**IE 440**

**FINAL**



**Question I**

**ALİ OZAN MEMETOĞLU**

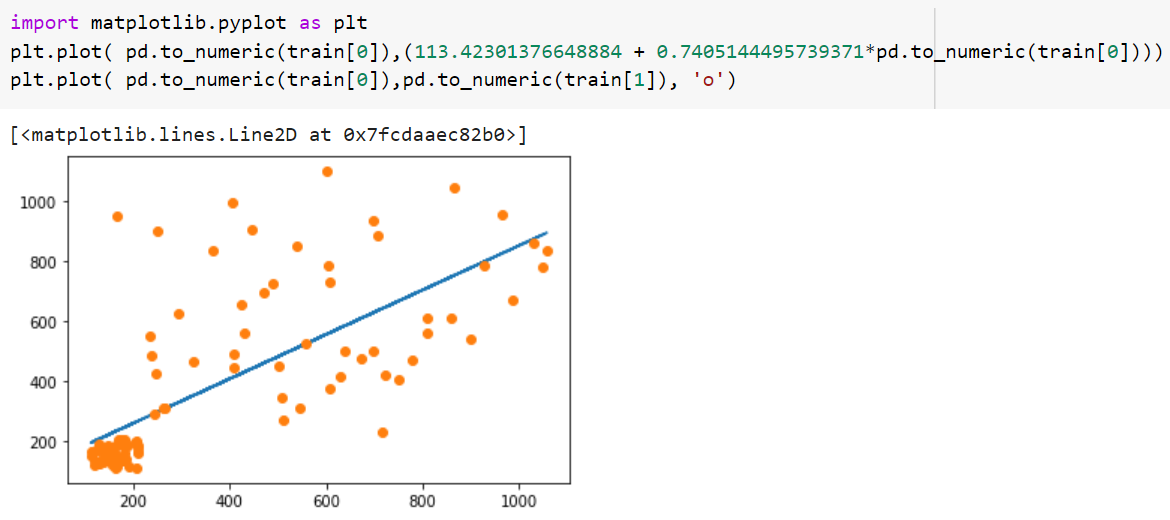
2016402060

**1. Least Square Method:**

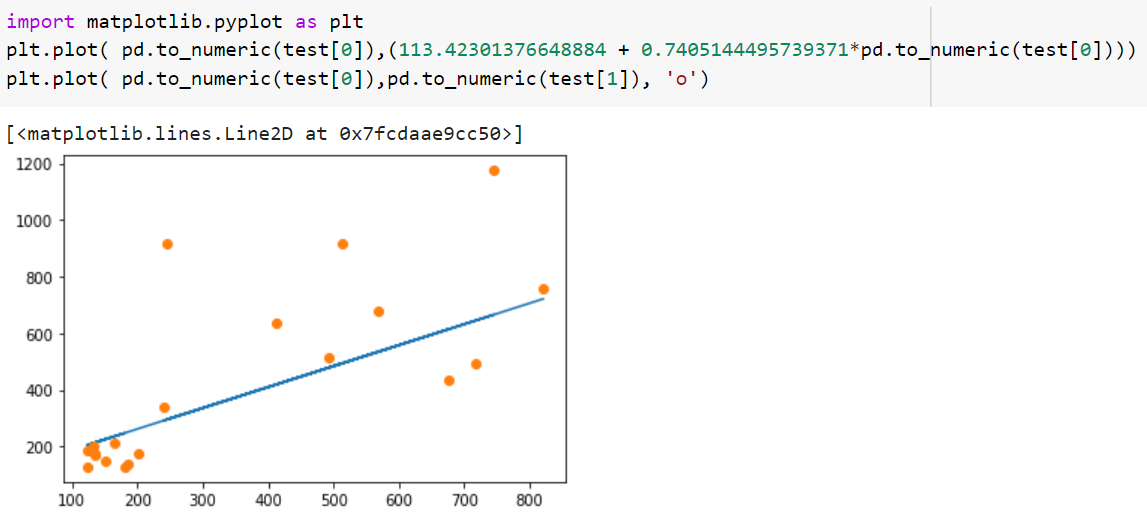
(a)

{W0, W1} = [[113.42343333874152], [0.740512684596673]]

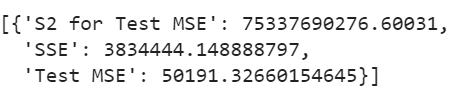
The training data and the regression functions:



The test data and the regression functions:



Output:

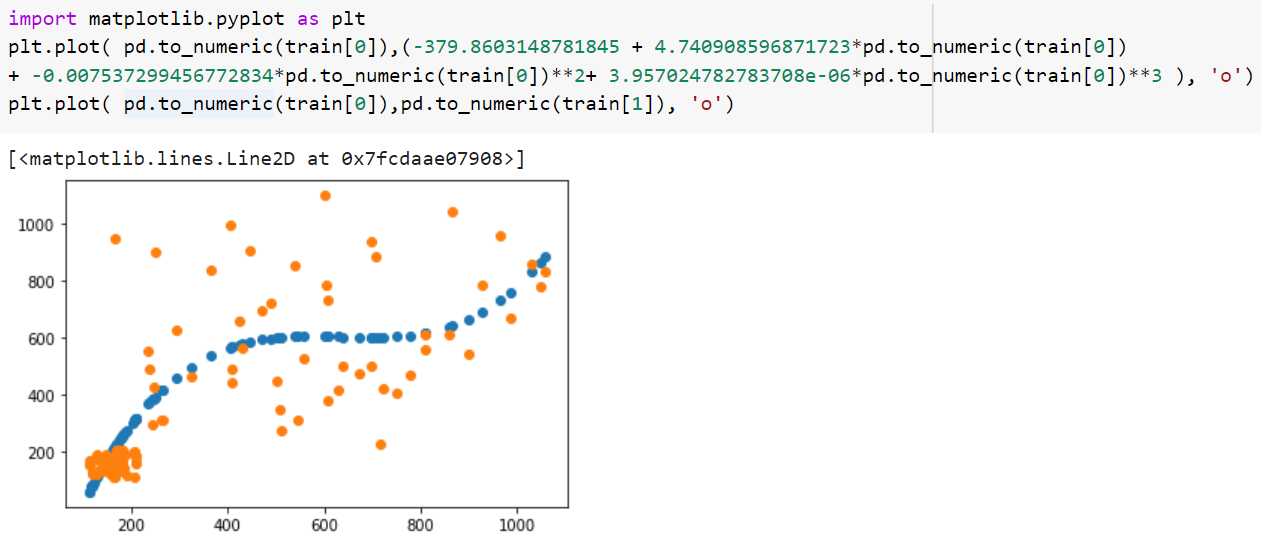


(b)

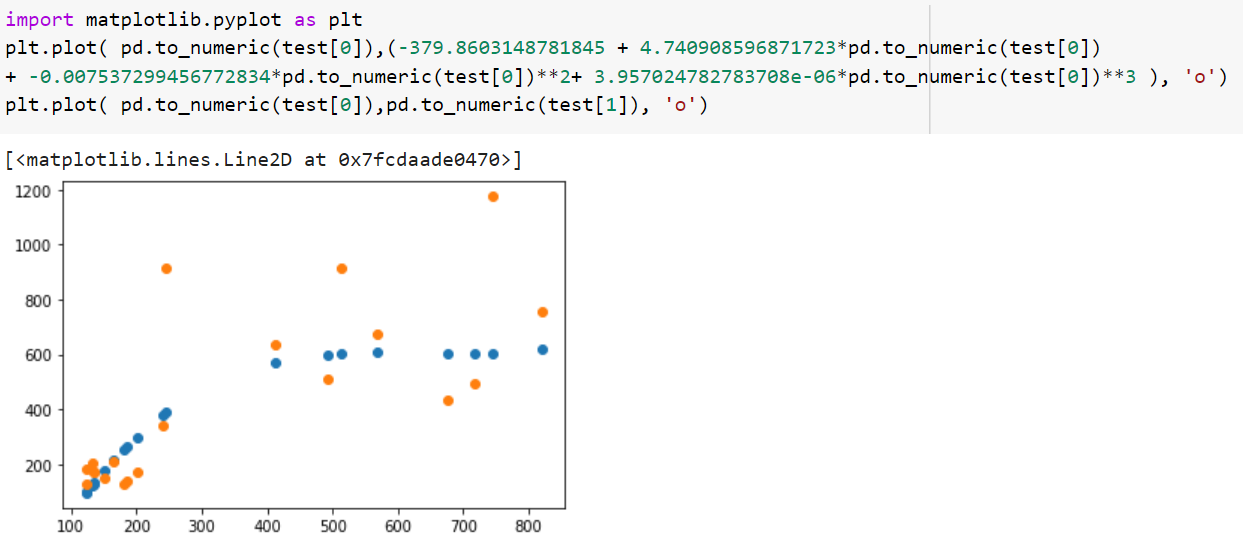
{W0, W1, W2, W3} =

[[-379.8608299111266], [4.740908720082598], [-0.007537300504473179], [3.957026697622502e-06]]

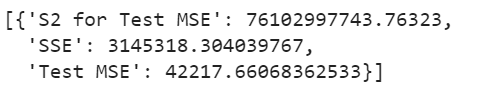
The training data and the regression functions:



The test data and the regression functions:



Output:

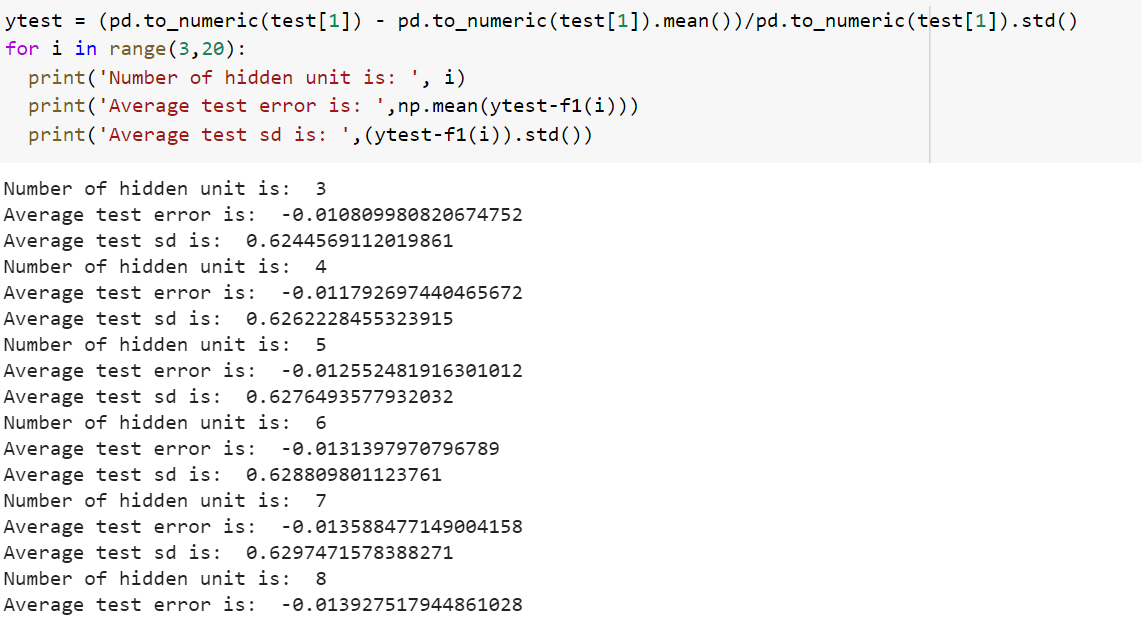


Remarks:

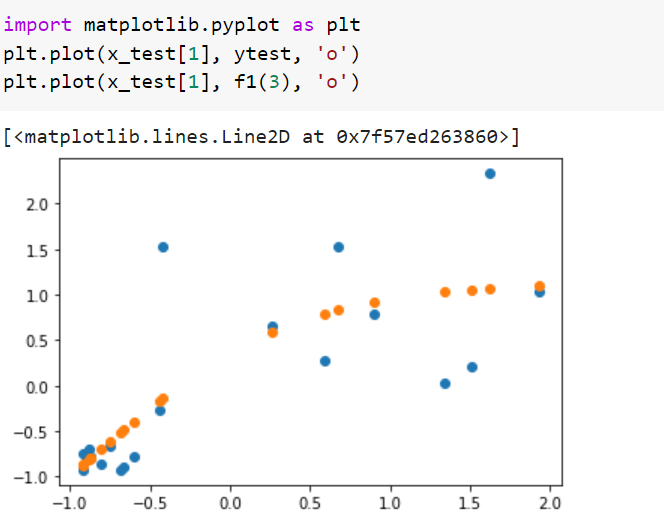
* Although training performance of the linear model is better, a more flexible cubic model has a better performance in these training and test datasets. This result confirms the fact of overfitting. A good model on the training set does not necessarily result in a good performance on the test set. A quadratic model on a scattered dataset like the one we have performs better in that sense.

**2. Nonlinear Regression:**

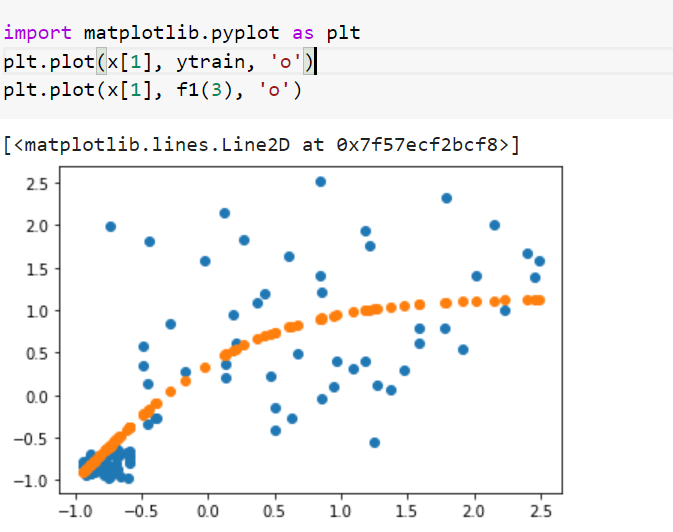
1. Number of hidden units is determined as 3 in the first network since mean error starts to increase after that point.

****

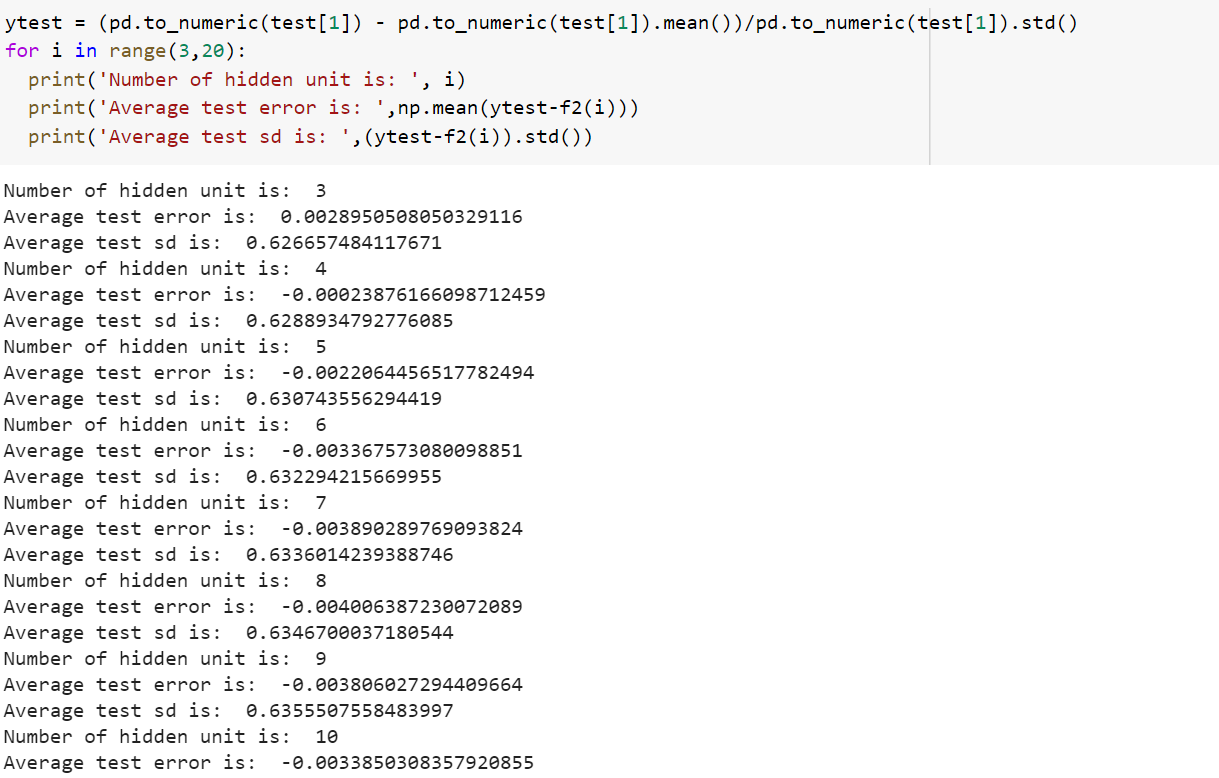
**Test Data Observations - Predictions**

****

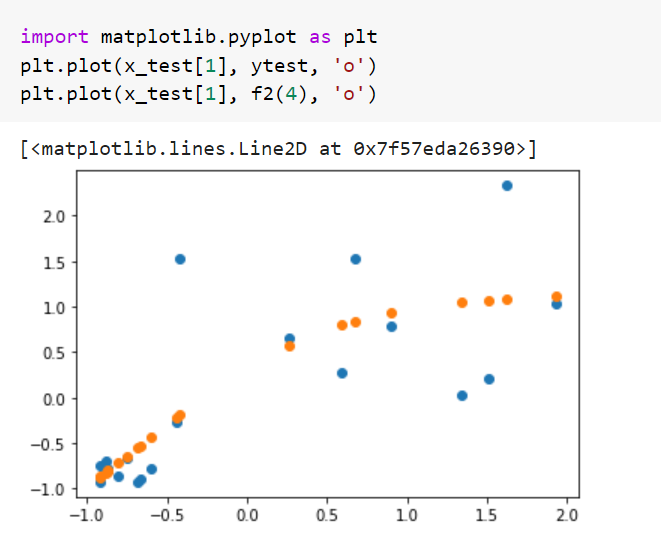
**Train Data Observations - Predictions**

****

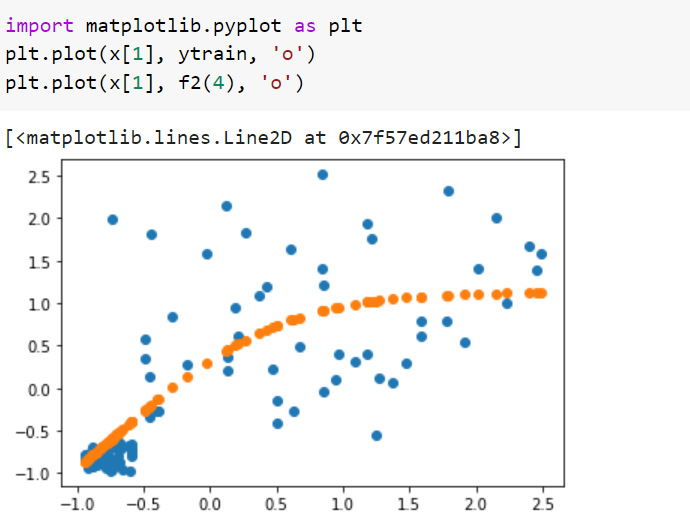
1. Number of hidden units is determined as 4 in the first network since mean error starts to increase after that point.



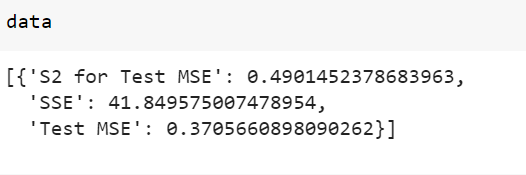
**Test Data Observations - Predictions:**



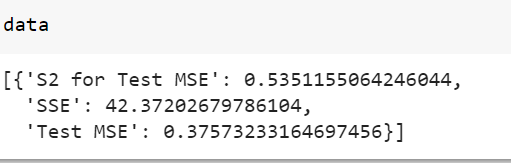
**Train Data Observations - Predictions:**



Normalized data report for f1:



Normalized data report for f2:



Appendix:

# IMPORT REQUIRED LIBRARIES

import numpy as np

from numpy import mean

import pandas as pd

import matplotlib.pyplot as plt

import math

from tqdm import tqdm

import random

from google.colab import files

from google.colab import auth

auth.authenticate\_user()

import gspread

from oauth2client.client import GoogleCredentials

gc = gspread.authorize(GoogleCredentials.get\_application\_default())

worksheet = gc.open('Untitled spreadsheet').get\_worksheet(0)

# get\_all\_values gives a list of rows.

rows = worksheet.get\_all\_values()

train = pd.DataFrame.from\_records(rows)

worksheet = gc.open('Untitled spreadsheet').get\_worksheet(1)

# get\_all\_values gives a list of rows.

rows = worksheet.get\_all\_values()

test = pd.DataFrame.from\_records(rows)

# INPUT FUNCTION

def f1(coord):

x1 = coord[0][0]

x2 = coord[1][0]

a = 0

for p in range(len(train)): #function is in open form

a = a + float(train.loc[p,1])\*\*2-2\*x1\*float(train.loc[p,1])-2\*float(train.loc[p,1])\*x2\*float(train.loc[p,0])+x1\*\*2+2\*x1\*x2\*float(train.loc[p,0])+(x2\*\*2)\*(float(train.loc[p,0])\*\*2)

return a

def f1\_derivative(coord):

x1 = coord[0][0]

x2 = coord[1][0]

g1 = 0

g2 = 0

for p in range(len(train)):

g1 = g1-2\*float(train.loc[p,1])+2\*x1+2\*x2\*float(train.loc[p,0])

g2 = g2-2\*float(train.loc[p,0])\*float(train.loc[p,1])+2\*x1\*float(train.loc[p,0])+2\*x2\*float(train.loc[p,0])\*\*2

return np.array([[g1 , g2]])

def f1\_hessian(coord):

x1 = coord[0][0]

x2 = coord[1][0]

h1 = 0

h2 = 0

h3 = 0

h4 = 0

for p in range(len(train)):

h1 = h1 + 2

h2 = h2 + 2\*float(train.loc[p,0])

h3 = h3 + 2\*float(train.loc[p,0])

h4 = h4 + 2\*float(train.loc[p,0])\*\*2

return np.array([[h1 , h2],

[h3 , h4]])

def f2(coord):

x1 = coord[0][0]

x2 = coord[1][0]

x3 = coord[2][0]

x4 = coord[3][0]

a = 0

for p in range(len(train)):

a = a + (float(train.loc[p,1])-x1-x2\*float(train.loc[p,0])-x3\*float(train.loc[p,0])\*\*2-x4\*float(train.loc[p,0])\*\*3)\*\*2

return a

def f2\_derivative(coord):

x1 = coord[0][0]

x2 = coord[1][0]

x3 = coord[2][0]

x4 = coord[3][0]

g1 = 0

g2 = 0

g3 = 0

g4 = 0

for p in range(len(train)):

g1 = g1-2\*(float(train.loc[p,1])-x1-x2\*float(train.loc[p,0])-x3\*float(train.loc[p,0])\*\*2-x4\*float(train.loc[p,0])\*\*3)

g2 = g2-2\*float(train.loc[p,0])\*(float(train.loc[p,1])-x1-x2\*float(train.loc[p,0])-x3\*float(train.loc[p,0])\*\*2-x4\*float(train.loc[p,0])\*\*3)

g3 = g3-2\*(float(train.loc[p,0])\*\*2)\*(float(train.loc[p,1])-x1-x2\*float(train.loc[p,0])-x3\*float(train.loc[p,0])\*\*2-x4\*float(train.loc[p,0])\*\*3)

g4 = g4-2\*(float(train.loc[p,0])\*\*3)\*(float(train.loc[p,1])-x1-x2\*float(train.loc[p,0])-x3\*float(train.loc[p,0])\*\*2-x4\*float(train.loc[p,0])\*\*3)

return np.array([[g1 , g2, g3, g4]])

def f2\_hessian(coord):

x1 = coord[0][0]

x2 = coord[1][0]

x3 = coord[2][0]

x4 = coord[3][0]

h1 = 0

h2 = 0

h3 = 0

h4 = 0

h5 = 0

h6 = 0

h7 = 0

h8 = 0

h9 = 0

h10 = 0

h11 = 0

h12 = 0

h13 = 0

h14 = 0

h15 = 0

h16 = 0

for p in range(len(train)):

h1 = h1 + 2

h2 = h2 + 2\*float(train.loc[p,0])

h3 = h3 + 2\*float(train.loc[p,0])\*\*2

h4 = h4 + 2\*float(train.loc[p,0])\*\*3

h5 = h5 + 2\*float(train.loc[p,0])

h6 = h6 + 2\*float(train.loc[p,0])\*\*2

h7 = h7 + 2\*float(train.loc[p,0])\*\*3

h8 = h8 + 2\*float(train.loc[p,0])\*\*4

h9 = h9 + 2\*float(train.loc[p,0])\*\*2

h10 = h10 + 2\*float(train.loc[p,0])\*\*3

h11 = h11 + 2\*float(train.loc[p,0])\*\*4

h12 = h12 + 2\*float(train.loc[p,0])\*\*5

h13 = h13 + 2\*float(train.loc[p,0])\*\*3

h14 = h14 + 2\*float(train.loc[p,0])\*\*4

h15 = h15 + 2\*float(train.loc[p,0])\*\*5

h16 = h16 + 2\*float(train.loc[p,0])\*\*6

return np.array([[h1, h2, h3 , h4],

[h5, h6, h7 , h8],

[h9, h10, h11 , h12],

[h13, h14, h15 , h16],])

# BISECTION METHOD

def bisection(a, b, eps, f):

x = 0

x\_values = [0]

if f == 1:

while abs(b-a) >= eps:

old\_x = x

x = (a+b)/2

x\_values.append(x)

if f(x) >= f(x+eps):

a = x

else:

b = x

new\_x = (a+b)/2

else:

while abs(b-a) >= eps:

old\_x = x

x = (a+b)/2

x\_values.append(x)

if f(x) >= f(x+eps):

a = x

else:

b = x

new\_x = (a+b)/2

return {

'x' : x,

'f(x)' : f(x)

}

def bfgs(x0, f, gradient\_f, hessian\_f, epsilon):

k = 0

H = [np.identity(len(x0))]

x = [x0]

data = []

while True:

d = -1 \* H[-1].dot(gradient\_f(x[-1]).T)

if np.linalg.norm(d) < eps or k > 1000:

break

func = lambda alpha : f(x[-1] + alpha \* d)

alpha = bisection(a=-100, b=100, eps=0.005, f=func)['x']

new\_x = x[-1] + alpha \* d

x.append(new\_x)

p = x[-1] - x[-2]

q = gradient\_f(x[-1]).T - gradient\_f(x[-2]).T

A = (p.T.dot(q) + q.T.dot(H[-1]).dot(q)) \* (p.dot(p.T)) / ((p.T.dot(q))\*\*2)

B = -1 \* (H[-1].dot(q).dot(p.T) + p.dot(q.T).dot(H[-1])) / (p.T.dot(q))

new\_H = H[-1] + A + B

H.append(new\_H)

k += 1

data.append({

"k" : k,

"x" : x[-2],

"f(x)" : f(x[-2]),

"d" : d,

"alpha" : alpha,

"x\_new" : x[-1]

})

return pd.DataFrame(data)

pd.options.display.max\_colwidth = 100

x0 = np.array([[np.random.uniform(-0.001, 0.001),np.random.uniform(-0.001, 0.001),np.random.uniform(-0.001, 0.001),np.random.uniform(-0.001, 0.001)]]).T

eps = 1E-5

bfgs(x0=x0, f=f2, gradient\_f=f2\_derivative, hessian\_f=f2\_hessian, epsilon=eps)

x0 = np.array([[np.random.uniform(-0.001, 0.001),np.random.uniform(-0.001, 0.001)]]).T

eps = 1E-5

bfgs(x0=x0, f=f1, gradient\_f=f1\_derivative, hessian\_f=f1\_hessian, epsilon=eps)

import matplotlib.pyplot as plt

plt.plot( pd.to\_numeric(train[0]),(113.42301376648884 + 0.7405144495739371\*pd.to\_numeric(train[0])))

plt.plot( pd.to\_numeric(train[0]),pd.to\_numeric(train[1]), 'o')

import matplotlib.pyplot as plt

plt.plot( pd.to\_numeric(test[0]),(113.42301376648884 + 0.7405144495739371\*pd.to\_numeric(test[0])))

plt.plot( pd.to\_numeric(test[0]),pd.to\_numeric(test[1]), 'o')

data = []

sse = sum((pd.to\_numeric(train[1]) - (113.42301376648884 + 0.7405144495739371\*pd.to\_numeric(train[0])))\*\*2)

mse = mean((pd.to\_numeric(test[1]) - (113.42301376648884 + 0.7405144495739371\*pd.to\_numeric(test[0])))\*\*2)

k = 0

for p in range(len(test)):

k = k + (mse - (float(test.loc[p,1]) - (113.42301376648884 + 0.7405144495739371\*float(test.loc[p,0]))\*\*2))\*\*2

k= k/(len(test)-1)

data.append({

"SSE" : sse,

"Test MSE" : mse,

"S2 for Test MSE" : k

})

data

[[-379.8603148781845], [4.740908596871723], [-0.007537299456772834], [3.957024782783708e-06]]

import matplotlib.pyplot as plt

plt.plot( pd.to\_numeric(train[0]),(-379.8603148781845 + 4.740908596871723\*pd.to\_numeric(train[0])

+ -0.007537299456772834\*pd.to\_numeric(train[0])\*\*2+ 3.957024782783708e-06\*pd.to\_numeric(train[0])\*\*3 ), 'o')

plt.plot( pd.to\_numeric(train[0]),pd.to\_numeric(train[1]), 'o')

import matplotlib.pyplot as plt

plt.plot( pd.to\_numeric(test[0]),(-379.8603148781845 + 4.740908596871723\*pd.to\_numeric(test[0])

+ -0.007537299456772834\*pd.to\_numeric(test[0])\*\*2+ 3.957024782783708e-06\*pd.to\_numeric(test[0])\*\*3 ), 'o')

plt.plot( pd.to\_numeric(test[0]),pd.to\_numeric(test[1]), 'o')

data = []

sse = sum((pd.to\_numeric(train[1]) - (-379.8603148781845 + 4.740908596871723\*pd.to\_numeric(train[0]) + -0.007537299456772834\*pd.to\_numeric(train[0])\*\*2+ 3.957024782783708e-06\*pd.to\_numeric(train[0])\*\*3 ))\*\*2)

mse = mean((pd.to\_numeric(test[1]) - (-379.8603148781845 + 4.740908596871723\*pd.to\_numeric(test[0]) + -0.007537299456772834\*pd.to\_numeric(test[0])\*\*2+ 3.957024782783708e-06\*pd.to\_numeric(test[0])\*\*3 ))\*\*2)

k = 0

for p in range(len(test)):

k = k + (mse - (float(test.loc[p,1]) - (-379.8603148781845 + 4.740908596871723\*float(test.loc[p,0]) + -0.007537299456772834\*float(test.loc[p,0])\*\*2+ 3.957024782783708e-06\*float(test.loc[p,0])\*\*3)\*\*2))\*\*2

k= k/(len(test)-1)

data.append({

"SSE" : sse,

"Test MSE" : mse,

"S2 for Test MSE" : k

})

data

# SECOND QUESTION

def sig(z):

if z<0:

return np.exp(z)/(1.0+np.exp(z))

else:

return 1.0/(1.0 + np.exp(-z))

def sig\_dif(x):

a = sig(x)\*(1.0-sig(x))

return a

def f1\_test(i): #returns f1 test preds

x = pd.DataFrame({ 0 : np.repeat(1.0,len(train[0])), 1 : ((pd.to\_numeric(train[0])-pd.to\_numeric(train[0]).mean())/pd.to\_numeric(train[0]).std())})

#train[1] = ((pd.to\_numeric(train[1])-pd.to\_numeric(train[1]).mean())/pd.to\_numeric(train[1]).std())

x\_test = pd.DataFrame({ 0 : np.repeat(1.0,len(test[0])), 1 : ((pd.to\_numeric(test[0])-pd.to\_numeric(test[0]).mean())/pd.to\_numeric(test[0]).std())})

ytrain = (pd.to\_numeric(train[1]) - pd.to\_numeric(train[1]).mean())/pd.to\_numeric(train[1]).std()

# x = pd.DataFrame({ 0 : np.repeat(1.0,len(train[0])), 1 :pd.to\_numeric(train[0]), 2 :pd.to\_numeric(train[0])\*\*2, 3 :pd.to\_numeric(train[0])\*\*3})

#x\_test = pd.DataFrame({ 0 : np.repeat(1.0,len(test[0])), 1 :pd.to\_numeric(test[0]), 2 :pd.to\_numeric(test[0])\*\*2, 3 :pd.to\_numeric(test[0])\*\*3})

P = len(train)

t = 0

epsilon= 0.001

n = 0.9

alpha = 0.5

J = i

K = 1

wjk = np.random.uniform(-0.001, 0.001,size=(J,K+1))

#np.random.uniform(-0.001, 0.001,size=(J,K+1))

Wij = np.random.uniform(-0.001, 0.001,size=(1,J+1))

#np.random.uniform(-0.001, 0.001,size=(1,J+1))

hjp = np.zeros(shape=(J,P))

Hjp = np.zeros(shape=(J,P))

Hjp = np.insert(Hjp, 0, np.repeat(1.0,P), axis=0)

op = np.zeros(shape = (1,P))

d\_ip = np.zeros(shape = (1,P))

d\_jp = np.zeros(shape=(J,P))

delta\_Wij = np.zeros(shape=(1,J+1))

delta\_wjk = np.zeros(shape=(J,K+1))

while alpha > epsilon:

#x = x.sample(frac=1).reset\_index(drop=True)

for p in range(P):

for j in range(J):

hjp[j,p] = np.dot(x.loc[p],wjk[j].transpose())

Hjp[j+1,p] = sig(hjp[j,p])

op[0,p] = np.dot(Wij[0],Hjp.transpose()[p])

d\_ip[0,p] = (ytrain[p])-op[0,p] #

for j in range(J):

d\_jp[j,p] = sig\_dif(hjp[j,p])\*(d\_ip[0,p]\*Wij[0,j+1]) #

for j in range(J+1):

#delta\_Wij[0,j] = alpha \* float(d\_ip[0,p]\*Hjp[j,p])

Wij[0,j] = Wij[0,j] + alpha \* d\_ip[0,p]\*Hjp[j,p]

for j in range(J):

for k in range(K+1):

#delta\_wjk[j,k] = alpha \* float(x.loc[p,k]\*d\_jp[j,p])

wjk[j,k] = wjk[j,k] + alpha \* x.loc[p,k]\*d\_jp[j,p]

alpha = alpha \* n

t = t + 1

unit = np.zeros(shape=(J,P))

a = []

P\_new = len(x\_test)

for p in range(P\_new):

for j in range(J):

unit[j,p]= unit[j,p] + sig(np.dot(x\_test.loc[p],wjk[j]))

for p in range(P\_new):

a.append(float(np.dot(np.insert(unit[:,p], 0, 1.0),Wij[0].transpose())))

return a

ytest = (pd.to\_numeric(test[1]) - pd.to\_numeric(test[1]).mean())/pd.to\_numeric(test[1]).std()

for i in range(3,20):

print('Number of hidden unit is: ', i)

print('Average test error is: ',np.mean(ytest-f1(i)))

print('Average test sd is: ',(ytest-f1(i)).std())

x\_test = pd.DataFrame({ 0 : np.repeat(1.0,len(test[0])), 1 : ((pd.to\_numeric(test[0])-pd.to\_numeric(test[0]).mean())/pd.to\_numeric(test[0]).std())})

import matplotlib.pyplot as plt

plt.plot(x\_test[1], ytest, 'o')

plt.plot(x\_test[1], f1(3), 'o')

# Online learning no index for output unit since there is only one

def f1\_train(i):

x = pd.DataFrame({ 0 : np.repeat(1.0,len(train[0])), 1 : ((pd.to\_numeric(train[0])-pd.to\_numeric(train[0]).mean())/pd.to\_numeric(train[0]).std())})

#train[1] = ((pd.to\_numeric(train[1])-pd.to\_numeric(train[1]).mean())/pd.to\_numeric(train[1]).std())

x\_test = pd.DataFrame({ 0 : np.repeat(1.0,len(test[0])), 1 : ((pd.to\_numeric(test[0])-pd.to\_numeric(test[0]).mean())/pd.to\_numeric(test[0]).std())})

ytrain = (pd.to\_numeric(train[1]) - pd.to\_numeric(train[1]).mean())/pd.to\_numeric(train[1]).std()

# x = pd.DataFrame({ 0 : np.repeat(1.0,len(train[0])), 1 :pd.to\_numeric(train[0]), 2 :pd.to\_numeric(train[0])\*\*2, 3 :pd.to\_numeric(train[0])\*\*3})

#x\_test = pd.DataFrame({ 0 : np.repeat(1.0,len(test[0])), 1 :pd.to\_numeric(test[0]), 2 :pd.to\_numeric(test[0])\*\*2, 3 :pd.to\_numeric(test[0])\*\*3})

P = len(train)

t = 0

epsilon= 0.001

n = 0.9

alpha = 0.5

J = i

K = 1

wjk = np.random.uniform(-0.001, 0.001,size=(J,K+1))

#np.random.uniform(-0.001, 0.001,size=(J,K+1))

Wij = np.random.uniform(-0.001, 0.001,size=(1,J+1))

#np.random.uniform(-0.001, 0.001,size=(1,J+1))

hjp = np.zeros(shape=(J,P))

Hjp = np.zeros(shape=(J,P))

Hjp = np.insert(Hjp, 0, np.repeat(1.0,P), axis=0)

op = np.zeros(shape = (1,P))

d\_ip = np.zeros(shape = (1,P))

d\_jp = np.zeros(shape=(J,P))

delta\_Wij = np.zeros(shape=(1,J+1))

delta\_wjk = np.zeros(shape=(J,K+1))

while alpha > epsilon:

#x = x.sample(frac=1).reset\_index(drop=True)

for p in range(P):

for j in range(J):

hjp[j,p] = np.dot(x.loc[p],wjk[j].transpose())

Hjp[j+1,p] = sig(hjp[j,p])

op[0,p] = np.dot(Wij[0],Hjp.transpose()[p])

d\_ip[0,p] = (ytrain[p])-op[0,p] #

for j in range(J):

d\_jp[j,p] = sig\_dif(hjp[j,p])\*(d\_ip[0,p]\*Wij[0,j+1]) #

for j in range(J+1):

#delta\_Wij[0,j] = alpha \* float(d\_ip[0,p]\*Hjp[j,p])

Wij[0,j] = Wij[0,j] + alpha \* d\_ip[0,p]\*Hjp[j,p]

for j in range(J):

for k in range(K+1):

#delta\_wjk[j,k] = alpha \* float(x.loc[p,k]\*d\_jp[j,p])

wjk[j,k] = wjk[j,k] + alpha \* x.loc[p,k]\*d\_jp[j,p]

alpha = alpha \* n

t = t + 1

#FITTING

unit = np.zeros(shape=(J,P))

a = []

P\_new = len(x)

for p in range(P\_new):

for j in range(J):

unit[j,p]= unit[j,p] + sig(np.dot(x.loc[p],wjk[j]))

for p in range(P\_new):

a.append(float(np.dot(np.insert(unit[:,p], 0, 1.0),Wij[0].transpose())))

return a

ytest = (pd.to\_numeric(test[1]) - pd.to\_numeric(test[1]).mean())/pd.to\_numeric(test[1]).std()

for i in range(3,7):

print('Number of hidden unit is: ', i)

print('Average test error is: ',np.mean(ytest-f2(i)))

print('Average test sd is: ',(ytest-f2(i)).std())

x\_test = pd.DataFrame({ 0 : np.repeat(1.0,len(test[0])), 1 : ((pd.to\_numeric(test[0])-pd.to\_numeric(test[0]).mean())/pd.to\_numeric(test[0]).std()),

2 : ((pd.to\_numeric(test[0])\*\*2-(pd.to\_numeric(test[0])\*\*2).mean())/(pd.to\_numeric(test[0])\*\*2).std()), 3 :((pd.to\_numeric(test[0])\*\*3-(pd.to\_numeric(test[0])\*\*3).mean())/(pd.to\_numeric(test[0])\*\*3).std())})

import matplotlib.pyplot as plt

plt.plot(x\_test[1], ytest, 'o')

plt.plot(x\_test[1], f2(4), 'o')

# Online learning no index for output unit since there is only one

def f2\_test(i): # Returns test pred

x = pd.DataFrame({ 0 : np.repeat(1.0,len(train[0])), 1 : ((pd.to\_numeric(train[0])-pd.to\_numeric(train[0]).mean())/pd.to\_numeric(train[0]).std()),

2 : ((pd.to\_numeric(train[0])\*\*2-(pd.to\_numeric(train[0])\*\*2).mean())/(pd.to\_numeric(train[0])\*\*2).std()), 3 :((pd.to\_numeric(train[0])\*\*3-(pd.to\_numeric(train[0])\*\*3).mean())/(pd.to\_numeric(train[0])\*\*3).std())})

#train[1] = ((pd.to\_numeric(train[1])-pd.to\_numeric(train[1]).mean())/pd.to\_numeric(train[1]).std())

x\_test = pd.DataFrame({ 0 : np.repeat(1.0,len(test[0])), 1 : ((pd.to\_numeric(test[0])-pd.to\_numeric(test[0]).mean())/pd.to\_numeric(test[0]).std()),

2 : ((pd.to\_numeric(test[0])\*\*2-(pd.to\_numeric(test[0])\*\*2).mean())/(pd.to\_numeric(test[0])\*\*2).std()), 3 :((pd.to\_numeric(test[0])\*\*3-(pd.to\_numeric(test[0])\*\*3).mean())/(pd.to\_numeric(test[0])\*\*3).std())})

ytrain = (pd.to\_numeric(train[1]) - pd.to\_numeric(train[1]).mean())/pd.to\_numeric(train[1]).std()

# x = pd.DataFrame({ 0 : np.repeat(1.0,len(train[0])), 1 :pd.to\_numeric(train[0]), 2 :pd.to\_numeric(train[0])\*\*2, 3 :pd.to\_numeric(train[0])\*\*3})

#x\_test = pd.DataFrame({ 0 : np.repeat(1.0,len(test[0])), 1 :pd.to\_numeric(test[0]), 2 :pd.to\_numeric(test[0])\*\*2, 3 :pd.to\_numeric(test[0])\*\*3})

P = len(train)

t = 0

epsilon= 0.001

n = 0.9

alpha = 0.5

J = i

K = 3

wjk = np.random.uniform(-0.001, 0.001,size=(J,K+1))

#np.random.uniform(-0.001, 0.001,size=(J,K+1))

Wij = np.random.uniform(-0.001, 0.001,size=(1,J+1))

#np.random.uniform(-0.001, 0.001,size=(1,J+1))

hjp = np.zeros(shape=(J,P))

Hjp = np.zeros(shape=(J,P))

Hjp = np.insert(Hjp, 0, np.repeat(1.0,P), axis=0)

op = np.zeros(shape = (1,P))

d\_ip = np.zeros(shape = (1,P))

d\_jp = np.zeros(shape=(J,P))

delta\_Wij = np.zeros(shape=(1,J+1))

delta\_wjk = np.zeros(shape=(J,K+1))

while alpha > epsilon:

#x = x.sample(frac=1).reset\_index(drop=True)

for p in range(P):

for j in range(J):

hjp[j,p] = np.dot(x.loc[p],wjk[j].transpose())

Hjp[j+1,p] = sig(hjp[j,p])

op[0,p] = np.dot(Wij[0],Hjp.transpose()[p])

d\_ip[0,p] = (ytrain[p])-op[0,p] #

for j in range(J):

d\_jp[j,p] = sig\_dif(hjp[j,p])\*(d\_ip[0,p]\*Wij[0,j+1]) #

for j in range(J+1):

#delta\_Wij[0,j] = alpha \* float(d\_ip[0,p]\*Hjp[j,p])

Wij[0,j] = Wij[0,j] + alpha \* d\_ip[0,p]\*Hjp[j,p]

for j in range(J):

for k in range(K+1):

#delta\_wjk[j,k] = alpha \* float(x.loc[p,k]\*d\_jp[j,p])

wjk[j,k] = wjk[j,k] + alpha \* x.loc[p,k]\*d\_jp[j,p]

alpha = alpha \* n

t = t + 1

#FITTING

unit = np.zeros(shape=(J,P))

a = []

P\_new = len(x\_test)

for p in range(P\_new):

for j in range(J):

unit[j,p]= unit[j,p] + sig(np.dot(x\_test.loc[p],wjk[j]))

for p in range(P\_new):

a.append(float(np.dot(np.insert(unit[:,p], 0, 1.0),Wij[0].transpose())))

return a

# TRAINING PLOTS

x = pd.DataFrame({ 0 : np.repeat(1.0,len(train[0])), 1 : ((pd.to\_numeric(train[0])-pd.to\_numeric(train[0]).mean())/pd.to\_numeric(train[0]).std()),

2 : ((pd.to\_numeric(train[0])\*\*2-(pd.to\_numeric(train[0])\*\*2).mean())/(pd.to\_numeric(train[0])\*\*2).std()), 3 :((pd.to\_numeric(train[0])\*\*3-(pd.to\_numeric(train[0])\*\*3).mean())/(pd.to\_numeric(train[0])\*\*3).std())})

ytrain = (pd.to\_numeric(train[1]) - pd.to\_numeric(train[1]).mean())/pd.to\_numeric(train[1]).std()

import matplotlib.pyplot as plt

plt.plot(x[1], ytrain, 'o')

plt.plot(x[1], f2(4), 'o')

def f2\_train(i): # Returns train pred

x = pd.DataFrame({ 0 : np.repeat(1.0,len(train[0])), 1 : ((pd.to\_numeric(train[0])-pd.to\_numeric(train[0]).mean())/pd.to\_numeric(train[0]).std()),

2 : ((pd.to\_numeric(train[0])\*\*2-(pd.to\_numeric(train[0])\*\*2).mean())/(pd.to\_numeric(train[0])\*\*2).std()), 3 :((pd.to\_numeric(train[0])\*\*3-(pd.to\_numeric(train[0])\*\*3).mean())/(pd.to\_numeric(train[0])\*\*3).std())})

#train[1] = ((pd.to\_numeric(train[1])-pd.to\_numeric(train[1]).mean())/pd.to\_numeric(train[1]).std())

x\_test = pd.DataFrame({ 0 : np.repeat(1.0,len(test[0])), 1 : ((pd.to\_numeric(test[0])-pd.to\_numeric(test[0]).mean())/pd.to\_numeric(test[0]).std())})

ytrain = (pd.to\_numeric(train[1]) - pd.to\_numeric(train[1]).mean())/pd.to\_numeric(train[1]).std()

# x = pd.DataFrame({ 0 : np.repeat(1.0,len(train[0])), 1 :pd.to\_numeric(train[0]), 2 :pd.to\_numeric(train[0])\*\*2, 3 :pd.to\_numeric(train[0])\*\*3})

#x\_test = pd.DataFrame({ 0 : np.repeat(1.0,len(test[0])), 1 :pd.to\_numeric(test[0]), 2 :pd.to\_numeric(test[0])\*\*2, 3 :pd.to\_numeric(test[0])\*\*3})

P = len(train)

t = 0

epsilon= 0.001

n = 0.9

alpha = 0.5

J = i

K = 3

wjk = np.random.uniform(-0.001, 0.001,size=(J,K+1))

#np.random.uniform(-0.001, 0.001,size=(J,K+1))

Wij = np.random.uniform(-0.001, 0.001,size=(1,J+1))

#np.random.uniform(-0.001, 0.001,size=(1,J+1))

hjp = np.zeros(shape=(J,P))

Hjp = np.zeros(shape=(J,P))

Hjp = np.insert(Hjp, 0, np.repeat(1.0,P), axis=0)

op = np.zeros(shape = (1,P))

d\_ip = np.zeros(shape = (1,P))

d\_jp = np.zeros(shape=(J,P))

delta\_Wij = np.zeros(shape=(1,J+1))

delta\_wjk = np.zeros(shape=(J,K+1))

while alpha > epsilon:

#x = x.sample(frac=1).reset\_index(drop=True)

for p in range(P):

for j in range(J):

hjp[j,p] = np.dot(x.loc[p],wjk[j].transpose())

Hjp[j+1,p] = sig(hjp[j,p])

op[0,p] = np.dot(Wij[0],Hjp.transpose()[p])

d\_ip[0,p] = (ytrain[p])-op[0,p] #

for j in range(J):

d\_jp[j,p] = sig\_dif(hjp[j,p])\*(d\_ip[0,p]\*Wij[0,j+1]) #

for j in range(J+1):

#delta\_Wij[0,j] = alpha \* float(d\_ip[0,p]\*Hjp[j,p])

Wij[0,j] = Wij[0,j] + alpha \* d\_ip[0,p]\*Hjp[j,p]

for j in range(J):

for k in range(K+1):

#delta\_wjk[j,k] = alpha \* float(x.loc[p,k]\*d\_jp[j,p])

wjk[j,k] = wjk[j,k] + alpha \* x.loc[p,k]\*d\_jp[j,p]

alpha = alpha \* n

t = t + 1

unit = np.zeros(shape=(J,P))

a = []

P\_new = len(x)

for p in range(P\_new):

for j in range(J):

unit[j,p]= unit[j,p] + sig(np.dot(x.loc[p],wjk[j]))

for p in range(P\_new):

a.append(float(np.dot(np.insert(unit[:,p], 0, 1.0),Wij[0].transpose())))

return a

def f2\_train(i): # Returns train pred

x = pd.DataFrame({ 0 : np.repeat(1.0,len(train[0])), 1 : ((pd.to\_numeric(train[0])-pd.to\_numeric(train[0]).mean())/pd.to\_numeric(train[0]).std()),

2 : ((pd.to\_numeric(train[0])\*\*2-(pd.to\_numeric(train[0])\*\*2).mean())/(pd.to\_numeric(train[0])\*\*2).std()), 3 :((pd.to\_numeric(train[0])\*\*3-(pd.to\_numeric(train[0])\*\*3).mean())/(pd.to\_numeric(train[0])\*\*3).std())})

#train[1] = ((pd.to\_numeric(train[1])-pd.to\_numeric(train[1]).mean())/pd.to\_numeric(train[1]).std())

x\_test = pd.DataFrame({ 0 : np.repeat(1.0,len(test[0])), 1 : ((pd.to\_numeric(test[0])-pd.to\_numeric(test[0]).mean())/pd.to\_numeric(test[0]).std())})

ytrain = (pd.to\_numeric(train[1]) - pd.to\_numeric(train[1]).mean())/pd.to\_numeric(train[1]).std()

# x = pd.DataFrame({ 0 : np.repeat(1.0,len(train[0])), 1 :pd.to\_numeric(train[0]), 2 :pd.to\_numeric(train[0])\*\*2, 3 :pd.to\_numeric(train[0])\*\*3})

#x\_test = pd.DataFrame({ 0 : np.repeat(1.0,len(test[0])), 1 :pd.to\_numeric(test[0]), 2 :pd.to\_numeric(test[0])\*\*2, 3 :pd.to\_numeric(test[0])\*\*3})

P = len(train)

t = 0

epsilon= 0.001

n = 0.9

alpha = 0.5

J = i

K = 3

wjk = np.random.uniform(-0.001, 0.001,size=(J,K+1))

#np.random.uniform(-0.001, 0.001,size=(J,K+1))

Wij = np.random.uniform(-0.001, 0.001,size=(1,J+1))

#np.random.uniform(-0.001, 0.001,size=(1,J+1))

hjp = np.zeros(shape=(J,P))

Hjp = np.zeros(shape=(J,P))

Hjp = np.insert(Hjp, 0, np.repeat(1.0,P), axis=0)

op = np.zeros(shape = (1,P))

d\_ip = np.zeros(shape = (1,P))

d\_jp = np.zeros(shape=(J,P))

delta\_Wij = np.zeros(shape=(1,J+1))

delta\_wjk = np.zeros(shape=(J,K+1))

while alpha > epsilon:

#x = x.sample(frac=1).reset\_index(drop=True)

for p in range(P):

for j in range(J):

hjp[j,p] = np.dot(x.loc[p],wjk[j].transpose())

Hjp[j+1,p] = sig(hjp[j,p])

op[0,p] = np.dot(Wij[0],Hjp.transpose()[p])

d\_ip[0,p] = (ytrain[p])-op[0,p] #

for j in range(J):

d\_jp[j,p] = sig\_dif(hjp[j,p])\*(d\_ip[0,p]\*Wij[0,j+1]) #

for j in range(J+1):

#delta\_Wij[0,j] = alpha \* float(d\_ip[0,p]\*Hjp[j,p])

Wij[0,j] = Wij[0,j] + alpha \* d\_ip[0,p]\*Hjp[j,p]

for j in range(J):

for k in range(K+1):

#delta\_wjk[j,k] = alpha \* float(x.loc[p,k]\*d\_jp[j,p])

wjk[j,k] = wjk[j,k] + alpha \* x.loc[p,k]\*d\_jp[j,p]

alpha = alpha \* n

t = t + 1

unit = np.zeros(shape=(J,P))

a = []

P\_new = len(x)

for p in range(P\_new):

for j in range(J):

unit[j,p]= unit[j,p] + sig(np.dot(x.loc[p],wjk[j]))

for p in range(P\_new):

a.append(float(np.dot(np.insert(unit[:,p], 0, 1.0),Wij[0].transpose())))

return a

# TRAINING PLOTS

x = pd.DataFrame({ 0 : np.repeat(1.0,len(train[0])), 1 : ((pd.to\_numeric(train[0])-pd.to\_numeric(train[0]).mean())/pd.to\_numeric(train[0]).std())})

ytrain = (pd.to\_numeric(train[1]) - pd.to\_numeric(train[1]).mean())/pd.to\_numeric(train[1]).std()

import matplotlib.pyplot as plt

plt.plot(x[1], ytrain, 'o')

plt.plot(x[1], f1(3), 'o')

f1\_test(3)

ytest[1]

ytrain = (pd.to\_numeric(train[1])-pd.to\_numeric(train[1]).mean())/pd.to\_numeric(train[1]).std()

ytest = (pd.to\_numeric(test[1])-pd.to\_numeric(test[1]).mean())/pd.to\_numeric(test[1]).std()

data = []

sse = sum((ytrain-f1\_train(3))\*\*2)

mse = mean((ytest - f1\_test(3))\*\*2)

k = 0

for p in range(len(ytest)):

k = k + (mse - (ytest[p] - f1\_test(3)[p])\*\*2)\*\*2

k= k/(len(ytest)-1)

#f1

data.append({

"SSE" : sse,

"Test MSE" : mse,

"S2 for Test MSE" : k

})

data

ytrain = (pd.to\_numeric(train[1])-pd.to\_numeric(train[1]).mean())/pd.to\_numeric(train[1]).std()

ytest = (pd.to\_numeric(test[1])-pd.to\_numeric(test[1]).mean())/pd.to\_numeric(test[1]).std()

data = []

sse = sum((ytrain-f2\_train(4))\*\*2)

mse = mean((ytest - f2\_test(4))\*\*2)

k = 0

for p in range(len(ytest)):

k = k + (mse - (ytest[p] - f2\_test(4)[p])\*\*2)\*\*2

k= k/(len(ytest)-1)

#f1

data.append({

"SSE" : sse,

"Test MSE" : mse,

"S2 for Test MSE" : k

})

data