In [0]:

```
import numpy as np
import pandas as pd
from sklearn.datasets import make_classification
```

In [0]:

```
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 [0]:

```
from sklearn.model_selection import train_test_split
```

In [0]:

```
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 [0]:

```
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=15, penalty='l2',
tol=1e-3, verbose=2, learning_rate='constant')
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=15, 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: 0.77, 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: 0.02 seconds.
-- Epoch 3
Norm: 0.98, NNZs: 15, Bias: -0.580082, T: 112500, Avg. loss: 0.385711
Total training time: 0.03 seconds.
-- Epoch 4
Norm: 1.02, NNZs: 15, Bias: -0.658292, T: 150000, Avg. loss: 0.382083
Total training time: 0.04 seconds.
-- Epoch 5
Norm: 1.04, NNZs: 15, Bias: -0.719528, T: 187500, Avg. loss: 0.380486
Total training time: 0.05 seconds.
-- Epoch 6
Norm: 1.05, NNZs: 15, Bias: -0.763409, T: 225000, Avg. loss: 0.379578
Total training time: 0.05 seconds.
-- Epoch 7
Norm: 1.06, NNZs: 15, Bias: -0.795106, T: 262500, Avg. loss: 0.379150
Total training time: 0.06 seconds.
-- Epoch 8
Norm: 1.06, NNZs: 15, Bias: -0.819925, T: 300000, Avg. loss: 0.378856
Total training time: 0.07 seconds.
-- Epoch 9
Norm: 1.07, NNZs: 15, Bias: -0.837805, T: 337500, Avg. loss: 0.378585
Total training time: 0.08 seconds.
-- Epoch 10
Norm: 1.08, NNZs: 15, Bias: -0.853138, T: 375000, Avg. loss: 0.378630
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=15, 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([[-0.42336692,  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 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
    - Update weights and intercept (check the equation number 32 in the above mentioned pdf):
      $w^{(t+1)} ← (1 − \frac{αλ}{N})w^{(t)} + αx_n(y_n − σ((w^{(t)})^{T} x_n+b^{t}))$
      $b^{(t+1)} ← (b^t + α(y_n - σ((w^{(t)})^{T} x_n+b^{t}))$
    - 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 [0]:

```
import numpy as np
```

Initializing W,B,eta0 and alpha

In [12]:

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

Out[12]:

37500

Computing Log-loss

In [0]:

```
import math
# you can free to change all these codes/structure
def compute_log_loss(y_train,pred):
    sum1 = 0
    for i in range(len(pred)):

        sum1 += ((y_train[i] * math.log(pred[i])) + ((1-y_train[i]) * math.log((1-pred[i]))))

    loss = (-sum1/len(y_train))
    return loss
```

Sigmoid and predict functions

In [14]:

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

def predict(X_train,w,b):
    pred = []
```

```
    for i in range(len(X_train)):
         pred.append(sigmoid(X_train[i],w,b))

    return pred

train_loss = []
test_loss  = []

#print(pred[:5])
train_pred = predict(X_train,w,b)
test_pred = predict(X_test,w,b)

#Computing log-loss
train_loss.append(compute_log_loss(y_train,train_pred))
test_loss.append(compute_log_loss(y_test,test_pred))
print(train_loss[0])
print(test_loss[0])
```

```
0.6931471805594285
0.6931471805600672
```

SGD with train and test loss

In [15]:

```
import random

for epoch in (range(100)):
    for i in range(N):
        batch = random.randrange(1,N)

        w = (( 1 - ( (alpha*eta0)/N) ) * w ) + ( (alpha*X_train[batch]) * ( y_train[batch] - sigmoid( X
_train[batch],w, b) ) )
        #w = (1 - ( (alpha * eta0)/N ) * w ) + ( ( alpha * X_train[batch] ) * ( y_train[batch] - sigmoi
d(X_train[batch],w,b ) ) )
        #b = b + (alpha * (y_train[batch] - sigmoid(X_train,w,b)))
        b = (b - ( alpha * ( -(y_train[batch]) + sigmoid(X_train[batch],w, b) ) ))


    y_train_ep = predict(X_train,w,b)
    y_test_ep = predict(X_test,w,b)


    train_loss.append(compute_log_loss(y_train,y_train_ep))
    test_loss.append(compute_log_loss(y_test,y_test_ep))


    print("Epoch",epoch,"train_loss",train_loss[-1:],'test_loss',test_loss[-1:])
```

```
Epoch 0 train_loss [0.4037381695280155] test_loss [0.4051409460739654]
Epoch 1 train_loss [0.3881853608997544] test_loss [0.3898325942293472]
Epoch 2 train_loss [0.3829884680624984] test_loss [0.3850343141489612]
Epoch 3 train_loss [0.3810364322337181] test_loss [0.3829650524208779]
Epoch 4 train_loss [0.3793307474020105] test_loss [0.3811837756963011]
Epoch 5 train_loss [0.37899058749291276] test_loss [0.38093730513735213]
Epoch 6 train_loss [0.37860451654347943] test_loss [0.3809161630106351]
Epoch 7 train_loss [0.37859248967793074] test_loss [0.3806523284867069]
Epoch 8 train_loss [0.37839479771533124] test_loss [0.38062307300682996]
Epoch 9 train_loss [0.3785997381435557] test_loss [0.3801598326075643]
Epoch 10 train_loss [0.37875586516661724] test_loss [0.3808460189708544]
Epoch 11 train_loss [0.3785334950483176] test_loss [0.38043706353596246]
Epoch 12 train_loss [0.378319311341773] test_loss [0.38046905269524517]
Epoch 13 train_loss [0.37849586907479216] test_loss [0.38106104382028905]
Epoch 14 train_loss [0.37844457709883084] test_loss [0.38077869343296433]
Epoch 15 train_loss [0.37828723193219166] test_loss [0.38057143966617346]
Epoch 16 train_loss [0.3782556257651049] test_loss [0.3802836106482321]
Epoch 17 train_loss [0.37828596979548795] test_loss [0.38025606448603394]
Epoch 18 train_loss [0.378790052102815] test_loss [0.3806353954847428]
Epoch 19 train_loss [0.3785487186734486] test_loss [0.3806716857464192]
```

```
Epoch 20 train_loss [0.37834039873327374] test_loss [0.380488759085167]
Epoch 21 train_loss [0.37856679503859286] test_loss [0.3808323157183337]
Epoch 22 train_loss [0.3784557350099683] test_loss [0.3805610330744679]
Epoch 23 train_loss [0.3782690945458786] test_loss [0.3801988431722665]
Epoch 24 train_loss [0.37833664657542676] test_loss [0.38059116355377753]
Epoch 25 train_loss [0.3784389083729466] test_loss [0.3804360786727016]
Epoch 26 train_loss [0.3784692967084271] test_loss [0.38015570499625095]
Epoch 27 train_loss [0.37841150141590435] test_loss [0.3806889376559375]
Epoch 28 train_loss [0.37831904368221814] test_loss [0.38042784510305055]
Epoch 29 train_loss [0.37826504783083614] test_loss [0.3801105079157159]
Epoch 30 train_loss [0.3784010305318321] test_loss [0.3802748021726142]
Epoch 31 train_loss [0.37868579423054605] test_loss [0.3806650095170454]
Epoch 32 train_loss [0.37839632632599973] test_loss [0.38008283843529606]
Epoch 33 train_loss [0.37860429504902243] test_loss [0.38096074308666156]
Epoch 34 train_loss [0.3783298607628876] test_loss [0.3804859460392227]
Epoch 35 train_loss [0.37824503502500256] test_loss [0.38015237314926836]
Epoch 36 train_loss [0.37879789758874044] test_loss [0.3803794993881466]
Epoch 37 train_loss [0.3785472200787773] test_loss [0.38040218673309906]
Epoch 38 train_loss [0.37838983389721187] test_loss [0.3802951235580614]
Epoch 39 train_loss [0.378407534818248] test_loss [0.38084593052827237]
Epoch 40 train_loss [0.37849607363090726] test_loss [0.38018167722818375]
Epoch 41 train_loss [0.37887814382749574] test_loss [0.3805813042564888]
Epoch 42 train_loss [0.37844404692467415] test_loss [0.38050030006947005]
Epoch 43 train_loss [0.3786995542218357] test_loss [0.38059106130396336]
Epoch 44 train_loss [0.3782801139447077] test_loss [0.3806497539785946]
Epoch 45 train_loss [0.3785891608233322] test_loss [0.3808594095194392]
Epoch 46 train_loss [0.37838072581047594] test_loss [0.3804360100608189]
Epoch 47 train_loss [0.3781714066728731] test_loss [0.38016317734765137]
Epoch 48 train_loss [0.37828268136228665] test_loss [0.38043582085017064]
Epoch 49 train_loss [0.3783174939444274] test_loss [0.38047780731136643]
Epoch 50 train_loss [0.37842365245193715] test_loss [0.3803371019224976]
Epoch 51 train_loss [0.3783860620355435] test_loss [0.3801683995299245]
Epoch 52 train_loss [0.37838055985588154] test_loss [0.3803110665815696]
Epoch 53 train_loss [0.3783420480114288] test_loss [0.380476787936208]
Epoch 54 train_loss [0.37865727386751236] test_loss [0.3804228973577421]
Epoch 55 train_loss [0.3782474839632092] test_loss [0.3804848064206839]
Epoch 56 train_loss [0.3782923586710222] test_loss [0.3804011586827127]
Epoch 57 train_loss [0.378460508474135] test_loss [0.380454314739438]
Epoch 58 train_loss [0.3785546435055251] test_loss [0.38099400105225767]
Epoch 59 train_loss [0.3783692168838415] test_loss [0.3808645734690277]
Epoch 60 train_loss [0.3790874604560089] test_loss [0.38078547195640866]
Epoch 61 train_loss [0.378385870247855] test_loss [0.38066898640585073]
Epoch 62 train_loss [0.37830267246996796] test_loss [0.38053803311555623]
Epoch 63 train_loss [0.37824347026778515] test_loss [0.38021824983708474]
Epoch 64 train_loss [0.3784408428316232] test_loss [0.38043835737265297]
Epoch 65 train_loss [0.37836581233019456] test_loss [0.38068339216397623]
Epoch 66 train_loss [0.37845100227737377] test_loss [0.3807824798604219]
Epoch 67 train_loss [0.37847992231806377] test_loss [0.3804682657252031]
Epoch 68 train_loss [0.3784411767775674] test_loss [0.38092653904334467]
Epoch 69 train_loss [0.3782921860759299] test_loss [0.38020459935435513]
Epoch 70 train_loss [0.3785881508556596] test_loss [0.3809583408438084]
Epoch 71 train_loss [0.3783378803963715] test_loss [0.3803961820293535]
Epoch 72 train_loss [0.37826974638864536] test_loss [0.38034150901414987]
Epoch 73 train_loss [0.378530156100854] test_loss [0.3807972266983999]
Epoch 74 train_loss [0.37858045151545444] test_loss [0.38051774743326866]
Epoch 75 train_loss [0.37820712538405743] test_loss [0.38036716256505965]
Epoch 76 train_loss [0.3784912937264594] test_loss [0.38031813672802245]
Epoch 77 train_loss [0.3784848584366938] test_loss [0.38055749435134995]
Epoch 78 train_loss [0.3783430587083698] test_loss [0.3807588672280358]
Epoch 79 train_loss [0.37837175280467616] test_loss [0.38047390286230287]
Epoch 80 train_loss [0.3784321673710929] test_loss [0.3807268129664003]
Epoch 81 train_loss [0.3788426523451305] test_loss [0.3811161603145213]
Epoch 82 train_loss [0.3787392300059855] test_loss [0.3810545793385992]
Epoch 83 train_loss [0.37859481559275476] test_loss [0.38074771166607935]
Epoch 84 train_loss [0.37836201947462694] test_loss [0.3805159208115878]
Epoch 85 train_loss [0.3784248806056946] test_loss [0.3805133354351515]
Epoch 86 train_loss [0.3786555140887769] test_loss [0.3814232068641636]
Epoch 87 train_loss [0.37863614401627127] test_loss [0.3812271984515548]
Epoch 88 train_loss [0.3785168930344105] test_loss [0.38013927520184143]
Epoch 89 train_loss [0.3786452168124955] test_loss [0.3808409840228766]
Epoch 90 train_loss [0.3786334208030699] test_loss [0.38057041732211777]
Epoch 91 train_loss [0.3785733640230413] test_loss [0.38056684276704683]
Epoch 92 train_loss [0.3784738321160625] test_loss [0.3809593605655751]
Epoch 93 train_loss [0.37839491220362464] test_loss [0.3807692128526246]
Epoch 94 train_loss [0.37847350866712937] test_loss [0.38017437066722504]
Epoch 95 train_loss [0.3783669260998517] test_loss [0.3805901461418767]
Epoch 96 train_loss [0.37831901692810516] test_loss [0.38038730964755211]
```

```
Epoch 97 train_loss [0.37850949084546265] test_loss [0.38023338233336923]
Epoch 98 train_loss [0.3782134534745643] test_loss [0.3802217437863191]
Epoch 99 train_loss [0.3784287208870628] test_loss [0.3803246990204749]
```

Plot of train-loss and test-loss

In [16]:

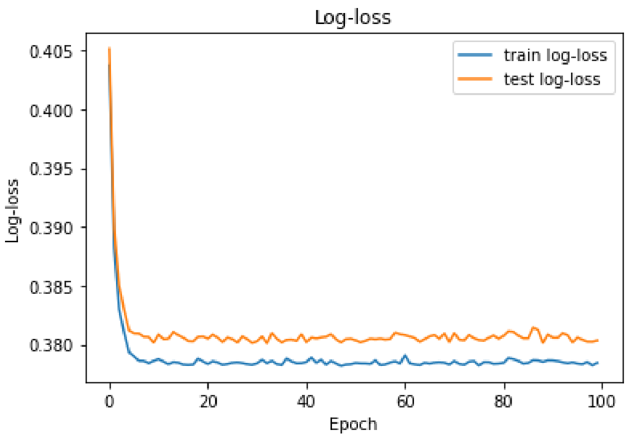
```python
import matplotlib.pyplot as plt

#plt.figure(figsize=(9,6))

plt.plot(train_loss[1:], label="train log-loss")
plt.plot(test_loss[1:], label="test log-loss")

plt.xlabel("Epoch")
plt.ylabel("Log-loss")
plt.legend()
plt.title("Log-loss")

plt.show()
```



Weight and intercept

In [17]:

```python
print("Weight vector(W) :",w)
print('Intercept(B) :',b)
```

```
Weight vector(W) : [-0.43026846  0.19244311 -0.14315103  0.34086735 -0.22428848  0.56679028
 -0.44654246 -0.09243658  0.22266273  0.17870868  0.20592653 -0.00220059
 -0.0813303   0.33423687  0.03268197]
Intercept(B) : -0.891685729814871
```

Diff btw skcit learn and custom implemented weights

In [18]:

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

Out[18]:

```
(array([[-0.00690154,  0.00696746,  0.00543933, -0.00057672, -0.01610178,
          0.0066245 ,  0.00588236,  0.00165155,  0.01338953, -0.00213258,
          0.00887463, -0.00641975, -0.0017266 , -0.00429115,  0.01001476]]),
 array([-0.03854743]))
```

Calculating accuracy

In [19]:

```
def pred(w,b, X):
    N = len(X)
    predict = []
    for i in range(N):
        if 1/(1+np.exp(-(np.dot(X[i],w)+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("Train accuracy : ",1-np.sum(y_train - pred(w,b,X_train))/len(X_train))
print("Test accuracy :",1-np.sum(y_test  - pred(w,b,X_test))/len(X_test))
```

```
Train accuracy :  0.9541866666666666
Test accuracy : 0.95256
```