

Machine Learning

Linear Regression with multiple variables

Multiple features

Multiple features (variables).

Size (feet ²)	Price (\$1000)
x	y
2104	460
1416	232
1534	315
852	178
...	...

$$h_{\theta}(x) = \theta_0 + \theta_1 x$$

Multiple features (variables).

Size (feet ²)	Number of bedrooms	Number of floors	Age of home (years)	Price (\$1000)	
x_1	x_2	x_3	x_4	y	
2104	5	1	45	460	
1416	3	2	40	232	
1534	3	2	30	315	
852	2	1	36	178	
...	

M=47

Notation:

n=4

n = number of features

$x^{(i)}$ = input (features) of i^{th} training example.

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

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

$$x_3^{(2)} = 2$$

Hypothesis:

Previously: $h_{\theta}(x) = \theta_0 + \theta_1 x$

Now: $h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$

E.g. $h_{\theta}(x) = 80 + 0.1x_1 + 0.02x_2 + 3x_3 - 2x_4$

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

For convenience of notation, define $x_0 = 1$. $\longrightarrow x_0^{(i)} = 1$

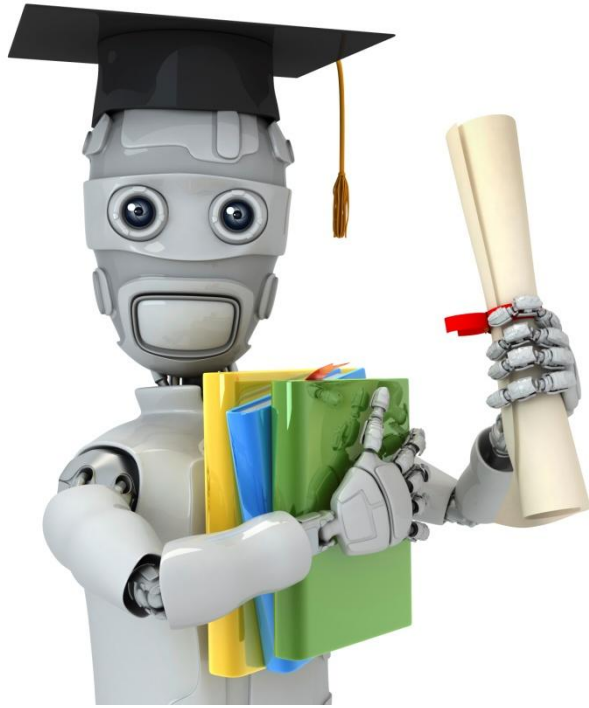
$$X = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in R^{n+1}$$

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \in R^{n+1}$$

$$\begin{aligned} h_{\theta}(x) &= \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n \\ &= \boldsymbol{\theta}^T \mathbf{X} \end{aligned}$$

$$\underbrace{[\theta_0, \theta_1, \dots, \theta_n]}_{\boldsymbol{\theta}^T} \underbrace{\begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}}_{\mathbf{X}}$$

Multivariate linear regression.



Machine Learning

Linear Regression with multiple variables

Gradient descent for multiple variables

Hypothesis: $h_{\theta}(x) = \theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$

Parameters: $\theta_0, \theta_1, \dots, \theta_n$

Cost function:

$$J(\theta_0, \theta_1, \dots, \theta_n) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

Gradient descent:

Repeat {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \dots, \theta_n)$$

}

(simultaneously update for every $j = 0, \dots, n$)

Gradient Descent

Previously ($n=1$):

Repeat {

$$\theta_0 := \theta_0 - \underbrace{\alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})}_{\frac{\partial}{\partial \theta_0} J(\theta)}$$

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

(simultaneously update θ_0, θ_1)

}

New algorithm ($n \geq 1$):

Repeat {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_0} J(\theta) \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

(simultaneously update θ_j for $j = 0, \dots, n$)

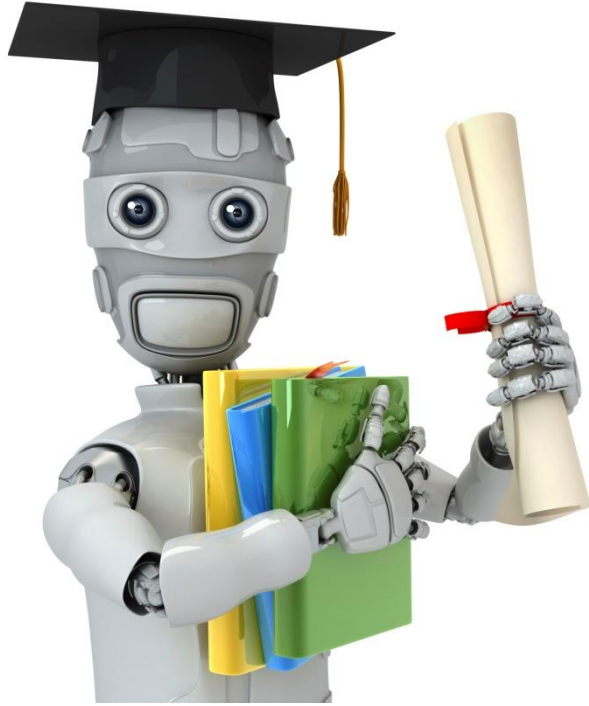
}

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

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

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

...



Machine Learning

Linear Regression with multiple variables

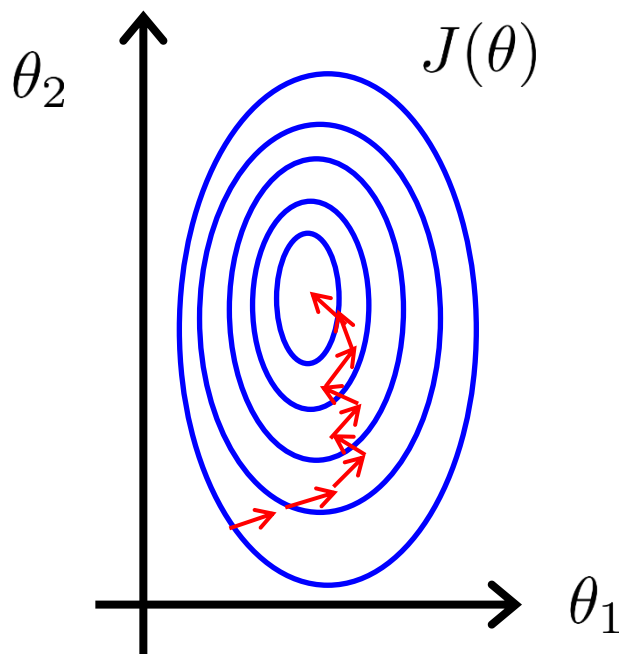
Gradient descent in practice I: Feature Scaling

Feature Scaling

Idea: Make sure features are on a similar scale.

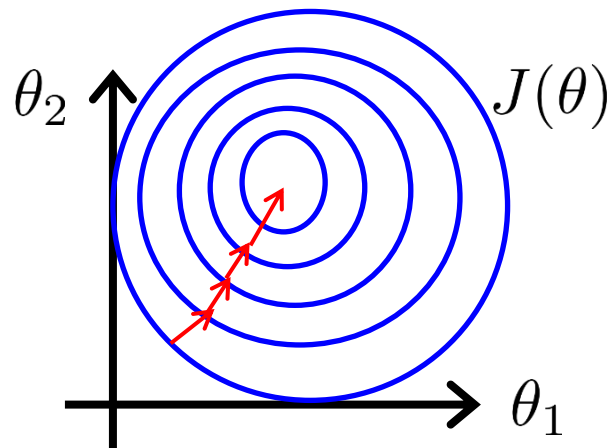
E.g. $x_1 = \text{size (0-2000 feet}^2\text{)}$

$x_2 = \text{number of bedrooms (1-5)}$



$$x_1 = \frac{\text{size (feet}^2\text{)}}{2000}$$

$$x_2 = \frac{\text{number of bedrooms}}{5}$$



Feature Scaling

Get every feature into approximately a $-1 \leq x_i \leq 1$ range.

$$0 \leq x_1 \leq 3 \quad \checkmark$$

$$-2 \leq x_2 \leq 0.5 \quad \checkmark$$

$$-100 \leq x_3 \leq 100 \quad \times$$

$$-0.0001 \leq x_4 \leq 0.0001 \quad \times$$

Ok

$$-\frac{1}{3} \leq x_i \leq \frac{1}{3}$$

$$-5 \leq x_i \leq 5$$

Mean normalization

Replace x_i with $x_i - \mu_i$ to make features have approximately zero mean
(Do not apply to $x_0 = 1$).

E.g. $x_1 = \frac{size - 1000}{2000}$

$$x_2 = \frac{\#bedrooms - 2}{5}$$

Average
value of x_i in
training set

other methods:

$$x = (x - \min) / (\max - \min)$$

$$-0.5 \leq x_1 \leq 0.5, -0.5 \leq x_2 \leq 0.5$$

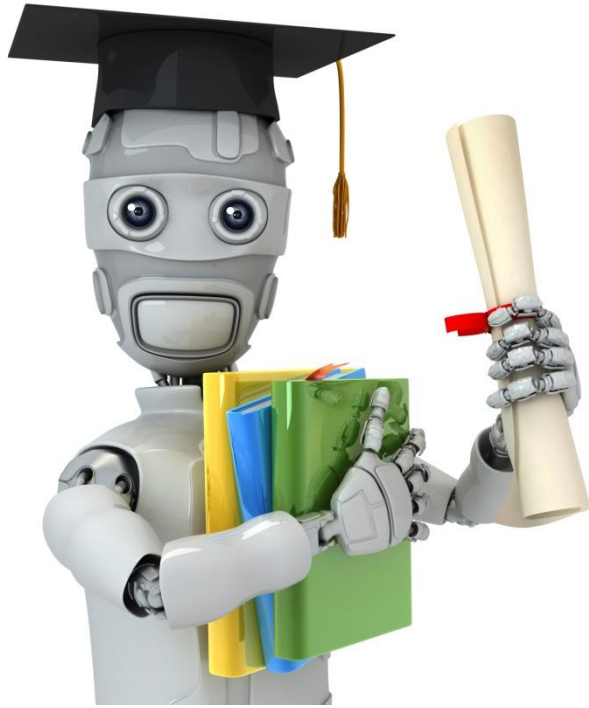
More general rule:

$$x_i = \frac{x_i - u_i}{s_i}$$

u_i is the average value of x_i in training set

s_i is the range of x_i , that is
maximum value of x_i - minimum value of x_i

s_i also can be the standard deviation



Machine Learning

Linear Regression with multiple variables

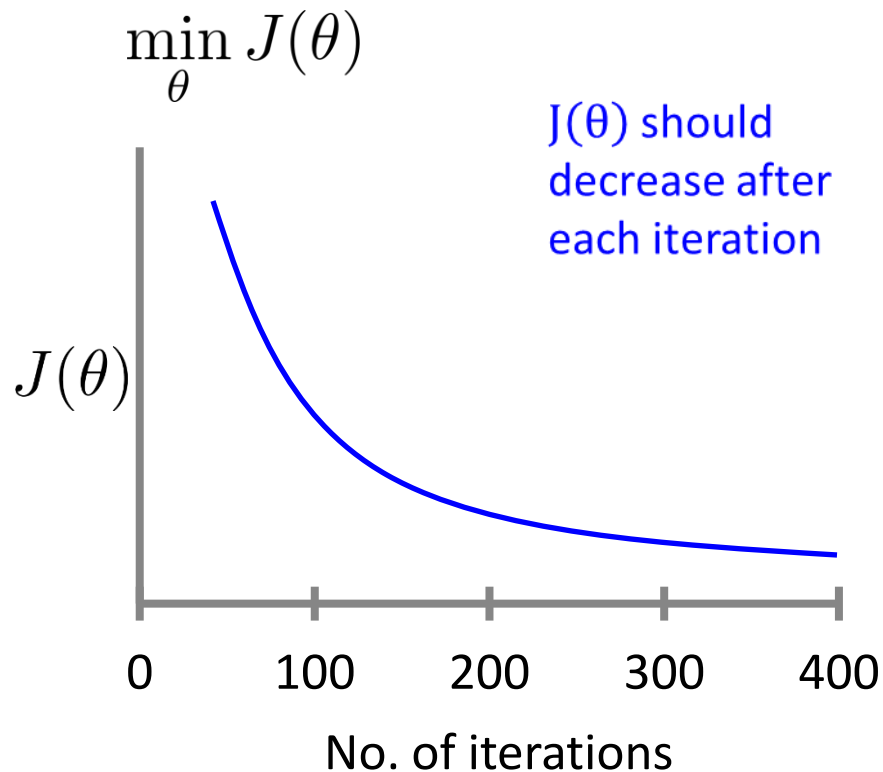
Gradient descent in practice II: Learning rate

Gradient descent

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

- “Debugging”: How to make sure gradient descent is working correctly.
- How to choose learning rate α .

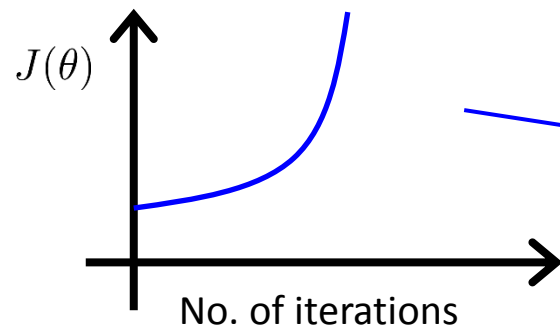
Making sure gradient descent is working correctly.



Example automatic convergence test:

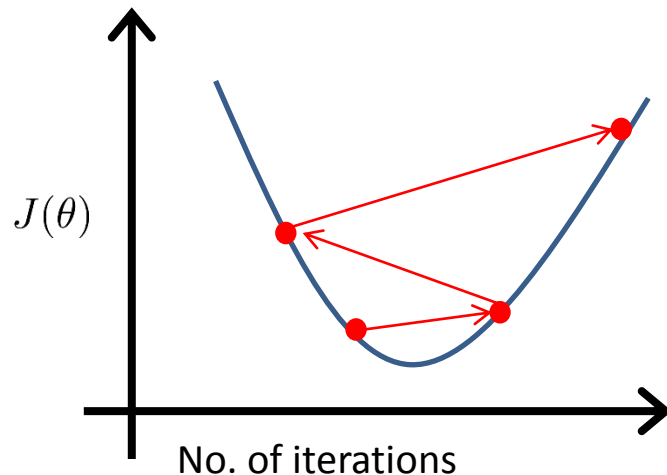
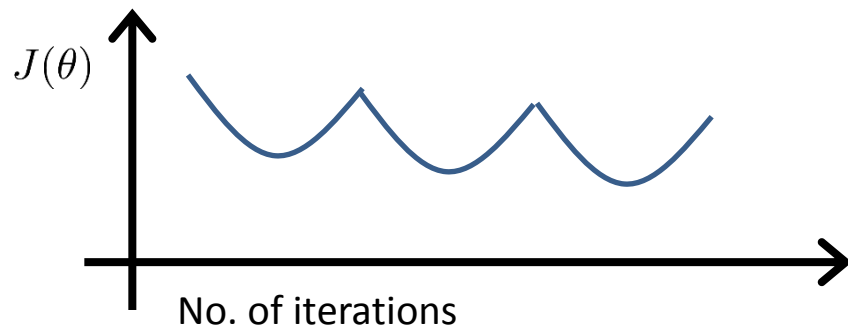
Declare convergence if $J(\theta)$ decreases by less than 10^{-3} in one iteration.

Making sure gradient descent is working correctly.



Gradient descent not working.

Use smaller α .



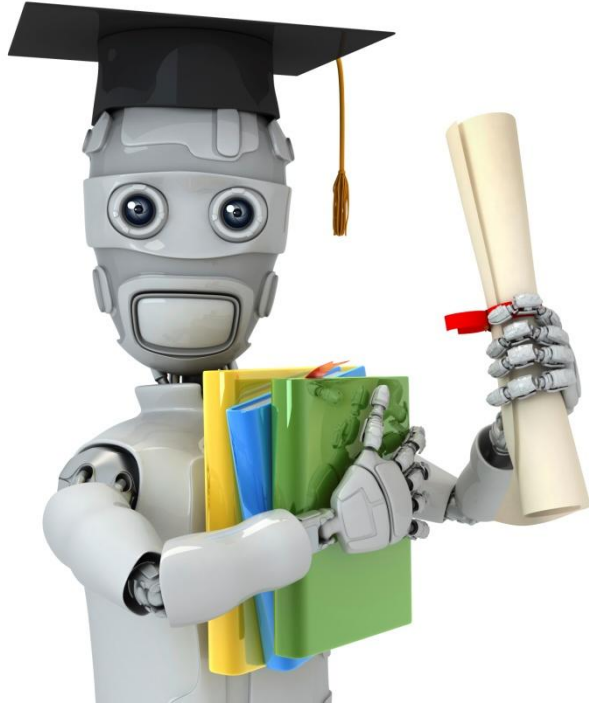
- For sufficiently small α , $J(\theta)$ should decrease on every iteration.
- But if α is too small, gradient descent can be slow to converge.

Summary:

- If α is too small: slow convergence.
- If α is too large: $J(\theta)$ may not decrease on every iteration; may not converge.

To choose α , try

$\dots, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, \dots$

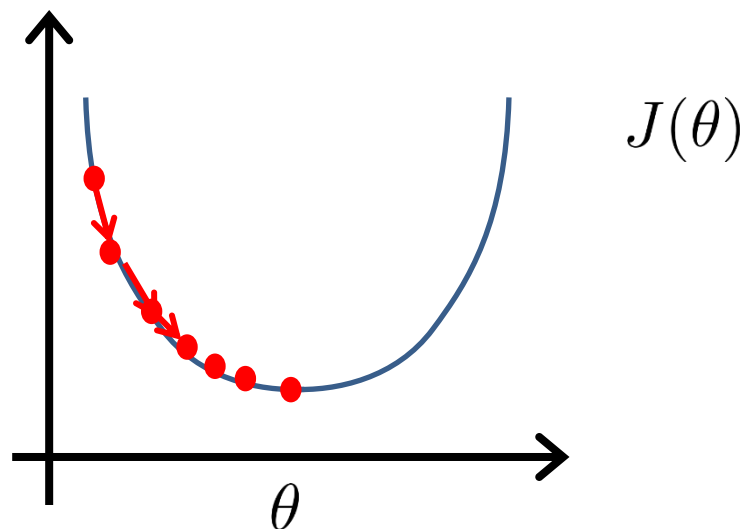


Machine Learning

Linear Regression with multiple variables

Normal equation

Gradient Descent



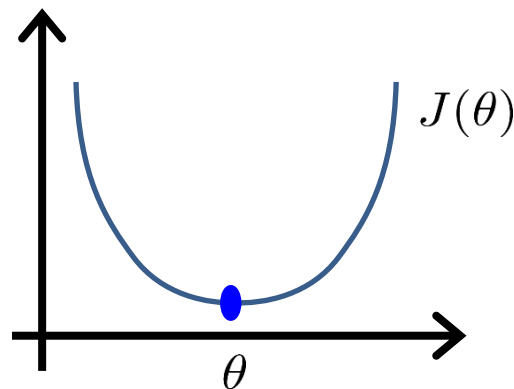
Normal equation: Method to solve for θ analytically.

Intuition: If 1D ($\theta \in \mathbb{R}$)

$$J(\theta) = a\theta^2 + b\theta + c$$

$$\frac{d}{d\theta}J(\theta) = \dots \stackrel{\text{set}}{=} 0$$

Solved for θ



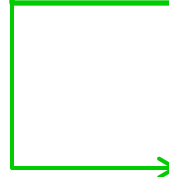
$$\theta \in \mathbb{R}^{n+1} \quad J(\theta_0, \theta_1, \dots, \theta_m) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

$$\frac{\partial}{\partial \theta_j} J(\theta) = \dots = 0 \quad (\text{for every } j)$$

Solve for $\theta_0, \theta_1, \dots, \theta_n$

Examples: $m = 4$.

	Size (feet ²)	Number of bedrooms	Number of floors	Age of home (years)	Price (\$1000)
x_0	x_1	x_2	x_3	x_4	y
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178


$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix} \quad y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$$

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

m **examples** $(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})$; n **features**.

$$x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} \in \mathbb{R}^{n+1}$$

$$\underset{\text{(Design matrix)}}{X} = \begin{bmatrix} (x^{(1)})^T \\ (x^{(2)})^T \\ \vdots \\ (x^{(m)})^T \end{bmatrix} \in \mathbb{R}^{m \times (n+1)}$$

E.g. If $x^{(i)} = \begin{bmatrix} 1 \\ x_1^{(i)} \end{bmatrix}$

$$X = \begin{bmatrix} 1 & x_1^{(1)} \\ 1 & x_1^{(2)} \\ \vdots & \vdots \\ 1 & x_1^{(m)} \end{bmatrix} \quad y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

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

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

is inverse of matrix $X^T X$.

Octave: $\underbrace{\text{pinv}(X' * X)}_{(X^T X)^{-1}} * X' * y$

For normal equation
Feature scaling is
not necessary.

m training examples, n features.

Gradient Descent

- Need to choose α .
- Needs many iterations.
- Works well even when n is large.

Normal Equation

- No need to choose α .
- Don't need to iterate.
- Need to compute $(X^T X)^{-1}$ $O(n^3)$
- Slow if n is very large.

Issue: numerical stability

