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_redundant=5,
                           n_classes=2, weights=[0.7], class_sep=0.7, random_state=15)
```

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_state=15)
```

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=15, penalty='l2',
tol=1e-3, verbose=2, learning_rate='constant')
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='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 [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: 0.02 seconds.
-- Epoch 2
Norm: 0.91, NNZs: 15, Bias: -0.472747, T: 75000, Avg. loss: 0.394686
Total training time: 0.03 seconds.
-- Epoch 3
Norm: 0.98, NNZs: 15, Bias: -0.580082, T: 112500, Avg. loss: 0.385711
Total training time: 0.05 seconds.
-- Epoch 4
Norm: 1.02, NNZs: 15, Bias: -0.658292, T: 150000, Avg. loss: 0.382083
Total training time: 0.06 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: 0.10 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: 0.12 seconds.
-- Epoch 9
Norm: 1.07, NNZs: 15, Bias: -0.837805, T: 337500, Avg. loss: 0.378585
Total training time: 0.14 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[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='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 [10]:

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

Out[10]:

```
(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 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
    - Update weights and intercept (check the equation number 32 in the above mentioned pdf):
      $w^{(t+1)} ← (1 − \frac{αλ}{N})w^{(t)} + αx_n(y_n − σ((w^{(t)})^{T} x_n+b^{t}))$
      $b^{(t+1)} ← (b^t + α(y_n - σ((w^{(t)})^{T} x_n+b^{t}))$
    - 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]:

```
# write your code to implement SGD as per the above instructions
# please choose the number of iternations on your own
import numpy as np
def sigmoid(w,x,b):
    return 1/(1+np.exp(-(w@x+b)))
```

In [12]:

```
def update_weights(X,y,w,b,lamda,alpha,N):
    w_new = (1-alpha*lamda/N)*w + alpha*X*(y-sigmoid(w,X.T,b))
    b_new = b + alpha*(y-sigmoid(w,X.T,b))
    return w_new,b_new
```

In [13]:

```
import math # you can free to change all these codes/structure
def compute_log_loss(A,n):# your code
    loss=0
    for Y in A:
        loss += Y[0]*math.log(Y[1])+(1-Y[0])*math.log(1-Y[1])
    loss = -loss/n
    return loss
```

In [14]:

```
w = np.zeros_like(X_train[0])
b = 0
lamda  = 0.0001
alpha = 0.0001
N = len(X_train)
```

In [15]:

```
print(w)
print(b)
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
0
```

In [16]:

```
y_pred = [sigmoid(w.T,x,b) for x in X]
loss = compute_log_loss(zip(y_train,y_pred),len(y_train))
print('\tIteration {} Loss: {}'.format(0,loss))
```

```
 Iteration 0 Loss: 0.6931471805594285
```

In [17]:

```
def next_batch(X, y, batchSize):
    # loop over our dataset `X` in mini-batches of size `batchSize`
    for i in np.arange(0, X.shape[0], batchSize):
        # yield a tuple of the current batched data and labels
        yield (X[i:i + batchSize], y[i:i + batchSize])
```

In [18]:

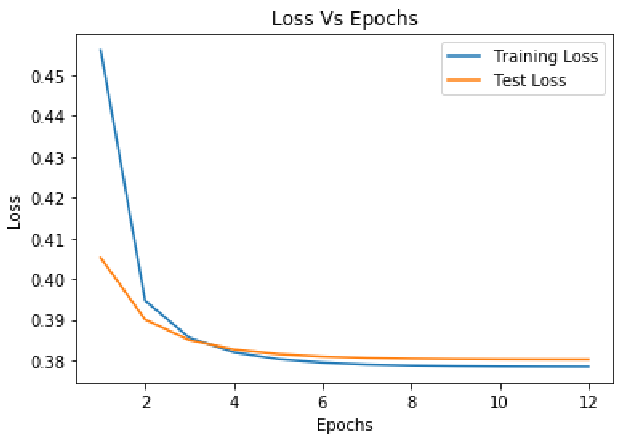
```
lossHistoryTrain = []
lossHistoryTest = []
epochs = [1,2,3,4,5,6,7,8,9,10,11,12]
for epoch in epochs:
    # initialize the total loss for the epoch
    epochLossTrain = []
    epochLossTest = []
    # loop over our data in batches
    for (batchX, batchY) in next_batch(X_train, y_train, 1):

        preds = sigmoid(w,batchX.T,b)
        loss = -(batchY*math.log(preds)+(1-batchY)*math.log(1-preds))
        epochLossTrain.append(loss)
        w, b = update_weights(batchX,batchY,w,b,lamda,alpha,N)

    avgLossTrain = np.average(epochLossTrain)
    lossHistoryTrain.append(avgLossTrain)
    print("iteration:{}".format(epoch))
    print("Training Loss:{}".format(avgLossTrain))
    y_pred = [sigmoid(w,x.T,b) for x in X_test]
    avgLossTest = compute_log_loss(zip(y_test,y_pred),len(y_test))
    lossHistoryTest.append(avgLossTest)
    print("Test Loss:{}".format(avgLossTest))
    print('='*75)
print('Final Weights:')
print(w)
print('Final Intercept:',b)
```

```
iteration:1
Training Loss:0.45610715612220937
Test Loss:0.40515077111050857
===========================================================================
iteration:2
Training Loss:0.3946913054134687
Test Loss:0.390056334234635
===========================================================================
iteration:3
Training Loss:0.38558684615141536
Test Loss:0.38500584031391916
===========================================================================
iteration:4
Training Loss:0.3820332220716334
Test Loss:0.38272919624702617
===========================================================================
iteration:5
Training Loss:0.38035055342868135
Test Loss:0.38158346823458383
===========================================================================
iteration:6
Training Loss:0.3794887755291076
Test Loss:0.3809755312339994
===========================================================================
iteration:7
Training Loss:0.3790295733085725
Test Loss:0.3806434506916476
```

```
==========================================================================
iteration:8
Training Loss:0.3787793587342117
Test Loss:0.38045882555929006
==========================================================================
iteration:9
Training Loss:0.3786411532813051
Test Loss:0.3803549569881218
==========================================================================
iteration:10
Training Loss:0.3785641439189919
Test Loss:0.3802960051315843
==========================================================================
iteration:11
Training Loss:0.3785209816370871
Test Loss:0.3802623039632136
==========================================================================
iteration:12
Training Loss:0.3784966943227099
Test Loss:0.38024291174496166
==========================================================================
Final Weights:
[[-0.42605234  0.19187028 -0.1470358   0.33812874 -0.21606161  0.56732498
  -0.44526597 -0.09091644  0.21964959  0.1715521   0.19676699  0.00103546
  -0.07935642  0.33892975  0.02250835]]
Final Intercept: [[-0.86857742]]
```

In [21]:

```python
import matplotlib.pyplot as plt
plt.plot(epochs,lossHistoryTrain)
plt.plot(epochs,lossHistoryTest)
plt.legend(['Training Loss','Test Loss'])
plt.title('Loss Vs Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()
```



In [20]:

```python
print(w-clf.coef_)
print(b-clf.intercept_)
```

```
[[-0.00268543  0.00639463  0.00155456 -0.00331533 -0.00787491  0.0071592
   0.00715886  0.00317169  0.01037639 -0.00928916 -0.00028492 -0.00318369
   0.00024727  0.00040174 -0.00015886]]
[[-0.01543912]]
```