شناسایی اماری الگو

تمرین 1

بهاره غلامی

40033626

مهسا ملایم

40032718

1401/9/13

قسمت اول :

در این قســمت باید رگرســیون خطی را با اســتفاده از روش close form solution و gradient descent به دست بیاوریم و سپس خطای مدل که به دست اوردیم را با استفاده از mean square error محاسبه کنیم .

: Closed Form Linear Regression

در رگرسیون خطی ما به دنبال خطی هستیم به فرم زیر :

$$h_\theta(x) = \theta_0 + \theta_1 x$$

که تتاها در فرمول بالا نماینگر پارامترهای رگرسیون هستند و در صورتی که ورودی های ما x باشد و به سایز ورودی را با n نمایش دهیم و yنماینگر متغییروابسته ما باشد تتا با استفاده از فرمول زیر به دست می اید :

Normal equation

$$\Theta = (X^T X)^{-1} X^T y$$

پیاده سازی :

برای خواندن دیتا از دستور زیر استفاده کرده ایم :

load dataset

```
In [17]: df = pd.read_csv('data.csv')
```

و با استفاده از دستور زیر دیتاهای اموزش و تست را از هم جدا کردیم که هفتاد در صد دیتا ها را اموزش و سی درصد را برای تست در نظر گرفتیم .

split_data

```
In [18]: train_x, test_x, train_y, test_y = ms.train_test_split(df['x'], df['y'], train_size=0.7)
```

نرمال سازی دیتا :

از روش minmax استفاده کردیم با استفاده از تابع زیر متغیر مستقل را به بازه 0تا1 میبریم .

```
]: def normalize(x):
       x = (x - x.min() ) / (x.max() - x.min())
       return x
```

:addbias

برای اضاف کردن ستون x0 به دیتای اموزش که برای محاسبه بایاس ضروری است از تابع زیر استفاده می کنیم :

```
]: def addbias(X):
       matrix = X.reshape(X.shape[0], 1)
       ones = np.ones((matrix.shape[0],1))
       concat=np.concatenate((ones,matrix), axis=1)
       return concat
```

خروجی این تابع به این صورت هست که یک ردیف یک به x_train یا x_test اضاف میکند .

: Leastsquared

از تابع زیر برای پیاده سازی فرمول بالا و به دست اوردن پارامتر های تتا استفاده میکنیم ورودی تابع x-train و y-train است و خروجی تابع مجموعه تتا :

```python
def leastsquared(x,y):
    y=y.reshape(y.shape[0],1)
    thetaset = inv(x.T.dot(x)).dot(x.T).dot(y)
    return thetaset
```

پیش بینی :

برای محاسبه پیش بینی دیتا های اموزش و تست و به دست اوردن y_hat از دستورات زیر استفاده کردیم

```python
yHat_trn = leastSquared_theta[0] + leastSquared_theta[1] * train_x

yHat_tst = leastSquared_theta[0] + leastSquared_theta[1] * test_x
```

Mse :

برای محاسبه mean square error از تابع زیر استفاده میکنیم :

```python
def Mse(y,y_hat):
    diff=y-y_hat
    mse_pow=np.power(diff, 2,dtype='float64')
    mse = np.mean(mse_pow)
    return mse
```

نتایج :

```python
print('MSE Train: ', mseTrain)
print('MSE Test: ', mseTest)
```

```
MSE Train:  17.019203409440927
MSE Test:  13.990372372323291
```

پارامترهای به دست امده :

```
print('theta0: ',leastSquared_theta[0])
print('theta1: ',leastSquared_theta[1])
```

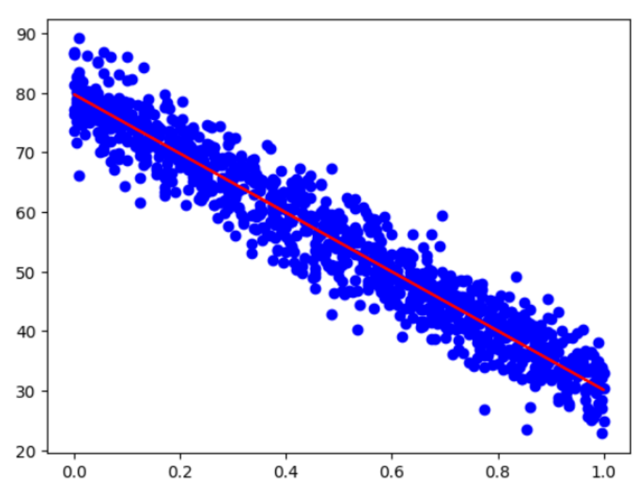
```
theta0:  [79.75387007]
theta1:  [-49.62369159]
```

فرمول خط :

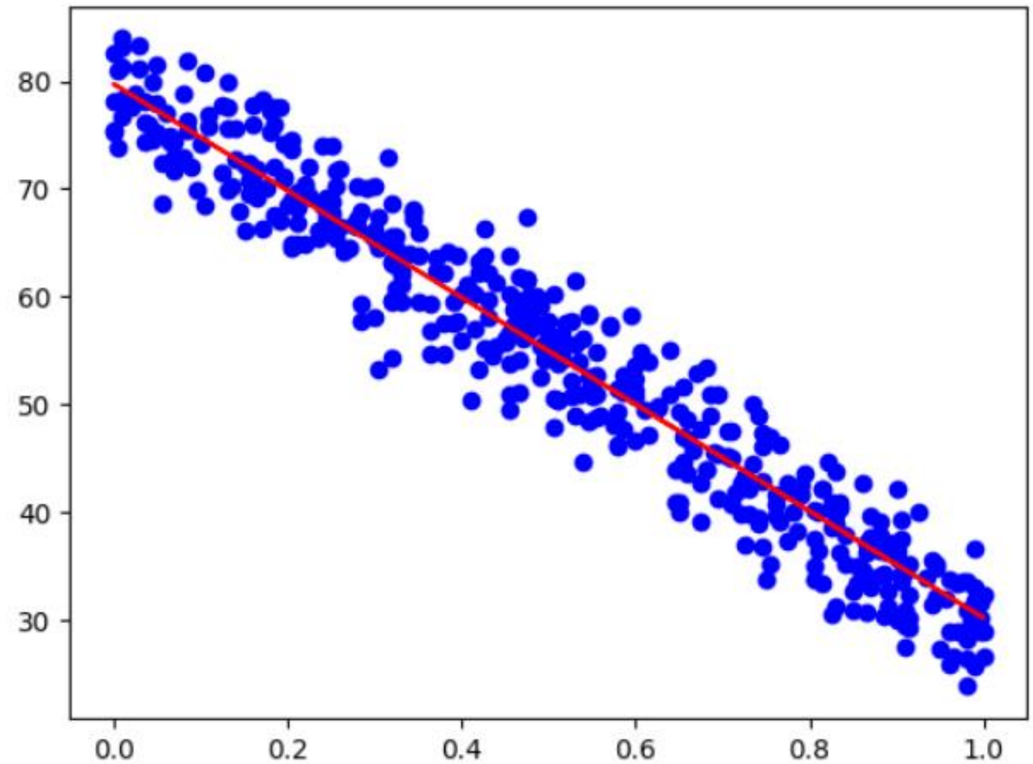
```
print('decision boundary formula:')
print( 'y ='+str(leastSquared_theta[0]) + ' + ' + str(leastSquared_theta[1]) + '*x1')
```

```
decision boundary formula:
y =[79.75387007] + [-49.62369159]*x1
```

نمودار دیتای اموزش :



نمودار دیتای تست :



Gradient Descent Linear Regression

ممکن هست که در بعضی از مسائل رگرسیون غیرخطی راه حل closed form وجود نداشته باشد یا حتی برای رگرسیون خطی هم در بعضی مواقع راه حل closed form

مناسب نیست به دلیل اینکه ممکن ورودی ها خیلی بزرگ باشند و انجام محاسابات برای این راه حل پرهزینه میتواند باشد .

دونمونه gradient descent داریم :

: Stochastic Gradient Descent

این روش برای داده های حجیم مناسب است که آوردن همه آن ها بصورت یکجا در رم و انجام محاسبات منطقی نیست. برای همین در هرepoch ، محاسبات بر روی یک نمونه از داده انجام شده و پارامتر ها آپدیت میشوند.

: Batch Gradient Descent

در این روش در هر epoch از کل دیتا ها بصورت یکجا برای محاسبه پارامتر ها استفاده میشود.

در این قسمت ما از batch برای پیاده سازی استفاده کردیم .

از تابع هزینه mse استفاده کردیم :

$$\text{Cost} = J(w) = \frac{1}{2m} \sum_{i=1}^{m} \left(h(w)^{(i)} - y^{(i)}\right)^2$$

و گرادیان مشتق این تابع بر حسب w است. (w همان تتا است)

$$\text{Gradient} = \frac{\partial J(w)}{\partial w_j} = \frac{1}{m} \sum_{i=1}^{m} \left(h(w)^{(i)} - y^{(i)}\right). X_j^{(i)}$$

برای اپدیت کردن وزن ها از فرمول زیر استفاده میکنیم :

$$\mathrm{w}_j = w_j - lr.\left(\frac{1}{m} \sum_{i=1}^{m} \left(h(w)^{(i)} - y^{(i)}\right). X_j^{(i)}\right)$$

پیاده سازی :

خواندن دیتا :

load dataset

```
In [17]: df = pd.read_csv('data.csv')
```

جداسازی دیتای تست از اموزش :

split_data

```
In [18]: train_x, test_x, train_y, test_y = ms.train_test_split(df['x'], df['y'], train_size=0.7)
```

نرمال کردن دیتا :

```
]: def normalize(x):
       x = (x - x.min() ) / (x.max() - x.min())
       return x
```

: Mse

```
def Mse(y,y_hat):
    diff=y-y_hat
    mse_pow=np.power(diff, 2,dtype='float64')
    mse = np.mean(mse_pow)
    return mse
```

: Addbias

```
def addbias(X):
    matrix = X.reshape(X.shape[0], 1)
    ones = np.ones((matrix.shape[0],1))
    concat=np.concatenate((ones,matrix), axis=1)
    return concat
```

Cost

از این تبع برای محاسبه هزینه استفاده میکنیم :

```
def cost(m,a):
    p=np.power(a,2,dtype='float64')
    err = np.sum(p)/(2*m)
    return err
```

ورودی این تابع a است که برابر

a=(y_hat-y)

و m برابر سایز دیتای اموزش هست .

: gradientDescent

از این تابع استفاده میکنیم برای پیاده سازی فرمول ها

```python
def gradientDescent(x, y):
    alpha=0.01
    m,n=x.shape
    theta = np.random.rand(n).reshape(n, 1)
    errors=[]
    i=0
    for i in range(10000):
        x1=x[:,[1]]
        y_hat = theta[1]*x1 +theta[0]
        a=(y_hat-y.reshape(y.shape[0],1))
        error= cost(m,a)
        errors.append(error)
        #a=(y_hat-y.reshape(y.shape[0],1))
        gra_theta1= np.sum(x1*a)/m
        gra_theta0 = np.sum(a)/m
        theta[1] = theta[1] - alpha* gra_theta1
        theta[0] = theta[0] - alpha*gra_theta0

    errors = np.array(errors)
    return theta,errors
```

مقدار lr را 0.01 و تعداد ایپاک را 10000 در نظر گرفتیم .

اموزش :

```python
theta,errors = gradientDescent( x_trnbias,train_y)
```

برای پیش بینی از کد زیر استفاده :

```python
y_hat_Train =theta[0] + theta[1]*train_x
y_hat_Test = theta[0] +theta[1]*test_x
```

نتایج :

پارامترهای به دست امده :

```
# Print Theta
print('theta0: ',theta[0])
print('theta1: ',theta[1])
```

```
theta0:  [79.73857454]
theta1:  [-49.54207482]
```

خط به دست امده :

```
print('decision boundary formula:')
print( 'y ='+str(theta[0]) + ' + ' + str(theta[1]) + '*x1')
```

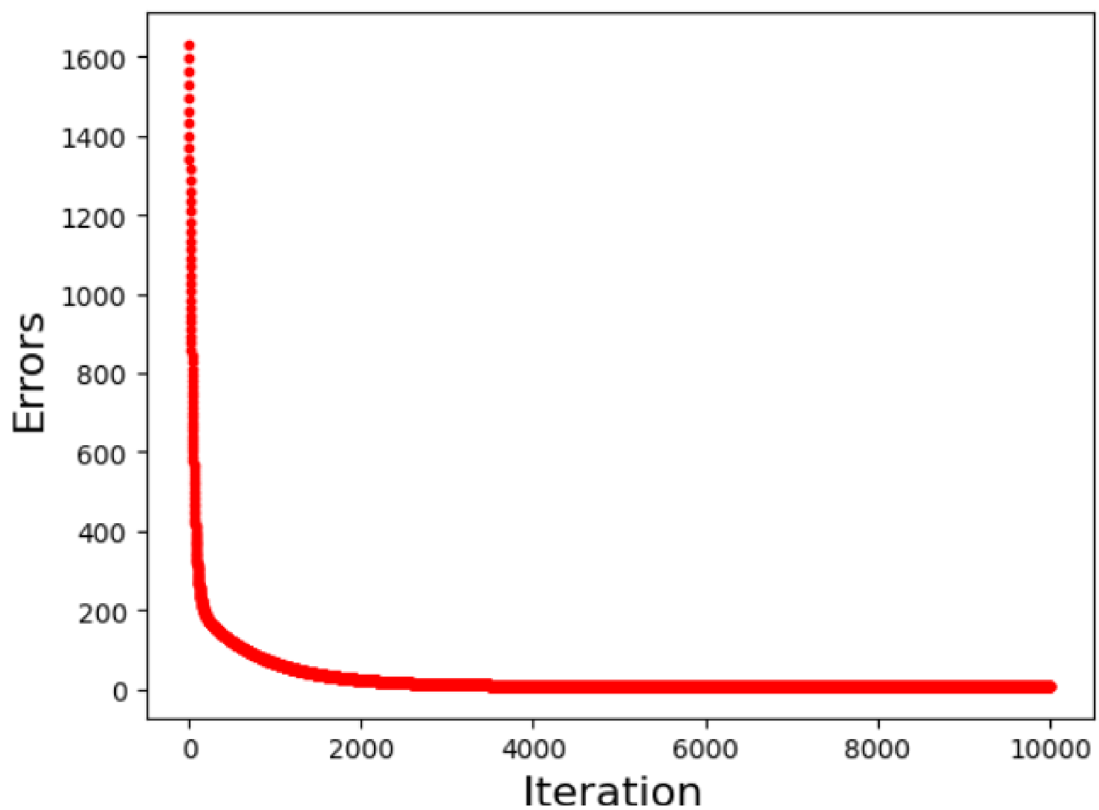
```
decision boundary formula:
y =[79.73857454] + [-49.54207482]*x1
```

Mse :

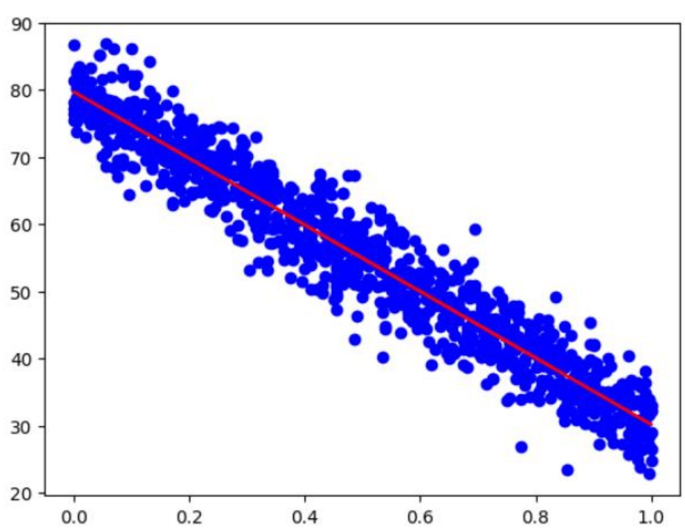
```
mseTrain = Mse(train_y,y_hat_Train)
mseTest = Mse(test_y,y_hat_Test)
print('MSE Train : ', mseTrain)
print('MSE Test : ', mseTest)
```

```
MSE Train :  16.028842593548845
MSE Test :  16.32632733058033
```

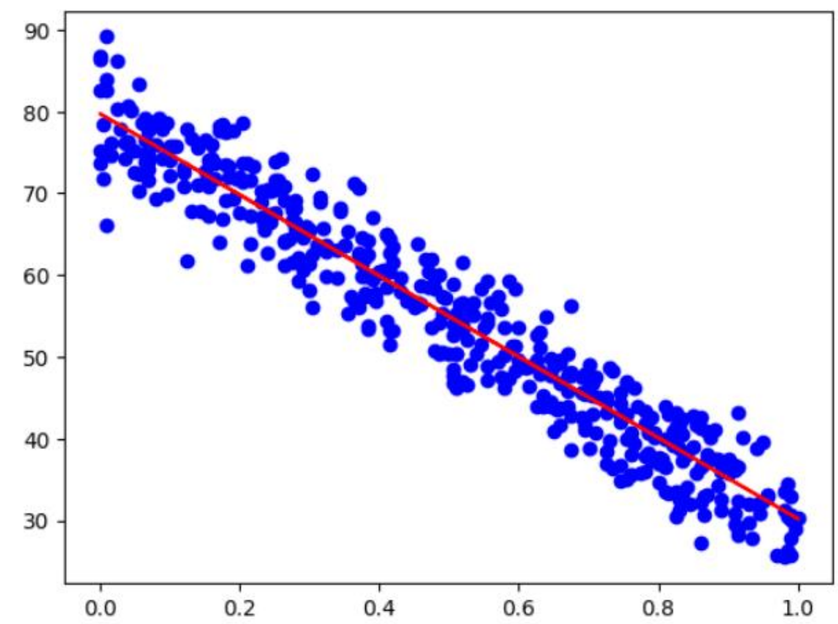
نمودار هزینه :



نمودار دیتای اموزش :



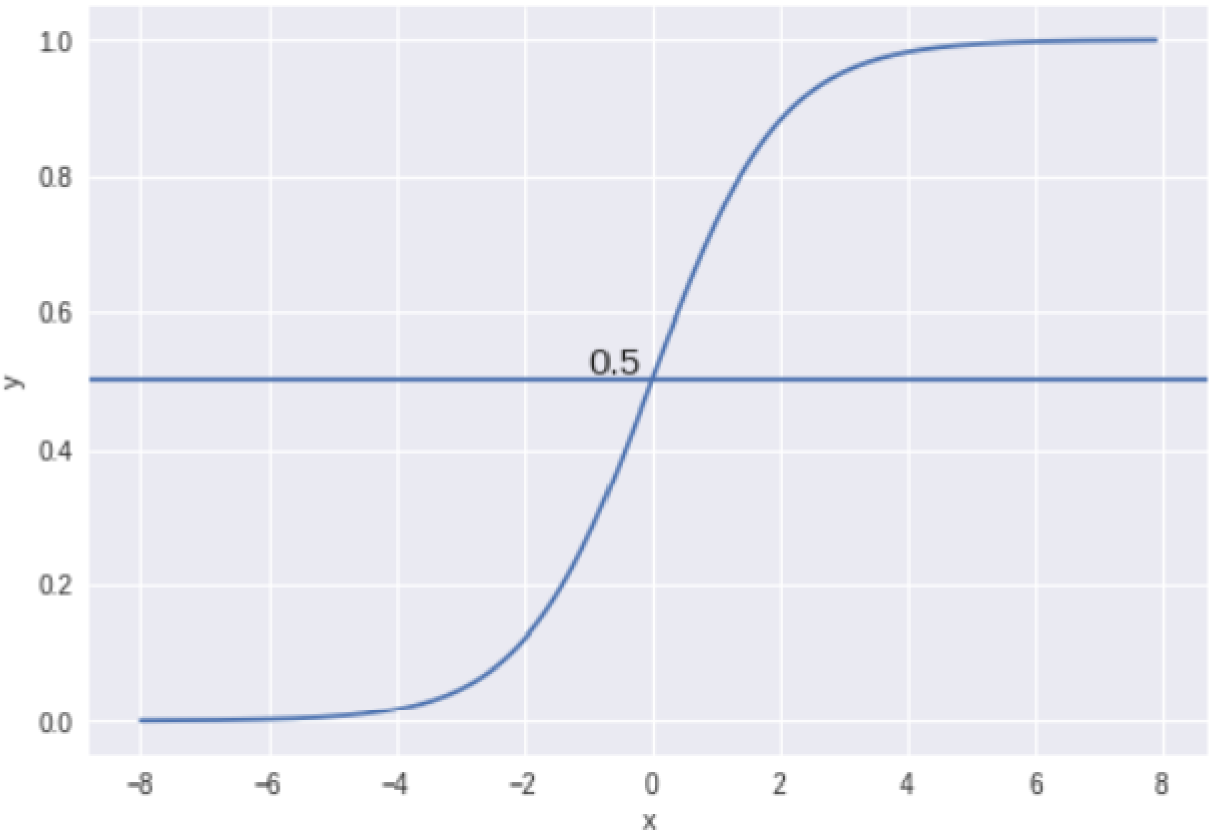
نمودار دیتای تست :



قسمت دوم :

Logistic Regression

از تابع سیگموید در این روش استفاده میکنیم که نمودار ان به شکل زیر است :



این تابع همیشه مقداری بین صفر و یک دارد و از این تابع استفاده میکنیم در صورتی که مقدار احتمالی که به دست می اوریم بیشتراز 0.5 بود مربوط به کلاس یک و در صورتی که کوچتر از 0.5 بود مربوط به کلاس 0 هست .

فرمول تابع سیگموید به صورت زیر است :

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}} \qquad 0 \le h_\theta(x) \le 1$$

مانند رگرسیون خطی باید یک تابع هزینه مشخص کنیم و ان را کمینه کنیم از تابع زیر استفاده میکنیم :

$$J(\theta)=\frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\log(h_\theta(x^{(i)}))-(1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right]$$

برای کمینه کردن معادله از gradient descent استفاده میکنیم .

$$\begin{aligned}&\textbf{repeat until convergence } \{\\&\qquad \boxed{\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta)} \quad j=0,1,\dots,n\\&\}\end{aligned}$$

$$\frac{\partial}{\partial\theta_j}J(\theta)=\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)}).x_j^{(i)}$$

پیاده سازی :

load_data :

از این تابع استفاده میکنیم و تنها فیچر ردیف سوم و هفتم را جدا کرده و دیتاهایی که مربوط به کلاس سه هست را حذف میکنیم و سپس دیتای اموزش و تست را از هم جدا میکنیم :

```
def load_data():
    df=pd.read_excel("seed.xlsx")
    df=df[['x3','x7','class']]
    cdf=df[df['class']<3]
    cdf_1=df[df['class'] == 1]
    cdf_2=df[df['class'] == 2]
    cdf_2=cdf_2.replace(to_replace = 2, value =0)
    g=cdf_1.append(cdf_2, ignore_index=True)
    train_x, test_x, train_y, test_y = ms.train_test_split(g[['x3','x7']],g['class'], train_size=0.7)
    return train_x, test_x, train_y, test_y
```

: Normalize

```
]: def normalize(x):
       x = (x - x.min() ) / (x.max() - x.min())
       return x
```

: Mse

```
def Mse(y,y_hat):
    diff=y-y_hat
    mse_pow=np.power(diff, 2,dtype='float64')
    mse = np.mean(mse_pow)
    return mse
```

: Sigmoid

```
def sigmoid(x,theta,theta0):
    temp=np.dot(x,theta )+theta0
    y = 1 / (1 + np.exp(-temp))
    return y
```

ورودی این تابع متغیر های مستقل اموزش و مجموعه تتا هست .

: Cost

```
1]: def cost(y_hat,y,m):
        d=-1/m * np.sum(y * np.log(y_hat) + (1 - y) * np.log(1-y_hat))
        return d
```

ورودی تابع مقدار y و y_hat و سایز دیتاهای اموزش است و مقدار هزینه را مشخص میکند در هر تکرار .

: LogisticRegression

```
def LogisticRegression(x, y,alpha,itr):
    m, n = x.shape
    theta=np.random.rand(n)
    theta0=0.001
    errors= []
    for i in range(itr):
        y_hat = sigmoid(x,theta,theta0)
        error= cost(y_hat,y,m)
        errors.append(error)
        theta= theta - alpha*(1/m * np.dot(x.T, (y_hat - y)))
        theta0=theta0 - alpha* (1/m * np.sum(y_hat - y))
    errors = np.array(errors)
    return theta,theta0,errors
```

بر اساس فرمول که توضیح دادیم مدل اموزش میبیند .

اموزش مدل :

```
: theta,theta0,errors = LogisticRegression(train_x,train_y, 0.05,70000)
```

برای اموزش مدل تعداد ایپاک را 70000و الفا را 0.05گرفتیم .

: decision_boundary

برای نمایش خط جداکنند دو کلاس از این تابع استفاده میکنیم .

```python
def decision_boundary(x,theta,theta0):
    return - (theta0 + np.dot(theta[0], x)) / theta[1]
```

```python
print('Decision boundary formula:')
print('-(' + str(theta0) + ' + ' + str(
    theta[0]) + '*x1) / (' + str(theta[1]) + '*x2)')
```

```
Decision boundary formula:
-(22.55000292540777 + 0.10106886184755208*x1) / (-27.298907690551186*x2)
```

نتایج :

پارامترهای به دست امده :

```python
print('theta0: ',theta0)
print('theta1: ',theta[0])
print('theta2: ',theta[1])
```

```
theta0:  22.55000292540777
theta1:  0.10106886184755208
theta2:  -27.298907690551186
```

: Mse

```python
print("MSE Train = ", Mse(train_y,sigmoid(train_x,theta,theta0)))
print("MSE Test  = ",Mse( test_y,sigmoid(test_x,theta,theta0)))
```

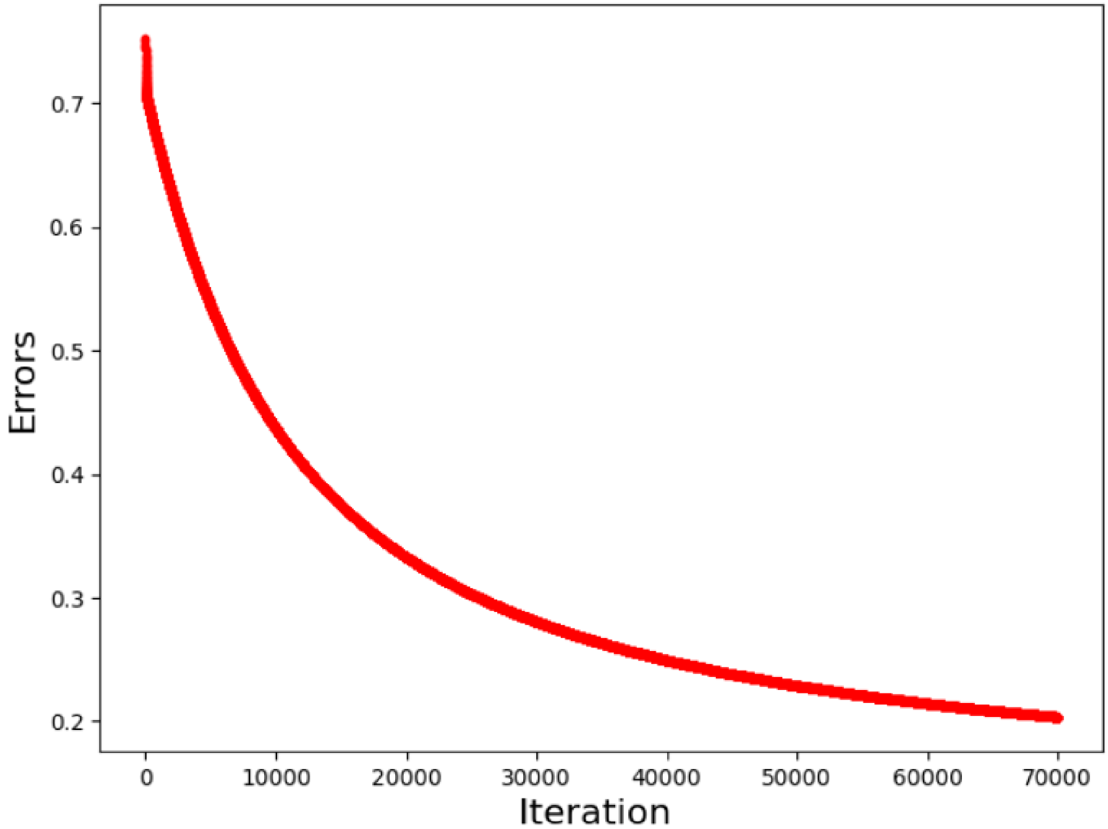
```
MSE Train =  0.04957891020411598
MSE Test  =  0.022715285122802936
```

فرمول خط :

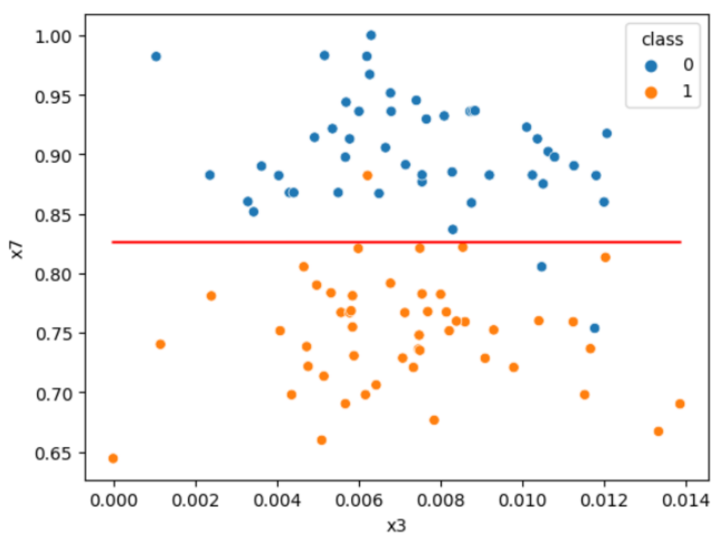
```python
print('decision boundary formula:')
print( 'y ='+str(theta0) + ' + ' + str(theta[0]) + '*x1' +  ' + ' +str(theta[1]) + '*x2')
```

```
decision boundary formula:
y =22.55000292540777 + 0.10106886184755208*x1 + -27.298907690551186*x2
```

نمودار هزینه :

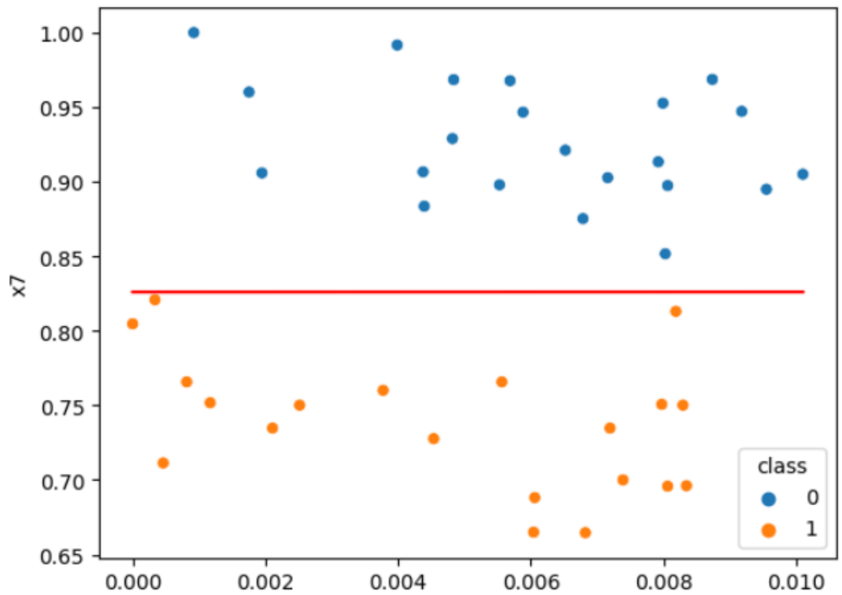


نمودار دیتای اموزش :




```
[935]: x_values = pd.DataFrame(data=test_x, columns=["x3", "x7"])
```

نمودار دیتای تست :



دقت دیتای اموزش و تست :

که از فرمول گفته شده سر کلاس استفاده کردیم .

```
y_hat_train =sigmoid(train_x,theta,theta0)
y_hat_train =[0 if y<=0.5 else 1 for y in y_hat_train ]
accuracy_train = np.sum(train_y == y_hat_train) / (train_y.shape[0])
print('Train accuracy:', accuracy_train)
```

```
Train accuracy: 0.9693877551020408
```

```
y_hat_test =sigmoid(test_x,theta,theta0)
y_hat_test =[0 if y<=0.5 else 1 for y in y_hat_test]
accuracy_test = np.sum(test_y == y_hat_test) / (test_y.shape[0])
print('Test accuracy:', accuracy_test)
```

```
Test accuracy: 1.0
```