بسم الله الرحمن الرحیم

گزارش تمرین ۱ :

رگرسیون خطی(Closed-Form)

نام و نام خانوادگی (شماره دانشجویی) :

زهرا امینی (۹۹۶۰۵۶۴)

امید توسلی فر (۹۹۳۱۱۵۱)

گرایش تحصیلی :

هوش مصنوعی

پاییز ۱۴۰۰

مقدمه :

رگرسیون[1] یکی از زیرشاخه‌های یادگیری بانظارت[2] است. رگرسیون یعنی تخمین رابطه بین یک متغیر وابسته و یک یا چند متغیر مستقل است. بطور خلاصه یعنی ، می‌خواهیم تابعی مانند " f " بدست آوریم که با استفاده از " x " ما را به " y " برساند. فرض کنید یک مجموعه نقطه دوبعدی (x,y) داریم ؛ حالا یادگیری ماشین به ما کمک می‌کند که رابطه بین ورودی و خروجی (f) را پیدا کنیم. خب در این صورت فرمول رگرسیون خطی باید به صورت ($y = \theta_0 x_0 + \theta_1 x_1$) باید باشد . با استفاده از این فرمول خطی برای فیت شدن داده ها (Samples) ایجاد می کنیم . ($x_0 = 1$)

بطور خلاصه برای آموزش مدل رگرسیون خطی باید این مراحل را انجام دهیم :

۱) مقادیر θ_1 و θ_0 را بدست می آوریم .
۲) همه داده های آموزشی x را به مدل رگرسیون خطی می‌دهیم و مقدار پیش بینی y_p را بدست می‌آوریم.
۳) میزان خطا را میان y_p و y محاسبه می‌کنیم .
۴) تابع خطای ($J(\theta)$) را بدست می آوریم .

خب برای پیاده سازی هر مدل نیاز به یک سری پیش نیازها می باشد. هر پیاده سازی شامل چهار(۴) فاز می باشد :

- فاز اول : گرفتن یا خواندن داده ها
- فاز دوم : پیش پردازش و نرمال سازی
- فاز سوم : پیاده سازی تابع اصلی از پایه
- فاز چهارم : نمایش یا پلات کردن خروجی خواسته شده

نکته : این تمرین در محیط ژوپیتر و به زبان پایتون پیاده سازی شده است .

---

[1] Regression

[2] Supervised Learning

❖ فاز اول : خواندن داده ها

در ابتدا برای انجام هر فرآیندی نیاز به کتابخانه های لازم برای آن کار است . در این تمرین ما مجاز به استفاده از سه کتابخانه زیر هستیم :

۱) **Numpy** : برای محاسبات ریاضی و انجام عملیات (تبدیل به ماتریس و غیره) بر روی داده ها
۲) **Panda** : برای گرفتن و خواندن فایل ورودی
۳) **Matplotlib** : برای نمایش و پلات کردن نمودار داده های خواسته شده

```
#import library we need
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

سپس با استفاده از کتابخانه **Panda** فایل داده های خود را فراخوانی می کنیم . تابع **train** برای فایل **data_train** و تابع **test** برای فایل **data_test** به صورت زیر می باشد .

```
#read our dataset
train = pd.read_csv('Data-Train.csv')
test = pd.read_csv('Data-Test.csv')
```

سپس داده ها (**sample**) فایل **train** را برای هر دو ستون به شکل زیر میخوانیم .

```
#get our sample from train data
X_train_raw = train.drop('y', axis=1)
y_train_raw = train['y']
y_train_raw = y_train_raw.to_frame()
```

```
print('The size of dataset is:' ,train.shape)
print('The size of X is:' ,X_train_raw.shape)
print('The size of y is:' ,y_train_raw.shape)
```

```
The size of dataset is: (1000, 2)
The size of X is: (1000, 1)
The size of y is: (1000, 1)
```

سپس داده ها (sample) فایل test را برای هر دو ستون به شکل زیر میخوانیم .

```
#get our sample from test data
X_test_raw = test.drop('y', axis=1)
y_test_raw = test['y']
y_test_raw = y_test_raw.to_frame()
```

```
print('The size of dataset is:' ,test.shape)
print('The size of X is:' ,X_test_raw.shape)
print('The size of y is:' ,y_test_raw.shape)
```

```
The size of dataset is: (300, 2)
The size of X is: (300, 1)
The size of y is: (300, 1)
```

❖ فاز دوم : پیش پردازش

در این فاز اگر نیاز به حذف داده های نویزی یا داده های گمشده (missing data) باشد در ابتدا انجام می دهیم که داده های ما هیچ کدام این مشکل را ندارند . پس به سراغ مرحله بعد آن یعنی نرمال سازی داده ها می رویم.

بهترین روش برای نرمال سازی داده های موجود روش نرمال سازی min.max است که معمولا برای داده های ترتیبی استفاده می کنیم و فرمول آن به صورت $x_i' = \frac{x_i - (min(x))}{max(x) - min(x)}$ می باشد . به راحتی می توان با این تابع داده های خود را در بازه (۰,۱) تعریف کرد. ابتدا تابعی بنام norm تعریف می کنیم . سپس m را براساس تعداد داده های x تعیین می کنیم (samples) . حلقه ای را تعریف می کنیم تا تمام بلوک های (داده های) x را در تابع محاسبه کند . سپس تابع ورودی جدید که بدست آمده را تبدیل به آرایه کرده و به X_norm می دهیم .

```
#main function for normalize our data with "minmax_normalization"
def norm(X):
    X_min = min(X)
    X_max = max(X)
    m=X.shape[0]
    X_norm = []

    for i in range(m):
        item = (X.iloc[i]-X_min) / (X_max-X_min)
        X_norm.append(item)

    X_norm = np.array(X_norm)
    X_norm = X_norm.reshape(m,1)
    return X_norm
```

اکنون تابع تعریف شده را برای داده های train انجام می دهیم . این کار را در یک فایل دیگر برای هردو ستون x و y انجام داده ایم ولی در این فایل خواسته شده تنها x نرمال سازی شود . در دوخط اول داده های ورودی x نرمال سازی شده اند.در خط سوم طبق فرمول گفته شده یک ستون جدید که تمام ورودی های آن یک می باشد برای محاسبه به این ماتریس اضافه می شود . داده های y به همان شکل خواسته شده فراخوانی شدند.

```
#normalized our train data
X_train_norm=norm(X_train_raw['x'])
X_train = X_train_norm
X_train = np.hstack([np.ones([X_train.shape[0],1]), X_train])
X_train.shape
y_train_norm=y_train_raw['y']
y_train = y_train_norm
y_train.shape
```

```
(1000,)
```

دوباره تابع تعریف شده را برای داده های test انجام می دهیم . این کار را در یک فایل دیگر برای هردو ستون x و y انجام داده ایم ولی در این فایل خواسته شده تنها x نرمال سازی شود . در دوخط اول داده های ورودی x نرمال سازی شده اند.در خط سوم طبق فرمول گفته شده یک ستون جدید که تمام ورودی های آن یک می باشد برای محاسبه به این ماتریس اضافه می شود . داده های y به همان شکل خواسته شده فراخوانی شدند.

```
#normalized our test data
X_test_norm=norm(X_test_raw['x'])
X_test = X_test_norm
X_test = np.hstack([np.ones([X_test.shape[0],1]), X_test])
X_test.shape
y_test_norm = y_test_raw['y']
y_test = y_test_norm
y_test.shape
```

```
(300,)
```

❖ فاز سوم : تابع اصلی

در این فاز تابع اصلی ما برای پردازش و یادگیری اتفاق می افتد . در این تمرین قصد داریم تابع رگرسیون خطی به روش فرم بسته (Closed Form) را پیاده سازی کنیم. در ابتدا فرمول ها و محاسبات لازم را توضیح می دهیم .

همانطور که اشاره شد فرمول رگرسیون خطی به صورت $y = \theta_0 x_0 + \theta_1 x_1$ که در آن θ_0 عرض از مبدا، θ_1 شیب خط و x_0 ماتریس اولیه که تمام ورودی های آن برابر یک می باشد و x_1 برابر داده ی ورودی ما است . شیب خط در حالت رگرسیون خطی ساده، نشان می‌دهد که میزان حساسیت متغیر وابسته به متغیر مستقل چقدر است. به این معنی که با افزایش یک واحد به مقدار متغیر مستقل چه میزان متغیر وابسته تغییر خواهد کرد. عرض از مبدا نیز بیانگر مقداری از متغیر وابسته است که به ازاء مقدار متغیر مستقل برابر با صفر محاسبه می‌شود. در اینجا هدف ما پیدا کردن θ_1 و θ_0 ، $h_\theta(x)$ ، تابع هزینه (Cost Function) یا خطای مربعی (Least Squared Method) می باشد .

ابتدا باید مقدار θ_1 و θ_0 را بدست آوریم که از فرمول $\theta = (x^T.x)^{-1} \times x^T y$ محاسبه می شوند.

```
#caculate theta0 & theta1
def find_theta(x, y):
    m = x.shape[0]

    theta = np.dot(np.linalg.inv(np.dot(x.T, x)), np.dot(x.T, y))
    return theta
```

سپس در تابع **hypothesis function** یا $h_\theta(x)$ قرار می دهیم که با فرمول $h_\theta(x^{(j)}) = \theta_0 + \theta_1 x_1^{(j)}$ محاسبه می شود .

```
#calculate h-theta or as we know hypothesis theta
def find_h_theta(x, theta):
    h_theta = 0
#h_theta += theta_0 + np.dot(theta_1, x.T)
    h_theta = np.dot(x, theta)
    return h_theta
```

در تابع قرار است مقدار خطای مربعی یا $J(\theta)$ را با فرمول $J(\theta) = \frac{1}{2} \times \sum_{j=1}^{m} (h(x^{(j)}) - y^{(j)})^2$ محاسبه می شود .

```
#calculate J-theta or as we know cost function(LSM)
def J_theta(x, y, h_theta):
    m=x.shape[0]
    J_theta = 0
    for i in range(m):
        J_theta += (1/2)*((h_theta[i] - y[i])**2)
    return J_theta
```

در خط اول تابع $h_\theta(x)$ **برای داده train بدست می آوریم و سپس با جایگذاری داده ها در تابع تعریف شده مقدار** $J(\theta)$ **یا همان خطا را بدست می آوریم.**

```
#calculate Cost function for train data
theta = find_theta(X_train, y_train)
h_theta_train = find_h_theta(X_train, theta)
y_pred_train = h_theta_train
J_train = J_theta(X_train, y_train, h_theta_train)
print('Least Squared Error Train:', J_train)
```

در خط اول تابع $h_\theta(x)$ **برای داده test بدست می آوریم و سپس با جایگذاری داده ها در تابع تعریف شده مقدار** $J(\theta)$ **یا همان خطا را بدست می آوریم.**

```
#calculate Cost function for test data
h_theta_test = find_h_theta(X_test, theta)
y_pred_test = h_theta_test
J_test = J_theta(X_test, y_test, h_theta_test)
print('Least Squared Error Test:', J_test)
```

❖ فاز چهارم : نمایش و پلات نمودارها

در این مرحله نمودارهای خواسته شده با استفاده از داده های فایل ورودی ترسیم می شود .

پلات داده های (samples) فایل train

```
#plot our final result of train data
plt.plot(X_train_norm, y_train, "bo")
plt.plot(X_train_norm, y_pred_train, "black")
plt.title("Train Data")
plt.xlabel("X")
plt.ylabel("y")
plt.show()
```

پلات داده های (samples) فایل test

```
#plot our final result of test data
plt.plot(X_test_norm, y_test, "bo")
plt.plot(X_test_norm, y_pred_test, "black")
plt.title("Test Data")
plt.xlabel("X")
plt.ylabel("y")
plt.show()
```

نحوه دیگر نمایش داده های هر دو فایل train و test

```
#our plot for train & test data (linear regression : Closed form model)
fig, plots = plt.subplots(1,2)
fig.suptitle('Closed Form ')

#plot our data train
plots[0].scatter(X_train_norm, y_train,color = "green", s = 0.1, label = 'Train data')
plots[0].plot(X_train_norm, y_pred_train, color = "black")
plots[0].set(xlabel='X', ylabel='Y')
plots[0].legend(loc = 'upper left')

#plot our data test
plots[1].scatter(X_test_norm, y_test,color = "red", s = 0.1, label = 'Test data')
plots[1].plot(X_test_norm, y_pred_test, color = "black")
plots[1].set(xlabel='X')
plots[1].legend(loc = 'upper left')

plt.show()
```

نتیجه گیری :

- **خروجی خطای مربعی (Least Squared Error)**

۱) خروجی تابع خطا مربعی برای داده train در دو وضعیت متفاوت به شکل زیر است :

در این محاسبات هر دو داده x و y نرمال شده بودند .

```
Least Squared Error Train: [0.32777373]
```

در این محاسبات فقط داده x نرمال سازی شده است .

```
Least Squared Error Train: 4164.00618578695
```

۲) خروجی تابع خطا مربعی برای داده test در دو وضعیت متفاوت به شکل زیر است :

در این محاسبات هر دو داده x و y نرمال شده بودند .

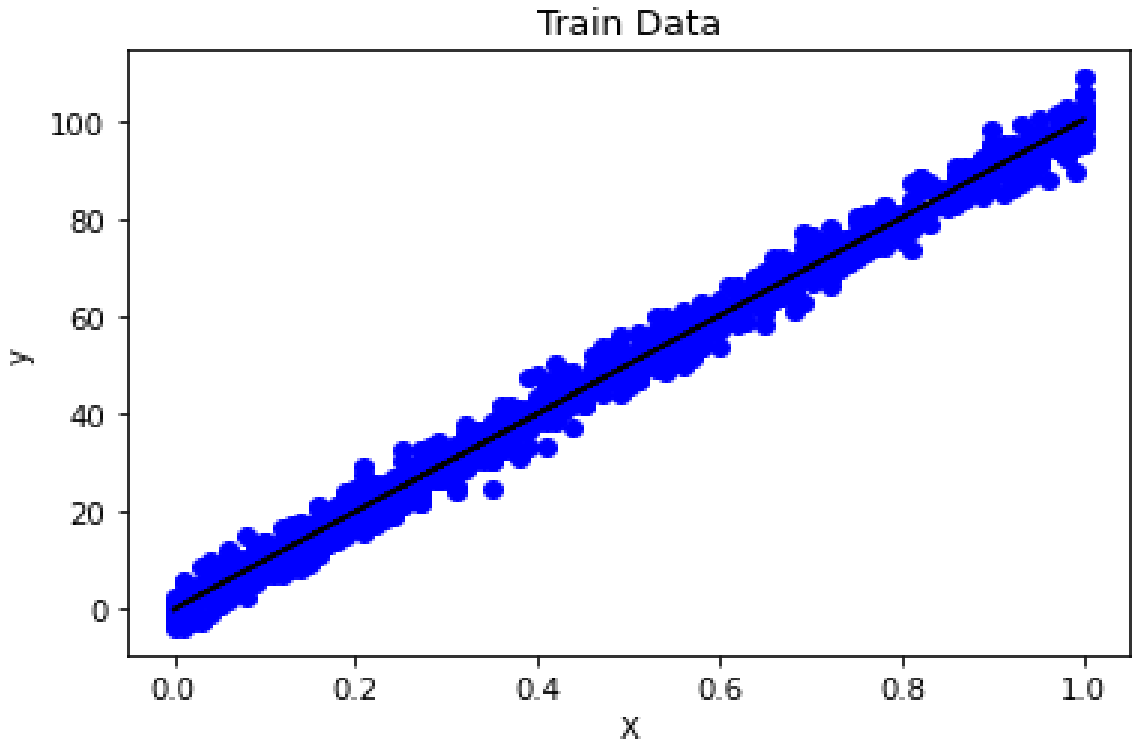
```
Least Squared Error Test: [0.16776734]
```

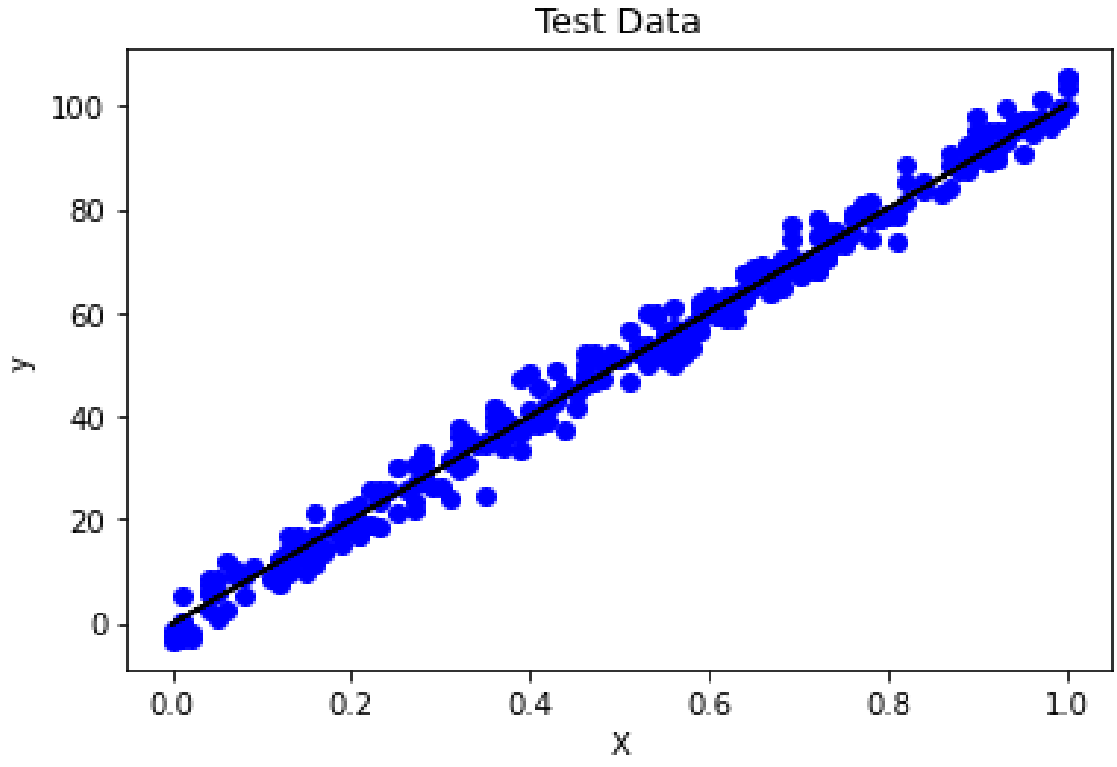
در این محاسبات فقط داده x نرمال سازی شده است .

```
Least Squared Error Test: 1394.35401427967
```

- **خروجی پلات ها**

خروجی پلات های train و test به صورت زیر می باشد که داده ها به رنگ آبی نمایش داده می شوند .





خروجی کلی داده ها با نمای دیگر