

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.datasets)

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://colab.research.google.com/drive/1uLDmIxxvOuALiOgDLv-G2\\_aFG6GUH0un#scrollTo=IN0RguI5RQf1&printMode=true

```

```

Norm: 1.98, NNZs: 15, Bias: -1.282485, T: 450000, Avg. loss: 0.379607
Total training time: 0.13 seconds.
-- Epoch 13
Norm: 2.01, NNZs: 15, Bias: -1.294386, T: 487500, Avg. loss: 0.379251
Total training time: 0.14 seconds.
-- Epoch 14
Norm: 2.03, NNZs: 15, Bias: -1.305805, T: 525000, Avg. loss: 0.378992
Total training time: 0.15 seconds.
Convergence after 14 epochs took 0.15 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()`)

$$\text{logloss} = -1 * \frac{1}{n} \sum_{\text{foreach } Y_t, Y_{\text{pred}}} (Y_t \log_{10}(Y_{\text{pred}}) + (1 - Y_t) \log_{10}(1 - Y_{\text{pred}}))$$

- 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.]
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

$$\text{sigmoid}(z) = 1/(1 + \exp(-z))$$

```

# 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

$$\text{logloss} = -1 * \frac{1}{n} \sum_{\text{foreach } Y_t, Y_{\text{pred}}} (Y_t \log_{10}(Y_{\text{pred}}) + (1 - Y_t) \log_{10}(1 - Y_{\text{pred}}))$$

```

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==0.07644900407910389)

```

```

assert np.isclose(2.613689585,
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 gradient 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'''

    # initialize 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("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=15
```

```
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 : 0.2072225627554
For EPOCH No : 1 Train Loss is : 0.18556307486085272 & Test Loss is : 0.1856528066548
For EPOCH No : 2 Train Loss is : 0.17659720169678442 & Test Loss is : 0.1768255063034
For EPOCH No : 3 Train Loss is : 0.17201336774946205 & Test Loss is : 0.1723528077114
For EPOCH No : 4 Train Loss is : 0.16938035193673734 & Test Loss is : 0.1698094791351
For EPOCH No : 5 Train Loss is : 0.16775362865599033 & Test Loss is : 0.1682558947995
For EPOCH No : 6 Train Loss is : 0.16669797634806888 & Test Loss is : 0.1672604644803
For EPOCH No : 7 Train Loss is : 0.16598855826504966 & Test Loss is : 0.1666010455006
For EPOCH No : 8 Train Loss is : 0.16549934802176475 & Test Loss is : 0.1661536436034
For EPOCH No : 9 Train Loss is : 0.1651552959688971 & Test Loss is : 0.16584476759725
For EPOCH No : 10 Train Loss is : 0.16490959570548366 & Test Loss is : 0.165628823267
For EPOCH No : 11 Train Loss is : 0.16473198368710804 & Test Loss is : 0.165476501326
For EPOCH No : 12 Train Loss is : 0.16460232428460345 & Test Loss is : 0.165368425348
For EPOCH No : 13 Train Loss is : 0.16450690802866458 & Test Loss is : 0.165291495712
For EPOCH No : 14 Train Loss is : 0.1644362237785439 & Test Loss is : 0.1652366955048
```

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([[ -0.0148813 ,  0.01507473, -0.00228084,  0.00662734, -0.00637896,
          0.0118657 , -0.00721556,  0.00192055,  0.00992739,  0.00363929,
          0.00457632,  0.00354051,  0.00058866,  0.00285896,  0.00054483]]),
array([ -0.00601054]))
```

Plot epoch number vs train , test loss

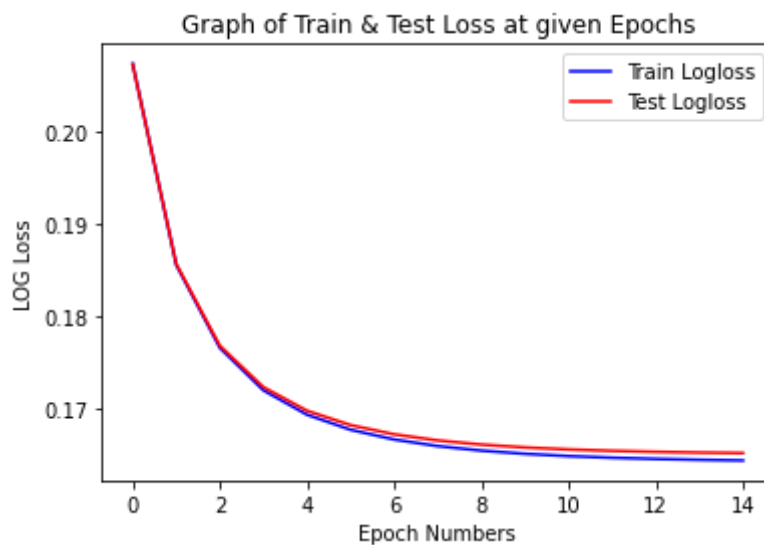
- epoch number on X-axis
- loss on Y-axis

```
import matplotlib.pyplot as plt

x = np.array([i for i in range(0, 15)])
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))

0.9506933333333334
0.94776
```

I have run the update function upto the 15 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 14th epoch

Thank you!!

✓ 0s completed at 1:18 PM

