# Perceptron Code:

import numpy as np

import pandas as pd

class PerceptronTraining():

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

np.random.seed(1)

self.weights = 2\* np.random.random((2,1)) - 1

def sig(self, x):

calculate = 1/(1+np.exp(-x))

calculate[calculate >= 0.5] = 1

calculate[calculate < 0.5] = 0

return calculate

def sig\_derivative(self,x):

cal = x \* (1-x)

return cal

def training\_time(self, train\_inputs, train\_outputs, no\_of\_epoch):

for i in range(no\_of\_epoch):

pred\_output = self.predValue(train\_inputs)

error = train\_outputs - pred\_output

ajustmentForWeights = np.dot(train\_inputs.T,0.001\*error\*self.sig\_derivative(pred\_output))

self.weights += ajustmentForWeights

def predValue(self,inputs):

inputs = inputs.astype(float)

output = self.sig(np.dot(inputs, self.weights))

return output

def checkLabel(self,inp):

if inp == '+':

return '1'

else:

return '-1'

def predAccuracy(self,originalLabel,predValue):

matched = 0

for i in range(len(originalLabel)):

if originalLabel[i] == predValue[i]:

matched += 1

accuracyVal = matched / float(len(originalLabel)) \* 100.0

return accuracyVal

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

rad =2

d =0

n\_samp =1000

width =3

if rad < width/2:

print('The radius should be at least larger than half the width')

if n\_samp % 2 != 0 :

print('Please make sure the number of samples is even')

aa= np.random.rand(2,(int)(n\_samp/2))

radius = (rad-width/2) + width\*aa[0,:]

radius=np.reshape(radius, (1,np.product(radius.shape)))

theta = 3.14\*aa[1,:]

theta=np.reshape(theta, (1,np.product(theta.shape)))

x = radius\*np.cos(theta)

x=np.reshape(x, (1,np.product(x.shape)))

y = radius\*np.sin(theta)

y=np.reshape(y, (1,np.product(y.shape)))

label = 1\*np.ones([1,x.size])

x1 = radius\*np.cos(-theta)+rad

x1=np.reshape(x1, (1,np.product(x1.shape)))

y1 = radius\*np.sin(-theta)-d

y1=np.reshape(y1, (1,np.product(y1.shape)))

label1 = 0\*np.ones([1,x.size])

data1 = np.vstack(( np.hstack((x,x1)),np.hstack((y,y1)) ))

data2 = np.hstack( (label,label1) )

data = np.concatenate( (data1,data2 ),axis=0 )

n\_row = data.shape[0]

n\_col = data.shape[1]

shuffle\_seq = np.random.permutation(n\_col)

data\_shuffled = np. random.rand(3,1000)

for i in range(n\_col):

data\_shuffled[:,i] = data[:,shuffle\_seq[i] ];

#print(data\_shuffled[0] [0])

#print(data\_shuffled[0].shape)

train\_input\_data = np.stack([data\_shuffled[0], data\_shuffled[1]], axis=1)

nn = PerceptronTraining()

outputLabel = data\_shuffled[2].reshape(1000,1)

nn.training\_time(train\_input\_data,outputLabel,1000)

rad =2

d =0

n\_samp =2000

width =3

if rad < width/2:

print('The radius should be at least larger than half the width')

if n\_samp % 2 != 0 :

print('Please make sure the number of samples is even')

aa= np.random.rand(2,(int)(n\_samp/2))

radius = (rad-width/2) + width\*aa[0,:]

radius=np.reshape(radius, (1,np.product(radius.shape)))

theta = 3.14\*aa[1,:]

theta=np.reshape(theta, (1,np.product(theta.shape)))

x = radius\*np.cos(theta)

x=np.reshape(x, (1,np.product(x.shape)))

y = radius\*np.sin(theta)

y=np.reshape(y, (1,np.product(y.shape)))

label = 1\*np.ones([1,x.size])

x1 = radius\*np.cos(-theta)+rad

x1=np.reshape(x1, (1,np.product(x1.shape)))

y1 = radius\*np.sin(-theta)-d

y1=np.reshape(y1, (1,np.product(y1.shape)))

label1 = 0\*np.ones([1,x.size])

data1 = np.vstack(( np.hstack((x,x1)),np.hstack((y,y1)) ))

data2 = np.hstack( (label,label1) )

data = np.concatenate( (data1,data2 ),axis=0 )

n\_row = data.shape[0]

n\_col = data.shape[1]

shuffle\_seq = np.random.permutation(n\_col)

data\_shuffled = np. random.rand(3,2000)

for i in range(n\_col):

data\_shuffled[:,i] = data[:,shuffle\_seq[i] ];

test\_input\_data = np.stack([data\_shuffled[0], data\_shuffled[1]], axis=1)

pred\_test\_data\_output = nn.predValue(test\_input\_data)

actual\_test\_data\_output = data\_shuffled[2].reshape(2000,1)

error\_rate = np.square(np.subtract(actual\_test\_data\_output,pred\_test\_data\_output)).mean()

#accuracy = accuracy\_score(pred\_test\_data\_output,actual\_test\_data\_output)

#print("Accuracy :",accuracy)

print("Moon Test Data Error Rate :",error\_rate\*100,"%")

#UCI Dataset

# Read data from file 'filename.csv'

# (in the same directory that your python process is based)

# Control delimiters, rows, column names with read\_csv (see later)

data = pd.read\_csv("cc.csv")

# Preview the first 5 lines of the loaded data

data.head()

inputData = pd.DataFrame(data, columns = ['a7', 'a10']).to\_numpy()

outLabelWithoutBinary = pd.DataFrame(data, columns = ['a15']).to\_numpy()

outLabelWithoutBinary[outLabelWithoutBinary == '+'] = 1

outLabelWithoutBinary[outLabelWithoutBinary == '-'] = 0

uciOutput = nn.predValue(inputData)

#error\_rate = np.square(np.subtract(uciOutput,outLabelWithoutBinary)).mean()

#acc = jaccard\_score(outLabelWithoutBinary, uciOutput)

accuracy = nn.predAccuracy(outLabelWithoutBinary,uciOutput)

#print("Accuracy :",accuracy)

#print("UCI Test Data Error :",error\_rate)

print("UCI Dataset Accuracy :",accuracy, "%")

# Error Rate for 2000 Moon Test Data Points:



# Accuracy Rate for UCI Dataset (Test Data):

