# I-ELM Code:

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

"""

Created on Wed Oct 2 13:53:37 2019

@author: 91755

"""

import numpy as np

import pandas as pd

import math

import time as t

b = 0.01

def predValue(inputs,w):

inputs = inputs.astype(float)

output = sig(np.dot(inputs, w) + b)

return output

def sig(x):

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

return calculate

def predValueRegression(inputs,w):

inputs = inputs.astype(float)

output = relu(np.dot(inputs, w) + b)

return output

def relu(x):

calculate = np.maximum(x, 0)

return calculate

def predAccuracy(originalLabel,predicted):

matched = 0

for i in range(len(originalLabel)):

if originalLabel[i] == predicted[i]:

matched += 1

accuracyVal = matched / float(len(originalLabel))

return accuracyVal

#-------------------------------------------training start--------------------------------------------------------

def trainClassification(train\_input\_data,outputLabel):

lMax = 20

E = outputLabel

l = 0

defAccuracy = 0.9

while l < lMax :

l = l + 1

hMatrix = np.zeros(shape = (train\_input\_data.shape[0],l))

for row in train\_input\_data:

k=0

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

h = []

for i in range(l):

weights = np.random.random((train\_input\_data.shape[1],1))

output = predValue(row,weights)

h.append(output)

h = np.reshape(h, (-1, l)) #l is 6 #this will generate 2d matrix of h

hMatrix[k] = h[0]

k = k + 1

beta = np.dot(np.linalg.pinv(hMatrix), outputLabel)

E = E - np.dot(hMatrix,beta)

E = sig(E)

E[E > 0.5] = 1

E[E <= 0.5] = 0

E\_VALC = predAccuracy(outputLabel,E)

if(E\_VALC > defAccuracy) :

print("Training Accuracy: ",E\_VALC\*100,"%")

break

return beta

def trainRegression(train\_input\_data,outputLabel):

outputLabel = sig(outputLabel)

lMax = 20

E = outputLabel

l = 0

defError = 0.1 #ac = 1 - 0.01

while l < lMax :

l = l + 1

hMatrix = np.zeros(shape = (train\_input\_data.shape[0],l))

for row in train\_input\_data:

k=0

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

h = []

for i in range(l):

weights = np.random.random((train\_input\_data.shape[1],1))

output = predValue(row,weights)

h.append(output)

h = np.reshape(h, (-1, l)) #l is 6 #this will generate 2d matrix of h

hMatrix[k] = h[0]

k = k + 1

beta = np.dot(np.linalg.pinv(hMatrix), outputLabel)

E = E - np.dot(hMatrix,beta)

E\_VAL = math.sqrt( np.square(np.subtract(outputLabel,E)).mean() )

if(E\_VAL < defError) :

print("Training RMSE: ",E\_VAL)

break

return beta

def testClassification(data, outputD,b):

la = b.shape[0]

hMatrix = np.zeros(shape = (1000,la))

for row in data:

k=0

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

h = []

for i in range(la):

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

output = predValue(row,weights)

h.append(output)

h = np.reshape(h, (-1, la)) #l is 6 #this will generate 2d matrix of h

hMatrix[k] = h[0]

k = k + 1

o = sig(np.dot(hMatrix , b))

o[o > 0.5] = 1

o[o <= 0.5] = 0

acc = predAccuracy(outputD, o)

print("Testing Accuracy",acc\*100,"%")

def testRegression(data, outputD,b):

outputD =sig(outputD)

la = b.shape[0]

hMatrix = np.zeros(shape = (data.shape[0],la))

for row in data:

k=0

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

h = []

for i in range(la):

weights = np.random.random((data.shape[1],1))

output = predValue(row,weights)

h.append(output)

h = np.reshape(h, (-1, la)) #l is 6 #this will generate 2d matrix of h

hMatrix[k] = h[0]

k = k + 1

o = sig(np.dot(hMatrix , b))

rmse = math.sqrt( np.square(np.subtract(outputD,o)).mean() )

print("Testing RMSE",rmse)

#-------------------------------------------Classification and Regression------------------------------------

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

#-------------------------------------------generate train data, full moon-----------------------------------------------

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\_classification = np.stack([data\_shuffled[0], data\_shuffled[1]], axis=1)

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

# print(outputLabel[0])

# o = np.reshape(outputLabel[0], (1,np.product(outputLabel[0].shape)))

# print(o)

startTimeForTrainingClassification = t.time()

betA = trainClassification(train\_input\_data\_classification,outputLabel\_classification) #second parameter: 1 for classification, 0 for regression

endTimeForTrainingClassification = t.time()

print("Training time for Classification :",abs(startTimeForTrainingClassification-endTimeForTrainingClassification))

#-------------------------------------------training end--------------------------------------------------------

#-------------------------------------------generating test data, full moon----------------------------------------------

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] ];

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

actual\_test\_data\_output = data\_shuffled[2].reshape(1000,1)

startTimeForTestingClassification = t.time()

testClassification(test\_input\_data, actual\_test\_data\_output,betA)

endTimeForTestingClassification = t.time()

print("Testing time for Classification :",abs(startTimeForTestingClassification-endTimeForTestingClassification))

#-------------------------------------------generating test data end----------------------------------------------

#-------------------------------------------generating data regression---------------------------------------

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

# Preview the first 5 lines of the loaded data

data.head()

inputData = pd.DataFrame(data, columns = ['a0', 'a1', 'a2','a3']).to\_numpy()

# inputData = inputData[0:2000,:]

outputData = pd.DataFrame(data, columns = ['a4']).to\_numpy()

# outputData = outputData[0:2000,:]

row, col = outputData.shape

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

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

#----------------------------------------------training regression-----------------------------------------------------

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

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

startTimeForTrainingRegression = t.time()

betaRegression = trainRegression(inputTrainData,outputTrainLabel) #0 for regression

endTimeForTrainingRegression = t.time()

print("Training time for Regression :",abs(startTimeForTrainingRegression-endTimeForTrainingRegression) )

# print(betaRegression)

#---------------------------------------------test Regression-------------------------------------------------

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

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

startTimeForTestingRegression = t.time()

testRegression(inputTestData,outputTestLabel,betaRegression)

endTimeForTestingRegression = t.time()

print("Testing time for Regression :",abs(startTimeForTestingRegression - endTimeForTestingRegression))

# Output:

