HOMEWORK#1: LINEAR REGRESSION

Statistical Pattern Recognition

چکیده

گزارش تمرین جلسه ۱ – محاسبه رگرسیون خطی با استفاده از روش های closed form و gradient descend
با استفاده از دیتای Train و Test ، نمایش مقادیر خروجی و رسم نمودارهای مربوطه

افروز راشدی اشرفی / ۴۰۰۶۲۶۹۳ - حسین توکلیان / ۴۰۰۶۲۳۵۳

آبان ۱۴۰۰

هدف: مدلسازی رابطه بین دو متغیر از طریق اعمال یک معادله خطی بر روی مجموعه ای از داده ها

روش اجرا: رگرسیون خطی (با استفاده از closed form و الگوریتم Gradient Descend (Stochastic)

ورودی: دیتاست های Data-Train و Data-Test

خروجی: پارامترهای Theta و مقدار MSE بر روی داده های test و train

زبان برنامه: Python

روش اول) Closed Form Linear Regression

برای کشف مدل رابطه خطی بین متغیرها، می توانیم از رگرسیون استفاده کنیم. در این روش فرض می کنیم متغیر (یا متغیرهایی) که مقدار آنها از سایر متغیرها مستقل است (وابستگی به دیگر متغیرها ندارند)، می توانند برای پیش بینی متغیر وابسته که پاسخ مسئله است استفاده شوند.

هدف، پیدا کردن مدلی است که کمترین خطا را داشته باشد. در این روش برای محاسبه خطا از روش MSE استفاده میکنیم. زمانی MSE کمینه خواهد بود که داده ها توزیع نرمال داشته باشند. بنابراین در ابتدا جهت پیش پردازش، داده ها را نرمال سازی میکنیم.

برای نمایش رابطه خطی بین ۲ متغیر مستقل و وابسته از نمودار نقطه ای (scatter) استفاده میکنیم.

توضیح کد:

در این کد از کتابخانه های pandas، numpy وmatplotlib استفاده شده است.

```
import numpy as np
import pandas as pd
from numpy.linalg import inv
import matplotlib.pyplot as plt
```

۱) تعریف تابع نرمالسازی

```
def normalize(X):
    Min = X.min()
    Max = X.max()
    return (X - Min) / (Max - Min), Min, Max
```

۲) تعریف تابع افزودن bias

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

X_0 عرض از مبدا (bias)، و همان ضریب تتاصفر (θ_0) است که همیشه ۱ قرار دارد. دلیل اصلی این است که اطمینان پیدا کنیم به طور پیش فرض خط از مبدا عبور نکند. بنابراین این تابع، اولین ستون ماتریس X را ۱ می گذارد.

۳) تابع leastSquared (بدست آوردن تتا)

```
def leastSquared(X, y):
    theta = inv(X.T.dot(X)).dot(X.T).dot(y)
    return theta
```

خروجی این تابع (تتا) یک وکتور با ابعاد (n+1)x1 است.

۴) تابع محاسبه خطا (Mean Squared Error)

```
def mseError(y,y_hat):
    mse = np.mean(np.power(y - y_hat, 2))
    return mse
```

این تابع، خطا را محاسبه می کند. $(y - y\hat{})^2$

۵) خواندن دیتا از فایل

داده ها با کمک کتابخانه pandas وارد می شوند.

```
train_data = pd.read_csv('Data-Train.csv')
test_data = pd.read_csv('Data-Test.csv')
```

۶) تفکیک داده های مستقل (X) و وابسته (y)

```
x_trn = train_data['x'].values
y_trn = train_data['y'].values

x_tst = test_data['x'].values
y_tst = test_data['y'].values
```

۷) نرمال سازی داده

```
x_trn, trnMinX, trnMaxX = normalize(x_trn)
x_tst, tstMinX, tstMaxX = normalize(x_tst)
```

داده ها (X) را نرمال میکنیم. نرمال سازی هم بر روی داده Train و هم بر روی داده Test انجام می شود.

۸) افزودن bias

```
x_trnBiasAdded = biasAddition(x_trn)
x_tstBiasAdded = biasAddition(x_tst)
```

ستون اول ماتریس X در داده های Train و Test، ستون ۱ اضافه می شود. (عرض از مبدا)

۹) محاسبه تتا (اجرای Least Squared)

```
leastSquared_theta = leastSquared(x_trnBiasAdded, y_trn)
```

با استفاده از روش Least Squared، تتا را بدست می آوریم.

۱۰) خروجی

```
yHat_trn = leastSquared_theta[0] + leastSquared_theta[1] * x_trn
yHat_tst = leastSquared_theta[0] + leastSquared_theta[1] * x_tst
```

۱۱) محاسبه خطا (MSE)

```
mseTrain = mseError(y_trn,yHat_trn)
mseTest = mseError(y_tst,yHat_tst)
```

۱۲) نمایش پیش بینی داده ها

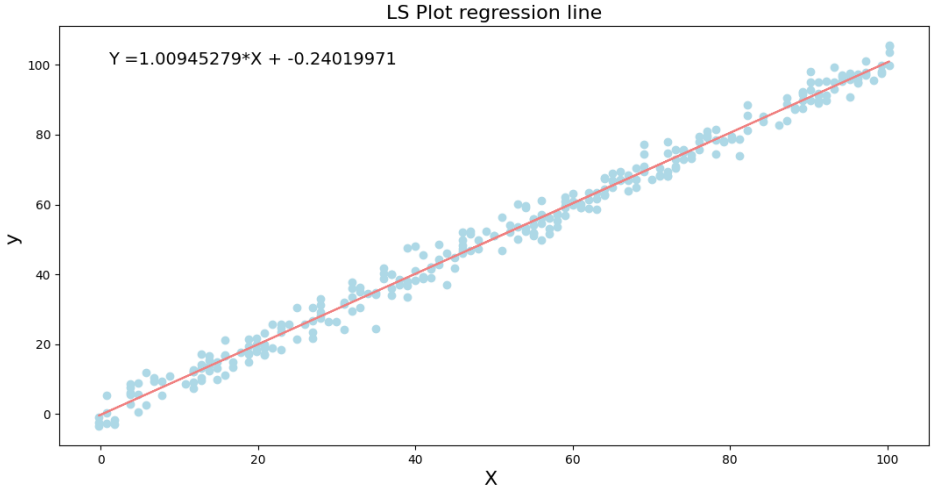
```
plt.figure(figsize=[8, 6])
plt.scatter(x_trn,y_trn, color='lightblue')
plt.scatter(x_tst,y_tst, color='lightcoral')
plt.legend(['Train data', 'Test Data'], fontsize=18)
plt.xlabel('Samples ', fontsize=16)
plt.ylabel('Errors ', fontsize=16)
plt.title('Least Squared Data Predicts', fontsize=16)
plt.show()
```

۱۳) نمایش نمودار رگرسیون خطی

```
theta1, theta0 = np.polyfit(yHat_tst, y_tst, 1)
plt.figure(figsize=[8, 6])
plt.scatter(yHat_tst, y_tst, color='lightblue')
plt.xlabel('X ', fontsize=16)
plt.ylabel('y ', fontsize=16)
plt.plot(yHat_tst, theta1 * yHat_tst + theta0, color='lightcoral')
plt.text(1, 100, 'Y =' + np.array2string(theta1) + '*X + ' +
np.array2string(theta0), fontsize=14)
plt.title('LS Plot regression line', fontsize=16)
plt.show()
```

در نهایت، پس از اجرای کد بالا، مقادیر زیر به عنوان خروجی نمایش داده می شوند:

```
################## Calculated Thetas ######################
 Closed Form Solution, theta0, theta1 :
  [ -0.21953584 100.48368424]
 -
MSE Train:
 8.328012371573907
 -
MSE Test:
 9.295693428531168
 -
```



روش دوم) Gradient Descent Regression (Stochastic)

روش Closed Form همیشه روش بهینه ای نیست و همچنین بعضی مواقع مدل خطی برای حل مسئله وجود ندارد. در این موارد از روش گرادیان کاهشی استفاده می کنیم. الگوریتم گرادیان کاهشی (gradient descent) الگوریتمی برای پیدا کردن مینیمم محلی یک تابع convex است. در این تمرین، روش استفاده شده برای پیاده سازی گرادیان کاهشی روش Stochastic می باشد.

روش gradient descent با حدس یک مقدار تتا شروع می شود، از تابع J(theta) روی همان theta مشتق جزئی گرفته می شود، در مقدار alpha (طول گام / نرخ یادگیری) ضرب می شود و از theta اولیه کم می شود. مقدار به دست آمده، theta دوم ما را نتیجه می دهد. این کار تا زمانی که به مینیمم تابع برسیم ادامه پیدا می کند.

توضیح کد:

در این کد از کتابخانه های pandas، numpy وmatplotlib استفاده شده است.

```
import numpy as np
import pandas as pd
from numpy.linalg import inv
import matplotlib.pyplot as plt
```

۱) تعریف تابع نرمالسازی

```
def normalize(X):
    Min = X.min()
    Max = X.max()
    return (X - Min) / (Max - Min), Min, Max
```

۲) تعریف تابع افزودن bias

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

X_0 عرض از مبدا (bias)، و همان ضریب تتا صفر (θ_0) است که همیشه ۱ قرار دارد. دلیل اصلی این است که اطمینان پیدا کنیم به طور پیش فرض خط از مبدا عبور نکند. بنابراین این تابع، اولین ستون ماتریس X را ۱ می گذارد.

۳) تابع محاسبه خطا

```
def ErrorCalculation(X, y, theta):
    return (y.reshape(y.shape[0], 1) -
X.dot(theta)).T.dot(y.reshape(y.shape[0], 1) - X.dot(theta))[0][0]
```

۴) تابع MSE

```
def mseError(y,y_hat):
    mse = np.mean(np.power(y - y_hat, 2))
    return mse
```

محاسبه mean squared error

۵) تابع gradientDescent

```
def gradientDescent(X, y, alpha=0.001, epsilon=0.0001):
    m, n = X.shape
    theta_0 = np.random.rand(n).reshape(n, 1)
    theta_1 = np.zeros((n, 1))
    i = 0
    errors = []
    while norm(theta_1 - theta_0) > epsilon:
        theta_0 = theta_1
        rss = ErrorCalculation(X, y, theta_0)
        temp = (X.T.dot(y))
        grad_rss = (np.matmul(X.T.dot(X), (theta_0)) -
                    temp.reshape(temp.shape[0], 1))
        theta_1 = theta_0 - alpha * grad_rss
        errors.append(np.squeeze(rss))
        i += 1

    errors = np.array(errors)
    return theta_1, errors
```

این تابع، گرادیان کاهشی را محاسبه می کند و تتا و خطا را برمی گرداند.

۶) خواندن دیتا از فایل

داده ها با کمک کتابخانه pandas وارد می شوند.

```
train_data = pd.read_csv('Data-Train.csv')
test_data = pd.read_csv('Data-Test.csv')
```

۷) تفکیک داده های مستقل (X) و وابسته (y)

```
x_trn = train_data['x'].values
y_trn = train_data['y'].values

x_tst = test_data['x'].values
y_tst = test_data['y'].values
```

۸) نرمال سازی داده

```
x_trn, trnMinX, trnMaxX = normalize(x_trn)
x_tst, tstMinX, tstMaxX = normalize(x_tst)
```

داده ها (X) را نرمال میکنیم. نرمال سازی هم بر روی داده Train و هم بر روی داده Test انجام می شود.

۹) افزودن bias

```
x_trnBiasAdded = biasAddition(x_trn)
x_tstBiasAdded = biasAddition(x_tst)
```

ستون اول ماتریس X در داده های Train و Test، ستون ۱ اضافه می شود. (عرض از مبدا)

۱۰) محاسبه تتا و خطا

```
gradientDescent_theta, Errors = gradientDescent(
    x_trnBiasAdded, y_trn, alpha=0.00001)
```

۱۱) محاسبه خروجی

```
y_GradientDescent_Train = gradientDescent_theta[0] + \
    gradientDescent_theta[1]*x_trn
y_GradientDescent_Test = gradientDescent_theta[0] + \
    gradientDescent_theta[1]*x_tst
```

۱۲) محاسبه mse

```
mseTrain = mseError(y_trn,y_GradientDescent_Train)
mseTest = mseError(y_tst,y_GradientDescent_Test)
```

۱۳) نمایش خطاهای Gradient Descent

```
plt.figure(figsize=[8, 6])
plt.plot(Errors, 'g.', linewidth=3.0)
plt.xlabel('Iteration ', fontsize=16)
plt.ylabel('Errors', fontsize=16)
plt.title('Gradient Descent Errors')
plt.show()
```

۱۴) نمایش پیش بینی داده

```
plt.figure(figsize=[8, 6])
plt.scatter(x_trn,y_trn, color='#458B74')
plt.scatter(x_tst,y_tst, color='#E3CF57')
plt.legend(['Train data', 'Test data'], fontsize=18)
plt.xlabel('Samples ', fontsize=16)
plt.ylabel('Errors ', fontsize=16)
plt.title('GD Data Predicts')
plt.show()
```

۱۵) نمایش نمودار رگرسیون خطی

```
theta1, theta0 = np.polyfit(y_GradientDescent_Test, y_tst, 1)
plt.figure(figsize=[8, 6])
plt.scatter(y_GradientDescent_Test, y_tst, color='#458B74')
plt.xlabel('Outputs ', fontsize=16)
plt.ylabel('Targets ', fontsize=16)
plt.plot(y_GradientDescent_Test, theta1*y_GradientDescent_Test + theta0,
color='#E3CF57')
plt.text(1, 100, 'Y =' + np.array2string(theta1) + '*X + ' +
np.array2string(theta0), fontsize=14)
plt.title('GD plot regression line')
plt.show()
```

در نهایت، پس از اجرای کد بالا، مقادیر زیر به عنوان خروجی نمایش داده می شوند:

################### Calculated Thetas ########################

Gradient Descent Solution, theta0, theta1:

[[-0.14696969]

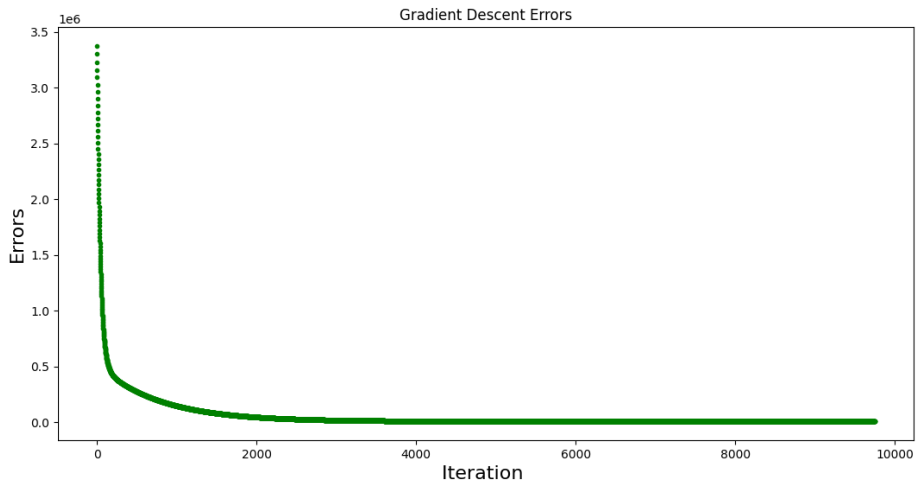
[100.3487619]]

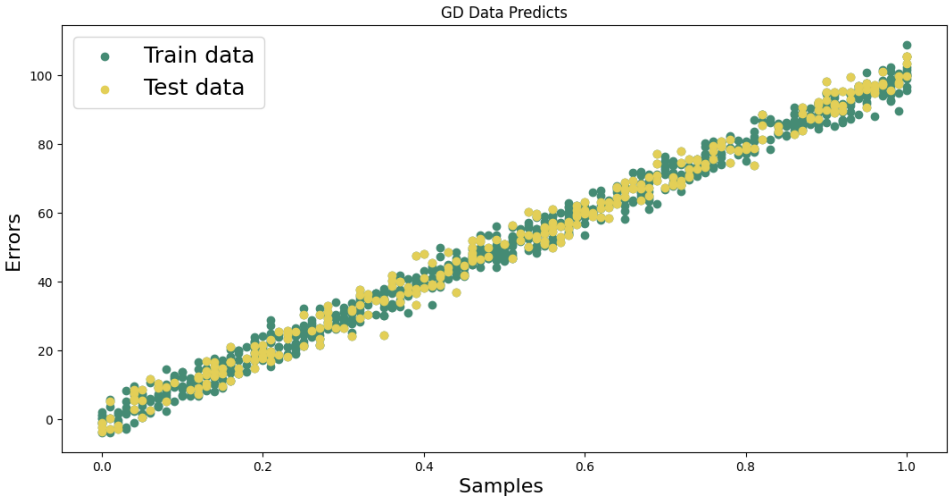
MSE Train:

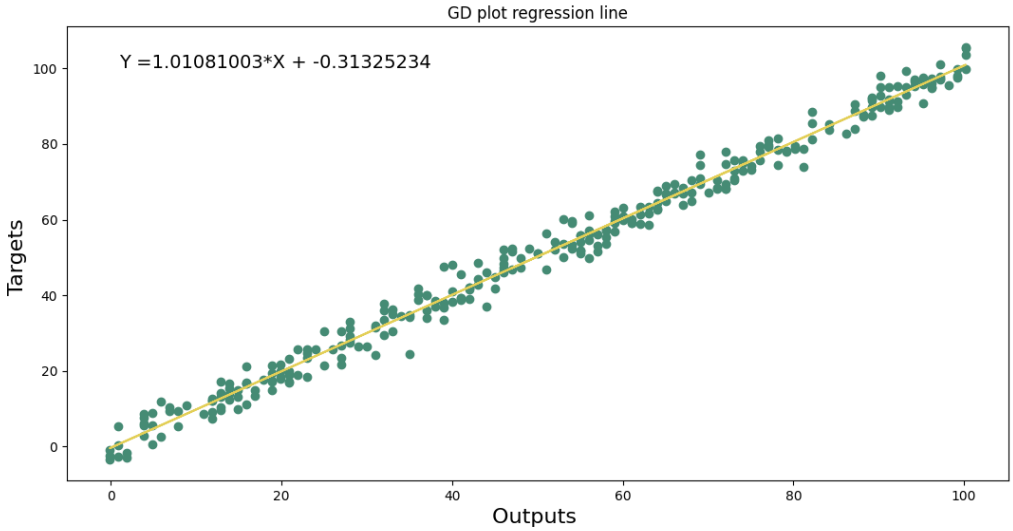
8.329543024155255

MSE Test:

9.316082460124539




GD Data Predicts
Train data
Test data
Errors
Samples
0
20
40
60
80
100
0.0
0.2
0.4
0.6
0.8
1.0


GD plot regression line
Y =1.01081003*X + -0.31325234
Targets
Outputs
0
20
40
60
80
100