Linear Regression with multiple variables – multiple features

Çoklu doğrusal regresyon analizi iki ve daha fazla bağımsız değişken ve bir bağımlı değişken arasındaki doğrusal ilişkiyi inceleyen analizdir.

$b_0, b_1, b_2 \dots b_n$ değerleri regresyon katsayıları olup bağımlı değişkene, bağımsız değişkenler X denilir.

Denklem: $y_i = (b_0 + b_1 x_1 + b_2 x_2 + \dots + b_n x_n) + \varepsilon_i$

Ör

x_1 Size in feet	Number of bedrooms x_2	Number of floors x_3	Age of home in years x_4	Price in $1000's y
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
...	...	...	...	...

$x_j = j^{th}$ feature

n = number of features

$\vec{x}^{(i)}$ = features of i^{th} traing example

$\vec{x}^{(2)} = [1416 \quad 3 \quad 2 \quad 40]$

$x_j^{(i)}$ = value of feature j in i^{th} training example

$x_3^{(2)} = 2$

* $f_{w,b}(x) = (w_1 x_1 + w_2 x_2 + \cdots + w_n x_n) + b$

$w = \vec{w} = [w_1 \quad w_2 \quad w_3 \ldots w_n]$ → parameters of model

vector $\vec{x} = [x_1 \quad x_2 \quad x_3 \ldots x_n]$

$$f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b = w_1 x_1 + w_2 x_1 + \cdots w_n x_n + b$$

(skaler çarpım)

Vectorization

NumPy → Numerical liner algebra library in Python.

Ex Parameters and features

$\vec{w} = [w_1 \quad w_2 \quad w_3] \quad n=3$

b is a number

$\vec{x} = [x_1 \quad x_2 \quad x_3]$

with numpy $w[0]$ $w[1]$ $w[2]$

```
w = np.array([1.0, 2.5, -3.3])
b = 4
x = np.array([10, 20, 30])
```

$x[0]$ $x[1]$ $x[2]$

} start counting 0.

$f_{\vec{w},b}(\vec{x}) = w[0] \cdot x[0] + w[1] \cdot x[1] + w[2] \cdot x[2] + b$

General equation: $f_{\vec{w},b}(\vec{x}) = \left(\sum_{j=1}^{n} (w_j x_j)\right) + b$

Code:

```
f = 0
for ( j = 0 ; j < n)
{f = f + w[j] * x[j] }
f = f + b
```

Vectorization code

```
f = np.dot(w,x) + b
```

→ Bu numpy işlevi, nokta çarpımı verimli bir şekilde hesaplamak için paralel donanım kullanır.

* Vektörizasyon sayesinde kodu daha kısa yazarız ve kod daha hızlı çalışır.

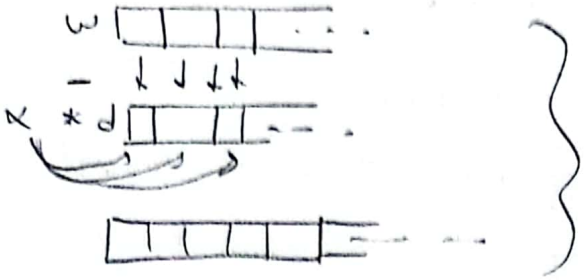
Gradient descent

Without vectorization → $w_1 = w_1 - \alpha d_1$

$w_2 = w_2 - \alpha d_2$

⋮

With vectorization → $\vec{w} = \vec{w} - \alpha \vec{d}$



Gradient descent for multiple linear regression

$J(\vec{w}, b)$ = cost function

$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} \underbrace{J(w_1, w_2, \dots w_n, b)}_{J(\vec{w}, b)}$$

$$b = b - \alpha \cdot \frac{\partial}{\partial b} \underbrace{J(w_1, w_2 \dots w_n, b)}_{J(\vec{w}, b)}$$

One feature

$$w = w - \alpha \frac{1}{m} \sum_{i=1}^{m} f_{w,b}(x^{(i)} - y^{(i)}) x^{(i)} \quad (j=1)$$

n features (n>1)

$$w_1 = w_1 - \alpha \cdot \underbrace{\frac{1}{m} \sum_{i=1}^{m} (f_{\vec{w},b}(\vec{x}^{(i)} - y^{(i)}) x_1^{(i)}}_{\frac{\partial}{\partial w_1} J(\vec{w}, b)}$$

⋮

$j=n$

$$w_n = w_n - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} f_{\vec{w},b}(\vec{x}^{(i)} - y^{(i)}) x_n^{(i)}$$

$$b = b - \alpha \frac{1}{m} \sum_{i=1}^{m} (f_{w,b}(x^{(i)} - y^{(i)}) \quad \rightarrow \quad b = b - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} (f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})$$

An alternative to gradient descent

Normal equation

- Only for linear regression
- Solve for w, b without iterations

Disadvantages

- Doesn't generalize to other learning algorithms
- Slow when numbers of features is large (> 10,000)

What you need to know

- Normal equation method may be used in ML libraries that implement linear regression.
- Gradient descent is the recommended method for finding parameters w, b.

Features Scaling

$300 \leq x_1 \leq 2000$

$x_1 \text{ scaled} = \frac{x_1}{2000}$

$0.15 \leq x_1 \text{ scaled} \leq 1$

$0 \leq x_2 \leq 5$

$x_2 \text{ scaled} = \frac{x_2}{5}$

$0 \leq x_2 \text{ scaled} \leq 1$

* Mean normalization (M)

$x_1 = \frac{x_1 - \mu_1}{2000 - 300} \Rightarrow x_1 = \frac{x_1 - \mu_1}{\max - \min} \rightarrow -0.18 \leq x_1 \leq 0.82$

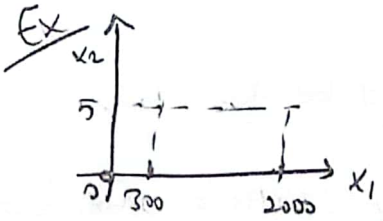
$x_2 = \frac{x_2 - \mu_2}{\max - \min} \rightarrow x_2 = \frac{x_2 - \mu_2}{5 - 0} \Rightarrow -0.46 \leq x_2 \leq 0.54$

* Z-score normalization

(σ) Standart deviation → standart sapma

1) Verilen aritmetik ortalaması bulunur.
2) Her bir veri ile AO. arasındaki fark bulunur
3) Bulunan farkların karesi alınır ve elde sayılar toplanır

Ex



$300 \leq x_1 \leq 2000$ ↓ $x_1 = \frac{x_1 - \mu_1}{\sigma_1}$

$0 \leq x_2 \leq 5$ ↓ $x_2 = \frac{x_2 - \mu_2}{\sigma_2}$

In feature scaling, aim for about $-1 \leq x_j \leq 1$ for each feature x_j

$-3 \leq x_j \leq 3$ ✓

$-0.3 \leq x_j \leq 0.3$ ✓

$0 \leq x_1 \leq 3$ ✓

$-2 \leq x_2 \leq 5$ ✓

$-100 \leq x_3 \leq 100$ → too large → rescale

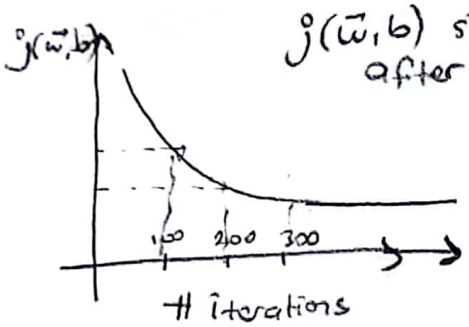
$-0.0001 \leq x_4 \leq 0.0001$ → too small → rescale

* Özellik ölçeklendirilmesi sayesinde gradyan inişi daha hızlı çalışır.

* Her değeri o özellik için max değere bölün

Checking Gradient Descent for Convergence

Maliyet fonksiyonunun parametrelerini global minimuma yakın bulmak, gradyan inişinin yakınsaklığını gösterir



$J(\vec{w},b)$ should decrease after every iteration.

Not: Eğer J iterasyon sayısı arttıkça artıyorsa, ya α çok büyük seçilmiştir ya da kodda bir yazılım hatası (bug) vardır.

(flatten out)

İterasyon sayısı arttıkça grafik düzleşir, bu da gradyen inişinin çok ya da az yakınsadığını belirtir. Çünkü eğri artık azalmaz.

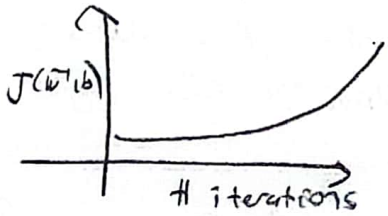
Automatic convergence Test

Let ε "epsilon" 10^{-3} → değiştirilir

If $J(\vec{w},b)$ decreases by $\leq \varepsilon$ in one iteration, declare convergence

Choosing the Learning Rate

Learning algorithm will run much better with an appropriate choice of learning rate.

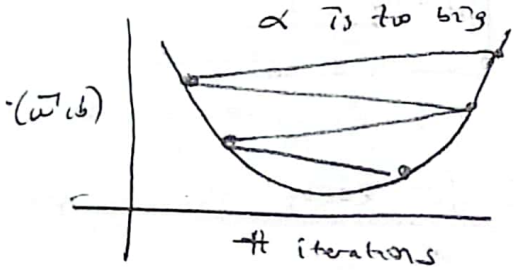


J fonk. (arttırır)

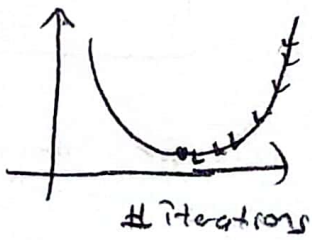
$w_1 = w_1 + \alpha \cdot d_1$ ✓

$w_1 = w_1 - \alpha \cdot d_1$ ✗ (J fonk. azaltır)

There could be a bug or learning rate could be too large.

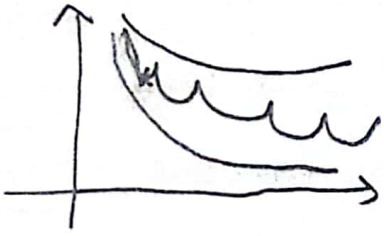


→ use smaller α



with a small enough α, $J(\vec{w},b)$ should decrease on every iteration.

* If α is very very small, J doesn't decrease on every iteration, that means there is a bug somewhere in the code.



* Maliyet fonk. arttığından, gradyen inişinin ıraksadığını biliyoruz, bu nedenle daha düşük bir öğrenme oranına ihtiyacımız var.

Feature Engineering

Using intuition (knowledge) to design new features, by transforming or combining original features of the problem in order to make it easier for the learning algorithm to make accurate predictions.

Polynomial Regression

Polynomial regression, let us fit curves, non-linear function to your data.

$$f_{\vec{w},b}(x) = w_1 x + w_2 x^2 + b$$

$$f_{\vec{w},b}(x) = w_1 (x) + w_2 (x^2) + w_3 (x^3) + b$$

or

x: size, $1 - 10^3$

x^2: size, $1 - 10^6$

x^3: size, $1 - 10^9$

farklı kullanabiliriz.

$$f_{\vec{w},b}(x) = w_1 x + w_2 \sqrt{x} + b$$

$\sqrt{x}$: $\sqrt{\text{size}}$

* scikit learn, Sınıflandırma, regresyon, kümeleme ve karar ağaçları gibi farklı makine öğrenme algoritmalarını birleştiren bir python kütüphanesidir