Implement SGD Classifier with Logloss and L2 regularization Using SGD without using sklearn

There will be some functions that start with the word "grader" ex: grader_weights(), grader_sigmoid(), grader_logloss() etc, you should not change those function definition.

Every Grader function has to return True.

Importing packages

In [3]:
```
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
```

Creating custom dataset

In [4]:
```
# please don't change random_state
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)
# make_classification is used to create custom dataset
# Please check this link (https://scikit-learn.org/stable/modules/generated/sk
learn.datasets.make_classification.html) for more details
```

In [5]:
```
X.shape, y.shape
```

Out[5]:
```
((50000, 15), (50000,))
```

Splitting data into train and test

In [6]:
```
#please don't change random state
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, rand
om_state=15)
```

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

In [9]:
```
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

Out[9]:
```
((37500, 15), (37500,), (12500, 15), (12500,))
```

SGD classifier

In [10]:
```
# 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
# Please check this documentation (https://scikit-learn.org/stable/modules/gen
erated/sklearn.linear_model.SGDClassifier.html)
```

Out[10]:
```
SGDClassifier(eta0=0.0001, learning_rate='constant', loss='log',
              random_state=15, verbose=2)
```

In [11]:
```
clf.fit(X=X_train, y=y_train) # fitting our model
```

```
-- Epoch 1
Norm: 0.70, NNZs: 15, Bias: -0.501317, T: 37500, Avg. loss: 0.552526
Total training time: 0.02 seconds.
-- Epoch 2
Norm: 1.04, NNZs: 15, Bias: -0.752393, T: 75000, Avg. loss: 0.448021
Total training time: 0.03 seconds.
-- Epoch 3
Norm: 1.26, NNZs: 15, Bias: -0.902742, T: 112500, Avg. loss: 0.415724
Total training time: 0.05 seconds.
-- Epoch 4
Norm: 1.43, NNZs: 15, Bias: -1.003816, T: 150000, Avg. loss: 0.400895
Total training time: 0.06 seconds.
-- Epoch 5
Norm: 1.55, NNZs: 15, Bias: -1.076296, T: 187500, Avg. loss: 0.392879
Total training time: 0.07 seconds.
-- Epoch 6
Norm: 1.65, NNZs: 15, Bias: -1.131077, T: 225000, Avg. loss: 0.388094
Total training time: 0.08 seconds.
-- Epoch 7
Norm: 1.73, NNZs: 15, Bias: -1.171791, T: 262500, Avg. loss: 0.385077
Total training time: 0.10 seconds.
-- Epoch 8
Norm: 1.80, NNZs: 15, Bias: -1.203840, T: 300000, Avg. loss: 0.383074
Total training time: 0.10 seconds.
-- Epoch 9
Norm: 1.86, NNZs: 15, Bias: -1.229563, T: 337500, Avg. loss: 0.381703
Total training time: 0.12 seconds.
-- Epoch 10
Norm: 1.90, NNZs: 15, Bias: -1.251245, T: 375000, Avg. loss: 0.380763
Total training time: 0.14 seconds.
-- Epoch 11
Norm: 1.94, NNZs: 15, Bias: -1.269044, T: 412500, Avg. loss: 0.380084
Total training time: 0.14 seconds.
-- Epoch 12
Norm: 1.98, NNZs: 15, Bias: -1.282485, T: 450000, Avg. loss: 0.379607
Total training time: 0.16 seconds.
-- Epoch 13
Norm: 2.01, NNZs: 15, Bias: -1.294386, T: 487500, Avg. loss: 0.379251
Total training time: 0.17 seconds.
-- Epoch 14
Norm: 2.03, NNZs: 15, Bias: -1.305805, T: 525000, Avg. loss: 0.378992
Total training time: 0.18 seconds.
Convergence after 14 epochs took 0.18 seconds
```

Out[11]:
```
SGDClassifier(eta0=0.0001, learning_rate='constant', loss='log',
              random_state=15, verbose=2)
```

In [12]:
```
clf.coef_, clf.coef_.shape, clf.intercept_
#clf.coef_ will return the weights
#clf.coef_.shape will return the shape of weights
#clf.intercept_ will return the intercept term
```

Out[12]:
```
(array([[-0.89007184,  0.63162363, -0.07594145,  0.63107107, -0.38434375,
          0.93235243, -0.89573521, -0.07340522,  0.40591417,  0.4199991 ,
          0.24722143,  0.05046199, -0.08877987,  0.54081652,  0.06643888]]),
 (1, 15),
 array([-1.30580538]))
```

```
# This is formatted as code
```

Implement Logistic Regression with L2 regularization Using SGD: without using sklearn

1. We will be giving you some functions, please write code in that functions only.
2. After every function, we will be giving you expected output, please make sure that you get that output.

- Initialize the weight_vector and intercept term to zeros (Write your code in def initialize_weights())
- Create a loss function (Write your code in def logloss())

  $logloss = -1 * \frac{1}{n}\Sigma_{foreachYt,Y_{pred}}(Ytlog10(Y_{pred}) + (1-Yt)log10(1-Y_{pred}))$
- 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 (write your code in def gradient_dw())

      $dw^{(t)} = x_n(y_n - \sigma((w^{(t)})^T x_n + b^t)) - \frac{\lambda}{N}w^{(t)})$
    - Calculate the gradient of the intercept (write your code in def gradient_db()) check this (https://drive.google.com/file/d/1nQ08-XY4zvOLzRX-lGf8EYB5arb7-m1H/view?usp=sharing)

      $db^{(t)} = y_n - \sigma((w^{(t)})^T x_n + b^t))$
    - 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)):
      $w^{(t+1)} \leftarrow w^{(t)} + \alpha(dw^{(t)})$

      $b^{(t+1)} \leftarrow b^{(t)} + \alpha(db^{(t)})$
  - 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)

Initialize weights

In [17]:
```
def initialize_weights(dim):
    ''' In this function, we will initialize our weights and bias'''
    #initialize the weights to zeros array of (1,dim) dimensions
    #you use zeros_like function to initialize zero, check this link https://d
ocs.scipy.org/doc/numpy/reference/generated/numpy.zeros_like.html
    #initialize bias to zero
    w = np.zeros_like(X_train[0])
    b = 0

    return w,b
```

In [18]:
```
X_train
```

Out[18]:
```
array([[-0.39348337, -0.19771903, -0.15037836, ..., -0.6624427 ,
        -0.68888516,  0.56015427],
       [ 0.73641397, -0.33354604,  0.13809494, ...,  0.07936129,
         0.39617118, -0.68713223],
       [ 0.85834602,  0.24584928, -0.38694362, ...,  0.04608795,
         0.92475102, -1.05700576],
       ...,
       [-1.10477655,  0.69177632, -0.69482772, ...,  0.0470451 ,
         0.25115792, -0.53547666],
       [ 0.83162118, -0.33112526, -0.5686103 , ..., -0.348647  ,
        -2.30536841,  0.63317857],
       [-1.57784861,  1.69046127,  0.57997222, ...,  0.937343  ,
        -0.65399896, -0.14789883]])
```

In [19]:
```
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 [20]:
```
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[20]:
```
True
```

Compute sigmoid

$$sigmoid(z) = 1/(1 + exp(-z))$$

In [21]:
```
import math
def sigmoid(z):
    ''' In this function, we will return sigmoid of z'''
    # compute sigmoid(z) and return

    return (1/(1+math.exp(-z)))
```

Grader function - 2

In [22]:
```
def grader_sigmoid(z):
  val=sigmoid(z)
  assert(val==0.8807970779778823)
  return True
grader_sigmoid(2)
```

Out[22]: True

Compute loss

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

In [44]:
```
def logloss(y_true,y_pred):
    '''In this function, we will compute log loss '''
    loss = 0
    A = list(zip(y_true, y_pred))
    for y, y_score in A:
        loss += (-1/len(A))*(y*np.log10(y_score) + (1-y) * np.log10(1-y_score
))
    return loss
```

Grader function - 3

In [45]:
```
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[45]: True

Compute gradient w.r.to 'w'

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

In [32]:
```
def gradient_dw(x,y,w,b,alpha,N):
    '''In this function, we will compute the gardient w.r.to w '''
    z = np.dot(w, x) + b
    dw = x*(y - sigmoid(z)) - ((1/alpha)*(1/N) * w)

    return dw
```

Grader function - 4

In [33]:
```
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, -1.147832
86,
       -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[33]: True

Compute gradient w.r.to 'b'

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

In [37]:
```
 def gradient_db(x,y,w,b):
     #'''In this function, we will compute gradient w.r.to b '''
        z = np.dot(w, x) + b
        db = y - sigmoid(z)
        return db
```

Grader function - 5

In [38]:
```
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, -1.147832
86,
       -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[38]: True

Implementing logistic regression

In [50]:
```
def train(X_train,y_train,X_test,y_test,epochs,alpha,eta0):
    ''' In this function, we will implement logistic regression'''
    #Here eta0 is learning rate
    #implement the code as follows
    # initalize the weights (call the initialize_weights(X_train[0]) function)
    w, b = initialize_weights(X_train[0])
    # for every epoch
    train_loss = []
    test_loss = []
    for epoch in range(epochs):
        # for every data point(X_train,y_train)
        for x, y in zip(X_train, y_train):
           #compute gradient w.r.to w (call the gradient_dw() function)
            dw = gradient_dw(x, y, w, b, alpha, len(X_train))
           #compute gradient w.r.to b (call the gradient_db() function)
            db = gradient_db(x, y, w, b)
           #update w, b
            w = w + eta0 * dw
            b = b + eta0 * db
        # predict the output of x_train[for all data points in X_train] using
 w,b
        y_pred = [sigmoid(np.dot(w, x)) for x in X_train]
        #compute the loss between predicted and actual values (call the loss f
unction)
        train_loss.append(logloss(y_train, y_pred))
        # store all the train loss values in a list
        # predict the output of x_test[for all data points in X_test] using w,
b
        y_pred_test = [sigmoid(np.dot(w, x)) for x in X_test]
        #compute the loss between predicted and actual values (call the loss f
unction)
        print(f"EPOCH: {epoch} Train Loss: {logloss(y_train, y_pred)} Test Los
s: {logloss(y_test, y_pred_test)}")
        # store all the test loss values in a list
        test_loss.append(logloss(y_test, y_pred_test))
        print(w,b)
        # you can also compare previous loss and current loss, if loss is not
 updating then stop the process and return w,b

    return w,b
```

In [51]:
```
alpha=0.0001
eta0=0.0001
N=len(X_train)
epochs=50
w,b=train(X_train,y_train,X_test,y_test,epochs,alpha,eta0)
```

```
EPOCH: 0 Train Loss: 0.25144227494751004 Test Loss: 0.2513654212336452
[-0.20563349  0.13203051  0.06866824  0.13751759 -0.02475651  0.2433104
 -0.2322048  -0.01373636  0.11231053  0.02100006  0.04131048 -0.00198031
  0.00777513  0.12114883  0.07052009] -0.491512930895856
EPOCH: 1 Train Loss: 0.2459770140128832 Test Loss: 0.24592538228072397
[-2.40587351e-01  1.57704754e-01  6.59429176e-02  1.63867110e-01
 -3.64931590e-02  2.80474334e-01 -2.72113289e-01 -1.63217336e-02
  1.27887561e-01  3.59099348e-02  5.04299035e-02  1.46408764e-04
  2.54915748e-03  1.46053212e-01  7.19299769e-02] -0.7117887557941109
EPOCH: 2 Train Loss: 0.2445698883752917 Test Loss: 0.24451670859455102
[-0.25037952  0.1644462   0.06596143  0.17114651 -0.03934435  0.29114008
 -0.2825514  -0.01668046  0.13297345  0.04063728  0.05243986  0.00045677
  0.00155007  0.15284849  0.07289946] -0.8163232677023791
EPOCH: 3 Train Loss: 0.24404960871689146 Test Loss: 0.24399169422583677
[-0.25403708  0.16665664  0.06675665  0.17380622 -0.03981721  0.29551688
 -0.28650705 -0.01672716  0.13513463  0.04204934  0.0529518   0.00035048
  0.00156094  0.15530756  0.07382323] -0.8671887394463159
EPOCH: 4 Train Loss: 0.24382027291120137 Test Loss: 0.24375910274058185
[-2.55633543e-01  1.67522835e-01  6.73642945e-02  1.74964159e-01
 -3.97899287e-02  2.97568831e-01 -2.88327640e-01 -1.67330812e-02
  1.36136731e-01  4.25016739e-02  5.31029733e-02  2.35372235e-04
  1.67744379e-03  1.56388727e-01  7.44032922e-02] -0.8922857541782269
EPOCH: 5 Train Loss: 0.24371116565507145 Test Loss: 0.2436481716471335
[-2.56385292e-01  1.67905613e-01  6.77198920e-02  1.75512387e-01
 -3.97065686e-02  2.98572498e-01 -2.89224852e-01 -1.67346607e-02
  1.36619853e-01  4.26645988e-02  5.31553221e-02  1.64379656e-04
  1.75793228e-03  1.56908450e-01  7.47204856e-02] -0.9047622628277987
EPOCH: 6 Train Loss: 0.2436576144428782 Test Loss: 0.24359366383011835
[-2.56751768e-01  1.68086424e-01  6.79105668e-02  1.75781098e-01
 -3.96468797e-02  2.99070428e-01 -2.89675149e-01 -1.67357477e-02
  1.36857364e-01  4.27306767e-02  5.31762647e-02  1.25999474e-04
  1.80251204e-03  1.57166360e-01  7.48852110e-02] -0.9109894108069829
EPOCH: 7 Train Loss: 0.24363100013736907 Test Loss: 0.24356656042150476
[-2.56933202e-01  1.68174658e-01  6.80092162e-02  1.75914604e-01
 -3.96124346e-02  2.99318796e-01 -2.89901816e-01 -1.67365102e-02
  1.36975283e-01  4.27600222e-02  5.31855541e-02  1.06170158e-04
  1.82564412e-03  1.57295560e-01  7.49690130e-02] -0.9141038487096876
EPOCH: 8 Train Loss: 0.2436177065385943 Test Loss: 0.24355301943253108
[-2.57023641e-01  1.68218361e-01  6.80594344e-02  1.75981286e-01
 -3.95940366e-02  2.99442976e-01 -2.90015828e-01 -1.67369793e-02
  1.37034112e-01  4.27738198e-02  5.31899332e-02  9.61033158e-05
  1.83737865e-03  1.57360415e-01  7.50112848e-02] -0.9156631469147914
EPOCH: 9 Train Loss: 0.24361105299840471 Test Loss: 0.24354624136239628
[-2.57068859e-01  1.68240150e-01  6.80848008e-02  1.76014661e-01
 -3.95845331e-02  2.99505137e-01 -2.90073104e-01 -1.67372433e-02
  1.37063532e-01  4.27805156e-02  5.31920642e-02  9.10292693e-05
  1.84328339e-03  1.57392970e-01  7.50325311e-02] -0.9164442578881155
EPOCH: 10 Train Loss: 0.24360772009167245 Test Loss: 0.24354284590997835
[-2.57091497e-01  1.68251046e-01  6.80975649e-02  1.76031378e-01
 -3.95796999e-02  2.99536271e-01 -2.90101851e-01 -1.67373845e-02
  1.37078262e-01  4.27838187e-02  5.31931175e-02  8.84796267e-05
  1.84624624e-03  1.57409303e-01  7.50431931e-02] -0.9168356527588645
EPOCH: 11 Train Loss: 0.24360604999503158 Test Loss: 0.24354114443274621
[-2.57102838e-01  1.68256501e-01  6.81039753e-02  1.76039756e-01
 -3.95772601e-02  2.99551870e-01 -2.90116271e-01 -1.67374579e-02
  1.37085641e-01  4.27854614e-02  5.31936419e-02  8.72002071e-05
  1.84773155e-03  1.57417495e-01  7.50485400e-02] -0.9170317979513392
```

```
EPOCH: 12 Train Loss: 0.24360521300297833 Test Loss: 0.24354029170440675
[-2.57108521e-01  1.68259235e-01  6.81071916e-02  1.76043954e-01
 -3.95760329e-02  2.99559687e-01 -2.90123502e-01 -1.67374954e-02
  1.37089338e-01  4.27862816e-02  5.31939040e-02  8.65585875e-05
  1.84847597e-03  1.57421602e-01  7.50512207e-02] -0.9171301018140844
EPOCH: 13 Train Loss: 0.24360479350817324 Test Loss: 0.24353986432038205
[-2.57111369e-01  1.68260604e-01  6.81088044e-02  1.76046058e-01
 -3.95754168e-02  2.99563604e-01 -2.90127126e-01 -1.67375144e-02
  1.37091191e-01  4.27866920e-02  5.31940351e-02  8.62369128e-05
  1.84884905e-03  1.57423661e-01  7.50525644e-02] -0.9171793713890335
EPOCH: 14 Train Loss: 0.24360458325501894 Test Loss: 0.24353965011254083
[-2.57112796e-01  1.68261290e-01  6.81096131e-02  1.76047113e-01
 -3.95751077e-02  2.99565568e-01 -2.90128943e-01 -1.67375240e-02
  1.37092120e-01  4.27868974e-02  5.31941008e-02  8.60756639e-05
  1.84903603e-03  1.57424694e-01  7.50532379e-02] -0.9172040655777203
EPOCH: 15 Train Loss: 0.2436044778738614 Test Loss: 0.24353954274911546
[-2.57113512e-01  1.68261634e-01  6.81100184e-02  1.76047641e-01
 -3.95749527e-02  2.99566552e-01 -2.90129854e-01 -1.67375288e-02
  1.37092585e-01  4.27870004e-02  5.31941337e-02  8.59948385e-05
  1.84912974e-03  1.57425211e-01  7.50535755e-02] -0.9172164425548942
EPOCH: 16 Train Loss: 0.24360442505545735 Test Loss: 0.2435394889371362
[-2.57113870e-01  1.68261807e-01  6.81102216e-02  1.76047906e-01
 -3.95748750e-02  2.99567045e-01 -2.90130311e-01 -1.67375312e-02
  1.37092819e-01  4.27870520e-02  5.31941502e-02  8.59543263e-05
  1.84917671e-03  1.57425470e-01  7.50537447e-02] -0.9172226460489279
EPOCH: 17 Train Loss: 0.24360439858213823 Test Loss: 0.24353946196581236
[-2.57114050e-01  1.68261893e-01  6.81103234e-02  1.76048039e-01
 -3.95748361e-02  2.99567292e-01 -2.90130540e-01 -1.67375324e-02
  1.37092935e-01  4.27870778e-02  5.31941585e-02  8.59340208e-05
  1.84920025e-03  1.57425600e-01  7.50538295e-02] -0.9172257553238788
EPOCH: 18 Train Loss: 0.24360438531332795 Test Loss: 0.24353944844739667
[-2.57114140e-01  1.68261937e-01  6.81103745e-02  1.76048105e-01
 -3.95748166e-02  2.99567416e-01 -2.90130654e-01 -1.67375330e-02
  1.37092994e-01  4.27870908e-02  5.31941626e-02  8.59238432e-05
  1.84921205e-03  1.57425665e-01  7.50538720e-02] -0.9172273137362328
EPOCH: 19 Train Loss: 0.24360437866281134 Test Loss: 0.24353944167177013
[-2.57114185e-01  1.68261958e-01  6.81104001e-02  1.76048139e-01
 -3.95748068e-02  2.99567478e-01 -2.90130712e-01 -1.67375333e-02
  1.37093023e-01  4.27870973e-02  5.31941647e-02  8.59187421e-05
  1.84921796e-03  1.57425698e-01  7.50538933e-02] -0.9172280948348762
EPOCH: 20 Train Loss: 0.24360437532947365 Test Loss: 0.2435394382757279
[-2.57114208e-01  1.68261969e-01  6.81104129e-02  1.76048155e-01
 -3.95748019e-02  2.99567509e-01 -2.90130741e-01 -1.67375335e-02
  1.37093038e-01  4.27871006e-02  5.31941658e-02  8.59161853e-05
  1.84922092e-03  1.57425714e-01  7.50539040e-02] -0.9172284863328356
EPOCH: 21 Train Loss: 0.24360437365875384 Test Loss: 0.24353943657357993
[-2.57114219e-01  1.68261975e-01  6.81104193e-02  1.76048164e-01
 -3.95747994e-02  2.99567525e-01 -2.90130755e-01 -1.67375335e-02
  1.37093046e-01  4.27871022e-02  5.31941663e-02  8.59149038e-05
  1.84922241e-03  1.57425723e-01  7.50539094e-02] -0.9172286825573066
EPOCH: 22 Train Loss: 0.2436043728213686 Test Loss: 0.2435394357204397
[-2.57114225e-01  1.68261977e-01  6.81104225e-02  1.76048168e-01
 -3.95747982e-02  2.99567533e-01 -2.90130762e-01 -1.67375336e-02
  1.37093049e-01  4.27871030e-02  5.31941665e-02  8.59142615e-05
  1.84922315e-03  1.57425727e-01  7.50539120e-02] -0.9172287809078882
EPOCH: 23 Train Loss: 0.2436043724016582 Test Loss: 0.2435394352928341
[-2.57114228e-01  1.68261979e-01  6.81104241e-02  1.76048170e-01
```

```
 -3.95747976e-02  2.99567537e-01 -2.90130766e-01 -1.67375336e-02
  1.37093051e-01  4.27871034e-02  5.31941667e-02  8.59139396e-05
  1.84922353e-03  1.57425729e-01  7.50539134e-02] -0.9172288302026285
EPOCH: 24 Train Loss: 0.24360437219129144 Test Loss: 0.2435394350785096
[-2.57114229e-01  1.68261979e-01  6.81104249e-02  1.76048171e-01
 -3.95747973e-02  2.99567539e-01 -2.90130768e-01 -1.67375336e-02
  1.37093052e-01  4.27871036e-02  5.31941667e-02  8.59137782e-05
  1.84922371e-03  1.57425730e-01  7.50539141e-02] -0.9172288549098744
EPOCH: 25 Train Loss: 0.24360437208585187 Test Loss: 0.2435394349710893
[-2.57114230e-01  1.68261980e-01  6.81104254e-02  1.76048172e-01
 -3.95747971e-02  2.99567540e-01 -2.90130769e-01 -1.67375336e-02
  1.37093053e-01  4.27871037e-02  5.31941668e-02  8.59136973e-05
  1.84922381e-03  1.57425730e-01  7.50539144e-02] -0.9172288672935104
EPOCH: 26 Train Loss: 0.24360437203300644 Test Loss: 0.2435394349172477
[-2.57114230e-01  1.68261980e-01  6.81104256e-02  1.76048172e-01
 -3.95747970e-02  2.99567540e-01 -2.90130769e-01 -1.67375336e-02
  1.37093053e-01  4.27871038e-02  5.31941668e-02  8.59136568e-05
  1.84922386e-03  1.57425731e-01  7.50539146e-02] -0.9172288735003772
EPOCH: 27 Train Loss: 0.24360437200651766 Test Loss: 0.24353943489026142
[-2.57114231e-01  1.68261980e-01  6.81104257e-02  1.76048172e-01
 -3.95747970e-02  2.99567540e-01 -2.90130769e-01 -1.67375336e-02
  1.37093053e-01  4.27871038e-02  5.31941668e-02  8.59136365e-05
  1.84922388e-03  1.57425731e-01  7.50539147e-02] -0.917228876611346
EPOCH: 28 Train Loss: 0.2436043719932405 Test Loss: 0.24353943487673518
[-2.57114231e-01  1.68261980e-01  6.81104257e-02  1.76048172e-01
 -3.95747970e-02  2.99567540e-01 -2.90130769e-01 -1.67375336e-02
  1.37093053e-01  4.27871038e-02  5.31941668e-02  8.59136263e-05
  1.84922389e-03  1.57425731e-01  7.50539147e-02] -0.917228878170618
EPOCH: 29 Train Loss: 0.24360437198658683 Test Loss: 0.24353943486995713
[-2.57114231e-01  1.68261980e-01  6.81104257e-02  1.76048172e-01
 -3.95747970e-02  2.99567540e-01 -2.90130769e-01 -1.67375336e-02
  1.37093053e-01  4.27871038e-02  5.31941668e-02  8.59136212e-05
  1.84922390e-03  1.57425731e-01  7.50539147e-02] -0.9172288789521417
EPOCH: 30 Train Loss: 0.24360437198325405 Test Loss: 0.2435394348665587
[-2.57114231e-01  1.68261980e-01  6.81104257e-02  1.76048172e-01
 -3.95747970e-02  2.99567540e-01 -2.90130769e-01 -1.67375336e-02
  1.37093053e-01  4.27871038e-02  5.31941668e-02  8.59136186e-05
  1.84922390e-03  1.57425731e-01  7.50539147e-02] -0.9172288793438514
EPOCH: 31 Train Loss: 0.24360437198158053 Test Loss: 0.2435394348648552
[-2.57114231e-01  1.68261980e-01  6.81104258e-02  1.76048172e-01
 -3.95747969e-02  2.99567540e-01 -2.90130769e-01 -1.67375336e-02
  1.37093053e-01  4.27871038e-02  5.31941668e-02  8.59136173e-05
  1.84922390e-03  1.57425731e-01  7.50539147e-02] -0.9172288795401733
EPOCH: 32 Train Loss: 0.24360437198074256 Test Loss: 0.24353943486400192
[-2.57114231e-01  1.68261980e-01  6.81104258e-02  1.76048172e-01
 -3.95747969e-02  2.99567541e-01 -2.90130769e-01 -1.67375336e-02
  1.37093053e-01  4.27871038e-02  5.31941668e-02  8.59136167e-05
  1.84922390e-03  1.57425731e-01  7.50539147e-02] -0.9172288796385855
EPOCH: 33 Train Loss: 0.24360437198032225 Test Loss: 0.24353943486357357
[-2.57114231e-01  1.68261980e-01  6.81104258e-02  1.76048172e-01
 -3.95747969e-02  2.99567541e-01 -2.90130769e-01 -1.67375336e-02
  1.37093053e-01  4.27871038e-02  5.31941668e-02  8.59136164e-05
  1.84922390e-03  1.57425731e-01  7.50539147e-02] -0.9172288796879
EPOCH: 34 Train Loss: 0.24360437198011214 Test Loss: 0.24353943486335905
[-2.57114231e-01  1.68261980e-01  6.81104258e-02  1.76048172e-01
 -3.95747969e-02  2.99567541e-01 -2.90130769e-01 -1.67375336e-02
  1.37093053e-01  4.27871038e-02  5.31941668e-02  8.59136162e-05
```

```
  1.84922390e-03  1.57425731e-01  7.50539147e-02] -0.917228879712628
EPOCH: 35 Train Loss: 0.24360437198000578 Test Loss: 0.2435394348632519
[-2.57114231e-01  1.68261980e-01  6.81104258e-02  1.76048172e-01
 -3.95747969e-02  2.99567541e-01 -2.90130769e-01 -1.67375336e-02
  1.37093053e-01  4.27871038e-02  5.31941668e-02  8.59136161e-05
  1.84922390e-03  1.57425731e-01  7.50539147e-02] -0.9172288797250095
EPOCH: 36 Train Loss: 0.24360437197995666 Test Loss: 0.24353943486319873
[-2.57114231e-01  1.68261980e-01  6.81104258e-02  1.76048172e-01
 -3.95747969e-02  2.99567541e-01 -2.90130769e-01 -1.67375336e-02
  1.37093053e-01  4.27871038e-02  5.31941668e-02  8.59136161e-05
  1.84922390e-03  1.57425731e-01  7.50539147e-02] -0.9172288797312259
EPOCH: 37 Train Loss: 0.2436043719799283 Test Loss: 0.2435394348631709
[-2.57114231e-01  1.68261980e-01  6.81104258e-02  1.76048172e-01
 -3.95747969e-02  2.99567541e-01 -2.90130769e-01 -1.67375336e-02
  1.37093053e-01  4.27871038e-02  5.31941668e-02  8.59136161e-05
  1.84922390e-03  1.57425731e-01  7.50539147e-02] -0.9172288797343364
EPOCH: 38 Train Loss: 0.2436043719799146 Test Loss: 0.2435394348631579
[-2.57114231e-01  1.68261980e-01  6.81104258e-02  1.76048172e-01
 -3.95747969e-02  2.99567541e-01 -2.90130769e-01 -1.67375336e-02
  1.37093053e-01  4.27871038e-02  5.31941668e-02  8.59136161e-05
  1.84922390e-03  1.57425731e-01  7.50539147e-02] -0.917228879735897
EPOCH: 39 Train Loss: 0.24360437197990784 Test Loss: 0.24353943486315124
[-2.57114231e-01  1.68261980e-01  6.81104258e-02  1.76048172e-01
 -3.95747969e-02  2.99567541e-01 -2.90130769e-01 -1.67375336e-02
  1.37093053e-01  4.27871038e-02  5.31941668e-02  8.59136161e-05
  1.84922390e-03  1.57425731e-01  7.50539147e-02] -0.9172288797366835
EPOCH: 40 Train Loss: 0.24360437197990373 Test Loss: 0.24353943486314794
[-2.57114231e-01  1.68261980e-01  6.81104258e-02  1.76048172e-01
 -3.95747969e-02  2.99567541e-01 -2.90130769e-01 -1.67375336e-02
  1.37093053e-01  4.27871038e-02  5.31941668e-02  8.59136161e-05
  1.84922390e-03  1.57425731e-01  7.50539147e-02] -0.9172288797370781
EPOCH: 41 Train Loss: 0.24360437197990142 Test Loss: 0.24353943486314608
[-2.57114231e-01  1.68261980e-01  6.81104258e-02  1.76048172e-01
 -3.95747969e-02  2.99567541e-01 -2.90130769e-01 -1.67375336e-02
  1.37093053e-01  4.27871038e-02  5.31941668e-02  8.59136161e-05
  1.84922390e-03  1.57425731e-01  7.50539147e-02] -0.9172288797372763
EPOCH: 42 Train Loss: 0.24360437197990026 Test Loss: 0.2435394348631451
[-2.57114231e-01  1.68261980e-01  6.81104258e-02  1.76048172e-01
 -3.95747969e-02  2.99567541e-01 -2.90130769e-01 -1.67375336e-02
  1.37093053e-01  4.27871038e-02  5.31941668e-02  8.59136161e-05
  1.84922390e-03  1.57425731e-01  7.50539147e-02] -0.917228879737374
EPOCH: 43 Train Loss: 0.2436043719798999 Test Loss: 0.2435394348631446
[-2.57114231e-01  1.68261980e-01  6.81104258e-02  1.76048172e-01
 -3.95747969e-02  2.99567541e-01 -2.90130769e-01 -1.67375336e-02
  1.37093053e-01  4.27871038e-02  5.31941668e-02  8.59136161e-05
  1.84922390e-03  1.57425731e-01  7.50539147e-02] -0.9172288797374236
EPOCH: 44 Train Loss: 0.24360437197989943 Test Loss: 0.24353943486314447
[-2.57114231e-01  1.68261980e-01  6.81104258e-02  1.76048172e-01
 -3.95747969e-02  2.99567541e-01 -2.90130769e-01 -1.67375336e-02
  1.37093053e-01  4.27871038e-02  5.31941668e-02  8.59136161e-05
  1.84922390e-03  1.57425731e-01  7.50539147e-02] -0.9172288797374492
EPOCH: 45 Train Loss: 0.24360437197989926 Test Loss: 0.24353943486314436
[-2.57114231e-01  1.68261980e-01  6.81104258e-02  1.76048172e-01
 -3.95747969e-02  2.99567541e-01 -2.90130769e-01 -1.67375336e-02
  1.37093053e-01  4.27871038e-02  5.31941668e-02  8.59136161e-05
  1.84922390e-03  1.57425731e-01  7.50539147e-02] -0.9172288797374631
EPOCH: 46 Train Loss: 0.24360437197989931 Test Loss: 0.24353943486314433
```

```
[-2.57114231e-01  1.68261980e-01  6.81104258e-02  1.76048172e-01
 -3.95747969e-02  2.99567541e-01 -2.90130769e-01 -1.67375336e-02
  1.37093053e-01  4.27871038e-02  5.31941668e-02  8.59136161e-05
  1.84922390e-03  1.57425731e-01  7.50539147e-02] -0.9172288797374706
EPOCH: 47 Train Loss: 0.24360437197989931 Test Loss: 0.24353943486314425
[-2.57114231e-01  1.68261980e-01  6.81104258e-02  1.76048172e-01
 -3.95747969e-02  2.99567541e-01 -2.90130769e-01 -1.67375336e-02
  1.37093053e-01  4.27871038e-02  5.31941668e-02  8.59136161e-05
  1.84922390e-03  1.57425731e-01  7.50539147e-02] -0.9172288797374738
EPOCH: 48 Train Loss: 0.24360437197989926 Test Loss: 0.24353943486314425
[-2.57114231e-01  1.68261980e-01  6.81104258e-02  1.76048172e-01
 -3.95747969e-02  2.99567541e-01 -2.90130769e-01 -1.67375336e-02
  1.37093053e-01  4.27871038e-02  5.31941668e-02  8.59136161e-05
  1.84922390e-03  1.57425731e-01  7.50539147e-02] -0.9172288797374752
EPOCH: 49 Train Loss: 0.24360437197989926 Test Loss: 0.24353943486314425
[-2.57114231e-01  1.68261980e-01  6.81104258e-02  1.76048172e-01
 -3.95747969e-02  2.99567541e-01 -2.90130769e-01 -1.67375336e-02
  1.37093053e-01  4.27871038e-02  5.31941668e-02  8.59136161e-05
  1.84922390e-03  1.57425731e-01  7.50539147e-02] -0.9172288797374758
```

Goal of assignment

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 [49]:
```
# these are the results we got after we implemented sgd and found the optimal
 weights and intercept
w-clf.coef_, b-clf.intercept_
```

Out[49]:
```
(array([[ 0.6329576 , -0.46336165,  0.14405187, -0.4550229 ,  0.34476895,
         -0.63278489,  0.60560444,  0.05666768, -0.26882112, -0.377212  ,
         -0.19402726, -0.05037607,  0.09062909, -0.38339079,  0.00861503]]),
 array([0.3885765]))
```

Plot epoch number vs train , test loss

- epoch number on X-axis
- loss on Y-axis

In [52]:
```
loss_train = history.history['train_loss']
loss_val = history.history['test_loss']
epochs = range(1,35)
plt.plot(epochs, loss_train, 'g', label='Training loss')
plt.plot(epochs, loss_val, 'b', label='validation loss')
plt.title('Training and Validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-52-2626947a62bf> in <module>
----> 1 loss_train = history.history['train_loss']
      2 loss_val = history.history['test_loss']
      3 epochs = range(1,35)
      4 plt.plot(epochs, loss_train, 'g', label='Training loss')
      5 plt.plot(epochs, loss_val, 'b', label='validation loss')

NameError: name 'history' is not defined
```

In [48]:
```
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: # 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(w,b,X_train))/len(X_train))
print(1-np.sum(y_test  - pred(w,b,X_test))/len(X_test))
```

```
0.8016266666666667
0.8022400000000001
```

In [ ]:

In [ ]: