سوال پنجم

در این سوال قرار است انواع رگرسیون را پیاده‌سازی کنیم. داده‌های این سوال در فایل‌های Data.mat و data.npz قرار دارند. این داده‌ها بر اساس رابطه زیر تولید شده‌اند

$$y = 4x_2^2 x_1 + 2x_x^2 + 3x_1 + 1$$

بنابراین نمونه‌ها به صورت $[x_1, x_2]$ و خروجی متناظرشان y است.

در فایل‌های ذکر شده شش آرایه یک بعدی، $y, x_1, x_2, y_{test}, x_{1,test}, x_{2,test}$ ، قرار دارد.

در هر سه حالت تابع هزینه ما تابع SSE یا Sum of Squared Errors خواهد بود.

سه حالت را با استفاده از Gradient decent و Stochastic gradient پیاده‌سازی کرده و نتایج را روی داده‌های تست با هم مقایسه کنید. فرض کنید که فرم کلی رابطه y و x ها را میدانیم و از فرمول بسته رگرسیون خطی / خطی تعمیم یافته برای محاسبه ضرایب w استفاده کنید.

برای داده تست، نتیجه کد خودتان و مقادیر صحیح خروجی که در آرایه ytest است را بر حسب x1,test و x2,test به صورت نمودار سه بعدی نمایش دهید.

مقدار تابع خطا روی داده‌های آموزش و داده‌های تست را برای هر یک از سه حالت گزارش نمایید.

a) روش کاهش گرادیان

P5a.py

فراخوانی کتابخانه‌های لازم

```
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
```

تابع آموزش گرادیان

ورودی‌ها شامل:

مقادیر ویژگی‌ها

مقادیر هدف

ماتریس وزن

نرخ یادگیری

ابعاد مسئله

تعداد تکرار آموزش

```
def Gradient_Descent(Input, Target, Weights, Learning_Rate, Dimen, Iter):
```

ابتدا ماتریس ویژگی‌ها را ترانهاده می‌کنیم برای استفاده آینده

```
    Input_Transpose = Input.transpose()
```

آرایه‌ای برای ذخیره خطای حین آموزش

```
    SSE_Array = np.zeros(Iter)
```

تا تعداد تکرار مشخص آموزش را انجام می‌دهیم

```
    for in_Iter in range(0, Iter):
```

ابتدا با توجه به وزن‌های فعلی مقدار خروجی را محاسبه می‌کنیم

```
        Predicted_Target = np.dot(Input, Weights)
```

خطای پیش‌بینی را محاسبه می‌کنیم

```
        Error = Predicted_Target - Target
```

مقدار خطای خواسته شده در سوال محاسبه می‌شود

```
        SSE = np.sum(Error ** 2)
```

گرادیان را به دست می‌آوریم با ضرب خطا در ورودی

```
        gradient = np.dot(Input_Transpose, Error) / Dimen
```

سپس وزن‌ها را به‌روزرسانی می‌کنیم

```
        Weights = Weights - Learning_Rate * gradient
```

مقدار خطا را ذخیره می‌کنیم

```
        SSE_Array[in_Iter] = SSE
```

نمایش اطلاعات آموزشی

```
        print("Iter %d with SSE: %f" % (in_Iter, SSE))
```

وزن‌ها و آرایه خطا را برمی‌گردانیم

```
    return Weights, SSE_Array
```

خواندن داده‌ها

```
# Main Code
Data_Raw = np.load('data.npz')
x1 = Data_Raw.f.x1
x1 test = Data Raw.f.x1 test
x2 = Data Raw.f.x2
x2_test = Data_Raw.f.x2_test
y = Data_Raw.f.y
y_test = Data_Raw.f.y_test
```

آماده کردن مقادیر داده‌ها و اضافه کردن ترم بایاس. داده‌های آموزش و تست جداگانه آماده می‌شوند.

این داده‌ها با توجه به رابطه صورت مسئله آماده می‌شوند.

```
Bias Train = np.ones([np.shape(x1)[0],Gradien Order])
Bias Test = np.ones([np.shape(x1 test)[0],Gradien Order])
x_train = np.column_stack((np.multiply(x1,x2 **2),x2 **2,x1,Bias_Train))
x_test = np.column_stack((np.multiply(x1_test,x2_test **2),x2_test **2,x1_test,Bias_Test))
```

به آخر ویژگی‌ها مقدار بایاس ۱ را اضافه می‌کنیم

```
x = np.column stack((x1,x2,Bias))
```

سایز نمونه‌ها

```
Sample Size, Dimen = np.shape(x train)
```

تعداد تکرار الگوریتم

```
Iter= 1000
```

نرخ یادگیری

```
Learning Rate = 0.0000001
```

مقداردهی اولیه ماتریس وزن‌ها

```
Weights = np.ones(Dimen)
```

آموزش گرادیان

```
Weights, SSE Array = Gradient Descent(x train, y, Weights, Learning Rate, Sample Size, Iter)
```

محاسبه خطای پیش‌بینی داده‌های آموزش و آزمایش با وزن‌های به دست آمده

```
y_p_test = Weights[0]*x_test[:,0] + Weights[1]*x_test[:,1] + Weights[2]*x_test[:,2]+
Weights[3]*x_test[:,3]
y_p_train = Weights[0]*x train[:,0] + Weights[1]*x train[:,1] + Weights[2]*x train[:,2]+
Weights[3]*x_train[:,3]


Error = y_p_test - y_test
SSE Test = np.sum(Error ** 2)
Error = y_p_train - y
SSE_Train = np.sum(Error ** 2)

print("SSE > Test = %f   &  Train =  %f " % (SSE_Test,SSE_Train))
```

ترسیم داده‌های تست و داده‌های پیش‌بینی شده

```
# Plot Target and Perdicted Target
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.scatter(x1_test, x2_test, y_test , c='red')
ax.scatter(x1_test, x2_test, y_p_test , c='blue')
plt.xlabel('x1')
plt.ylabel('x2')
plt.ylabel('y')
ax.legend(['Target','Prediction'])
```

ترسیم نمودار خطای حین آموزش

```
# Plot Train SSE line
fig = plt.figure()
plt.plot(range(0,Iter),SSE_Array,c='red')
plt.xlabel('Iteration')
plt.ylabel('SSE')
plt.legend(['Train SSE Line'])
plt.show()
```

b) روش کاهش گرادیان تصادفی

P5b.py

است. بنابراین از توضیحات مکررات خودداری کرده و تنها تفاوت‌ها را توضیح می‌دهیم P5aاکثر توضیحات مشابه با فایل

```
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt


def Stochastic_Gradient_Descent(Input, Target, Weights, Learning_Rate, Dimen, Iter):
    Input_Transpose = Input.transpose()
    SSE_Array = np.zeros(Iter)
```

طول داده‌ها را محاسبه می‌کنیم

```
    m = len(Target)
    for in_Iter in range(0,Iter):

        Final SSE = 0
```

در این حلقه، داده‌ها را به صورت تصادفی انتخاب کرده و به الگوریتم کاهش گرادیان تصادفی می‌دهیم

```
        for i in range(m):
```

انتخاب یک اندیس تصادفی

```
            rand ind = np.random.randint(0,m)
```

انتخاب یک داده بر اساس اندیس تصادفی

```
            Random_Input = Input[rand_ind,:].reshape(1,Input.shape[1])
```

انتخاب برچسب آن داده

```
            Random Target = Target[rand ind].reshape(1,1)
```

محاسبه خطای وزن‌های فعلی، روی این داده تصادفی و محاسبه خطای پیش‌بینی

```
            Predicted Target = np.dot(Random Input,Weights)
            Error = Predicted Target - Random Target
            SSE = np.sum(Error ** 2)
```

محاسبه گرادیان

```
            gradient = np.dot(Random Input.T, Error) / Dimen
```

به‌روزرسانی وزن‌ها

```
            Weights = Weights  - Learning Rate * np.transpose(gradient)[0]
```

محاسبه مجموع خطا برای هر تکرار و به ازای نمونه‌های تصادفی انتخاب شده

```
            Final SSE += SSE
        SSE Array[in Iter] = Final SSE
        print("Iter %d with SSE: %f" % (in_Iter, Final_SSE))
    return Weights, SSE_Array


# Main Code
Data_Raw = np.load('data.npz')
x1 = Data_Raw.f.x1
x1_test = Data_Raw.f.x1_test
x2 = Data_Raw.f.x2
x2_test = Data_Raw.f.x2_test
```

```
y = Data Raw.f.y
y test = Data Raw.f.y test

Gradien_Order = 1
Bias_Train = np.ones([np.shape(x1)[0],Gradien_Order])
Bias Test = np.ones([np.shape(x1 test)[0],Gradien Order])
x train = np.column stack((np.multiply(x1,x2 **2),x2 **2,x1,Bias Train))
x test = np.column stack((np.multiply(x1 test,x2 test **2),x2 test **2,x1 test,Bias Test))

Sample_Size, Dimen = np.shape(x_train)
Iter= 1000
Learning Rate = 0.0000001
Weights = np.ones(Dimen)
Weights, SSE Array = Stochastic Gradient Descent(x train, y, Weights, Learning Rate, Sample Size, Iter)

y_p_test = Weights[0]*x_test[:,0] + Weights[1]*x_test[:,1] + Weights[2]*x_test[:,2]+
Weights[3]*x test[:,3]
y p train = Weights[0]*x train[:,0] + Weights[1]*x train[:,1] + Weights[2]*x train[:,2]+
Weights[3]*x train[:,3]

Error = y_p_test - y_test
SSE_Test = np.sum(Error ** 2)
Error = y p train - y
SSE Train = np.sum(Error ** 2)

print("SSE > Test = %f   &  Train =  %f " % (SSE_Test,SSE_Train))

# Plot Target and Perdicted Target
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.scatter(x1_test, x2_test, y_test , c='red')
ax.scatter(x1_test, x2_test, y_p_test , c='blue')
plt.xlabel('x1')
plt.ylabel('x2')
plt.ylabel('y')
ax.legend(['Target','Prediction'])

# Plot Train SSE line
fig = plt.figure()
plt.plot(range(0,Iter),SSE_Array,c='red')
plt.xlabel('Iteration')
plt.ylabel('SSE')
plt.legend(['Train SSE Line'])
plt.show()
```