HW1

Shirin Mohebbi, 9935597

Razie masoudi, 9935598

Pattern Recognition

December 8, 2020

Gradient Descent algorithm :

first we should add x0 = 1 to all our X samples and normalize the rest of features in X in order to, in each feature, mean become 0 and std become 1

so X is m * 2 matrix (m is number of samples)

each row is in the form of : [x0 = 1, x1(normalized feature)]

in order to do that i did this:

```
#normalization
x1s = [i[1] for i in X]
std = np.std(np.array(x1s))
mean = np.mean(np.array(x1s))

for i in range(len(X)):
  X[i][1] = (X[i][1] - mean) / (std)
```

Then

I used batch gradient descent for getting more accurate estimation and samples was not that much many so i chose batch.

for implementing it, i used matrix formula.

$$\theta_i = \theta_i - \alpha \sum_{j=1}^{m} (h_\theta(\mathbf{X}^{(j)}) - y^{(j)}) x_i^{(j)}$$

here is my function

X: features

Y: y

Alpha: alpha in the formula

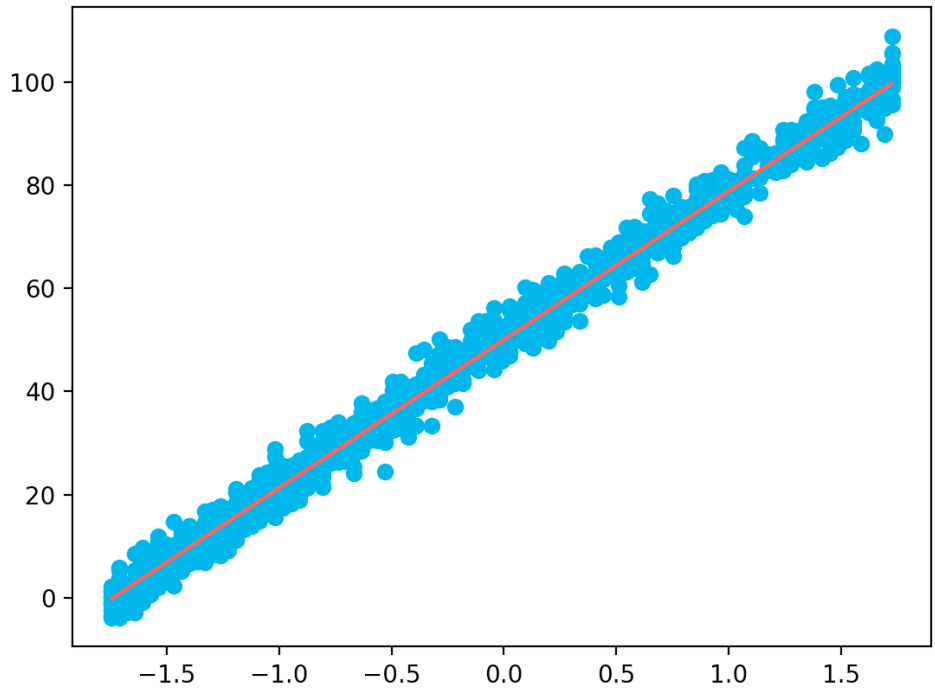
Iterations: max number of iterations, i picked 10000

Epsilon: if second norm of difference between two thetas, be less than epsilon, iteration will stop, and we found thetas

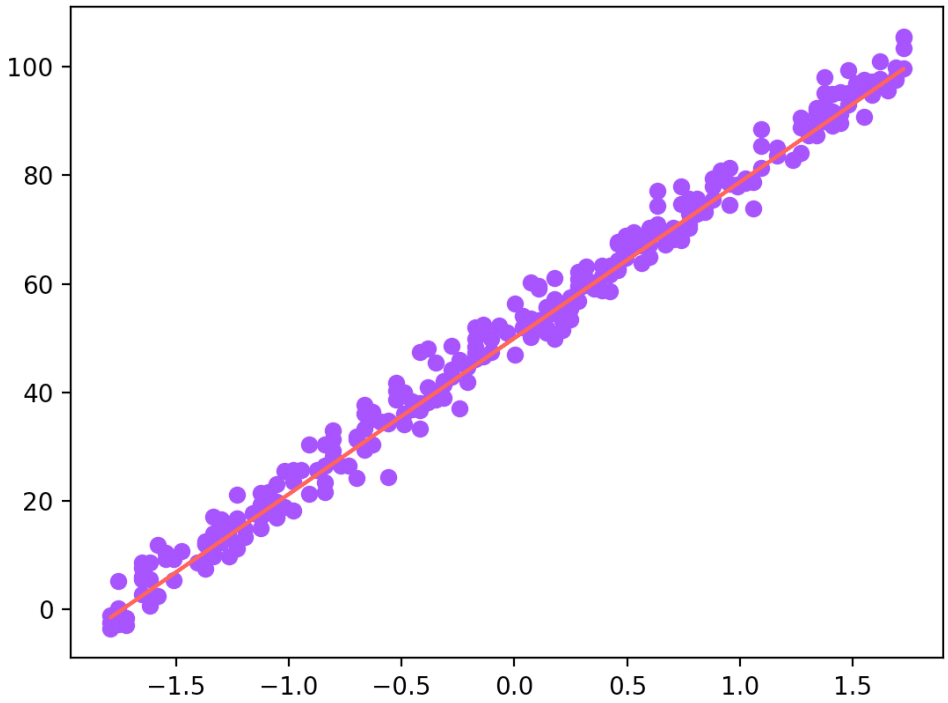
```
def gradient_decent(X, y, teta, alpha, iterations, epsilon):

  alpha = alpha / len(y)
  preTeta = teta
  for i in range(iterations):
    teta = teta - ((alpha) * (X.T @ ( (X @ teta) - y )))
    diffTeta = teta - preTeta
    normDiffTeta = np.linalg.norm(diffTeta)
    if normDiffTeta < epsilon:
      return teta
    preTeta = teta
```

after 170 iterations, thetas found:

theta0: 50.024515 - theta1: 28.770650



In this plot, blue points are train data, and red line is our linear re-gression model with thetas we found by gradient descent



In this plot, purple points are test data, and red line is our linear regression model with thetas we found by gradient descent

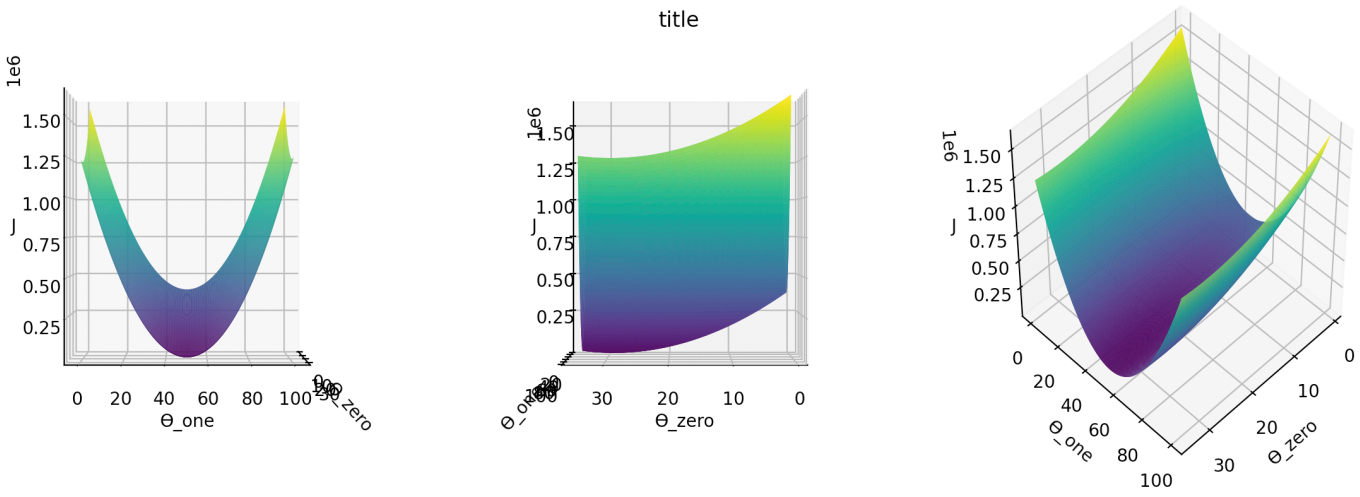
And this is the cost function:

$$\frac{1}{2}(X\theta - \mathbf{y})^T \cdot (X\theta - \mathbf{y}) = \frac{1}{2}\sum_{i=1}^{m}(h(\mathbf{X}^{(j)}) - y^{(j)})^2 = J(\theta)$$

Which implement in the matrix formula:

```
def j_function(t0, t1, X, Y):
  zValue = []
  for i in range(len(t0)):
    zRow = []
    for j in range(len(t0[i])):
      teta = np.array([[t0[i][j]], [t1[i][j]]])
      error = (X @ teta) - Y
      z = (0.5 * ((error.T) @ error))[0][0]
      zRow += [z]
    zValue += [zRow]

  return zValue
```

This is cost function plot, which shows our minimum.



And finally we calculate MSE to see how well we work

```
def calMSE(X, Y, teta):
  error = (X @ teta) - Y
  return((((error.T) @ error)[0][0]) ** 0.5) / len(Y)
```

And this is the result:

```
min squere error data train:  0.1001093003203769
min squere error data test:  0.19640155987301425
```

Logistic Regression(Binary classification):

Tt is pretty much like above algorithm:

First we add x0 = 1 to all our X samples and normalize rest of the features in X in order to, in each feature, mean become 0 and std become 1. and then we set features to be between 0 and 1 by doing below operations

```
#normalization
x1s = [i[1] for i in dataSet]
x2s = [i[2] for i in dataSet]
std1 = np.std(np.array(x1s))
mean1 = np.mean(np.array(x1s))
std2 = np.std(np.array(x2s))
mean2 = np.mean(np.array(x2s))

for i in range(len(dataSet)):
    dataSet[i][1] = (dataSet[i][1] - mean1) / (std1)
    dataSet[i][2] = (dataSet[i][2] - mean2) / (std2)

maxf1 = max(dataSet, key=lambda x: x[1])[1]
minf1 = min(dataSet, key=lambda x: x[1])[1]
maxf2 = max(dataSet, key=lambda x: x[2])[2]
minf2 = min(dataSet, key=lambda x: x[2])[2]

for i in range(len(dataSet)):
    dataSet[i][1] = (dataSet[i][1] - minf1) / (maxf1 - minf1)
    dataSet[i][2] = (dataSet[i][2] - minf2) / (maxf2 - minf2)
```

And for y, because we have 2 class, we label them as 0 and 1

```
if (content[i][4] == 'Iris-setosa'):
    y.append([0.0])
else:
    y.append([1.0])
```

We use gradient ascent based on this formula, implemented by matrix form

$$\theta_i = \theta_i + \alpha \sum_{i=1}^{m} (y^{(j)} - h_\theta(\mathbf{X}^{(j)}))x_i^{(j)}$$

$$h_\theta(\mathbf{X}) = g(\theta^T \mathbf{X}) = \frac{1}{1 + e^{-\theta^T \mathbf{X}}}$$

Like this:

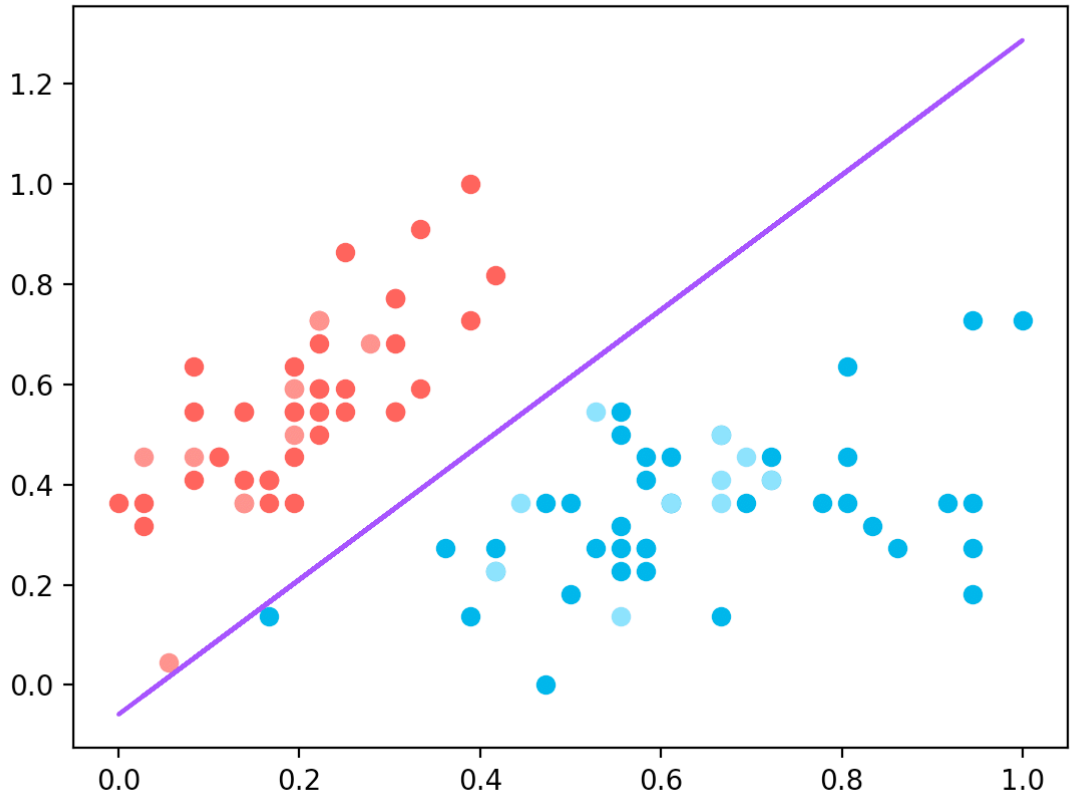
```
def gradient_ascent(X, y, teta, alpha, iterations, epsilon):

    preTeta = teta
    for _ in range(iterations):
        teta = teta + ((alpha) * (X.T @ ( y - sigmoid(X @ teta) )))
        diffTeta = teta - preTeta
        normDiffTeta = np.linalg.norm(diffTeta)
        if normDiffTeta < epsilon:
            return teta
        preTeta = teta
    return teta
```

parameters are the same as gradient descent we had in linear regression,

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

the only difference here is H(theta) which is sigmoid(between 1 and 0) function.

Results:



In plot above, red dots are from class with y label 1 and blue dots are from class with label 0

lighter red dots are train data and lighter blue dots are test data

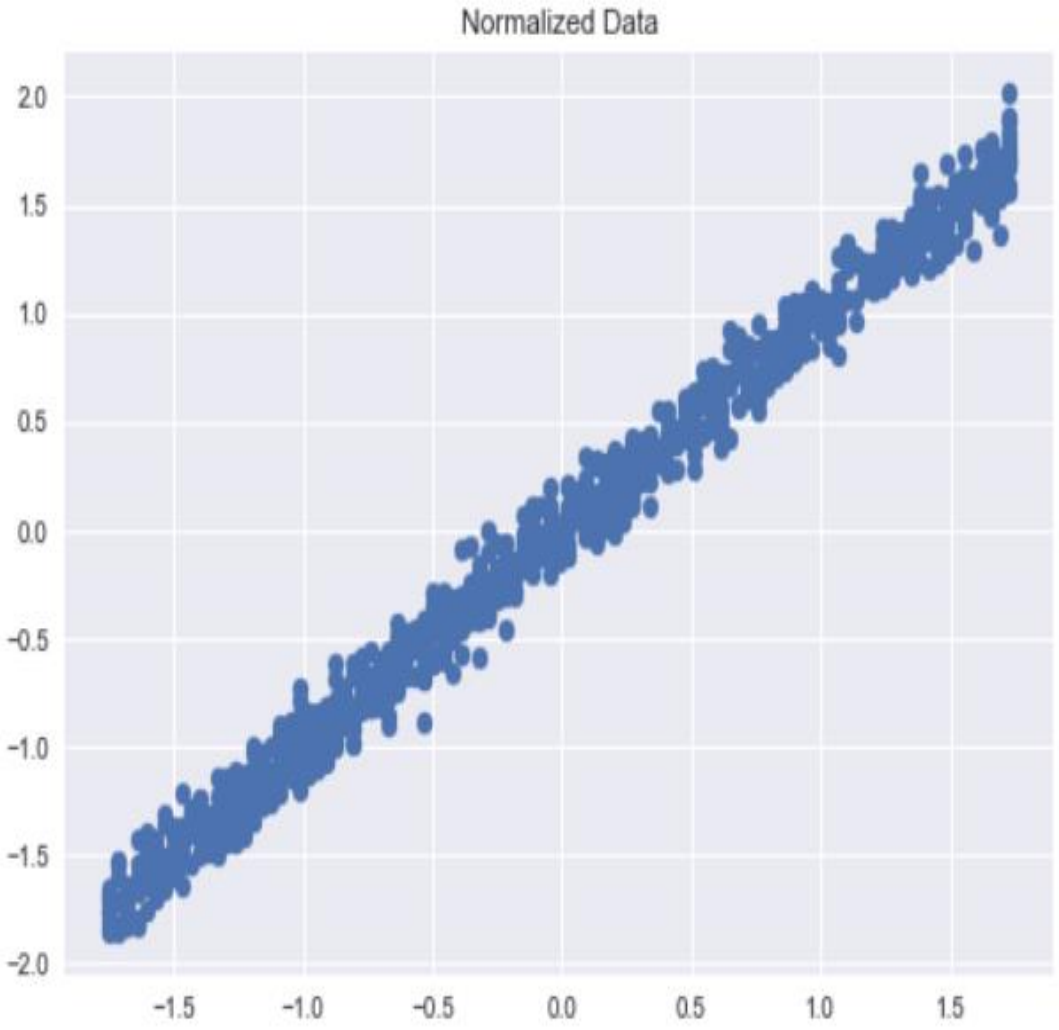
and the purple line is our decision boundary based on thetas

```
teta0: -0.678042
teta1: 15.499087
teta2: -11.517173
accuracy train 100.0
accuracy test 100.0
decision boundary y = 1.345737 * x + -0.058872
```

closed form linear regression

ابتدا کتابخانه های numpy، pandas و matplotlib را اضافه می کنیم سپس از کتابخانه sklearn تابع make_regression را اضافه می کنیم سپس توسط متد read_csv داده ها را درون دیتا فریم dataset می ریزیم. سپس ستون x آن که شامل feature ها است را در سری X می ریزیم و X را تبدیل به آرایه کرده و در X1 می ریزیم این مراحل را برای لیبل ها که درون ستون y از dataset هستند اجرا کرده و آرایه نهایی را درون Y1 می ریزیم. هر دو آرایه ی X1 و Y1 که به شکل (1,1000) هستند را توسط reshape به آرایه هایی با ابعاد (1000,1) تبدیل می کنیم.

در خط 6 مقادیر feature ها و label ها را با تفریق از میانگین آنها و تقسیم مقدار حاصل بر انحراف معیار داده ها نرمال می کنیم؛ سپس نمودار نمونه ها را توسط متد های کتابخانه matplotlib که با کلید واژه plt آن را صدا می زنیم، با عنوان normalized data، رسم می کنیم.



در خط 9 یک آرایه با نام ones با ابعاد (1000,1) که همان x0 در فرمول است توسط دستور hstack به آرایه X1 اضافه می کنیم تا آرایه ی جدیدی به نام new_X با ابعاد (1000,2)

خط 10: در این خط دو تابع با نام های predict و closedform وجود دارد.

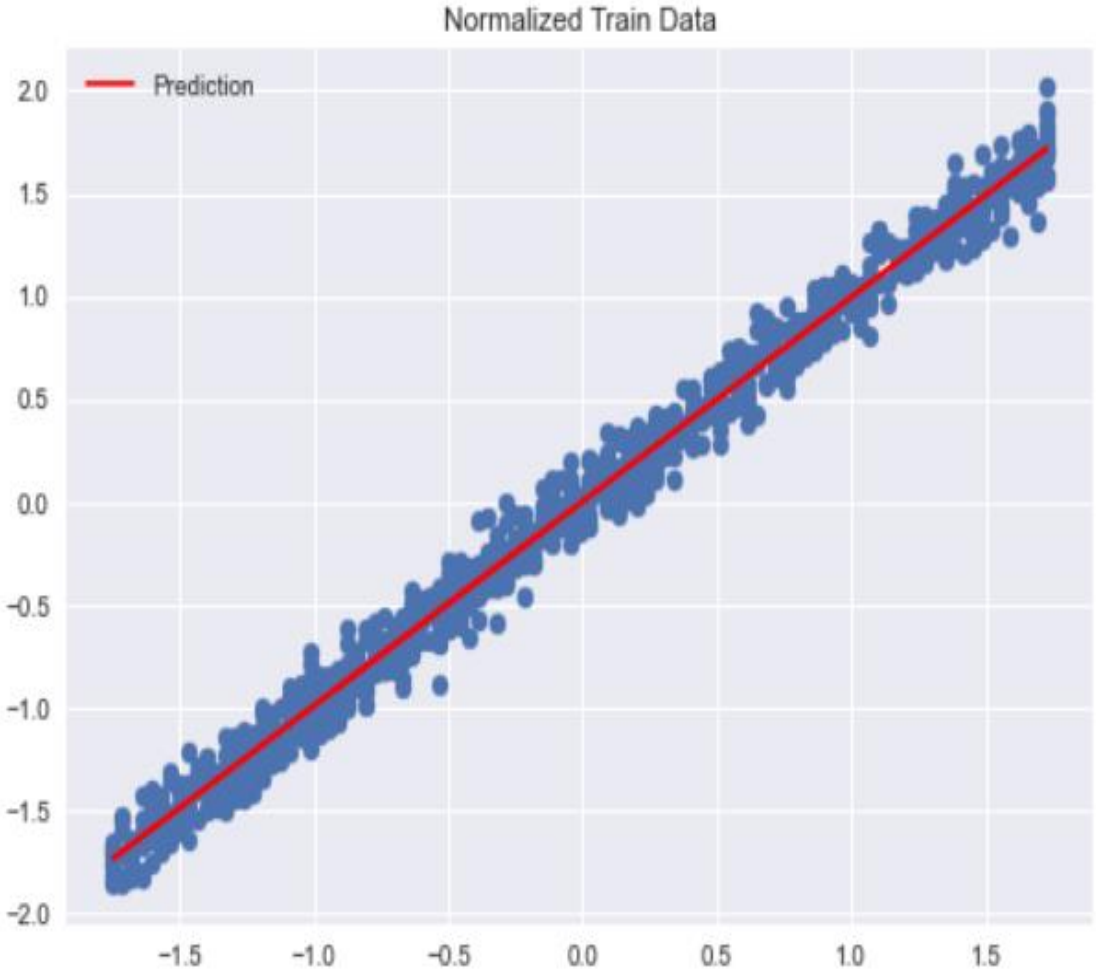
تابع predict: این تابع feature ها و theta های به وجود آمده از تابع closedform را به عنوان ورودی می گیرد و آن ها را با هم dot (ضرب داخلی) می کند که همان مقدار پیش بینی شده توسط مدل است را بر می گرداند.

$$y^{\wedge}(j) = h(X(j)) = h\theta(X(j)) = \theta 0 + \theta 1 x(j)1 + \ldots + \theta n x(j)n = Xn$$
$$i=0\ \theta i\ x(j)i = \theta^{T} X(j)\ j = 1, 2, \ldots, m$$

تابع closedform: این تابع دو آرایه ی X وY را به عنوان ورودی می گیرد سپس طبق فرمول زیر theta ها را بر می گرداند.

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

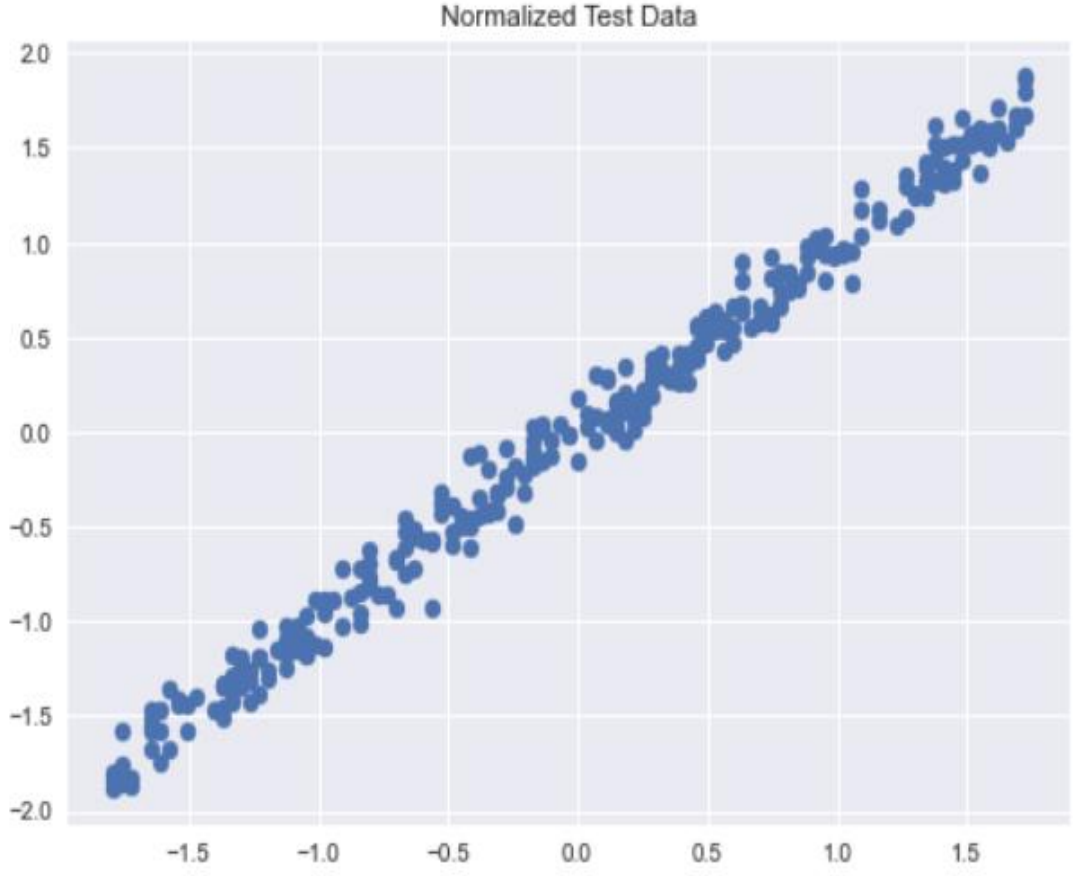
خط 11: یک متغیر به نام theta تعریف می کنیم تا thetaهای بدست آمده از تابع closedform با ورودی های new_X و Y1 را در آن بریزیم. با توجه به theta های بدست آمده تابع predict را صدا می زنیم تا مقادیر پیش بینی شده را در متغیر y_train بریزد؛ سپس در خط 15 نمودار(خط) مدل پیشبینی شده را بر روی مجموعه نمونه ها رسم می کنیم.



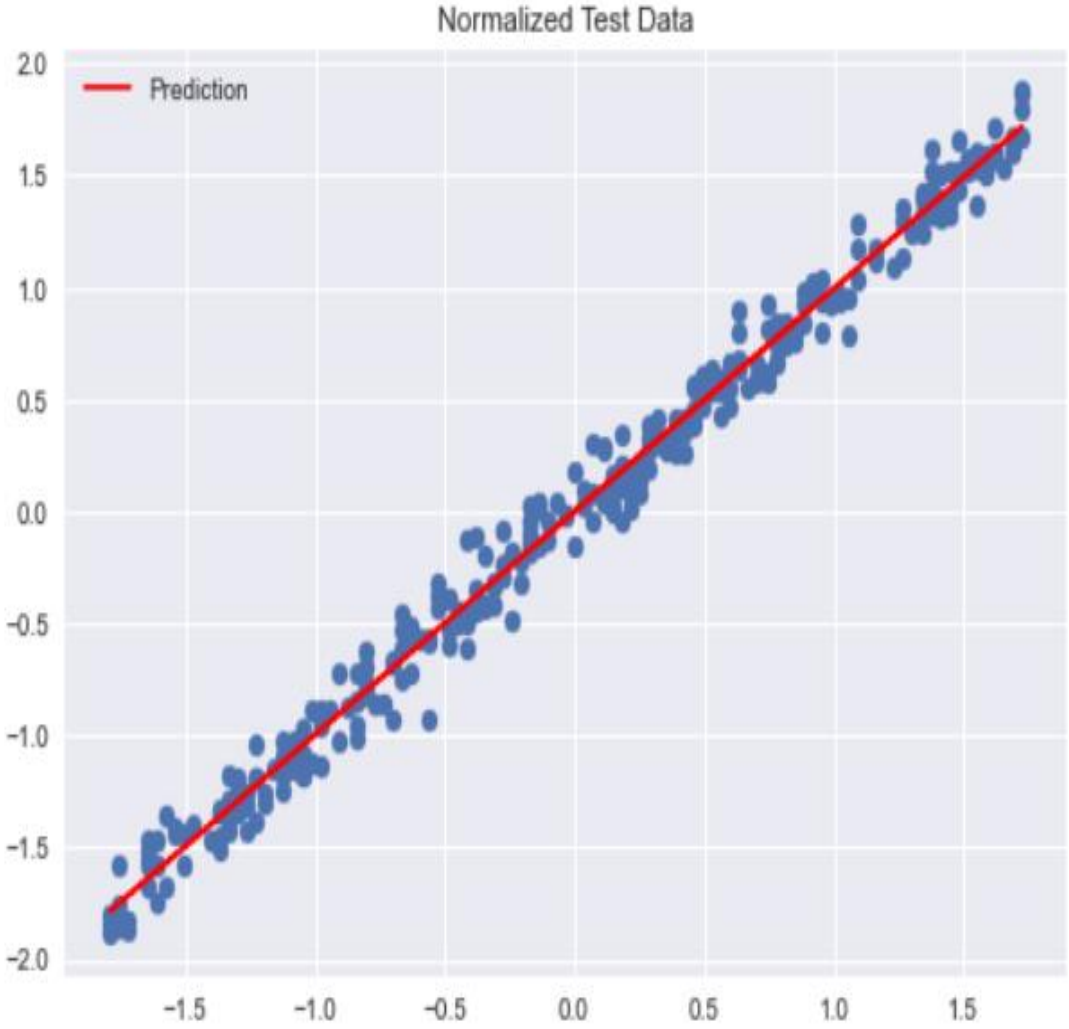
خط 13 و 14 : مقدار خطای آموزشی را طبق فرمول زیر محاسبه کرده و آن را نمایش می دهیم.

$$Error = (y\hat{}\ (j) - y\ (j))^2$$

در نهایت تمام مراحل بالا اعم از loading، normalization و visualization را برای داده های آزمایشی که شامل X_test و Y_test است، طی می کنیم و نمودار data test را تحت عنوان normalized test data نمایش می دهیم.



در خط 19 مقادیر x0 را توسط آرایه ones با ابعاد (300,1) به X_test جهت انجام محاسبات اضافه می کنیم و در آرایه ی Xnew_test می ریزیم، سپس مقدار پیش بینی شده توسط تابع predict با آرگمان های Xnew_test و theta به دست آمده از مراحل قبل، را در y_ می ریزیم تا بتوان در خط 24 نمودار(خط) مدل پیشبینی شده را بر روی مجموعه نمونه های test رسم کرد.



خط 21 و 23 : مقدار خطای آزمایشی را طبق فرمول زیر محاسبه کرده و آن را نمایش می دهیم.

$Error = (\hat{y}(j) - y(j))^2$

$\hat{y}(j) = y\_$

$y(j) = Y\_test$