parth.pandey13103447@gmail.com_9

May 6, 2020

1 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

[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 import linear_model

import matplotlib.pyplot as plt
```

Creating custom dataset

[3]:
```
# 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.make_classification.html) for more details
```

[4]:
```
X.shape, y.shape
```

[4]:
```
((50000, 15), (50000,))
```

Splitting data into train and test

[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)
```

[6]:
```
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

[6]: ((37500, 15), (37500,), (12500, 15), (12500,))

2 SGD classifier

[7]:
```
# 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/sklearn.linear_model.SGDClassifier.html)
```

[7]:
```
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)
```

[8]:
```
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: 0.02 seconds.
-- Epoch 4
Norm: 1.02, NNZs: 15, Bias: -0.658292, T: 150000, Avg. loss: 0.382083
Total training time: 0.03 seconds.
-- Epoch 5
Norm: 1.04, NNZs: 15, Bias: -0.719528, T: 187500, Avg. loss: 0.380486
Total training time: 0.04 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: 0.07 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
```

[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='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)
```

[9]:
```
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
```

[9]:
```
(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]))
```

2.1 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 co
```

```
    $dw^{(t)} = x_n(y_n -  ((w^{(t)})^{T} x_n+b^{t}))- \frac{ }{N}w^{(t)})$ <br>

    - Calculate the gradient of the intercept (write your code in <font color='blue'> def grad

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

    - Update weights and intercept (check the equation number 32 in the above mentioned <a hre
    $w^{(t+1)}← w^{(t)}+ (dw^{(t)}) $<br>

    $b^{(t+1)}←b^{(t)}+ (db^{(t)}) $
- calculate the log loss for train and test with the updated weights (you can check the python
- And if you wish, you can compare the previous loss and the current loss, if it is not updatin
    you can stop the training
- append this loss in the list ( this will be used to see how loss is changing for each epoch a
```

Initialize weights

[10]:
```python
def initialize_weights(dim):
    ''' In this function, we will initialize our weights and bias'''
    #initialize the weights to zeros array of (dim,1) 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
```

[11]:
```python
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

[12]:
```python
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)
```

[12]: True

Compute sigmoid

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

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

Grader function - 2

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

```
[14]: True
```

Compute loss

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

```
[16]: def logloss(y_true,y_pred):
          '''In this function, we will compute log loss '''
          # Convert to numpy
          if type(y_true) == list:
              y_true = np.array(y_true)

          if type(y_pred) == list:
              y_pred = np.array(y_pred)

          l1 = np.log10(y_pred)
          l2 = np.log10(1-y_pred)
          n = y_true.shape[0]
          loss = np.sum((l1 * y_true) + (l2 * (1-y_true)))
          return loss * -(1/n)
```

Grader function - 3

```
[17]: def grader_logloss(true,pred):
          loss=logloss(true,pred)
          print(loss)
          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)
```

```
0.07644900402910389
```

```
[17]: True
```

Compute gradient w.r.to 'w'

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

[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,x) + b) - (alpha/N)* w)
    return dw
```

Grader function - 4

[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)
```

[19]:
```
True
```

Compute gradient w.r.to 'b'

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

[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,x) + b)
    return db
```

Grader function - 5

[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)
```

[21]: True

Implementing logistic regression

```
[31]: 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 epochr
        # 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␣
 ↪w,b
        #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 w,b
        #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 then stop the process and return w,b
    w,b = initialize_weights(X_train[0])
    N = X_train.shape[0]
    train_loss = 0
    test_loss  = 0
    for epoch in range(1,epochs+1):
        # Updating the weights and biases
        for x,y in zip(X_train,y_train):
            # Here the update function has + sign because
            # we already considered the - sign in gradient calculations        ␣
 ↪
            w = w + (eta0 * gradient_dw(x,y,w,b,alpha,N))
            b = b + (eta0 * gradient_db(x,y,w,b))
        # predicting the output for x_train
        train_predict = []
        for x_q in X_train:
            y_q = sigmoid(np.dot(w,x_q) + b)
            train_predict.append(y_q)

        # predicting the output for x_test
```

```
        test_predict = []
        for x_q in  X_test:
            y_q = sigmoid(np.dot(w,x) + b)
            test_predict.append(y_q)

        # finding loss for each epoch
        old_test_loss = test_loss
        old_train_loss = train_loss
        print('Train Loss')
        train_loss = logloss(y_true=y_train, y_pred=train_predict)
        print('Test Loss')
        test_loss  = logloss(y_true=y_test,  y_pred=test_predict )
        statement = 'Epoch = {} Train Loss = {} Test Loss = {}'.
 ↪format(epoch,train_loss,test_loss)
        print(statement)
        train_loss_list.append(train_loss)
        test_loss_list.append(test_loss)
        epoch_list.append(epoch)

        #  Defining the stopping conditions
        if abs(test_loss - old_test_loss) <= 10**-5  or abs(train_loss -␣
 ↪old_train_loss) <= 10**-5:
            return w,b
    return w,b
```

[23]:
```
alpha=0.0001
eta0=0.0001
N=len(X_train)
epochs=50
train_loss_list = []
test_loss_list  = []
epoch_list = []
w,b=train(X_train,y_train,X_test,y_test,epochs,alpha,eta0)
```

```
Train Loss
Test Loss
Epoch = 1 Train Loss = 0.1754692621456728 Test Loss = 0.2960561311694524
Train Loss
Test Loss
Epoch = 2 Train Loss = 0.16868174428744456 Test Loss = 0.29125313128923724
Train Loss
Test Loss
Epoch = 3 Train Loss = 0.16639953373764638 Test Loss = 0.2900200962120205
Train Loss
Test Loss
Epoch = 4 Train Loss = 0.1653740489745895 Test Loss = 0.28949152437298104
Train Loss
```

```
Test Loss
Epoch = 5 Train Loss = 0.16486122000648354 Test Loss = 0.2891709807899862
Train Loss
Test Loss
Epoch = 6 Train Loss = 0.16459114503615363 Test Loss = 0.28894337994285557
Train Loss
Test Loss
Epoch = 7 Train Loss = 0.1644447987233685 Test Loss = 0.2887738709447197
Train Loss
Test Loss
Epoch = 8 Train Loss = 0.1643641152081749 Test Loss = 0.2886464436621983
Train Loss
Test Loss
Epoch = 9 Train Loss = 0.16431912309464214 Test Loss = 0.28855070356718565
Train Loss
Test Loss
Epoch = 10 Train Loss = 0.16429382914512278 Test Loss = 0.2884789063139989
Train Loss
Test Loss
Epoch = 11 Train Loss = 0.16427952012680933 Test Loss = 0.28842512745037063
Train Loss
Test Loss
Epoch = 12 Train Loss = 0.16427138330906932 Test Loss = 0.2883848560232998
```

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

[24]:
```python
# these are the results we got after we implemented sgd and found the optimal␣
↪weights and intercept
w-clf.coef_, b-clf.intercept_
```

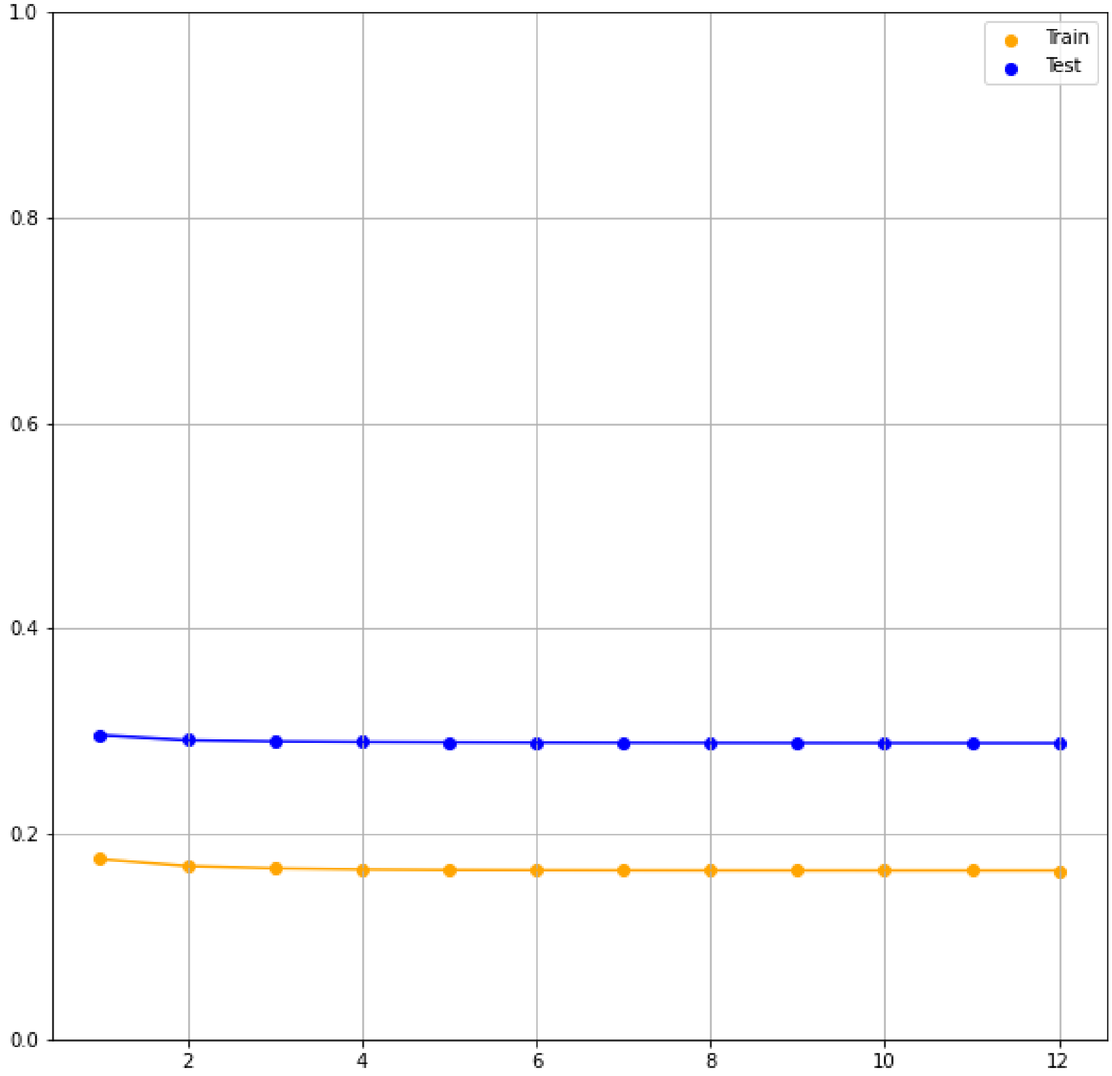
[24]:
```
(array([[-0.00263921,  0.00638027,  0.00156229, -0.00330247, -0.00780605,
          0.00713878,  0.00715471,  0.00315014,  0.01034534, -0.00931498,
         -0.00031121, -0.00315245,  0.00027729,  0.00040767, -0.00017047]]),
 array([-0.01498811]))
```

Plot epoch number vs train , test loss

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

[29]:
```python
_ ,ax = plt.subplots(1,1,figsize=(10,10))
ax.scatter(epoch_list,train_loss_list,color='orange',label='Train')
ax.plot(epoch_list,train_loss_list,color='orange')
ax.scatter(epoch_list,test_loss_list,color='blue',label='Test')
ax.plot(epoch_list,test_loss_list,color='blue')
ax.set_ylim(0,1)
```

```
plt.legend()
plt.grid()
plt.show()
```



[30]:
```
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)
```

```python
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))
```

```
0.9542933333333333
0.95192
```