# LAB5 [BACK PROPAGATION ALGORITHM]:

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

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 6 [NAÏVE BAYESIAN CLASSIFIER]:

**PROGRAM:** 6. Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

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 8 [K-NEAREST NEIGHBOR ALGORITHM]:

**PROGRAM:** Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

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 7 [EM-KMEANS]:

**PROGRAM: Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set**

# for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

**[EM-KMEANS]:**

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)

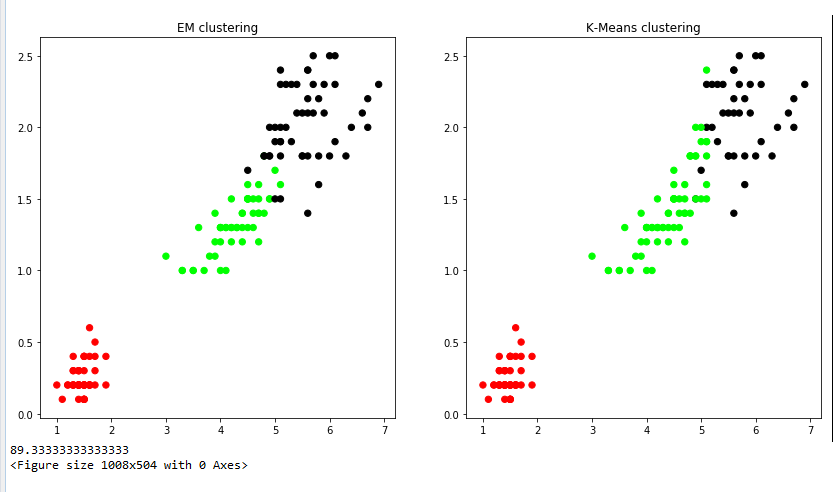
model.labels\_ plt.figure(figsize=(14,7)) colormap=np.array(['red','lime','black']) plt.subplot(1,2,1)

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 [LOCALLY WEIGHTED REGRESSION ALGORITHM]:**

**PROGRAM:** Implement the non-parametric Locally Weighted Regressionalgorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs

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:

