In [2]:
```
import numpy as np
import pandas as pd
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import linear_model
```

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]:
```
#splitting datset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=15)
```

In [8]:
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

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 [79]:
```
clf = linear_model.SGDClassifier(eta0=0.0001, alpha=0.0001, loss='log', random_state=15, penalty='l2', tol=1e-3, verbo
se=2, learning_rate='constant')
clf
```

Out[79]:
```
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 [80]:
```
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: 0.03 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: 0.06 seconds.
-- Epoch 5
Norm: 1.04, NNZs: 15, Bias: -0.719528, T: 187500, Avg. loss: 0.380486
Total training time: 0.07 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[80]:
```
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 [81]:
```
clf.coef_, clf.coef_.shape, clf.intercept_
```

Out[81]:
```
(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

In [3]:
```
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 [4]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=15)
```

Initialize weights

In [5]:
```
def initialize_weights(dim):

    w=np.zeros_like(dim)
    b=0
    return w,b
```

In [6]:
```
dim=X_train[0]
w,b = initialize_weights(dim)
print('w =',(w))
print('b =',str(b))
```

```
w = [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
b = 0
```

Grader function-1

In [7]:
```
dim=X_train[0]
w,b = initialize_weights(dim)
def grader_weights(w,b):
  assert((len(w)==len(dim)) and b==0 and np.sum(w)==0.0)
  return True
grader_weights(w,b)
```

Out[7]: True

Compute sigmoid

In [8]:
```
def sigmoid(z):
    sig=1/(1+np.exp(-z))
    return sig
```

Grader function - 2

In [9]:
```
def grader_sigmoid(z):
  val=sigmoid(z)
  assert(val==0.8807970779778823)
  return True
grader_sigmoid(2)
```

Out[9]: True

Compute loss

In [10]:
```
def logloss(y_true,y_pred):
    n=len(y_true)
    sum=0
    for i in range(n):
        sum +=(y_true[i] * np.log10(y_pred[i])) + ((1 - y_true[i]) * np.log10(1 - y_pred[i]))
    loss= -1*(1/n)*sum
    return loss
```

Grader function - 3

In [11]:
```
def grader_logloss(true,pred):
  loss=logloss(true,pred)
  assert(loss==0.07644900402910389)
  return True
true=[1,1,0,1,0]
pred=[0.9,0.8,0.1,0.8,0.2]
grader_logloss(true,pred)
```

Out[11]: True

Compute gradient w.r.to 'w'

In [12]:
```
def gradient_dw(x,y,w,b,alpha,N):
    z=np.dot(w,x)+b
    dw=x*(y-sigmoid(z)-(alpha/N)*w)
    return dw
```

Grader function - 4

In [13]:
```
def grader_dw(x,y,w,b,alpha,N):
  grad_dw=gradient_dw(x,y,w,b,alpha,N)
  assert(np.sum(grad_dw)==2.613689585)
  return True
grad_x=np.array([-2.07864835,  3.31604252, -0.79104357, -3.87045546, -1.14783286,
       -2.81434437, -0.86771071, -0.04073287,  0.84827878,  1.99451725,
        3.67152472,  0.01451875,  2.01062888,  0.07373904, -5.54586092])
grad_y=0
grad_w,grad_b=initialize_weights(grad_x)
alpha=0.0001
N=len(X_train)
grader_dw(grad_x,grad_y,grad_w,grad_b,alpha,N)
```

Out[13]: True

Compute gradient w.r.to 'b'

In [14]:
```
def gradient_db(x,y,w,b):
    z=np.dot(w,x)+b
    db = y - sigmoid(z)
    return db
```

Grader function - 5

In [15]:
```
def grader_db(x,y,w,b):
  grad_db=gradient_db(x,y,w,b)
  assert(grad_db==-0.5)
  return True
grad_x=np.array([-2.07864835,  3.31604252, -0.79104357, -3.87045546, -1.14783286,
       -2.81434437, -0.86771071, -0.04073287,  0.84827878,  1.99451725,
        3.67152472,  0.01451875,  2.01062888,  0.07373904, -5.54586092])
grad_y=0
grad_w,grad_b=initialize_weights(grad_x)
alpha=0.0001
N=len(X_train)
grader_db(grad_x,grad_y,grad_w,grad_b)
```

Out[15]: True

Implementing logistic regression

In [127]:
```
def train(X_train,y_train,X_test,y_test,epochs,alpha,eta0):
    train_loss=[]
    test_loss=[]
    dim=X_train[0]

    w,b = initialize_weights(dim)

    for i in range(epochs):


        for j in range(N):

            dw=gradient_dw(X_train[j],y_train[j],w,b,alpha,N)
            db=gradient_db(X_train[j],y_train[j],w,b)
            w = w + eta0 * dw
            b = b + eta0 * db


        train_pred=[sigmoid(np.dot(w, X_train[p]) + b) for p in range(len(X_train))]
        test_pred=[sigmoid(np.dot(w, X_test[q]) + b) for q in range(len(X_test))]



        loss1 = logloss(y_train, train_pred)
        train_loss.append(loss1)

        loss2 = logloss(y_test, test_pred)
        test_loss.append(loss2)



    return w,b,train_loss,test_loss
```

In [128]:
```
alpha=0.0001
eta0=0.0001
N=len(X_train)
epochs=10
w,b,train_loss,test_loss=train(X_train,y_train,X_test,y_test,epochs,alpha,eta0)
```

In [130]:
```
print("weights of custom implementation")
w
```

```
weights of custom implementation
```

Out[130]:
```
array([-0.42320237,  0.19097504, -0.14588903,  0.33813462, -0.21204108,
        0.56528022, -0.44537758, -0.09169275,  0.21798654,  0.16980148,
        0.19524869,  0.00226124, -0.0778474 ,  0.33881858,  0.02215503])
```

In [131]:
```
print("intercept of custom implementation")
b
```

```
intercept of custom implementation
```

Out[131]:
```
-0.8505912823639118
```

Goal of assignment

In [132]:
```
diff=np.subtract(w,clf.coef_)
diff
```

Out[132]:
```
array([[ 0.00016455,  0.00549939,  0.00270133, -0.00330945, -0.00385437,
         0.00511443,  0.00704724,  0.00239538,  0.00871335, -0.01103978,
        -0.00180321, -0.00195792,  0.0017563 ,  0.00029056, -0.00051218]])
```

In [133]:
```
k=[]
for list in diff:
    for i in list:
        n = "{:e}".format(i)
        k.append(n)
```

In [134]:
```
print("Difference of weights:")
print(np.array(k))
```

```
Difference of weights:
['1.645475e-04' '5.499392e-03' '2.701329e-03' '-3.309448e-03'
 '-3.854374e-03' '5.114433e-03' '7.047242e-03' '2.395376e-03'
 '8.713346e-03' '-1.103978e-02' '-1.803215e-03' '-1.957920e-03'
 '1.756299e-03' '2.905625e-04' '-5.121807e-04']
```

In [135]:
```
print("Difference of intercepts:")
np.subtract(b,clf.intercept_)
```

```
Difference of intercepts:
```

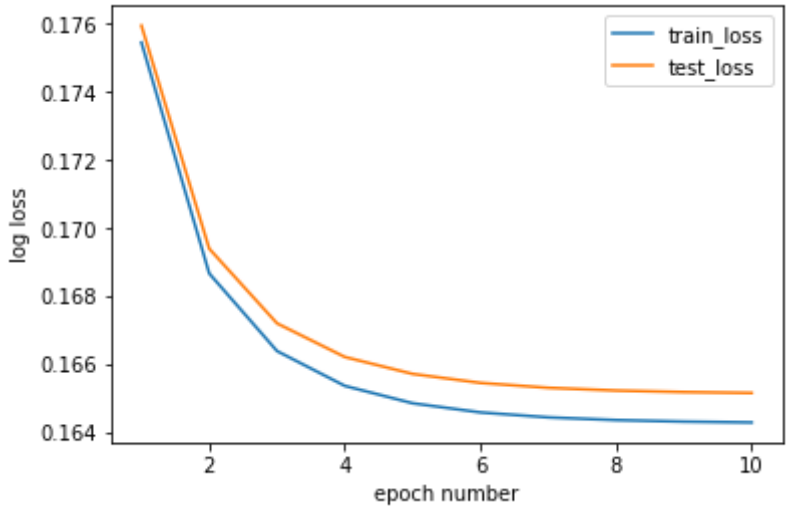
Out[135]:
```
array([0.00254701])
```

Plot epoch number vs train , test loss

In [136]:
```
from matplotlib import pyplot as plt
epoch = [i for i in range(1,11,1)]

plt.plot(epoch,train_loss , label='train_loss')
plt.plot(epoch,test_loss, label='test_loss')
plt.xlabel("epoch number")
plt.ylabel("log loss")
plt.legend()
plt.show
```

Out[136]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



In [137]:
```
def pred(w,b, X):
    N = len(X)
    predict = []
    for i in range(N):
        z=np.dot(w,X[i])+b
        if sigmoid(z) >= 0.5:
            predict.append(1)
        else:
            predict.append(0)
    return np.array(predict)
print(1-np.sum(y_train - pred(w,b,X_train))/len(X_train))
print(1-np.sum(y_test  - pred(w,b,X_test))/len(X_test))
```

```
0.9553333333333334
0.95288
```

In [ ]:
```
!jupyter nbconvert --to html LRUsingSGD_solve.ipynb
```