**#PROGRAM1 : Implement A\* Search algorithm.**

from collections import deque

class Graph:

def \_\_init\_\_(self, adjac\_lis):

self.adjac\_lis = adjac\_lis

def get\_neighbors(self, v):

return self.adjac\_lis[v]

# This is heuristic function which is having equal values for all nodes

def h(self, n):

H = {

'A': 11,

'B': 6,

'C': 99,

'D': 1,

'E': 7,

'G': 0,

}

return H[n]

def a\_star\_algorithm(self, start, stop):

# In this open\_lst is a lisy of nodes which have been visited, but who's

# neighbours haven't all been always inspected, It starts off with the start

#node

# And closed\_lst is a list of nodes which have been visited

# and who's neighbors have been always inspected

open\_lst = set([start])

closed\_lst = set([])

# poo has present distances from start to all other nodes

# the default value is +infinity

poo = {}

poo[start] = 0

# par contains an adjac mapping of all nodes

par = {}

par[start] = start

while len(open\_lst) > 0:

n = None

# it will find a node with the lowest value of f() -

for v in open\_lst:

if n == None or poo[v] + self.h(v) < poo[n] + self.h(n):

n = v;

if n == None:

print('Path does not exist!')

return None

# if the current node is the stop

# then we start again from start

if n == stop:

reconst\_path = []

while par[n] != n:

reconst\_path.append(n)

n = par[n]

reconst\_path.append(start)

reconst\_path.reverse()

print('Path found: {}'.format(reconst\_path))

return reconst\_path

# for all the neighbors of the current node do

for (m, weight) in self.get\_neighbors(n):

# if the current node is not presentin both open\_lst and closed\_lst

# add it to open\_lst and note n as it's par

if m not in open\_lst and m not in closed\_lst:

open\_lst.add(m)

par[m] = n

poo[m] = poo[n] + weight

# otherwise, check if it's quicker to first visit n, then m

# and if it is, update par data and poo data

# and if the node was in the closed\_lst, move it to open\_lst

else:

if poo[m] > poo[n] + weight:

poo[m] = poo[n] + weight

par[m] = n

if m in closed\_lst:

closed\_lst.remove(m)

open\_lst.add(m)

# remove n from the open\_lst, and add it to closed\_lst

# because all of his neighbors were inspected

open\_lst.remove(n)

closed\_lst.add(n)

print('Path does not exist!')

return None

adjac\_lis = {

'A': [('B', 2), ('E', 3)],

'B': [('C', 1),('G', 9)],

'C': None,

'E': [('D', 6)],

'D': [('G', 1)],

}

graph1 = Graph(adjac\_lis)

graph1.a\_star\_algorithm('A', 'G')

OUTPUT:

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

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

******

**Program2: Implement AO\* Search algorithm.**

from collections import deque

class Graph:

def \_\_init\_\_(self, adjac\_lis):

self.adjac\_lis = adjac\_lis

def get\_neighbors(self, v):

return self.adjac\_lis[v]

# This is heuristic function which is having equal values for all nodes

def h(self, n):

H = {

'S': 14,

'A': 7,

'B': 12,

'C': 13,

'D': 5,

'E': 6,

'G': 7,

'F': 5,

'H': 2

}

return H[n]

def a\_star\_algorithm(self, start, stop):

# In this open\_lst is a lisy of nodes which have been visited, but who's

# neighbours haven't all been always inspected, It starts off with the start

#node

# And closed\_lst is a list of nodes which have been visited

# and who's neighbors have been always inspected

open\_lst = set([start])

closed\_lst = set([])

# poo has present distances from start to all other nodes

# the default value is +infinity

poo = {}

poo[start] = 0

# par contains an adjac mapping of all nodes

par = {}

par[start] = start

while len(open\_lst) > 0:

n = None

# it will find a node with the lowest value of f() -

for v in open\_lst:

if n == None or poo[v] + self.h(v) < poo[n] + self.h(n):

n = v;

if n == None:

print('Path does not exist!')

return None

# if the current node is the stop

# then we start again from start

if n == stop:

reconst\_path = []

while par[n] != n:

reconst\_path.append(n)

n = par[n]

reconst\_path.append(start)

reconst\_path.reverse()

print('Path found: {}'.format(reconst\_path))

return reconst\_path

# for all the neighbors of the current node do

for (m, weight) in self.get\_neighbors(n):

# if the current node is not presentin both open\_lst and closed\_lst

# add it to open\_lst and note n as it's par

if m not in open\_lst and m not in closed\_lst:

open\_lst.add(m)

par[m] = n

poo[m] = poo[n] + weight

# otherwise, check if it's quicker to first visit n, then m

# and if it is, update par data and poo data

# and if the node was in the closed\_lst, move it to open\_lst

else:

if poo[m] > poo[n] + weight:

poo[m] = poo[n] + weight

par[m] = n

if m in closed\_lst:

closed\_lst.remove(m)

open\_lst.add(m)

# remove n from the open\_lst, and add it to closed\_lst

# because all of his neighbors were inspected

open\_lst.remove(n)

closed\_lst.add(n)

print('Path does not exist!')

return None

adjac\_lis = {

'S': [('A', 1), ('B', 1),('C', 1)],

'A': [('D', 1),('E', 1)],

'C': [('F', 1), ('G', 1)],

'D': [('H', 1)],

}

graph1 = Graph(adjac\_lis)

graph1.a\_star\_algorithm('S', 'H')

OUTPUT:

Path found: ['S', 'A', 'D', 'H']

['S', 'A', 'D', 'H']

**AO\* Algorithm**

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.

The decomposition of the problem or problem reduction generates AND arcs.

**AND-OR Graph**

**The figure shows an AND-OR graph**

1. To pass any exam, we have two options, either cheating or hard work.

2. In this graph we are given two choices, first do cheating **or (The red line)** work hard and **(The arc)** pass.

3. When we have more than one choice and we have to pick one, we apply **OR condition** to

choose one.(That's what we did here).

Basically the **ARC** here denote **AND condition**.

Here we have replicated the arc between the work hard and the pass because by doing

the hard work possibility of passing an exam is more than cheating.

**A\* Vs AO\***

1. Both are part of informed search technique and use heuristic values to solve the problem.

2. The solution is guaranteed in both algorithm.

3. A\* **always** gives an **optimal solution** (shortest path with low cost) But It is not guaranteed to that **AO\*** always provide **an optimal solutions**.

4. **Reason:** Because AO\* does not explore all the solution path once it got solution

**Program3: #CANDIDATE ELIMINATION ALGORITHM PROGRAM3**

#Importing Important Libraries

import numpy as np

import pandas as pd

data = pd.DataFrame(data=pd.read\_csv('D:\\Jyoti W\\2020-21 ML Program\\enjoysport.csv'))

print(data)

concepts = np.array(data.iloc[:,0:-1])

target = np.array(data.iloc[:,-1]) +

print(target)

print(concepts)

#Defining Model (Candidate Elimination algorithm concepts)

def learn(concepts, target):

specific\_h = concepts[0].copy()

print("Initialization of specific\_h and general\_h")

print("specific\_h: ",specific\_h)

general\_h = [["?" for i in range(len(specific\_h))] for i in range(len(specific\_h))]

print("general\_h: ",general\_h)

print("concepts: ",concepts)

for i, h in enumerate(concepts):

if target[i] == "yes":

for x in range(len(specific\_h)):

#print("h[x]",h[x])

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("\nSteps of Candidate Elimination Algorithm: ",i+1)

print("Specific\_h: ",i+1)

print(specific\_h,"\n")

print("general\_h :", i+1)

print(general\_h)

indices = [i for i, val in enumerate(general\_h) if val == ['?', '?', '?', '?', '?', '?']]

print("\nIndices",indices)

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)

print("Final General\_h:",g\_final)

**OUTPUT:**

sky airtemp humidity wind water forcast enjoysport

0 sunny warm normal strong warm same yes

1 sunny warm high strong warm same yes

2 rainy cold high strong warm change no

3 sunny warm high strong cool change yes

['yes' 'yes' 'no' 'yes']

[['sunny' 'warm' 'normal' 'strong' 'warm' 'same']

['sunny' 'warm' 'high' 'strong' 'warm' 'same']

['rainy' 'cold' 'high' 'strong' 'warm' 'change']

['sunny' 'warm' 'high' 'strong' 'cool' 'change']]

Initialization of specific\_h and general\_h

specific\_h: ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']

general\_h: [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

concepts: [['sunny' 'warm' 'normal' 'strong' 'warm' 'same']

['sunny' 'warm' 'high' 'strong' 'warm' 'same']

['rainy' 'cold' 'high' 'strong' 'warm' 'change']

['sunny' 'warm' 'high' 'strong' 'cool' 'change']]

Steps of Candidate Elimination Algorithm: 4

Specific\_h: 4

['sunny' 'warm' '?' 'strong' '?' '?']

general\_h : 4

[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Indices [2, 3, 4, 5]

Final Specific\_h: ['sunny' 'warm' '?' 'strong' '?' '?']

Final General\_h: [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]

**Date set:**

**enjoysport.csv**

Sky,AirTemp,Humidity,Wind,Water,Forecast,EnjoySport

Sunny,Warm,Normal,Strong,Warm,Same,1

Sunny,Warm,High,Strong,Warm,Same,1

Rainy,Cold,High,Strong,Warm,Change,0

Sunny,Warm,High,Strong,Cool,Change,1

**Program4: #ID3 ALGORITHM PROGRAM4**

import pandas as pd

from sklearn import tree

from sklearn.preprocessing import LabelEncoder

from sklearn.tree import DecisionTreeClassifier

from sklearn.externals.six import StringIO

data = pd.read\_csv('C:\\Users\\LAB\\Desktop\\SKSVMACET LAB MANUAL1\\data(csv files)\\tennis.csv')

print("The first 5 values of data is \n",data.head())

X = data.iloc[:,:-1]

print("\nThe first 5 values of Train data is \n",X.head())

y = data.iloc[:,-1]

print("\nThe first 5 values of Train output is \n",y.head())

le\_outlook = LabelEncoder()

X.Outlook = le\_outlook.fit\_transform(X.Outlook)

le\_Temperature = LabelEncoder()

X.Temperature = le\_Temperature.fit\_transform(X.Temperature)

le\_Humidity = LabelEncoder()

X.Humidity = le\_Humidity.fit\_transform(X.Humidity)

le\_Windy = LabelEncoder()

X.Windy = le\_Windy.fit\_transform(X.Windy)

print("\nNow the Train data is",X.head())

le\_PlayTennis = LabelEncoder()

y = le\_PlayTennis.fit\_transform(y)

print("\nNow the Train data is\n",y)

classifier = DecisionTreeClassifier()

classifier.fit(X,y)

def labelEncoderForInput(list1):

list1[0] = le\_outlook.transform([list1[0]])[0]

list1[1] = le\_Temperature.transform([list1[1]])[0]

list1[2] = le\_Humidity.transform([list1[2]])[0]

list1[3] = le\_Windy.transform([list1[3]])[0]

return [list1]

inp = ["Rainy","Mild","High","False"]

inp1=["Rainy","Cool","High","False"]

pred1 = labelEncoderForInput(inp1)

y\_pred = classifier.predict(pred1)

print("\nfor input {0}, we obtain {1}".format(inp1, le\_PlayTennis.inverse\_transform(y\_pred[0])))

**OUTPUT:**

Outlook

overcast

b'yes'

rain

Wind

b'strong'

b'no'

b'weak'

b'yes'

sunny

Humidity

b'high'

b'no'

b'normal'

b'yes'

{'Outlook': {'overcast': 'yes', 'rain': {'Wind': {'weak': 'yes', 'strong': 'no'}}, 'sunny': {'Humidity': {'high':

'no', 'normal': 'yes'}}}}

Date Set:

**#tennisdata.csv**

Outlook,Temperature,Humidity,Windy,PlayTennis

Sunny,Hot,High,FALSE,No

Sunny,Hot,High,TRUE,No

Overcast,Hot,High,FALSE,Yes

Rainy,Mild,High,FALSE,Yes

Rainy,Cool,Normal,FALSE,Yes

Rainy,Cool,Normal,TRUE,No

Overcast,Cool,Normal,TRUE,Yes

Sunny,Mild,High,FALSE,No

Sunny,Cool,Normal,FALSE,Yes

Rainy,Mild,Normal,FALSE,Yes

Sunny,Mild,Normal,TRUE,Yes

Overcast,Mild,High,TRUE,Yes

Overcast,Hot,Normal,FALSE,Yes

Rainy,Mild,High,TRUE,No

***#PROGRAM.No.5* Implementation of Back propagation Algorithm**

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)

def sigmoid(x):

return 1 / (1 + np.exp(-x))

def der\_sigmoid(x):

return x \* (1 - x)

epoch = 5000

lr = 0.01

neurons\_i = 2

neurons\_h = 3

neurons\_o = 1

weight\_h = np.random.uniform(size=(neurons\_i, neurons\_h))

bias\_h = np.random.uniform(size=(1, neurons\_h))

weight\_o = np.random.uniform(size=(neurons\_h, neurons\_o))

bias\_o = np.random.uniform(size=(1, neurons\_o))

for i in range(epoch):

inp\_h = np.dot(X, weight\_h) + bias\_h

out\_h = sigmoid(inp\_h)

inp\_o = np.dot(out\_h, weight\_o) + bias\_o

out\_o = sigmoid(inp\_o)

err\_o = y - out\_o

grad\_o = der\_sigmoid(out\_o)

delta\_o = err\_o \* grad\_o

err\_h = delta\_o.dot(weight\_o.T)

grad\_h = der\_sigmoid(out\_h)

delta\_h = err\_h \* grad\_h

weight\_o += out\_h.T.dot(delta\_o) \* lr

weight\_h += X.T.dot(delta\_h) \* lr

print('Input: ', X)

print('Actual: ', y)

print('Predicted: ', out\_o)

OUTPUT:

Input: [[0.66666667 1. ]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual: [[0.92]

[0.86]

[0.89]]

Predicted: [[0.91842143]

[0.90887464]

[0.9184287 ]]

**#PROGRAM.NO.6 Naive bayes Classifier**

import pandas as pd

msg=pd.read\_csv('C:\\Users\\LAB\\Desktop\\SKSVMACET LAB MANUAL1\\data(csv files)\\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)

from sklearn.model\_selection import train\_test\_split

xtrain,xtest,ytrain,ytest=train\_test\_split(X,y)

print(xtest.shape)

print(xtrain.shape)

print(ytest.shape)

print(ytrain.shape)

from sklearn.feature\_extraction.text import CountVectorizer

count\_vect = CountVectorizer()

xtrain\_dtm = count\_vect.fit\_transform(xtrain)

xtest\_dtm=count\_vect.transform(xtest)

from sklearn.naive\_bayes import MultinomialNB

clf = MultinomialNB().fit(xtrain\_dtm,ytrain)

predicted = clf.predict(xtest\_dtm)

from sklearn import metrics

print('Accuracy metrics')

print('Accuracy of the classifer is',metrics.accuracy\_score(ytest,predicted))

print('Confusion matrix')

print(metrics.confusion\_matrix(ytest,predicted))

print('Recall and Precison ')

print(metrics.recall\_score(ytest,predicted))

print(metrics.precision\_score(ytest,predicted))

**output:**

The dimensions of the dataset (18, 2)

0 I love this sandwich

1 This is an amazing place

2 I feel very good about these beers

3 This is my best work

4 What an awesome view

5 I do not like this restaurant

6 I am tired of this stuff

7 I can't deal with this

8 He is my sworn enemy

9 My boss is horrible

10 This is an awesome place

11 I do not like the taste of this juice

12 I love to dance

13 I am sick and tired of this place

14 What a great holiday

15 That is a bad locality to stay

16 We will have good fun tomorrow

17 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

(5,)

(13,)

(5,)

(13,)

Accuracy metrics

Accuracy of the classifer is 0.8

Confusion matrix

[[2 0]

[1 2]]

Recall and Precison

0.6666666666666666

1.0

**Date set: naivetext.csv**

I love this sandwich,pos

This is an amazing place,pos

I feel very good about these beers,pos

This is my best work,pos

What an awesome view,pos

I do not like this restaurant,neg

I am tired of this stuff,neg

I can't deal with this,neg

He is my sworn enemy,neg

My boss is horrible,neg

This is an awesome place,pos

I do not like the taste of this juice,neg

I love to dance,pos

I am sick and tired of this place,neg

What a great holiday,pos

That is a bad locality to stay,neg

We will have good fun tomorrow,pos

I went to my enemy's house today,neg

**#PROGRAM .No.7 KNN Algorithm**

import numpy as np

import pandas as pd

from matplotlib import pyplot as plt

from sklearn.mixture import GaussianMixture

from sklearn.cluster import KMeans

data = pd.read\_csv('C:\\Users\\LAB\\Desktop\\ANEEL-ML LAB\\SKSVMACET LAB MANUAL\\DATA-SET\\data(csv files)\\ex.csv')

f1 = data['V1'].values

f2 = data['V2'].values

X = np.array(list(zip(f1, f2)))

print("x: ", X)

print('Graph for whole dataset')

plt.scatter(f1, f2, c='black') # size can be set by adding s=size as param

plt.show()

kmeans = KMeans(2)

labels = kmeans.fit(X).predict(X)

print("labels for kmeans:", labels)

print('Graph using Kmeans Algorithm')

plt.scatter(f1, f2, c=labels)

centroids = kmeans.cluster\_centers\_

print("centroids:", centroids)

plt.scatter(centroids[:, 0], centroids[:, 1], marker='\*', c='red')

plt.show()

gmm = GaussianMixture(2)

labels = gmm.fit(X).predict(X)

print("Labels for GMM: ", labels)

print('Graph using EM Algorithm')

plt.scatter(f1, f2, c=labels)

plt.show()

output:

x: [[1. 1. ]

[1.5 2. ]

[3. 4. ]

[5. 7. ]

[3.5 5. ]

[4.5 5. ]

[3.5 4.5]]

Graph for whole dataset

<Figure size 640x480 with 1 Axes>

labels for kmeans: [0 0 1 1 1 1 1]

Graph using Kmeans Algorithm

centroids: [[1.25 1.5 ]

[3.9 5.1 ]]

<Figure size 640x480 with 1 Axes>

Labels for GMM: [1 1 0 0 0 0 0]

Graph using EM Algorithm

<Figure size 640x480 with 1 Axes>

**Date set: ex.csv**

n,V1,V2

1,1,1

2,1.5,2

3,3,4

4,5,7

5,3.5,5

6,4.5,5

7,3.5,4.5

**#PROGRAM8: EM algorithm**

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()

#print(dataset)

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))

OUTPUT:

Class : number

setosa : 0

versicolor : 1

virginica : 2

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= 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

**PROGRAM .NO .9(Locally Weighted Regression Algorithm)**

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('D:\\Jyoti W\\2020-21 ML Program\\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:

GRAPH will generate

**Date set: tips.csv (enjoying the particular party bill)**

total\_bill,tip,sex,smoker,day,time,size

16.99,1.01,Female,No,Sun,Dinner,2

10.34,1.66,Male,No,Sun,Dinner,3

21.01,3.5,Male,No,Sun,Dinner,3

23.68,3.31,Male,No,Sun,Dinner,2

24.59,3.61,Female,No,Sun,Dinner,4

25.29,4.71,Male,No,Sun,Dinner,4

8.77,2,Male,No,Sun,Dinner,2

26.88,3.12,Male,No,Sun,Dinner,4

15.04,1.96,Male,No,Sun,Dinner,2

14.78,3.23,Male,No,Sun,Dinner,2

10.27,1.71,Male,No,Sun,Dinner,2

35.26,5,Female,No,Sun,Dinner,4

15.42,1.57,Male,No,Sun,Dinner,2

18.43,3,Male,No,Sun,Dinner,4

14.83,3.02,Female,No,Sun,Dinner,2

21.58,3.92,Male,No,Sun,Dinner,2

10.33,1.67,Female,No,Sun,Dinner,3

16.29,3.71,Male,No,Sun,Dinner,3

16.97,3.5,Female,No,Sun,Dinner,3

20.65,3.35,Male,No,Sat,Dinner,3

17.92,4.08,Male,No,Sat,Dinner,2

20.29,2.75,Female,No,Sat,Dinner,2

15.77,2.23,Female,No,Sat,Dinner,2

39.42,7.58,Male,No,Sat,Dinner,4

19.82,3.18,Male,No,Sat,Dinner,2

17.81,2.34,Male,No,Sat,Dinner,4

13.37,2,Male,No,Sat,Dinner,2

12.69,2,Male,No,Sat,Dinner,2

21.7,4.3,Male,No,Sat,Dinner,2

19.65,3,Female,No,Sat,Dinner,2

9.55,1.45,Male,No,Sat,Dinner,2

18.35,2.5,Male,No,Sat,Dinner,4