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

```
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

```
# 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.datase
```

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

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

Splitting data into train and test

```
#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)
```

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

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

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

SGD classifier

```
# 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, p
clf
# Please check this documentation (https://scikit-learn.org/stable/modules/generated/sklea
```

```
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)
```

```
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: 0.415724
Total training time: 0.03 seconds.
-- Epoch 4
Norm: 1.43, NNZs: 15, Bias: -1.003816, T: 150000, Avg. loss: 0.400895
Total training time: 0.04 seconds.
-- Epoch 5
Norm: 1.55, NNZs: 15, Bias: -1.076296, T: 187500, Avg. loss: 0.392879
Total training time: 0.05 seconds.
-- Epoch 6
Norm: 1.65, NNZs: 15, Bias: -1.131077, T: 225000, Avg. loss: 0.388094
Total training time: 0.06 seconds.
-- Epoch 7
Norm: 1.73, NNZs: 15, Bias: -1.171791, T: 262500, Avg. loss: 0.385077
Total training time: 0.07 seconds.
-- Epoch 8
Norm: 1.80, NNZs: 15, Bias: -1.203840, T: 300000, Avg. loss: 0.383074
Total training time: 0.08 seconds.
-- Epoch 9
Norm: 1.86, NNZs: 15, Bias: -1.229563, T: 337500, Avg. loss: 0.381703
Total training time: 0.09 seconds.
-- Epoch 10
Norm: 1.90, NNZs: 15, Bias: -1.251245, T: 375000, Avg. loss: 0.380763
Total training time: 0.10 seconds.
-- Epoch 11
Norm: 1.94, NNZs: 15, Bias: -1.269044, T: 412500, Avg. loss: 0.380084
Total training time: 0.11 seconds.
-- Epoch 12
```

```
Norm: 1.98, NNZs: 15, Bias: -1.282485, T: 450000, Avg. loss: 0.379607
Total training time: 0.12 seconds.
-- Epoch 13
Norm: 2.01, NNZs: 15, Bias: -1.294386, T: 487500, Avg. loss: 0.379251
Total training time: 0.13 seconds.
-- Epoch 14
Norm: 2.03, NNZs: 15, Bias: -1.305805, T: 525000, Avg. loss: 0.378992
Total training time: 0.14 seconds.
Convergence after 14 epochs took 0.14 seconds
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)
```

```
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
```

```
(array([[-0.89007184,  0.63162363, -0.07594145,  0.63107107, -0.38434375,
          0.93235243, -0.89573521, -0.07340522,  0.40591417,  0.4199991 ,
          0.24722143,  0.05046199, -0.08877987,  0.54081652,  0.06643888]]),
 (1, 15),
 array([-1.30580538]))
```

```
# 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

Initialize weights

```
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


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/
  #initialize bias to zero
  w = np.zeros_like(X_train[0])
  b=0
  return w,b


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

```
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)
```

```
True
```

Compute sigmoid

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

```
# compute sigmoid(z) and return
def sigmoid(z):
  ''' In this function, we will return sigmoid of z'''
  return 1.0/(1 + np.exp(-z))
```

Grader function - 2

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

```
True
```

Compute loss

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

```
def logloss(y_true,y_pred):
  '''In this function, we will compute log loss '''
  summation_of_log_loss=0
  for i in range(len(y_true)):
    summation_of_log_loss += ((y_true[i] * np.log10(y_pred[i])) + ((1-y_true[i]) * np.log1

  loss = -1*(1/len(y_true))*summation_of_log_loss

  return loss
```

Grader function - 3

```
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)
```

```
True
```

Compute gradient w.r.to 'w'

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

```
def gradient_dw(x,y,w,b,alpha,N):
  '''In this function, we will compute the gardient w.r.to w '''
  z = np.dot(w, x) + b
  dw = x*(y - sigmoid(z)) - ((alpha)*(1/N) * w)
  return dw
```

Grader function - 4

```
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)
```

```
True
```

Compute gradient w.r.to 'b'

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

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

  return db
```

Grader function - 5

```
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)
```

```
True
```

Implementing logistic regression

```
#Here eta0 is learning rate
#implement the code as follows

def train(X_train, y_train, X_test, y_test, epochs, alpha, eta0, tol=1e-3):
  ''' In this function, we will implement logistic regression'''

  # initalize the weights (call the initialize_weights(X_train[0]) function)
  w, b = initialize_weights(X_train[0])
  train_loss = []
  test_loss = []
  N = len(X_train)
  loss_threshold = 0.0001
  while True:
    for epoch in range(epochs):  # for every epoch
      for row in range(N - 1): # for every data point(X_train,y_train)

        #compute gradient w.r.to w (call the gradient_dw() function)
        delta_weights = gradient_dw(X_train[row], y_train[row], w, b, alpha, len(X_train))

        #compute gradient w.r.to b (call the gradient_db() function)
        delta_bias = gradient_db(X_train[row], y_train[row], w, b)

        #update w, b
        w = w + eta0 * delta_weights
        b = b + eta0 * delta_bias

      # predict the output of x_train[for all data points in X_train] using w,b
      predictedOP_y_train = [sigmoid(np.dot(w, x_row) + b) for x_row in X_train]

      #compute the loss between predicted and actual values (call the loss function)
      train_loss.append(logloss(y_train, predictedOP_y_train)) # store all the train loss

      # predict the output of x_test[for all data points in X_test] using w,b
      predictedOP_y_test = [sigmoid(np.dot(w, x_row) + b) for x_row in X_test]
```

```
      print(f"For EPOCH No : {epoch} Train Loss is : {logloss(y_train, predictedOP_y_train


      #compute the loss between predicted and actual values (call the loss function)
      test_loss.append(logloss(y_test, predictedOP_y_test)) # store all the test loss valu


    # you can also compare previous loss and current loss, if loss is not updating then st

    if (len(test_loss) > 3 and (test_loss[-2] - test_loss[-1]) > 0 and (test_loss[-2] - te
      break

  return w,b,train_loss, test_loss
```

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

```
For EPOCH No : 0 Train Loss is : 0.20729876546330706 & Test Loss is : 0.207222562755
For EPOCH No : 1 Train Loss is : 0.18556307486085272 & Test Loss is : 0.185652806654
For EPOCH No : 2 Train Loss is : 0.17659720169678442 & Test Loss is : 0.176825506303
For EPOCH No : 3 Train Loss is : 0.17201336774946205 & Test Loss is : 0.172352807711
For EPOCH No : 4 Train Loss is : 0.16938035193673734 & Test Loss is : 0.169809479135
For EPOCH No : 5 Train Loss is : 0.16775362865599033 & Test Loss is : 0.168255894799
For EPOCH No : 6 Train Loss is : 0.16669797634806888 & Test Loss is : 0.167260464480
For EPOCH No : 7 Train Loss is : 0.16598855826504966 & Test Loss is : 0.166601045500
For EPOCH No : 8 Train Loss is : 0.16549934802176475 & Test Loss is : 0.166153643603
For EPOCH No : 9 Train Loss is : 0.1651552959688971 & Test Loss is : 0.165844767597
For EPOCH No : 10 Train Loss is : 0.16490959570548366 & Test Loss is : 0.16562882326
For EPOCH No : 11 Train Loss is : 0.16473198368710804 & Test Loss is : 0.16547650132
For EPOCH No : 12 Train Loss is : 0.16460232428460345 & Test Loss is : 0.16536842534
For EPOCH No : 13 Train Loss is : 0.16450690802866458 & Test Loss is : 0.16529149571
For EPOCH No : 14 Train Loss is : 0.1644362237785439 & Test Loss is : 0.165236695504
For EPOCH No : 15 Train Loss is : 0.1643835702537931 & Test Loss is : 0.165197727269
For EPOCH No : 16 Train Loss is : 0.1643441649416412 & Test Loss is : 0.165170140961
For EPOCH No : 17 Train Loss is : 0.16431455786708293 & Test Loss is : 0.16515076223
For EPOCH No : 18 Train Loss is : 0.16429223772054577 & Test Loss is : 0.16513730975
For EPOCH No : 19 Train Loss is : 0.16427536242580076 & Test Loss is : 0.16512813443
For EPOCH No : 20 Train Loss is : 0.16426257197021846 & Test Loss is : 0.16512203905
For EPOCH No : 21 Train Loss is : 0.16425285664591543 & Test Loss is : 0.16511815186
For EPOCH No : 22 Train Loss is : 0.16424546323044661 & Test Loss is : 0.16511583706
For EPOCH No : 23 Train Loss is : 0.16423982751795413 & Test Loss is : 0.16511463074
For EPOCH No : 24 Train Loss is : 0.1642355253815806 & Test Loss is : 0.165114194731
For EPOCH No : 25 Train Loss is : 0.16423223701153766 & Test Loss is : 0.16511428308
For EPOCH No : 26 Train Loss is : 0.16422972061123853 & Test Loss is : 0.16511471765
For EPOCH No : 27 Train Loss is : 0.16422779294022757 & Test Loss is : 0.16511537018
For EPOCH No : 28 Train Loss is : 0.16422631485041644 & Test Loss is : 0.16511614910
For EPOCH No : 29 Train Loss is : 0.16422518048759102 & Test Loss is : 0.16511698978
For EPOCH No : 30 Train Loss is : 0.1642243091988613 & Test Loss is : 0.165117847362
For EPOCH No : 31 Train Loss is : 0.1642236394478516 & Test Loss is : 0.165118691430
For EPOCH No : 32 Train Loss is : 0.1642231242262012 & Test Loss is : 0.165119502059
For EPOCH No : 33 Train Loss is : 0.16422272758461876 & Test Loss is : 0.16512026689
For EPOCH No : 34 Train Loss is : 0.1642224220044243 & Test Loss is : 0.165120978982
For EPOCH No : 35 Train Loss is : 0.16422218640203787 & Test Loss is : 0.16512163519
For EPOCH No : 36 Train Loss is : 0.16422200461137335 & Test Loss is : 0.16512223505
```

```
For EPOCH No : 37 Train Loss is : 0.1642218642279109 & Test Loss is : 0.165122779835
For EPOCH No : 38 Train Loss is : 0.16422175572706202 & Test Loss is : 0.16512327199
For EPOCH No : 39 Train Loss is : 0.16422167179097136 & Test Loss is : 0.16512371469
For EPOCH No : 40 Train Loss is : 0.16422160679389303 & Test Loss is : 0.16512411145
For EPOCH No : 41 Train Loss is : 0.16422155640845765 & Test Loss is : 0.16512446598
For EPOCH No : 42 Train Loss is : 0.16422151730414705 & Test Loss is : 0.16512478197
For EPOCH No : 43 Train Loss is : 0.1642214869162373 & Test Loss is : 0.165125062991
For EPOCH No : 44 Train Loss is : 0.1642214632686021 & Test Loss is : 0.165125312459
For EPOCH No : 45 Train Loss is : 0.16422144483775472 & Test Loss is : 0.16512553357
For EPOCH No : 46 Train Loss is : 0.16422143044845963 & Test Loss is : 0.16512572928
For EPOCH No : 47 Train Loss is : 0.16422141919355185 & Test Loss is : 0.16512590231
For EPOCH No : 48 Train Loss is : 0.16422141037230878 & Test Loss is : 0.16512605514
For EPOCH No : 49 Train Loss is : 0.164221403443085 & Test Loss is : 0.165126190012
```

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

```
# these are the results we got after we implemented sgd and found the optimal weights and
w-clf.coef_, b-clf.intercept_
```

```
(array([[-0.08131849,  0.06368005, -0.03055416,  0.05043961, -0.06035257,
          0.07561722, -0.04763114,  0.000364  ,  0.04056776,  0.05818547,
          0.02680422,  0.00971967, -0.00728562,  0.02954035, -0.00241388]]),
 array([-0.06322387]))
```

Plot epoch number vs train , test loss

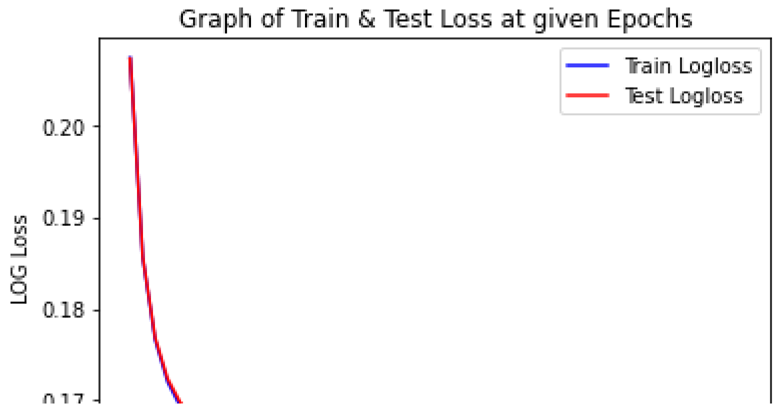
- epoch number on X-axis
- loss on Y-axis

```
import matplotlib.pyplot as plt

x = np.array([i for i in range(0, 50)])
train_log_loss = np.array(train_loss)
test_log_loss = np.array(test_loss)

plt.plot(x, train_log_loss, "-b", label="Train Logloss")
plt.plot(x, test_log_loss, "-r", label="Test Logloss")

plt.legend(loc="upper right")
plt.xlabel('Epoch Numbers')
plt.ylabel('LOG Loss')
plt.title('Graph of Train & Test Loss at given Epochs')
plt.show()
```




```
#Train Accuracy score & Test Accuracy Score:
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))
```

```
0.95184
0.94936
```

I have run the update funtion upto the 50 iterations, and got least train and test error, where I have analysed both test and train error continuously decreasing, it didn't increase anywhere between 0 to 49th epoch.

Thank you!!

✓ 0s completed at 9:54 PM