# Code:

# Part 1

# -\*- coding: utf-8 -\*-

"""

Created on Sat Sep 21 18:50:34 2019

@author: 91755

"""

import numpy as np

import pandas as pd

class LeastMeanSquare():

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

np.random.seed(1)

self.weights = np.random.random((2,1))

def activation(self,x):

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

calculate[calculate >= 0.5] = 1

calculate[calculate < 0.5] = -1

return calculate

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

a = [0.1,0.01,0.001]

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

l = (int) (i / 17)

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

error = train\_outputs - pred\_output

ajustmentForWeights = a[l] \* np.dot(train\_inputs.T , error)

self.weights += ajustmentForWeights

def predValue(self,inputs):

np.set\_printoptions(suppress=True) #prevents exponential value

inputs = inputs.astype(float)

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

o = self.activation(output)

# threshold = (np.amax(output) + np.amin(output)) / 2

# output[ output >= threshold ] = 1

# output[ output < threshold ] = -1

return o

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 = -1\*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 = LeastMeanSquare()

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

nn.training\_time(train\_input\_data,outputLabel,50)

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 = -1\*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 using LMS :", error\_rate\*100 ,"%")

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

# Preview the first 5 lines of the loaded data

data.head()

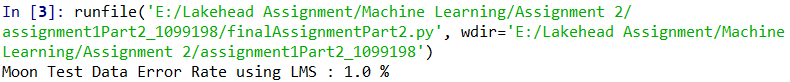
inputData = pd.DataFrame(data, columns = ['a7', 'a10']).to\_numpy()

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

outLabelWithoutBinary[outLabelWithoutBinary == '+'] = 1

outLabelWithoutBinary[outLabelWithoutBinary == '-'] = -1

# Error Rate for 2000 Moon Test Data Points using LMS:



# 

# Part 2:

# -\*- coding: utf-8 -\*-

"""

Created on Sat Sep 21 20:02:04 2019

@author: 91755

"""

import numpy as np

import pandas as pd

import numdifftools as nd

class NewtonMethod():

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

np.random.seed(1)

self.weights = np.random.random((2,1))

def activation(self,x):

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

calculate[calculate >= 0.7] = 1

calculate[calculate < 0.7] = -1

return calculate

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

#a = [0.1,0.01,0.001]

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

#l = (int) (i / 17)

learning\_rate = 0.01

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

error = train\_outputs - pred\_output

mse = np.square(np.subtract(train\_outputs,pred\_output)).mean()

func = lambda x: mse\*x[0]\*\*2 + mse\*x[1]\*\*3 + x[0]\*x[1]

w = self.weights

w1 = w[0][0]

w2 = w[1][0]

w\_array = [w1,w2]

# grad = gradient(w\_array, func, eps=1e-4)

# hess = hessian(w\_array, func, eps=1e-4)

nd\_hess = nd.Hessian(func)

nd\_hess\_vals = nd\_hess(w\_array)

gradient\_matrix = (1/train\_inputs.size) \*(np.dot(train\_inputs.T, error))

hessian\_matrix = nd\_hess\_vals

#gradient\_vector = np.array([] [])

hessian\_inverse = np.linalg.inv(hessian\_matrix)

ajustmentForWeights = learning\_rate \* np.dot(hessian\_inverse,gradient\_matrix)

self.weights = self.weights - ajustmentForWeights

#print(np.amax(pred\_output))

def predValue(self,inputs):

np.set\_printoptions(suppress=True) #prevents exponential value

inputs = inputs.astype(float)

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

o = self.activation(output)

# threshold = (np.amax(output) + np.amin(output)) / 2

# output[ output >= threshold ] = 1

# output[ output < threshold ] = -1

return o

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\_\_":

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 == '-'] = -1

row, col = outLabelWithoutBinary.shape

no\_of\_train\_data = (int) (row/2)

no\_of\_test\_data = row - no\_of\_train\_data

#train data

inputTrainData = inputData[0 : no\_of\_train\_data,:]

outputTrainLabel = outLabelWithoutBinary[0 : no\_of\_train\_data,:]

nn = NewtonMethod()

nn.training\_time(inputTrainData,outputTrainLabel,50)

#test data

inputTestData = inputData[no\_of\_test\_data: row,:]

outputTestLabel = outLabelWithoutBinary[no\_of\_test\_data : row,:]

uciOutput = nn.predValue(inputTestData)

accuracy = nn.predAccuracy(outputTestLabel,uciOutput)

print("UCI Dataset Accuracy using Newton's Method :",accuracy, "%")

# Accuracy Rate for UCI Dataset using Newton’s Method (Test Data):

