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 [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 import linear_model
import math
```

Creating custom dataset

In [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
```

In [3]:

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

Out[3]:

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

Splitting data into train and test

In [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)
```

In [5]:

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

Out[5]:

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

SGD classifier

In [6]:

```
# 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='l
2', 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)
```

Out[6]:

```
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)
```

In [7]:

```
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: 0.03 seconds.
-- Epoch 2
Norm: 0.91, NNZs: 15, Bias: -0.472747, T: 75000, Avg. loss: 0.394686
Total training time: 0.05 seconds.
-- Epoch 3
Norm: 0.98, NNZs: 15, Bias: -0.580082, T: 112500, Avg. loss: 0.385711
Total training time: 0.07 seconds.
-- Epoch 4
Norm: 1.02, NNZs: 15, Bias: -0.658292, T: 150000, Avg. loss: 0.382083
Total training time: 0.08 seconds.
-- Epoch 5
Norm: 1.04, NNZs: 15, Bias: -0.719528, T: 187500, Avg. loss: 0.380486
Total training time: 0.09 seconds.
-- Epoch 6
Norm: 1.05, NNZs: 15, Bias: -0.763409, T: 225000, Avg. loss: 0.379578
Total training time: 0.11 seconds.
-- Epoch 7
Norm: 1.06, NNZs: 15, Bias: -0.795106, T: 262500, Avg. loss: 0.379150
Total training time: 0.12 seconds.
-- Epoch 8
Norm: 1.06, NNZs: 15, Bias: -0.819925, T: 300000, Avg. loss: 0.378856
Total training time: 0.14 seconds.
-- Epoch 9
Norm: 1.07, NNZs: 15, Bias: -0.837805, T: 337500, Avg. loss: 0.378585
Total training time: 0.15 seconds.
-- Epoch 10
Norm: 1.08, NNZs: 15, Bias: -0.853138, T: 375000, Avg. loss: 0.378630
Total training time: 0.16 seconds.
Convergence after 10 epochs took 0.16 seconds
```

Out[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)
```

In [8]:

```
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[8]:

```
(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())

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

- 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 − σ((w^{(t)})^{T} x_n+b^{t}))- \frac{λ}{N}w^{(t)})
    - Calculate the gradient of the intercept (write your code in def gradient_db()) check this

      db^{(t)} = y_n- σ((w^{(t)})^{T} x_n+b^{t}))
    - Update weights and intercept (check the equation number 32 in the above mentioned pdf):
      w^{(t+1)}← w^{(t)}+α(dw^{(t)})

      b^{(t+1)}←b^{(t)}+α(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 [9]:

```
import random
import matplotlib.pyplot as plt

import numpy as np

def sigmoid(w, X,b):
    z=np.dot(X,w.T)+b
    value=1/(1+np.exp(-z))
    return value
```

In [10]:

```
def sigmoid2(x):

    return 1.0 / (1 + np.exp(-x))

def next_batch(X, y, batchSize):

    for i in np.arange(0, X.shape[0], batchSize):

        yield (X[i:i + batchSize], y[i:i + batchSize])
```

In [11]:

```
def initialize_weights(X_train):
    W = np.zeros_like(X_train[0])
    return W
```

In [12]:

```
def logloss(pred_test, y_test): #test
    log_loss = y_test*np.log(pred_test)+(1-y_test)*np.log(1-pred_test)
    return log_loss
```

In [13]:

```
def gradient_db(batchX, error):
    gradient2 = batchX.T.dot(error) / batchX.shape[0]
    return gradient2
```

In [14]:

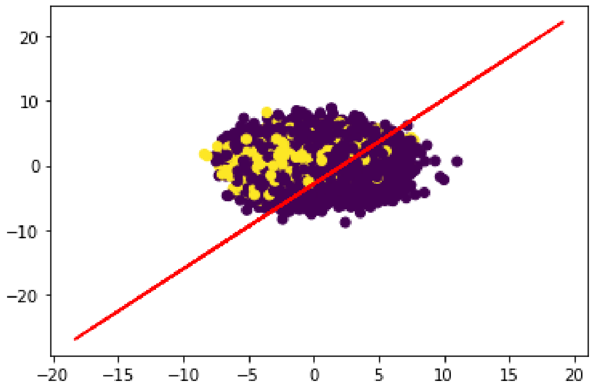
```
def gradient_dw(batchX, error, alpha, W):
    gradient1 = batchX.T.dot(error) / batchX.shape[0] -((alpha)/batchX.shape[0])*W
    return gradient1
```

In [21]:

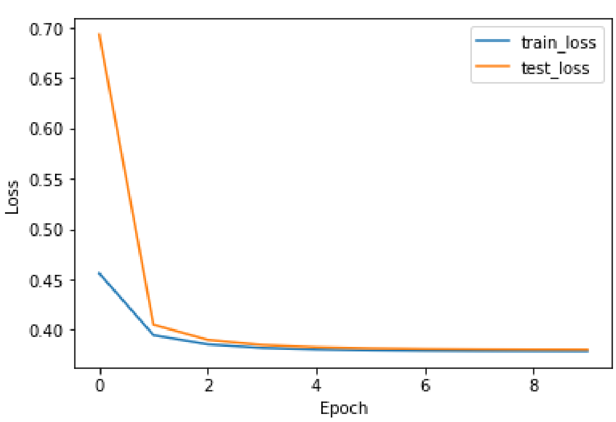
```
W=initialize_weights(X_train)
b = 0
eta0  = 0.0001
alpha = 0.0001
batch_size=1


lossHistory = []
losstest =[]


for epoch in np.arange(0, 10):
    ep_test =[]
    ep_Loss = []
    pred_test=sigmoid2(X_test.dot(W)+b)

    loss_test = logloss(pred_test,y_test)


    ep_test.append(abs(loss_test))

    for (batchX, batchY) in next_batch(X_train, y_train, batch_size):

        preds = sigmoid2(batchX.dot(W)+b)

        error = preds - batchY
        s=abs(error)

        loss = batchY*np.log(preds)+(1-batchY)*np.log(1-preds)#training

        ep_Loss.append(abs(loss))

        gradient = gradient_dw(batchX, error, alpha, W)

        W +=   -eta0 * gradient
        b += -eta0*error

    lossHistory.append(np.average(ep_Loss))
    losstest.append(np.average(ep_test))
    print("Epoch= {}".format(epoch))
    print("Training Loss= {}".format(np.average(ep_Loss)))
    print("Test Loss = {}".format(np.average(ep_test)))

Y = (-W[0] - (W[1] * X_train)) / W[2]

plt.figure()
plt.scatter(X[:, 0], X[:, 1], marker="o", c=y)
plt.plot(X_train, Y, "r-")



fig = plt.figure()
plt.plot(np.arange(0,10), lossHistory,label = "train_loss")
```

```
plt.plot(np.arange(0,10), losstest,label = "test_loss")
fig.suptitle("Epoch vs Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.show()
```

```
Epoch= 0
Training Loss= 0.4561018272057361
Test Loss = 0.6931471805599453
Epoch= 1
Training Loss= 0.39468462281533234
Test Loss = 0.4051439504935396
Epoch= 2
Training Loss= 0.3855816920714233
Test Loss = 0.39005133142859516
Epoch= 3
Training Loss= 0.38202939082576426
Test Loss = 0.385002796881333
Epoch= 4
Training Loss= 0.3803476760958345
Test Loss = 0.38272761827702007
Epoch= 5
Training Loss= 0.37948657557421145
Test Loss = 0.381582925323185
Epoch= 6
Training Loss= 0.3790278703507683
Test Loss = 0.3809757253845749
Epoch= 7
Training Loss= 0.37877803660043274
Test Loss = 0.38064418714211234
Epoch= 8
Training Loss= 0.3786401328811608
Test Loss = 0.38045997505785634
Epoch= 9
Training Loss= 0.37856336687784825
Test Loss = 0.38035642910416334
```



Epoch vs Loss



In [19]:

```
def pred(w,b, X):
```

```python
def pred(W,b, X):
    N = len(X)
    predict = []
    for i in range(N):
        if sigmoid(W, X[i], b) >= 0.5:
            predict.append(1)
        else:
            predict.append(0)
    return np.array(predict)
print('Train')
print(1-np.sum(y_train - pred(W,b,X_train))/len(X_train))
print('Test')
print(1-np.sum(y_test  - pred(W,b,X_test))/len(X_test))
```

```
Train
0.9554133333333333
Test
0.95296
```

In [48]:

```python
W-clf.coef_, clf.coef_.shape, b-clf.intercept_
```

Out[48]:

```
(array([[-6.99735029e-05,  5.59907877e-03,  2.54469960e-03,
         -3.11029141e-03, -4.04381494e-03,  5.45256230e-03,
          6.87579892e-03,  2.34677210e-03,  8.81868548e-03,
         -1.08906686e-02, -1.68767678e-03, -1.94683987e-03,
          1.69748804e-03,  4.59826266e-04, -5.27691669e-04]]),
 (1, 15),
 array([0.00245376]))
```

In [0]:

```python

```

In [0]:

```python

```