Binary Classification based on Logistic Regression

September 24, 2019

1 Binary Classification based on Logistic Regression

20166450 YoungminKim

```
In [302]: import matplotlib.pyplot as plt
          from random import *
          import numpy as np
```

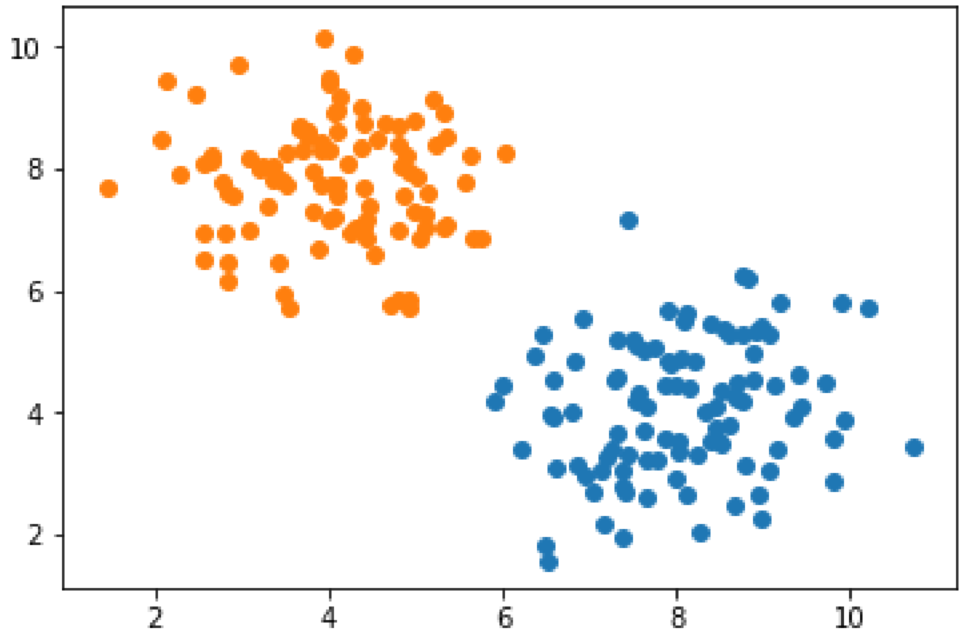
1.1 1. Plot two clusters of points for training dateset

- Generate two sets of separable random point clusters in R^2
- Let ${ x_i }_{i=1}^n $ be a set of points and $\{y_i\}_{i=1}^n$ be their corresponding labels Plot the point clusters in the training dataset using different colors depending on their labels

```
In [621]: s1=8
          t1=4
          s2=4
          t2=8
```

```
In [622]: x1_train=np.random.randn(100)
          y1_train=np.random.randn(100)
          x2_train=np.random.randn(100)
          y2_train=np.random.randn(100)
```

```
In [623]: xy0_train=np.column_stack([s1-x1_train,t1-y1_train])
          xy1_train=np.column_stack([s2-x2_train,t2-y2_train])
          one_arr=np.array([1.0]*100)
          xy0_train_plus1=np.column_stack([s1-x1_train,t1-y1_train,one_arr])
          xy1_train_plus1=np.column_stack([s2-x2_train,t2-y2_train,one_arr])
          plt.scatter(s1-x1_train,t1-y1_train)
          plt.scatter(s2-x2_train,t2-y2_train)
          plt.show()
```



1.2 2. Plot two clusters of points for testing dataset

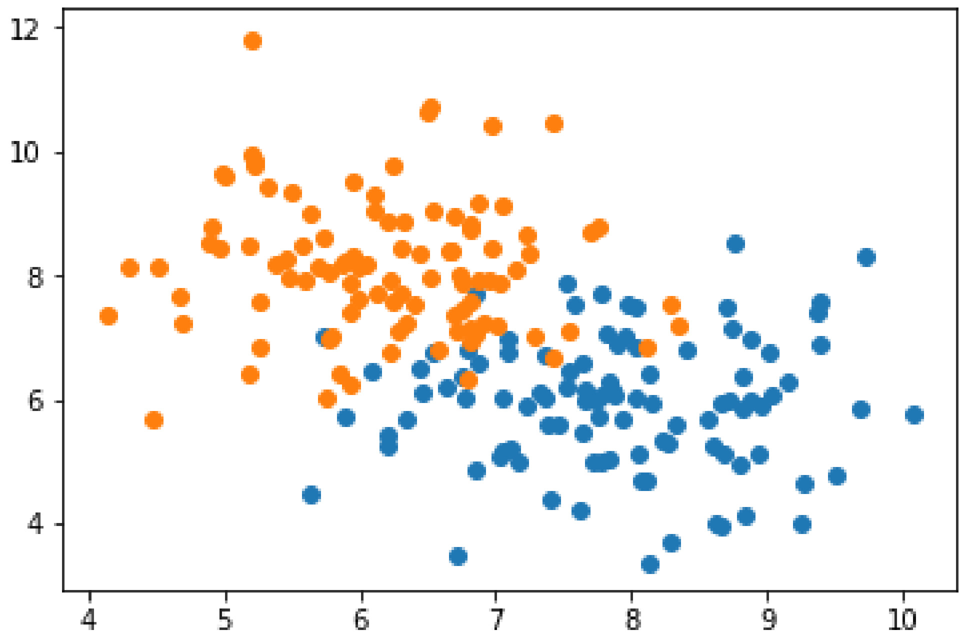
- Generate two sets of separable random point clusters in R^2 for a testing dataset using the same centroid and the standard deviation of random generator as the training dataset
- Plot the point clusters in the testing dataset using different colors depending on their labels (different colors from the training dataset)

In [624]:
```
s1_test=8
t1_test=6
s2_test=6
t2_test=8
```

In [625]:
```
x1_test=np.random.randn(100)
y1_test=np.random.randn(100)
x2_test=np.random.randn(100)
y2_test=np.random.randn(100)
```

- Each of Tesing data sets is nearer than Training data sets.
- Set it on purpose

In [626]:
```
xy0_test=np.column_stack([s1_test-x1_test,t1_test-y1_test])
xy1_test=np.column_stack([s2_test-x2_test,t2_test-y2_test])
xy0_test_plus1=np.column_stack([s1_test-x1_test,t1_test-y1_test,one_arr])
xy1_test_plus1=np.column_stack([s2_test-x2_test,t2_test-y2_test,one_arr])
plt.scatter(s1_test-x1_test,t1_test-y1_test)
plt.scatter(s2_test-x2_test,t2_test-y2_test)
plt.show()
```



1.3 3. Plot the learning curves

- Apply the gradient descent algorithm
- Plot the training loss at every iteration
- Plot the testing loss at every iteration
- Plot the training accuracy at every iteration
- Plot the testing accuracy at every iteration

In [627]:
```
label0=0
label1=1

def sigmoid(z):
    return 1/(1+np.exp(-z))


def f_z(x,y):
    z=ux+vy+b
    return z


def h_func(xy,u,v,b):
    h=np.zeros(100)

    for i in range(len(xy)):
        z=u*xy[i][0]+v*xy[i][1]+b
        h[i]=sigmoid(z)
```

```
        return h


    def cross_entropy(h,label):

        if(h==0 ):
            ln=10000.0
            f= -(label*ln+(1-label)*(np.log(1-h)))

        elif(h==1):
            ln=10000.0
            f= -(label*np.log(h)+(1-label)*ln)

        else:
            f= -(label*np.log(h)+(1-label)*(np.log(1-h)))

        return f


    def cross_entropy_total(h_label0,h_label1,label0,label1):
        total=0
        for i in range(len(h_label0)):
            total+=cross_entropy(h_label0[i],label0)
        for i in range(len(h_label1)):
            total+=cross_entropy(h_label1[i],label1)

        return total/200




    def partial_differential_of_total_cross_entropy(label0,label1,h_label0,h_label1,xy0,
        L=0

        for i in range(100):
            L+=(-(label0*(1-h_label0[i])*xy0[i]-(1-label0)*(h_label0[i])*xy0[i]))
        for i in range(100):
            L+=(-(label1*(1-h_label1[i])*xy1[i]-(1-label0)*(h_label1[i])*xy1[i]))

        L=L/200

        return L
```

- Function of accuracy comparing to real labels.

```
In [628]: def accuracy_algorithms(h_label0,h_label1):
              label_h0=np.zeros(100)
```

```
        label_h1=np.zeros(100)
        right=0

        for i in range(100):
            if(h_label0[i]>=0.5):
                label_h0[i]=1
            else:
                label_h0[i]=0

            if(h_label1[i]>=0.5):
                label_h1[i]=1
            else:
                label_h1[i]=0

        for i in range(100):
            if(label_h0[i]==0):
                right+=1
            if(label_h1[i]==1):
                right+=1

        accuracy=right/180
        print(accuracy)
        print(h_label1)
        return accuracy
```

- Gradient Descent Algorithms!

```
In [640]: def gradient_descent_algorithms(label0,label1,h_label0,h_label1,xy0,xy1):
        learning_rate = 0.03
        u=0
        v=0
        b=0
        i=0
        total_new=0
        total=cross_entropy_total(h_label0,h_label1,label0,label1)
        iteration=0
        total_loss=np.zeros(100000)
        accuracy_array = np.zeros(100000)
        accuracy_array[0]=accuracy_algorithms(h_label0,h_label1)

        while abs(total_new-total)>0.000001:
            total=total_new
            u = u - learning_rate* partial_differential_of_total_cross_entropy(label0,lal
            v = v - learning_rate* partial_differential_of_total_cross_entropy(label0,lal
            b = b - learning_rate* partial_differential_of_total_cross_entropy(label0,lal
            h_label0=h_func(xy0_train,u,v,b)
            h_label1=h_func(xy1_train,v,v,b)
#            print(h_label1)
```

```
                total_new=cross_entropy_total(h_label0,h_label1,label0,label1)
                total_loss[i]=total_new
                iteration+=1
                i+=1
                accuracy_array[i+1]=accuracy_algorithms(h_label0,h_label1)


            return total_loss,iteration,accuracy_array
```

- Training loss and accuracy.

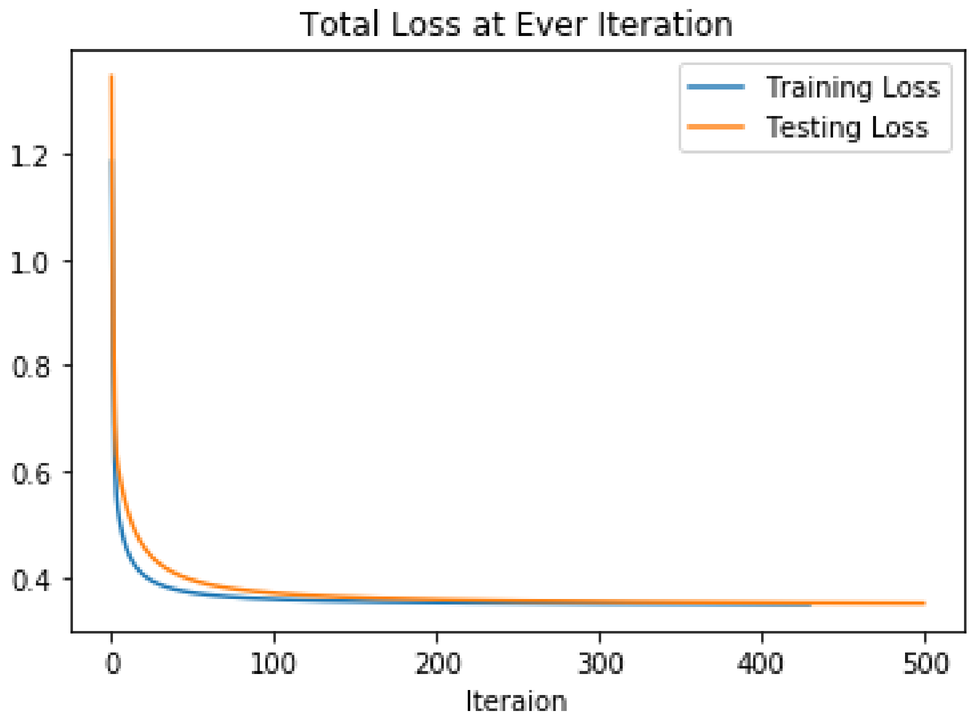
```
In [641]: h_label0_training=h_func(xy0_train,10,10,10)
          h_label1_training=h_func(xy1_train,10,10,10)
          training_total_loss,training_iteration,accuracy_array_train= gradient_descent_algori
                                                                    h_label1_trainin
```

- Testing loss and accuracy.

```
In [642]: h_label0_testing=h_func(xy0_test,10,10,10)
          h_label1_testing=h_func(xy1_test,10,10,10)
          testing_total_loss,testing_iteration,accuracy_array_test= gradient_descent_algorithm
                                                                  h_label1_testing,x
```

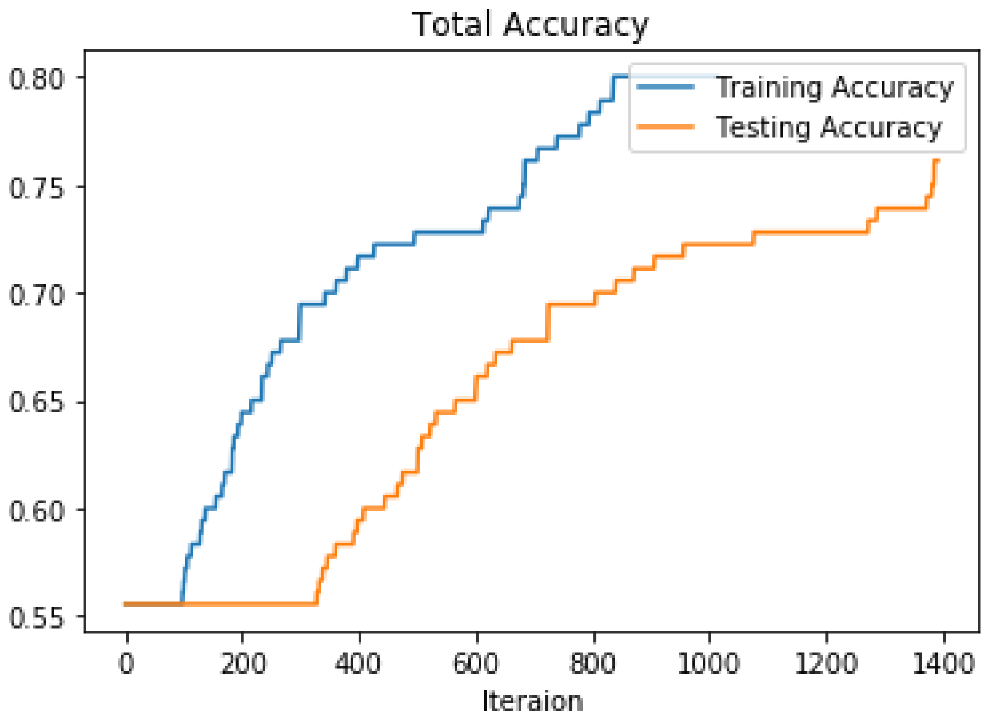
- Visualize each loss!

```
In [648]: plt.plot(training_total_loss[:430],label='Training Loss')
          plt.plot(testing_total_loss[:500],label='Testing Loss')
          plt.legend(loc='upper right')
          plt.title("Total Loss at Ever Iteration")
          plt.xlabel("Iteraion")

Out[648]: Text(0.5, 0, 'Iteraion')
```



- Visualize each Accuracy!

```
In [652]: plt.plot(accuracy_array_train[100:1115],label='Training Accuracy')
          plt.plot(accuracy_array_test[100:1492],label='Testing Accuracy')
          plt.legend(loc='upper right')
          plt.title("Total Accuracy")
          plt.xlabel("Iteraion")

Out[652]: Text(0.5, 0, 'Iteraion')
```


Total Accuracy
Training Accuracy
Testing Accuracy
0.80
0.75
0.70
0.65
0.60
0.55
0
200
400
600
800
1000
1200
1400
Iteraion