Lineer Regression with Multiple Features

Şuana kadar lineer regresyonu tek özellik üzerinden anlattık. Ancak günlelik hayattaki problemlerde birden fazla özellik ile karşı karşıya kalıyoruz. Örneğin ev fiyatı tahmininde sadece m^2 boyutu değil, evin yaşı, yatak oda sayısı, toplam oda sayısı gibi özellikler etki etmektedir.

örn Ev fiyat tahmini örneği.

size (feet²) x_1	Number of bedrooms x_2	Number of floors x_3	Age of home (years) x_4	Price y
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
...	...	...	...	...

Notation

n = özellik sayısı

$x^{(i)}$ = eğitim (training) örnekleri girişi

$x_j^{(i)}$ = eğitim örneğinin j. özellik değeri

$n=4 \quad x^{(2)} = \begin{bmatrix} 1416 \\ 3 \\ 2 \\ 40 \end{bmatrix}$

$x_3^{(2)} = 2$

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$$ } Genel Lineer Regresyon Formülü

Not: x_0, 1'e eşit olduğu için özellik aranmaz ancak θ_0 denklemde vardır. Bu yüzden özellik vektörümüz ve θ vektörümüz $(n+1)$ boyutundadır.

$$h\theta(x) = [\theta_0 \ \theta_1 \ \dots \ \theta_n] \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} = \theta^T x$$

Cost Function Vectorization

$$J(\theta) = \frac{1}{2m} (X^*\theta - \vec{y})^T * (X^*\theta - \vec{y})$$

Gradient Descent

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$ } Cost Function (square error)

$(n \geq 1)$

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) x_2^{(i)}$$

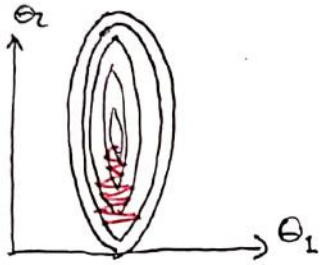
...

Gradient Descent

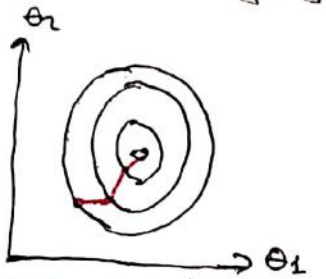
$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) x_j^{(i)}$$

Gradient Descent - Feature Scale

Girdi değerlerimizin her birini kabaca aynı aralıkta tutarak degrade inişi hızlandırabiliriz. Örneğin evin metrekaresi 0-2000, oda sayısı 0-5 arasında olan iki özellik arasında oluşacak kontur grafiği ince ve uzun bir görüntüm elde edecek ve gradient descent algoritması minimuma ulaşmak için çok fazla zaman harcayacak. Bir özellikleri ölçeklendirerek daire bir görüntü elde ederek daha hızlı bir gradyan iniş elde ederiz.



Feature Scale edilmemiş



Feature Scale edilmiş (kontür grafik)

Giriş değişkenlerinin ideal aralıkları;

$$-1 \leq x_{(i)} \leq 1 \quad \text{veya} \quad -0.5 \leq x_{(i)} \leq 0.5$$

Bu aralıklara getirebilmek için iki yöntem var, feature scale ve mean normalization (ortalama normalleştirme).

Feature scale; giriş değerlerinin, giriş değişkeninin aralığına (yani maksimum değer eksi minimum değer) bölünmesi ile $-1 \leq x_i \leq 1$ aralığına indirgenmesidir.

$$x_i := \frac{x_i}{S_{max} - S_{min}}$$

Mean normalization; giriş değerini, giriş değerlerinin ortalamasından çıkarıp girişlerin değer aralığına (maks-min) veya standart sapmasına bölünme işlemidir.

$$x_i := \frac{x_i - \mu_i}{S_i}$$

μ_i = ortalama

S_i = değer aralığı veya standart sapma

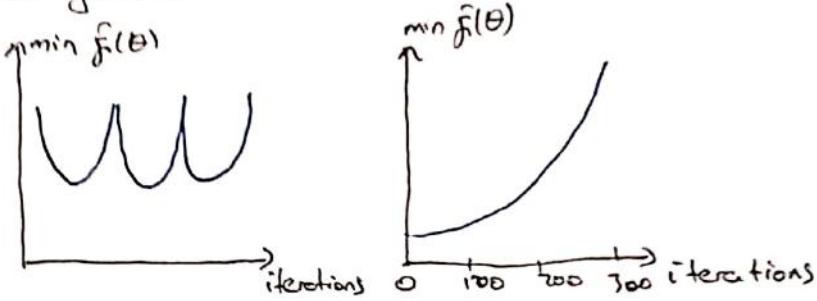
örn: x_i 100 ila 2000 aralığında ve ortalama 1000 değeri ile konut fiyatlarını temsil eder;

$$x_i := \frac{x_i - 1000}{1900}$$

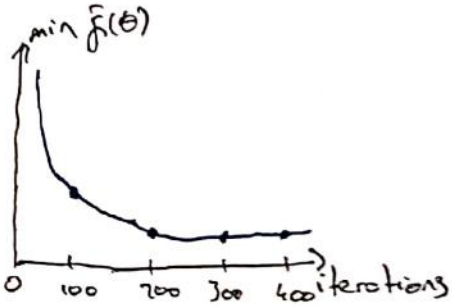
Gradient Descent - Learning Rate

Gradient descentin doğru çalıştığını gözlemlemek için maliyet fonksiyonu ($J(\theta)$) ve değrade iniş yineleme sayısının grafiğini çizin. Grafik artan yöndeyse α'yı azalmanız gerekir.

Yandaki tabloda gibi bir grafik elde ediyorsanız α'yı çok daha küçük seçmelisiniz.



α'yı çok küçük seçerseniz yavaş yakınsar, α'yı çok büyük seçerseniz yinelemede azalma göstermeyebilir ve yakınsama olmayabilir.



Otomatik yakınsama (convergence) testi: Bir yinelemede $J(\theta)$ ϵ'den daha az bir değer alırsa yakınsama olduğunu bildirin. ϵ 10^{-3} gibi küçük bir eşik değeri seçilmelidir.

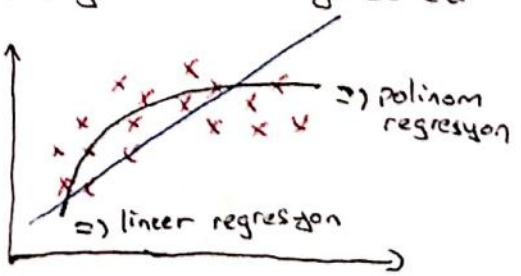
0,001 $\xrightarrow{\times 3}$ 0,003 $\xrightarrow{\times 3}$ 0,009 $\xrightarrow{\times 3}$ 0,03 $\xrightarrow{\times 3}$ 0,1 $\xrightarrow{\times 3}$ 0,3

Features ve Polinom Regresyon

Bize verilen özelliklerden kullanabileceğimiz yeni bir özellik çıkarıp modelimizi bu feature ile eğitebiliriz. Örneğin bir evin genişliğini ve enini özellik olarak verip fiyat tahmini isteyebilir. Bizde verilen özelliklerden m^2 özelliği çıkararak ev tahminini bu özellik üzerinden yaparız.

Verilere iyi uymuyorsa hipotez fonksiyonumuzun doğrusal olması gerekmez. Hipotez fonksiyonumuzun davranışını veya eğrisini; 2. dereceden, kübik veya karekök fonksiyonu yaparak değiştirebiliriz.

$$h_\theta(x) = \theta_0 + \theta_1 x_1$$



2. derece $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2$ $\quad x_2 = x_1^2$

kübik $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^3$ $\quad x_2 = x_1^2$, $x_3 = x_1^3$

karekök $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 \sqrt{x_1}$ $\quad x_2 = \sqrt{x_1}$

Yandaki gibi hipotezimizi diğer fonksiyonlara çevirebiliriz.

Not: Feature scale'in önem arz ettiği ortaya çıkıyor. Çünkü değerin karesi ve küpü alınarak değeri büyüyor.

Normal Equation

Θ parametrelerini bulmilmek için gradient descent algoritması ile yinelemeli bir şekilde buluyorduk. Ancak normal equation ile Θ parametrelerini direk bulabiliyoruz. Yineleme yerine tek seferde Θ için en uygun değeri bulabilirim

Θ parametresini minimize edebilmek için cost function'ın türevini alıp 0'a eşitlemek gerekir. Normal denklem yönteminde de bunu yapıcaz.

$$\Theta = (X^T X)^{-1} X^T y$$

Burada X design matrix (tasarım matris) oluyor. Özellik vektör değerlerinin transpozu alınarak elde edilir.

x_0	Size x_1	Number of Bedrooms x_2	Number of floors x_3	Age of home x_4	Price
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	317
1	852	2	1	36	178

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix}$$

$$X = \begin{bmatrix} — & x_1^T & — \\ — & x^2{}^T & — \\ & \vdots & \\ — & x^{m T} & — \end{bmatrix} \quad m \times (n+1)$$

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \quad m \times 1$$

$$y^T = \begin{bmatrix} 460 & 232 & 315 & 178 \end{bmatrix}$$

Not Non-invertibility (Tersine çevrilemez) $(X^T X)^{-1}$

- Gereksiz özellik olması nedenlerinden biri. Varsa sil.
- $m \leq n$ ise nedenlerden biridir. Özellikleri azaltmalıyım veya regularization kullanılmalı.

Not Normal equation yönteminde feature scale yapmamıza gerek yoktur. Hatta geniş aralıklar daha iyidir.

Not Normal equation logistic regression, classification gibi algoritmalarda kullanılmaz.

Gradient Descent	Normal Equation
α seçmeniz lazım.	α seçmenize gerek yok.
Yinelemeye (iteration) ihtiyaç var.	Yinelemeye ihtiyaç yok.
$O(kn^2)$	$O(n^3)$
n büyük olsada çalışır.	n çok büyükse yavaş.

Not Pratikte n, 10.000'i aştığında gradient descent'e geçilmelidir.

Matlab / Octave

% => Yorum işareti olarak algılar.
== => Eşit mi operatörü
~= => Eşit değil mi operatörü
&& => And logic
|| => Or logic
xor() => Xor logic

pi => 3.1416 pi sayısına eşit.
disp() => yalnız değeri yazdırır.

```
> a = pi;
> a
> a = 3.1416
> disp(a)
> 3.1416
```

sprintf => ekrana yazı basar. (tt printf kullanımı

```
> disp(sprintf('2 decimal: %0.2f', a))
> 2 decimal: 3.14
```

format long => ondalık sayıları uzun gösterir.
format short => ondalık sayıları kısa gösterir.

Matris Tanımlama

```
> A = [1 2; 3 4; 5 6]
> A =  1 2
       3 4      (3x2)
       5 6
```

```
> v = 1:0.2:2 => 1'den 2'ye kadar 0.2 artışla
>1.00 1.20 1.40    satır vektör oluşturur.
 1.60 1.80 2.00
>v = 1:6
>1 2 3 4 5 6
```

ones(_,_) => 1'ler matrisi oluşturur.
zeros(_,_) => 0 matrisi oluşturur.
rand(_,_) => random 0-1 arası matris oluşturur.
randn(_,_) => Gauss rastgele dağılımına göre
eye(_) => Birim matris oluşturur.

Not: hist() => Verileri görselleştirmeye yarar.

Veri Taşıma (Moving Data)

size("matrix") => Matrisin boyutunu satır vektörü olarak verir.
size("matrix",1) => Matrisin satır sayısını verir.
size("matrix",2) => Matrisin sütun sayısını verir.

length("vektör") => Vektörün uzunluğunu verir. Matrisler için boyutundaki en büyük sayıyı döndürür.

Not: pwd komutu Octave'in bulunduğu dizini döndürür.
cd komutu girilmek istenen dizin için yazılır.
ls komutu bulunduğumuz dizindeki dosyaları döndürür.

load feature X.dat => Veri kümesini octave'a yükler.
load ('feature X.dat') => Veri kümesini stringlere (dizelere) kaydeder.

who komutu octave çalışma alanında hangi değişkenlerin olduğunu gösterir.
whos komutu değişkenlerin boyutunu, türünü, kapladığı alanı gösterir.

clear "değişken adı" => değişkeni siler. Yalnız clear hepsini siler.

Save 'dosya-adi.mat" => Programdaki tüm değişkenleri dosyaya kaydeder. Dosya isminden sonra değişken belirtirseniz onu kaydeder.
save "dosyaadi.txt" -ascii => Verileri text olarak kaydeder ve okuyabilirsiniz.

```
> A = [1 2; 3 4; 5 6]
> A(3,2);
> 6
```

```
> A(2,:)
> 3  4
```

=> ":" hepsini döndür (row/column)

```
>A(:,2)
> 2
  4
  6
```

```
> A([1 3],:)   => A'nın 1. ve 3. satırlarından değer aldık.
> 1  2
  5  6
```

Atama İşlemi Matrislerde

```
> A(:,2) = [10; 11; 12]
>  1  10
   3  11
   5  12
```

```
> A(:)   => Bize matrisi vektöre kaydeder döndürür.
> 1
  3
  5
  10
  11
  12
  100
  101
  102
```

Ekleme İşlemi Matrislerde

```
> A = [A, [100; 101; 102]]
> 1  10  100
  3  11  101
  5  12  102
```

```
> B = [11 12; 13 14; 15 16]
> C = [A B]
> 1  2  11  12
  3  4  13  14
  5  6  15  16
```

=> Concat eder ve isterseniz virgül konabilir.

```
> C = [A; B]   => Noktalı virgül aşağı atar.
>  1  2
   3  4
   5  6
   11 12
   13 14
```

Veri Hesaplama İşlemleri (Computing on Data)

A .* B => Elemansal çarpma yapar.
A * B => Matris çarpması yapar.
A .^ 2 => Elemansal olarak üs alır.
1 ./ A => Elemansal bölme.
log(v) => Her elemanın logaritmasını verir.
exp(v) => Her elemanın euler'ini verir.
abs(v) => Elemanların mutlak değerini alır.

v + ones(length(v), 1) = v+1 => Elemanların değerini 1 arttırır.

A' => A matrisinin transpozunu döndürür.
max(a) => En büyük sayıyı verir. Geriye değer ve indis döndürür. [val, ind] = max(v)
A < 3 => Mantıksal sorgulama yapar ve geriye 0-1 döndürür.
find(x < 3) => Kurala uyan elemanların indislerini döndürür. Matriste tüm elemanı döndürür.
magic("sayi") => Sihirli kareler matrisi oluşturur.
sum(v) => Vektörün elemanlarını toplar.
prod(v) => Vektörün elemanlarını çarpar.
floor(v) => Elemanları aşağı yuvarlar.
ceil(v) => Elemanları en yakın tam sayıya yuvarlar.

max(A, [1,1]) => Sütunların max'larını verir.
max(A, [1,2]) => Satırların max'larını verir.
max(max(A)) = max(A(:)) => Matrisin max'unu verir.

eye("sayi") => Birim matris döndürür.
flipud(A) => Matrisi dikey döndürür.
pinv(A) => Matrisin tersini verir.

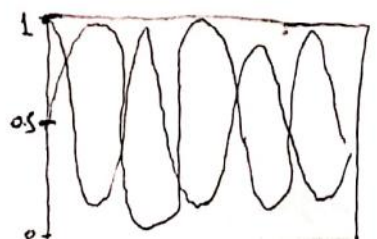
Veri Görselleştirme (Plotting Data)

```
> t = [0:0.01:0.98]
> y2 = cos(2*pi*t*4)
> y1 = sin(2*pi*4*t)
> plot(t, y1);
> hold on; % üstüste grafik çizer.
> plot(t, y2, 'r');
> xlabel('time'); % eksetiket belirler.
> ylabel('value');
```

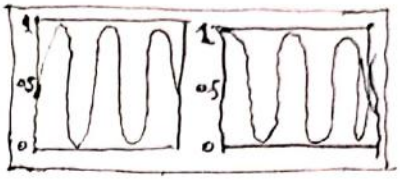
```
> legend('sin', 'cos') % anahtar kutu oluşturur.
> title('my plot') % Grafik başlığı koyar.

> print -dpng 'myplot.png' % Grafiği png kaydeder.

> figure(1); plot(t, y1);
> figure(2); plot(t, y2);
```



```
> subplot(1,2,1)  % Alanı 2'ye ayırdık ve ilkine sin yazıcak.
> plot(t,y1)
> subplot(1,2,2) % İkincisine cos yazıcak.
> plot(t,y2)
> axis([0.5 1  -1 1]) % İkinci grafiği x = 0.5 1
                        y = -1 1
                      aralığında gösterecek şekilde
                      ayarladık.
```



Not clf; => Grafik ekranı temizler.

```
> A = magic(5)
> imagesc(A)  % Grafiği 5x5 böler ve sayılara göre farklı renklerde gösterir.
> imagesc(A), colorbar  % Ayrıyeten renk barı koyar.
> imagesc(A), colorbar, colormap gray  % Ayrıyeten renkleri griye çevirir.
```

for, while, if statement

```
> v = zeros(10,1)
> for i=1:10,
>   v(i) = 2^i;
> end;
```

```
> i=1;
> While true
>     v(i) = 999;
>     i = i+1;
>     if i==6
>         break;
>     end;
> end;
```

```
> v(1) = 2;
> if v(1) == 1,
>     disp('The value is one');
> elseif v(1)==2,
>     disp('The value is two');
>  else
>     disp('The value is not one or two');
> end;
> The value is two
```

Function

```
function y = SquereThisNumber(x)
y = x^2

function [y1,y2] = squereandcubeThisNumber(x)
y1 = x^2
y2 = x^3

function J = costFunction(X, y, theta)
m = size(X,1);
predictions = X*theta
sqrErrors = (predictions-y).^2;
J = 1/(2*m) * sum(sqrErrors);
```

Code Exercise

```
fprintf ('Plotting data ...\n');
data = load ('ex1data1.txt')
X = data (:,1);   y = data (:,2);
m = length (y);

plotData (X, y);
```

```
function plotData(x,y)
figure (1);
plot (x, y, 'rx', 'MarkerSize', 10);
xlabel ('Population of City in 10000's');
ylabel ('Profit in $10,000's);
```

Cost and Gradient Descent

```
X = [ones (m,1), data(:,1)];
theta = zeros(2,1);

fpintf ('Testing the cost function ...\n');
J = computeCost (X, y, theta);
fprintf('with theta -[0 0]\n Cost computed = %0.2f\n', J);
fprintf('Expected cost value 32.07\n');
```

```
function J = computeCost(X, y, theta)
m = length (y);
J=0;

h= X * theta
meansquare = (h-y).^2;
J = 1/(2*m) * sum (meansquare);
```

```
iterations =1500;
alpha =0.01;
fprintf ('\n Running Gradient Descent ...');
[theta, J_history] = gradientDescent (X, y, theta, alpha,
                                      iterations);
```

```
function [theta, J_history] = gradientDescent
m= length (y);
J_history = zeros (num_iters, 1);

for iter = 1 : num_iters
    meansq = X * theta - y;
    delta = meansq' * X;
    theta = theta - alpha *(1/m)* delta';

    J_history(iter) = computeCost(X, y, theta)
end
end
```

θ değerleri

```
fprintf('Theta found by gradient descent:\n');
fprintf('%f %0.4f\n', theta);
fprintf('Expected theta values -3.6303  1.1664\n');
```

Modelimizin Grafiği

```
plot (X, y, 'rx', 'MarkerSize', 10);   % Veri Dağılımı
hold on;
plot (X(:,2), X * theta, '-');         % Model Doğrusu
legend ('Trening data', 'Lineer Regression');
hold off;
```

Modelde Tahmin Yapma

```
predict1 = [1, 3.5] * theta;
fprintf ('For population 35,000, we
  predict a profit of %f\n',
                    predict1 * 10000);
```

Visualizing $J(\theta_0, \theta_1)$

```
theta0 = linspace(-10, 10, 100);
theta1 = linspace(-1, 4, 100);

J = zeros(length(theta0), length(theta1));

for i=1:length(theta0)
    for j=1:length(theta1)
        t = [theta0(i); theta1(j)];
        J(i,j) = computeCost(x,y,t);
        end
end
```

Surface plot

```
J = J';
figure;
surf(theta0, theta1, J);
xlabel('theta0'); ylabel('theta1');
```

Contour plot

```
figure;
contour(theta0, theta1, J, logspace(-2,3,20));
xlabel('theta0'); ylabel('theta1');
hold on;
plot(theta(1), theta(2), 'rx', 'MarkerSize', 10,
                'LineWidth', 2);
```

Part II Multiple Linear Regression

```
data = load('ex1data2.txt');
X = data(:, 1:2);
y = data(:, 3);
m = length(y);
```

x = [2104, 3] y = 399900
x = [1600, 3] y = 329900
x = [2400, 3] y = 369000
x = [1416, 2] y = 232000
⋮

m = 47
n = 2

Mean Normalization

```
fprintf('Normalizing Features...\n');

[X mu sigma] = featureNormalize(X);

X = [ones(m,1), X]
```

```
function [X_norm, mu, sigma] = featureNormalize(X)
X_norm = X
mu = zeros(1, size(X,2));
sigma = zeros(1, size(X,2));
for i=1:size(X,2)
    mu(i) = mean(X_norm(:,i));
    sigma(i) = std(X_norm(:,i));

    X_norm(:,i) = (X_norm(:,i) - mu(i))/sigma(i);
end
```

Cost Function

$\frac{1}{2m}(X*\theta - \vec{y})^T (X*\theta - \vec{y})$

```
function J = computeCostMulti(X, y, theta)
meansq = X*theta - y;
J = 1/(2*m) * (meansq)' * (meansq);
end
```

Gradient Descent

```
alpha = 0.3
num_iters = 50;

theta = zeros(3,1);
[theta, J] = gradientDescentMulti(X, y, theta, alpha, num_iters);

figure;
plot(1:numel(J), J, '-b', 'LineWidth', 2);
xlabel('Number of iterations');
ylabel('Cost J');
```

```
function [theta, J] = gradientDescentMulti(...)
m = length(y);
J = zeros(num_iters, 1);
    for iter = 1:num_iters
      delta = X' * (X * theta - y);
      theta = theta - alpha * (1/m) * delta;

    J(iter) = computeCostMulti(X, y, theta);
    end
end
```

Tahmin (Predict)

```
price = 0
predict = [1650, 3];
predict = (predict - mu) / sigma;
price = [1 predict] * theta;
```

Normal Equations

```
data = csvread('ex1data2.txt');
X = data(:, 1:2);
y = data(:, 3);
m = length(y);

X = [ones(m,1) X];

theta = normalEqn(X, y);

fprintf('%f\n', theta);

price = 0
price = [1 650 3] * theta;
fprintf('%f\n' price);
```

```
function [theta] = normalEqn(X, y)
theta = zeros(size(X,2), 1);

theta = (pinv(X' * X)) * X' * y;

end
```