Implementation of Stochastic Gradient Descent Classifier with Logloss function and L2 regularization (a.k.a. Logistic Regression with SGD) without using scikit-learn

In [1]:
```python
# Importing Libraries
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
```

In [2]:
```python
# Creating custom dataset
X, y = make_classification(n_samples=50000, n_features=15, n_informative=10, n_redundant=5,
                           n_classes=2,)
```

In [3]:
```python
X.shape, y.shape
```

Out[3]:
```
((50000, 15), (50000,))
```

In [4]:
```python
# Splitting data into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
```

In [5]:
```python
# Standardizing the data.
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [6]:
```python
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

Out[6]:
```
((37500, 15), (37500,), (12500, 15), (12500,))
```

Initializing the weights

In [7]:
```python
# Initializing Weights
def initialize_weights(dim):
    # In this function, we are initializing our weights and bias.
    # Initializing weights to zero to keep it simple.
    w=np.zeros_like(dim)
    # Typically bias are initialized as zero.
    b=0
    return w,b
```

Computing sigmoid function

In [8]:
```python
# This function calculates sigmoid value : sigmoid = 1/(1+exp(-z))
def sigmoid(z):

    sig=1/(1+np.exp(-z))

    return sig
```

Computing logloss function: $logloss = -1 * \frac{1}{n}\Sigma_{foreachYt,Y_{pred}}(Ytlog10(Y_{pred}) + (1 - Yt)log10(1 - Y_{pred}))$

In [9]:
```python
## Computing loss function
def logloss(y_true,y_pred):
    loss=0
    for i in range(len(y_true)):
        yt=y_true[i]
        yp=y_pred[i]
        func=(yt*np.log10(yp)) + ((1-yt)*(np.log10(1-yp)))
        loss=loss+func
    #print(loss)
    loss=-loss/len(y_true)
    #print(loss)
    return loss
```

Computing gradient of w : $dw^{(t)} = x_n(y_n - \sigma((w^{(t)})^T x_n + b^t)) - \frac{\lambda}{N}w^{(t)}$

In [10]:
```python
## Gradient of w calculation function
def gradient_dw(x,y,w,b,alpha,N):

    s=sigmoid(np.dot(w.T,x)+b)
    #print(s)
    dw=x*(y-s)-(alpha*w/N)
    #print(dw)
    return dw
```

Computing gradient of b : $db^{(t)} = y_n - \sigma((w^{(t)})^T x_n + b^t)$

In [11]:
```python
## Gradient of b calculation function
def gradient_db(x,y,w,b):
    s=sigmoid(np.dot(w.T,x)+b)
    #print(s)
    db=y-s
    return db
```

Implementing logistic regression using stochastic gradient descent.

In [12]:
```python
# SGD Logistic Regression function
def train(X_train,y_train,X_test,y_test,epochs,alpha,eta0):

    # eta0 is learning rate
    # Initalize the weights (call the initialize_weights(X_train[0]) function)

    w,b=initialize_weights(X_train[0])

    train_losses=[]
    test_losses=[]

    for epoch in range(epochs): # Iterating thorugh each epoch

        for i in range(len(X_train)): # Iterating thorugh each data point

            x=X_train[i]
            y=y_train[i]
            dw=gradient_dw(x,y,w,b,alpha,N) # Computing gradient of w

            db=gradient_db(x,y,w,b) # Computing gradient of b

            # Updating w and b
            w=w+(eta0*dw)
            b=b+(eta0*db)

        # Predict the output of X_train using w,b
        y_pred=[]
        for j in X_train:
            z=np.dot(w.T,j)+b
            y_p=sigmoid(z)
            y_pred.append(y_p)

        # Computing the loss between predicted and actual values.
        loss=logloss(y_train,y_pred)

        # Store all the train loss values in a list
        train_losses.append(loss)

        # predict the output of X_test using w,b
        y_pred=[]
        for k in X_test:
            z=np.dot(w.T,k)+b
            y_p=sigmoid(z)
            y_pred.append(y_p)

        # Computing the loss between predicted and actual values
        loss=logloss(y_test,y_pred)

        # Store all the test loss values in a list
        test_losses.append(loss)

    return w,b,train_losses,test_losses
```

In [13]:
```python
alpha=0.0001
eta0=0.0001
N=len(X_train)
epochs=14
w,b,train_loss,test_loss=train(X_train,y_train,X_test,y_test,epochs,alpha,eta0)
```

In [14]:
```python
train_loss
```

Out[14]:
```
[0.21604927494749188,
 0.2018519246086247,
 0.19663983235634006,
 0.19419077111910116,
 0.19289381643119002,
 0.19215701956085446,
 0.19171900104747927,
 0.19145039298539557,
 0.19128200776134538,
 0.1911747460381558,
 0.19110560330149445,
 0.19106063233815765,
 0.1910311835150574,
 0.19101179903992474]
```

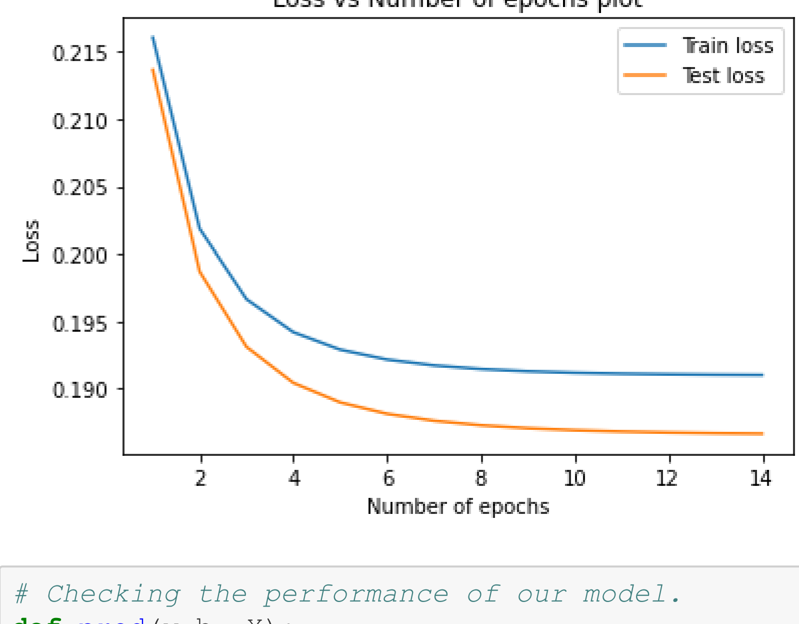
In [15]:
```python
test_loss
```

Out[15]:
```
[0.21364411866247926,
 0.19868874443095638,
 0.193101488792856 7,
 0.19042519389184873,
 0.1889746946137371,
 0.18812712841871612,
 0.18760587534998843,
 0.18727306048156522,
 0.18705427449773884,
 0.18690697324674027,
 0.18680575330829913,
 0.1867349281365113,
 0.18668454846907312,
 0.18664816332654732]
```

Ploting epoch number vs train, test loss

In [16]:
```python
import matplotlib.pyplot as plt
```

In [17]:
```python
plt.plot(range(1,epochs+1),train_loss,label='Train loss')
plt.plot(range(1,epochs+1),test_loss,label='Test loss')
plt.title('Loss vs Number of epochs plot')
plt.xlabel('Number of epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



In [18]:
```python
# Checking the performance of our model.
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))
```

```
0.998
0.99064
```