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_s
tate=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, rand
om_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’ sch
edules.

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=None,
       n_iter=None, 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: 0.76, NNZs: 15, Bias: -0.314605, T: 37500, Avg. loss: 0.455801
Total training time: 0.02 seconds.
-- Epoch 2
Norm: 0.92, NNZs: 15, Bias: -0.469578, T: 75000, Avg. loss: 0.394737
Total training time: 0.03 seconds.
-- Epoch 3
Norm: 0.98, NNZs: 15, Bias: -0.580452, T: 112500, Avg. loss: 0.385561
Total training time: 0.03 seconds.
-- Epoch 4
Norm: 1.02, NNZs: 15, Bias: -0.660824, T: 150000, Avg. loss: 0.382161
Total training time: 0.05 seconds.
-- Epoch 5
Norm: 1.04, NNZs: 15, Bias: -0.717218, T: 187500, Avg. loss: 0.380474
Total training time: 0.06 seconds.
-- Epoch 6
Norm: 1.06, NNZs: 15, Bias: -0.761816, T: 225000, Avg. loss: 0.379481
Total training time: 0.06 seconds.
-- Epoch 7
Norm: 1.06, NNZs: 15, Bias: -0.793932, T: 262500, Avg. loss: 0.379096
Total training time: 0.08 seconds.
-- Epoch 8
Norm: 1.07, NNZs: 15, Bias: -0.820446, T: 300000, Avg. loss: 0.378826
Total training time: 0.09 seconds.
-- Epoch 9
Norm: 1.07, NNZs: 15, Bias: -0.840093, T: 337500, Avg. loss: 0.378604
Total training time: 0.09 seconds.
-- Epoch 10
Norm: 1.08, NNZs: 15, Bias: -0.850329, T: 375000, Avg. loss: 0.378615
Total training time: 0.11 seconds.
Convergence after 10 epochs took 0.11 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=None,
       n_iter=None, 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_
# print(clf.coef_[0])
```

Out[10]:
```
(array([[-0.42328902,  0.18380407, -0.14437354,  0.34064016, -0.21316099,
          0.56702655, -0.44910569, -0.09094413,  0.21219292,  0.17750247,
          0.19931732, -0.00506998, -0.07781235,  0.33343476,  0.0320374 ]]),
 (1, 15),
 array([-0.85032916]))
```

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 [159]:
```
import math
from tqdm import tqdm
```

In [13]:
```
def sigmoid(x,w,b):
    a = np.dot(x,w)+b
    return 1/(1 + math.exp(-a))
```

In [206]:
```
class CustomSGDClassifier:
    def __init__(self,alpha = 0.0001,N = len(X_train),b = 0,w = np.zeros_like(
X_train[0]),eta0 = 0.0001,ephocs = 1):
        self.alpha = alpha
        self.N = N
        self.b = b
        self.w = w
        self.eta0 = eta0
        self.ephocs = ephocs
        self.train_loss = []
        self.test_loss = []

    def calculateLoss(self,w,b,X,Y):
        loss = []
        for i in np.arange(0,len(X)):
            sig = sigmoid(X[i],w,b)
            loss.append(-Y[i]*np.log(sig) - (1-Y[i])*np.log(1-sig) + self.alph
a*np.dot(w,w)/2)
        return np.mean(loss)

    def fit(self,X,Y):
        initial_loss = self.calculateLoss(w,b,X,Y)
#         print('Initial Loss:',initial_loss)
        for ep in range(self.ephocs):
            for i in range(N):
                sig = sigmoid(X[i],self.w,self.b)
                w_new = (1- (self.alpha*self.eta0)/self.N)*self.w + self.eta0*
X[i]*(Y[i]-sig)
                b_new = self.b + self.eta0*(Y[i]-sig)
                self.w = w_new
                self.b = b_new
            next_loss = self.calculateLoss(self.w,self.b,X,Y)
            next_test_loss = self.calculateLoss(self.w,self.b,X_test,y_test)
            self.train_loss.append(next_loss)
            self.test_loss.append(next_test_loss)
            print('-- Epoch: {}, Avg. Train Loss: {}, Avg. Test Loss: {}'.form
at(ep+1,next_loss,next_test_loss))
            if (next_loss < initial_loss) & ((initial_loss-next_loss)<0.0001):
                break
            initial_loss = next_loss
```

In [207]:
```
w = np.zeros_like(X_train[0])
b = 0
eta0  = 0.0001
alpha = 0.0001
N = len(X_train)

model = CustomSGDClassifier(alpha,N,b,w,eta0,ephocs=30)
%time model.fit(X_train,y_train)
```

```
-- Epoch: 1, Avg. Train Loss: 0.40403467542916205, Avg. Test Loss: 0.40517965
84402604
-- Epoch: 2, Avg. Train Loss: 0.3884224663906394, Avg. Test Loss: 0.390098156
77240994
-- Epoch: 3, Avg. Train Loss: 0.38317923880097976, Avg. Test Loss: 0.38505407
73222086
-- Epoch: 4, Avg. Train Loss: 0.38082636890629157, Avg. Test Loss: 0.38278103
942293956
-- Epoch: 5, Avg. Train Loss: 0.37965148006379457, Avg. Test Loss: 0.38163750
58602411
-- Epoch: 6, Avg. Train Loss: 0.3790337899254833, Avg. Test Loss: 0.381030985
0590714
-- Epoch: 7, Avg. Train Loss: 0.37869975088299546, Avg. Test Loss: 0.38069986
14313595
-- Epoch: 8, Avg. Train Loss: 0.37851603647385695, Avg. Test Loss: 0.38051590
616990844
-- Epoch: 9, Avg. Train Loss: 0.37841389926707225, Avg. Test Loss: 0.38041251
899165923
-- Epoch: 10, Avg. Train Loss: 0.3783566969355589, Avg. Test Loss: 0.38035391
95881942
Wall time: 17.4 s
```

In [208]:
```
model.w, model.b
# model.train_loss, model.test_loss
```

Out[208]:
```
(array([-0.42320236,  0.19097504, -0.14588903,  0.33813461, -0.21204107,
         0.56528021, -0.44537758, -0.09169276,  0.21798654,  0.16980147,
         0.19524869,  0.00226123, -0.0778474 ,  0.33881857,  0.02215503]),
 -0.8505912797715787)
```

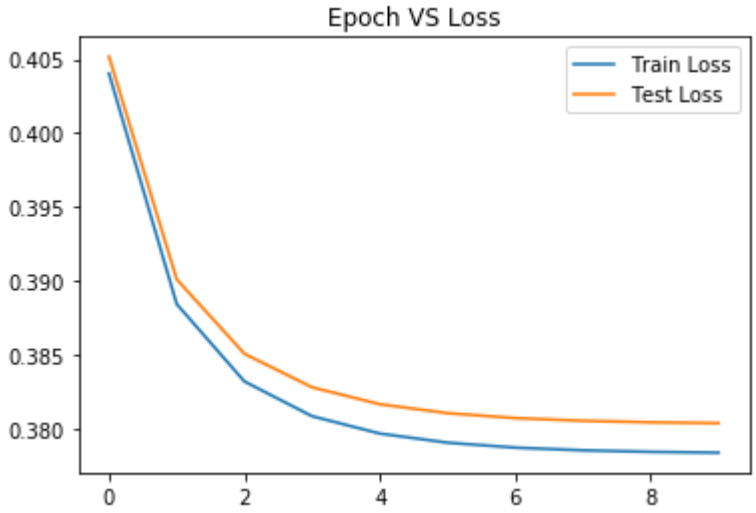
In [209]:
```
# these are the results we got after we implemented sgd and found the optimal
 weights and intercept
model.w-clf.coef_, model.b-clf.intercept_
```

Out[209]:
```
(array([[ 8.66526892e-05,  7.17096975e-03, -1.51548550e-03,
         -2.50554953e-03,  1.11991916e-03, -1.74634334e-03,
          3.72810459e-03, -7.48633412e-04,  5.79362170e-03,
         -7.70099691e-03, -4.06863374e-03,  7.33121135e-03,
         -3.50496760e-05,  5.38380705e-03, -9.88236480e-03]]),
 array([-0.00026212]))
```

In [212]:
```
import matplotlib.pyplot as plt

# print(model.train_loss)
# print(model.test_loss)

epoch = len(model.train_loss)
# print(epoch)
plt.plot(range(epoch),model.train_loss, label='Train Loss')
plt.plot(range(epoch),model.test_loss, label='Test Loss')
plt.title('Epoch VS Loss')
plt.legend()
plt.show()
```



In [214]:
```
def pred(w,b, X):
    N = len(X)
    predict = []
    for i in range(N):
        if sigmoid(X[i],w, b) >= 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)
print(1-np.sum(y_train - pred(model.w,model.b,X_train))/len(X_train))
print(1-np.sum(y_test  - pred(model.w,model.b,X_test))/len(X_test))
```

```
0.9553333333333334
0.95288
```