Implement SGD Classifier with Logloss and L2 regularization Using SGD without using sklearn

There will be some functions that start with the word "grader" ex: grader_weights(), grader_sigmoid(), grader_logloss() etc, you should not change those function definition.

Every Grader function has to return True.

Importing packages

In [2]:

```
import numpy as np
import pandas as pd
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import linear_model
```

Creating custom dataset

In [3]:

```
# please don't change random_state
X, y = make_classification(n_samples=50000, n_features=15, n_informative=10, n_r
edundant=5,
                           n_classes=2, weights=[0.7], class_sep=0.7, random_sta
te=15)
# make_classification is used to create custom dataset
# Please check this link (https://scikit-learn.org/stable/modules/generated/skle
arn.datasets.make_classification.html) for more details
```

In [4]:

```
X.shape, y.shape
```

Out[4]:

```
((50000, 15), (50000,))
```

Splitting data into train and test

In [5]:

```
#please don't change random state
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random
_state=15)
```

In [6]:

```
# Standardizing the data.
scaler = StandardScaler()
x_train = scaler.fit_transform(X_train)
x_test = scaler.transform(X_test)
```

In [7]:

```
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

Out[7]:

```
((37500, 15), (37500,), (12500, 15), (12500,))
```

SGD classifier

In [8]:

```
# alpha : float
# Constant that multiplies the regularization term.

# eta0 : double
# The initial learning rate for the ‘constant’, ‘invscaling’ or ‘adaptive’ sched
ules.

clf = linear_model.SGDClassifier(eta0=0.0001, alpha=0.0001, loss='log', random_s
tate=15, penalty='l2', tol=1e-3, verbose=2, learning_rate='constant')
clf
# Please check this documentation (https://scikit-learn.org/stable/modules/gener
ated/sklearn.linear_model.SGDClassifier.html)
```

Out[8]:

```
SGDClassifier(alpha=0.0001, average=False, class_weight=None,
              early_stopping=False, epsilon=0.1, eta0=0.0001,
              fit_intercept=True, l1_ratio=0.15, learning_rate='cons
tant',
              loss='log', max_iter=1000, n_iter_no_change=5, n_jobs=
None,
              penalty='l2', power_t=0.5, random_state=15, shuffle=Tr
ue,
              tol=0.001, validation_fraction=0.1, verbose=2, warm_st
art=False)
```

In [9]:

```
clf.fit(X=x_train, y=y_train) # fitting our model
```

```
-- Epoch 1
Norm: 0.70, NNZs: 15, Bias: -0.501317, T: 37500, Avg. loss: 0.552526
Total training time: 0.01 seconds.
-- Epoch 2
Norm: 1.04, NNZs: 15, Bias: -0.752393, T: 75000, Avg. loss: 0.448021
Total training time: 0.02 seconds.
-- Epoch 3
Norm: 1.26, NNZs: 15, Bias: -0.902742, T: 112500, Avg. loss: 0.41572
4
Total training time: 0.03 seconds.
-- Epoch 4
Norm: 1.43, NNZs: 15, Bias: -1.003816, T: 150000, Avg. loss: 0.40089
5
Total training time: 0.04 seconds.
-- Epoch 5
Norm: 1.55, NNZs: 15, Bias: -1.076296, T: 187500, Avg. loss: 0.39287
9
Total training time: 0.06 seconds.
-- Epoch 6
Norm: 1.65, NNZs: 15, Bias: -1.131077, T: 225000, Avg. loss: 0.38809
4
Total training time: 0.07 seconds.
-- Epoch 7
Norm: 1.73, NNZs: 15, Bias: -1.171791, T: 262500, Avg. loss: 0.38507
7
Total training time: 0.08 seconds.
-- Epoch 8
Norm: 1.80, NNZs: 15, Bias: -1.203840, T: 300000, Avg. loss: 0.38307
4
Total training time: 0.09 seconds.
-- Epoch 9
Norm: 1.86, NNZs: 15, Bias: -1.229563, T: 337500, Avg. loss: 0.38170
3
Total training time: 0.10 seconds.
-- Epoch 10
Norm: 1.90, NNZs: 15, Bias: -1.251245, T: 375000, Avg. loss: 0.38076
3
Total training time: 0.11 seconds.
-- Epoch 11
Norm: 1.94, NNZs: 15, Bias: -1.269044, T: 412500, Avg. loss: 0.38008
4
Total training time: 0.12 seconds.
-- Epoch 12
Norm: 1.98, NNZs: 15, Bias: -1.282485, T: 450000, Avg. loss: 0.37960
7
Total training time: 0.13 seconds.
-- Epoch 13
Norm: 2.01, NNZs: 15, Bias: -1.294386, T: 487500, Avg. loss: 0.37925
1
Total training time: 0.14 seconds.
-- Epoch 14
Norm: 2.03, NNZs: 15, Bias: -1.305805, T: 525000, Avg. loss: 0.37899
2
Total training time: 0.15 seconds.
Convergence after 14 epochs took 0.15 seconds
```

Out[9]:

```
SGDClassifier(alpha=0.0001, average=False, class_weight=None,
              early_stopping=False, epsilon=0.1, eta0=0.0001,
              fit_intercept=True, l1_ratio=0.15, learning_rate='cons
tant',
              loss='log', max_iter=1000, n_iter_no_change=5, n_jobs=
None,
              penalty='l2', power_t=0.5, random_state=15, shuffle=Tr
ue,
              tol=0.001, validation_fraction=0.1, verbose=2, warm_st
art=False)
```

In [10]:

```
clf.coef_, clf.coef_.shape, clf.intercept_
#clf.coef_ will return the weights
#clf.coef_.shape will return the shape of weights
#clf.intercept_ will return the intercept term
```

Out[10]:

```
(array([[-0.89007184,  0.63162363, -0.07594145,  0.63107107, -0.3843
4375,
          0.93235243, -0.89573521, -0.07340522,  0.40591417,  0.4199
991 ,
          0.24722143,  0.05046199, -0.08877987,  0.54081652,  0.0664
3888]]),
 (1, 15),
 array([-1.30580538]))
```

Implement Logistic Regression with L2 regularization Using SGD: without using sklearn

1. We will be giving you some functions, please write code in that functions only.
2. After every function, we will be giving you expected output, please make sure that you get that output.

- Initialize the weight_vector and intercept term to zeros (Write your code in def initialize_weights())
- Create a loss function (Write your code in def logloss())

  $logloss = -1 * \frac{1}{n}\Sigma_{foreachYt,Y_{pred}}(Ytlog10(Y_{pred}) + (1 - Yt)log10(1 - Y_{pred}))$
- for each epoch:
  - for each batch of data points in train: (keep batch size=1)
    - calculate the gradient of loss function w.r.t each weight in weight vector (write your code in def gradient_dw())

      $dw^{(t)} = x_n(y_n - \sigma((w^{(t)})^T x_n + b^t)) - \frac{\lambda}{N}w^{(t)})$
    - Calculate the gradient of the intercept (write your code in def gradient_db()) check this (https://drive.google.com/file/d/1nQ08-XY4zvOLzRX-lGf8EYB5arb7-m1H/view?usp=sharing)

      $db^{(t)} = y_n - \sigma((w^{(t)})^T x_n + b^t))$
    - Update weights and intercept (check the equation number 32 in the above mentioned pdf (https://drive.google.com/file/d/1nQ08-XY4zvOLzRX-lGf8EYB5arb7-m1H/view?usp=sharing)):
      $w^{(t+1)} \leftarrow w^{(t)} + \alpha(dw^{(t)})$

      $b^{(t+1)} \leftarrow b^{(t)} + \alpha(db^{(t)})$
  - calculate the log loss for train and test with the updated weights (you can check the python assignment 10th question)
  - And if you wish, you can compare the previous loss and the current loss, if it is not updating, then you can stop the training
  - append this loss in the list (this will be used to see how loss is changing for each epoch after the training is over)

Initialize weights

In [11]:

```
def initialize_weights(dim):
    ''' In this function, we will initialize our weights and bias'''
    #initialize the weights to zeros array of (1,dim) dimensions
    #you use zeros_like function to initialize zero, check this link https://doc
s.scipy.org/doc/numpy/reference/generated/numpy.zeros_like.html
    #initialize bias to zero
    w = np.zeros_like(dim)
    b = 0
    return w,b
```

In [12]:

```
dim = x_train[0]
w,b = initialize_weights(dim)
print('w =',(w), w.size)
print('b =',str(b))
```

```
w = [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.] 15
b = 0
```

Grader function - 1

In [13]:

```
dim=x_train[0]
w,b = initialize_weights(dim)

def grader_weights(w,b):
  assert((len(w)==len(dim)) and b==0 and np.sum(w)==0.0)
  return True
grader_weights(w,b)
```

Out[13]:

```
True
```

Compute sigmoid

$sigmoid(z) = 1/(1 + exp(-z))$

In [14]:

```
def sigmoid(z):
    ''' In this function, we will return sigmoid of z'''
    sig = 1/(1 + np.exp(-z))
    return sig
```

Grader function - 2

In [15]:

```
def grader_sigmoid(z):
  val=sigmoid(z)
  assert(val==0.8807970779778823)
  return True
grader_sigmoid(2)
```

Out[15]:

```
True
```

Compute loss

$logloss = -1 * \frac{1}{n}\Sigma_{foreachYt,Y_{pred}}(Ytlog10(Y_{pred}) + (1 - Yt)log10(1 - Y_{pred}))$

In [16]:

```
import math

def logloss(y_true, y_pred):
    '''In this function, we will compute log loss '''
    '''
    y_true = np.array(y_true)
    y_pred = np.array(y_pred)
    logloss = -1 * np.mean( y_true*np.log(y_pred) + (1-y_true)*np.log(1-y_pred))
    '''
    logloss=0
    for i in range(len(y_true)):
        logloss += ((y_true[i]*math.log10(y_pred[i]))+((1-y_true[i])*math.log10(
1-y_pred[i])))
    logloss = -1*(1/len(y_true))*logloss
    return logloss
```

Grader function - 3

In [17]:

```
def grader_logloss(true1,pred):
  loss=logloss(true1,pred)
  assert(loss==0.07644900402910389)
  return True
true1=[1,1,0,1,0]
pred=[0.9,0.8,0.1,0.8,0.2]
grader_logloss(true1,pred)
```

Out[17]:

```
True
```

Compute gradient w.r.to 'w'

$$dw^{(t)} = x_n(y_n - \sigma((w^{(t)})^T x_n + b^t)) - \frac{\lambda}{N} w^{(t)}$$

In [18]:

```
def gradient_dw(x,y,w,b,alpha,N):
    '''In this function, we will compute the gardient w.r.to w '''
    dw = x * (y-sigmoid(np.dot(w.T,x)+b)) - ((alpha*w)/N)

    return dw
```

Grader function - 4

In [19]:

```
def grader_dw(x,y,w,b,alpha,N):
  grad_dw=gradient_dw(x,y,w,b,alpha,N)
  assert(np.sum(grad_dw)==2.613689585)
  return True
grad_x=np.array([-2.07864835,  3.31604252, -0.79104357, -3.87045546, -1.14783286
,
       -2.81434437, -0.86771071, -0.04073287,  0.84827878,  1.99451725,
        3.67152472,  0.01451875,  2.01062888,  0.07373904, -5.54586092])
grad_y=0
grad_w,grad_b=initialize_weights(grad_x)
alpha=0.0001
N=len(X_train)
grader_dw(grad_x,grad_y,grad_w,grad_b,alpha,N)
```

Out[19]:

```
True
```

Compute gradient w.r.to 'b'

$$db^{(t)} = y_n - \sigma((w^{(t)})^T x_n + b^t)$$

In [20]:

```
def gradient_db(x,y,w,b):
    '''In this function, we will compute gradient w.r.to b '''
    db = y-sigmoid(np.dot(w.T,x)+b)
    return db
```

Grader function - 5

In [21]:

```
def grader_db(x,y,w,b):
  grad_db=gradient_db(x,y,w,b)
  assert(grad_db==-0.5)
  return True
grad_x=np.array([-2.07864835,  3.31604252, -0.79104357, -3.87045546, -1.14783286
,
       -2.81434437, -0.86771071, -0.04073287,  0.84827878,  1.99451725,
        3.67152472,  0.01451875,  2.01062888,  0.07373904, -5.54586092])
grad_y=0
grad_w,grad_b=initialize_weights(grad_x)
alpha=0.0001
N=len(X_train)
grader_db(grad_x,grad_y,grad_w,grad_b)
```

Out[21]:

```
True
```

Implementing logistic regression

In [62]:

```
def train(X_train, y_train, X_test, y_test, epochs, alpha, eta0):
    w,b = initialize_weights(X_train[0])
    N=len(X_train)
    log_loss_train = []
    log_loss_test = []

    for i in range(0, epochs):
        for j in range(N):
            grad_dw = gradient_dw(X_train[j], y_train[j], w, b, alpha, N)
            grad_db = gradient_db(X_train[j], y_train[j], w, b)
            w       = np.array(w) + (eta0*(np.array(grad_dw)))
            b       = b + (eta0*(grad_db))

        # predict the output of x_train[for all data points in X_train] using w,
b
        predict_train = []
        for m in range(len(y_train)):
            z = np.dot(w, X_train[m])+b
            predict_train.append(sigmoid(z))

        # compute the loss between predicted and actual values (call the loss fu
nction)
        # store all the train loss values in a list
        train_loss = logloss(y_train, predict_train)

        # predict the output of x_test[for all data points in X_test] using w,b
        predict_test = []
        for m in range(len(y_test)):
            z = np.dot(w, X_test[m])+b
            predict_test.append(sigmoid(z))

        # compute the loss between predicted and actual values (call the loss fu
nction)
        # store all the test loss values in a list
        test_loss = logloss(y_test, predict_test)

        # you can also compare previous loss and current loss, if loss is not up
dating then stop the process and return w,b
        if log_loss_train and train_loss > log_loss_train[-1]: # and log_loss_te
st and test_loss > log_loss_test[-1]:
          return w, b, log_loss_train, log_loss_test

        log_loss_train.append(train_loss)
        log_loss_test.append(test_loss)

    return w, b, log_loss_train, log_loss_test
```

In [63]:

```
alpha  = 0.0001
eta0   = 0.0001
epochs = 50
w, b, log_loss_train, log_loss_test = train(x_train, y_train, x_test, y_test, ep
ochs, alpha, eta0)
```

In [64]:

```
print ("weight vector: ", w)
print ("Intercept: ", b)
print ("log loss train", log_loss_train)
print ("log loss test", log_loss_test)
```

```
weight vector:  [-0.9712546   0.6951594  -0.10648865  0.68159052 -0.
44472549  1.00799631
 -0.94341139 -0.07316671  0.44633494  0.47814801  0.27402297  0.0601
3629
 -0.09610535  0.57042941  0.06404647]
Intercept:  -1.369139915843557
log loss train [0.20729781784140838, 0.18556210141426163, 0.17659652
085620509, 0.17201289496451905, 0.16938000886115878, 0.1677533657545
5, 0.1666977629761566, 0.1659883750043287, 0.16549918227604976, 0.16
515513945496194, 0.16490944296095902, 0.16473183113949116, 0.1646021
696464541, 0.16450674989278702, 0.16443606134127778, 0.1643834031232
729, 0.16434399300380859, 0.16431438119164662, 0.1642920564949399,
0.16427517690890994, 0.16426238246016317, 0.1642526634578411, 0.1642
4526668148529, 0.16423962791724517, 0.1642353230239534, 0.1642320321
7373442, 0.16422951354993254, 0.16422758389129435, 0.164226104029038
06, 0.16422496808893378, 0.1642240953990856, 0.16422342440533785, 0.
16422290808287712, 0.1642225104673144, 0.164222040262155, 0.1642219
676635313, 0.16422178520193, 0.16422164422678262, 0.1642215352044480
4, 0.16422145080897918, 0.1642213854074215, 0.16422133466598673, 0.1
642212952484606, 0.16422126458506267, 0.1642212406951902, 0.16422122
20514016, 0.16422120747495308, 0.16422119605559377, 0.16422118708987
574, 0.16422118003373862]
log loss test [0.20722219781181883, 0.1856525943467828, 0.1768256772
0849302, 0.17235324848189565, 0.1698100984080047, 0.1682566349822005
6, 0.1672612889069227, 0.16660192986644845, 0.16615457121757737, 0.1
6584572669386236, 0.16562980540333389, 0.16547750036682654, 0.165369
43679761335, 0.16529251625482547, 0.16523772268630538, 0.16519875926
88669, 0.165171176424903, 0.16515180015954947, 0.16513834939454072,
0.16512917523384424, 0.1651230806039871, 0.1651191938733821, 0.16511
68793161568, 0.16511567308220967, 0.16511523704234513, 0.16511532529
485734, 0.16511575972396544, 0.16511641208166614, 0.1651171908033833
6, 0.16511803127917835, 0.16511888866014965, 0.1651197325326434, 0.1
651205429733953, 0.16512130762843552, 0.16512201955262923, 0.1651226
756151328, 0.16512327532633586, 0.1651238199786852, 0.16512431202125
48, 0.16512475460810555, 0.16512515127566163, 0.16512550571555762,
0.16512582161788045, 0.16512610256603702, 0.16512635196923017, 0.165
12657302209208, 0.16512676868368145, 0.16512694167010694, 0.16512709
445649618, 0.16512722928519263]
```

Goal of assignment

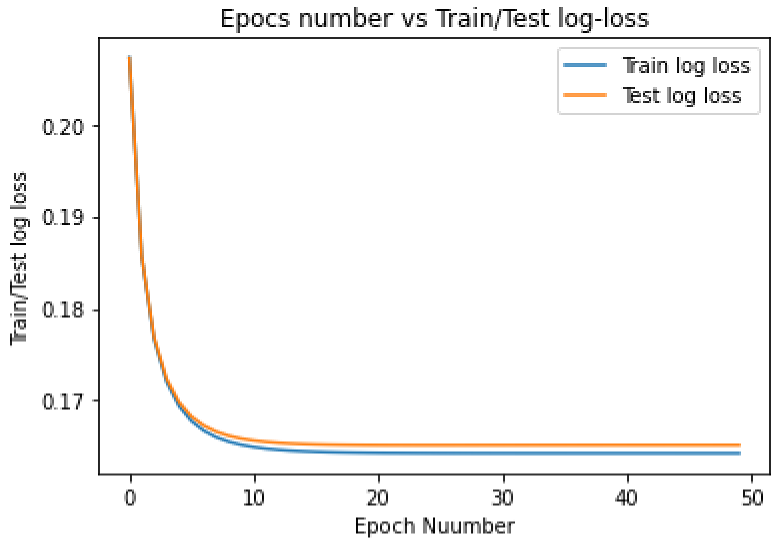
Compare your implementation and SGDClassifier's the weights and intercept, make sure they are as close as possible i.e difference should be in terms of 10^-3

Plot epoch number vs train , test loss

- epoch number on X-axis
- loss on Y-axis

In [79]:

```
import matplotlib.pyplot as plt
plt.title("Epocs number vs Train/Test log-loss")
x = [x for x in range(len(log_loss_train))]
plt.xlabel("Epoch Nuumber")
plt.ylabel("Train/Test log loss")
plt.plot(x, log_loss_train, label='Train log loss')
plt.plot(x, log_loss_test, label='Test log loss')
plt.legend()
plt.show()
```



In [80]:

```
def pred(w, b, X):
    N = len(X)
    predict = []
    for i in range(N):
        z=np.dot(w,X[i])+b
        if sigmoid(z) >= 0.5: # sigmoid(w,x,b) returns 1/(1+exp(-(dot(x,w)+b)))
            predict.append(1)
        else:
            predict.append(0)
    return np.array(predict)
print(1-np.sum(y_train - pred(w,b,x_train))/len(x_train))
print(1-np.sum(y_test  - pred(w,b,x_test))/len(x_test))
```

```
0.9518666666666666
0.94936
```