```
import pandas as pd
import math
import numpy as np
import matplotlib.pyplot as plt


from google.colab import drive, files
# drive.mount("/content/gdrive")
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving data_q2_q3.xlsx to data_q2_q3.xlsx

```
# function for normalising data
def norm(data):
  # norm_data = data
  mean = np.mean(data, axis = 0)
  std = np.std(data, axis = 0)
  # print(mean, std)
  norm_data = (data-mean)/std
  return norm_data


# function for defining the hypothesis
def hypothesis(X, wt):
  hyp = np.dot(X, wt.T)
  return hyp


# defining the cost function
def costfunction(X, y, wt):
  hyp = np.dot(X, wt.T)
  J = (0.5/len(y))*np.sum((hyp - y)**2)
  return J


# function for regularisation
def wt_regularisation(lamb, wt, reg = 1):
  wt_reg = np.zeros(wt.shape)
  if reg == 1:
    costreg = (lamb/2)*np.sum(np.abs(wt))
    wt_reg = (lamb/2)*np.sign(wt)
  elif reg == 2:
    costreg = (lamb/2)*np.sum((wt)**2)
    wt_reg = lamb*wt
    # print(wt_reg)
  return wt_reg


# function for regularisation
def J_regularisation(lamb, wt, reg =1):
  costreg = 0
  if reg == 1:
    costreg = (lamb/2)*np.sum(np.abs(wt))
  elif reg == 2:
```

```
  elif reg == 2:
    costreg = (lamb/2)*np.sum((wt)**2)
  return costreg


def bgd(alpha, lamb, iters, X, y):
  w = np.random.rand(X.shape[1])
  for i in range(iters):
    hyp = np.dot(X, w.T)
    # w = w - (alpha/len(y))*(np.dot(hyp - y, X) - wt_regularisation(lamb, w))
    w = w*(1 - (alpha/len(y))*lamb) - (alpha/len(y))*np.dot(hyp - y, X)
  return w


def sgd(alpha, lamb, iters, X, y):
  w = np.random.rand(X.shape[1])
  for i in range(iters):
    rand_ind = np.random.randint(len(y))
    X_ind = X[rand_ind:rand_ind + 1]
    y_ind = y[rand_ind:rand_ind + 1]
    hyp = np.dot(X_ind, w.T)
    # print(hyp.shape)
    # print(y_ind.shape)
    w = w - (alpha/len(y))*(np.dot(hyp - y_ind, X_ind) - wt_regularisation(lamb, w))
  return w


def mbgd(alpha, lamb, iters, batch_size, X, y):
  w = np.random.rand(X.shape[1])
  for i in range(iters):
    rand_ind = np.random.randint(len(y))
    X_batch = X[rand_ind:rand_ind + batch_size]
    y_batch = y[rand_ind:rand_ind + batch_size]
    hyp = np.dot(X_batch, w.T)
    w = w - (alpha/len(y))*(np.dot(hyp-y_batch, X_batch) - wt_regularisation(lamb, w))
  return w


def error(y_testing, y_predicted):
  k = len(y_testing)
  # print(k)
  mae = (0.5/k)*(np.sum(abs(y_testing - y_predicted)))
  mse = (0.5/k)*np.sum((y_testing - y_predicted)**2)
  y_testingdiff = y_testing - np.mean(y_testing)
  y_predicteddiff = y_predicted - np.mean(y_predicted)
  # cc = np.sum(np.multiply(y_testingdiff, y_predicteddiff))/(np.sqrt(np.sum((y_testingdif
  cc = np.corrcoef(y_predicted, y_testing).mean()
  return print('MAE:', mae, '\nMSE:', mse,'\nCC:', cc)


# extracting the data and separating
data = pd.read_excel("data_q2_q3.xlsx")
data = np.asarray(data)
np.random.shuffle(data)
data = norm(data)


# splitting the data into training, testing and validation
```

```
# splitting the data into training, testing and validation
rows = data.shape[0]
training_data = data[0:int(rows*0.7)]
testing_data = data[int(rows*0.7) : int(rows*0.9)]
validation_data = data[int(rows*0.9) : rows+1]

print(rows, len(training_data), len(testing_data), len(validation_data))
```

```
115 80 23 12
```

```
# splitting into input and output
X_tr = training_data[:, 0:-1]  #input
y_tr = training_data[:,-1]    #output

X_ts = testing_data[:, 0:-1]  #input
y_ts = testing_data[:,-1]    #output

X_val = validation_data[:, 0:-1]  #input
y_val = validation_data[:,-1]    #output

print(X_tr.shape, X_ts.shape, X_val.shape)
print(y_tr.shape, y_ts.shape, y_val.shape)


# defining X for regression model
m = X_tr.shape[0]
one_tr = np.ones([m,1])
X_tr = np.append(one_tr, X_tr, axis = 1)

m = X_val.shape[0]
one_val = np.ones([m,1])
X_val = np.append(one_val, X_val, axis = 1)

m = X_ts.shape[0]
one_ts = np.ones([m,1])
X_ts = np.append(one_ts, X_ts, axis = 1)

print(X_tr.shape, X_ts.shape, X_val.shape)
```

```
(80, 4) (23, 4) (12, 4)
(80,) (23,) (12,)
(80, 5) (23, 5) (12, 5)
```

```
wt_bgd = np.random.rand(X_tr.shape[1])
alpha_vals = np.linspace(0.001,0.1,num=50)
l_vals = np.linspace(0.0001,0.1,num=50)
err_bgd = 1000000
n = 0
for a in alpha_vals:
  for l in l_vals:
    wt = bgd(a, l, 1000, X_tr, y_tr)
    # print(wt)
    temp_err = (1/len(y_val))*np.sum((y_val - np.dot(X_val, wt.T)**2))
    # print(wt)
    # print(mse_err)
```

```
    # print(mse_err)
    if temp_err < err_bgd:
      wt_bgd = wt
      err_bgd = temp_err
      # print(wt_bgd, '\n\n')
    n+=1

print(wt_bgd)
y_pred_bgd = np.dot(X_ts, wt_bgd.T)
# print(y_pred_bgd, '\n\n', y_ts)
error(y_ts, y_pred_bgd) ###mae, mse, cc
```

```
[0.02052465 0.7191028  0.21064487 0.17726646 0.46546875]
MAE: 0.2281819213453144
MSE: 0.16619577646073191
CC: 0.9006619372577962
```

```
wt_sgd = np.random.rand(X_tr.shape[1])
alpha_vals = np.logspace(-4,-2,num=50)
l_vals = np.logspace(-3,0,num=50)
err_sgd = 1000000
n = 0
for a in alpha_vals:
  for l in l_vals:
    wt = sgd(a, l, 1000, X_tr, y_tr)
    # print(wt)
    temp_err = (1/len(y_val))*np.sum((y_val - np.dot(X_val, wt.T)**2))
    # print(wt)
    # print(mse_err)
    if temp_err < err_sgd:
      wt_sgd = wt
      err_sgd = temp_err
      # print(wt_bgd, '\n\n')
    n+=1

# print(wt_sgd)
y_pred_sgd = np.dot(X_ts, wt_sgd.T)
# print(y_pred_sgd, '\n\n', y_ts)
error(y_ts, y_pred_sgd) ###mae, mse, cc
```

```
MAE: 0.6654999860284054
MSE: 1.5241571404861634
CC: 0.675963260805293
```

```
# fine tuning the parameters for bgd
m = X_validation.shape[0]
one_validation = np.ones([m,1])
X_validation = np.append(one_validation, X_validation, axis = 1)
wt_validation = np.random.rand(X_validation.shape[1])
```

```
wt_mbgd = np.random.rand(X_tr.shape[1])
alpha_vals = np.logspace(0.001,0.1,num=50)
l_vals = np.logspace(-3,0,num=50)
err_mbgd = 1000000
n = 0
```

```
n = 0
for a in alpha_vals:
  for l in l_vals:
    wt = mbgd(a, l, 1000, 30, X_tr, y_tr)
    # print(wt)
    temp_err = (1/len(y_val))*np.sum((y_val - np.dot(X_val, wt.T)**2))
    # print(wt)
    # print(mse_err)
    if temp_err < err_mbgd:
      wt_mbgd = wt
      err_mbgd = temp_err
      # print(wt_bgd, '\n\n')
    n+=1

# print(wt_mbgd)
y_pred_mbgd = np.dot(X_ts, wt_mbgd.T)
# print(y_pred_mbgd, '\n\n', y_ts)
error(y_ts, y_pred_mbgd) ###mae, mse, cc
```

```
MAE: 0.2719173883231418
MSE: 0.21983671617985548
CC: 0.8771808883678038
```