Exp No: 1
Date:

FIND-S ALGORITHM

Aim:

To implement FIND-S algorithm for finding the most specific hypothesis on a given input.

Algorithm:

Step 1: Initialize h to the most specific hypothesis in H

Step 2:For each positive training instance x

For each attribute constraint ai in h

If the constraint ai is satisfied by x.

Then do nothing

Else replace ai in h by the next more general constraint that is satisfied by x

Step 3: Output hypothesis h

Program:

```
import csv
num_attributes = 6 a = []
print("\n The given input is \n")
with open('enjoysport.csv', 'r') as csvfile:
reader = csv.reader(csvfile)
for row in reader:
 a.append(row)
 print(row)
 print("\n The initial value of hypothesis:")
 hypothesis=['0']* num_attributes
 print(hypothesis)
 for j in range(0,num_attributes):
 hypothesis[j] = a[0][j];
 print("\n Find S: Finding a Maximally Specific Hypothesis\n")
 for i in range(0,len(a)):
 if a[i][num_attributes]=='yes':
 for j in range(0,num_attributes):
 if a[i][j]!=hypothesis[j]:
 hypothesis[j]='?' else :
 hypothesis[j]= a[i][j]
 print(" For Training instance No:{0} the hypothesis is ".format(i),hypothesis)
 print("\n The Maximally Specific Hypothesis for a given input :\n")
 print(hypothesis)
```

Input:

Sunny	Warm	normal	strong	warm	same	Yes
Sunny	Warm	High	strong	warm	same	Yes
Rainy	Cold	High	strong	warm	change	No
Sunny	Warm	High	strong	cool	change	Yes

Output:

The given input is

['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes']
['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'yes']
['rainy', 'cold', 'high', 'strong', 'warm', 'change', 'no']
['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'yes']

The initial value of hypothesis:
['0', '0', '0', '0', '0', '0']

Find S: Finding a Maximally Specific Hypothesis For Training Example No:0 the hypothesis is
['sunny', 'warm', 'normal', 'strong', 'warm', 'same']

For Training Example No:1 the hypothesis is ['sunny', 'warm', '?', 'strong', 'warm', 'same']

For Training Example No:2 the hypothesis is ['sunny', 'warm', '?', 'strong', 'warm', 'same']

For Training Example No:3 the hypothesis is 'sunny', 'warm', '?', 'strong', '?', '?']

The Maximally Specific Hypothesis for a given Training Examples:
['sunny', 'warm', '?', 'strong', '?', '?']

PAAVAI ENGINEERING COLLEGE (Autonomous)		
DESCRIPTION	MAX. MARKS	MARKS AWARDED
Preparation & Conduction	10	
Observation & Results	20	
Record Completion	05	
Viva Voce	05	
TOTAL	40	

Result:

Exp No: 2
Date:

Candidate-Elimination Algorithm

Aim:

To implement the Candidate-Elimination algorithm for producing a set of all hypotheses consistent with the given input.

Algorithm:

Step 1: Initialize G to the set of maximally general hypotheses in H

Step 2: Initialize S to the set of maximally specific hypotheses in H. For each input d, do

Step 3: If d is a positive input

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

Add to S all minimal generalizations h of s such that h is consistent with d, and some member of G is more general than h
Remove from S any hypothesis that is more general than another hypothesis in S

Step 4: If d is a negative example

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

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

Program:

```
import numpy as np
import pandas as pd
data = pd.DataFrame(data=pd.read_csv('enjoysport.csv'))
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 i, h in enumerate(concepts):
if target[i] == "yes":
if h[x]!= specific_h[x]:
specific_h[x] ='?' general_h[x][x] ='?'
print(specific_h)
print(specific_h)
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("Final Specific_h:", s_final, sep="\n")
print("Final General_h:", g_final, sep="\n")
```

Input:

Sky	AirTemp	Humidity	Wind	Water	Forecast	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

Output:

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

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

PAAVAI ENGINEERING COLLEGE (Autonomous)		
DESCRIPTION	MAX. MARKS	MARKS AWARDED
Preparation & Conduction	10	
Observation & Results	20	
Record Completion	05	
Viva Voce	05	
TOTAL	40	

Result:

Exp No: 3
Date:

ID3 Algorithm

Aim:

To demonstrate the working of the decision tree based ID3 algorithm using an appropriate data set and classifying a new sample.

Algorithm:

Step 1: Create a Root node for the tree
Step 2: If all programs are positive, Return the single-node tree Root, with label = +
Step 3: If all programs are negative, Return the single-node tree Root, with label = -
Step 4: If Attributes is empty, Return the single-node tree Root, with label = most common value of Target_attribute.
Step 5: Otherwise Begin A ← the attribute from Attributes that best* classifies programs
The decision attribute for Root ← A
For each possible value, vi, of A,
Step 6: Add a new tree branch below Root, corresponding to the test A = vi
Let programsvi, be the subset of programs that have value vi for A
Step 7: If programsvi is empty, then below this new branch add a leaf node with label = most common value of Target_attribute in programs.
Else below this new branch add the subtree ID3(programs vi, Target_attribute, Attributes – {A}))
Step 8: End
Step 9: Return Root

Program:

```
import math
import csv
def load_csv(filename):
lines=csv.reader(open(filename,"r"));
dataset = list(lines)
headers = dataset.pop(0)
return dataset,headers
class Node:
def_init_(self,attribute):
self.attribute=attribute
self.children=[]
self.answer=""
def subtables(data,col,delete):
dic={}
coldata=[row[col] for row in data]
attr=list(set(coldata))
counts=[0]*len(attr) r=len(data)
c=len(data[0])
for x in range(len(attr)):
for y in range(r):
if data[y][col]==attr[x]:
counts[x]+=1
for x in range(len(attr)):
dic[attr[x]]=[[0 for i in range(c)] for j in range(counts[x])]
pos=0
for y in range(r):
if data[y][col]==attr[x]:
if delete:
```

```
del data[y][col] dic[attr[x]][pos]=data[y] pos+=1
return attr,dic
def entropy(S):
attr=list(set(S))
if len(attr)==1:
return 0
counts=[0,0]
for i in range(2):
counts[i]=sum([1 for x in S if attr[i]==x])/(len(S)*1.0)
sums=0
for cnt in counts:
sums+=-1*cnt*math.log(cnt,2)
return sums
def compute_gain(data,col):
attr,dic = subtables(data,col,delete=False)
total_size=len(data)
entropies=[0]*len(attr)
ratio=[0]*len(attr)
total_entropy=entropy([row[-1] for row in data])
for x in range(len(attr)):
ratio[x]=len(dic[attr[x]])/(total_size*1.0)
entropies[x]=entropy([row[-1] for row in dic[attr[x]]])
total_entropy-=ratio[x]*entropies[x]
return total_entropy
def build_tree(data,features):
lastcol=[row[-1] for row in data]
if(len(set(lastcol)))==1:
node=Node("")
node.answer=lastcol[0]
return node
n=len(data[0])-1 gains=[0]*n
for col in range(n):
gains[col]=compute_gain(data,col)
split=gains.index(max(gains))
node=Node(features[split])
fea = features[:split]+features[split+1:]
attr,dic=subtables(data,split,delete=True)
for x in range(len(attr)):
child=build_tree(dic[attr[x]],fea)
node.children.append((attr[x],child))
return node
def print_tree(node,level):
if node.answer!="":
print(" "*level,node.answer)
return
print(" "*level,node.attribute)
for value,n in node.children:
print(" "*(level+1),value)
print_tree(n,level+2)
def classify(node,x_test,features):
if node.answer!="":
print(node.answer) return
pos=features.index(node.attribute)
for value, n in node.children:
```

```
if x_test[pos]==value:
classify(n,x_test,features)


'''Main program'''
dataset,features=load_csv("data3.csv")
node1=build_tree(dataset,features)
print("The decision tree for the dataset using ID3 algorithm is")
print_tree(node1,0) testdata,features=load_csv("data3_test.csv")
for xtest in testdata:
print("The test instance:",xtest)
print("The label for test instance:",end="      ")
classify(node1,xtest,features)
```

Input:

Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Input Dataset:

Day	*Outlook*	*Temperature*	*Humidity*	*Wind*
T1	Rain	Cool	Normal	Strong
T2	Sunny	Mild	Normal	Strong

Output:

The decision tree for the dataset using ID3 algorithm is

Outlook rain
Wind
Overcast
Yes
Strong weak
No yes
sunny
Humidity
normal
yes
high
no

The test instance: ['rain', 'cool', 'normal', 'strong']
The label for test instance: no

The test instance: ['sunny', 'mild', 'normal', 'strong']
The label for test instance: yes

PAAVAI ENGINEERING COLLEGE (Autonomous)		
DESCRIPTION	MAX. MARKS	MARKS AWARDED
Preparation & Conduction	10	
Observation & Results	20	
Record Completion	05	
Viva Voce	05	
TOTAL	40	

Result:

Exp No 4: Date:	**Build an Artificial Neural Network using Back Propagation Algorithm**

Aim:

To build an artificial neural network using back propagation algorithm.

Algorithm:

1. Create a feed-forward network with ni inputs, nhidden hidden units, and nout output units.

Initialize all network weights to small random numbers

Until the termination condition is met, do

For each ($\vec{x}$, t), in training examples, do

Propagate the input forward through the network:

Input the instance $\vec{x}$, to the network and compute the output o_u of every unit u in the network.

Propagate the errors backward through the network:

2. For each network output unit k, calculate its error term δ_k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

3. For each hidden unit h, calculate its error term δ_h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{h,k}\delta_k$$

4. Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

Where

$$\Delta w_{ji} = \eta \delta_j x_{i,j}$$

Program:

```
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)
```

Input:

Input	Sleep	Study	Expected % in Exams
1	2	9	92
2	1	5	86
3	3	6	89

Normalize the input

Input	Sleep	Study	Expected % in Exams
1	2/3 = 0.66666667	9/9 = 1	0.92
2	1/3 = 0.33333333	5/9 = 0.55555556	0.86
3	3/3 = 1	6/9 = 0.66666667	0.89

Output:

Input:
[[0.66666667 1.]
[0.33333333 0.55555556]
[1. 0.66666667]]

Actual Output: [[0.92]
[0.86]
[0.89]]

Predicted Output: [[0.89726759]
[0.87196896]
[0.9000671]]

PAAVAI ENGINEERING COLLEGE (Autonomous)		
DESCRIPTION	MAX. MARKS	MARKS AWARDED
Preparation & Conduction	10	
Observation & Results	20	
Record Completion	05	
Viva Voce	05	
TOTAL	40	

Result:

Exp No: 5 Date:	**Naive Bayesian Classifier**

Aim:

To implement the naive bayesian classifier for computing the accuracy of the classifier.

Algorithm:

Step 1: Calculating the posterior probability for a number of different hypotheses h.

Step 2: Calculate the posterior probability of each candidate hypothesis.

Step 3: Calculate the probabilities for input values for each class using a frequency.

Step 4: With real- valued inputs, calculate the mean and standard deviation of input values (x) for each class to summarize the distribution.

Program:

```
import csv
import random import math
def loadcsv(filename):
lines = csv.reader(open(filename, "r")); dataset = list(lines)
for i in range(len(dataset)):
 #converting strings into numbers for processing
dataset[i] = [float(x) for x in dataset[i]]
return dataset

def splitdataset(dataset, splitratio):
 #67% training size trainsize = int(len(dataset) * splitratio);
trainset = []
copy = list(dataset);
while len(trainset) < trainsize:
 #generate indices for the dataset list randomly to pick ele fortraining data
index = random.randrange(len(copy)); trainset.append(copy.pop(index))
return [trainset, copy]

def separatebyclass(dataset):
separated = {} #dictionary of classes 1 and 0
 #creates a dictionary of classes 1 and 0 where the values are#the instances belonging to each class
for i in range(len(dataset)): vector = dataset[i]
if (vector[-1] not in separated): separated[vector[-1]] = []
separated[vector[-1]].append(vector) return separated

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

def stdev(numbers):
avg = mean(numbers)
variance = sum([pow(x-avg,2) for x in numbers])/float(len(numbers)-1)
return math.sqrt(variance)
def summarize(dataset): #creates a dictionary of classes summaries =[(mean(attribute),stdev(attribute))
for attribute in zip(*dataset)];
del summaries[-1] #excluding labels +ve or -ve return summaries

def summarizebyclass(dataset):
separated = separatebyclass(dataset); #print(separated)
summaries = {}
```

```
for classvalue, instances in separated.items():
 #for key,value in dic.items()
 #summaries is a dic of tuples(mean,std) for each class value
summaries[classvalue] = summarize(instances)
#summarize is used to cal to mean and std
return summaries

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

def calculateclassprobabilities(summaries, inputvector):
# probabilities contains the all prob of all class of test data probabilities = {}
for classvalue, classsummaries in summaries.items(): #class and attribute information as mean and sd
probabilities[classvalue] = 1
for i in range(len(classsummaries)):
mean, stdev = classsummaries[i] #take mean and sd of every attribute for class 0 and 1 seperaely
x = inputvector[i] #testvector's first attribute probabilities[classvalue] *=
calculateprobability(x, mean, stdev);#use normal dist return probabilities

def predict(summaries, inputvector): #training and test data is passed
probabilities = calculateclassprobabilities(summaries, inputvector)
bestLabel, bestProb = None, -1
for classvalue, probability in probabilities.items(): #assigns that class which has the highest prob
if bestLabel is None or probability > bestProb: bestProb = probability
bestLabel = classvalue return bestLabel
def getpredictions(summaries, testset): predictions = []
for i in range(len(testset)):
result = predict(summaries, testset[i]) predictions.append(result)
return predictions

def getaccuracy(testset, predictions): correct = 0
for i in range(len(testset)):
if testset[i][-1] == predictions[i]: correct += 1
return (correct/float(len(testset))) * 100.0

def main():
filename = 'naivedata.csv' splitratio = 0.67
dataset = loadcsv(filename);

trainingset, testset = splitdataset(dataset, splitratio) print('Split {0} rows into train={1} and test={2}
rows'.format(len(dataset), len(trainingset), len(testset))) # prepare model
summaries = summarizebyclass(trainingset); #print(summaries)
# test model
predictions = getpredictions(summaries, testset) #find the predictions of test data with the training data
accuracy = getaccuracy(testset, predictions) print('Accuracy of the classifier is :
{0}%'.format(accuracy)) main()
```

Input:

Input	Pregnancies	Glucose	Blood Pressure	Skin Thickness	Insulin	BMI	Diabetic Pedigree Function	Age	Outcome
1	6	148	72	35	0	33.6	0.627	50	1
2	1	85	66	29	0	26.6	0.351	31	0
3	8	183	64	0	0	23.3	0.672	32	1
4	1	89	66	23	94	28.1	0.167	21	0
5	0	137	40	35	168	43.1	2.288	33	1
6	5	116	74	0	0	25.6	0.201	30	0
7	3	78	50	32	88	31	0.248	26	1
8	10	115	0	0	0	35.3	0.134	29	0
9	2	197	70	45	543	30.5	0.158	53	1
10	8	125	96	0	0	0	0.232	54	1

Output:

Split 768 rows into train=514 and test=254 rows Accuracy of the classifier is : 71.65354330708661%

PAAVAI ENGINEERING COLLEGE (Autonomous)		
DESCRIPTION	MAX. MARKS	MARKS AWARDED
Preparation & Conduction	10	
Observation & Results	20	
Record Completion	05	
Viva Voce	05	
TOTAL	40	

Result:

Exp No: 6 Date:	**Naive Bayesian Classifier Model**

Aim:

To calculate the accuracy, precision, and recall the data set using Naive Bayesian Classifier Model.

Algorithm:

Step 1: Collect all words, punctuation, and other tokens that occurs.

Step 2: Calculate the required P(vj) and P(wk|vj) probability terms

For each target value vj in V do

P(wk|vj) ← (nk + 1) / (n + | Vocabulary|)

Step 3: classify_naive_bayes_text (doc)

Step 4: Return the estimated target value for the document Doc. ai denotes the word found in the ith position within Doc.

Step 5: Return VNB

Program:

```
import pandas as pd
msg=pd.read_csv('naivetext.csv',names=['message','label'])
print('The dimensions of the dataset',msg.shape)
msg['labelnum']=msg.label.map({'pos':1,'neg':0})
X=msg.message y=msg.labelnum
print(X)
print(y)

#splitting the dataset into train and test data from sklearn.model_selection
import train_test_split xtrain,xtest,ytrain,ytest=train_test_split(X,y)
print ('\n The total number of Training Data :',ytrain.shape)
print ('\n The total number of Test Data :',ytest.shape)

#Output of count vectoriser is a sparse matrix from sklearn.feature_extraction.text
import CountVectorizer
count_vect = CountVectorizer()
xtrain_dtm = count_vect.fit_transform(xtrain)
xtest_dtm=count_vect.transform(xtest)
print('\n The words or Tokens in the text documents \n')
print(count_vect.get_feature_names())
df=pd.DataFrame(xtrain_dtm.toarray(),columns=count_vect.get_fe ature_names())

# Training Naive Bayes (NB) classifier on training data from sklearn.naive_bayes
import MultinomialNB
clf = MultinomialNB().fit(xtrain_dtm,ytrain)
predicted = clf.predict(xtest_dtm)

#printing accuracy, Confusion matrix, Precision and Recall  from sklearn import metrics
print('\n Accuracy of the classifer is',
metrics.accuracy_score(ytest,predicted))

print('\n Confusion matrix')
print(metrics.confusion_matrix(ytest,predicted))
print('\n The value of Precision' , metrics.precision_score(ytest,predicted))
print('\n The value of Recall' , metrics.recall_score(ytest,predicted))
```

Input:

	Text Documents	Label
1	I love this sandwich	Pos
2	This is an amazing place	Pos
3	I feel very good about these beers	Pos
4	This is my best work	Pos
5	What an awesome view	Pos
6	I do not like this restaurant	Neg
7	I am tired of this stuff	Neg
8	I can't deal with this	Neg
9	He is my sworn enemy	Neg
10	My boss is horrible	Neg
11	This is an awesome place	Pos
12	I do not like the taste of this juice	Neg
13	I love to dance	Pos
14	I am sick and tired of this place	Neg
15	What a great holiday	Pos
16	That is a bad locality to stay	Neg
17	We will have good fun tomorrow	Pos
18	I went to my enemy's house today	Neg

Output:

The dimensions of the dataset (18, 2)
I love this sandwich
This is an amazing place
I feel very good about these beers
This is my best work
What an awesome view
I do not like this restaurant
I am tired of this stuff
I can't deal with this
He is my sworn enemy
My boss is horrible
This is an awesome place
I do not like the taste of this juice
I love to dance
I am sick and tired of this place
What a great holiday
That is a bad locality to stay
We will have good fun tomorrow
I went to my enemy's house today

Name: message, dtype: object 0 1
1 1

2 1
3 1
4 1
5 0
6 0
7 0
8 0
9 0
10 1
11 0
12 1
13 0
14 1
15 0
16 1
17 0

Name: labelnum, dtype: int64

The total number of Training Data: (13,)
The total number of Test Data: (5,)

The words or Tokens in the text documents
['about', 'am', 'amazing', 'an', 'and', 'awesome', 'beers', 'best', 'can', 'deal', 'do', 'enemy', 'feel', 'fun', 'good', 'great', 'have', 'he', 'holiday', 'house', 'is', 'like', 'love', 'my', 'not', 'of', 'place', 'restaurant', 'sandwich', 'sick', 'sworn', 'these', 'this', 'tired', 'to', 'today', 'tomorrow', 'very', 'view', 'we', 'went', 'what', 'will', 'with', 'work']

Accuracy of the classifier is 0.8 Confusion matrix
[[2 1]
[0 2]]
The value of Precision 0.6666666666666666 The value of Recall 1.0

PAAVAI ENGINEERING COLLEGE (Autonomous)		
DESCRIPTION	MAX. MARKS	MARKS AWARDED
Preparation & Conduction	10	
Observation & Results	20	
Record Completion	05	
Viva Voce	05	
TOTAL	40	

Result:

Exp No: 7 Date:	**Bayesian Network Considering Medical Data**

Aim:

To construct a bayesian network for diagnosing heart patients using standard heart disease data set.

Algorithm:

Step 1: Collect the data set.

Step 2: Split the data into observed and unobserved causes

Step 3: Calculate the posterior conditional probability distribution of each of the possible unobserved causes given the observed evidence, i.e. P [Cause | Evidence].

Step 4: Compute the necessary operations in heart disease data set.

Step 5: Display the details in Bayesian network.

Program:

```
import numpy as np
import pandas as pd
import csv from pgmpy.estimators
import MaximumLikelihood Estimator from pgmpy.models
import BayesianModel from pgmpy.inference
import  VariableElimination

#read Cleveland Heart Disease
data heartDisease = pd.read_csv('heart.csv')
heartDisease = heartDisease.replace('?',np.nan)

#display the data
print('Sample instances from the dataset are given below')
print(heartDisease.head())

#display the Attributes names and datatypes
print('\n Attributes and datatypes')
print(heartDisease.dtypes)

#Create Model- Bayesian Network model = BayesianModel([('age','heartdisease'),
('sex','heartdisease'), ('exang','heartdisease'),('cp','heartdisease'),('heartdisease', 'restecg'),('heartdisease','chol')])

#Learning CPDs using Maximum Likelihood Estimators
print('\n Learning CPD using Maximum likelihood estimators')
model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)

#Inferencing with Bayesian Network
print('\n Inferencing with Bayesian Network:')
HeartDiseasetest_infer = VariableElimination(model)
```

```
#computing the Probability of HeartDisease given restecg
 print('\n 1.Probability of HeartDisease given evidence= restecg :1')
  q1=HeartDiseasetest_infer.query(variables=['heartdisease'],evi dence={'restecg':1})
  print(q1)

#computing the Probability of HeartDisease given cp
 print('\n 2.Probability of HeartDisease given evidence= cp:2 ')
 q2=HeartDiseasetest_infer.query(variables=['heartdisease'],evi dence={'cp':2})
print(q2)
```

Input Data Set:

Heart Disease Databases

The Cleveland database contains 76 attributes, but all published experiments refer to using a subset of 14 of them. In particular, the Cleveland database is the only one that has been used by ML researchers to this date. The "Heartdisease" field refers to the presence of heart disease in the patient. It is integer valued from 0 to 4.

Database:	0	1	2	3	4	Total
Cleveland:	164	55	36	35	13	303

Attribute Information:

age: age in years

sex: sex (1 = male; 0 = female)

cp: chest pain type

Value 1: typical angina

Value 2: atypical angina

Value 3: non-anginal pain

Value 4: asymptomatic

trestbps: resting blood pressure (in mm Hg on admission to the hospital)

chol: serum cholestoral in mg/dl

fbs: (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)

restecg: resting electrocardiographic results

Value 0: normal

Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV)

Value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria

thalach: maximum heart rate achieved

exang: exercise induced angina (1 = yes; 0 = no)

oldpeak = ST depression induced by exercise relative to rest

slope: the slope of the peak exercise ST segment

Value 1: upsloping

Value 2: flat

Value 3: downsloping

thal: 3 = normal; 6 = fixed defect; 7 = reversable defect

Heartdisease: It is integer valued from 0 (no presence) to 4.

Some instance from the dataset:

Age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	Heart disease
63	1	1	145	233	1	2	150	0	2.3	3	0	6	0
67	1	4	160	286	0	2	108	1	1.5	2	3	3	2
67	1	4	120	229	0	2	129	1	2.6	2	2	7	1
41	0	2	130	204	0	2	172	0	1.4	1	0	3	0
62	0	4	140	268	0	2	160	0	3.6	3	2	3	3
60	1	4	130	206	0	2	132	1	2.4	2	2	7	4

Output:

	age	sex	cp	trest bps	chol	fbs	rest ecg	thalac h	exan g	old peak	slo pe	ca	thal	heart disea se
0	63	1	1	145	233	1	2	150	0	2.3	3	0	6	0
1	67	1	4	160	286	0	2	108	1	1.5	2	3	3	2
2	67	1	4	120	229	0	2	129	1	2.6	2	2	7	1
3	37	1	3	130	250	0	2	135	1	3.5	3	0	3	0
4	41	0	2	130	204	0	2	172	0	1.4	1	0	3	0

[5 rows * 14 columns]

Attributes and datatypes

Age	int 64
Sex	int 64
Cp	int 64
trestbps	int 64
Chol	int 64
Fbs	int 64
restecg	int 64
thalach	int 64
Exang	int 64
oldpeak	float64
Slope	int 64
Ca	Object
Thal	Object
heartdisease	int 64

dtype:object

Learning CPD using Maximum likelihood estimators

Inferencing with Bayesian Network:

1. Probability of HeartDisease given evidence=restecg

heartdisease	PHI(heartdisease)
heartdisease(0)	0.1012
heartdisease(1)	0.0000
heartdisease(2)	0.2392
heartdisease(3)	0.2015
heartdisease(4)	0.4581

2. Probability of HeartDisease given evidence = cp

heartdisease	PHI(heartdisease)
heartdisease(0)	0.3610
heartdisease(1)	0.2159
heartdisease(2)	0.1373
heartdisease(3)	0.1537
heartdisease(4)	0.1321

PAAVAI ENGINEERING COLLEGE (Autonomous)		
DESCRIPTION	MAX. MARKS	MARKS AWARDED
Preparation & Conduction	10	
Observation & Results	20	
Record Completion	05	
Viva Voce	05	
TOTAL	40	

Result:

Exp No: 8
Date:

CLUSTERING A SET OF DATA USING EM ALGORITHM

Aim:

To cluster a set of data using EM algorithm and k-means algorithm inorder to compare the quality of clustering.

Algorithm:

EM Algorithm:

Step 1: Computes the latent variables i.e. expectation of the log-likelihood using the current parameter estimates.

Step 2: Determines the parameters that maximize the expected log-likelihood obtained in the E step, and corresponding model parameters are updated based on the estimated latent variables.

k-mean Algorithm:

Step 3: Determines the best value for K center points or centroids by an iterative process.

Step 4: Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

Program:

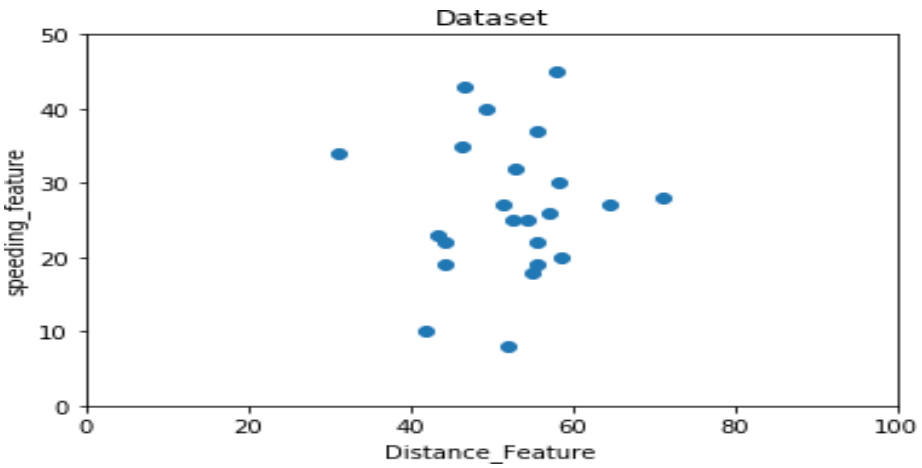
```
from sklearn.cluster import KMeans
#from sklearn import metrics
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
data=pd.read_csv("kmeansdata.csv")
df1=pd.DataFrame(data)
print(df1
f1 = df1['Distance_Feature'].values
f2 = df1['Speeding_Feature'].values
X=np.matrix(list(zip(f1,f2)))
plt.plot()
plt.xlim([0, 100])
plt.ylim([0, 50])
plt.title('Dataset')
plt.ylabel('speeding_feature')
plt.xlabel('Distance_Feature')
plt.scatter(f1,f2)
plt.show()
#create new plot and dataplt.plot()
colors = ['b', 'g', 'r']
markers = ['o', 'v', 's']
#KMeans algorithm
#K = 3
kmeans_model = KMeans(n_clusters=3).fit(X)
plt.plot()
for i, l in enumerate(kmeans_model.labels):
plt.plot(f1[i], f2[i], color=colors[l], marker=markers[l],ls='None')
plt.xlim([0, 100])
plt.ylim([0, 50])
plt.show()
```

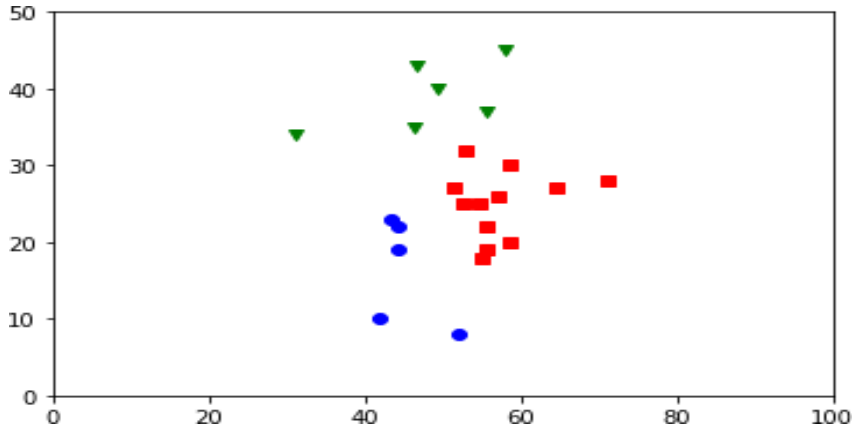
Input:

```
Driver_ID,Distance_Feature,Speeding_Feature
3423311935,71.24,28
3423313212,52.53,25
3423313724,64.54,27
3423311373,55.69,22
3423310999,54.58,25
3423313857,41.91,10
3423312432,58.64,20
3423311434,52.02,8
3423311328,31.25,34
3423312488,44.31,19
3423311254,49.35,40
3423312943,58.07,45
3423312536,44.22,22
3423311542,55.73,19
3423312176,46.63,43
3423314176,52.97,32
3423314202,46.25,35
3423311346,51.55,27
3423310666,57.05,26
3423313527,58.45,30
3423312182,43.42,23
3423313590,55.68,37
3423312268,55.15,18
```

Output:

Driver_ID	Distance_Feature	Speeding_Feature
3423311935,71.24,28	71.24	28
3423313212,52.53,25	52.53	25
3423313724,64.54,27	64.54	27
3423311373,55.69,22	55.69	22
3423310999,54.58,25	54.58	25
3423313857,41.91,10	41.91	10
3423312432,58.64,20	58.64	20
3423311434,52.02,8	52.02	8
3423311328,31.25,34	31.25	34
3423312488,44.31,19	44.31	19
3423311254,49.35,40	49.35	40
3423312943,58.07,45	58.07	45
3423312536,44.22,22	44.22	22
3423311542,55.73,19	55.73	19
3423312176,46.63,43	46.63	43
3423314176,52.97,32	52.97	32
3423314202,46.25,35	46.25	35
3423311346,51.55,27	51.55	27
3423310666,57.05,26	57.05	26
3423313527,58.45,30	58.45	30
3423312182,43.42,23	43.42	23
3423313590,55.68,37	55.68	37
3423312268,55.15,18	55.15	18





PAAVAI ENGINEERING COLLEGE (Autonomous)		
DESCRIPTION	MAX. MARKS	MARKS AWARDED
Preparation & Conduction	10	
Observation & Results	20	
Record Completion	05	
Viva Voce	05	
TOTAL	40	

Result:

Exp No: 9 Date:	**Implementing k-Nearest Neighbour Algorithm**

Aim:

To implement the k-Nearest Neighbour algorithm for classifying the iris data set.

Algorithm:

Steps: Given a query instance xq to be classified,

Let x1... xk denote the k instances from training examples that are nearest to xq

Return

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{k}$$

Program:

```
from sklearn.model_selection
import train_test_split from sklearn.neighbors
import KNeighborsClassifier from sklearn.metrics
import classification_report, confusion_matrix from sklearn import datasets
  """ Iris Plants Dataset, dataset contains 150 (50 in each of three classes)Number of Attributes: 4
  numeric, predictive attributes and the Class"""
iris=datasets.load_iris()
  """ The x variable contains the first four columns of the dataset(i.e. attributes) while y contains the
  labels."""
x = iris.data y = iris.target
print ('sepal-length', 'sepal-width', 'petal-length', 'petal-width')
print(x)
print('class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica') \
print(y)
  """ Splits the dataset into 70% train data and 30% test data. Thismeans that out of total 150
  records, the training set will contain 105 records and the test set contains 45 of those records"""
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3)
#To Training the model and Nearest nighbors K=5
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(x_train, y_train)
#to make predictions on our test data  y_pred=classifier.predict(x_test)
  """ For evaluating an algorithm, confusion matrix, precision, recalland f1 score are the most
  commonly used metrics."""
print('Confusion Matrix')
print(confusion_matrix(y_test,y_pred))
print('Accuracy Metrics')
print(classification_report(y_test,y_pred))
```

Input:

Data Set:

Iris Plants Dataset: Dataset contains 150 instances (50 in each of three classes) Number of Attributes: 4 numeric, predictive attributes and the Class

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

Output:

sepal-length sepal-width petal-length petal-width

```
 [[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]
.          .          .          .          .
.          .          .          .          .

 [6.2          3.4          5.4          2.3]
 [5.9          3.           5.1          1.8]]
```

class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica

[0 0 0 ……0 0 1 1 1 ……1 1 2 2 2 …… 2 2]

Confusion Matrix

```
[[20  0  0]
 [ 0 10  0]
 [ 0  1 14]]
```

Accuracy Metrics

	Precision	recall	f1-score	support
0	1.00	1.00	1.00	20
1	0.91	1.00	0.95	10
2	1.00	0.93	0.97	15
avg / total	0.98	0.98	0.98	45

PAAVAI ENGINEERING COLLEGE (Autonomous)		
DESCRIPTION	MAX. MARKS	MARKS AWARDED
Preparation & Conduction	10	
Observation & Results	20	
Record Completion	05	
Viva Voce	05	
TOTAL	40	

Result:

Exp No: 10 Date:	**Non-Parametric Locally Weighted Algorithm**

Aim:

To implement the non-parametric locally weighted regression algorithm in order to fit data points.

Algorithm:

Step 1: Read the Given data Sample to X and the curve (linear or non linear) to Y
Step 2: Set the value for Smoothening parameter or Free parameter say τ
Step 3: Set the bias /Point of interest set x0 which is a subset of X
Step 4: Determine the weight matrix using :

$$w(x, x_0) = e^{-\frac{(x-x_0)^2}{2\tau^2}}$$

Step 5: Determine the value of model term parameter β using :

$$\hat{\beta}(x_0) = (X^T W X)^{-1} X^T W y$$

Step 6: Prediction = x0*β:

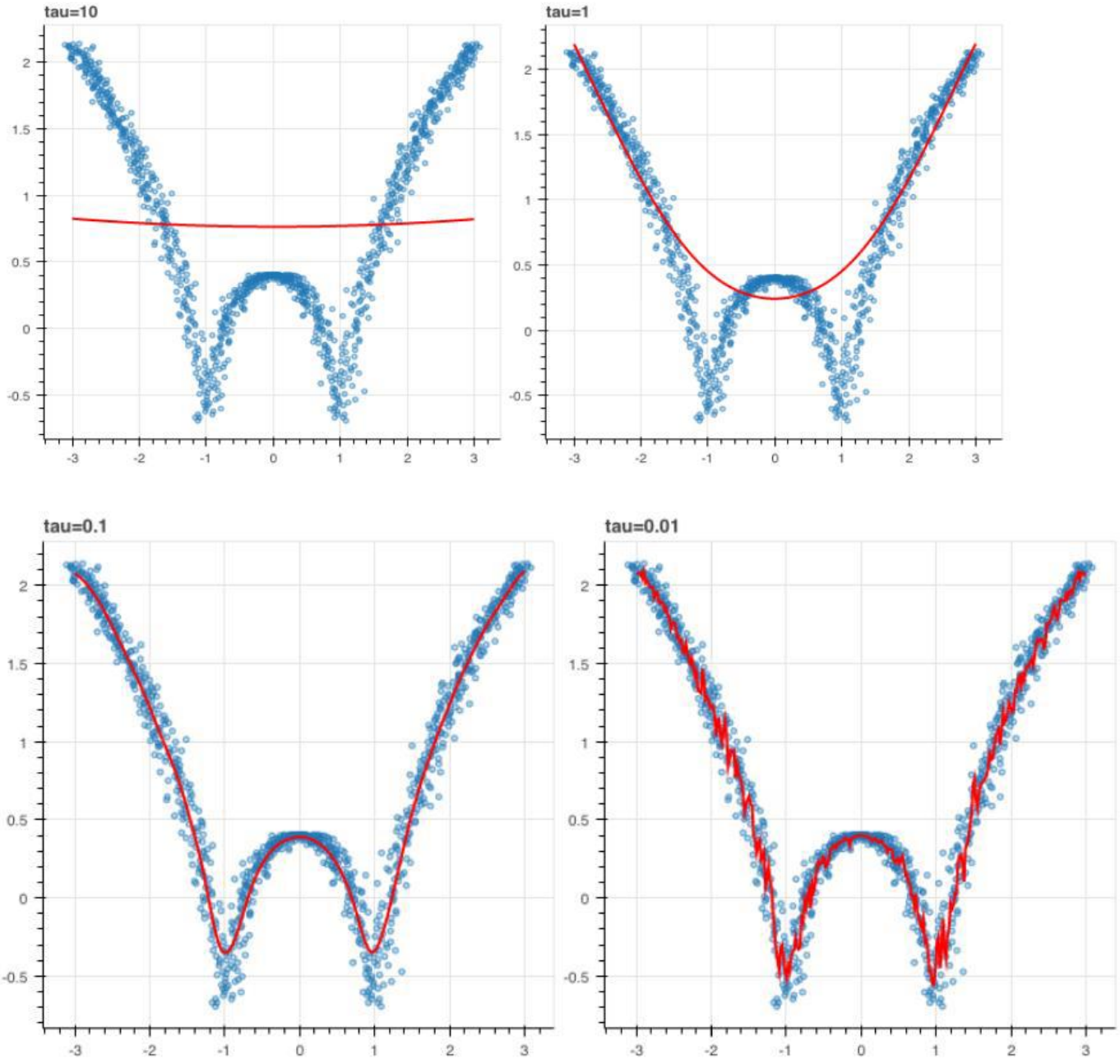
Program:

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

def local_regression(x0, X, Y, tau):
# add bias term x0 = np.r_[1, x0]
# Add one to avoid the loss in information
X = np.c_[np.ones(len(X)), X]

# fit model: normal equations with kernel
xw = X.T * radial_kernel(x0, X, tau)
# XTranspose * W
beta = np.linalg.pinv(xw @ X) @ xw @ Y
#@ Matrix Multiplication or Dot Product
# predict value
return x0 @ beta # @ Matrix Multiplication or Dot Product for prediction
def radial_kernel(x0, X, tau):
return np.exp(np.sum((X - x0) ** 2, axis=1) / (-2 * tau * tau))
# Weight or Radial Kernal Bias Function
n = 1000
# generate dataset
X = np.linspace(-3, 3, num=n)
print("The Data Set ( 10 Samples) X :\n",X[1:10])
Y = np.log(np.abs(X ** 2 - 1) + .5)
print("The Fitting Curve Data Set (10 Samples) Y
:\n",Y[1:10])
# jitter X
X += np.random.normal(scale=.1, size=n)
print("Normalised (10 Samples) X :\n",X[1:10])
domain = np.linspace(-3, 3, num=300)
print(" Xo Domain Space(10 Samples) :\n",domain[1:10]) def plot_lwr(tau):
# prediction through regression
```

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

show(gridplot([[plot_lwr(10.), plot_lwr(1.)],
[plot_lwr(0.1), plot_lwr(0.01)]]))


# -*- coding: utf-8 -*- """
Spyder Editor
This is a temporary script file. """
from numpy import * from os
import listdir
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np1
import numpy.linalg as np
from scipy.stats.stats
import pearsonr

def kernel(point,xmat, k):
m,n = np1.shape(xmat)
weights = np1.mat(np1.eye((m)))
for j in range(m):
diff = point - X[j]
weights[j,j] = np1.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 = np1.shape(xmat)

ypred = np1.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 = np1.array(data.total_bill)
tip = np1.array(data.tip)

#preparing and add 1 in bill mbill = np1.mat(bill)
mtip = np1.mat(tip)
# mat is used to convert to n dimesiona to 2 dimensional array form m= np1.shape(mbill)[1]
# print(m) 244 data is stored in m one = np1.mat(np1.ones(m))
```

```
X= np1.hstack((one.T,mbill.T)) # create a stack of bill from ONE #print(X)
#set k here
ypred = localWeightRegression(X,mtip,0.3) 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=5)
plt.xlabel('Total bill')
plt.ylabel('Tip')
plt.show();
```

Output:



PAAVAI ENGINEERING COLLEGE (Autonomous)		
DESCRIPTION	MAX. MARKS	MARKS AWARDED
Preparation & Conduction	10	
Observation & Results	20	
Record Completion	05	
Viva Voce	05	
TOTAL	40	

Result: