import pandas as pd

import numpy as np

import math

import statistics

df = pd.read\_csv("/Users/haowu/Desktop/Boston\_University\_Graduate\_Study/github/MET-CS/CS 767/Data/Assignment\_3\_Hitters.csv")

df.dtypes

df.isna().sum()

temp = df['Salary'].median()

df.fillna(temp, inplace = True)

df['Salary'].isna().sum()

df['League'].value\_counts()

df['Division'].value\_counts()

league\_temp = [0] \* len(df['League'])

division\_temp = [0] \* len(df['Division'])

for i in range(len(df['League'])):

if df['League'][i] == 'A':

league\_temp[i] = 1

else:

league\_temp[i] = 0

for i in range(len(df['Division'])):

if df['Division'][i] == 'W':

division\_temp[i] = 1

else:

division\_temp[i] = 0

df['League'] = league\_temp

df['Division'] = division\_temp

def leaky\_relu(x):

return np.maximum(0.05\* x , x)

from sklearn.metrics import mean\_squared\_error, r2\_score

from sklearn.model\_selection import train\_test\_split

y = df['Salary']

x = df.drop(['Salary','Unnamed: 0','NewLeague'],axis = 1)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.2, random\_state=10)

def linear\_fit(x\_train,y\_train):

x\_b = np.hstack([np.ones((len(x\_train),1)),x\_train])

lf\_theta = np.linalg.pinv(np.dot(x\_b.T,x\_b)).dot(x\_b.T).dot(y\_train)

lf\_theta = leaky\_relu(lf\_theta)

return lf\_theta

def predict(a):

x\_b = np.hstack([np.ones((len(x\_train),1)),x\_train])

return np.dot(x\_b,reg)

reg = linear\_fit(x\_train,y\_train)

y\_predict = predict(x\_train)

print("Inter:",reg[0],"\nCoef:",reg[1:],"\nR2score:",r2\_score(y\_train,y\_predict))

import matplotlib.pyplot as plt

def batch\_gradient\_descent(x,y,theta,lr,iters):

x\_norm = (x - x.min()) / (x.max() - x.min()) ## normalization our data

x = x\_norm

cost\_history = [0]\*iters

for i in range(0,iters):

y\_pred = np.dot(x,theta.T)

cost = np.sum((y\_pred - y) \*\* 2)/(2\*len(y))

theta = theta - lr \* (np.dot((y\_pred - y),x)/len(y))

cost\_history[i] = cost

return cost\_history

a =np.arange(0.1,10.1,0.1)

best\_lr = [0]\*100

for i in range(0,100):

theta = np.ones((x\_train.shape[1]))

history = batch\_gradient\_descent(x\_train,y\_train,theta,a[i],1000)

best\_lr[i] = sum(history)/1000

best\_lr.index(min(best\_lr)) \* 0.1

def bgd\_L2(x,y,theta,lr,iters):

x\_norm = (x - x.min()) / (x.max() - x.min()) ## normalization our data

x = x\_norm

cost\_history = [0]\*iters

for i in range(0,iters):

y\_pred = np.dot(x,theta.T)

cost = np.sum((y\_pred - y) \*\* 2) + (0.01\* np.sum(theta.T[1:])\*\*2)/(2\*len(y))

theta = theta - (lr/len(y)) \* (np.dot((y\_pred - y),x) + (0.01\* np.sum(theta.T[1:])\*\*2))

cost\_history[i] = cost

return cost\_history

theta = np.ones((x\_train.shape[1]))

BGD\_History = bgd\_L2(x\_train,y\_train,theta,0.7,1000)

plt.plot(BGD\_History)

plt.ylabel('Mean Square Error')

plt.xlabel('Iterations')

plt.title('BDG Train')

def bgd\_L1(x,y,theta,lr,iters):

x\_norm = (x - x.min()) / (x.max() - x.min()) ## normalization our data

x = x\_norm

cost\_history = [0]\*iters

for i in range(0,iters):

y\_pred = np.dot(x,theta.T)

cost = np.sum((y\_pred - y) \*\* 2) + abs(0.01\* np.sum(theta.T[1:]))/(2\*len(y))

theta = theta - (lr/len(y)) \* (np.dot((y\_pred - y),x) +abs(0.01\* np.sum(theta.T[1:])))

cost\_history[i] = cost

return cost\_history

theta = np.ones((x\_train.shape[1]))

BGD\_History = bgd\_L1(x\_train,y\_train,theta,0.7,1000)

plt.plot(BGD\_History)

plt.ylabel('Mean Square Error')

plt.xlabel('Iterations')

plt.title('BDG Train')

def batch\_gradient\_descent(x,y,theta,lr,iters):

x\_norm = (x - x.min()) / (x.max() - x.min()) ## normalization our data

x = x\_norm

cost\_history = [0]\*iters

for i in range(0,iters):

y\_pred = np.dot(x,theta.T)

cost = np.sum((y\_pred - y) \*\* 2)/(2\*len(y))

theta = theta - lr \* (np.dot((y\_pred - y),x)/len(y))

cost\_history[i] = cost

return theta

def bgd\_L1(x,y,theta,lr,iters):

x\_norm = (x - x.min()) / (x.max() - x.min()) ## normalization our data

x = x\_norm

cost\_history = [0]\*iters

for i in range(0,iters):

y\_pred = np.dot(x,theta.T)

cost = np.sum((y\_pred - y) \*\* 2) + abs(0.01\* np.sum(theta.T[1:]))/(2\*len(y))

theta = theta - (lr/len(y)) \* (np.dot((y\_pred - y),x) +abs(0.01\* np.sum(theta.T[1:])))

cost\_history[i] = cost

return theta

def bgd\_L2(x,y,theta,lr,iters):

x\_norm = (x - x.min()) / (x.max() - x.min()) ## normalization our data

x = x\_norm

cost\_history = [0]\*iters

for i in range(0,iters):

y\_pred = np.dot(x,theta.T)

cost = np.sum((y\_pred - y) \*\* 2) + (0.01\* np.sum(theta.T[1:])\*\*2)/(2\*len(y))

theta = theta - (lr/len(y)) \* (np.dot((y\_pred - y),x) + (0.01\* np.sum(theta.T[1:])\*\*2))

cost\_history[i] = cost

return theta

theta = np.ones((x\_train.shape[1]))

Weight1 = batch\_gradient\_descent(x\_train,y\_train,theta,0.7,1000)

Weight2 = bgd\_L1(x\_train,y\_train,theta,0.7,1000)

Weight3 = bgd\_L2(x\_train,y\_train,theta,0.7,1000)

def bgd\_L1\_2(x,y,theta,lr,iters):

x\_norm = (x - x.min()) / (x.max() - x.min()) ## normalization our data

x = x\_norm

cost\_history = [0]\*iters

for i in range(0,iters):

y\_pred = np.dot(x,theta.T)

cost = np.sum((y\_pred - y) \*\* 2) + abs(10\* np.sum(theta.T[1:]))/(2\*len(y))

theta = theta - (lr/len(y)) \* (np.dot((y\_pred - y),x) +abs(10\* np.sum(theta.T[1:])))

cost\_history[i] = cost

return theta

def bgd\_L2\_2(x,y,theta,lr,iters):

x\_norm = (x - x.min()) / (x.max() - x.min()) ## normalization our data

x = x\_norm

for i in range(0,iters):

y\_pred = np.dot(x,theta.T)

theta = theta - (lr/len(y)) \* (np.dot((y\_pred - y),x) + np.sum(10\*theta.T[1:]))

return theta

Weight4 = bgd\_L1\_2(x\_train,y\_train,theta,0.7,1000)

Weight5 = bgd\_L2\_2(x\_train,y\_train,theta,0.7,1000)

from pandas.core.frame import DataFrame

a = {

"Original": Weight1,

"L1\_0.01": Weight2,

"L2\_0.01": Weight3,

"L1\_10": Weight4,

"L2\_10": Weight5

}

data = DataFrame(a)

data