9_Implement_SGD_Classifier_with_Logloss_and_L2_regularization_U

November 2, 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

[1]:
```
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

[2]:
```
# 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
```

[3]:
```
X.shape, y.shape
```

[3]:
```
((50000, 15), (50000,))
```

Splitting data into train and test

[4]:
```
#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)
```

[5]:
```
# Standardizing the data.
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
```

```
x_test = scaler.transform(x_test)
```

[6]:
```
x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

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

[6]:
```
```

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: 0.70, 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: 0.03 seconds.
-- Epoch 3
Norm: 1.26, NNZs: 15, Bias: -0.902742, T: 112500, Avg. loss: 0.415724
Total training time: 0.04 seconds.
-- Epoch 4
Norm: 1.43, NNZs: 15, Bias: -1.003816, T: 150000, Avg. loss: 0.400895
Total training time: 0.05 seconds.
-- Epoch 5
Norm: 1.55, NNZs: 15, Bias: -1.076296, T: 187500, Avg. loss: 0.392879
Total training time: 0.06 seconds.
-- Epoch 6
Norm: 1.65, NNZs: 15, Bias: -1.131077, T: 225000, Avg. loss: 0.388094
```

```
Total training time: 0.07 seconds.
-- Epoch 7
Norm: 1.73, NNZs: 15, Bias: -1.171791, T: 262500, Avg. loss: 0.385077
Total training time: 0.08 seconds.
-- Epoch 8
Norm: 1.80, NNZs: 15, Bias: -1.203840, T: 300000, Avg. loss: 0.383074
Total training time: 0.09 seconds.
-- Epoch 9
Norm: 1.86, NNZs: 15, Bias: -1.229563, T: 337500, Avg. loss: 0.381703
Total training time: 0.10 seconds.
-- Epoch 10
Norm: 1.90, NNZs: 15, Bias: -1.251245, T: 375000, Avg. loss: 0.380763
Total training time: 0.11 seconds.
-- Epoch 11
Norm: 1.94, NNZs: 15, Bias: -1.269044, T: 412500, Avg. loss: 0.380084
Total training time: 0.12 seconds.
-- Epoch 12
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
```

[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([[-0.89007184,  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
```

2.1 Implement Logistic Regression with L2 regularization Using SGD: without using sklearn

- 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 gradi

       $ 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 updati
    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]:
```
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
    b = 0
    w = np.zeros_like(dim)

    return w,b
```

[10]:
```
```

[11]:
```
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]:
```
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
$logloss = -1 * \frac{1}{n}\Sigma_{foreachYt,Y_{pred}}(Ytlog10(Y_{pred}) + (1-Yt)log10(1-Y_{pred}))$

[15]:
```
import math
def logloss(y_true,y_pred):
    '''In this function, we will compute log loss '''
    l=0
    #print(y_pred)
    for i in range(len(y_true)):
      l+= ( (y_true[i]*(math.log10(y_pred[i]))) + ((1-y_true[i])*(math.
 ↪log10(1-y_pred[i]))) )

    loss = (-1/len(y_true))*l
    return loss
```

Grader function - 3

[16]:
```
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)
```

[16]:
```
True
```

Compute gradient w.r.to 'w'
$dw^{(t)} = x_n(y_n((w^{(t)})^T x_n + b^t)) - \frac{}{N} w^{(t)}$

[16]:
```
```

[17]:
```
def gradient_dw(x,y,w,b,alpha,N):
    '''In this function, we will compute the gardient w.r.to w '''
    #print(x.shape,w.shape)
    z = np.dot(x,w) + b
    #print(z)
    #print(((y-sigmoid(z))),x)
    a = ((y-sigmoid(z))*x)
    #print(a)
    b = ((alpha/N)*w)
    dw = a-b
    return dw
```

Grader function - 4

[18]:
```
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)
```

[18]:
```
True
```

Compute gradient w.r.to 'b'
$ dbˆ{(t)} = y_n- ((w^{(t)})^{T} x_n+bˆ{t})$

[19]:
```
 def gradient_db(x,y,w,b):
     '''In this function, we will compute gradient w.r.to b '''
     z = ( (np.dot(x,w)) + b )
     db = (y-(sigmoid(z)))
     return (db)
```

Grader function - 5

[20]:
```
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)
```

[20]:
```
True
```

Implementing logistic regression

[21]:
```
def pred(w,b, X):
    N = len(X)
    predict = []
    w = list(w)
    for i in range(N):
        z = np.dot(X[i],w) + b
        predict.append(sigmoid(z))
    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))
```

```
1.1978933333333333
1.19864
```

[22]:
```
from tqdm import tqdm
from collections import OrderedDict
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␣
 ↪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=len(x_train)
    tr_loss,ts_loss = [],[]
    #loss = OrderedDict()
    initial_loss = 0
    for epoch in tqdm(range(epochs)):
      #print(epoch)
      for i in range(x_train.shape[0]):
        dw = gradient_dw(x_train[i],y_train[i],w,b,alpha,N)
        db = gradient_db(x_train[i],y_train[i],w,b)
        w += (eta0*dw)
        b += (eta0*db)
      y_pred_tr,y_pred_ts = pred(w,b,x_train),pred(w,b,x_test)
      tr_loss.append(logloss(y_train,y_pred_tr))
      ts_loss.append(logloss(y_test,y_pred_ts))
      #loss[logloss(y_test,y_pred_ts)] = w
      #print(y_train,y_pred_tr)
      #print(y_test,y_pred_ts)
      #ts_los = logloss(y_test,y_pred_ts)
      #if abs(initial_loss - ts_los  ) <= (10**(-3)):
      #  print('(ts_los - initial_loss):',(ts_los, initial_loss),abs(ts_los -␣
↪initial_loss),'\n')
      #  break
      #print('(ts_los - initial_loss):',(ts_los, initial_loss),abs(ts_los -␣
↪initial_loss),'\n')
      #initial_loss = ts_los
    return w,b,tr_loss,ts_loss
```

[22]:

[23]:
```
alpha=0.0001
eta0=0.0001
N=len(x_train)
epochs=14
w,b,tr_loss,ts_loss=train(x_train,y_train,x_test,y_test,epochs,alpha,eta0)
```

```
100%|| 14/14 [00:20<00:00,  1.49s/it]
```

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

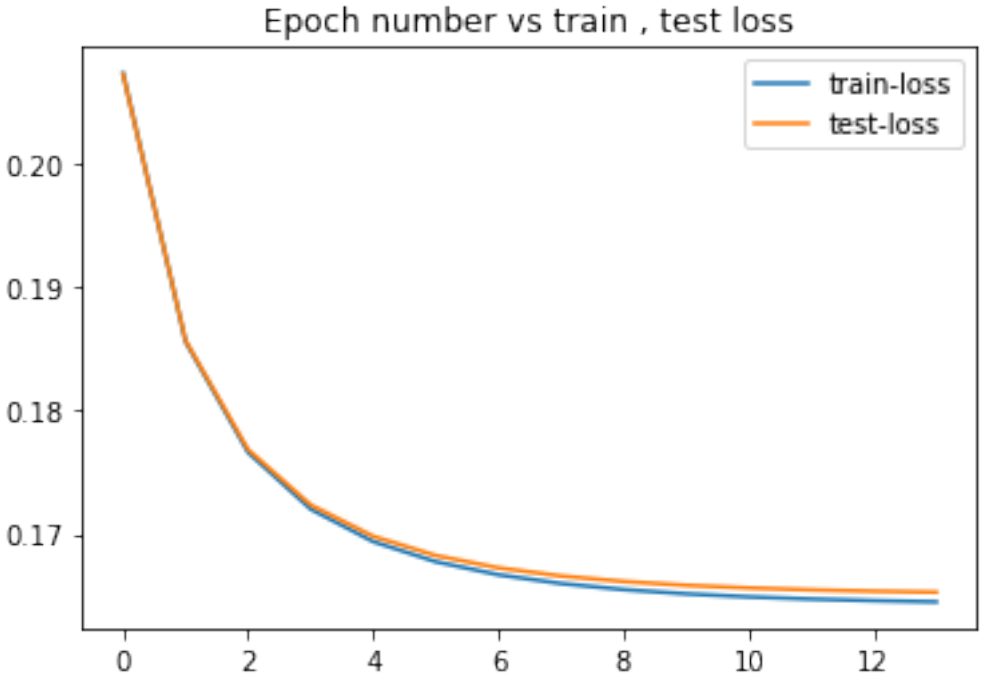
[24]:
```
# these are the results we got after we implemented sgd and found the optimal␣
↪weights and intercept
w-clf.coef_, b-clf.intercept_
```

[24]:
```
(array([[-4.75139040e-03,  7.60245639e-03,  1.85102713e-03,
          6.50362355e-05,  1.54498740e-03,  2.34086809e-03,
         -9.09928936e-04,  2.16124544e-03,  5.21959720e-03,
         -4.49834999e-03,  1.23628554e-03,  2.54417563e-03,
          1.74962845e-03, -1.28756176e-03,  1.05365463e-03]]),
 array([0.00279952]))
```

Plot epoch number vs train , test loss

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

[25]:
```
import matplotlib.pyplot as plt
plt.title('Epoch number vs train , test loss')
plt.plot(tr_loss,label='train-loss')
plt.plot(ts_loss,label='test-loss')
plt.legend()
plt.show()
```



[25]: