دانشگاه تهران

دانشکده علوم مهندسی

تکتم سمیعی
۸۱۰۸۹۶۰۵۴

یادگیری ماشین – دکتر سایه میرزایی

تمرین اول

اسفند ۹۹

سوال اول

بخش ۱) ابتدا کتابخانه های matplotlib , numpy , pandas را import میکنیم .

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from pandas import Series
from numpy.random import randn
from mpl_toolkits import mplot3d
```

Matplotlib : امکان plot کردن نمودار های مختلف را مانند متلب برای ما فراهم میکند .

Pandas : کتابخانه ای برای استفاده از ساختمان های داده و تحلیل داده هاست .

Numpy : کتابخانه ای برای استفاده از آرایه ها و داده های بزرگ و کارکردن با توابع جبری و ریاضی .

mpl_toolkits : برای رسم نمودار های سه بعدی استفاده شده است .

در این قسمت داده های آموزشی را موجود در فایل test.csv را با دستور read_csv از کتابخانه ی pandas لود میکنیم . داده های آموزشی در دو ستون y, x قرار دارند.

برای مدل رگرسیون خطی ، انواع توابع خطا وجود دارد . ۵ نوع آن را به صورت خلاصه در زیر توضیح میدهیم :

۱ . تابع هزینه ی میانگین خطای مربعات(MSE) : رایج ترین تابع هزینه ی رگرسیون خطی است که به صورت میانگین مجموع توان ۲ ی اختلاف بین مقدار هدف و مقدار پیش بینی شده است :

$$MSE = \frac{\Sigma_{i=1}^{n}(y_i - h_\theta(x_i))^2}{2n}$$

۲ . تابع هزینه ی خطای قدر مطلق(MAE) : در این روش به جای به توان ۲ رساندن اختلاف بین مقدار هدف و مقدار پیش بینی شده ، از قدر مطلق این اختلاف استفاده میشود :

$$MAE = \frac{\Sigma_{i=1}^{n}|y_i - h_\theta(x_i)|}{2n}$$

اگر احتمال دهیم که دیتای ما با داده های پرت خراب شده است ، استفاده از تابع هزینه ی خطای قدر مطلق بهتر است زیرا این اخلاف زیاد بین مقدار پیش بینی شده و مقدار هدف را مانند روش خطای مربعات به توان ۲ نمیرساند . اما روش MAE به دلیل پیوسته نبودن مشتقات ، برای رسین به جواب خیلی کارآمد نیست .

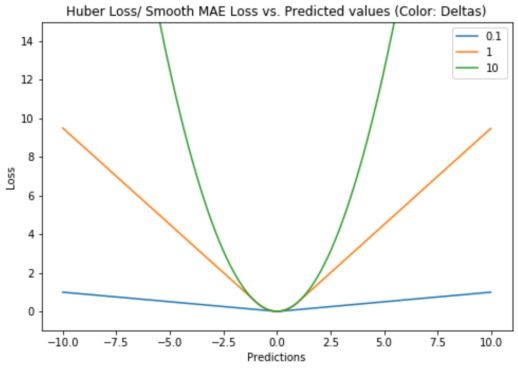
نتیجه : اگر داده های پرت نشان دهنده ی ناهنجاری باشند و برای تحلیل ضروری باشند ، از روش MSE استفاده میکنیم . در غیر این صورت برای کم کردن اثر این داده ها از روش MAE باید استفاده کرد .

۳ . تابع هزینه ی میانگین خطای قدرمطلق نرم یا Huber loss : این روش نسبت به MSE نسبت به داده های پرت کمتر حساسیت دارد و همچنین در صفر مشتق پذیر است . در حقیقت تلفیقی از هردو روش MAE , MSE است . برای زمانی که اختلاف بین مقدار هدف و مقدار پیش بینی شده کم باشد از خطای مربعات استفاده میکند و در صورت زیاد بودن آن ، از MAE استفاده میکند.

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for} |y - f(x)| \le \delta, \\ \delta\,|y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases}$$

تشخیص این امر ، با متغیر δ امکان پذیر است .

در تصویر روبه رو میبینید که تابع هزینه به متغیر دلتا وابسته است .



Plot of Hoss Loss (Y-axis) vs. Predictions (X-axis). True value = 0

توضیح کد :

ابتدا باید تابع gradient descent را تعریف کنیم . به این صورت که که بردار های x , y را به عنوان ورودی میگیرد (داده های آموزشی) و همچنین بردار theta که به صورت رندوم در ابتدا تعیین میشود . دیگر ورودی های این مسئله alpha است که learning rate است و انتخاب آن در همگرایی مسئله و رسیدن به جواب بسیار اهمیت دارد . دیگر ورودی مسئله تعداد تکرار هاست که میزان تکرار حلقه ی for است . انتخاب این پارامتر نیز بسیار مهم است و اگر از حد مورد نظر کمتر انتخاب شود ، به جواب درست نمیرسیم و اگر عدد بزرگی انتخاب شود زمان بیهوده ای را سپری کرده ایم .

در این تابع ، در هر تکرار حلقه ی for مقدار H را که در اینجا prediction مینامیم از رابطه ی زیر میابیم :

$$prediction = \Theta^T X$$

سپس از رابطه ی $\theta := \theta - \alpha(X^T(prediction - Y))$ در هر تکرار ، تتای جدیدی به دست می آوریم و با مقدار جدید ، تکرار بعدی را انجا میدهیم . با انتخاب صحیح alpha و iterations ، در نهایت به جواب همگرا میشویم .

```
def grad_desc(x,y,theta,alpha,iterations):
    m=len(y)
    # this loop updates the magnitude of theta vector
    for i in range(iterations):
        #we start prediction with initial condition
        prediction = np.dot(x,theta)
        theta = theta - (1/m)*alpha*(np.dot(x.T,(prediction - y)))
    cost = call_cost(theta,x,y)
    plt.plot(x[:,1], y ,'ro',x[:,1],prediction , 'g^' )
    return theta , cost
```

در این تابع ، همزمان داده های آموزشی (نقاط قرمز) و مدل رگرسیون به دست آمده (مثلث های سبز) را نمایش میدهیم .

بخش ۲)

برای محاسبه ی هزینه ، تابع call_cost را تعریف میکنیم که با داشتن ورودی های theta ی نهایی و x , y (نمونه های آموزشی اولیه) مقدار خطارا با رابطه ی زیر محاسبه میکنیم :

$$cost = \frac{1}{2m}\Sigma(\theta^T X - Y)^2$$

```
def call_cost(theta,x,y):
    m=len(x)
    prediction = np.dot(x,theta)
    cost = (1/(2*m))* np.sum(np.square(prediction - y ))
    return cost
```

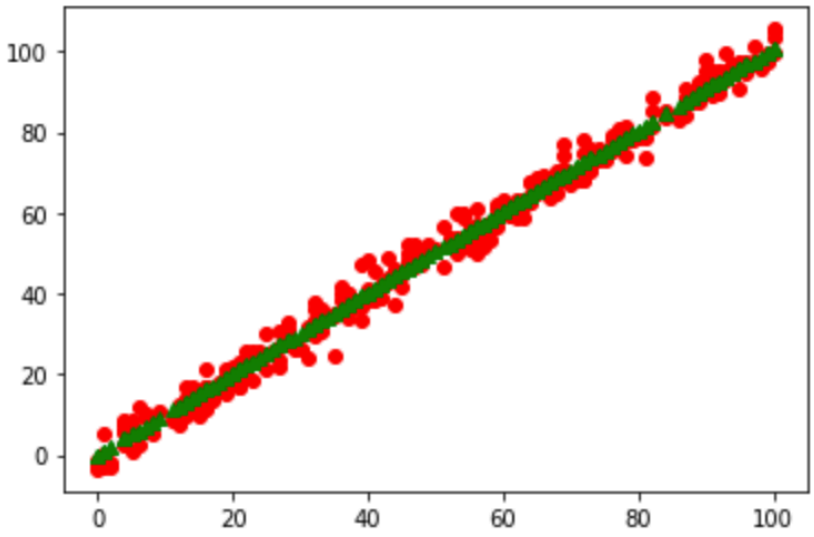
در این قسمت ، از روش میانگین خطای مربعات استفاده کردیم .

در نهایت با مقدار دهی alpha و iterations و مقدار رندوم theta ، مقدار نهایی theta و cost را محاسبه میکنیم .

```
alpha = 0.0000001
iterations = 100000
theta = np.random.randn(2,1)  # random initial condition
# this line concatinate 1 to a x vector to make theta vector and x vector the same size
X_b = np.c_[np.ones((len(x),1)),x]
theta_ , cost = grad_desc (X_b , y , theta ,alpha , iterations)
print("theta0 : ",theta_[0][0],sep=' ')
print("theta1 : ",theta_[1][0],sep=' ')
print("H_theta = ",theta_[0][0], " + ","X ", theta_[1][0])
print("cost : ",cost,sep=' ')
```

```
theta0 : 1.0816682526550885
theta1 : 0.9912397183315408
H_theta =  0.030043391474020305  +  X  1.006975562756334
cost  : 4.86548536083836
```

بخش ۳)



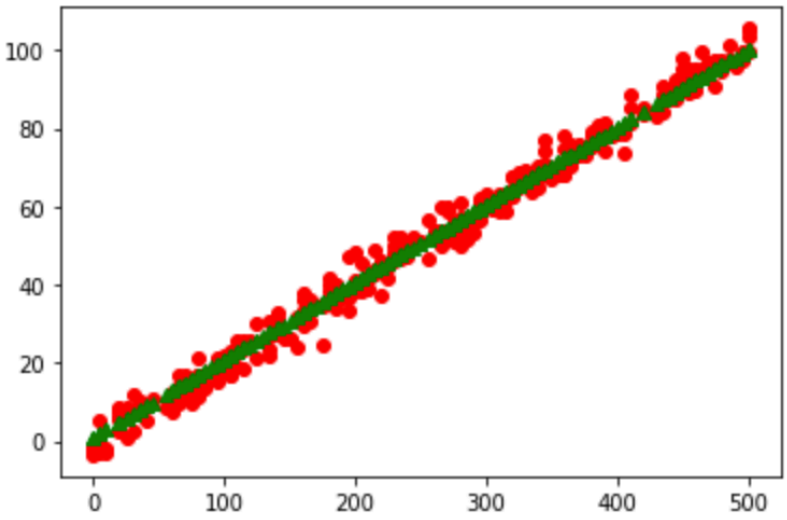
با توجه به شکل ، میتوانیم تطابق مدل رگرسیون به دست آمده را بر داده های آموزشی ببینیم .

بخش ۴) متغیر جدیدی تعریف میکنیم به صورتی که این متغیر ۵ برابر متغیر قبلی است و مجددا مدلی بر داده های آموزشی به دست می آوریم :

```
X_b_2 = np.c_[np.ones((len(x),1)),5*x]
theta_ , cost = grad_desc (X_b_2 , y , theta ,alpha , iterations)
print("theta0 : ",theta_[0][0],sep=' ')
print("theta1 : ",theta_[1][0],sep=' ')
print("cost : ",cost,sep=' ')
```

```
theta0 : 1.0457694037357341
theta1 : 0.1983556797315305
cost : 4.852458495722277
```

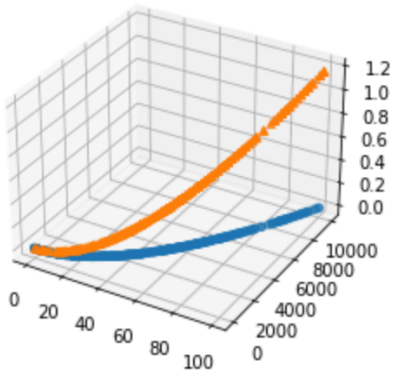
با مقایسه ی نتایج به دست آمده از از این قسمت و قسمت قبل ، متوجه میشویم که عرض از مبدا دو قسمت تقریبا یکسان هستند ولی شیب قسمت قبل تقریبا ۵ برابر شیب خط به دست آمده در این بخش است زیرا داده های ورودی (X) در این قسمت ۵ برابر شده اند و منطقی است که شیب خط ۱/۵ شود .



بخش ۵) چون قسمت plot در درون تابع grad_desc نوشته شده بود و در این قسمت نیاز به ترسیم تابع دو بعدی داریم ، مجددا برای این بخش تابع جدیدی تعریف میکنیم و بخش plot را سه بعدی میکنیم :

```
def grad_desc_2(x,y,theta,alpha,iterations):
    m=len(y)

    # this loop updates the magnitude of theta vector
    for i in range(iterations):
        #we start prediction with initial condition
        prediction = np.dot(x,theta)
        theta = theta - (1/m)*alpha*(np.dot(x.T,(prediction - y)))
    cost = call_cost(theta,x,y)
    fig = plt.figure()
    ax = plt.axes(projection='3d')
    ax.scatter(x[:,1],x[:,2],y,marker='o')
    ax.scatter(x[:,1],x[:,2],prediction,marker='^')
    plt.show()
    return theta , cost
```



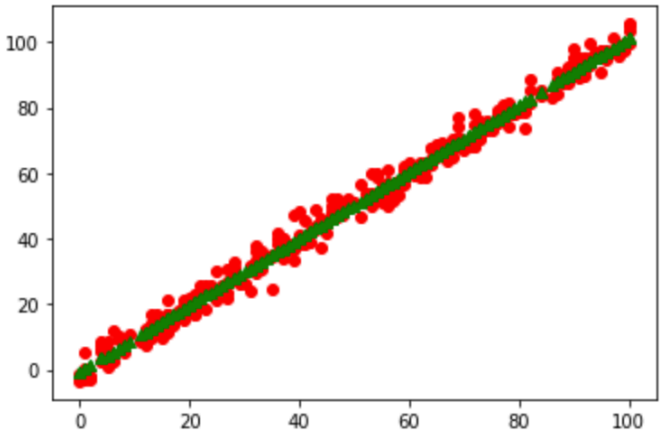

```
alpha = 0.0000001
iterations = 1000
x_2 = x**2
X_b_3 = np.c_[np.ones((len(x),1)),x,x_2]
theta_ = np.random.randn(3,1)
theta_ , cost = grad_desc_2 (X_b_3 , y , theta_ ,alpha , iterations)
print("theta0 : ",theta_[0][0],sep=' ')
print("theta1 : ",theta_[1][0],sep=' ')
print("theta2 : ",theta_[2][0],sep=' ')
print("cost : ",cost,sep=' ')
```

بخش ۶) به روش normalization ، ضرایب معادله را محاسبه میکنیم و با نتایج به دست آمده از روش gradient descent مقایسه میکنیم . نتایج کاملا مشابه هستند . روابط روش نرمالیزیشن به صورت زیر است :

$$\theta_{(n+1)\times 1} = (X^T X)^{-1} X^T Y$$

```
X = test.iloc[:, 0].values
Y = test.iloc[:, 1].values
A = (len(X)*((np.dot(X,Y)/len(X))- (np.mean(X))*(np.mean(Y))))/((len(X)-1)*(np.var(X)));
B = np.mean(Y)-A*np.mean(X);
MSE=sum((Y-(A*X+B))**2)/len(X);
print('A : ',A)
print('B : ',B)
print(MSE)
plt.plot(X, Y ,'ro',X,A*X+B , 'g^' )
```

A : 1.0177277810563725
B : -0.6346096427980967
9.173607326182905



در این روش ، چون باید ماتریس معکوس حساب کنیم ، در شرایطی که $(X^T X)$ ماتریسی سینگولار باشد ، یعنی دارای دترمینان صفر باشد ، دیگر نمیتوان به جواب رسید و باید از روش های دیگر مانند gradient descent استفاده کرد . البته روشی برای اطمینان از عدم سیگولاریتی ماتریس $(X^T X)$ وجود دارد که به این صورت است که به این ماتریس یه جمله به صورت زیر اضافه میشود و ماتریس دیگر دارای دترمینان صفر نخواهد بود ؛

$$\theta_{(n+1)\times 1} = (X^T X + \lambda \Phi)^{-1} X^T Y$$

ماتریس فی مانند ماتریس همانی است فقط : $\phi_{1,1} = 0$.

از مزایای این روش این است که دیگر نیازی به اسکیل کردن و نرمال کردن داده ها نداریم و دیگر نیازی به انتخاب alpha و تکرار زیاد نیست . اما در صورتی که تعداد داده ها زیاد باشد ، محاسبات ماتریس معکوس به شدت زیاد میشود و بهتر است برای داده های بزرگ از gradient descent استفاده کنیم .

سوال دوم

الگوریتم جست و جوی خط :

به الگوریتم هایی که به دنبال یافتن مینیمم یک تابع غیرخطی با انتخاب یک بردار جهت معقول و زمانی که به طول تکرارشونده با گام های مناسبی باشد ، الگوریتم های جست و جوی خط میگویند .

به عنوان مثال برای یافتن مینیمم تابع f(x) ، یه مقدار اولیه ی دلخواه x_k انتخاب میکنیم و به صورت زیر آن را افزایش میدهیم تا به مینیمم تابع برسیم : $x_{k+1} = x_k + \alpha_k p_k$

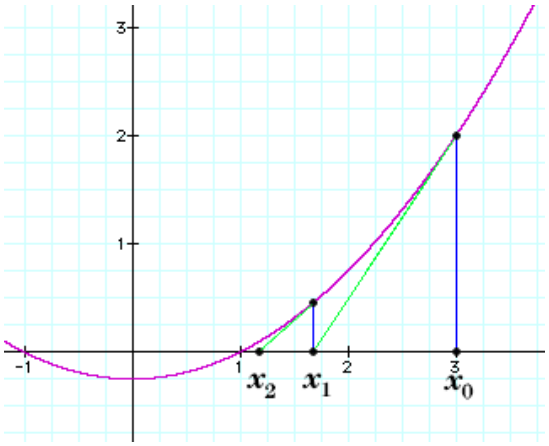
که در آن alpha یک مقدار مثبت است که اندازه ی گام نامیده میشود و بردار p هم جهت گام را مشخص میکند .

روش نیوتن :

روش نیوتن برای یافتن رشه های یک تابع یا برای یافتن مینیمم یک تابع بر پایه ی الگوریتم جست و جوی خط است که گام های آن به صورت زیر است :

$$\theta := \theta - \frac{f(\theta)}{f'(\theta)}$$

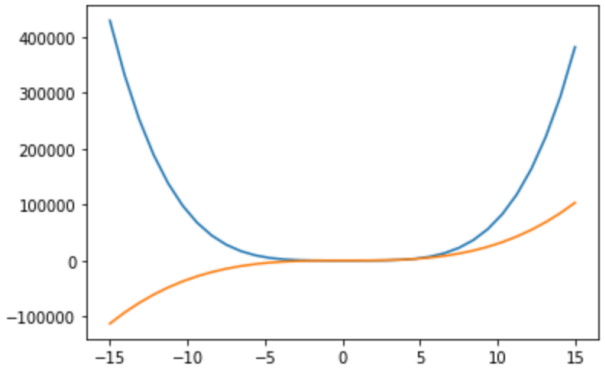
مانند روش جست و جوی خط ، یک مقدار اولیه به به theta میدهیم ، سپس در هر تکرار ، عبارت $\frac{f(\theta)}{f'(\theta)}$ را از آن کم میکنیم . تعبیر هندسی آن در شکل زیر روشن میشود به این صورت که عبارت $\frac{f(\theta)}{f'(\theta)}$ ریشه را در هر مرحله در راستای افقی به توجه به مقدار مشتق تابع در نقطه ی آغازین ، به ریشه نزدیک تر میکند .



توضیح کد :

```
f = lambda x: x**2 -7*x**3 +8*x**4 - 12
f_prim = lambda x: 2*x - 21*x**2 + 32*x**3
f_zegond = lambda x: 2 - -42*x + 96*x**2
X =  np.linspace(-15,15,33)
plt.plot(X,f(X))
plt.plot(X,f_prim(X))
```

در این قسمت ابتدا با دستور lambda توابع $f''(x)$ $f(x)$ $f'(x)$ را تعریف میکنیم و توابع. $f(x)$ $f'(x)$ را رسم میکنیم تا بدانیم مقدار اولیه را در کدام محدوده انتخاب کنیم :



با توجه به نمودار و صفر شدن شیب نمودار دربازه ی (-5,5) ، در میابیم مینیمم تابع در همین بازه قرار دارد . پس شرط اولیه را در همین محدوده انتخاب میکنیم :

```
def newton(f,Df,x0,epsilon,iteration):
    values =[]
    values.append(x0)
    xn = x0
    for n in range(0,iteration):
        if abs(f(xn)) < epsilon:
            print('Found solution after',n,'iterations.')
            return xn , values
        xn = xn - f(xn)/Df(xn)
        values.append(xn)
    print('No solution found.')
    return None
```

در این سوال چون به دنبال مینیمم تابع f هستیم ۷ یعنی به دنبال ریشه ی تابع مشتق f ، پس ورودی های تابع newton. باید مشتق اول و مشتق دوم تابع f باشند .

شرط توقف این الگوریتم ، مقدار متغیر epsilon است که با توجه به دقتی که نیاز داریم آن را تعیین میکنیم . در اینجا ما epsilon را 10^{-5} فرض میکنیم .در این تابع در هر مرحله xn را در لیست values نگه میداریم و در انتها در صورت

برقراری شرط $f(xn) < epsilon$ ، xn و لیست values را برمیگردانیم . در صورتی هم که پس از اتمام دوره های تکرار ، به شرط $f(xn) < epsilon$ دست نیافیتم ، پرینت میکنیم که به جواب نرسیدیم .

در انتها تفاضل هردو درایه ی مجاور لیست values را در یک نمودار میله ای نمایش میدهیم . همانطور که مشاهده میکنید ، این نمودار روندی نزولی دارد که نشان میدهد در هر مرحله تفاضل بین دو x به دست آمده کاهش مییابد و این نشان دهنده ی این موضوع است که در هر مرحله اندازه ی مشتق کاهش مییابد و در نتیجه به مینیم تابع f نزدیک تر میشویم .

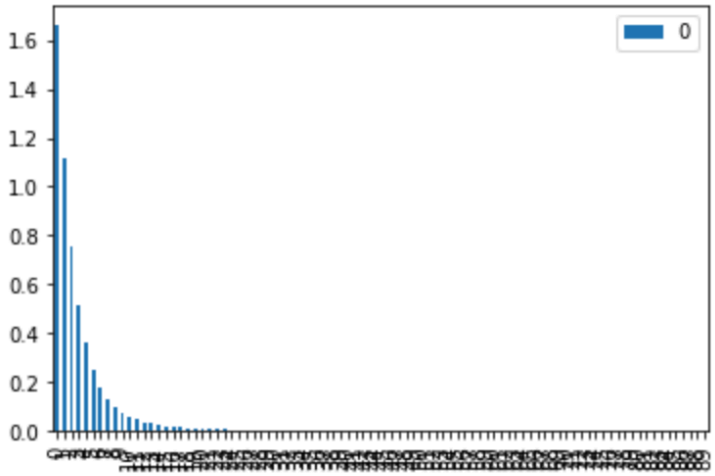
```
f = lambda x: x**2 -7*x**3 +8*x**4 - 12
f_prim = lambda x: 2*x - 21*x**2 + 32*x**3
f_zegond = lambda x: 2 - -42*x + 96*x**2
ans_1,values1 = newton(f_prim,f_zegond,6,1e-5,1000)
ans_2,values2 = newton(f_prim,f_zegond,-8,1e-5,1000)
print("The minimum is at the point x =  ", ans_1 , " and the value of the functions is equal to f(x) = ",f(ans_1))
diff = []
for i in range(len(values1)-1):
    diff.append(values1[i]-values1[i+1])
df = pd.DataFrame(diff)
df.plot.bar()
```

Found solution after 90 iterations.

Found solution after 91 iterations.

The minimum is at the point x = 0.5406492645999833 and the value of the functions is equal to f(x) = -12.13040506098638

در این سوال برای اطمینان از یکتا بودن مقدار مینیمم به دست آمده ،یک شرط اولیه در x=6 و شرطی دیگر در x=-8 داده است . هردو جواب به مقدار یکسانی دست یافتند ، پس با توجه به نمودار و جواب مسئله ، این مسئله فقط یه مینیمم سراسری دارد .



سوال سوم

دیتاست این مسئله دارای ۸ ستون ویژگی است و دارای دو ستون هدف . هدف این سوال پیدا کردن رگرسیون خطی چندمتغیره برای این دیتاست است . طبق گفته ی صورت سوال ابتدا داده هارا نرمال میکنیم .

```
data1 = (data-data.min())/(data.max() - data.min());
```

قسمت اول)

تابع gradient descent استفاده شده در سوال را میتوانیم مجددا در این قسمت استفاده کنیم :

```
def grad_desc_3(x,y,theta,alpha,iterations):
    m=len(y)

    # this loop updates the magnitude of theta vector
    for i in range(iterations):
        #we start prediction with initial condition
        prediction = np.dot(x,theta)
        theta = theta - (1/m)*alpha*(np.dot(x.T,(prediction - y)))
    cost = call_cost(theta,x,y)
    return theta , cost
```

سپس ۸ ویژگی اول را در بردار X ریخته و دو ستون هدف را بردار Y سپس تابع grad_desc3 رابرای این بردار ها صدا میزنیم . نتایج به صورت زیر است :

```
alpha = 0.0000001
iterations = 100000
theta_3 = np.random.randn(9,1)

training_data = data.iloc[:600,:]
test_data = data.iloc[601:,:]

X = pd.DataFrame(training_data , columns = ['X1','X2','X3','X4','X5','X6','X7','X8'])
Y1 = pd.DataFrame(training_data , columns = ['Y1'])
Y2 = pd.DataFrame(training_data , columns = ['Y2'] )
X = np.c_[np.ones((len(X),1)),X]
print(Y1.iloc[0])
```

```
theta_Y1 , cost_Y1 = grad_desc_3 (X , Y1 , theta_3 ,alpha , iterations)
theta_Y2 , cost_Y2 = grad_desc_3 (X , Y2 , theta_3 ,alpha , iterations)


print("theta_Y1 : ",theta_Y1,sep=' ')
print("theta_Y2 : ",theta_Y2,sep=' ')
print("cost_Y1 : ",cost_Y1,sep=' ')
print("cost_Y2 : ",cost_Y2,sep=' ')
```

theta_Y1 : cost_Y1 = 24.66528

[[0.49339005]
[-0.92596492]
[0.25038764]
[-0.09749899]
[-0.6237677]
[-0.42616539]
[-0.1175105]
[0.50806881]
[-1.20597001]]

theta_Y2 : cost_Y2 = 25.610919

[[0.49381615]
[-0.92531399]
[0.25506045]
[-0.10233202]
[-0.61901478]
[-0.4207984]
[-0.11424188]
[0.50736189]
[-1.21015633]]

سپس داده های تست را به تابع call_cost میدهیم :

```
X_test = pd.DataFrame(test_data, columns = ['X1','X2','X3','X4','X5','X6','X7','X8'])
Y1_test = pd.DataFrame(test_data , columns = ['Y1'])
Y2_test = pd.DataFrame(test_data , columns = ['Y2'] )
X_test = np.c_[np.ones((len(X_),1)),X_test]
```

```
cost_Y1_test = call_cost(theta_Y1,X_test,Y1_test)
cost_Y2_test = call_cost(theta_Y2,X_test,Y2_test)
print("cost_Y1_test : ",cost_Y1_test)
print("cost_Y2_test : ",cost_Y2_test)
```

cost_Y1_test : Y1 28.69356
cost_Y2_test : Y2 28.448248

همانطور که میبیند خطای داده های تست بیشتر از خطای داده های training است که منطقی است ،چون theta های محاسبه شده در جهت کم کردن خطای داده های training طراحی شده اند . پس افزایش cost داده های تست منطقی به نر میرسد .

قسمت دوم)

اعتبار سنجی یک تکنیک بسیار مفید برای ارزیابی کارایی مدل های یادگیری ماشین است.این روش کمک می کند متوجه شویم به چه صورت مدل یادگیری ماشین که ایجاد کرده ایم ، به یک مجموعه داده مستقل تعمیم داده می شود. زمانی که یک مسائله یادگیری ماشین به ما داده می شود ، معمولا با دو مجموعه داده سر کار داریم که داده های شناخته شده (training data set) و داده های ناشناخته (test data set) می باشند.با استفاده از ارزیابی ضربدری(Cross Validation) می توانیم مدل یادگیری ماشین خود را در فاز آموزش (training) برای چک نمودن کارایی و بدست آوردن یک ایده و نظر از چگونگی تعمیم مدل یادگیری ماشین خود به داده های مستقل تست نماییم (testing).در واقع ارزیابی و صحت مدل یادگیری ماشین خود را در همان مرحله آموزشی انجام می دهیم. یعنی به این صورت که کل داده های آموزشی را به n بخش تقسیم میکنیم و در هر بخش مثلا ۹۰ درصد از داده های آموزشی را صرف training میکنیم و ۱۰ درصد از داده ها را تست میکنیم تا در حین develop مدل ، ایده ای از نحوه ی عملکرد آن به دست آوریم . سپس در مرحله ی بعد از ضرایبی به دست آمده از مرحله ی قبل استفاده میکنیم .

قسمت سوم)

```
cost_train_Y1 = []
cost_train_Y2 = []
cost_eval_Y1 = []
cost_eval_Y2 = []
theta_3 = np.random.randn(9,1)
for i in range(12):
    print(i)
    X_train = X[50*i: 50*(i+1) -5 ,:]
    Y1_train = Y1.iloc[50*i: 50*(i+1) -5 ,:]
    Y2_train = Y2.iloc[50*i: 50*(i+1) -5 ,:]
    theta_Y1 , cost_Y1 = grad_desc_3 (X_train , Y1_train , theta_3 ,alpha , iterations)
    theta_Y2 , cost_Y2 = grad_desc_3 (X_train , Y2_train , theta_3 ,alpha , iterations)
    cost_train_Y1.append(cost_Y1)
    cost_train_Y2.append(cost_Y2)

    X_eval = X[50*(i+1)-5: 50*(i+1),:]
    Y1_eval = Y1[50*(i+1)-5: 50*(i+1),:]
    Y2_eval = Y2[50*(i+1)-5: 50*(i+1),:]
    cost_eval_Y1.append(call_cost(theta_Y1,X_eval,Y1_eval))
    cost_eval_Y2.append(call_cost(theta_Y2,X_eval,Y2_eval))
t = list(range(12))
plt.subplot(1, 2, 1)
plt.plot(t, cost_train_Y1 ,'ro',t,cost_eval_Y1 , 'g^' )
plt.subplot(1, 2, 2)
plt.plot(t, cost_train_Y2 ,'ro',t,cost_eval_Y2 , 'g^' )
```

قسمت چهارم)

Feature selection

روش های انتخاب ویژگی :

۱ . Filte Methods : روش های فیلتری ویژگی های حیاتی را با استفاده از آمار تک متغیری انتخاب میکند . ای روش سرعت بیشتر و بار محاسباتی کمتری دارد . برای داده های با ابعاد زیاد ، این روش هزینه ی کمتری دارد . روش های Fisher's Score ، square Test-Chi و Information Gain از زیرمجموعه های روش filtering هستند .

۲ . Wrapper Methods: wrapper ها نیازمند روش هایی برای جست و جوی فضای همه ی زیرمجموعه های ممکن ، برای ارزیابی کیفیت آن ها با یادگیری و ارزیابی یک طبقه بند با آن زیرمجموعه ی ویژگی است . فرآیند انتخاب ویژگی بر اساس الگوریتم خاصی برای یادگیری ماشین است که سعی داریم برای یک مجموعه ی داده مشخص مدلی پیدا کنیم .

روش های زیر مجموعه ی wrapper :

- Forward feature selection
- Backward feature elimination
- Exhaustive feature selection
- Recursive feature elimination

۳ . Embedded Methods : این روش مزایای های هردو روش بالا را داراست ، با دربرداشتن تاثیر ویژگی ها برهم و همچنین بار محاسباتی معقول تر . این روش ها به گونه ای هستند که از هر تکرار در فرآیند یادگیری مراقبت میکنند و ویژگی هایی که بیشترین کمک را به آموزش دارند ، به دقت استخراج میکنند . روش های زیر از این متد استفاده میکنند .

LASSO regularization , Random forest importance ,

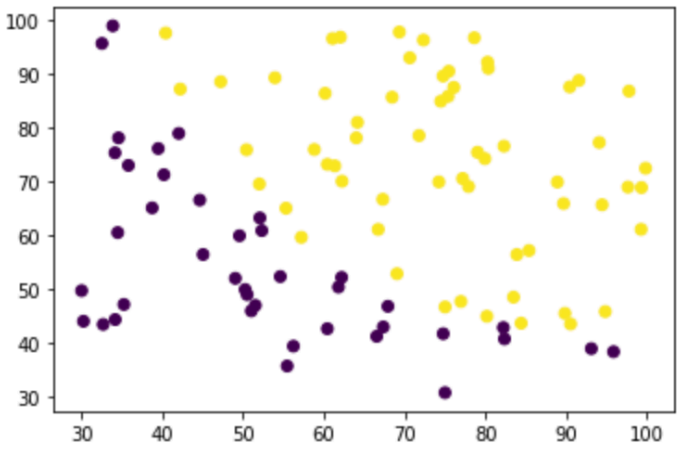
سوال چهارم

ابتدا داده های مسئله را با دستور real_exel و نصب کتابخانه ی مورد نظر لود میکنیم .

قسمت اول)

نمودار مربوط به این سوال را بر روی ۲ محور میزان گلوکز و اکسیژن خون رسم میکنیم و نتایج بیمار بودن و نبودن را با دو رنگ متفاوت نشان میدهیم :

```
import scipy.io
Data_ = scipy.io.loadmat('data_logistic.mat')
Data = Data_['logistic_data']
plt.scatter(Data[:,0],Data[:,1],c=Data[:,2])
plt.show()
```



همان طور که در شکل میبینید ، داده هارا میتوان به صورت تقریبی با یک خط از هم جدا کرد .

قسمت دوم)

برای مسائل لاجیستیک از تابع سیگموئید استفاده میکنیم زیرا این تابع فقط مقادیر بین ۰ و ۱ را میگیرد و با تعیین یک حد آستانه ، میتوان داده هارا به دو دسته تقسیم کرد در حالیکه رگرسیون خطی محدوده ی وسیعی از داده هارا شامل میشود و تقسیم بندی مناسبی برای یک متغیر گسسته ی لاجیستیک نیست .

تابع سیگموئید :

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

با صفر شدن قسمت نمایی ، تابع h به نصف مقدار خود یعنی ۱/۲ میرسد . تابع H نشان دهنده ی احتمال بیماری فرد است

پس با استفاده از روش گرادیان کاهشی برای رگرسیون لاجیستیک ، ضرایب theta را میابیم و با صفر قرار دادن معادله ی آن ، به خط مورد نظر برای جدا کردن افراد بیمار از غیر بیمار میپردازیم :

برای محاسبه ی تابع gradient descent رگرسیون لاجیستیک ، از روش میانگین خطای مربعات استفاده میکنیم و به جای $h(\theta)$ ، از رابطه ی تابع سیگموئید استفاده میکنیم :

$$\theta_j := \theta_j - \frac{\alpha}{m}\Sigma(h_\theta(x^{(i)}) - y^{(i)})X^{(i)}$$

همچنین :

$$h(\theta) > th \quad \rightarrow y = 1 \quad \rightarrow \theta^T X > 0$$

$$h(\theta) < th \quad \rightarrow y = 0 \quad \rightarrow \theta^T X < 0$$

```
def grad_desc4(x,y,theta,alpha,iterations):
    m=len(y)
    for i in range(iterations):
        #we start prediction with initial condition
        prediction = 1/(1 + np.exp(-1*(np.dot(x,theta))))
        theta = theta - (1/m)*alpha*(np.dot(x.T,(prediction - y)))
    cost = cost4(theta,x,y)
    return theta , cost
```

برای محاسبه ی تابع cost، چون روش میانگین خطای مربعات ، تابع محدبی دیگر نیست یعنی ممکن است چند مینیمم محلی داشته باشد و در این صورت دیگر به مینیمم سراسری همگرا نمیشود ، از تابع زیر استفاده میکنیم که همگرا ست :

$$j(\theta) = \frac{1}{m}\Sigma[y^i log(h + \theta(x^i) + (1 - y^i) log(1 - h_\theta(x^i)]$$

```
def cost4(theta,x,y):
    m=len(x)
    o = x.dot(theta)
    prediction_ = 1/(1 + np.exp(-1*(np.dot(x,theta))))
    cost = (-1/m) * np.sum((np.dot(y.T , np.log(prediction_)) + np.dot((1-y.T) , np.log(1-prediction_))))
    return cost
```

```
alpha = 0.00001
iterations = 100000
theta_ = np.random.randn(3,1)  # random initial condition
# this line concatinate 1 to a x vector to make theta vector and x vector the same size
X_b_ = np.c_[np.ones((len(Data[:,1]),1)),Data[:,0],Data[:,1]]
y = np.c_[Data[:,2]]
thetaa , costt = grad_desc4 (X_b_ , y , theta_ ,alpha , iterations)
print("thetaa",thetaa)
print("costt : ",costt,sep=' ')
```

نتایج به صورت زیر است :

thetaa :
[[0.39620067]
[0.00768626]
[-0.00262194]]
costt : 0.6581603531304808

قسمت سوم)

در این بخش از با استفاده از L2_norm تابع هزینه را تغییر میدهیم . بدین صورت که به تابع هزینه ، یک ترم دیگر از مجموع توان های ۲ ضرایب خط رگرسیون ، اضافه میشود . این روش جزو روش های پایداری برای جلوگیری از overfiitting است . به این شکل که اگر یک ضریب به شدت افزایش افزایش یابد ، تابع هزینه هم افزایش میابد و با مینیمم کردن تابع هزینه ، جلوی این افزایش گرفتع خواهد شد . به علاوه به کمک ضریب landa ، رشد این ضرایب کنترل میشود :

$$j(\theta) = \frac{1}{m}\Sigma(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\Sigma\theta_j^2$$

با مشتق گرفتن از رابطه ی بالا برحسب θ_j و ساده سازی ، برای آپدیت کردن تتا ، رابطه ی زیر به دست می آید :

$$\theta_j := \theta_j(1 - \frac{\alpha\lambda}{m}) - \frac{\alpha}{m}\Sigma(h_\theta(x^{(i)}) - y^{(i)})X^{(i)}$$

```
def L2_norm(x,y,theta,alpha,landa,iterations):
    m=len(y)
    for i in range(iterations):
        #we start prediction with initial condition
        prediction = 1/(1 + np.exp(-1*(np.dot(x,theta))))
        theta = theta*(1 - (alpha*landa/m)) - (1/m)*alpha*(np.dot(x.T,(prediction - y)))
    cost = cost_L2(theta,x,y,landa)
    return theta , cost
```

```
def cost_L2(theta,x,y,landa):
    m=len(x)
    o = x.dot(theta)
    prediction_ = 1/(1 + np.exp(-1*(np.dot(x,theta))))
    cost = (-1/m) * np.sum((np.dot(y.T , np.log(prediction_))
                            + np.dot((1-y.T) , np.log(1-prediction_)))) + landa * np.sum(theta ** 2)
    return cost
```

```
alpha = 0.00001
iterations = 100000
landa = 0.1
# theta_ = np.random.randn(3,1)  # random initial condition
# this line concatinate 1 to a x vector to make theta vector and x vector the same size
X_b_ = np.c_[np.ones((len(Data[:,1]),1)),Data[:,0],Data[:,1]]
y = np.c_[Data[:,2]]
thetaa , costt = L2_norm (X_b_ , y , theta_ ,alpha, landa, iterations)
print("thetaa",thetaa)
print("costt : ",costt,sep=' ')
```

در نهایت ضرایب و مقدار cost به صورت زیر حاصل میشود :

thetaa :
[0.39576991]]
[0.00768916]
[[-0.00261854]
costt :0.6737987844108141

نتیجه :

همان طور که مشاهده میکنید تغییری در اندازه ی ضرایب ایجاد نشد ولی مقدار cost افزایش پیدا میکند .

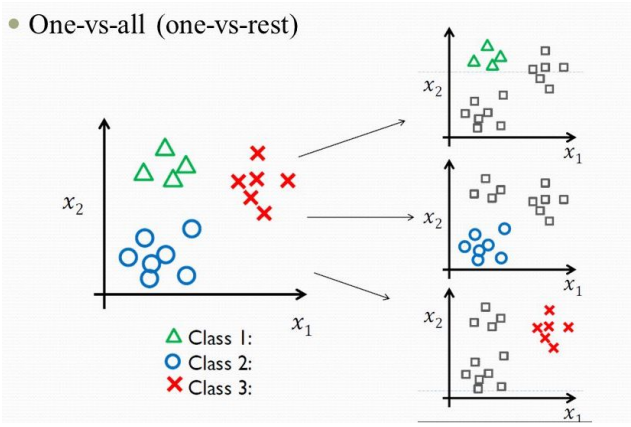
قسمت چهارم) بهترین دقت برای روش اول است زیرا cost کمتری داشت .

قسمت پنجم)

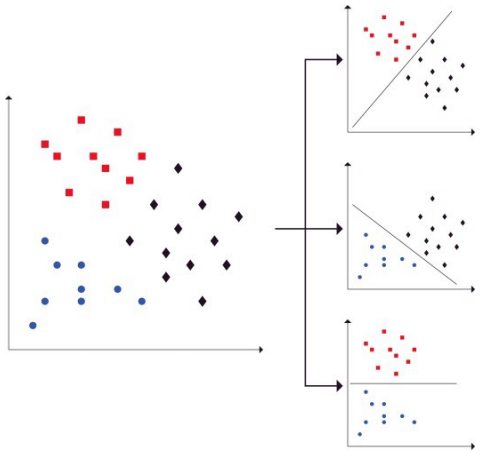
دو روش متداول برای طبقه بندی چند کلاسی وجود دارد :

۱ . **One-vs-all** : در این روش مرحله به مرحله به دنبال یافتن خطی هستیم که یک کلاس را از بقیه کلاس ها جدا کند و آن را h_i مینامیم . سپس به ازای xtest ، ماگزیمم مقدار بین توابع h را به عنوان خروجی در نظر میگیریم .

$$h_i(\theta) = p(y = 1 \mid x, \theta) \qquad \rightarrow \qquad y_{test} = max h_i(\theta)(x_{test})$$



۲ . **One-vs-one** : در این روش دو به دو برای هردو کلاس به دنبال خط جداکننده هستیم ، یعنی انتخاب ۲ از n حالت که بسیار زیاد میشود به همین دلیل این روش رایج نیست . برای یک xtest هم مانند قبل بین تمام hi ها ماکزیمم حساب میشود :



سوال پنجم

تعبیر احتمالاتی رگرسیون خطی :

اگر به جای فرض deterministic بودن متغیرها ، به متغیر y یک نویز تصادفی اضافه شود یا به علت خطاهای ناشی از اثرات مدل نشده ، ترمی تصادفی به آن اضافه شود ،دیگر متغیرهای ما معین نیستند :

$$y^i = \theta^T x^i + \epsilon^i$$

اگر ϵ را ترمی مستقل با توزیع گوسی فرض کنیم ، در این صورت توزیع احتمالاتی متغیر y به صورت زیر خواهد بود :

$$P(y^i \mid x^i, \theta) = \frac{1}{\sqrt{(2\pi\sigma)}} exp(\frac{y^i - \theta^T x^i}{2\sigma^2})$$

میتوان اینگونه بیان کرد که هدف ما ، افزایش تابع احتمالاتی فوق است که معادل است با کاهش دادن تابع هزینه رگرسیون خطی .اگر تابع $l(\theta)$ را به صورت لگاریتم توزیع احتمالاتی بالا تعریف کنیم ، و با ماکزیمم کردن آن به تنای بهینه برسیم :

$$P_\theta(y \mid x) = \Pi P(y^i \mid x^i, \theta) \quad \rightarrow \quad l(\theta) = log(P(\theta)) \quad \rightarrow \quad max l(\theta) = min \frac{1}{2} \Sigma (y^i - \theta^T x^i)^2$$

همان طور که میبینید ماگزیمم کردن تابع احتملاتی توزیع گوسی ، معادل مینیمم کردن تابع هزینه ی رگرسیون خطی است .

تعبیر احتمالاتی رگرسیون لاجیستیک :

میدانیم : $y^i \in (0,1)$ $\qquad h_\theta = P(y = 1 \mid x, \theta) \quad and \quad 1 - h_\theta = P(y = 0 \mid x, \theta)$

میتونیم تابع احتمالاتی لاجیستیک را به صورت زیر تعریف کنیم که عینا مطابق با تعریف بالاست :

$$P(y \mid x, \theta) = h_\theta^y (1 - h_\theta)^{1-y}$$

مانند آنچه در قسمت قبل انجام دادیم ، ابتدا از عبارت بالا log میگیریم و آن را max میکنیم . در نتیجه :

$$L(\theta) = \Pi P(y^i \mid x^i, \theta) = \Pi h_\theta^{y^i} (1 - h_\theta)^{1-y^i}$$

$$l(\theta) = log(P(\theta)) \quad \rightarrow \quad max \quad l(\theta) = min \quad (\Sigma y^i h_\theta + (1 - y^i)(1 - h_\theta))$$

همانطور که میبینید ، ماکزیمم کردن توزیع احتمالی برنولی ، منجر به مینیمم کردن تابع هزینه ی لاجیستیک میشود .

مدل خطی تعمیم یافته رگرسیون پواسون:

همانطور که میدانید توزیع خانواده ی نمایی به صورت زیر است : $P(y;\eta) = b(y)exp(\eta^T T(y) - a(y))$

توزیع احتمالاتی پوآسون به شکل زیر است : $P(y;\eta) = \frac{\lambda^y e^{-\lambda}}{y!}$

مانند آنچه برای توزیع برنولی و گوسی انجان دادیم ، همزمان از تابع احتمالاتی بالا exp و log میگیریم . با توجه به این که :$e^{log(x)} = x$

در نتیجه خواهیم داشت :

$$exp(log\frac{\lambda^y e^{-\lambda}}{y!}) = exp(ylog^{\lambda} - \lambda - log^{y!}) = exp(ylog^{\lambda}) \times exp(-\lambda) \times \frac{1}{y!}$$

در نهایت : $exp(ylog^{\lambda} - \lambda)\frac{1}{y!}$

که میبینیم :

$\eta = log^{\lambda} \quad \rightarrow \quad \lambda = e^{\eta}$

$b(y) = \frac{1}{y!}$

$T(y) = y$

$a(y) = \lambda = e^{\eta}$

References :

1 . https://algorithmia.com/blog/feature-selection-in-machine-learning#:~:text=Feature%20selection%20in%20machine%20learning%20refers%20to%20the%20process%20of,and%20improve%20accuracy%2C%20or%20both.

2 . https://machinelearningmastery.com/feature-selection-with-real-and-categorical-data/

3 .https://towardsdatascience.com/validating-your-machine-learning-model-25b4c8643fb7

4 . https://www.analyticsvidhya.com/blog/2020/10/feature-selection-techniques-in-machine-learning/