```
import numpy as np
import pandas as pd
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
import math as m
from sklearn import linear_model
import warnings
warnings.filterwarnings("ignore")
```

### Splitting data into train and test

```
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)

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=15)

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

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

### Intializing initial parameters

```
w = np.zeros_like(X_train[0])# initial weight vector
b = 0      # initial intercept value
eta0  = 0.0001  # learning rate
alpha = 0.0001 # lambda value
N = len(X_train)

def sigmoid(w,x,b):
    return 1/(1+np.exp(-(np.dot(x,w.T)+b))) #return 1/1+e(-x)

def logloss(w,x,y,b,reg=0):
    val=sigmoid(w,x,b)
    return -np.mean(y*np.log10(val)+(1-y)*np.log10(1-val))+reg  # cost
function of logistic regression

print("INITIAL LOG LOSS:")
logloss(w,X_train,y_train,b)
```

```
INITIAL LOG LOSS:

0.3010299956639812
```

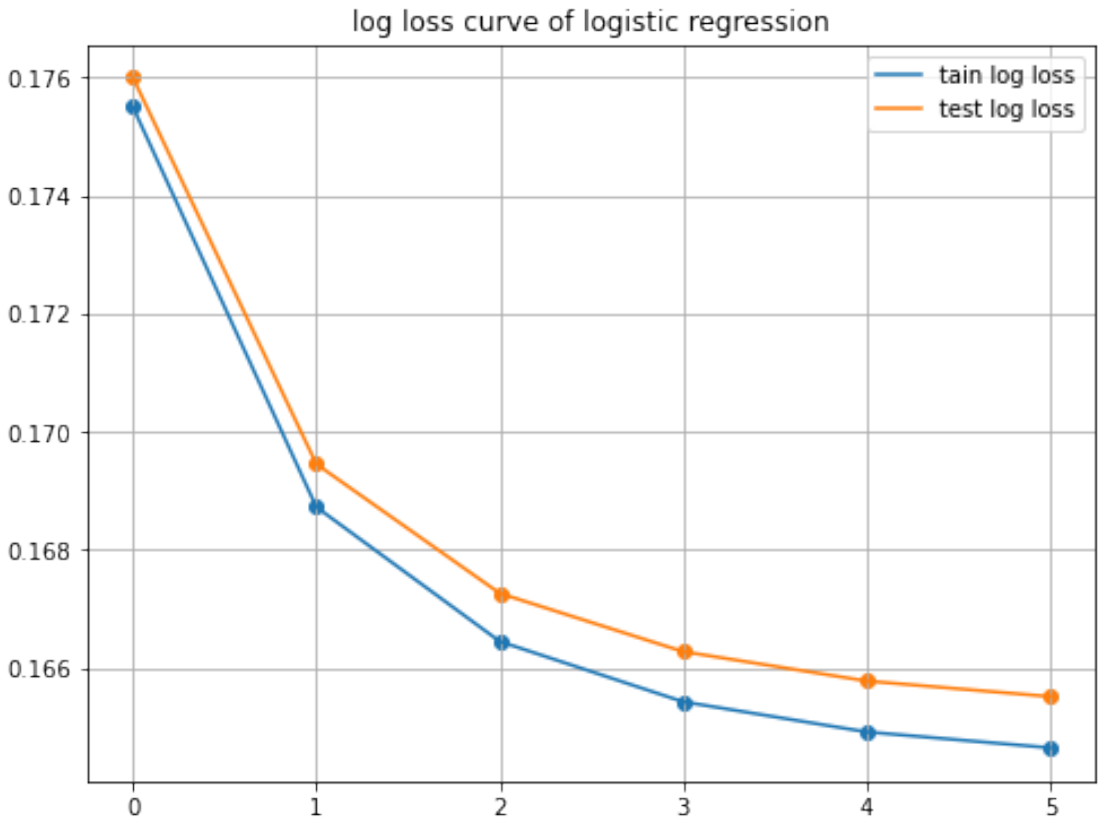
### SGD Alorithm

```
def sgd_algo(x_train,y_train,x_test,y_test,eta0,alpha,w,b,epoch):
    train_loss=[]
    test_loss=[]
    epoc=[]
    for i in  range(0,epoch):
        epoc.append(i)
        for j in range(0,N):
```

```
            reg=alpha/2*np.dot(w.T,w) #regulrization term
            w = ((1-eta0*(alpha/N))*w)+((eta0*x_train[j])*(y_train[j]-
sigmoid(w,x_train[j],b))) # updating weight vector
            b = b+(eta0*(y_train[j]-sigmoid(w,x_train[j],b)))       #
updatind intercept
        train=logloss(w,x_train,y_train,b,reg)
        train_loss.append(train)    # calculating train and test loss
for updated w,b on each epoch
        test=logloss(w,x_test,y_test,b,reg)
        test_loss.append(test)
        """
       if i==0 :
            continue                                                #
block to check covergence
        else:                                                       #
but checking not getting optimium value as sklearn implementation
            if abs(train_loss[i]-train_loss[i-1])>.001:
                continue
            else:
                break"""
    return w,b ,train_loss,test_loss,epoc

epoch=6
w,b,tr,te,epoc=sgd_algo(X_train,y_train,X_test,y_test,eta0,
alpha,w,b,epoch)

print("optimal weight vector:")
print("\n")
print(w)

optimal weight vector:


[-0.40819512  0.18608803 -0.13873193  0.33720386 -0.19058573
0.55321631
 -0.44625706 -0.09550674  0.20929646  0.16027648  0.18715532
0.00822064
 -0.0701101   0.33795986  0.0206259 ]

print("optimal intercept value:")
print("\n")
print(b)

optimal intercept value:


-0.7613483856830058

%matplotlib inline
import matplotlib.pyplot as plt
plt.figure(figsize=(8,6))
```

```
plt.grid()
plt.plot(epoc,tr, label='tain log loss')
plt.plot(epoc,te, label='test log loss')
plt.scatter(epoc,tr)
plt.scatter(epoc,te)
plt.title('log loss curve of logistic regression')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x21a588551c0>
```




### sklearn implementation

```
clf = linear_model.SGDClassifier(eta0=0.0001, alpha=0.0001,
loss='log', random_state=15, penalty='l2', tol=.001, verbose=2,
learning_rate='constant')
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: 0.05 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: 0.08 seconds.
-- Epoch 7
Norm: 1.06, NNZs: 15, Bias: -0.795106, T: 262500, Avg. loss: 0.379150
Total training time: 0.10 seconds.
-- Epoch 8
Norm: 1.06, NNZs: 15, Bias: -0.819925, T: 300000, Avg. loss: 0.378856
Total training time: 0.11 seconds.
-- Epoch 9
Norm: 1.07, NNZs: 15, Bias: -0.837805, T: 337500, Avg. loss: 0.378585
Total training time: 0.13 seconds.
-- Epoch 10
Norm: 1.08, NNZs: 15, Bias: -0.853138, T: 375000, Avg. loss: 0.378630
Total training time: 0.14 seconds.
Convergence after 10 epochs took 0.14 seconds

SGDClassifier(eta0=0.0001, learning_rate='constant', loss='log',
              random_state=15, verbose=2)

clf.coef_

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]])

clf.coef_-w

array([[-1.51717989e-02, -6.12381954e-04, -9.85842395e-03,
         4.24020697e-03, -1.76009715e-02,  6.94947845e-03,
        -6.16776313e-03,  1.41860939e-03, -2.32639087e-05,
         2.05647827e-02,  9.89658543e-03, -4.00147868e-03,
        -9.49360054e-03,  5.68155244e-04,  2.04130644e-03]])
```

differnce between sklearn and custom implementation is almost in terms of 10^-3

```
def pred(w,b, X):
    N = len(X)
    predict = []
    for i in range(N):
        if sigmoid(w, X[i], b) >= 0.5:
            predict.append(1)
        else:
```

```
            predict.append(0)
    return np.array(predict)
print("train accuracy :")
print(1-np.sum(y_train - pred(w,b,X_train))/len(X_train))
print("test accuracy :")
print(1-np.sum(y_test  - pred(w,b,X_test))/len(X_test))
```

```
train accuracy :
0.9617866666666667
test accuracy :
0.95952
```