GİRİŞ

Eğitmen

Özgür YILDIRIM

linkedin.com/in/Ozgur-yldrm/

github.com/OzgurYldrm

Sunum + Notebook + Homework → Github

- Giriş
- Veri
- Linear Regression
- Loss Functions
- Gradient Descent
- Logistic Regression
- Batching

Input-Output

Klasik kodlama: [Input Verisi] → [Kod] → [Output Tahmini]



Supervised: [Input Verisi] → [Yapay Zeka Modeli] → [Output Tahmini]



Unsupervised: [Input Verisi] → [Yapay Zeka Modeli] → [Output Tahmini]




Reinforcement: [Input Verisi] → [Yapay Zeka Modeli] → [Output Tahmini]

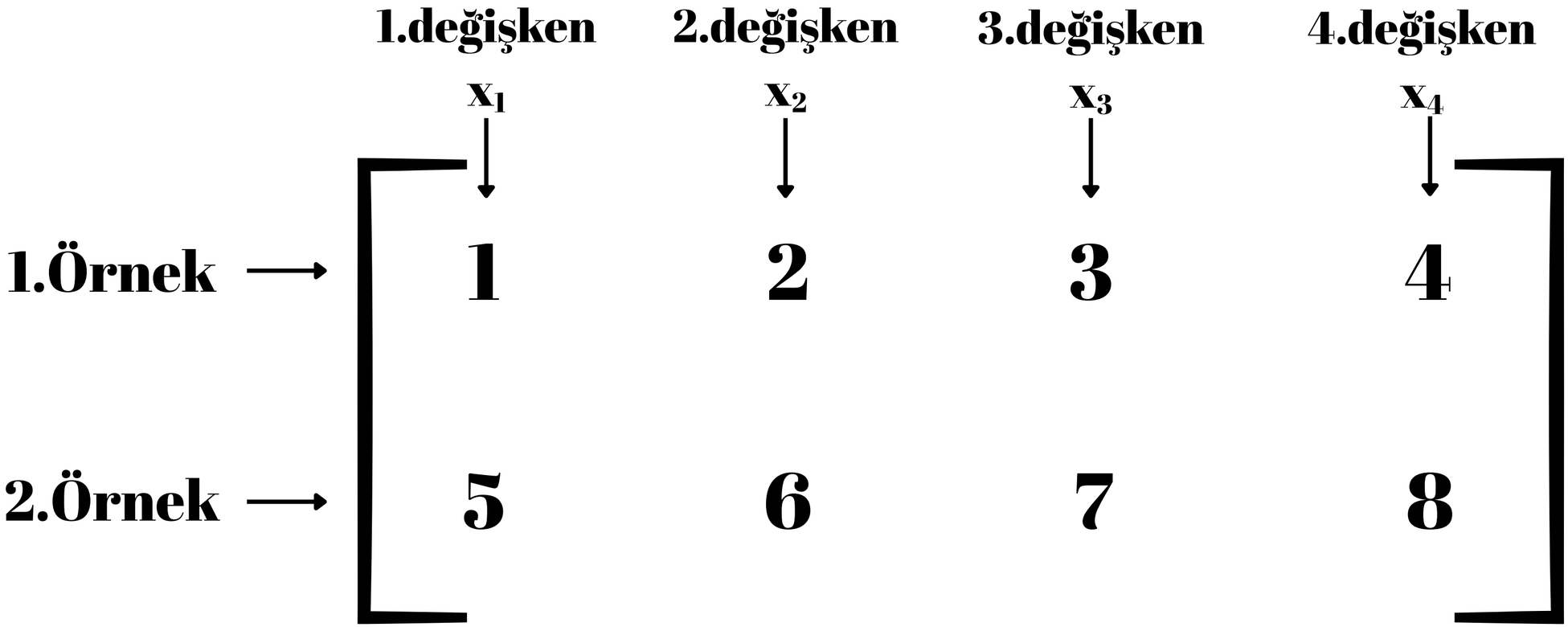


Input-Output Türleri

INPUT (Features)	ÖRNEK
Integer	3, 5, -7
Float	3.14 , 0.21
Boolean	Sosyal sigorta? , Çalışıyor mu? , Evli mi?
Class	Cinsiyet? , Ehliyet tipi? , kanser tipi?
Metin, Resim, Ses ...	Sayı ile ifade çabası

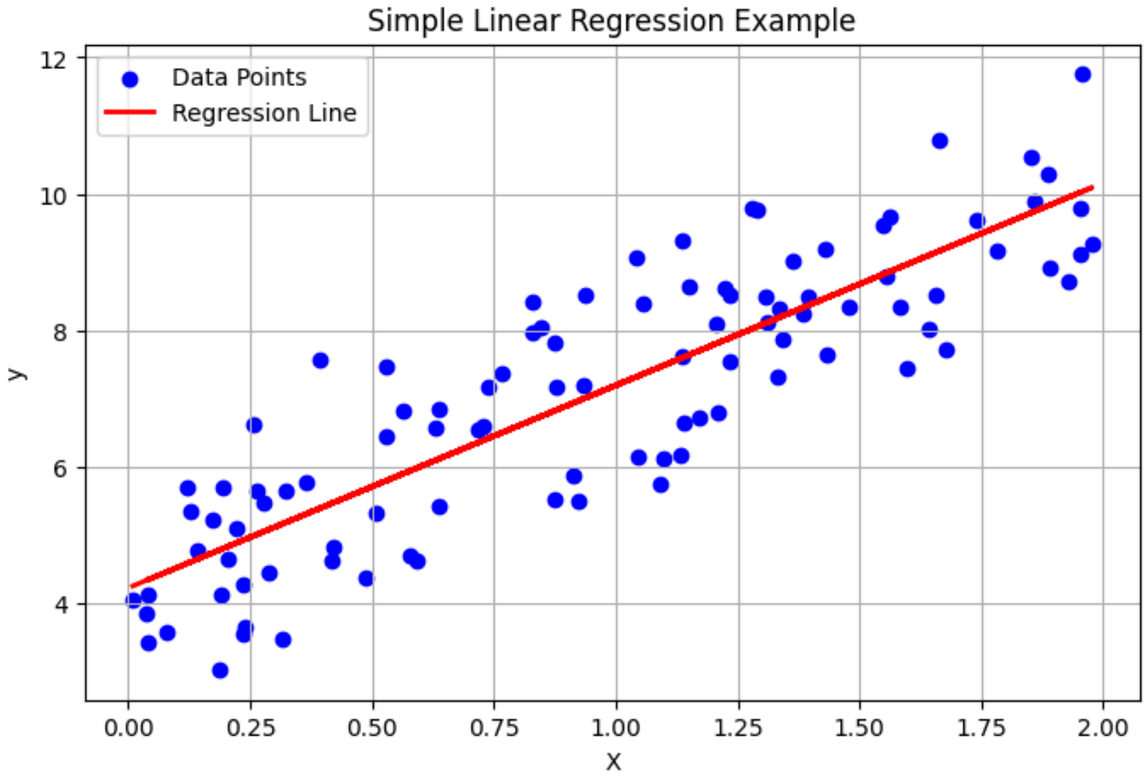
OUTPUT	ÖRNEK
Continuous	0.21 , 8.56
Class	0: köpke 1: kedi 2: kuş
Rank	8: en iyi sigorta 0: en kötü sigorta
Metin, Resim, Ses	Sayısal çıktıların dönüştürülmesi ile oluşur.

Matrix gösterim



Modül içerisindeki kodlar bazen for döngüsü ile eleman bazlı bazen de matrix işlemleri ile "vectorized" olarak yazılmıştır.

Linear Regression Amacı

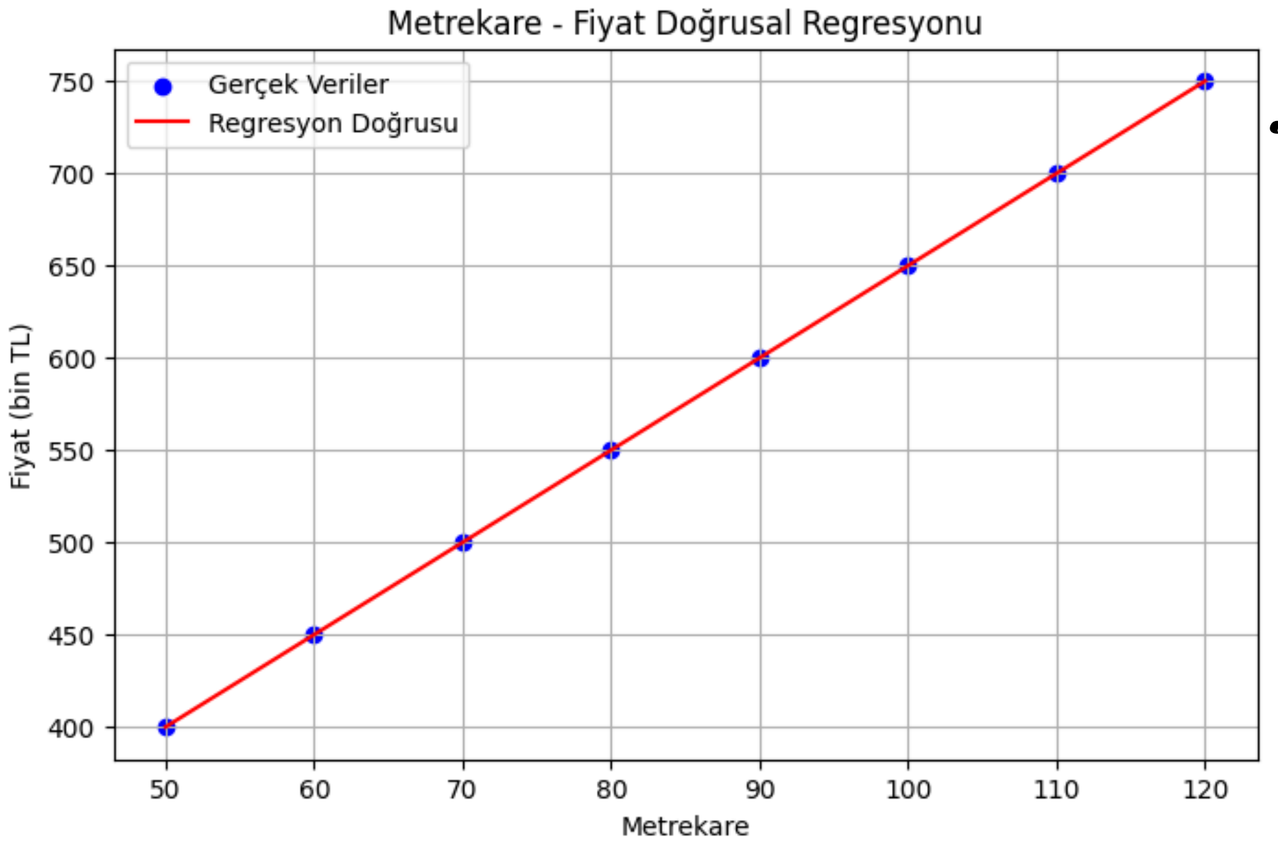


- **Sürekli çıktı üreterek en yakın tahmini yapmak (Function Approximation)**
  - **Ev fiyatı tahmini**
  - **Proje bütçe hesaplaması ...**

* Grafiği üreten python kodu ChatGPT tarafından hazırlanmıştır

Linear Regression Amacı

Metrekare (X)	Fiyat (Y, bin TL)
50	400
60	450
70	500
80	550
100	650



$$y = 5x + 150$$

Parametre

- **Eğitilmiş modeli kullarak farklı evler için tahmin yapabiliriz.**

120 m^2 evin tahmini fiyatı (y') → 5.(120) + 150 = 750

* Grafiği üreten python kodu ChatGPT tarafından hazırlanmıştır

Linear Regression Sorular

- Parametreleri nasıl buluruz?
- Sadece tek bir değişken olmak zorunda değil?
- Algoritmanın ne kadar doğru çalıştığını nasıl bileceğiz?
- Tamamen doğru sonuç vermiyor :(

Loss Functions

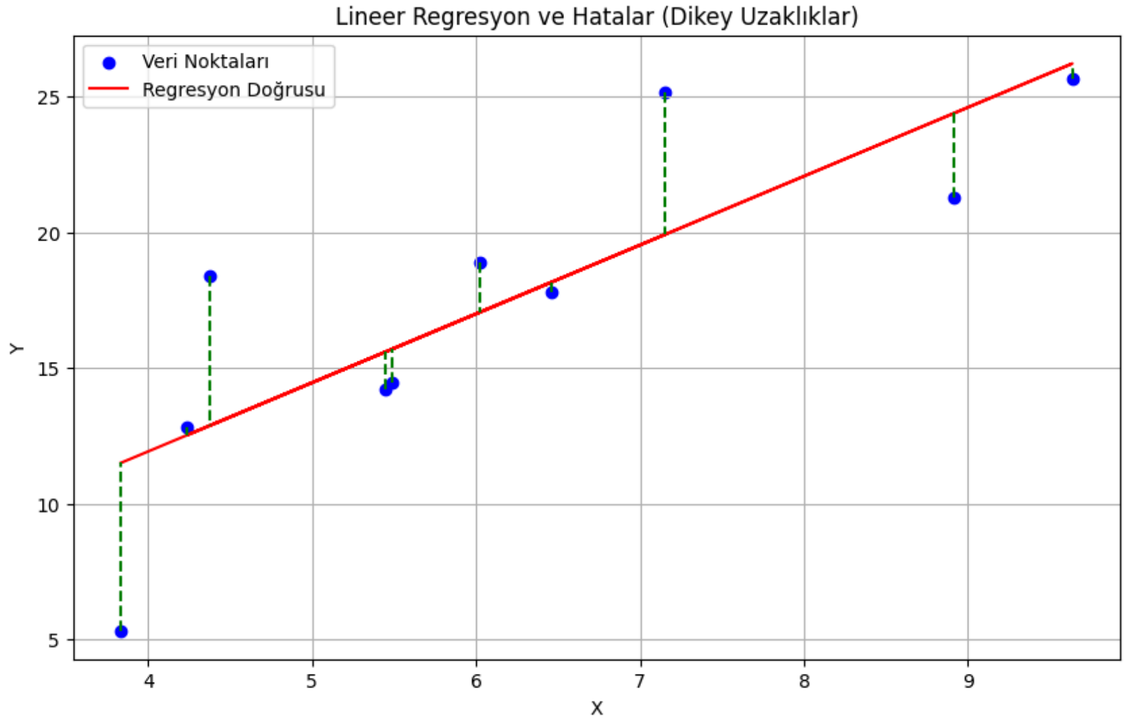
Ground Truth (y)	Prediction (y')
10	12
20	18
30	30

Ground Truth: Tahmin edilmeye çalışılan gerçek değer
Prediction: Algoritma tarafından tahmin edilen değer
Loss/Error: Gerçek değer ile tahmin arasındaki fark
Loss function: Loss'u tahmin etmek için kullanılan model

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

Mean Squared Error

Y_i : i. gerçek değer
Y'_i : i. tahmin
n: örnek sayısı



Residual

* Grafiği üreten python kodu ChatGPT tarafından hazırlanmıştır

MSE
Mean Squared Error

MSE

Ground Truth (y)	Prediction (y')	Loss
10	12	4
20	18	4
30	30	0

Loss = 8/3

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

Kare olmazsa

Ground Truth (y)	Prediction (y')	Loss
10	12	-2
20	18	2
30	30	0

Loss = 0

***Loss genellikle negatif olmaz**

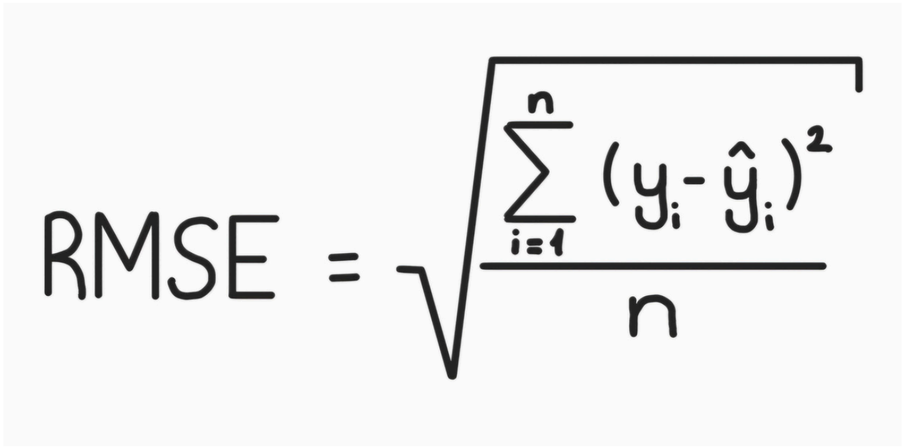
***Ayrıca kare alma işlemi büyük hataları daha büyük yaparak algoritmanın daha hızlı öğrenmesini sağlayabilir**

RMSE
Root Mean Squared Error

RMSE

Ground Truth (y)	Prediction (y')	Loss
10	12	4
20	18	4
30	30	0

$$\text{Loss} = (8/3)^{1/2}$$



MAE
Mean Absolute Error

Ground Truth (y)	Prediction (y')	Loss
10	12	2
20	18	2
30	30	0

$$MAE = \frac{\sum_{i=1}^{n} |y_i - \hat{y}_i|}{n}$$

Loss = 4/3

***Farklı loss function'ların farklı avantajları olabilir. Birbirinden farklı sonuç vermeleri doğrudan iyi veya kötü olarak yorumlanamaz**

Huber Loss

Ground Truth (y)	Prediction (y')	Abs	Loss
10	15	5	12
20	18	2	2
30	40	10	32

$$Huber = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{2}(y_i - \hat{y}_i)^2 \qquad |y_i - \hat{y}_i| \leq \delta$$

$$Huber = \frac{1}{n}\sum_{i=1}^{n}\delta\left(|y_i - \hat{y}_i| - \frac{1}{2}\delta\right) \qquad |y_i - \hat{y}_i| > \delta$$

Loss = 46/3

Delta: 4

- **Küçük hatalara duyarlı, büyük hatalara karşı dayanıklıdır.**
- **Gürültünün fazla olduğu verilerde uç noktalardaki verilerin (outlier) algoritmayı olumsuz etkilemesinin önüne geçer.**

***Huber loss ve Smooth L1 Loss fonksiyonun formülünde bulunan Beta ve Sigmaya bölme ve çarpma işlemleri parçalı fonksiyonun geçiş noktalarında süreklilik ve türevin sağlanması açısından bulunmaktadır.**

Smooth L1 Loss

Ground Truth (y)	Prediction (y')	Abs	Loss
10	15	5	2.5
20	18	2	0.4
30	40	10	7.5

$$L_n = \begin{cases} 0.5 \cdot (\hat{y}_n - y_n)^2 / \beta & , \text{ if } |\hat{y}_n - y_n| < \beta \\ |\hat{y}_n - y_n| - 0.5 \cdot \beta & , \text{ otherwise} \end{cases}$$

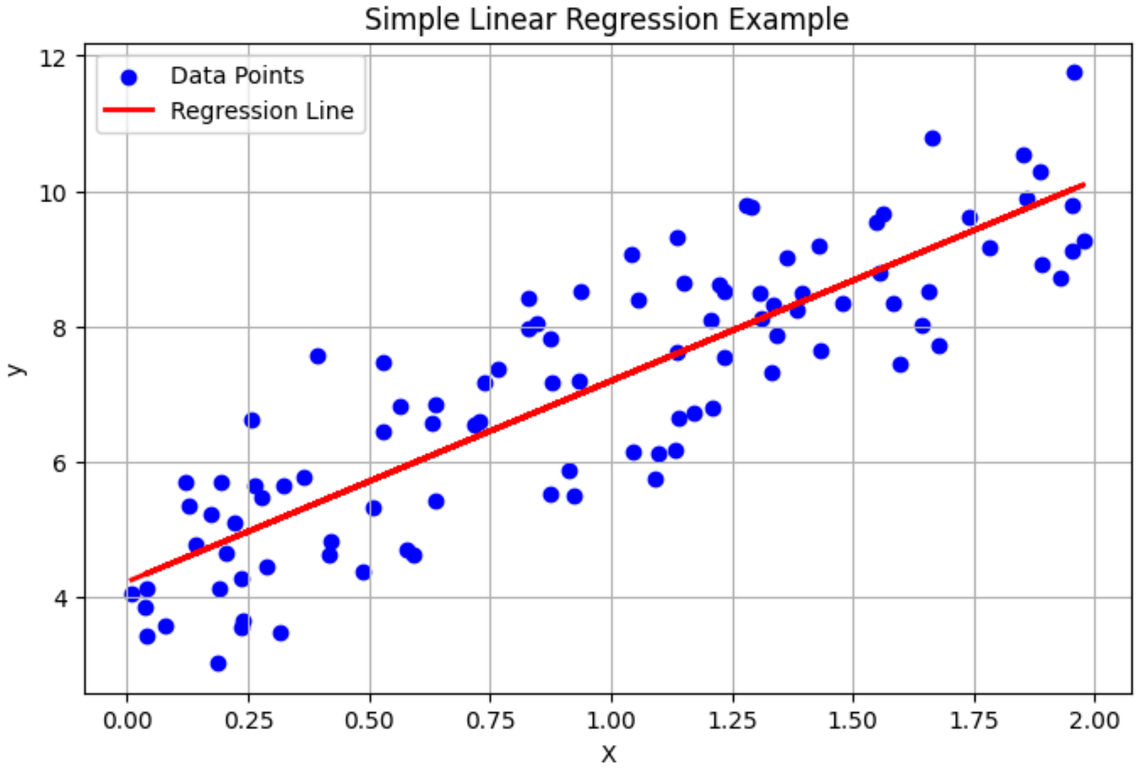
Loss = 10.4/3=3.47

Beta = 5

- **Küçük hatalara duyarlı, büyük hatalara karşı dayanıklıdır.**
- **Gürültünün fazla olduğu verilerde uç noktalardaki verilerin (outlier) algoritmayı olumsuz etkilemesinin önüne geçer.**

Huber ve SL1 loss birbirine benzemektedir. SL1'in kare kısmı betaya bölünür. Literatürde isim ve formül karışıklığı var. Ama hepsi aynı fikir etrafında.

Linear Regression Model



$$y = wx + b$$

w: weight

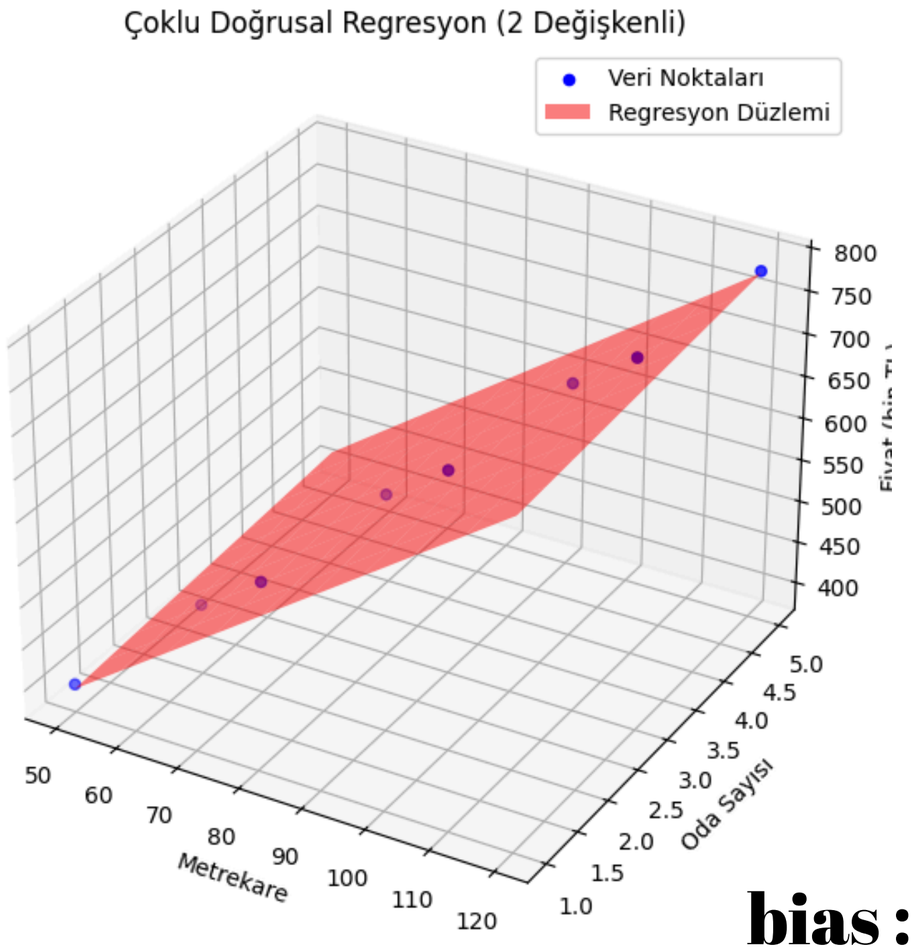
b: bias

Amaç en iyi w ve b değerlerini bulmak

Amaç en iyi çizgiyi çekmek

* Grafiği üreten python kodu ChatGPT tarafından hazırlanmıştır

Multivariable Linear Regression



Genel formül

$$w_1x_1 + w_2x_2 + \dots + w_nx_n + b$$

w_i : i. weight

x_i : i. özellik

b : bias

bias : 133

w_1: 5.1

w_2: 6.5

$\longrightarrow$ X_1 : 75, X_2 : 2 $\longrightarrow$ (5.1 * 75) + (6.5 * 2) + 133

* Grafiği üreten python kodu ChatGPT tarafından hazırlanmıştır

Multivariable Linear Regression Matrix gösterimi

$$w_1 = 1,\quad w_2 = 2,\quad w_3 = 3 \Rightarrow w = \begin{bmatrix} 1 \to w_1 \\ 2 \to w_2 \\ 3 \to w_3 \end{bmatrix}$$

3×1

$$* \quad y = w_1 x_1 + w_2 x_2 + w_3 x_3 + \cdots + w_n x_n$$

$$x = \begin{bmatrix} x_1 & x_2 & x_3 \\ 1 & 2 & 3 \\ 4 & 5 & 6 \\ -1 & -2 & -3 \\ -4 & -5 & -6 \end{bmatrix}$$

4×3

$$x \cdot w = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ -1 & -2 & -3 \\ -4 & -5 & -6 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 14 \\ 32 \\ -14 \\ -32 \end{bmatrix}$$

$4 \times 3 \quad 3 \times 1 \quad 4 \times 1$

***Hint: np.matmul()**

Gradient Descent

Gradient Descent

Repeat until converge {

$$w = w - \alpha \left[\frac{\partial Loss}{\partial w}\right]$$

$$b = b - \alpha \left[\frac{\partial Loss}{\partial b}\right]$$

}

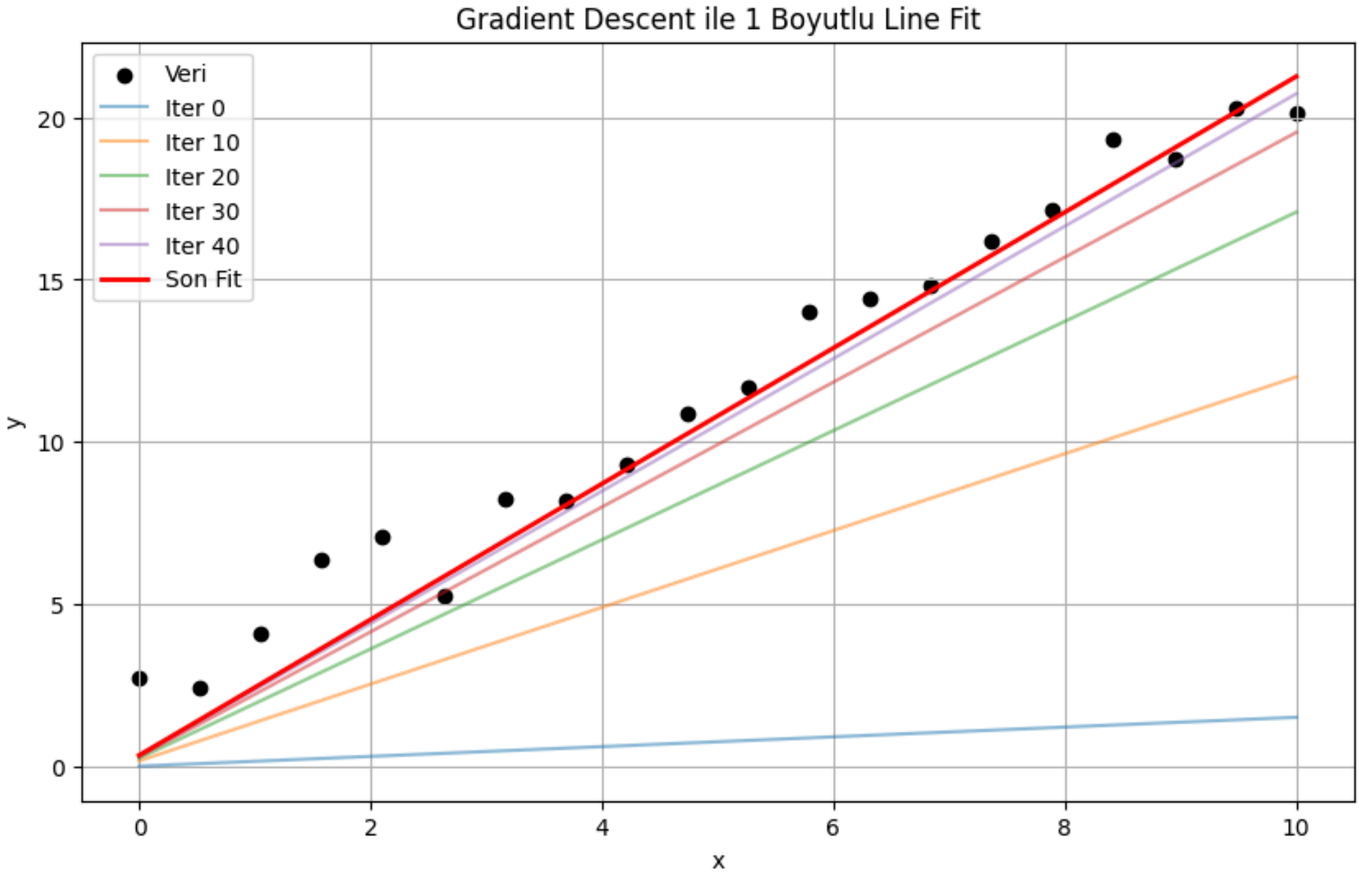
w: weight

b: bias

alpha: Learning rate

Partial Derivation

Gradient Descent Nasıl görünüyor



Repeat until converge { -> **ne demek**

converge:**Hatanın minimuma yaklaşması**

iteration: **weight-bias güncellemelerinin yapıldığı her bi adım**

* Grafiği üreten python kodu ChatGPT tarafından hazırlanmıştır

Partial Derivative

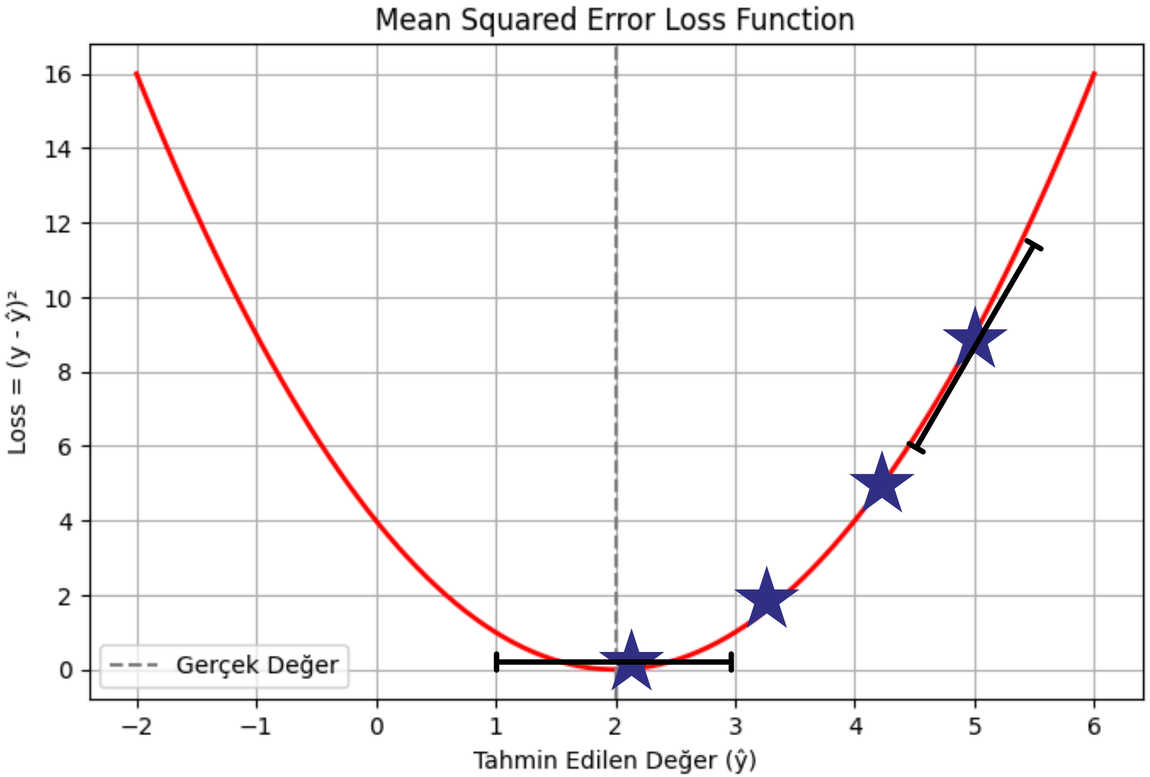
$$MSE = \frac{1}{n}\sum (y-\hat{y})^2 \qquad (\hat{y} = w_1 \cdot x_1 + w_2 \cdot x_2 + \cdots + w_m \cdot x_m + b)$$

$$MSE = \frac{1}{n}\sum (y - [w_1 \cdot x_1 + w_2 \cdot x_2 + \cdots + w_m \cdot x_m + b])^2$$

$$\frac{\partial (MSE)}{\partial w_i} = \frac{-2}{n}\sum (y - [w_1 \cdot x_1 + w_2 \cdot x_2 + \cdots + w_m \cdot x_m + b]) \cdot x_i$$

$$\frac{\partial (MSE)}{\partial b} = \frac{-2}{n}\sum (y - [w_1 \cdot x_1 + w_2 \cdot x_2 + \cdots + w_m \cdot x_m + b])$$

Gradient Descent



istenilen değer

İstenilen fonksiyon → y = 2x

Adım	w	y - (wx)	- [y-(wx)] * x * alpha
1	5	-3	0.3
2	4.7	-2.7	0.27
n	2.01	-0.01	0.001

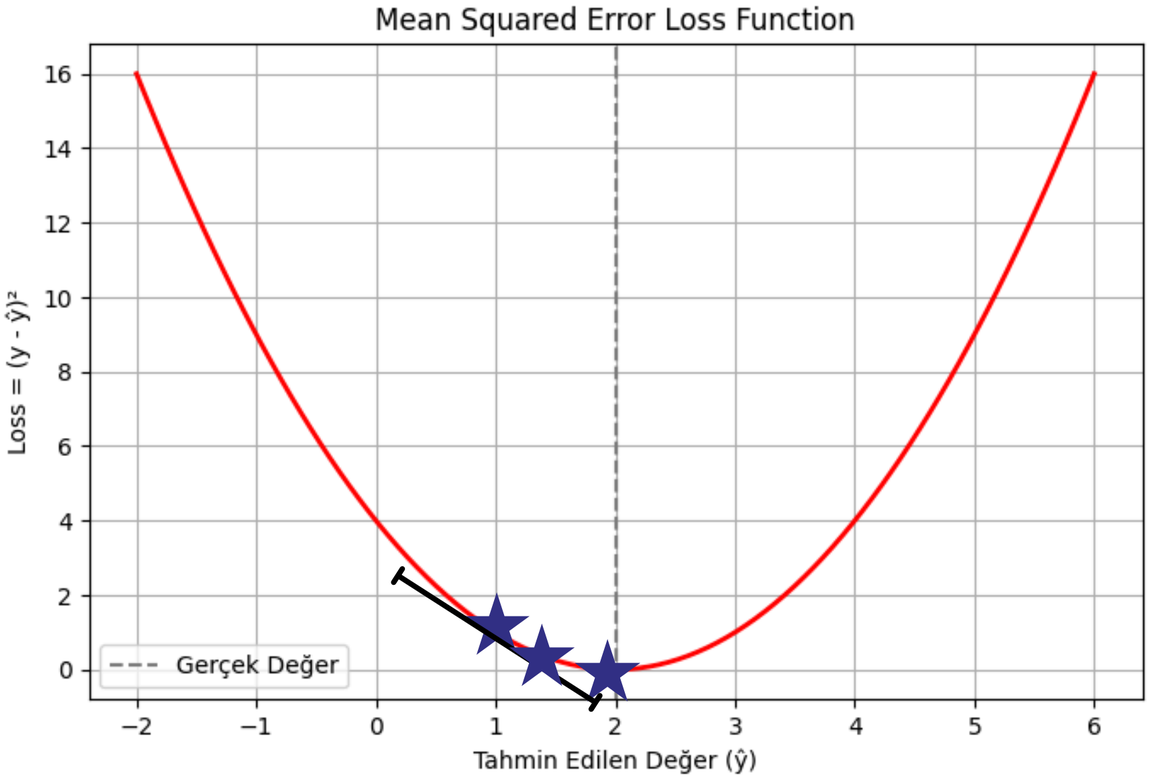
x=1 y=2 alpha=0.1

$$w = w - \alpha \left[\frac{\partial Loss}{\partial w}\right]$$

$$b = b - \alpha \left[\frac{\partial Loss}{\partial b}\right]$$

* Grafiği üreten python kodu ChatGPT tarafından hazırlanmıştır

Gradient Descent



x=1

İstenilen fonksiyon → y = 2x

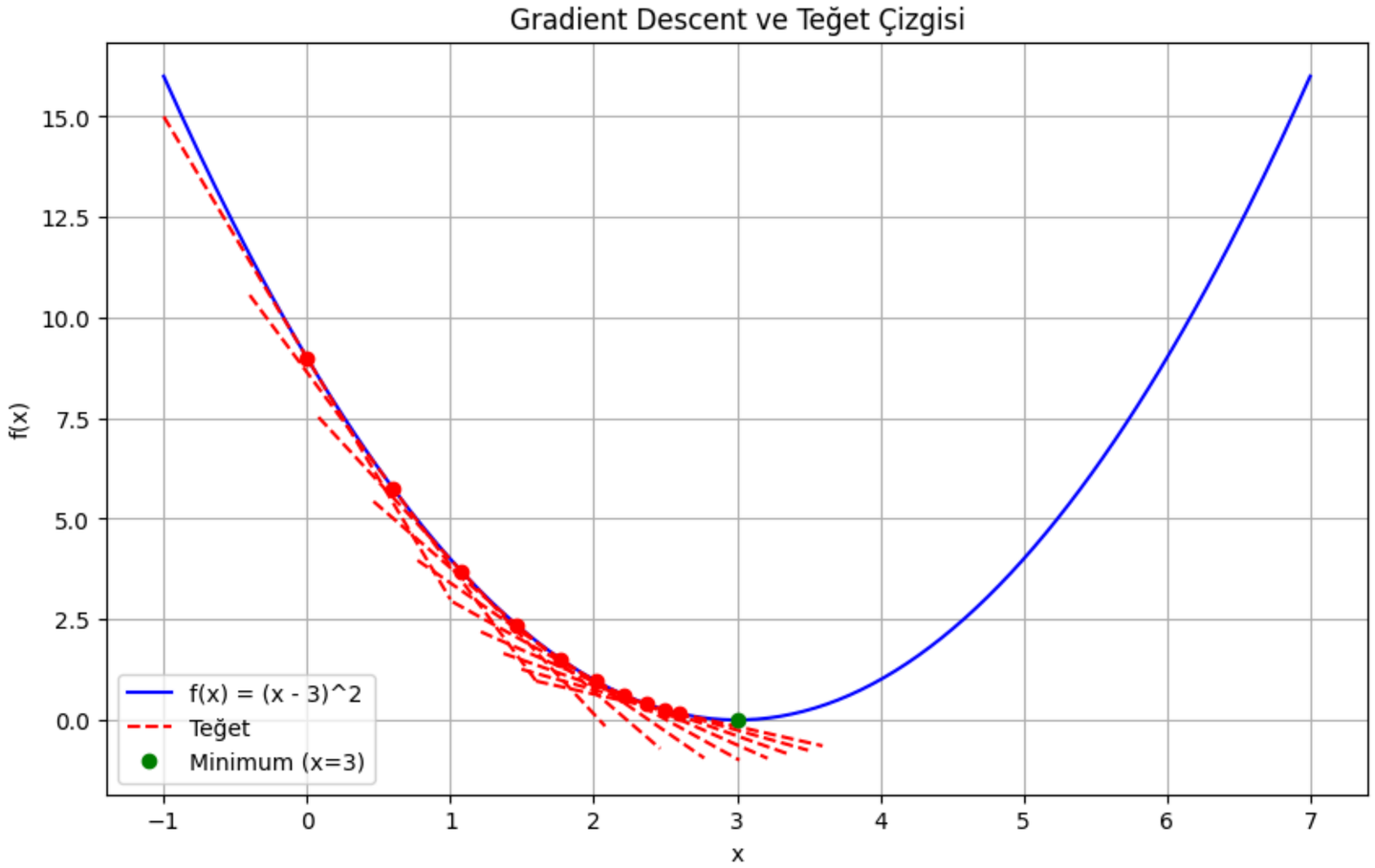
Adım	w	y - (wx)	- [y-(wx)] * x * alpha
1	1	1	-0.1
2	1.1	0.89	-0.08
n	1.99	0.005	-0.0005

$$w = w - \alpha \left[\frac{\partial Loss}{\partial w}\right]$$

$$b = b - \alpha \left[\frac{\partial Loss}{\partial b}\right]$$

* Grafiği üreten python kodu ChatGPT tarafından hazırlanmıştır

Türevin önemi

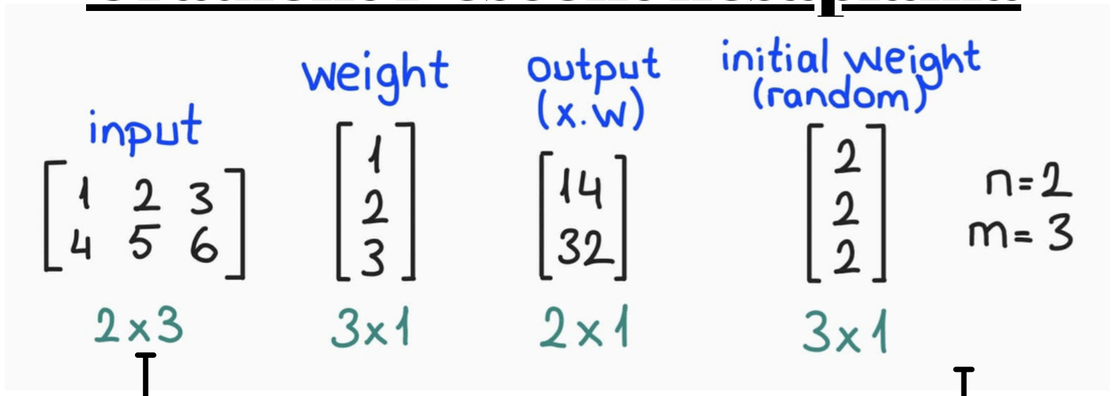


1. Güncelleme yönü
2. Güncelleme büyüklüğü
3. Minimumda 0 olma işi

* Grafiği üreten python kodu ChatGPT tarafından hazırlanmıştır

*Videolarda element-wise olarak kullandığım ifade tam anlamıyla doğru değil. Doğru kullanım non-vectorized ya da scalar implementation şeklinde olmalıydı.

Gradient Descent hesaplama



For w_1

1.örnek

$y' = (1*2) + (2*2) + (3*2) = 12$

$Error\ (E) = 14-12 = 2$

$Gradient = 2*1 = 2$

2.örnek

$y' = (4*2) + (5*2) + (6*2) = 30$

$Error\ (E) = 32-30 = 2$

$Gradient = 2*4 = 8$

$w_1 = 2 - (0.01)*(-2/2)*(2+8) = 2.1$

Alpha, -2/n, Partial Derivation

For w_2

$y' = (1*2) + (2*2) + (3*2) = 12$

$Error\ (E) = 14-12 = 2$

$Gradient = 2*2 = 4$

$y' = (4*2) + (5*2) + (6*2) = 30$

$Error\ (E) = 32-30 = 2$

$Gradient = 2*5 = 10$

$w_2 = 2 - (0.01)*(-2/2)*(4+10) = 2.14$

Alpha, -2/n, Partial Derivation

For w_3

$y' = (1*2) + (2*2) + (3*2) = 12$

$Error\ (E) = 14-12 = 2$

$Gradient = 2*3 = 6$

$y' = (4*2) + (5*2) + (6*2) = 30$

$Error\ (E) = 32-30 = 2$

$Gradient = 2*6 = 12$

$w_3 = 2 - (0.01)*(-2/2)*(6+12) = 2.18$

Alpha, -2/n, Partial Derivation

Gradient Descent Matrix gösterimi

input $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ 2x3

weight $\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$ 3x1

output (x.w) $\begin{bmatrix} 14 \\ 32 \end{bmatrix}$ 2x1

initial weight (random) $\begin{bmatrix} 2 \\ 2 \\ 2 \end{bmatrix}$ 3x1

$n=2$
$m=3$

Prediction:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 12 \\ 30 \end{bmatrix} \quad (x \cdot w)$$

Error (E):

$$\begin{bmatrix} 14 \\ 32 \end{bmatrix} - \begin{bmatrix} 12 \\ 30 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \end{bmatrix} \quad (y - x \cdot w)$$

Gradient:

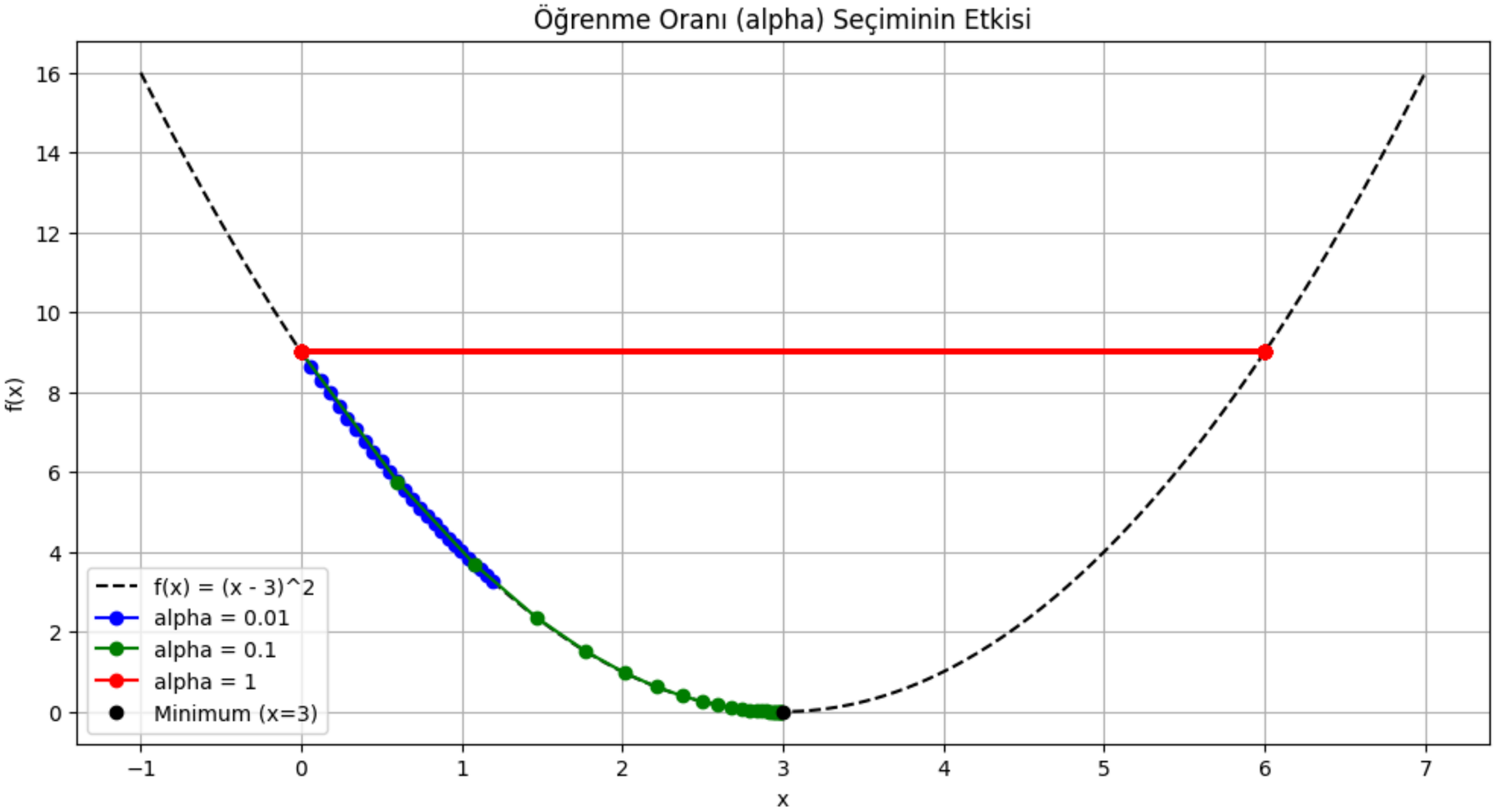
$$\frac{-2\left(\begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 2 \end{bmatrix}\right)}{2} = \begin{bmatrix} -10 \\ -14 \\ -18 \end{bmatrix} \quad \left(\frac{-2}{n}[x^T \cdot E]\right)$$

New Weigths:

$$\begin{bmatrix} 2 \\ 2 \\ 2 \end{bmatrix} - \begin{bmatrix} -0.10 \\ -0.14 \\ -0.18 \end{bmatrix} = \begin{bmatrix} 2.10 \\ 2.14 \\ 2.18 \end{bmatrix} \quad (\alpha = 0.01)$$

MSE Before: 4.0 MSE After: 0.59

Learning Rate Seçimi



$$w = w - \alpha \left[\frac{\partial Loss}{\partial w}\right]$$

$$b = b - \alpha \left[\frac{\partial Loss}{\partial b}\right]$$

Her bir güncelleme adımının büyüklüğünü belirler.
büyük alpha → büyük adım

* Grafiği üreten python kodu ChatGPT tarafından hazırlanmıştır

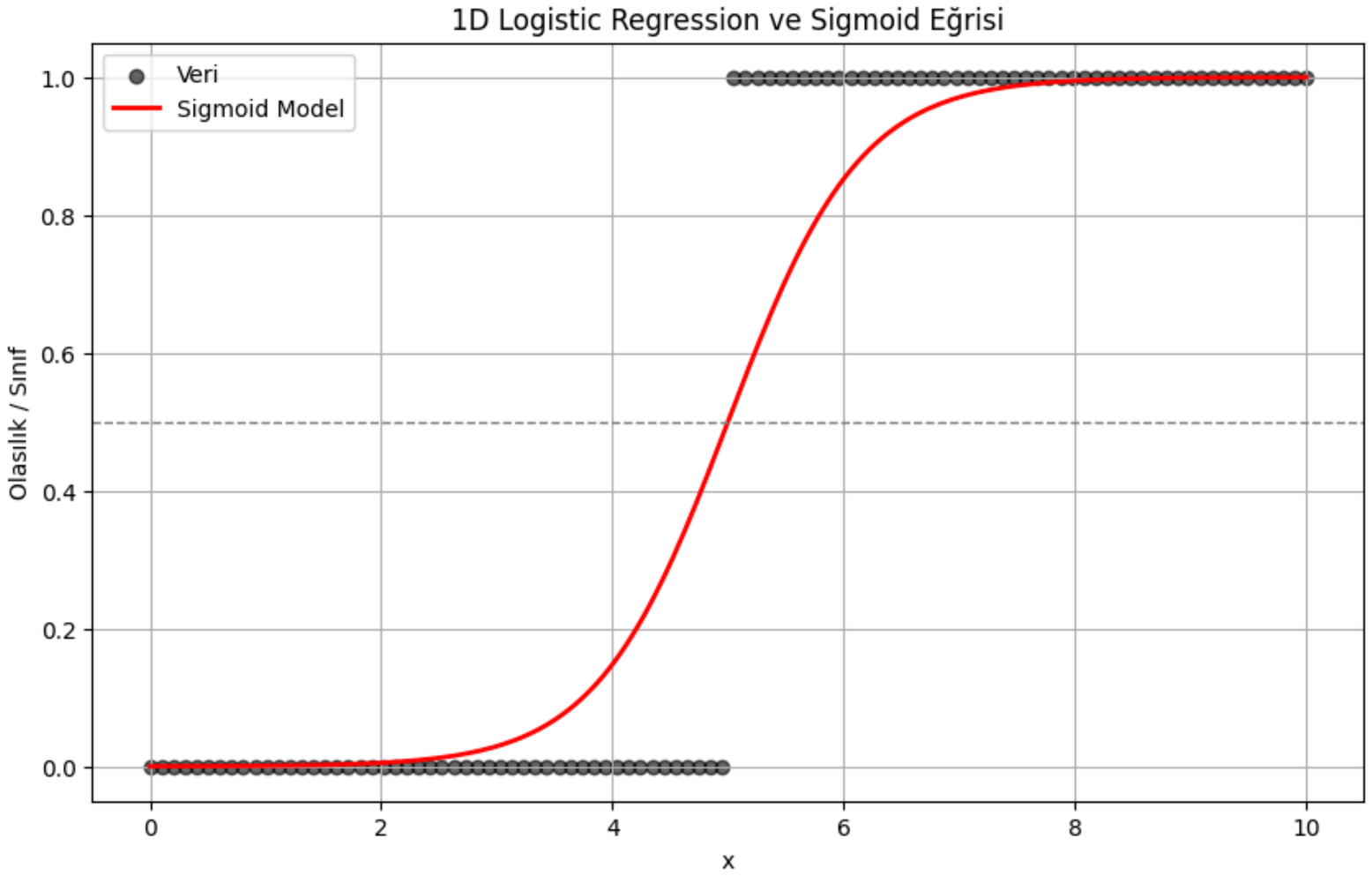
Normal Equation

Linear Regression parametrelerini bulmak için kullanılan kapalı form çözüm yöntemidir.

$$w = (x^T \cdot x)^{-1} \cdot x^T \cdot y$$

- **Büyük matrix'ler için hesaplaması pahalı**
- **Tersi alınmayan matrix olma ihtimali var**

Logistic Regression



Sınıfsal çıktı üreterek verilen girdilere göre sınıflandırma yapmak

- **Kitle büyüklüğüne göre kanser analizi**
- **Gelir durumuna göre vergi verip vermeyeceği**

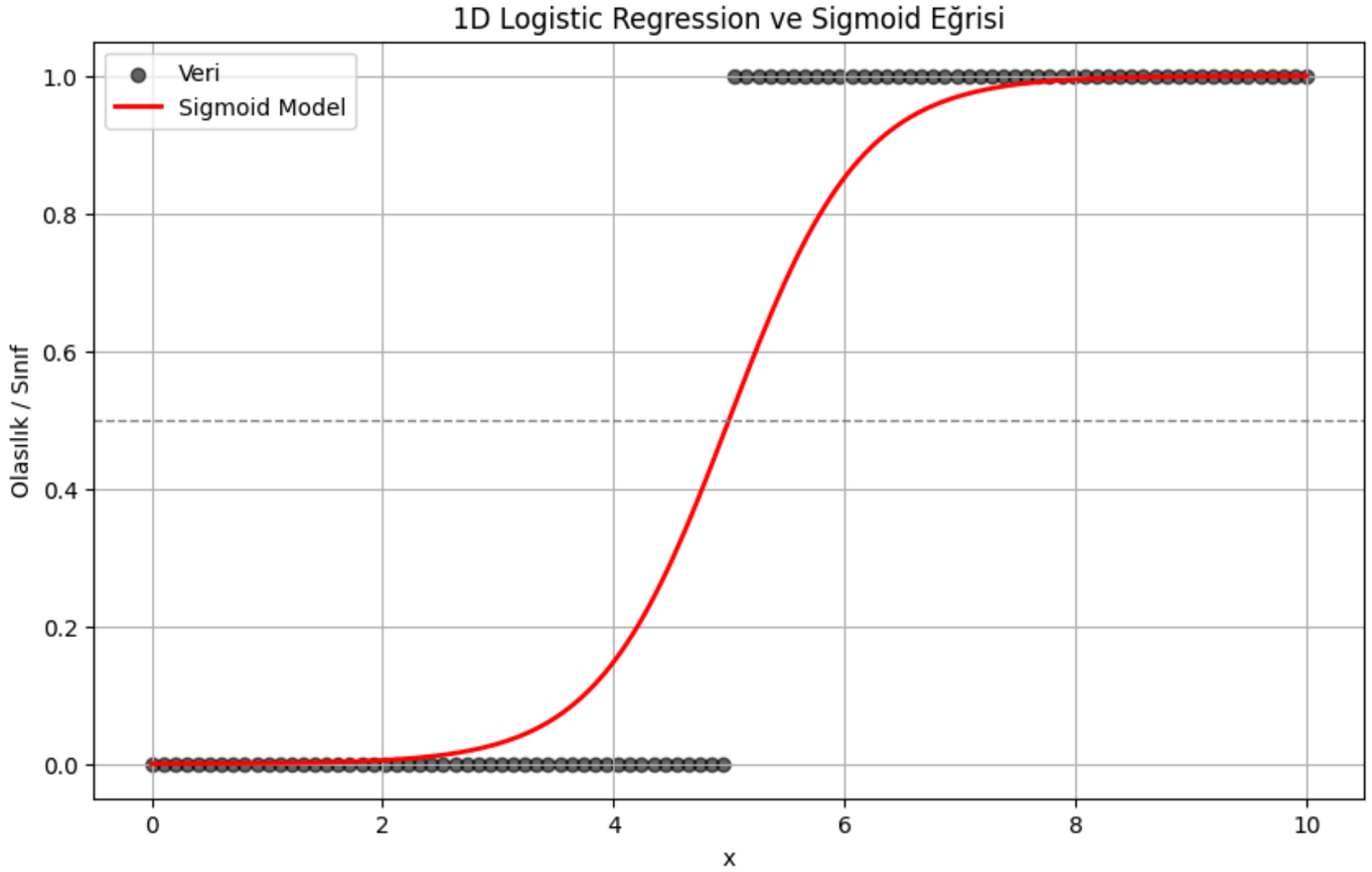
Genellikle

1 → pozitif sınıf

0 → negatif sınıf

* Grafiği üreten python kodu ChatGPT tarafından hazırlanmıştır

Logistic Regression amacı



x=7 inputu için → %90 pozitif sınıf

Eğer fonksiyon elimizde olursa yeni girdiler için sınıflandırma yapabiliriz

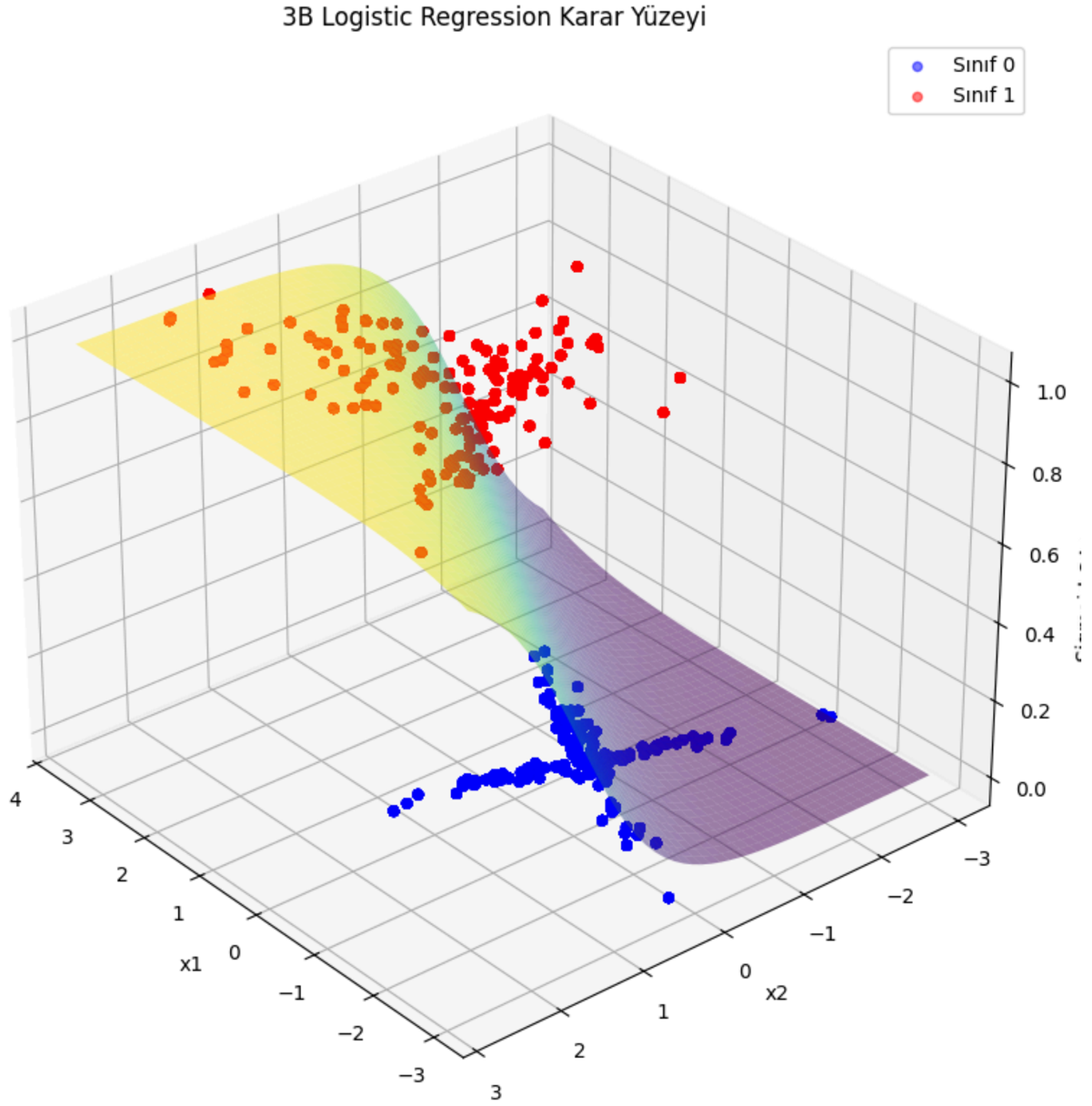
Gerçek çıktılar sadece 0 ya da 1

(Daha genel olarak sadece sınıflara karşılık gelen tam sayılar

***Multilabel → Belki sonra**

* Grafiği üreten python kodu ChatGPT tarafından hazırlanmıştır

Multivariable Logistic Regression nasıl görünüyor



* Grafiği üreten python kodu ChatGPT tarafından hazırlanmıştır

BCE
Binary Cross Entropy

$$BCE = -\frac{1}{n}\sum_{i=1}^{n}(y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i))$$

y	y'	y log(y')	(1-y) log(1-y')
1	0.9	-0.10	0
1	0.7	-0.35	0
0	0.3	0	-0.35

Loss: -0.80/-3 = 0.27

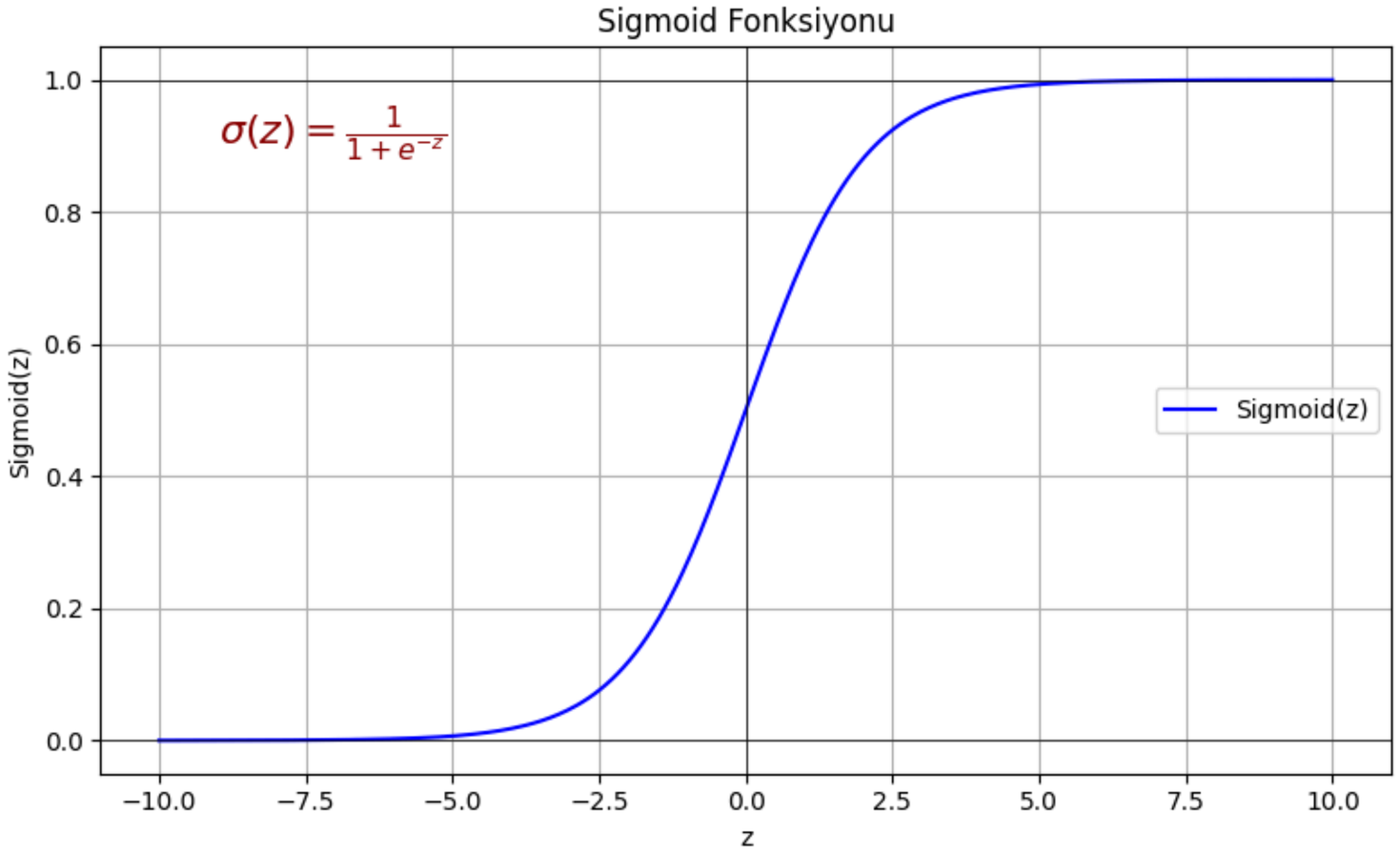
1.kısım $y_i \cdot \log(\hat{y}_i)$

2.kısım $(1 - y_i) \cdot \log(1 - \hat{y}_i)$

-1/n önemli ! hata pozitif olmalı

iç kısım her zaman negatif (Sigmoid)

Sigmoid



$$z = w_1x_1 + w_2x_2 + \dots + w_nx_n + b$$

w: weight

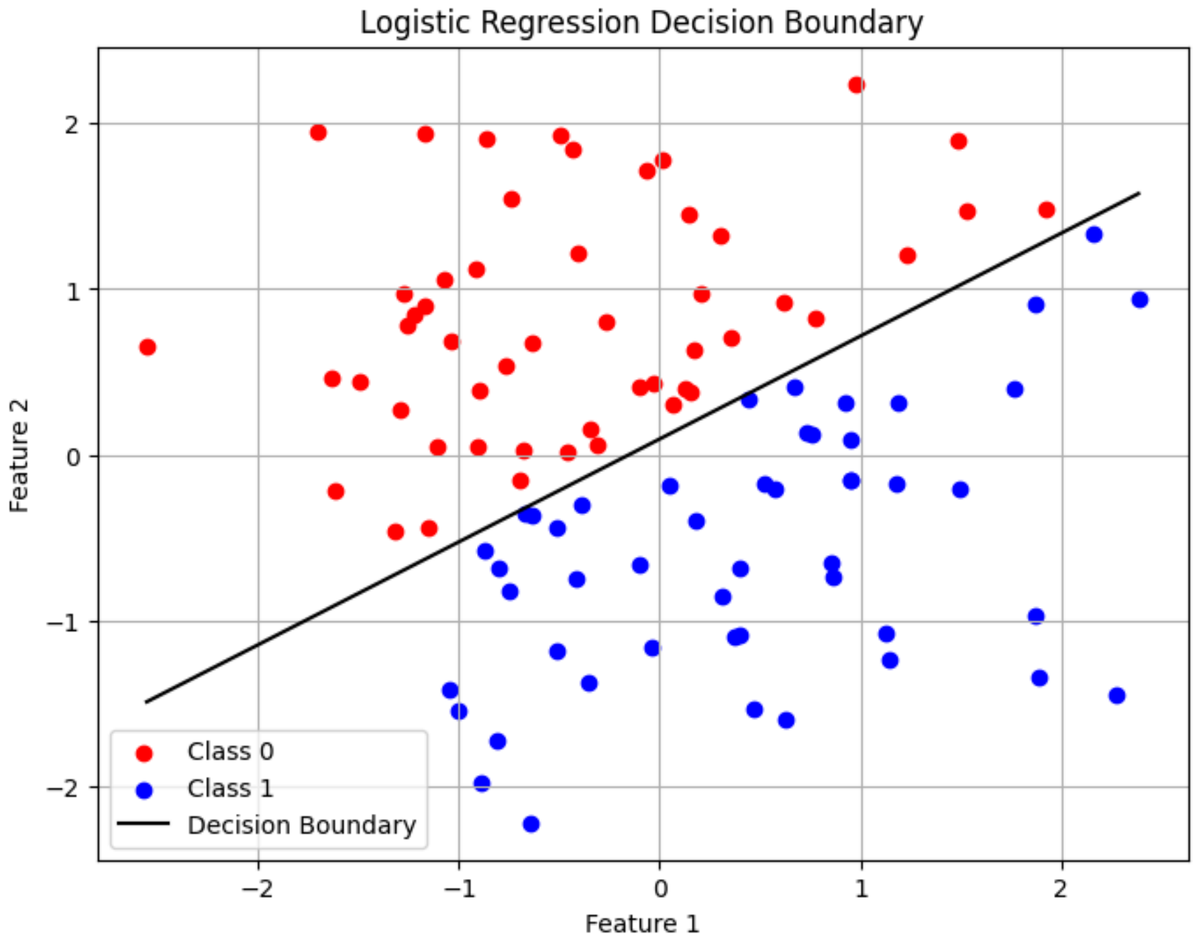
b: bias

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

Her zaman 0 ile 1 arasında çıktı

* Grafiği üreten python kodu ChatGPT tarafından hazırlanmıştır

Logistic Regression Prediction



$w_1 = 2.8$

$w_2 = -4.5$

$b = 0.4$

New input

$x_1 = 1$

$x_2 = 1.5$

$z = (2.8*1) + (-4.5 * 1.5) + 0.4 = -3.55$

$y' = 1/(1+e^{3.55}) = 0.02$

$y' = 0.02 < 0.5 \rightarrow$ Class 0

Matrix ile tahmin etmek için: sigmoid(x.W + b)

* Grafiği üreten python kodu ChatGPT tarafından hazırlanmıştır

Decision Boundry Belirleme

Pozitif sınıfları gözden kaçırma ihtimali

> %50

%50 → bir çok durumda ideal

Pozitif sınıfları daha rahat tespit etme

< %50

Boundry uğraşılan proje özelinde değerlendirilmesi gereken bir durumdur
Projenin detaylarının bilinmesi ve mantıksal çıkarımlar yapılması gerekir
***Training/Validating ile ilgili olan modülde daha detaylı bahsedilecek**

Grad Descent for Logistic Regression

Gradient Descent

Repeat until converge {

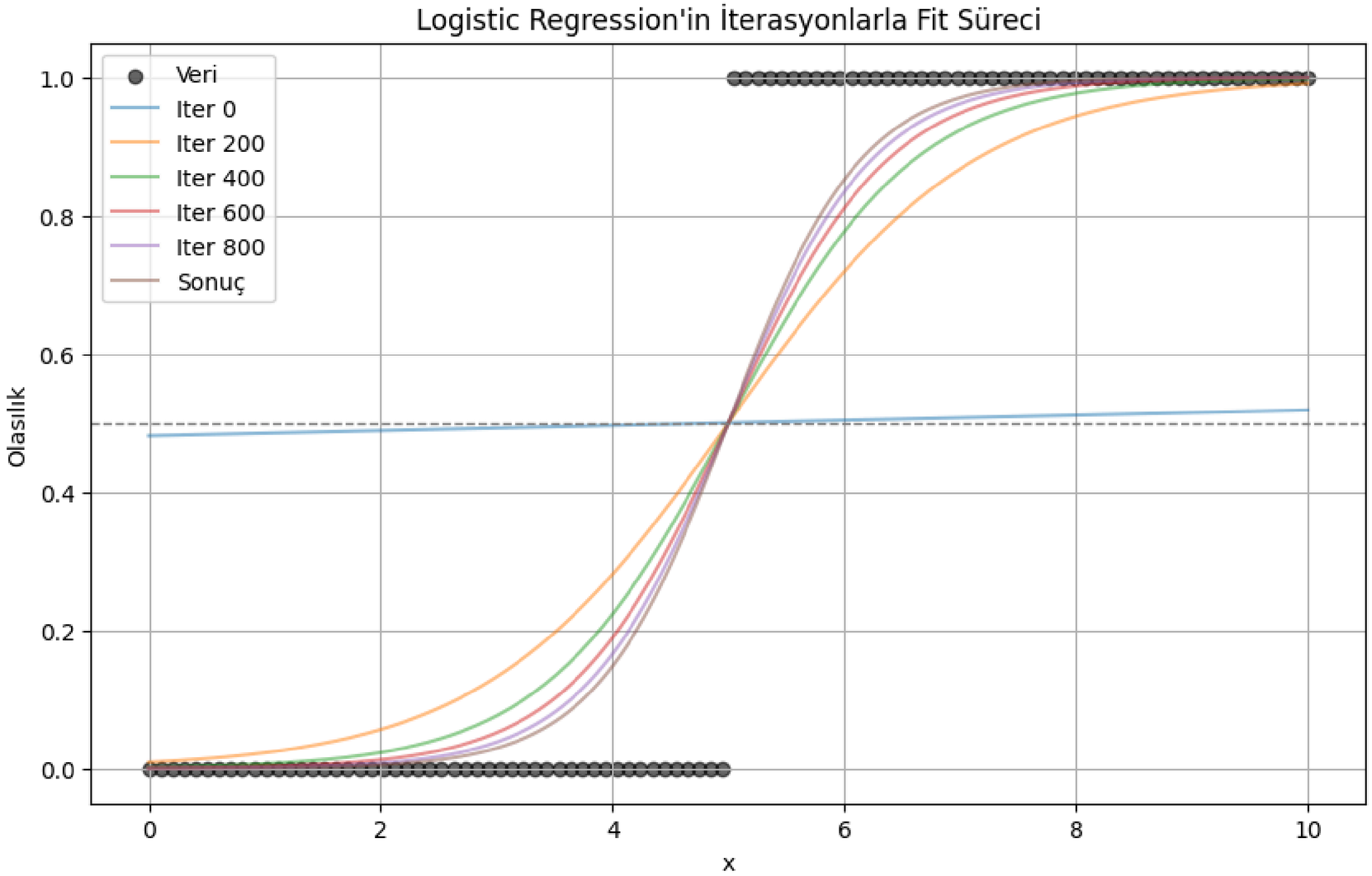
$$w = w - \alpha \left[\frac{\partial Loss}{\partial w}\right]$$

$$b = b - \alpha \left[\frac{\partial Loss}{\partial b}\right]$$

}

***Loss function değişti**

Grad Descent Nasıl görünüyor



* Grafiği üreten python kodu ChatGPT tarafından hazırlanmıştır

Partial Derivation of BCE

https://stats.stackexchange.com/questions/278771/how-is-the-cost-function-from-logistic-regression-differentiated

or

How is the cost function from Logistic Regression differentiated from “Cross Validated”

Gradient Descent for Logistic Regression

$$w = w - \frac{\alpha}{n} \sum [\sigma(z) - y] \cdot x$$

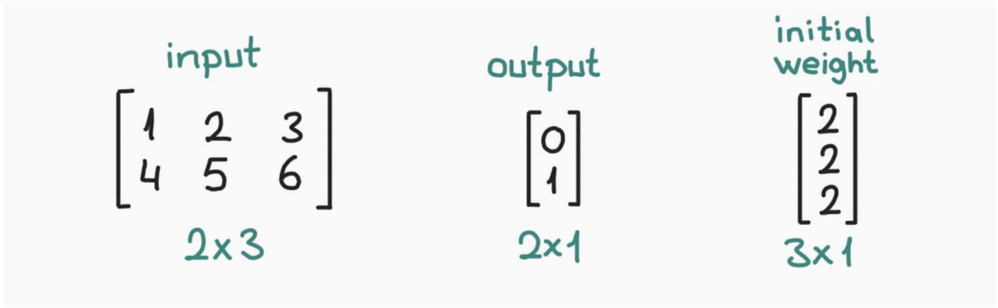
$$b = b - \frac{\alpha}{n} \sum [\sigma(z) - y]$$

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

σ: Sigmoid

$z = w_1 \cdot x_1 + \cdots + w_m x_m + b$

Gradient Descent for Logistic Regression



For w_1

$z = (1*2) + (2*2) + (3*2) = 12$

$y' = \text{sigmoid}(12) = 0.99$

$\text{Error (E)} = 0.99 - 0 = 0.99$

$\text{Gradient} = 0.99*1 = 0.99$

$z = (4*2) + (5*2) + (6*2) = 30$

$y' = \text{sigmoid}(30) = 1$

$\text{Error (E)} = 1-1 = 0$

$\text{Gradient} = 0*4 = 0$

$w_1 = 2 - (0.1)*(1/2)*(0+0.99) = 1.95$

Alpha, 1/n, Partial Derivation

For w_2

$z = (1*2) + (2*2) + (3*2) = 12$

$y' = \text{sigmoid}(12) = 0.99$

$\text{Error (E)} = 0.99 - 0 = 0.99$

$\text{Gradient} = 0.99*2 = 1.98$

$z = (4*2) + (5*2) + (6*2) = 30$

$y' = \text{sigmoid}(30) = 1$

$\text{Error (E)} = 1-1 = 0$

$\text{Gradient} = 0*5 = 0$

$w_2 = 2 - (0.1)*(1/2)*(0+1.98) = 1.90$

Alpha, 1/n, Partial Derivation

For w_3

$z = (1*2) + (2*2) + (3*2) = 12$

$y' = \text{sigmoid}(12) = 0.99$

$\text{Error (E)} = 0.99 - 0 = 0.99$

$\text{Gradient} = 0.99*3 = 2.97$

$z = (4*2) + (5*2) + (6*2) = 30$

$y' = \text{sigmoid}(30) = 1$

$\text{Error (E)} = 1-1 = 0$

$\text{Gradient} = 0*6 = 0$

$w_3 = 2 - (0.1)*(1/2)*(0+2.97) = 1.85$

Alpha, 1/n, Partial Derivation

Gradient Descent Matrix gösterimi

input

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

2x3

output

$$\begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

2x1

initial weight

$$\begin{bmatrix} 2 \\ 2 \\ 2 \end{bmatrix}$$

3x1

$$z = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 12 \\ 30 \end{bmatrix} \quad x \cdot w + b$$

$$\hat{y} = \text{sigmoid}\left(\begin{bmatrix} 12 \\ 30 \end{bmatrix}\right) = \begin{bmatrix} 0.99 \\ 1 \end{bmatrix} \quad \sigma(z)$$

$$\text{error}(E) = \begin{bmatrix} 0.99 \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.99 \\ 0 \end{bmatrix} \quad \hat{y} - y$$

$$\text{grad} = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix} \cdot \begin{bmatrix} 0.99 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.99 \\ 1.98 \\ 2.97 \end{bmatrix} \quad x^T \cdot (\hat{y} - y)$$

$$\text{new weight} = \begin{bmatrix} 2 \\ 2 \\ 2 \end{bmatrix} - \begin{bmatrix} 0.049 \\ 0.099 \\ 0.149 \end{bmatrix} = \begin{bmatrix} 1.95 \\ 1.9 \\ 1.85 \end{bmatrix} \quad w = w - \frac{\alpha}{n} \cdot \frac{\partial Loss}{\partial w}$$

$(\alpha = 0.1)$

BCE Before: 6.4 BCE After: 5.6

Batching

Instance based (online) Batch G. Descent → Verileri tek tek algoritmadan geçirmek
(Full) Batch Gradient Descent → Verileri toplu bir şekilde algoritmadan geçirmek
Mini batch gradient Descent → Verileri ufak kümeler halinde algoritmadan geçirmek

*Online batching paralel işlemler için yeterince uygun değildir. Eğitim çok uzun sürebilir
*Full batching büyük veri setleri için uygun değildir. RAM ve GPU RAM dolmasına sebep olabilir.
*Günümüzde en sık kullanılan mini batch (16-32-64-128) yöntemidir.

Batch size → Bir batch içerisinde kaç tane veri var

İlerisi için

Birden fazla sınıf olursa → Preprocess ve neural network modülleri
Algoritma değerlendirme/iyileştirme → Validation modülü
Veriyi nasıl işlicez → Preprocess modülü