Machine Learning

Lab1:

Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples.

Program:

import pandas as pd

import numpy as np

data = pd.read\_csv('lab1.csv')

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

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

def search(con,tar):

for i,val in enumerate(tar):

if val=="yes":

specifichyp=con[i].copy()

break

for i,val in enumerate(con):

if tar[i]=="yes":

for x in range(len(specifichyp)):

if val[x]!=specifichyp[x]:

specifichyp[x]="?"

else:

pass

return specifichyp

print(search(concepts, target))

Output:

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

Lab2:

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.

Program:

import numpy as np

import pandas as pd

data=pd.read\_csv('data.csv')

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

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

def candidate\_elimination(con,tar):

s\_hyp=con[0].copy()

g\_hyp=[["?" for i in range(len(s\_hyp))] for i in range(len(s\_hyp))]

for i,val in enumerate(con):

if tar[i]=="yes":

for x in range(len(s\_hyp)):

if val[x]!=s\_hyp[x]:

s\_hyp[x]="?"

g\_hyp[x][x]="?"

if tar[i]=="no":

for x in range(len(s\_hyp)):

if val[x]!=s\_hyp[x]:

g\_hyp[x][x]=s\_hyp[x]

else:

g\_hyp[x][x]="?"

indices=[i for i,val in enumerate(g\_hyp) if val==["?","?","?","?","?","?"]]

for i in indices:

g\_hyp.remove(["?","?","?","?","?","?"])

return s\_hyp,g\_hyp

s\_final,g\_final=candidate\_elimination(concepts,target)

print(s\_final)

print(g\_final)

Output:

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

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

Lab3:

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.

Program:

import pandas as pd

import math

import numpy as np

data = pd.read\_csv("3-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="")

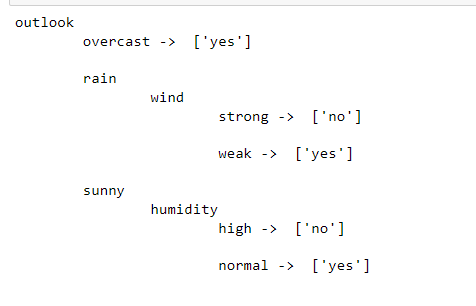
if root.isLeaf:

print(" -> ", root.pred)

print()

for child in root.children:

printTree(child, depth + 1)



Lab4:

Implement the Linear Regression algorithm in order to fit data points. Select

appropriate data set for your experiment and draw graphs.

Program:

import numpy as np

import matplotlib.pyplot as plt

def estimate\_coef(x, y):

# number of observations/points

n = np.size(x)

# mean of x and y vector

m\_x = np.mean(x)

m\_y = np.mean(y)

# calculating cross-deviation and deviation about x

SS\_xy = np.sum(y\*x) - n\*m\_y\*m\_x

SS\_xx = np.sum(x\*x) - n\*m\_x\*m\_x

# calculating regression coefficients

b\_1 = SS\_xy / SS\_xx

b\_0 = m\_y - b\_1\*m\_x

return (b\_0, b\_1)

def plot\_regression\_line(x, y, b):

# plotting the actual points as scatter plot

plt.scatter(x, y, color = "m",

marker = "o", s = 30)

# predicted response vector

y\_pred = b[0] + b[1]\*x

# plotting the regression line

plt.plot(x, y\_pred, color = "g")

# putting labels

plt.xlabel('x')

plt.ylabel('y')

# function to show plot

plt.show()

def main():

# observations / data

x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12])

y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12,13,15,14])

# estimating coefficients

b = estimate\_coef(x, y)

print("Estimated coefficients:\nb\_0 = {} \

\nb\_1 = {}".format(b[0], b[1]))

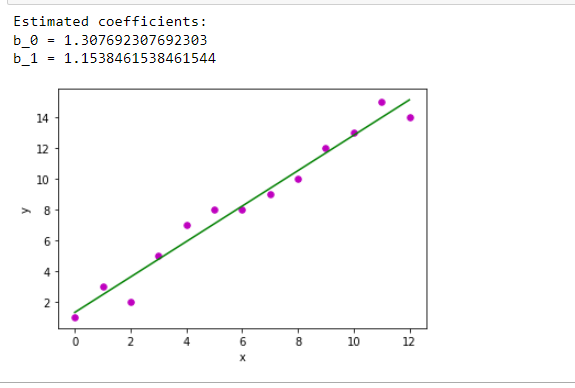
# plotting regression line

plot\_regression\_line(x, y, b)

if \_\_name\_\_ == "\_\_main\_\_":

main()

Output:



Lab5:

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

Program:

import numpy as np

import pandas as pd

from sklearn import preprocessing

from sklearn.naive\_bayes import MultinomialNB

le=preprocessing.LabelEncoder()

clf = MultinomialNB()

data=pd.read\_csv('NB.csv')

features=[feat for feat in data]

targetLabel=features[-1]

features.remove(features[-1])

features

diff\_values=[]

for f in features:

for v in data[f]:

if v not in diff\_values:

diff\_values.append(v)

diff\_values

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

dataArray

le.fit(diff\_values)

list(le.classes\_)

trans=[]

for d in dataArray:

trans.append(le.transform(d))

trans

target=data[targetLabel]

target

target=np.array(target)

tar=[]

for t in target:

if t == "yes":

tar.append(1)

else:

tar.append(0)

tar

clf.fit(trans,tar)

predicting=["sunny","cool","high","strong"]

pre\_array=le.transform(predicting)

pre\_array=np.reshape(pre\_array,(1,4))

pre\_array

print(clf.predict(pre\_array))

Output: [0]

Lab6:

Apply k-Means algorithm to cluster a set of data stored in a .CSV file.

Program:

# importing libraries

import numpy as nm

import matplotlib.pyplot as mtp

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('Kmeans\_data.csv')

x = dataset.iloc[:, [3, 4]].values

#finding optimal number of clusters using the elbow method

from sklearn.cluster import KMeans

wcss\_list= [] #Initializing the list for the values of WCSS

#Using for loop for iterations from 1 to 10.

for i in range(1, 11):

kmeans = KMeans(n\_clusters=i, init='k-means++', random\_state= 42)

kmeans.fit(x)

wcss\_list.append(kmeans.inertia\_)

mtp.plot(range(1, 11), wcss\_list)

mtp.title('The Elobw Method Graph')

mtp.xlabel('Number of clusters(k)')

mtp.ylabel('wcss\_list')

mtp.show()

#training the K-means model on a dataset

kmeans = KMeans(n\_clusters=5, init='k-means++', random\_state= 42)

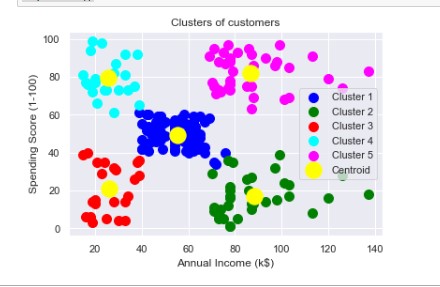
y\_predict= kmeans.fit\_predict(x)

#training the K-means model on a dataset

kmeans = KMeans(n\_clusters=5, init='k-means++', random\_state= 42)

y\_predict= kmeans.fit\_predict(x)

Output:



Lab7:

Write a program to construct a Bayesian network considering training data. Use this model to make predictions.

Program:

import pgmpy.models

import pgmpy.inference

import networkx as nx

import pylab as plt

# Create a bayesian network

model = pgmpy.models.BayesianModel([('Burglary', 'Alarm'),

('Earthquake', 'Alarm'),

('Alarm', 'JohnCalls'),

('Alarm', 'MaryCalls')])

# Define conditional probability distributions (CPD)

# Probability of burglary (True, False)

cpd\_burglary = pgmpy.factors.discrete.TabularCPD('Burglary', 2, [[0.001], [0.999]])

# Probability of earthquake (True, False)

cpd\_earthquake = pgmpy.factors.discrete.TabularCPD('Earthquake', 2, [[0.002], [0.998]])

# Probability of alarm going of (True, False) given a burglary and/or earthquake

cpd\_alarm = pgmpy.factors.discrete.TabularCPD('Alarm', 2, [[0.95, 0.94, 0.29, 0.001],

[0.05, 0.06, 0.71, 0.999]],

evidence=['Burglary', 'Earthquake'],

evidence\_card=[2, 2])

# Probability that John calls (True, False) given that the alarm has sounded

cpd\_john = pgmpy.factors.discrete.TabularCPD('JohnCalls', 2, [[0.90, 0.05],

[0.10, 0.95]],

evidence=['Alarm'],

evidence\_card=[2])

# Probability that Mary calls (True, False) given that the alarm has sounded

cpd\_mary = pgmpy.factors.discrete.TabularCPD('MaryCalls', 2, [[0.70, 0.01],

[0.30, 0.99]],

evidence=['Alarm'],

evidence\_card=[2])

# Add CPDs to the network structure

model.add\_cpds(cpd\_burglary, cpd\_earthquake, cpd\_alarm, cpd\_john, cpd\_mary)

# Check if the model is valid, throw an exception otherwise

model.check\_model()

# Print probability distributions

print('Probability distribution, P(Burglary)')

print(cpd\_burglary)

print()

print('Probability distribution, P(Earthquake)')

print(cpd\_earthquake)

print()

print('Joint probability distribution, P(Alarm | Burglary, Earthquake)')

print(cpd\_alarm)

print()

print('Joint probability distribution, P(JohnCalls | Alarm)')

print(cpd\_john)

print()

print('Joint probability distribution, P(MaryCalls | Alarm)')

print(cpd\_mary)

print()

# Plot the model

nx.draw(model, with\_labels=True)

plt.savefig('C:\\Users\\admin\\Desktop')

plt.close()

# Perform variable elimination for inference

# Variable elimination (VE) is a an exact inference algorithm in bayesian networks

infer = pgmpy.inference.VariableElimination(model)

# Calculate the probability of a burglary if John and Mary calls (0: True, 1: False)

posterior\_probability = infer.query(['Burglary'], evidence={'JohnCalls': 0, 'MaryCalls': 0})

# Print posterior probability

print('Posterior probability of Burglary if JohnCalls(True) and MaryCalls(True)')

print(posterior\_probability)

print()

# Calculate the probability of alarm starting if there is a burglary and an earthquake (0: True, 1: False)

posterior\_probability = infer.query(['Alarm'], evidence={'Burglary': 0, 'Earthquake': 0})

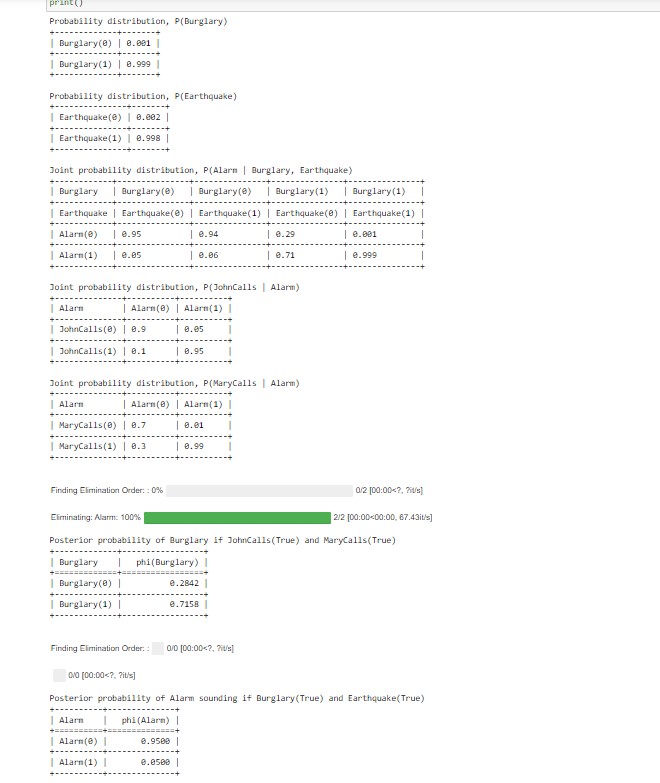
# Print posterior probability

print('Posterior probability of Alarm sounding if Burglary(True) and Earthquake(True)')

print(posterior\_probability)

print()

Output:



Lab8:

Apply EM algorithm to cluster a set of data stored in a .CSV file. Compare the

results of k-Means algorithm and EM algorithm.

Program:

# import libraries

# For plotting

import matplotlib.pyplot as plt

import seaborn as sns

sns.set\_style("white")

%matplotlib inline

#for matrix math

import numpy as np

#for normalization + probability density function computation

from scipy import stats

#for data preprocessing

import pandas as pd

from math import sqrt, log, exp, pi

from random import uniform

print("import done")

random\_seed=36788765

np.random.seed(random\_seed)

Mean1 = 2.0 # Input parameter, mean of first normal probability distribution

Standard\_dev1 = 4.0 #@param {type:"number"}

Mean2 = 9.0 # Input parameter, mean of second normal probability distribution

Standard\_dev2 = 2.0 #@param {type:"number"}

# generate data

y1 = np.random.normal(Mean1, Standard\_dev1, 1000)

y2 = np.random.normal(Mean2, Standard\_dev2, 500)

data=np.append(y1,y2)

# For data visiualisation calculate left and right of the graph

Min\_graph = min(data)

Max\_graph = max(data)

x = np.linspace(Min\_graph, Max\_graph, 2000) # to plot the data

print('Input Gaussian {:}: μ = {:.2}, σ = {:.2}'.format("1", Mean1, Standard\_dev1))

print('Input Gaussian {:}: μ = {:.2}, σ = {:.2}'.format("2", Mean2, Standard\_dev2))

sns.distplot(data, bins=20, kde=False)

from sklearn.mixture import GaussianMixture

gmm = GaussianMixture(n\_components = 2, tol=0.000001, max\_iter = 100)

gmm.fit(np.expand\_dims(data, 1)) # Parameters: array-like, shape (n\_samples, n\_features), 1 dimension dataset so 1 feature

Gaussian\_nr = 1

print('Input Gaussian {:}: μ = {:.2}, σ = {:.2}'.format("1", Mean1, Standard\_dev1))

print('Input Gaussian {:}: μ = {:.2}, σ = {:.2}'.format("2", Mean2, Standard\_dev2))

for mu, sd, p in zip(gmm.means\_.flatten(), np.sqrt(gmm.covariances\_.flatten()), gmm.weights\_):

print('Gaussian {:}: μ = {:.2}, σ = {:.2}, weight = {:.2}'.format(Gaussian\_nr, mu, sd, p))

g\_s = stats.norm(mu, sd).pdf(x) \* p

plt.plot(x, g\_s, label='gaussian sklearn');

Gaussian\_nr += 1

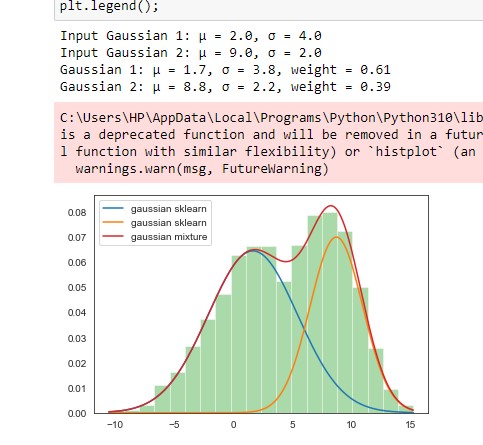
sns.distplot(data, bins=20, kde=False, norm\_hist=True)

gmm\_sum = np.exp([gmm.score\_samples(e.reshape(-1, 1)) for e in x]) #gmm gives log probability, hence the exp() function

plt.plot(x, gmm\_sum, label='gaussian mixture');

plt.legend();

Output:



Lab9:

Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions.

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=datasets.load\_iris()

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)

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)

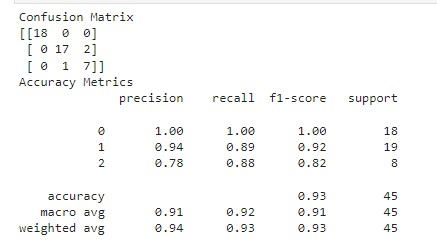
print('Confusion Matrix')

print(confusion\_matrix(y\_test,y\_pred))

print('Accuracy Metrics')

print(classification\_report(y\_test,y\_pred))

Output:

Lab10:

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

Program:

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,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='blue')

ax.plot(xsort[:,1],ypred[SortIndex], color = 'red', linewidth=5)

plt.xlabel('Total bill')

plt.ylabel('Tip')

plt.show();

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

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)]]))

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:

