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 [44]:

```
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 [45]:

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

In [46]:

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

Out[46]:

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

Splitting data into train and test

In [47]:

```
#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=1
5)
```

In [48]:

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

In [49]:

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

Out[49]:

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

SGD classifier

In [50]:

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

# eta0 : double
# The initial learning rate for the 'constant', 'invscaling' or 'adaptive' schedules.

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

Out[50]:

```
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='constant',
              loss='log', max_iter=1000, n_iter_no_change=5, n_jobs=None,
              penalty='l2', power_t=0.5, random_state=15, shuffle=True,
              tol=0.001, validation_fraction=0.1, verbose=2, warm_start=False)
```

In [51]:

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

```
-- Epoch 1
Norm: 0.77, NNZs: 15, Bias: -0.316653, T: 37500, Avg. loss: 0.455552
Total training time: 0.01 seconds.
-- Epoch 2
Norm: 0.91, NNZs: 15, Bias: -0.472747, T: 75000, Avg. loss: 0.394686
Total training time: 0.02 seconds.
-- Epoch 3
Norm: 0.98, NNZs: 15, Bias: -0.580082, T: 112500, Avg. loss: 0.385711
Total training time: 0.03 seconds.
-- Epoch 4
Norm: 1.02, NNZs: 15, Bias: -0.658292, T: 150000, Avg. loss: 0.382083
Total training time: 0.04 seconds.
-- Epoch 5
Norm: 1.04, NNZs: 15, Bias: -0.719528, T: 187500, Avg. loss: 0.380486
Total training time: 0.05 seconds.
-- Epoch 6
Norm: 1.05, NNZs: 15, Bias: -0.763409, T: 225000, Avg. loss: 0.379578
Total training time: 0.05 seconds.
-- Epoch 7
Norm: 1.06, NNZs: 15, Bias: -0.795106, T: 262500, Avg. loss: 0.379150
Total training time: 0.06 seconds.
-- Epoch 8
Norm: 1.06, NNZs: 15, Bias: -0.819925, T: 300000, Avg. loss: 0.378856
Total training time: 0.07 seconds.
-- Epoch 9
Norm: 1.07, NNZs: 15, Bias: -0.837805, T: 337500, Avg. loss: 0.378585
Total training time: 0.08 seconds.
-- Epoch 10
Norm: 1.08, NNZs: 15, Bias: -0.853138, T: 375000, Avg. loss: 0.378630
Total training time: 0.09 seconds.
Convergence after 10 epochs took 0.09 seconds
```

Out[51]:

```
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='constant',
              loss='log', max_iter=1000, n_iter_no_change=5, n_jobs=None,
              penalty='l2', power_t=0.5, random_state=15, shuffle=True,
              tol=0.001, validation_fraction=0.1, verbose=2, warm_start=False)
```

In [52]:

```
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[52]:

```
(array([[-0.42336692,  0.18547565, -0.14859036,  0.34144407, -0.2081867 ,
         0.56016579, -0.45242483, -0.09408813,  0.2092732 ,  0.18084126,
         0.19705191,  0.00421916, -0.0796037 ,  0.33852802,  0.02266721]]),
 (1, 15),
 array([-0.8531383]))
```

```
   # This is formatted as code
```

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**

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

    - **Update weights and intercept (check the equation number 32 in the above mentioned pdf):**

$$w^{(t+1)} \leftarrow w^{(t)} + \alpha(dw^{(t)})$$

$$b^{(t+1)} \leftarrow b^{(t)} + \alpha(db^{(t)})$$

  - **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 [53]:

```
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://docs.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 [54]:

```
dim=X_train[0]
w,b = initialize_weights(dim)
print('w =',(w))
print('b =',str(b))
```

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

Grader function - 1

In [55]:

```
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[55]:

```
True
```

Compute sigmoid

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

In [56]:

```
#import math
def sigmoid(z):
    ''' In this function, we will return sigmoid of z'''
    # compute sigmoid(z) and return
    #https://www.educative.io/edpresso/calculating-the-exponential-value-in-python
    #sigmoid_z = 1/(1+math.exp(-z))
    #https://numpy.org/doc/stable/reference/generated/numpy.exp.html
    #we can also use math library to calculate the exponential value
    sigmoid_z = 1/(1+np.exp(-z))
    return sigmoid_z
```

Grader function - 2

In [57]:

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

```
  return True
grader_sigmoid(2)
```

Out[57]:

```
True
```

Compute loss

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

In [58]:

```
def logloss(y_true,y_pred):
    '''In this function, we will compute log loss '''
    loss = 0
    n = len(y_true)
    for i in range(n):
        # we can also use either math.log10 or math.log(value,10)
        #loss += y_true[i]*math.log10(y_pred[i]) + (1-y_true[i])*math.log10(1-y_pred[i])
        #https://www.geeksforgeeks.org/numpy-log10-python/#:~:text=About%20%3A,all%20the%20input%20array%20elements.
        loss += y_true[i]*np.log10(y_pred[i]) + (1-y_true[i])*np.log10(1-y_pred[i])
    loss = -1 * (loss/n)

    return loss
```

Grader function - 3

In [59]:

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

Out[59]:

```
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 [60]:

```
def gradient_dw(x,y,w,b,alpha,N):
    '''In this function, we will compute the gardient w.r.to w '''
    #https://numpy.org/doc/stable/reference/generated/numpy.dot.html
    dw = x * (y - sigmoid(np.dot(w,x)+b)) - (alpha/N)*w
    return dw
```

Grader function - 4

In [61]:

```
def grader_dw(x,y,w,b,alpha,N):
  grad_dw=gradient_dw(x,y,w,b,alpha,N)
  #print(type(grad_dw))
  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[61]:

```
True
```

Compute gradient w.r.to 'b'

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

In [62]:

```
def gradient_db(x,y,w,b):
     '''In this function, we will compute gradient w.r.to b '''
     db = y - sigmoid(np.dot(w,x) + b)
     return db
```

Grader function - 5

In [63]:

```
def grader_db(x,y,w,b):
  grad_db=gradient_db(x,y,w,b)
  #print(type(grad_db))
  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[63]:

```
True
```

Implementing logistic regression

In [65]:

```
def train(X_train,y_train,X_test,y_test,epochs,alpha,eta0):
    ''' In this function, we will implement logistic regression'''
    train_loss = []
    test_loss = []
    w,b = initialize_weights(X_train[0])
    for i in range(0,epochs):
        #https://stackoverflow.com/questions/49783594/for-loop-and-zip-in-python
        #zip allows you to iterate two lists at the same time
        for x,y in zip(X_train,y_train):
              grad_dw = gradient_dw(x,y,w,b,alpha,N)
              grad_db = gradient_db(x,y,w,b)
              w = w + (eta0*grad_dw)
              b = b + (eta0*grad_db)

        y_train_pred = []
        for x_test in X_train:
             y_train_pred.append(sigmoid(np.dot(w,x)+b))
```

```
        #computing and storing all the train loss values in a list
        train_loss.append(logloss(y_train,y_train_pred))

        y_test_pred = []
        for x in X_test:
            y_test_pred.append(sigmoid(np.dot(w,x)+b))
        #computing and storing all the test loss values in a list
        test_loss.append(logloss(y_test,y_test_pred))

        #https://stackoverflow.com/questions/455612/limiting-floats-to-two-decimal-points
        #limiting the train and test loss upto 5 decimal places
        print(f"EPOCH: {i} Train Loss: {logloss(y_train, y_train_pred):.5f} Test Loss: {l
ogloss(y_test, y_test_pred):.5f}")


        for i,loss in enumerate(train_loss):

            #print(i, loss)
            if i > 0:
                previous_loss = train_loss[i-1]
                current_loss = loss
                #print(previous_loss,current_loss)
                if(previous_loss > current_loss) and (previous_loss - current_loss < 0.0
001):
                    break
        else:
            continue
        break

    return w,b,train_loss,test_loss
```

In [66]:

```
alpha=0.0001
eta0=0.0001
N=len(X_train)
epochs=50
w,b,loss_train,loss_test=train(X_train,y_train,X_test,y_test,epochs,alpha,eta0)
```

```
EPOCH: 0 Train Loss: 0.29606 Test Loss: 0.17595
EPOCH: 1 Train Loss: 0.29128 Test Loss: 0.16940
EPOCH: 2 Train Loss: 0.29005 Test Loss: 0.16721
EPOCH: 3 Train Loss: 0.28953 Test Loss: 0.16622
EPOCH: 4 Train Loss: 0.28921 Test Loss: 0.16572
EPOCH: 5 Train Loss: 0.28899 Test Loss: 0.16546
EPOCH: 6 Train Loss: 0.28882 Test Loss: 0.16531
EPOCH: 7 Train Loss: 0.28869 Test Loss: 0.16523
EPOCH: 8 Train Loss: 0.28860 Test Loss: 0.16519
```

In [67]:

```
print(w)
print(b)
#print(train_loss)
#print(test_loss)
```

```
[-0.42101199  0.19028065 -0.14496425  0.33810803 -0.20893349  0.56366252
 -0.44548271 -0.09228306  0.21670984  0.16844415  0.1940773   0.00318758
 -0.07669313  0.33872841  0.02189547]
-0.8369332521112822
```

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

In [68]:

```
# these are the results we got after we implemented sgd and found the optimal weights and
intercept
```

```
w-clf.coef_, b-clf.intercept_
```

Out[68]:

```
(array([[ 0.00235492,  0.00480499,  0.00362611, -0.00333604, -0.00074678,
          0.00349673,  0.00694212,  0.00180507,  0.00743664, -0.01239711,
         -0.00297461, -0.00103157,  0.00291056,  0.0002004 , -0.00077174]]),
 array([0.01620505]))
```

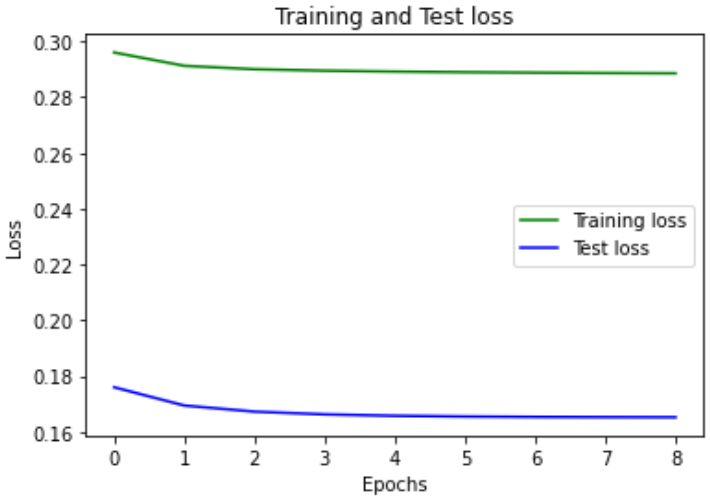
Plot epoch number vs train , test loss

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

In [69]:

```
import matplotlib.pyplot as plt

epochs = range(0,9)
plt.plot(epochs, loss_train, 'g', label='Training loss')
plt.plot(epochs, loss_test, 'b', label='Test loss')
plt.title('Training and Test loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



In [70]:

```
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))
```

```
0.95648
0.95416
```