In [1]:

```
import numpy as np
import pandas as pd
from sklearn.datasets import make_classification
```

In [2]:

```
X, y = make_classification(n_samples=50000, n_features=15, n_informative=10, n_redur
                           n_classes=2, weights=[0.7], class_sep=0.7, random_state=1
```

In [3]:

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

Out[3]:

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

In [4]:

```
from sklearn.model_selection import train_test_split
```

In [5]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_sta
```

In [6]:

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

Out[6]:

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

In [7]:

```
from sklearn import linear_model
```

In [8]:

```
# 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
clf
```

Out[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='consta
nt',
              loss='log', max_iter=1000, n_iter_no_change=5, n_jobs=No
ne,
              penalty='l2', power_t=0.5, random_state=15, shuffle=Tru
e,
              tol=0.001, validation_fraction=0.1, verbose=2, warm_star
t=False)
```

In [9]:

```
clf.fit(X=X_train, y=y_train)
```

```
-- 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: 0.04 seconds.
-- Epoch 4
Norm: 1.02, NNZs: 15, Bias: -0.658292, T: 150000, Avg. loss: 0.382083
Total training time: 0.05 seconds.
-- Epoch 5
Norm: 1.04, NNZs: 15, Bias: -0.719528, T: 187500, Avg. loss: 0.380486
Total training time: 0.06 seconds.
-- Epoch 6
Norm: 1.05, NNZs: 15, Bias: -0.763409, T: 225000, Avg. loss: 0.379578
Total training time: 0.07 seconds.
-- Epoch 7
Norm: 1.06, NNZs: 15, Bias: -0.795106, T: 262500, Avg. loss: 0.379150
Total training time: 0.08 seconds.
-- Epoch 8
Norm: 1.06, NNZs: 15, Bias: -0.819925, T: 300000, Avg. loss: 0.378856
Total training time: 0.10 seconds.
-- Epoch 9
Norm: 1.07, NNZs: 15, Bias: -0.837805, T: 337500, Avg. loss: 0.378585
Total training time: 0.12 seconds.
-- Epoch 10
Norm: 1.08, NNZs: 15, Bias: -0.853138, T: 375000, Avg. loss: 0.378630
Total training time: 0.13 seconds.
Convergence after 10 epochs took 0.13 seconds
```

Out[9]:

```
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='consta
nt',
              loss='log', max_iter=1000, n_iter_no_change=5, n_jobs=No
ne,
              penalty='l2', power_t=0.5, random_state=15, shuffle=Tru
e,
              tol=0.001, validation_fraction=0.1, verbose=2, warm_star
t=False)
```

In [10]:

```
clf.coef_, clf.coef_.shape, clf.intercept_
```

Out[10]:

```
(array([[-0.42336692,  0.18547565, -0.14859036,  0.34144407, -0.208186
7 ,
          0.56016579, -0.45242483, -0.09408813,  0.2092732 ,  0.180841
26,
          0.19705191,  0.00421916, -0.0796037 ,  0.33852802,  0.022667
21]]),
 (1, 15),
 array([-0.8531383]))
```

Implement Logistc Regression with L2 regularization Using SGD: without using sklearn

Instructions

- Load the datasets(train and test) into the respective arrays
- Initialize the weight_vector and intercept term randomly
- Calculate the initlal log loss for the train and test data with the current weight and intercept and store it in a list
- 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
    - Calculate the gradient of the intercept check this (https://drive.google.com/file/d/1nQ08-XY4zvOLzRX-lGf8EYB5arb7-m1H/view?usp=sharing)
    - Update weights and intercept (check the equation number 32 in the above mentioned pdf (https://drive.google.com/file/d/1nQ08-XY4zvOLzRX-lGf8EYB5arb7-m1H/view?usp=sharing)):
      $w^{(t+1)} \leftarrow (1 - \frac{\alpha\lambda}{N})w^{(t)} + \alpha x_n(y_n - \sigma((w^{(t)})^T x_n + b^t))$
      $b^{(t+1)} \leftarrow (b^t + \alpha(y_n - \sigma((w^{(t)})^T x_n + b^t))$
    - 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)
- Plot the train and test loss i.e on x-axis the epoch number, and on y-axis the loss
- **GOAL**: 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

In [11]:

```
''' initalizing the default values '''
w = np.zeros_like(X_train[0])
b = 0
eta0 = 0.0001
alpha = 0.0001
N = len(X_train)
```

In [12]:

```
 len(w), w
```

Out[12]:

```
(15, array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
0.]))
```

In [13]:

```
# write your code to implement SGD as per the above instructions
# please choose the number of iternations on your own
```

In [14]:

```
import math
''' Function for calculating loss'''
def compute_log_loss(ground_truth ,model_predection):
    # intitalising the variables, for calculating sum
    total = 0

    for item in zip(ground_truth ,model_predection):
        ''' Applying the formula '''
#         print(item)
        total += ( ( item[0]*math.log(item[1],10) ) + ((1.0-item[0])*math.log(1.0-it
    return (-1)*(1.0/len(ground_truth))*(total)
```

In [15]:

```
''' sigmod function '''
def sigmoid(w,X,b):
#     print(w.shape, X.shape)
    z = np.dot(X, w)+b
    return 1/(1+np.exp(-z))
```

In [16]:

```
''' Model for predictions '''
def pred(w,b, X):
    N = len(X)
    predict = []
    for i in range(N):
        predict.append(sigmoid(w, X[i], b))
    return np.array(predict)
```

In [17]:

```
from tqdm import tqdm
import random

'''  intintializing the values '''
total_records = len(X_train)
train_loss_list = []
test_loss_list  = []
reducing_weights = [w]
reducing_bais    = [b]
''' Looping through each epoch '''
for each_point in tqdm(range(0, 70)):
    ''' Iterating for each batch '''
    for each_batch in range(0, total_records):
        ''' Getting random index '''
        get_random_input =each_batch
        ''' calculating the error '''
        error = y_train[get_random_input] - sigmoid(w.T,X_train[get_random_input],b)
        ''' Update weight vector '''
        w = (1- ((alpha*eta0)/total_records))*w + (alpha*X_train[get_random_input]*
                                                    (error))
        ''' update intercept'''
        b = (b+ alpha*(y_train[get_random_input] - sigmoid(w.T,X_train[get_random_ir
    ''' stroing the optimized weigths and bais for each epoch'''
    reducing_weights.append(w)
    reducing_bais.append(b)
    print('current weight is')
    print(w)
    print('current bias is {}'.format(b))
    ''' calculate model predection on train data  '''
    get_train_prediction = pred(w.T,b, X_train)
    ''' calculate train loss '''
    get_train_loss = compute_log_loss(y_train ,get_train_prediction)
    train_loss_list.append(get_train_loss)
    print(' training error is {} for {} record in train dataset'.format(get_train_lc
    ''' calculate model predection on test data  '''
    get_train_prediction = pred(w.T,b, X_test)
    ''' calculate test loss '''
    get_train_loss = compute_log_loss(y_test ,get_train_prediction)
    test_loss_list.append(get_train_loss)
    print(' test error is {} for {} record in test dataset'.format(get_train_loss,ge
```

```
 training error is 0.16426033881448804 for 37499 record in train datas
et
 test error is 0.16512554382245748 for 37499 record in test dataset
current weight is
[-4.29756022e-01  1.93023835e-01 -1.48464492e-01  3.38103415e-01
 -2.21229065e-01  5.69932660e-01 -4.45183637e-01 -8.99209545e-02
  2.21804886e-01  1.73809503e-01  1.98727752e-01 -5.59489671e-04
 -8.13106733e-02  3.39094300e-01  2.29785009e-02]
current bias is -0.8918931627961899

100%|██████████| 70/70 [02:31<00:00,  2.17s/it]

 training error is 0.16426033881447086 for 37499 record in train datas
```

```
et
 test error is 0.16512554382240752 for 37499 record in test dataset
```

Learned from mistake's

```
- Do not use get_random_input =random.randrange(0, total_records) because mo
del will not
  learn from all point in  vanilla sgd
- To overcome it get_random_input = each_batch
```

In [18]:

```
train_loss_list,test_loss_list
```

```
 0.16436411522525887,
 0.1643191231082832,
 0.1642938291559788,
 0.16427952013540809,
 0.16427138331585123,
 0.16426673469647687,
 0.1642640668101494,
 0.16426252835732985,
 0.16426163646238537,
 0.16426111618838604,
 0.1642608104485638,
 0.1642606291821092,
 0.16426052056735924,
 0.1642604546634997,
 0.16426041408844078,
 0.16426038869248608,
 0.16426037250710632,
 0.16426036199220245,
 0.16426035502610167,
```

In [19]:

```
''' Displaying the final weights & intecept '''
w,b
```

Out[19]:

```
(array([-4.29756022e-01,  1.93023835e-01, -1.48464492e-01,  3.38103415
e-01,
       -2.21229065e-01,  5.69932660e-01, -4.45183637e-01, -8.99209545
e-02,
        2.21804886e-01,  1.73809503e-01,  1.98727752e-01, -5.59489671
e-04,
       -8.13106733e-02,  3.39094300e-01,  2.29785009e-02]),
 -0.8918931627961899)
```

In [20]:

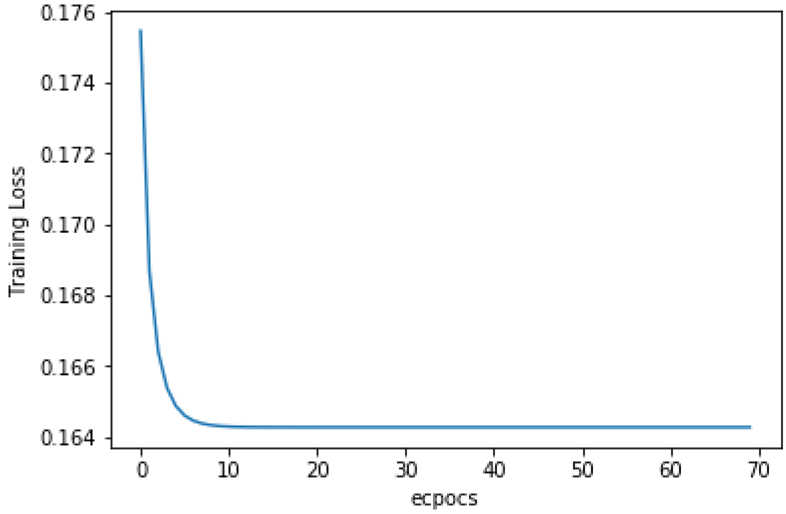
```
import matplotlib.pyplot as plt
GET_Weights_NUMPY = np.array(reducing_weights)
''' Each feature weights are is updating '''
for each_loop in range(GET_Weights_NUMPY.shape[0]):
    plt.plot(GET_Weights_NUMPY[:,])
    plt.xlabel('ecpocs')
    plt.ylabel('weights value')
plt.show()
''' Intercept is updating '''
plt.plot(reducing_bais)
plt.xlabel('ecpocs')
plt.ylabel('bais value')
plt.show()
```

```
<Figure size 640x480 with 1 Axes>

<Figure size 640x480 with 1 Axes>
```

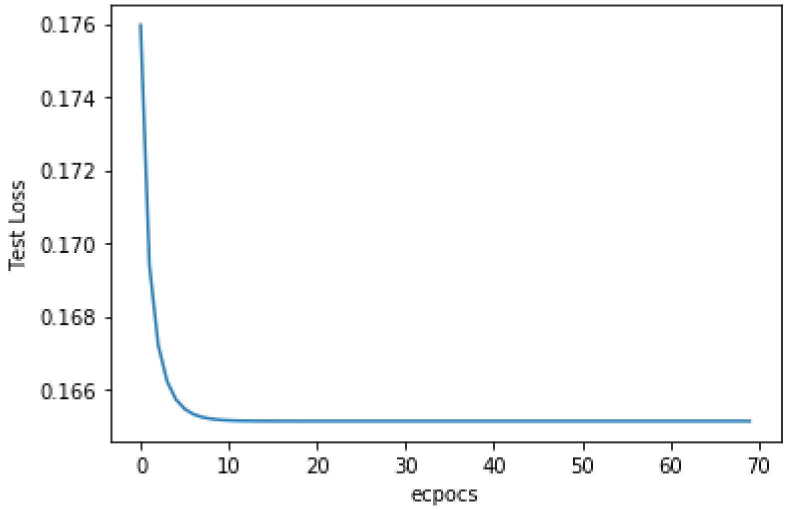
In [21]:

```
''' Plotting the training loss'''
plt.plot(train_loss_list)
plt.xlabel('ecpocs')
plt.ylabel('Training Loss')
plt.show()
```



In [22]:

```
''' Plotting the test loss '''
plt.plot(test_loss_list)
plt.xlabel('ecpocs')
plt.ylabel('Test Loss')
plt.show()
```



In [23]:

```
print(w-clf.coef_)
```

```
[[-0.00638911  0.00754818  0.00012587 -0.00334066 -0.01304236  0.00976
687
   0.00724119  0.00416717  0.01253169 -0.00703176  0.00167585 -0.00477
865
  -0.00170698  0.00056628  0.00031129]]
```

In [24]:

```
def get_pred(w,b, X):
    N = len(X)
    predict = []
    for i in range(N):
        if sigmoid(w, X[i], b) >= 0.5: # sigmoid(w,x,b) returns 1/(1+exp(-(dot(x,w)+
            predict.append(1)
        else:
            predict.append(0)
    return np.array(predict)
print(1-np.sum(y_train - get_pred(w,b,X_train))/len(X_train))
print(1-np.sum(y_test  - get_pred(w,b,X_test))/len(X_test))
```

```
0.95224
0.95
```