[udaylunawat@gmail.com (mailto:udaylunawat@gmail.com)](mailto:udaylunawat@gmail.com)

In [1]:
```
import numpy as np
import pandas as pd
from sklearn.datasets import make_classification
import matplotlib.pyplot as plt
```

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]:
```
from sklearn.model_selection import train_test_split
```

In [5]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=15)
```

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

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

In [7]:
```
from sklearn import linear_model
```

In [8]:

```
# alpha : float
# Constant that multiplies the regularization term.

# eta0 : double
# The initial learning rate for the ‘constant’, ‘invscaling’ or ‘adaptive’ schedules.

clf = linear_model.SGDClassifier(eta0=0.0001, alpha=0.0001, loss='log', random_state=1, penalty='l2', tol=1e-3, verbose=
clf
```

Out[8]:

```
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=1, shuffle=True,
              tol=0.001, validation_fraction=0.1, verbose=2, warm_start=False)
```

In [9]:
```
clf.fit(X=X_train, y=y_train)
```

```
-- Epoch 1
Norm: 0.76, NNZs: 15, Bias: -0.314219, T: 37500, Avg. loss: 0.456332
Total training time: 0.01 seconds.
-- Epoch 2
Norm: 0.91, NNZs: 15, Bias: -0.471117, T: 75000, Avg. loss: 0.394828
Total training time: 0.02 seconds.
-- Epoch 3
Norm: 0.98, NNZs: 15, Bias: -0.581147, T: 112500, Avg. loss: 0.385666
Total training time: 0.03 seconds.
-- Epoch 4
Norm: 1.02, NNZs: 15, Bias: -0.660030, T: 150000, Avg. loss: 0.382073
Total training time: 0.04 seconds.
-- Epoch 5
Norm: 1.04, NNZs: 15, Bias: -0.719106, T: 187500, Avg. loss: 0.380409
Total training time: 0.04 seconds.
-- Epoch 6
Norm: 1.05, NNZs: 15, Bias: -0.763106, T: 225000, Avg. loss: 0.379544
Total training time: 0.05 seconds.
-- Epoch 7
Norm: 1.06, NNZs: 15, Bias: -0.794459, T: 262500, Avg. loss: 0.379092
Total training time: 0.06 seconds.
-- Epoch 8
Norm: 1.07, NNZs: 15, Bias: -0.819748, T: 300000, Avg. loss: 0.378926
Total training time: 0.07 seconds.
-- Epoch 9
Norm: 1.07, NNZs: 15, Bias: -0.837385, T: 337500, Avg. loss: 0.378727
Total training time: 0.08 seconds.
-- Epoch 10
Norm: 1.08, NNZs: 15, Bias: -0.851748, T: 375000, Avg. loss: 0.378574
Total training time: 0.08 seconds.
Convergence after 10 epochs took 0.08 seconds
```

Out[9]:
```
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=1, shuffle=True,
              tol=0.001, validation_fraction=0.1, verbose=2, warm_start=False)
```

In [10]:
```
clf.coef_, clf.coef_.shape, clf.intercept_
```

Out[10]:
```
(array([[-0.4167019 ,  0.18364588, -0.14293025,  0.34812889, -0.21566194,
          0.56418028, -0.44306199, -0.09788853,  0.21219686,  0.17592993,
          0.1925475 ,  0.0017679 , -0.08466201,  0.34080004,  0.03251288]]),
 (1, 15),
 array([-0.85174791]))
```

In [11]:
```
clf.intercept_[0]
```

Out[11]:
```
-0.8517479126246573
```

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)):

$$w^{(t+1)} \leftarrow (1 - \frac{\alpha\lambda}{N})w^{(t)} + \alpha x_n(y_n - \sigma((w^{(t)})^T x_n + b^t))$$
$$b^{(t+1)} \leftarrow (1 - \frac{\alpha\lambda}{N})b^{(t)} + \alpha(y_n - \sigma((w^{(t)})^T x_n + b^t))$$

  - 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 [236]:
```
w = np.zeros_like(X_train[0])
b = 0
eta0  = 0.0001
alpha = 0.0001
N = len(X_train)
```

In [163]:
```
def sigmoid(w,x,b):
    return 1/(1+np.exp(-(np.dot(x,w)+b)))
```

In [164]:
```
def predict(w,b,x):
    pred = []
    for i in range(len(x)):
        pred.append(sig(w, X[i], b))
    return np.array(pred)
```

In [165]:

```
import math
def compute_log_loss(true,pred):
    loss = 0
    for t,p in zip(true,pred):
        y1 = t
        y2 = p
        loss += (y1* np.log(y2)) + ((1-y1)* np.log(1-y2))
        # loss = float("{0:.4f}".format(loss))
    return (-1*(loss)/len(true))
```

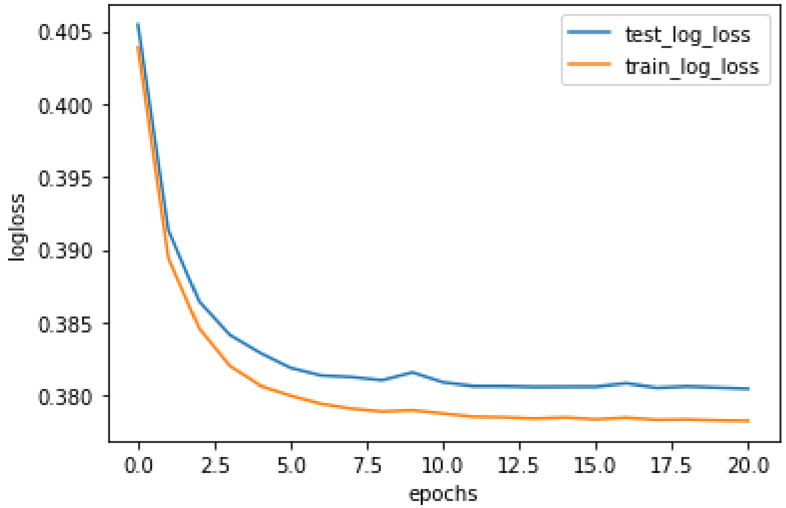
$w^{(t+1)} \leftarrow (1 - \frac{\alpha\lambda}{N})w^{(t)} + \alpha x_n(y_n - \sigma((w^{(t)})^T x_n + b^t))$
$b^{(t+1)} \leftarrow (1 - \frac{\alpha\lambda}{N})b^{(t)} + \alpha(y_n - \sigma((w^{(t)})^T x_n + b^t))$ "

In [237]:
```python
from tqdm import tqdm
a = alpha
l = eta0
train_loss = []
test_loss = []
print("epoch\t log loss")

for epoch in range(30):
    for j in range(N):
        r = np.random.randint(N)
        Xn = X_train[r]
        yn = y_train[r]

        weight_update = ( 1- (a*l)/N) *w + (a*(Xn*(yn-sigmoid(w,Xn,b))))
        intercept_update = (1 - (a*l)/N) *b + (a*(yn-sigmoid(w,Xn,b)))

        w = weight_update
        b = intercept_update

    y_train_pred = map(lambda i: sigmoid(w,i,b), X_train)
    y_test_pred = map(lambda i: sigmoid(w,i,b), X_test)

    loss = compute_log_loss(y_train,y_train_pred)

    train_loss.append(loss)
    test_loss.append(compute_log_loss(y_test,y_test_pred))

    print(epoch,'\t',loss)

    if abs(b-clf.intercept_)[0] < 0.01 and np.allclose(w,clf.coef_,rtol = 1e-01, atol = 1e-02):
        print("converged")
        break
```

```
epoch    log loss
0        0.40433005648347553
1        0.3884187924554347
2        0.38344655505551983
3        0.3807626284133473
4        0.3795749054664712
5        0.37897757910248303
```

```
6         0.3787680772563091
7         0.378490169081829
8         0.3783933590871815
9         0.379236863992006
converged
```

In [213]:
```
plt.plot(test_loss,label='test_log_loss')
plt.plot(train_loss,label='train_log_loss')
plt.legend()
plt.xlabel('epochs')
plt.ylabel('logloss')
plt.show();
```



In [238]:
```
print(w,b)
```

```
[-0.42675708  0.18378038 -0.13647368  0.33474876 -0.21133386  0.56084653
 -0.43580897 -0.09492469  0.23051905  0.17307646  0.18122995  0.00582301
 -0.07340738  0.33828899  0.02947331] -0.8485635519484128
```

In [215]:
```
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(-(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))
```

```
0.9552266666666667
0.95336
```

In [216]:
```
print(1-np.sum(y_train - pred(clf.coef_[0], clf.intercept_,X_train))/len(X_train))
print(1-np.sum(y_test  - pred(clf.coef_[0], clf.intercept_,X_test))/len(X_test))
```

```
0.95288
0.95256
```

In [217]:
```
print(w-clf.coef_)
print(b-clf.intercept_)
np.allclose(w,clf.coef_,rtol = 1e-02, atol = 1e-02)
```

```
[[-0.00353859  0.00196883 -0.00089057 -0.00210887  0.00644648 -0.00087634
  -0.00680417  0.00294099  0.00227197  0.00071759 -0.00043365  0.00125569
   0.00174226  0.0014     -0.00426724]]
[0.00861239]
```

Out[217]:
```
True
```

In [ ]: