**ML LAB PROGRAMS**

**LAB1 [FIND-S ALGORITHM] :**

**PROGRAM:**

import csv

with open('file1.csv','r') as f:

reader=csv.reader(f)

your\_list=list(reader)

h=[['0','0','0','0','0','0']]

for i in your\_list:

print(i)

if i[-1]=="TRUE":

j=0

for x in i:

if x!="TRUE":

if x!=h[0][j] and h[0][j]=='0':

h[0][j]=x

elif x!=h[0][j] and h[0][j]!='0':

h[0][j]='?'

else:

pass

j=j+1

print("Most Specific Hyothesis")

print(h)

**INPUT[EXCEL]:**file1.csv

sunny,warm,normal,strong,warm,same,TRUE

sunny,warm,high,strong,warm,same,TRUE

rainy,cold,high,strong,warm,change,FALSE

sunny,warm,high,strong,cold,change,TRUE

**OUTPUT:**

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

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

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

['sunny', 'warm', 'high', 'strong', 'cold', 'change', 'TRUE']

Most Specific Hypothesis

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

**LAB2 [CANDIDATE ELIMINATION ALGORITHM]:**

**PROGRAM:**

dataarr=[]

with open('laba2.csv') as f:

for line in f:

dataarr.append(line.strip().split(','))

rows=len(dataarr)

cols=len(dataarr[0])

shypo=['0']\*(cols-1)

ghypo=[['?']\*(cols-1)]

print "initial specific hypothesis is: ",shypo

print "initial general hypothesis is: ",ghypo

for x in range(1,rows):

lst=dataarr[x]

if lst[cols-1]=="1":

for i in range(0,cols-1):

if shypo[i]==lst[i]:

continue

shypo[i]='?' if shypo[i]!='0' else lst[i]

for g in ghypo:

if g[i]!='?' and shypo[i]=='?':

ghypo.remove(g)

elif lst[cols-1]=="0":

for i in range(0,cols-1):

if lst[i]!=shypo[i] and shypo[i]!='?':

temp\_list=['?']\*i+[shypo[i]]+(['?']\*(cols-2-i))

if temp\_list not in ghypo:

ghypo.append(temp\_list)

print "S",[x],"=",shypo

print "G",[x],"=",ghypo

print "S[4]=",shypo

print "G[4]=",ghypo

print "\nFinal hypothesis is: ",shypo

**INPUT[EXCEL]:**lab2.csv

sunny,warm,normal,strong,warm,same,1

sunny,warm,normal,strong,warm,same,1

rainy,cold,high,strong,warm,change,0

sunny,warm,high,strong,cool,change,1

**OUTPUT:**

initial specific hypothesis is: ['0', '0', '0', '0', '0', '0']

initial general hypothesis is: [['?', '?', '?', '?', '?', '?']]

S [1] = [‘sunny’,’warm’,’normal’,’strong’,’warm’,’same’]

G [1] = [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’]

S [2] = ['sunny', 'warm', '?', 'strong', '?', '?']

G [2] = [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’]

S [3] = ['sunny', 'warm', '?', 'strong', '?', '?']

G [3] = [['?', '?', '?', '?', '?', '?'], ['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]

S[4]= ['sunny', 'warm', '?', 'strong', '?', '?']

G[4]= [['?', '?', '?', '?', '?', '?'], ['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]

Final hypothesis is: ['sunny', 'warm', '?', 'strong', '?', '?']

**LAB3 [ID3 ALGORITHM]:**

**PROGRAM:**

**ID3.PY**

import numpy as np

import math

from data\_loader import read\_data

class Node:

def \_\_init\_\_(self,attribute):

self.attribute=attribute

self.children=[]

self.answer=""

def subtables(data,col,delete):

dict={}

items=np.unique(data[:,col])

count=np.zeros((items.shape[0],1),dtype=np.int32)

for x in range(items.shape[0]):

for y in range(data.shape[0]):

if data[y,col]==items[x]:

count[x]+=1

for x in range(items.shape[0]):

dict[items[x]]=np.empty((int(count[x]),data.shape[1]),dtype="S32")

pos=0

for y in range(data.shape[0]):

if data[y,col]==items[x]:

dict[items[x]][pos]=data[y]

pos+=1

if delete:

dict[items[x]]=np.delete(dict[items[x]],col,1)

return items,dict

def entropy(S):

items=np.unique(S)

if items.size==1:

return 0

counts=np.zeros((items.shape[0],1))

sums=0

for x in range(items.shape[0]):

counts[x]=sum(S==items[x])/(S.size\*1.0)

for count in counts:

sums+=-1\*count\*math.log(count,2)

return sums

def gain\_ratio(data,col):

items,dict=subtables(data,col,delete=False)

total\_size=data.shape[0]

entropies=np.zeros((items.shape[0],1))

intrinsic=np.zeros((items.shape[0],1))

for x in range(items.shape[0]):

ratio=dict[items[x]].shape[0]/(total\_size\*1.0)

entropies[x]=ratio\*entropy(dict[items[x]][:,-1])

intrinsic[x]=ratio\*math.log(ratio,2)

total\_entropy=entropy(data[:,-1])

iv=-1\*sum(intrinsic)

for x in range(entropies.shape[0]):

total\_entropy-=entropies[x]

return total\_entropy/iv

def create\_node(data,metadata):

if(np.unique(data[:,-1])).shape[0]==1:

node=Node("")

node.answer=np.unique(data[:,-1])[0]

return node

gains=np.zeros((data.shape[1]-1,1))

for col in range(data.shape[1]-1):

gains[col]=gain\_ratio(data,col)

split=np.argmax(gains)

node=Node(metadata[split])

metadata=np.delete(metadata,split,0)

items,dict=subtables(data,split,delete=True)

for x in range(items.shape[0]):

child=create\_node(dict[items[x]],metadata)

node.children.append((items[x],child))

return node

def empty(size):

s=""

for x in range(size):

s+=" "

return s

def print\_tree(node,level):

if node.answer!="":

print(empty(level),node.answer)

return

print(empty(level),node.attribute)

for value,n in node.children:

print(empty(level+1),value)

print\_tree(n,level+2)

metadata,traindata=read\_data("lab3.csv")

data=np.array(traindata)

node=create\_node(data,metadata)

print\_tree(node,0)

**data\_loader.py**

import csv

def read\_data(filename):

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

datareader=csv.reader(csvfile,delimiter=',')

headers=next(datareader)

metadata=[]

traindata=[]

for name in headers:

metadata.append(name)

for row in datareader:

traindata.append(row)

return(metadata,traindata)

**INPUT[EXCEL]:**lab3.csv

OUTLOOK,AIR\_TEMP,HUMIDITY,WIND,ENJOYSPORT

SUNNY,HOT,HIGH,WEAK,NO

SUNNY,HOT,HIGH,STRONG,NO

OVERCAST,HOT,HIGH,WEAK,YES

RAIN,MILD,HIGH,WEAK,YES

RAIN,COOL,NORMAL,WEAK,YES

RAIN,COOL,NORMAL,STRONG,NO

OVERCAST,COOL,NORMAL,STRONG,YES

SUNNY,MILD,HIGH,WEAK,NO

SUNNY,COOL,NORMAL,WEAK,YES

RAIN,MILD,NORMAL,WEAK,YES

SUNNY,MILD,NORMAL,STRONG,YES

OVERCAST,MILD,HIGH,STRONG,YES

OVERCAST,HOT,NORMAL,WEAK,YES

RAIN,MILD,HIGH,STRONG,NO

**OUTPUT:**

('', 'OUTLOOK')

(' ', 'OVERCAST')

(' ', 'YES')

(' ', 'RAIN')

(' ', 'WIND')

(' ', 'STRONG')

(' ', 'NO')

(' ', 'WEAK')

(' ', 'YES')

(' ', 'SUNNY')

(' ', 'HUMIDITY')

(' ', 'HIGH')

(' ', 'NO')

(' ', 'NORMAL')

(' ', 'YES')

**LAB4 [BACK PROPAGATION ALGORITHM]:**

**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)

y=y/100

def sigmoid(x):

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

def derivatives\_sigmoid(x):

return x\*(1-x)

epoch=7000

lr=0.1

inputlayer\_neurons=2

hiddenlayer\_neurons=3

output\_neurons=1

wh=np.random.uniform(size=(inputlayer\_neurons,hiddenlayer\_neurons))

bh=np.random.uniform(size=(1,hiddenlayer\_neurons))

wout=np.random.uniform(size=(hiddenlayer\_neurons,output\_neurons))

bout=np.random.uniform(size=(1,output\_neurons))

for i in range(epoch):

hinp1=np.dot(x,wh)

hinp=hinp1+bh

hlayer\_act=sigmoid(hinp)

outinp1=np.dot(hlayer\_act,wout)

outinp=outinp1+bout

output=sigmoid(outinp)

print("Input:\n"+str(x))

print("Actual output:\n"+str(y))

print("Predicted output:\n",output)

**OUTPUT:**

[[0.66666667 1. ]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual output:

[[0.92]

[0.86]

[0.89]]

('Predicted output:\n', array([[0.87550341],

[0.86351179],

[0.87336642]]))

**LAB 5 [NAÏVE BAYESIAN CLASSIFIER]:**

**PROGRAM:**

from sklearn.datasets import load\_iris

iris=load\_iris()

x=iris.data

y=iris.target

from sklearn.naive\_bayes import GaussianNB

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

xtrain,xtest,ytrain,ytest=train\_test\_split(x,y,test\_size=0.4,random\_state=2)

print("training data",xtrain)

print("training data",ytrain)

print("testing data",xtest)

print("testing data",ytest)

gnb=GaussianNB()

gnb.fit(xtrain,ytrain)

y\_pred=gnb.predict(xtest)

from sklearn import metrics

print("accuracy is",metrics.accuracy\_score(ytest,y\_pred)\*100)

**OUTPUT:**

**('accuracy is', 93.33333333333333)**

**LAB 6 [BAYESIAN TEXT CLASSIFIER MODEL]:**

**PROGRAM:**

import pandas as pd

msg=pd.read\_csv('laba6.txt',names=['message','label'])

print"dimensions of 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)

print(count\_vect.get\_feature\_names())

df=pd.DataFrame(xtrain\_dtm.toarray(),columns=count\_vect.get\_feature\_names())

print(df)

print(xtrain\_dtm)

from sklearn.naive\_bayes import MultinomialNB

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

predicted=df.predict(xtest\_dtm)

from sklearn import metrics

print'accuracy metrics'

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

print'confusion matrix'

print metrics.confusion\_matrix(ytest,predicted)

print'recall and precision'

print metrics.precision\_score(ytest,predicted)

**INPUT[EXCEL]:**lab6.txt

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

**OUTPUT:**

[13 rows x 45 columns]

(0, 25) 1

(0, 1) 1

(0, 2) 1

(0, 17) 1

(0, 34) 1

(1, 4) 1

(1, 25) 1

(1, 2) 1

(1, 17) 1

.

.

.

(11, 34) 1

(12, 26) 1

(12, 19) 1

(12, 23) 1

(12, 8) 1

(12, 34) 1

accuracy metrics

accuracy of the classifier is 0.6

confusion matrix

[[2 0]

[2 1]]

recall and precision

1.0

**LAB 7 [BAYESIAN NETWORK]:**

**PROGRAM:**

from pomegranate import \*

Asia=DiscreteDistribution({'True':0.5,'False':0.5})

Tuberculosis=ConditionalProbabilityTable(

[['True','True',0.2],

['True','False',0.8],

['False','True',0.1],

['False','False',0.98]],[Asia])

smoking=DiscreteDistribution({'True':0.5,'False':0.5})

Lung=ConditionalProbabilityTablse(

[['True','True',0.75],

['True','False',0.25],

['False','True',0.02],

['False','False',0.98]],[smoking])

Bronchitis=ConditionalProbabilityTable(

[['True','True',0.92],

['True','False',0.08],

['False','True',0.03],

['False','False',0.98]],[smoking])

Tuberculosis\_or\_cancer=ConditionalProbabilityTable(

[['True','True','True',1.0],

['True','True','False',0.0]

['True','False','True',1.0],

['True','False','False',0.0],

['False','True','True',1.0],

['False','True','False',0.0],

['False','False','True',1.0]

['False','False','False',0.0]],[Tuberculosis,Lung])

Xray=ConditionalProbabilityTable(

[['True','True',0.885],

['True','False',0.115],

['False','True',0.04],

['False','False',0.96]],[Tuberculosis\_or\_cancer])

dyspnea=ConditionalProbabilityTable(

[['True','True','True',0.96],

['True','True','False',0.04]

['True','False','True',0.89],

['True','False','False',0.11],

['False','True','True',0.96],

['False','True','False',0.04],

['False','False','True',0.89]

['False','False','False',0.11]],[Tuberculosis\_or\_cancer,Bronchitis])

s0=State(Asia,name="Asia")

s1=State(Tuberculosis,name="Tuberculosis")

s2=State(smoking,name="smoker")

network=BayesianNetwork("asia")

network.add\_nodes(s0,s1,s2)

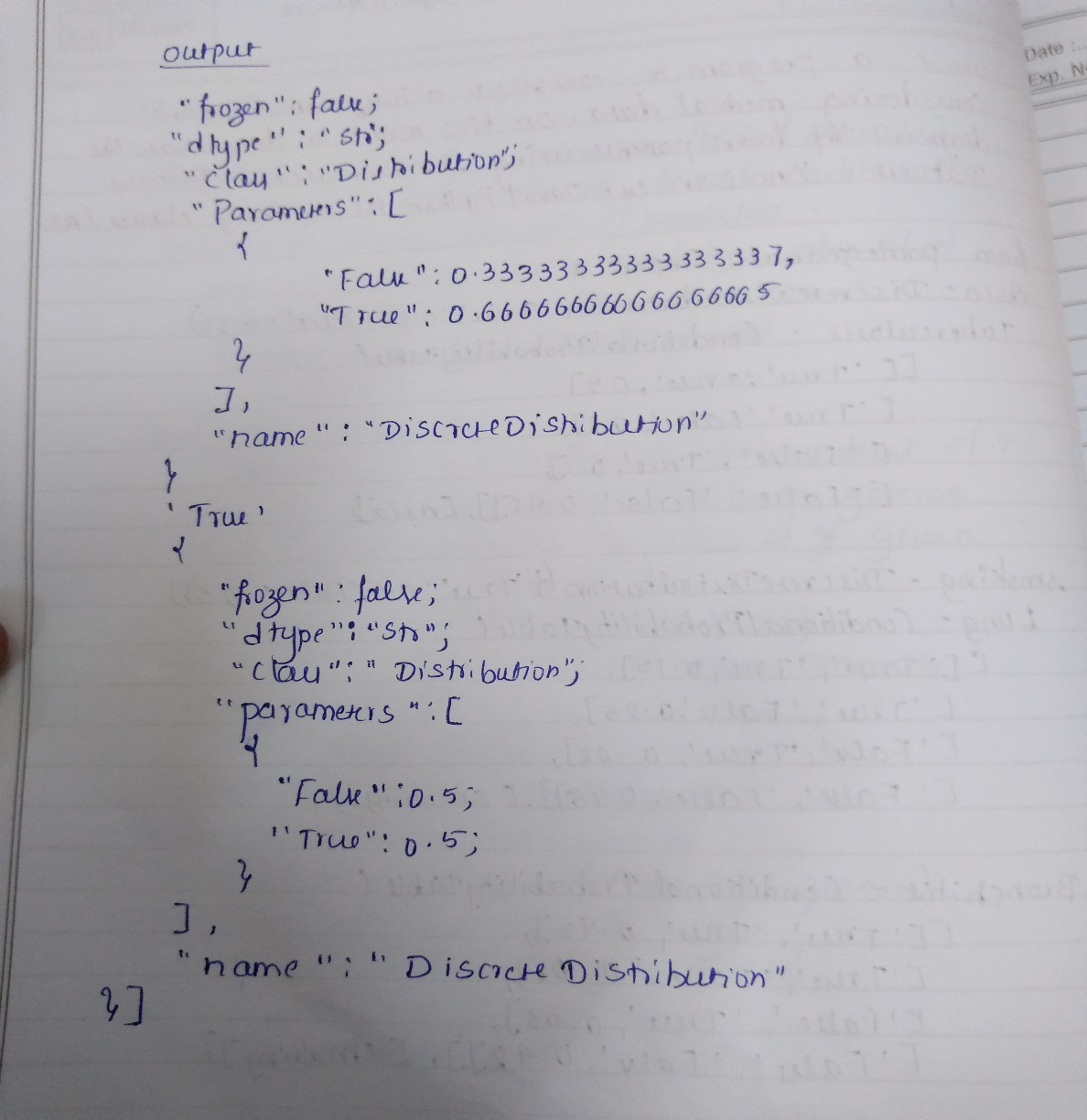
network.add\_edge(s0,s1)

network.add\_edges(s1,s2)

network.bake()

Print(network.predict\_probal({'tuberculosis':'True'}))

**OUTPUT :**



**LAB 8 [EM-KMEANS]:**

**PROGRAM:**

import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn.cluster import KMeans

import pandas as pd

import numpy as np

import sklearn.metrics as sm

iris=datasets.load\_iris()

X=pd.DataFrame(iris.data)

X.columns=['Sepal\_Length','Sepal\_Width','Petal\_Length','Petal\_Width']

y=pd.DataFrame(iris.target)

y.columns=['Targets']

plt.figure(figsize=(14,7))

model=KMeans(n\_clusters=3)

model.fit(X)

model.labels\_

plt.figure(figsize=(14,7))

colormap=np.array(['red','lime','black'])

plt.subplot(1,2,1)

plt.scatter(X.Petal\_Length,X.Petal\_Width,c=colormap[y.Targets],s=40)

plt.title('EM clustering')

plt.subplot(1,2,2)

plt.scatter(X.Petal\_Length,X

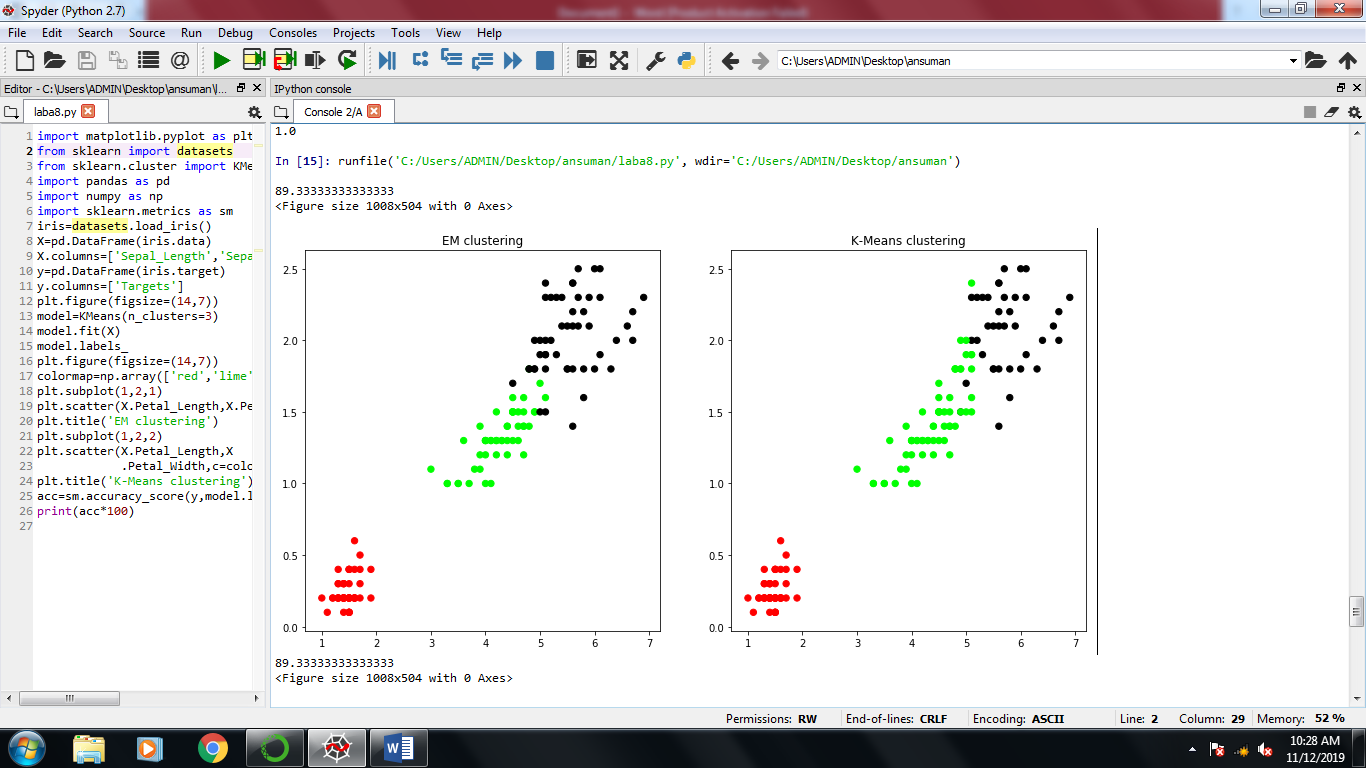
.Petal\_Width,c=colormap[model.labels\_],s=40)

plt.title('K-Means clustering')

acc=sm.accuracy\_score(y,model.labels\_)

print(acc\*100)

**OUTPUT:**



**LAB 9 [K-NEAREST NEIGHBOR ALGORITHM]:**

**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()

iris\_data=iris.data

iris\_labels=iris.target

print(iris\_data)

print(iris\_labels)

x\_train,x\_test,y\_train,y\_test=train\_test\_split(iris\_data,iris\_labels,test\_size=0.30)

classifier=KNeighborsClassifier(n\_neighbors=5)

classifier.fit(x\_train,y\_train)

y\_pred=classifier.predict(x\_test)

print('confusion matrix is as follows')

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

print('Accuracy metrices')

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

OUTPUT:

confusion matrix is as follows

[[17 0 0]

[ 0 14 1]

[ 0 1 12]]

Accuracy metrices

precision recall f1-score support

0 1.00 1.00 1.00 17

1 0.93 0.93 0.93 15

2 0.92 0.92 0.92 13

avg / total 0.96 0.96 0.96 45

**LAB 10 [LOCALLY WEIGHTED REGRESSION ALGORITHM]:**

**PROGRAM:**

import matplotlib.pyplot as plt

import pandas as pd

#import numpy.linalg as np

import numpy as np1

#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

data=pd.read\_csv('10data.csv')

bill=np1.array(data.total\_bill)

tip=np1.array(data.tip)

mbill=np1.mat(bill)

mtip=np1.mat(tip)

m=np1.shape(mbill)[1]

one=np1.mat(np1.ones(m))

x=np1.hstack((one.T,mbill.T))

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=3)

plt.xlabel('total bill')

plt.ylabel('tip')

**OUTPUT:**

