**Program 1**

from heuristicsearch.a\_star\_search import AStar

adjacency\_list = {

'S': [('A', 1), ('G', 10)],

'A': [('B', 2), ('C', 1)],

'B': [('D', 5)],

'C': [('D', 3), ('G', 4)],

'D': [('G', 2)]

}

heuristics = {'S': 1, 'A': 1, 'B': 1, 'C': 1, 'D': 1, 'G': 1}

graph = AStar(adjacency\_list, heuristics)

graph.apply\_a\_star(start= ’S’, stop= ’G’)

**Program 2**

from heuristicsearch.ao\_star import AOStar

print("Graph - 1")

heuristic = {'A': 1, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}

adjacency\_list = {

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

'B': [[('G', 1)], [('H', 1)]],

# 'C': [[('J', 1)]],

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

# 'G': [[('I', 1)]]

}

graph = AOStar(adjacency\_list, heuristic, 'A')

graph.applyAOStar()

**Program 4**

from decisiontree.ID3Algorithm import ID3

id3\_2 = ID3(dataset\_train.csv,headers\_train.csv,dataset\_test.csv,headers\_test.csv)

# dataset\_train contains the training dataset with headers as headers\_train

# dataset\_test contains unlabled data with headers as headers\_test

# all the agruments are of type list

id3\_2.build\_tree()

id3\_2.classify()

**Program 5**

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)

hiddengrad = derivatives\_sigmoid(hlayer\_act)#how much hidden layer wts contributed to error

d\_hiddenlayer = EH \* hiddengrad

wout += hlayer\_act.T.dot(d\_output) \*lr# dotproduct of nextlayererror and currentlayerop

#bout += np.sum(d\_output, axis=0,keepdims=True) \*lr

wh += X.T.dot(d\_hiddenlayer) \*lr

#bh += np.sum(d\_hiddenlayer, axis=0,keepdims=True) \*lr

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

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

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

**Program 7(Sample-1)**

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 data

plt.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()

**Program 7( Sample 2)**

import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn.cluster import KMeans

import pandas as pd

import numpy as np

# import some data to play with

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']

# Build the K Means Model

model = KMeans(n\_clusters=3)

model.fit(X) # model.labels\_ : Gives cluster no for which samples belongs to

# # Visualise the clustering results

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

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

# Plot the Original Classifications using Petal features

plt.subplot(2, 2, 1)

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

plt.title('Real Clusters')

plt.xlabel('Petal Length')

plt.ylabel('Petal Width')

# Plot the Models Classifications

plt.subplot(2, 2, 2)

plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colormap[model.labels\_], s=40)

plt.title('K-Means Clustering')

plt.xlabel('Petal Length')

plt.ylabel('Petal Width')

# General EM for GMM

from sklearn import preprocessing

# transform your data such that its distribution will have a

# mean value 0 and standard deviation of 1.

scaler = preprocessing.StandardScaler()

scaler.fit(X)

xsa = scaler.transform(X)

xs = pd.DataFrame(xsa, columns = X.columns)

from sklearn.mixture import GaussianMixture

gmm = GaussianMixture(n\_components=3)

gmm.fit(xs)

gmm\_y = gmm.predict(xs)

plt.subplot(2, 2, 3)

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

plt.title('GMM Clustering')

plt.xlabel('Petal Length')

plt.ylabel('Petal Width')

print('Observation: The GMM using EM algorithm based clustering matched the true labels more closely than the Kmeans.')

**Program 8**

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. This means 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, recall and

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))

**Program 9**

import numpy as np

import matplotlib.pyplot as plt

def local\_regression(x0, X, Y, tau):

x0 = [1, x0]

X = [[1, i] for i in X]

X = np.asarray(X)

xw = (X.T) \* np.exp(np.sum((X - x0) \*\* 2, axis=1) / (-2 \* tau))

beta = np.linalg.pinv(xw @ X) @ xw @ Y @ x0

return beta

def draw(tau):

prediction = [local\_regression(x0, X, Y, tau) for x0 in domain]

plt.plot(X, Y, 'o', color='black')

plt.plot(domain, prediction, color='red')

plt.show()

X = np.linspace(-3, 3, num=1000)

domain = X

Y = np.log(np.abs(X \*\* 2 - 1) + .5)

draw(10)

draw(0.1)

draw(0.01)

draw(0.001)