بسم الله الرحمن الرحیم

گزارش تمرین ۱ :

رگرسیون خطی (Stochastic Gradient Descent)

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. در این تمرین قرار است تا از الگوریتم گرادیان کاهشی تصادفی[3] برای پیاده سازی استفاده کنیم . الگوریتم گرادیان کاهشی، یک الگوریتم بهینه‌سازی تکراری مرتبه-اول هست که مینیمم محلی در یک تابع مشتق‌پذیر را پیدا می‌کند. در ادامه با نحوه کار و پیاده سازی این الگوریتم بیشتر آشنا می شویم .

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**
- **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

[3] Stochastic Gradient Descent

❖ 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')
```

❖ فاز دوم : پیش پردازش

در این مرحله داده های دو فایل **train** و **test** رو تبدیل به ماتریس کرده سپس **transpose** می کنیم . (تبدیل به یک وکتور)

```
#change our dataset to matrix and transpose them
x_train = np.transpose(np.asmatrix(train['x']))
y_train = np.transpose(np.asmatrix(train['y']))

x_test = np.transpose(np.asmatrix(test['x']))
y_test = np.transpose(np.asmatrix(test['y']))
```

ماتریس $x_0 = 1$ را به ماتریس **x(train)** اضافه می کنیم و ماتریس جدید را تولید می کنیم .

```
#add x0 to our data
ones = np.ones((x_train.shape[0],1))
x_new = np.append(ones, x_train, 1)
```

در این مرحله **m** و **n** را تعریف می کنیم که **m** تعداد داده ها(**sample**) ماتریس **x** (که با ۱۰۰۰ برابر است) را می گیرد و **n** تعداد ستون های ماتریس **x** (که برابر ۲ است) را می گیرد . سپس تتا را تعریف می کنیم که یک ماتریس (۲*۱) است . سپس آنرا **transpose** می کنیم .

```
#creat our m , n  and theta fill with 1 (2*1)
m, n = np.shape(x_new)
theta = np.ones((1,n))
theta = np.transpose(theta)
theta_new = theta
```

❖ فاز سوم : تابع اصلی

گرادیان کاهشی تصادفی (**Stochastic**) روشی مبتنی بر تکرار برای بهینه‌سازی یک تابع مشتق‌پذیر به نام تابع هدف (تابع هزینه) است که یک تقریب تصادفی از روش گرادیان کاهشی می‌باشد. در حقیقت گرادیان کاهشی تصادفی الگوریتمی در اختیار ما قرار می‌دهد که طی چند حلقه تکرار مقدار کمینه یک تابع و مقادیری را که با ازای آن‌ها تابع کمینه مقدار خود را می‌گیرد، بدست بیاوریم. در ابتدا به صورت دستی تعداد تکرار ها (**iteration**) و نرخ یادگیری(**Learning Rate**) را تعریف می کنیم . در اینجا تابع θ را که قبلا تعریف کردیم ، استفاده می کنیم .تابع هزینه (**LSM**) را تعریف می کنیم و تمام ورودی های آن را برابر با صفر قرار می دهیم. سپس در حلقه اول براساس تعداد داده ها (**m =۱۰۰۰**) شروع به اجرای حلقه داخلی برای محاسبه θ می کنیم که از فرمول $\theta_i = \theta_i - \alpha \sum_{j=1}^{m} (h_\theta(x^{(j)}) - y^{(j)}) x_i^{(j)}$ محاسبه می شود.

پس از اتمام حلقه داخلی و پیدا کردن θ_0وθ_1 تابع $(h_\theta(x^{(j)})$ را محاسبه می کنیم . در حلقه بعدی تعداد تکرار برای تابع هزینه را انجام می دهیم که از فرمول $j(\theta)_{new} = \frac{j(\theta) + \sum (y^{(j)} - h_\theta(x^{(j)}))^2}{2 \times iteration}$ برای گرادیان کاهشی تصادفی استفاده می شود.

```
#choose our iteration & Learning Rate

itr= 200
alpha = 0.00004

#calculate our Gradient Descent (Stochastic)
LSM = np.zeros((itr,2))
d = 0

while d < itr:
    for i in range(0, m):
        for j in range(0, n):
            theta_new[j] = theta[j] - (alpha * ((x_new[i]*theta - y_train[i])*x_new[i, j]))

        theta = theta_new

    y_pred = theta[0] + theta[1]*x_train.T

    Loop_LSM = 0
    for i in range(0, m):
        Loop_LSM += (y_train.A1[i] - y_pred.A1[i])**2
    Loop_LSM = Loop_LSM/(2*m)
    LSM[d,0] = d
    LSM[d,1] = Loop_LSM

    d += 1

print('Number of iteration : ',itr)
print('Learning Rate : ',alpha)
print('Theta[0] = ', theta[0])
print('Theta[1] = ', theta[1])
```

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 our h(theta) or y_predection for both data
y_pred_train = theta[0] + theta[1]*x_train.T

y_pred_test = theta[0] + theta[1]*x_test.T
```

تابع هزینه یا خطای مربعی را برای داده های train محاسبه می کنیم . (فرمول گفته شده)

```
#calculate our test data cost function
train_LSM = 0
m = y_train.shape[0]

for i in range(m):
    train_LSM += ((y_train.A1[i] - y_pred_train.A1[i])**2)
train_LSM = train_LSM/(2*m)

print("Train Least Squared Error : ", train_LSM)
```

همچنین این عمل را مجددا برای داده های **test** نیز انجام می دهیم .

```
#calculate our test data cost function
test_LSM = 0
m = y_test.shape[0]

for i in range(m):
    test_LSM += ((y_test.A1[i] - y_pred_test.A1[i])**2)
test_LSM = test_LSM/(2*m)

print("Test Least Squared Error : ", test_LSM)
```

❖ فاز چهارم : نمایش و پلات نمودارها

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

```
#plot our data and Gradient
fig, plots = plt.subplots(2,2)
fig.suptitle(' Stochastic Gradient Descent')

plots[0,0].scatter(train['x'],train['y'],color = "green", s = 0.1, label = 'Train data')
plots[0,0].plot(x_train.T.A1, y_pred_train.A1, color = "black")
plots[0,0].set(xlabel='X', ylabel='Y')
plots[0,0].legend(loc = 'upper left')

plots[0,1].scatter(test['x'],test['y'],color = "red", s = 0.1, label = 'Test data')
plots[0,1].plot(x_train.T.A1, y_pred_train.A1, color = "black")
plots[0,1].set(xlabel='X')
plots[0,1].legend(loc = 'upper left')

plots[1,0].plot(LSM.T[0], LSM.T[1], color = "black")
plots[1,0].set(xlabel='Iteration', ylabel='LSM')
plots[1,0].legend(loc = 'upper left')

plt.show()
```

نتیجه گیری :

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

۱) خروجی تابع خطا مربعی برای داده **train** به شکل زیر است :

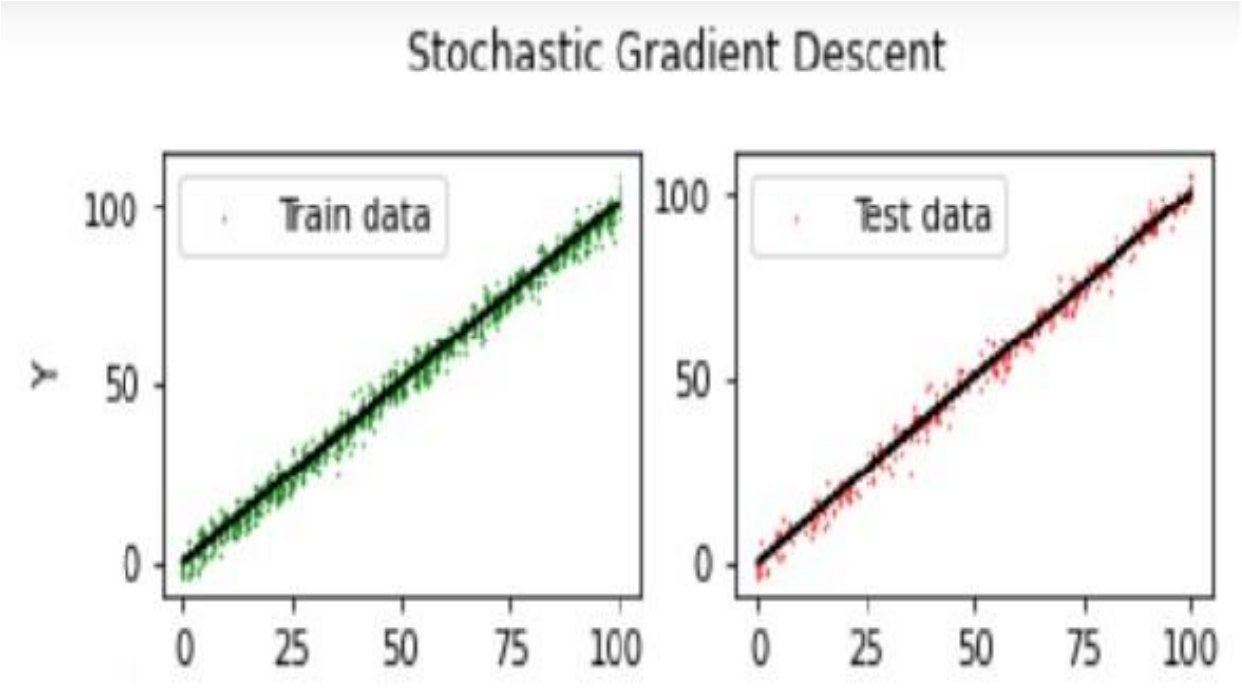
```
Train Least Squared Error :  4.185689874184812
```

۲) خروجی تابع خطا مربعی برای داده **test** به شکل زیر است :

```
Test Least Squared Error :  4.610875131374379
```

- **خروجی پلات ها**

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



نمودار LSM برای تعداد ۲۰۰ تکرار (iteration) که به صورت زیر می باشد .