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 [630]:

```
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
from ipykernel import kernelapp as app
import matplotlib.pyplot as plt
```

Creating custom dataset

In [631]:

```
# 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 [632]:

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

Out[632]:

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

Splitting data into train and test

In [633]:

```
#please don't change random state
# you need not standardize the data as it is already standardized
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=1
5)
```

In [634]:

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

Out[634]:

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

SGD classifier

In [635]:

```
# 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[635]:

```
SGDClassifier(eta0=0.0001, learning_rate='constant', loss='log',
              random_state=15, verbose=2)
```

In [636]:

```
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: 0.06 seconds.
-- Epoch 7
Norm: 1.06, NNZs: 15, Bias: -0.795106, T: 262500, Avg. loss: 0.379150
Total training time: 0.07 seconds.
-- Epoch 8
Norm: 1.06, NNZs: 15, Bias: -0.819925, T: 300000, Avg. loss: 0.378856
Total training time: 0.08 seconds.
-- Epoch 9
Norm: 1.07, NNZs: 15, Bias: -0.837805, T: 337500, Avg. loss: 0.378585
Total training time: 0.09 seconds.
-- Epoch 10
Norm: 1.08, NNZs: 15, Bias: -0.853138, T: 375000, Avg. loss: 0.378630
Total training time: 0.10 seconds.
Convergence after 10 epochs took 0.10 seconds
```

Out[636]:

```
SGDClassifier(eta0=0.0001, learning_rate='constant', loss='log',
              random_state=15, verbose=2)
```

In [637]:

```
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[637]:

```
(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]))
```

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 [638]:

```
def initialize_weights(row_vector):
    ''' In this function, we will initialize our weights and bias'''
    #initialize the weights as 1d array consisting of all zeros similar to the dimensions of row_vector
    #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(row_vector)
    b=0
    return w,b
```

In [639]:

```
row_vector=X_train[0]
w,b = initialize_weights(row_vector)
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 [640]:

```
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[640]:

```
True
```

Compute sigmoid

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

In [641]:

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

Grader function - 2

In [642]:

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

Out[642]:

```
True
```

Compute loss

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

In [643]:

```
def logloss(y_true,y_pred):
    # you have been given two arrays y_true and y_pred and you have to calculate the logl
```

```
oss
    #while dealing with numpy arrays you can use vectorized operations for quicker calcul
ations as compared to using loops
    #https://www.pythonlikeyoumeanit.com/Module3_IntroducingNumpy/VectorizedOperations.ht
ml
    #https://www.geeksforgeeks.org/vectorized-operations-in-numpy/
    #write your code here
    #y_pred[np.where(y_pred == 0.0)] = 0.0001
    #y_pred[np.where(y_pred == 1.0)] = 0.9999
    loss = (-1) * ((1/len(y_true)) * ((y_true * np.log10(y_pred)) + ((1-y_true) * np.log
10(1-y_pred))).sum())
    return loss
```

Grader function - 3

In [644]:

```
#round off the value to 8 values
def grader_logloss(true,pred):
  loss=logloss(true,pred)
  assert(np.round(loss,6)==0.076449)
  return True
true=np.array([1,1,0,1,0])
pred=np.array([0.9,0.8,0.1,0.8,0.2])
grader_logloss(true,pred)
```

Out[644]:

```
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 [645]:

```
#make sure that the sigmoid function returns a scalar value, you can use dot function ope
ration
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,x) + b))) - ((alpha/N)*w)
    return dw
```

Grader function - 4

In [646]:

```
def grader_dw(x,y,w,b,alpha,N):
  grad_dw=gradient_dw(x,y,w,b,alpha,N)
  assert(np.round(np.sum(grad_dw),5)==4.75684)
  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=np.array([ 0.03364887,  0.03612727,  0.02786927,  0.08547455, -0.12870234,
       -0.02555288,  0.11858013,  0.13305576,  0.07310204,  0.15149245,
       -0.05708987, -0.064768  ,  0.18012332, -0.16880843, -0.27079877])
grad_b=0.5
alpha=0.0001
N=len(X_train)
grader_dw(grad_x,grad_y,grad_w,grad_b,alpha,N)
```

Out[646]:

```
True
```

Compute gradient w.r.to 'b'

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

In [647]:

```
#sb should be a scalar value
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 [648]:

```
def grader_db(x,y,w,b):
  grad_db=gradient_db(x,y,w,b)
  assert(np.round(grad_db,4)==-0.3714)
  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=0.5
grad_b=0.1
grad_w=np.array([ 0.03364887,  0.03612727,  0.02786927,  0.08547455, -0.12870234,
       -0.02555288,  0.11858013,  0.13305576,  0.07310204,  0.15149245,
       -0.05708987, -0.064768  ,  0.18012332, -0.16880843, -0.27079877])
alpha=0.0001
N=len(X_train)
grader_db(grad_x,grad_y,grad_w,grad_b)
```

Out[648]:

```
True
```

In [649]:

```
# prediction function used to compute predicted_y given the dataset X
def pred(w,b, X):
    N = len(X)
    predict = []
    for i in range(N):
        z=np.dot(w,X[i])+b
        predict.append(sigmoid(z))
    return np.array(predict)
```

Implementing logistic regression

In [650]:

```
def train(X_train,y_train,X_test,y_test,epochs,alpha,eta0):
    ''' In this function, we will implement logistic regression'''
    #Here eta0 is learning rate
    #implement the code as follows
    # initalize the weights (call the initialize_weights(X_train[0]) function)
    # for every epoch
        # for every data point(X_train,y_train)
           #compute gradient w.r.to w (call the gradient_dw() function)
           #compute gradient w.r.to b (call the gradient_db() function)
           #update w, b
        # predict the output of x_train [for all data points in X_train] using pred funct
ion with updated weights
        # compute the loss between predicted and actual values (call the loss function)
        # store all the train loss values in a list
        # predict the output of x_test [for all data points in X_test] using pred functio
```

```
n with updated weights
        # compute the loss between predicted and actual values (call the loss function)
        # store all the test loss values in a list
        # you can also compare previous loss and current loss, if loss is not updating th
en stop the process
        # you have to return w,b , train_loss and test loss

    train_loss = []
    test_loss = []
    w,b = initialize_weights(X_train[0]) # Initialize the weights
    #write your code to perform SGD
    for epc in range(epochs):
      print('*' * 10 + '   Starting Epoch ' + str(epc+1) + ' ' +'*'*10)
      for dp in range(len(X_train)):
        w_grad = gradient_dw(X_train[dp],y_train[dp],w,b,alpha,len(X_train))
        b_grad = gradient_db(X_train[dp],y_train[dp],w,b)
        w = w + eta0*w_grad
        b = b + eta0*b_grad
      #print('*' * 30 )
      #print(w)
      #print(b)
      #print('*' * 30 )
      y_train_pred = pred(w,b, X_train)
      y_test_pred = pred(w,b, X_test)
      #print('*' * 30 )
      #print(y_train_pred)
      #print(y_test_pred)
      #print('*' * 30 )
      trainloss = logloss(y_train,y_train_pred)
      testloss = logloss(y_test,y_test_pred)
      #print('*' * 30 )
      #print(trainloss)
      #print(testloss)
      #print('*' * 30 )
      train_loss.append(trainloss)
      test_loss.append(testloss)
      #print('*' * 30 )
      print('train loss is : {0}  and test loss is : {1}'.format(trainloss,testloss))
      '''
      # auto stopping critieria
      if epc > 1:
        if test_loss[-1] - test_loss[-2] < 0.00000001:
          print('Stopping execution as loss did not improve from {0} to {1} '.format(test
_loss[-2], test_loss[-1]))
          break
        '''
    return w,b,train_loss,test_loss
```

In [651]:

```
alpha=0.001
eta0=0.001
N=len(X_train)
epochs=20
w,b,train_loss,test_loss=train(X_train,y_train,X_test,y_test,epochs,alpha,eta0)
```

```
**********   Starting Epoch 1 **********
train loss is : 0.16507868561275768  and test loss is : 0.16617155343771076
**********   Starting Epoch 2 **********
train loss is : 0.16505852702121138  and test loss is : 0.16614624611227138
**********   Starting Epoch 3 **********
train loss is : 0.165059232423908 1  and test loss is : 0.1661466517170205
**********   Starting Epoch 4 **********
train loss is : 0.16505928381261364  and test loss is : 0.16614668482266418
**********   Starting Epoch 5 **********
train loss is : 0.16505928704995199  and test loss is : 0.1661466869209223
**********   Starting Epoch 6 **********
train loss is : 0.16505928725221397  and test loss is : 0.16614668705206817
**********   Starting Epoch 7 **********
train loss is : 0.16505928726484442  and test loss is : 0.16614668706025792
**********   Starting Epoch 8 **********
train loss is : 0.16505928726563307  and test loss is : 0.1661466870607693
```

```
**********   Starting Epoch 9 **********
train loss is : 0.16505928726568234  and test loss is : 0.16614668706080124
**********   Starting Epoch 10 **********
train loss is : 0.16505928726568542  and test loss is : 0.1661466870608032
**********   Starting Epoch 11 **********
train loss is : 0.1650592872656856  and test loss is : 0.16614668706080332
**********   Starting Epoch 12 **********
train loss is : 0.1650592872656856  and test loss is : 0.16614668706080335
**********   Starting Epoch 13 **********
train loss is : 0.1650592872656856  and test loss is : 0.16614668706080335
**********   Starting Epoch 14 **********
train loss is : 0.16505928726568558  and test loss is : 0.16614668706080332
**********   Starting Epoch 15 **********
train loss is : 0.16505928726568564  and test loss is : 0.16614668706080332
**********   Starting Epoch 16 **********
train loss is : 0.16505928726568564  and test loss is : 0.16614668706080335
**********   Starting Epoch 17 **********
train loss is : 0.16505928726568564  and test loss is : 0.16614668706080335
**********   Starting Epoch 18 **********
train loss is : 0.16505928726568564  and test loss is : 0.16614668706080335
**********   Starting Epoch 19 **********
train loss is : 0.16505928726568564  and test loss is : 0.16614668706080335
**********   Starting Epoch 20 **********
train loss is : 0.16505928726568564  and test loss is : 0.16614668706080335
```

In [652]:

```
#print thr value of weights w and bias b
print(w)
print(b)
```

```
[-0.41395277  0.19245295 -0.15005228  0.32635321 -0.22516684  0.58646736
 -0.42720457 -0.10028013  0.21483928  0.15555184  0.17881025 -0.01318754
 -0.06496902  0.36313889 -0.00985012]
-0.9016735833888502
```

In [653]:

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

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

Out[653]:

```
(array([[ 0.00941414,  0.0069773 , -0.00146193, -0.01509086, -0.01698014,
          0.02630157,  0.02522026, -0.006192  ,  0.00556608, -0.02528942,
         -0.01824166, -0.0174067 ,  0.01463468,  0.02461087, -0.03251733]]),
 array([-0.04853529]))
```

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 order of 10^-2

Grader function - 6

In [654]:

```
#this grader function should return True
#the difference between custom weights and clf.coef_ should be less than or equal to 0.05
def differece_check_grader(w,b,coef,intercept):
    val_array=np.abs(np.array(w-coef))
    assert(np.all(val_array<=0.05))
    print('The custom weights are correct')
    return True
differece_check_grader(w,b,clf.coef_,clf.intercept_)
```

```
The custom weights are correct
```

Out[654]:

```
True
```

In [654]:

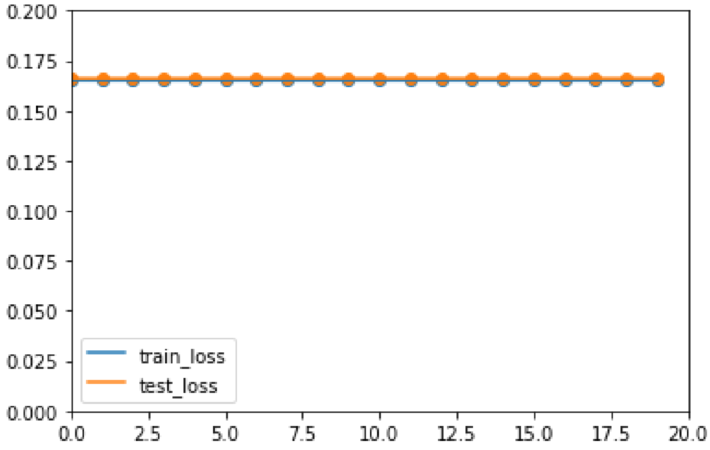
```

```

Plot your train and test loss vs epochs

plot epoch number on X-axis and loss on Y-axis and make sure that the curve is converging

In [655]:

```
plt.ylim(0,0.2)
plt.xlim(0,20)
plt.scatter(range(len(train_loss)),train_loss)
plt.scatter(range(len(test_loss)),test_loss)
plt.plot(range(len(train_loss)),train_loss, label = 'train_loss')
plt.plot(range(len(test_loss)),test_loss, label = 'test_loss')
plt.legend()
plt.show()
```



Both train and Test loss have great convergence

In [655]:

```

```

In [655]:

```

```

In [655]:

```

```