In [42]:
```
import numpy as np
import pandas as pd
from sklearn.datasets import make_classification
import matplotlib.pyplot as plt
```

In [43]:
```
X, y = make_classification(n_samples=50000, n_features=15, n_informative=10,
                           n_classes=2, weights=[0.7], class_sep=0.7, random
```

In [44]:
```
X.shape, y.shape
```

Out[44]: ((50000, 15), (50000,))

In [45]:
```
from sklearn.model_selection import train_test_split
```

In [46]:
```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.25, ra
```

In [47]:
```
x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

Out[47]: ((37500, 15), (37500,), (12500, 15), (12500,))

In [48]:
```
from sklearn import linear_model
```

In [49]:
```
# alpha : float
# Constant that multiplies the regularization term.

# eta0 : double
# The initial learning rate for the 'constant', 'invscaling' or 'adaptive' s

clf = linear_model.SGDClassifier(eta0=0.0001, alpha=0.0001, loss='log', rand
clf
```

Out[49]:
```
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 [50]:
```
1 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: 0.07 seconds.
-- Epoch 5
Norm: 1.04, NNZs: 15, Bias: -0.719528, T: 187500, Avg. loss: 0.380486
Total training time: 0.08 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: 0.14 seconds.
-- Epoch 9
Norm: 1.07, NNZs: 15, Bias: -0.837805, T: 337500, Avg. loss: 0.378585
Total training time: 0.17 seconds.
-- Epoch 10
Norm: 1.08, NNZs: 15, Bias: -0.853138, T: 375000, Avg. loss: 0.378630
Total training time: 0.19 seconds.
Convergence after 10 epochs took 0.20 seconds
```

Out[50]:
```
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 [51]:
```
1 clf.coef_, clf.coef_.shape, clf.intercept_
```

Out[51]:
```
(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 (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 [52]:
```
w = np.zeros_like(x_train[0])
b = 0
eta0  = 0.0001
alpha = 0.0001
N = len(x_train)
```

In [53]:
```
# write your code to implement SGD as per the above instructions
# please choose the number of iternations on your own
def sigmoid(w,x,b):
    return 1/(1+np.exp(-(np.dot(x,w.T)+b)))
```

In [54]:
```
# log loss function Formula -(ylog10(y_hat) - (1-y)log10(1-y_hat))
def log_function(w,x,y,b):
    p = sigmoid(w, x, b)
    loss =(-(y * np.log10(p)))-((1 - y) * np.log10(1 - p))
    return np.mean(loss)
```

In [ ]:
```

```

In [55]:
```
from tqdm import tqdm
def trainSGD(x_train, y_train, x_test, y_test, eta0, alpha, epochs, w, b):
    train_loss=[]
    test_loss=[]
    for k in tqdm(range(0, epochs)):
        for i in range(0,len(x_train)):
            y = y_train[i]
            x = x_train[i]
            w = ((1-eta0*(alpha/N))*w)+((eta0*x)*(y-sigmoid(w,x,b)))
            b = b+(eta0*(y-sigmoid(w,x,b)))

      # calculating the train loss
#         logloss_train=0
#         for x in x_train:
        logloss_train= log_function(w,x_train,y_train,b)
        train_loss.append(logloss_train)

      #calcilating the test loss
#         logloss_test=0
#         for x in x_train:
        logloss_test= log_function(w,x_test,y_test,b)
        test_loss.append(logloss_test)

    return w,b,train_loss, test_loss
```

In [56]:
```
epochs=10
w1,b1,trainloss,testloss =trainSGD(x_train, y_train, x_test, y_test,eta0, al
```

```
100%|██████████| 10/10 [00:07<00:00,  1.30it/s]
```

In [57]:
```
print(w1)
print(b1)
print(trainloss)
```

```
[-0.42315311  0.19095979 -0.14588118  0.33814991 -0.21196623  0.56525978
 -0.44538357 -0.09171679  0.21795314  0.16977398  0.19522044  0.00229554
 -0.07781461  0.33882618  0.02214234]
-0.8500967712837224
[0.17546926223702466, 0.16868174436540248, 0.16639953379688374, 0.1653740490192
8135, 0.1648612200408247, 0.16459114506307726, 0.16444479874475637, 0.164364115
2252568, 0.16431912310828212, 0.16429382915597823]
```

In [58]:
```
# these are the results we got after we implemented sgd and found the optima
w-clf.coef_, b-clf.intercept_
```

Out[58]:
```
(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]]),
 array([0.8531383]))
```

```
Difference between sklearn clf coefficient and w1 that is updated by the
code.
```

In [60]:
```
x=w1-clf.coef_
y=b1-clf.intercept_

print(x)
print(y)
```

```
[[ 0.0002138   0.00548413  0.00270918 -0.00329416 -0.00377953  0.00509399
   0.00704126  0.00237134  0.00867994 -0.01106728 -0.00183147 -0.00192361
   0.00178909  0.00029817 -0.00052487]]
[0.00304153]
```

In [61]:
```
from sklearn.metrics import accuracy_score
def 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(-(d
            predict.append(1)
        else:
            predict.append(0)
    return np.array(predict)
print(1-np.sum(y_train - pred(w1,b1,x_train))/len(x_train))
print(1-np.sum(y_test  - pred(w1,b1,x_test))/len(x_test))
```

```
0.95536
0.95296
```

In [62]:
```
plt.plot(np.array(trainloss), color='blue', label='Train loss')
plt.plot(np.array(testloss), color='red', label='Test loss')
plt.ylabel('LogLoss')
plt.xlabel('epochs')
plt.title('Log loss curve')
plt.legend()
plt.show()
```