average of the values predicted by the decision trees.

➢ Naïve Bayes:

It is an algorithm that is based upon Bayes' theorem. It assumes that any particular feature is independent of the inclusion of other features. i.e. They are not correlated to one another. It generally does not work well with complex data due to this assumption as in most of the data sets there exists some kind of relationship between the features.

➢ Support Vector Machine:

It represents the data points in multi-dimensional space. These data points are then segregated into classes with the help of hyperplanes. It plots an n-dimensional space for the n number of features in the dataset and then tries to create the hyperplanes such that it divides the data points with maximum margin.

5. Clustering:

Clustering is a type of unsupervised machine learning algorithm. It is used to group data points having similar characteristics as clusters. Ideally, the data points in the same cluster should exhibit similar properties and the points in different clusters should be as dissimilar as possible.

Clustering is divided into two groups:

- Hard clustering.
- Soft clustering.

In hard clustering, the data point is assigned to one of the clusters only whereas in soft clustering, it provides a probability likelihood of a data point to be in each of the clusters.

Types of Clustering algorithms:

- **K-Means Clustering**:

It initializes a pre-defined number of k clusters and uses distance metrics to calculate the distance of each data point from the centroid of each cluster. It assigns the data points into one of the k clusters based on its distance.

- **Agglomerative Hierarchical Clustering (Bottom-Up Approach)**:

It considers each data point as a cluster and merges these data points on the basis of distance metric and the criterion which is used for linking these clusters.

- **Divisive Hierarchical Clustering (Top-Down Approach)**:

It initializes with all the data points as one cluster and splits these data points on the basis of distance metric and the criterion. Agglomerative and Divisive clustering can be represented as a dendrogram and the number of clusters to be selected by referring to the same.

- **DBSCAN (Density-based Spatial Clustering of Applications with Noise):**

It is a density-based clustering method. Algorithms like K-Means work well on the clusters that are fairly separated and create clusters that are spherical in shape. DBSCAN is used when the data is in arbitrary shape and it is also less sensitive to the outliers. It groups the data points that have many neighbouring data points within a certain radius.

- **OPTICS (Ordering Points to Identify Clustering Structure)**:

It is another type of density-based clustering method and it is similar in process to DBSCAN except that it considers a few more parameters. But it is more computationally complex than DBSCAN. Also, it does not separate the data points into clusters, but it creates a reachability plot which can help in the interpretation of creating clusters.

- **BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies)**:

It creates clusters by generating a summary of the data. It works well with huge datasets as it first summarises the data and then uses the same to create clusters.

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LABORATORY

PROGRAMS:

1. EXPERIMENT NO: 1

2. TITLE: Implement A* Search algorithm.

3. THEORY: This Algorithm is the advanced form of the BFS algorithm (Breadth-first search), which searches for the shorter path first than, the longer paths. It is a **complete** as well as an **optimal** solution for solving path and grid problems.

Optimal – find the least cost from the starting point to the ending point.

Complete – It means that it will find all the available paths from start to end.

4. ALGORITHM:

1. Firstly, Place the starting node into OPEN and find its f (n) value.

2. Then remove the node from OPEN, having the smallest f (n) value. If it is a goal node, then stop and return to success.

3: Else remove the node from OPEN, and find all its successors.

4: Find the f (n) value of all the successors, place them into OPEN, and place the removed node into CLOSE.

5: Goto Step-2.

6: Exit.

5. SOLUTION:

```
def aStarAlgo(start_node, stop_node):

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

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


        while len(open_set) > 0:
            n = None

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


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

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

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

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

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

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

            path.append(start_node)

            path.reverse()

            print('Path found: {}'.format(path))
            return path
        # remove n from the open_list, and add it to closed_list
        # because all of his neighbors were inspected
```

```
            open_set.remove(n)
            closed_set.add(n)

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

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

        }

        return H_dist[n]

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

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

6. OUTPUT:

Path found: ['A', 'E', 'D', 'G']

1. EXPERIMENT NO: 2

2. TITLE: Implement AO* Search algorithm.

3. THEORY:

AO* Algorithm basically based on problem decompositon (Breakdown problem into small pieces)
When a problem can be divided into a set of sub problems, where each sub problem can be solved separately and a combination of these will be a solution, AND-OR graphs or AND - OR trees are used for representing the solution.

4. ALGORITHM:

Step-1: Create an initial graph with a single node (start node).
Step-2: Transverse the graph following the current path, accumulating node that has not yet been expanded or solved.
Step-3: Select any of these nodes and explore it. If it has no successors then call this value- FUTILITY else calculate f'(n) for each of the successors.
Step-4: If **f'(n)=0**, then mark the node as **SOLVED**.
Step-5: Change the value of f'(n) for the newly created node to reflect its successors by backpropagation.
Step-6: Whenever possible use the most promising routes, If a node is marked as SOLVED then mark the parent node as SOLVED.
Step-7: If the starting node is SOLVED or value is greater than **FUTILITY** then stop else repeat from Step-2.

5. SOLUTION:

```
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): # 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:
                print("c, weight",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
            print("Cost to Child Node",costToChildNodeListDict)
            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}
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()
```

6. OUTPUT:

FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE START NODE: A

{'I': [], 'G': ['I'], 'B': ['G'], 'J': [], 'C': ['J'], 'A': ['B', 'C']}

1. EXPERIMENT NO: 3

2. TITLE: 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

3. THEORY:

- The key idea in the Candidate-Elimination algorithm is to output a description of the set of all hypotheses consistent with the training examples.
- It computes the description of this set without explicitly enumerating all of its members.
- This is accomplished by using the more-general-than partial ordering and maintaining a compact representation of the set of consistent hypotheses.
- The algorithm represents the set of all hypotheses consistent with the observed training examples. This subset of all hypotheses is called the version space with respect to the hypothesis space H and the training examples D, because it contains all plausible versions ofthe target concept.
- A version space can be represented with its general and specific boundary sets.
- The Candidate-Elimination algorithm represents the version space by storing only its mostgeneral members G and its most specific members S.
- Given only these two sets S and G, it is possible to enumerate all members of a version space by generating hypotheses that lie between these two sets in general-to-specific partial ordering over hypotheses. Every member of the version space lies between these boundaries.

4. ALGORITHM:

Step1: Load Data set
Step2: Initialize General Hypothesis and Specific Hypothesis.
Step3: For each training example
Step4: If example is positive example
    if attribute_value == hypothesis_value:
      Do nothing
    else:

replace attribute value with '?' (Basically generalizing it)

Step5: If example is Negative example

Make generalize hypothesis more specific.

5. DATASET:

sky	airtemp	humidity	wind	water	forcast	enjoysport
sunny	warm	normal	strong	warm	same	yes
sunny	warm	high	strong	warm	same	yes
rainy	cold	high	strong	warm	change	no
sunny	warm	high	strong	cool	change	yes

6. SOLUTION:

```
import numpy as np
import pandas as pd
data=pd.DataFrame(data=pd.read_csv('D:/enjoysport.csv'))
print(data)
concepts=np.array(data.iloc[:,0:-1])
print(concepts)
target=np.array(data.iloc[:,-1])
print(target)
def learn(concepts,target):
    specific_h=concepts[0].copy()
    print("initialization of specific_h and general_h")
    print(specific_h)
    general_h=[["?" for i in range(len(specific_h))] for i in
range(len(specific_h))]
    print(general_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("steps of candidate Elimination Algorithm",i+1)
            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("\nFinal Specific_h:", s_final, sep="\n")
print("\nFinal General_h:", g_final, sep="\n")
```

7. OUTPUT:

```
Final Specific_h:
['sunny' 'warm' '?' 'strong' '?' '?']

Final General_h:
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

1. EXPERIMENT NO: 4

2. TITLE: 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

3. THEORY:

- ID3 algorithm is a basic algorithm that learns decision trees by constructing them topdown, beginning with the question "which attribute should be tested at the root of the tree?".
- To answer this question, each instance attribute is evaluated using a statistical test to determine how well it alone classifies the training examples. The best attribute is selected and used as the test at the root node of the tree.
- A descendant of the root node is then created for each possible value of this attribute, and the training examples are sorted to the appropriate descendant node (i.e., down the branch corresponding to the example's value for this attribute).
- The entire process is then repeated using the training examples associated with each descendant node to select the best attribute to test at that point in the tree.
- A simplified version of the algorithm, specialized to learning boolean-valued functions (i.e.,concept learning), is described below.

4. ALGORITHM:

ID3 (Examples, Target_Attribute, Attributes)
Create a root node for the tree
If all examples are positive, Return the single-node tree Root, with label = +.
If all examples are negative, Return the single-node tree Root, with label = -.
If number of predicting attributes is empty, then Return the single node tree Root,
with label = most common value of the target attribute in the examples.
Otherwise Begin
A ← The Attribute that best classifies examples.
Decision Tree attribute for Root = A.

For each possible value, v_i, of A,
Add a new tree branch below Root, corresponding to the test A = v_i.
Let Examples (v_i) be the subset of examples that have the value v_i for A.
If Example v_i is empty
Then below this new branch add a leaf node with label = most common target value in the examples
Else below this new branch add the subtree ID3 (Examples(v_i), Target_Attribute, Attributes – {A})
End
Return Root

5. DATASET:

Outlook	Temperature	Humidity	Wind	Target
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

6. SOLUTION:

```
import pandas as pd
import math
import numpy as np

data = pd.read_csv("D:/dataset.csv")
```

```
features = [feat for feat in data]
features.remove("answer")

class Node:
    def __init__(self):
        self.children = []
        self.value = ""
        self.isLeaf = False
        self.pred = ""

def entropy(examples):
    pos = 0.0
    neg = 0.0
    for _, row in examples.iterrows():
        if row["answer"] == "yes":
            pos += 1
        else:
            neg += 1
    if pos == 0.0 or neg == 0.0:
        return 0.0
    else:
        p = pos / (pos + neg)
        n = neg / (pos + neg)
        return -(p * math.log(p, 2) + n * math.log(n, 2))

def info_gain(examples, attr):
    uniq = np.unique(examples[attr])
    #print ("\n",uniq)
    gain = entropy(examples)
    #print ("\n",gain)
    for u in uniq:
        subdata = examples[examples[attr] == u]
        #print ("\n",subdata)
        sub_e = entropy(subdata)
        gain -= (float(len(subdata)) / float(len(examples))) * sub_e
```

```
            #print ("\n",gain)
        return gain

    def ID3(examples, attrs):
        root = Node()

        max_gain = 0
        max_feat = ""
        for feature in attrs:
            #print ("\n",examples)
            gain = info_gain(examples, feature)
            if gain > max_gain:
                max_gain = gain
                max_feat = feature
        root.value = max_feat
        #print ("\nMax feature attr",max_feat)
        uniq = np.unique(examples[max_feat])
        #print ("\n",uniq)
        for u in uniq:
            #print ("\n",u)
            subdata = examples[examples[max_feat] == u]
            #print ("\n",subdata)
            if entropy(subdata) == 0.0:
                newNode = Node()
                newNode.isLeaf = True
                newNode.value = u
                newNode.pred = np.unique(subdata["answer"])
                root.children.append(newNode)
            else:
                dummyNode = Node()
                dummyNode.value = u
                new_attrs = attrs.copy()
                new_attrs.remove(max_feat)
                child = ID3(subdata, new_attrs)
                dummyNode.children.append(child)
```

```
                root.children.append(dummyNode)
            return root

        def printTree(root: Node, depth=0):
            for i in range(depth):
                print("\t", end="")
            print(root.value, end="")
            if root.isLeaf:
                print(" -> ", root.pred)
            print()
            for child in root.children:
                printTree(child, depth + 1)

        root = ID3(data, features)
printTree(root)
```

7. OUTPUT:

```
outlook
        overcast ->  ['yes']

        rain
                wind
                        strong ->  ['no']

                        weak ->  ['yes']

        sunny
                humidity
                        high ->  ['no']

                        normal ->  ['yes']
```

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LABORATORY

1. EXPERIMENT NO: 5

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

3. THEORY:

- Artificial neural networks (ANNs) provide a general, practical method for learning real- valued, discrete-valued, and vector-valued functions from examples.
- Algorithms such as BACKPROPAGATION gradient descent to tune network parameters tobest fit a training set of input-output pairs.
- ANN learning is robust to errors in the training data and has been successfully applied to problems such as interpreting visual scenes, speech recognition, and learning robot controlstrategies.

4. ALGORITHM:

1. Create a feed-forward network with ni inputs, nhidden hidden units, and nout output units.
2. Initialize each wi to some small random value (e.g., between -.05 and .05).
3. Until the termination condition is met, do

   For each training example <(x1,…xn),t>, do

   // Propagate the input forward through the network:

   a. Input the instance (x1, ..,xn) to the n/w & compute the n/w outputs ok for every unit

   // Propagate the errors backward through the network:

   b. For each output unit k, calculate its error term δk ; o k = ok(1-ok)(tk-ok)

   c. For each hidden unit h, calculate its error term oh; oh=oh(1-oh) Σk wh,k δk

   d. For each network weight wi,j do; wi,j=wi,j+Δwi,j where Δwi,j= η δj xi,j

5. SOLUTION:

```
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)      # 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)
#Variable initialization
epoch = 5000                    #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))
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
for i in range(epoch):

#Forward Propogation
    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)

#how much hidden layer wts contributed to error
    hiddengrad = derivatives_sigmoid(hlayer_act)
    d_hiddenlayer = EH * hiddengrad
```

```
# dotproduct of nextlayererror and currentlayerop
    wout += hlayer_act.T.dot(d_output) *lr
    wh += X.T.dot(d_hiddenlayer) *lr
print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
```

6. OUTPUT:

```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.89504121]
 [0.88252504]
 [0.8926262 ]]
```

1. EXPERIMENT NO: 6

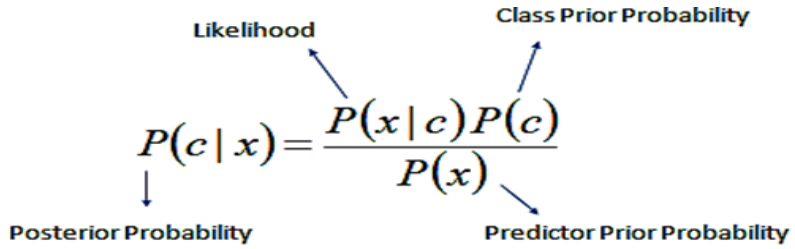
2. TITLE: Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

3. THEORY & ALGOTITHM:

Naive Bayes algorithm is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as 'Naive'.

Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c). Look at the equation below:



$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

where

P(c|x) is the posterior probability of class (c, target) given predictor (x, attributes).
P(c) is the prior probability of class.
P(x|c) is the likelihood which is the probability of predictor given class.
P(x) is the prior probability of predictor.

The naive Bayes classifier applies to learning tasks where each instance x is described by a conjunction of attribute values and where the target function f (x) can take on any value from some finite set V. A set of training examples of the target function is provided, and a new instance is presented, described by the tuple of attribute values (a1, a2, ... ,an). The learner is asked to predict the target value, or classification, for this new instance.

The Bayesian approach to classifying the new instance is to assign the most probable target value,

v_{MAP}, given the attribute values (a_1, a_2, ..., a_n) that describe the instance.

$$v_{MAP} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j | a_1, a_2 \ldots a_n)$$

We can use Bayes theorem to rewrite this expression as

$$v_{MAP} = \underset{v_j \in V}{\operatorname{argmax}} \frac{P(a_1, a_2 \ldots a_n | v_j) P(v_j)}{P(a_1, a_2 \ldots a_n)}$$

$$= \underset{v_j \in V}{\operatorname{argmax}} P(a_1, a_2 \ldots a_n | v_j) P(v_j)$$

Now we could attempt to estimate the two terms in Equation based on the training data. It is easy to estimate each of the P(vj) simply by counting the frequency with which each target value vj occurs in the training data.

The naive Bayes classifier is based on the simplifying assumption that the attribute values are conditionally independent given the target value. In other words, the assumption is that given the target value of the instance, the probability of observing the conjunction $a_1, a_2, \ldots, a_n$, is just the product of the probabilities for the individual attributes: $P(a_1, a_2, \ldots, a_n \mid v_j) = \Pi_i P(a_i \mid v_j)$. Substituting this, we have the approach used by the naive Bayes classifier.

$$v_{NB} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j) \prod_i P(a_i|v_j)$$

where v_{NB} denotes the target value output by the naive Bayes classifier.

When dealing with continuous data, a typical assumption is that the continuous values associated with each class are distributed according to a Gaussian distribution. For example, suppose the training data contains a continuous attribute, x. We first segment the data by the class, and then compute the mean and variance of x in each class.

Let μ be the mean of the values in x associated with class Ck, and let $\sigma^2 k$ be the variance of the values in x associated with class Ck. Suppose we have collected some observation value v. Then, the probability distribution of v given a class Ck, p(x=v|Ck) can be computed by plugging v into the equation for a Normal distribution parameterized by μ and $\sigma^2 k$. *That is*

$$p(x = v \mid C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{-\frac{(v-\mu_k)^2}{2\sigma_k^2}}$$

4. DATASET:

Outlook	Temperature	Humidity	Wind	Target
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

5. SOLUTION:

```
import numpy as np

import math

import csv


def read_data(filename):


    with open(filename,'r') as csvfile:
```

```
        datareader = csv.reader(csvfile)
        metadata = next(datareader)
        traindata=[]
        for row in datareader:
            traindata.append(row)

    return (metadata, traindata)

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

def classify(data,test):

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

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

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

probYes=countYes/total_size
probNo= countNo / total_size

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



prob0 =np.zeros((test.shape[1]-1))
prob1 =np.zeros((test.shape[1]-1))
```

```
accuracy=0
print("instance prediction  target")


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


    probno=probNo
    probyes=probYes
    for i in range(test.shape[1]-1):
        probno=probno*prob0[i]
        probyes=probyes*prob1[i]
    if probno>probyes:
```

```
        predict='no'
    else:
        predict='yes'

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

metadata,traindata= read_data("d:/dataset.csv")
splitRatio=0.6
trainingset, testset=splitDataset(traindata, splitRatio)
training=np.array(trainingset)
testing=np.array(testset)

classify(training,testing)
```

6. OUTPUT:

```
training data size= 8
test data size= 6
target   count   probability
Yes      4       0.5
No       4       0.5
```

```
instance prediction  target
1        no          yes
2        yes         yes
3        no          yes
4        yes         yes
5        yes         yes
6        no          no
accuracy 66.66666666666666 %
```

1. EXPERIMENT NO: 7

2. TITLE: 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.

3. THEORY & ALGORITHM:

Expectation Maximization algorithm

- The basic approach and logic of this clustering method is as follows.
- Suppose we measure a single continuous variable in a large sample of observations. Further, suppose that the sample consists of two clusters of observations with different means (and perhaps different standard deviations); within each sample, the distribution of values for the continuous variable follows the normal distribution.
- The goal of EM clustering is to estimate the means and standard deviations for each cluster so as to maximize the likelihood of the observed data (distribution).
- Put another way, the EM algorithm attempts to approximate the observed distributions of values based on mixtures of different distributions in different clusters. The results of EM clustering are different from those computed by k-means clustering.
- The latter will assign observations to clusters to maximize the distances between clusters. The EM algorithm does not compute actual assignments of observations to clusters, but classification probabilities.
- In other words, each observation belongs to each cluster with a certain probability. Ofcourse, as a final result we can usually review an actual assignment of observations to clusters, based on the (largest) classification probability.

K means Clustering

- The algorithm will categorize the items into k groups of similarity. To calculate that similarity, we will use the euclidean distance as measurement.
- The algorithm works as follows:
  1. First we initialize k points, called means, randomly.
  2. We categorize each item to its closest mean and we update the mean's coordinates, which are the averages of the items categorized in that mean so far.
  3. We repeat the process for a given number of iterations and at the end, we have ourclusters.
- The "points" mentioned above are called means, because they hold the mean values of the items categorized in it. To initialize these means, we have a lot of options. An intuitive method is to initialize the means at random items in the data set. Another method is to initialize the means at random values between the boundaries of the data set (if for a feature x the items have values in [0,3], we will initialize the means with values for x at [0,3]).

Pseudocode:

1. Initialize k means with random values
2. For a given number iterations:
   Iterate through items:
   Find the mean closest to the item
   Assign item to mean
   Update mean

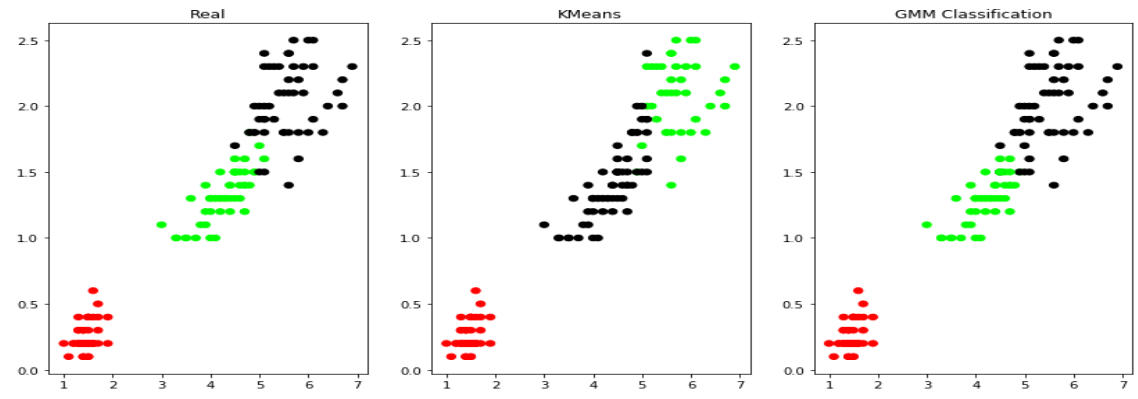
4. SOLUTION:

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

dataset=load_iris()
# print(dataset)

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

plt.figure(figsize=(14,7))
colormap=np.array(['red','lime','black'])

# REAL PLOT
plt.subplot(1,3,1)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y.Targets],s=40)
plt.title('Real')
```

```
# K-PLOT
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=40)
plt.title('KMeans')

# GMM PLOT
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=40)
plt.title('GMM Classification')
```

5. OUTPUT:

```
Text(0.5, 1.0, 'GMM Classification')
```



ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LABORATORY

1. EXPERIMENT NO: 8

2. TITLE: 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.

3. THEORY:

- K-Nearest Neighbors is one of the most basic yet essential classification algorithms in Machine Learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining and intrusion detection.
- It is widely disposable in real-life scenarios since it is non-parametric, meaning, it does not make any underlying assumptions about the distribution of data.

4. ALGORITHM:

Input: Let m be the number of training data samples. Let p be an unknown point.Method:

1. Store the training samples in an array of data points arr[]. This means each elementof this array represents a tuple (x, y).
2. for i=0 to m
   Calculate Euclidean distance d(arr[i], p).
3. Make set S of K smallest distances obtained. Each of these distances correspond toan already classified data point.
4. Return the majority label among S.

5. SOLUTION:

```
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import numpy as np

dataset=load_iris()
X_train,X_test,y_train,y_test=train_test_split(dataset["data"],dataset["target"
],random_state=0)
```

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

for i in range(len(X_test)):
    x=X_test[i]
    x_new=np.array([x])
    prediction=kn.predict(x_new)
    print("TARGET=",y_test[i],dataset["target_names"]
[y_test[i]],"PREDICTED=",prediction,dataset["target_names"][prediction])

print(kn.score(X_test,y_test))
```

6. OUTPUT:

```
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 1 versicolor PREDICTED= [2] ['virginica']
0.9736842105263158
```

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LABORATORY

1. EXPERIMENT NO: 9

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

3. THEORY:

- Given a dataset X, y, we attempt to find a linear model h(x) that minimizes residual sum ofsquared errors. The solution is given by Normal equations.
- Linear model can only fit a straight line, however, it can be empowered by polynomial features to get more powerful models. Still, we have to decide and fix the number and typesof features ahead.
- Alternate approach is given by locally weighted regression.
- Given a dataset X, y, we attempt to find a model h(x) that minimizes residual sum ofweighted squared errors.
- The weights are given by a kernel function which can be chosen arbitrarily and in my case Ichose a Gaussian kernel.
- The solution is very similar to Normal equations, we only need to insert diagonal weightmatrix W.

4. ALGORITHM:

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

        # fit model: normal equations with
      kernel
        xw = X.T * radial_kernel(x0, X, tau)
        beta = np.linalg.pinv(xw @ X) @ xw @ Y
```

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

5. DATASET:

Tips.csv(256 rows)

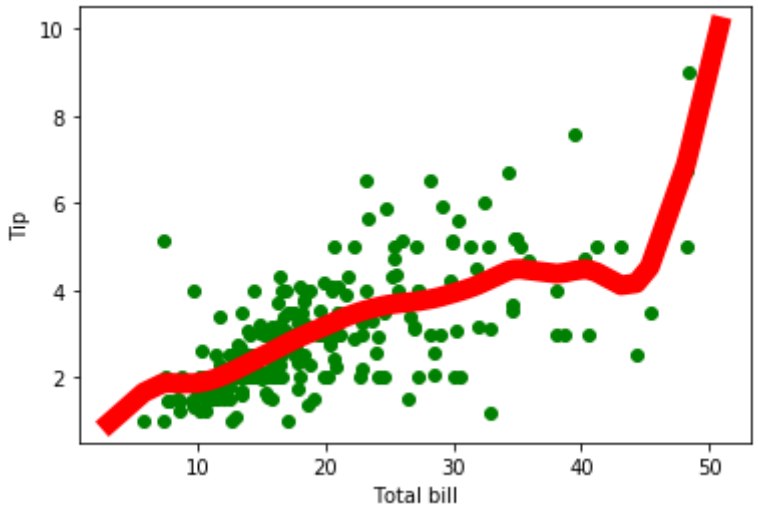
6. SOLUTION:

```
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('D:/10-Dataset.csv')
bill = np.array(data.total_bill)
tip = np.array(data.tip)

#preparing and add 1 in bill
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,2)
SortIndex = X[:,1].argsort(0)
```

```
xsort = X[SortIndex][:,0]


fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.scatter(bill,tip, color='green')
ax.plot(xsort[:,1],ypred[SortIndex], color = 'red', linewidth=10)
plt.xlabel('Total bill')
plt.ylabel('Tip')
plt.show();
```

7. OUTPUT:



ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LABORATORY

VIVA QUESTIONS:

- What is A* algorithm.
- What is the formula used in A* algorithm.
- What is AO* algorithm.
- What is the formula used in AO* algorithm.
- What is Artificial Intelligence.
- What is Machine Learning.
- Define Supervised learning.
- Define Unsupervised learning.
- Define Reinforcement learning.
- What is Classification.
- What is clustering.
- What do you mean by hypothesis.
- What is Gain.
- What is Entropy.
- How k-means clustering is different from KNN.
- State Bayes Theorem.
- Define Bias.
- What is learning rate? Why it is needed.
- Why KNN algorithm is known as lazy learning algorithm.
- What is K-means clustering algorithm.
- What are the advantages and disadvantages of K-means clustering algorithm.
- What is K in KNN algorithm.
- What are the advantages and disadvantages of KNN algorithm.
- What are the some applications of K-Means clustering.
- What is the difference between training data and test data.
- What is naïve in naïve bayes algorithm.