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


from google.colab import drive, files
uploaded = files.upload()
```

Choose Files data_q4_q5.xlsx

- **data_q4_q5.xlsx**(application/vnd.openxmlformats-officedocument.spreadsheetml.sheet) - 108274 byt
10/1/2021 - 100% done
Saving data_q4_q5.xlsx to data_q4_q5.xlsx

```
def sigmoid(z):
  z = z.astype(float)
  z_out = 1/(1 + np.exp(-z))
  return z_out


def hypothesis(X, wt):
  hyp = sigmoid(np.dot(X, wt.T))
  return hyp


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


# function for regularisation
def wt_regularisation(lamb, wt, reg):
  wt_reg = np.zeros(wt.shape)
  if reg == 0:
    wt_reg = 0
  elif reg == 1:
    wt_reg = (lamb/2)*np.sign(wt)
  elif reg == 2:
    wt_reg = lamb*wt
  return wt_reg


def bgd(alpha, lamb, iters, X, y, reg):
  w = np.zeros(X.shape[1], dtype=np.longfloat)
  for i in range(iters):
    hyp = hypothesis(X, wt)
    w = w - (alpha/len(y))*(np.dot(hyp - y, X) - wt_regularisation(lamb, w, reg))
    #w = w*(1 - (alpha/len(y))*lamb) - (alpha/len(y))*np.dot(hyp - y, X)
  return w


def mbgd(alpha, lamb, iters, batch_size, X, y, reg):
```

```
··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·=·hypothesis(X_batch,·wt)
····w·=·w·-·(alpha/len(y))*(np.dot(hyp-y_batch,·X_batch)·-·wt_regularisation(lamb,·w,·reg)
··return·w


def·sgd(alpha,·lamb,·iters,·X,·y,·reg):
··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·=·hypothesis(X_ind,·wt)
····#·print(hyp.shape)
····#·print(y_ind.shape)
····w·=·w·-·(alpha/len(y))*(np.dot(hyp·-·y_ind,·X_ind)·-·wt_regularisation(lamb,·w,·reg))
··return·w


#·splitting·the·data·into·folds
def·cross_validation(k,·X,·y):
··#·print(len(X))
··fold_size·=·int(len(X)/5)
··#·print(fold_size)
··X_ts·=·X[k*fold_size:(k+1)*fold_size]
··y_ts·=·y[k*fold_size:(k+1)*fold_size]
··X_tr·=·np.delete(X,·slice(k*fold_size,·(k+1)*fold_size),·axis·=·0)
··y_tr·=·np.delete(y,·slice(k*fold_size,·(k+1)*fold_size),·axis·=·0)
··#·print(len(X_ts),·len(y_ts))
··#·print(len(X_tr),·len(y_tr))
··return·X_ts,·y_ts,·X_tr,·y_tr


def·classification(X_ts,·wt):
··m·=·X_ts.shape[0]
··y_sig·=·hypothesis(X_ts,·wt)
··#·print(y_sig)
··y_pred·=·np.zeros(m)
··for·i·in·range(m):
····if·y_sig[i]>0.5:
·····y_pred[i]·=·1
····elif·y_sig[i]<=0.5:
·····y_pred[i]·=·2
··return·y_pred


def confusion_mat(y_predicted, y_testing):
  a, b, c, d = 0 , 0 , 0 , 0
  for i in range(len(y_testing)):
    if y_testing[i] == 1 :
      if y_predicted[i] == 1 :
        a += 1
```

```
        if y_predicted[i] == 2 :
          b += 1
      if y_testing[i] == 2 :
        if y_predicted[i] == 1 :
          c += 1
        if y_predicted[i] == 2 :
          d += 1
    ··acc·=·(a+d)/(a+b+c+d)
    ··sens·=·(a)/(a+b)
    ··spec·=·(d)/(d+c)
      return sens, spec, acc


    #·extracting·the·data·and·separating
    data·=·pd.read_excel("data_q4_q5.xlsx")
    data·=·np.asarray(data)
    np.random.shuffle(data)


    #·splitting·into·input·and·output
    X·=·data[:,·:-1]··#input
    y·=·data[:,-1]····#output
    for·i·in·range(len(y)):
    ··if·y[i]·==·'M':
    ····y[i]·=·1
    ··elif·y[i]·==·'B':
    ····y[i]·=·2
    #·print(y)

    #·normalizing·X·and·y
    X·=·X.astype(float)
    X·=·norm(X)

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


    alpha_vals = np.linspace(0.001, 1, 20)
    l_vals = np.linspace(0.001, 1, 20)

    wt = np.random.rand(X.shape[1])
    SE = []
    SP = []
    A = []
    for i in range(5):
      X_ts, y_ts, X_tr, y_tr = cross_validation(i, X, y)
      a = 0.0005
      l = 0.0001
      wt = bgd(a, l, 1000, X_tr, y_tr, 0)
      y_pred = classification(X_ts, wt)
      sens, spec, acc = confusion_mat(y_pred, y_ts)
      SE.append(sens)
      SP.append(spec)
      A.append(acc)
```

```
avg_sens = sum(SE)/len(SE)
avg_spec = sum(SP)/len(SP)
avg_acc = sum(A)/len(A)
print('BGD LOR without regularisation')
print(f"Sensitivity: {avg_sens}\nSpecificity: {avg_spec}\nAccuracy: {avg_acc}\n")
```

```
BGD LOR without regularisation
Sensitivity: 0.875010769275548
Specificity: 0.7024460713553828
Accuracy: 0.7646017699115044
```

```
wt·=·np.random.rand(X.shape[1])
SE·=·[]
SP·=·[]
A·=·[]
for·i·in·range(5):
··X_ts,·y_ts,·X_tr,·y_tr·=·cross_validation(i,·X,·y)
··a·=·0.0005
··l·=·0.0001
··wt·=·bgd(a,·l,·1000,·X_tr,·y_tr,·1)
··y_pred·=·classification(X_ts,·wt)
··sens,·spec,·acc·=·confusion_mat(y_pred,·y_ts)
··SE.append(sens)
··SP.append(spec)
··A.append(acc)

avg_sens·=·sum(SE)/len(SE)
avg_spec·=·sum(SP)/len(SP)
avg_acc·=·sum(A)/len(A)
print('BGD·LOR·with·L1·regularisation')
print(f"Sensitivity:·{avg_sens}\nSpecificity:·{avg_spec}\nAccuracy:·{avg_acc}\n")
```

```
BGD LOR with L1 regularisation
Sensitivity: 0.9164234362720842
Specificity: 0.7110611427760725
Accuracy: 0.7893805309734514
```

```
wt·=·np.random.rand(X.shape[1])
SE·=·[]
SP·=·[]
A·=·[]
for·i·in·range(5):
··X_ts,·y_ts,·X_tr,·y_tr·=·cross_validation(i,·X,·y)
··a·=·0.0005
··l·=·0.0001
··wt·=·bgd(a,·l,·1000,·X_tr,·y_tr,·2)
··y_pred·=·classification(X_ts,·wt)
··sens,·spec,·acc·=·confusion_mat(y_pred,·y_ts)
··SE.append(sens)
··SP.append(spec)
··A.append(acc)
```

```
avg_sens·=·sum(SE)/len(SE)
avg_spec·=·sum(SP)/len(SP)
avg_acc·=·sum(A)/len(A)
print('BGD·LOR·with·L2·regularisation')
print(f"Sensitivity:·{avg_sens}\nSpecificity:·{avg_spec}\nAccuracy:·{avg_acc}\n")
```

```
BGD LOR with L2 regularisation
Sensitivity: 0.8733633611782169
Specificity: 0.7447387997240608
Accuracy: 0.7929203539823009
```

```
wt = np.random.rand(X.shape[1])
SE = []
SP = []
A = []
for i in range(5):
  X_ts, y_ts, X_tr, y_tr = cross_validation(i, X, y)
  a = 0.075
  l = 0.01
  wt = sgd(a, l, 1000, X_tr, y_tr, 0)
  y_pred = classification(X_ts, wt)
  sens, spec, acc = confusion_mat(y_pred, y_ts)
  SE.append(sens)
  SP.append(spec)
  A.append(acc)

avg_sens = sum(SE)/len(SE)
avg_spec = sum(SP)/len(SP)
avg_acc = sum(A)/len(A)
print('SGD LOR without regularisation')
print(f"Sensitivity: {avg_sens}\nSpecificity: {avg_spec}\nAccuracy: {avg_acc}\n")
```

```
SGD LOR without regularisation
Sensitivity: 0.9261216054952411
Specificity: 0.8573777958934887
Accuracy: 0.8814159292035398
```

```
wt = np.random.rand(X.shape[1])
SE = []
SP = []
A = []
for i in range(5):
  X_ts, y_ts, X_tr, y_tr = cross_validation(i, X, y)
  a = 0.01
  l = 0.05
  wt = sgd(a, l, 1000, X_tr, y_tr, 1)
  y_pred = classification(X_ts, wt)
  sens, spec, acc = confusion_mat(y_pred, y_ts)
  SE.append(sens)
  SP.append(spec)
  A.append(acc)

avg_sens = sum(SE)/len(SE)
```

```
avg_spec = sum(SP)/len(SP)
avg_acc = sum(A)/len(A)
print('SGD LOR with L1 regularisation')
print(f"Sensitivity: {avg_sens}\nSpecificity: {avg_spec}\nAccuracy: {avg_acc}\n")
```

```
SGD LOR with L1 regularisation
Sensitivity: 0.902912797974212
Specificity: 0.8745006256676102
Accuracy: 0.8831858407079647
```

```
wt = np.random.rand(X.shape[1])
SE = []
SP = []
A = []
for i in range(5):
  X_ts, y_ts, X_tr, y_tr = cross_validation(i, X, y)
  a = 0.0005
  l = 0.0001
  wt = sgd(a, l, 1000, X_tr, y_tr, 2)
  y_pred = classification(X_ts, wt)
  sens, spec, acc = confusion_mat(y_pred, y_ts)
  SE.append(sens)
  SP.append(spec)
  A.append(acc)

avg_sens = sum(SE)/len(SE)
avg_spec = sum(SP)/len(SP)
avg_acc = sum(A)/len(A)
print('SGD LOR with L2 regularisation')
print(f"Sensitivity: {avg_sens}\nSpecificity: {avg_spec}\nAccuracy: {avg_acc}\n")
```

```
SGD LOR with L2 regularisation
Sensitivity: 0.8981709695258608
Specificity: 0.8745704662368439
Accuracy: 0.8814159292035398
```

```
wt = np.random.rand(X.shape[1])
SE = []
SP = []
A = []
for i in range(5):
  X_ts, y_ts, X_tr, y_tr = cross_validation(i, X, y)
  a = 0.0009
  l = 0.0001
  wt = mbgd(a, l, 1000, 30, X_tr, y_tr, 0)
  y_pred = classification(X_ts, wt)
  sens, spec, acc = confusion_mat(y_pred, y_ts)
  SE.append(sens)
  SP.append(spec)
  A.append(acc)

avg_sens = sum(SE)/len(SE)
avg_spec = sum(SP)/len(SP)
```

```
avg_acc = sum(A)/len(A)
print('MBGD LOR without regularisation')
print(f"Sensitivity: {avg_sens}\nSpecificity: {avg_spec}\nAccuracy: {avg_acc}\n")
```

```
MBGD LOR without regularisation
Sensitivity: 0.896939779375382
Specificity: 0.8723759230192385
Accuracy: 0.8814159292035398
```

```
wt = np.random.rand(X.shape[1])
SE = []
SP = []
A = []
for i in range(5):
  X_ts, y_ts, X_tr, y_tr = cross_validation(i, X, y)
  a = 0.0009
  l = 0.001
  wt = mbgd(a, l, 1000, 30, X_tr, y_tr, 1)
  y_pred = classification(X_ts, wt)
  sens, spec, acc = confusion_mat(y_pred, y_ts)
  SE.append(sens)
  SP.append(spec)
  A.append(acc)

avg_sens = sum(SE)/len(SE)
avg_spec = sum(SP)/len(SP)
avg_acc = sum(A)/len(A)
print('MBGD LOR with L1 regularisation')
print(f"Sensitivity: {avg_sens}\nSpecificity: {avg_spec}\nAccuracy: {avg_acc}\n")
```

```
MBGD LOR with L1 regularisation
Sensitivity: 0.916074599062782
Specificity: 0.8729429797992309
Accuracy: 0.888495575221239
```

```
wt = np.random.rand(X.shape[1])
SE = []
SP = []
A = []
for i in range(5):
  X_ts, y_ts, X_tr, y_tr = cross_validation(i, X, y)
  a = 0.0009
  l = 0.0015
  wt = mbgd(a, l, 1000, 30, X_tr, y_tr, 2)
  y_pred = classification(X_ts, wt)
  sens, spec, acc = confusion_mat(y_pred, y_ts)
  SE.append(sens)
  SP.append(spec)
  A.append(acc)

avg_sens = sum(SE)/len(SE)
avg_spec = sum(SP)/len(SP)
avg_acc = sum(A)/len(A)
```

```
print('MBGD LOR with L2 regularisation')
print(f"Sensitivity: {avg_sens}\nSpecificity: {avg_spec}\nAccuracy: {avg_acc}\n")
```

```
MBGD LOR with L2 regularisation
Sensitivity: 0.9016816078237332
Specificity: 0.8562489669165585
Accuracy: 0.8725663716814159
```

✓ 0s completed at 10:36 PM