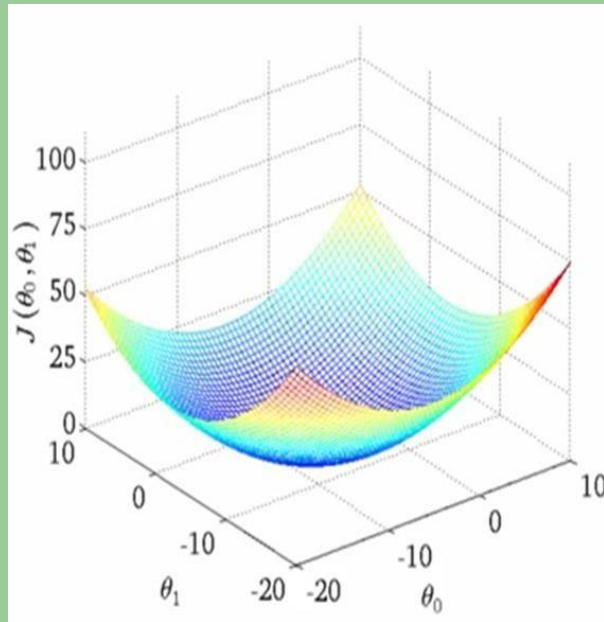


Machine Learning with Python



Lecture 3



**Gradient Descent
Normal Equation
Features Scaling**

If we have n features

$$h(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_j x_j \quad \text{Where } x_0 = 1$$

Let's simplify

$$h(x) = \sum_{j=0}^n \theta_j x_j$$

n = # of features
 $j = j^{th}$, counter

The job of the learning algorithm is to find values of θ_n such that $h(x) \approx y$

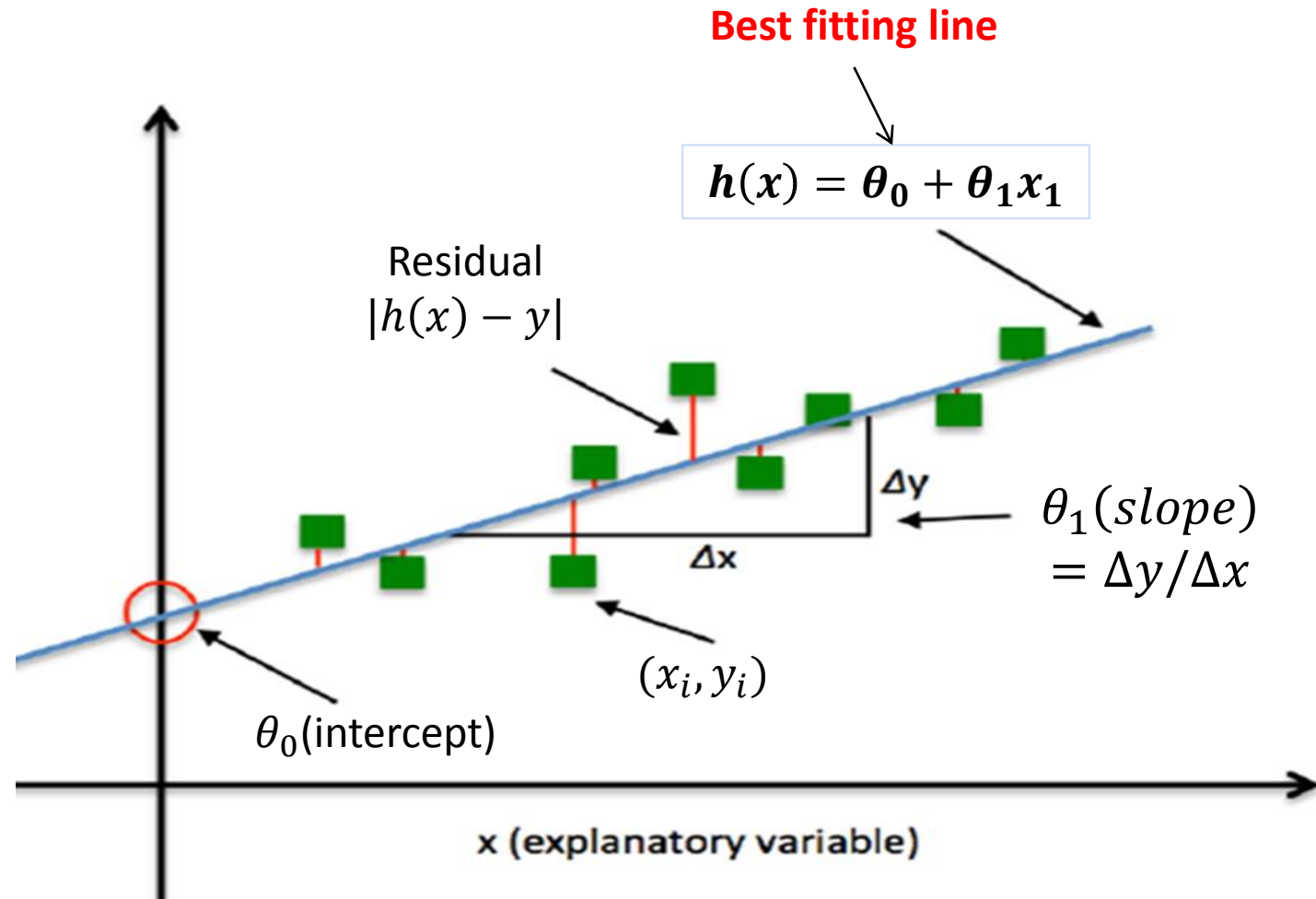
Cost Function

- Least Squares (LS)

$$\text{LS} = \min \sum_{i=1}^m (h(x^{(i)}) - y^{(i)})^2$$

This cost function is called Ordinary Least Squares, and defined as follows:

$$j(\theta) = \frac{1}{2m} \sum_{i=1}^m (h(x^{(i)}) - y^{(i)})^2$$



The **goal** of learning is to **minimize** the cost function $j(\theta)$.

Cost Function (also known as a **Loss Function**)

$$j(\theta) = \frac{1}{2m} \sum_{i=1}^m (h(x^{(i)}) - y^{(i)})^2$$

- What is the optimal value of θ_1 that minimizes $J(\theta_1)$?
- How to find the best value for θ_1 ?

To minimize $j(\theta)$, we use an **optimization algorithm** called **Gradient Descent**.

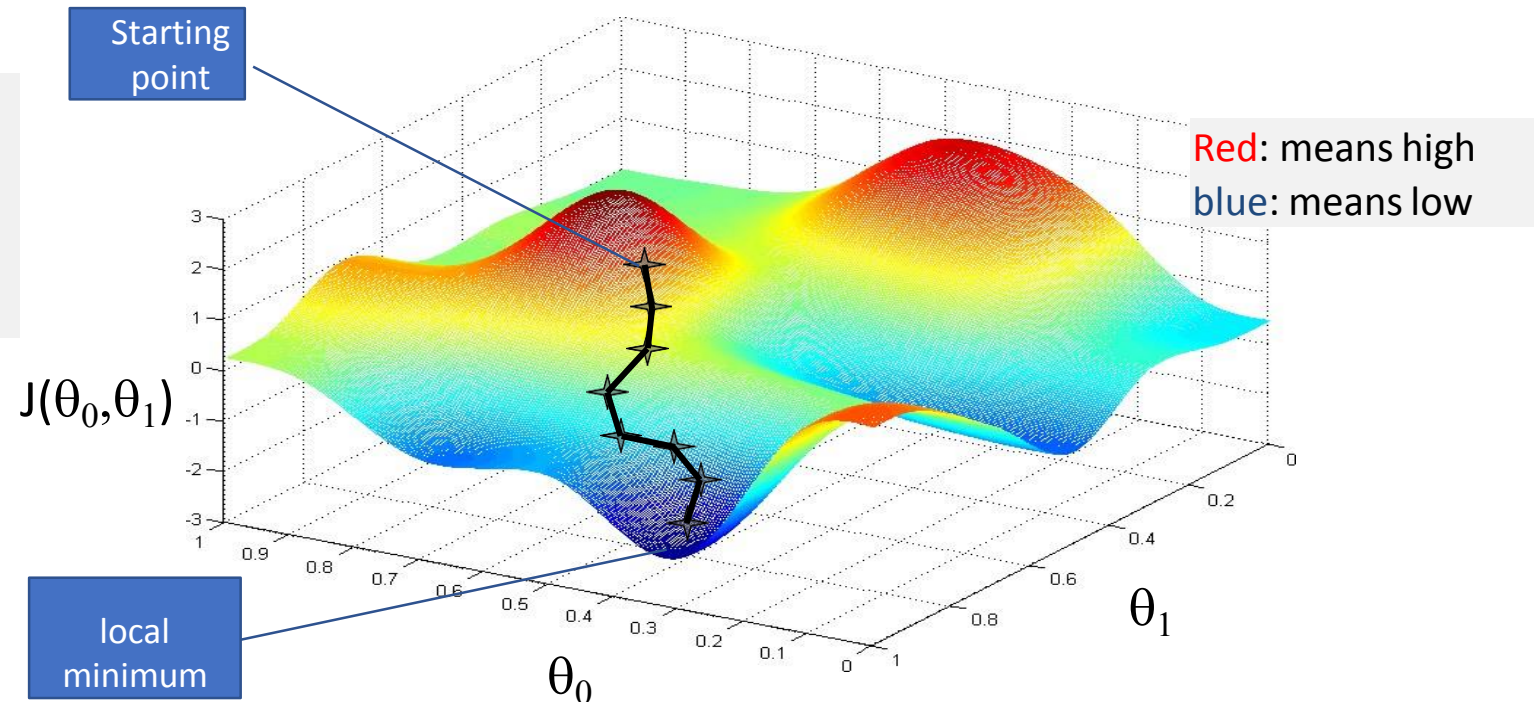
Gradient descent optimization algorithm for finding a local minimum (optimal values for parameters (coefficients)) of a **differentiable function**.

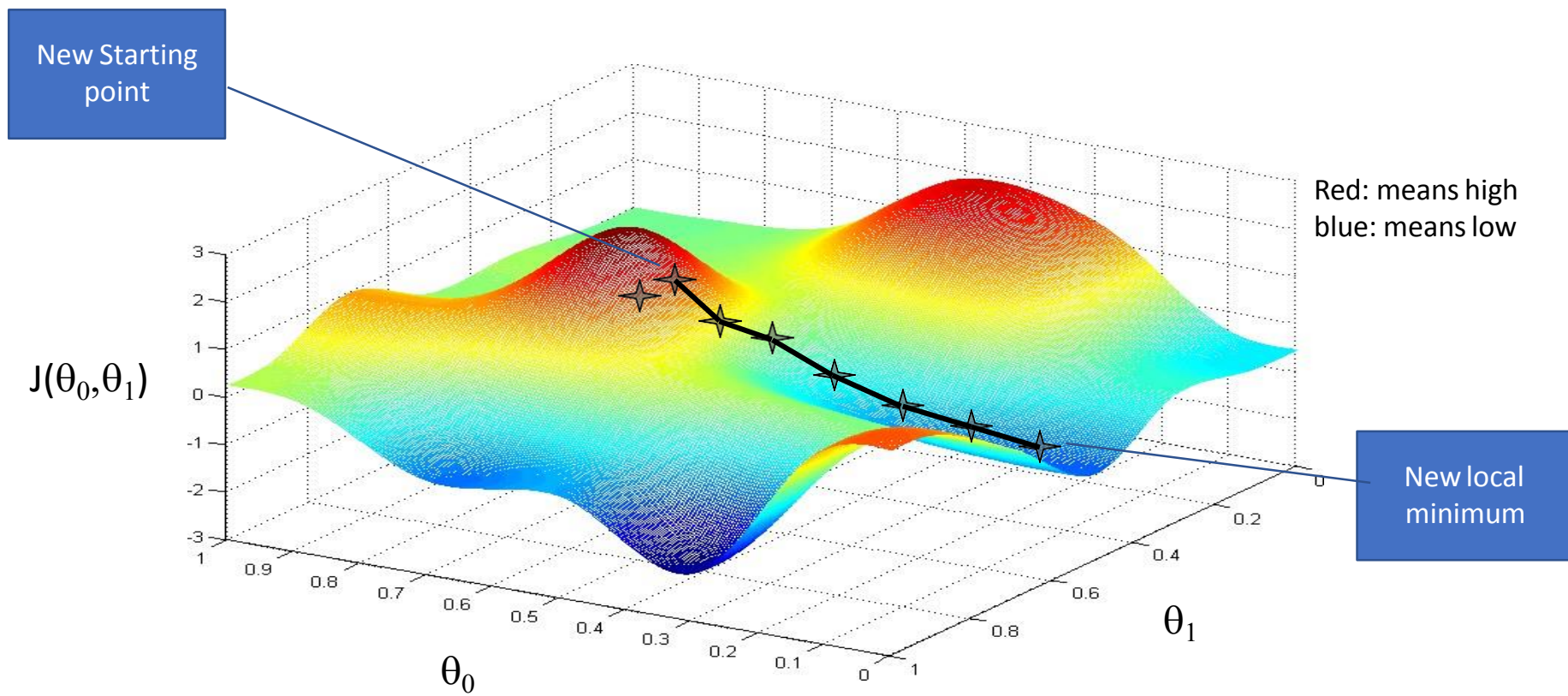
Gradient Descent for Machine Learning

Gradient Descent is used in machine learning to find the values of a function's parameters (coefficients θ s) that minimize a **cost function** as far as possible.

$$j(\theta) = \frac{1}{2m} \sum_{i=1}^m (h(x^{(i)}) - y^{(i)})^2$$

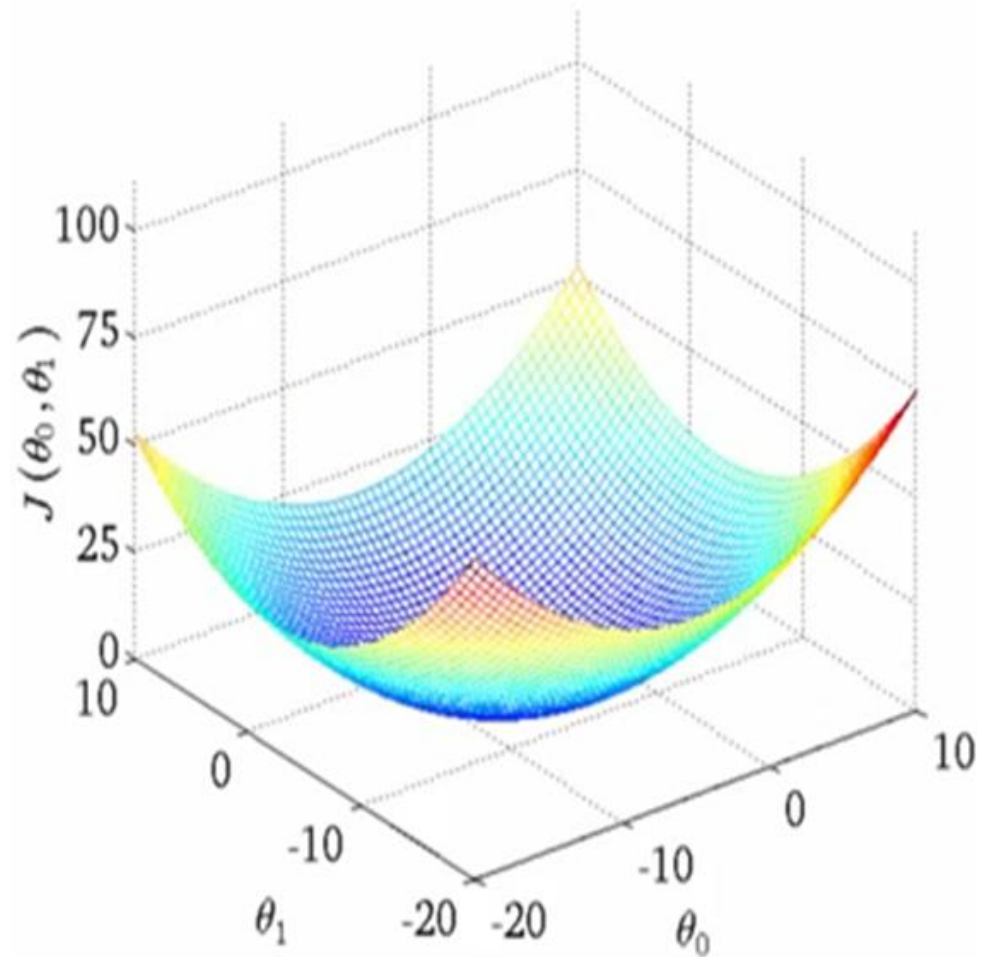
- Imagine that this is a landscape of grassy park, and you want to go to the lowest point in the park as rapidly as possible



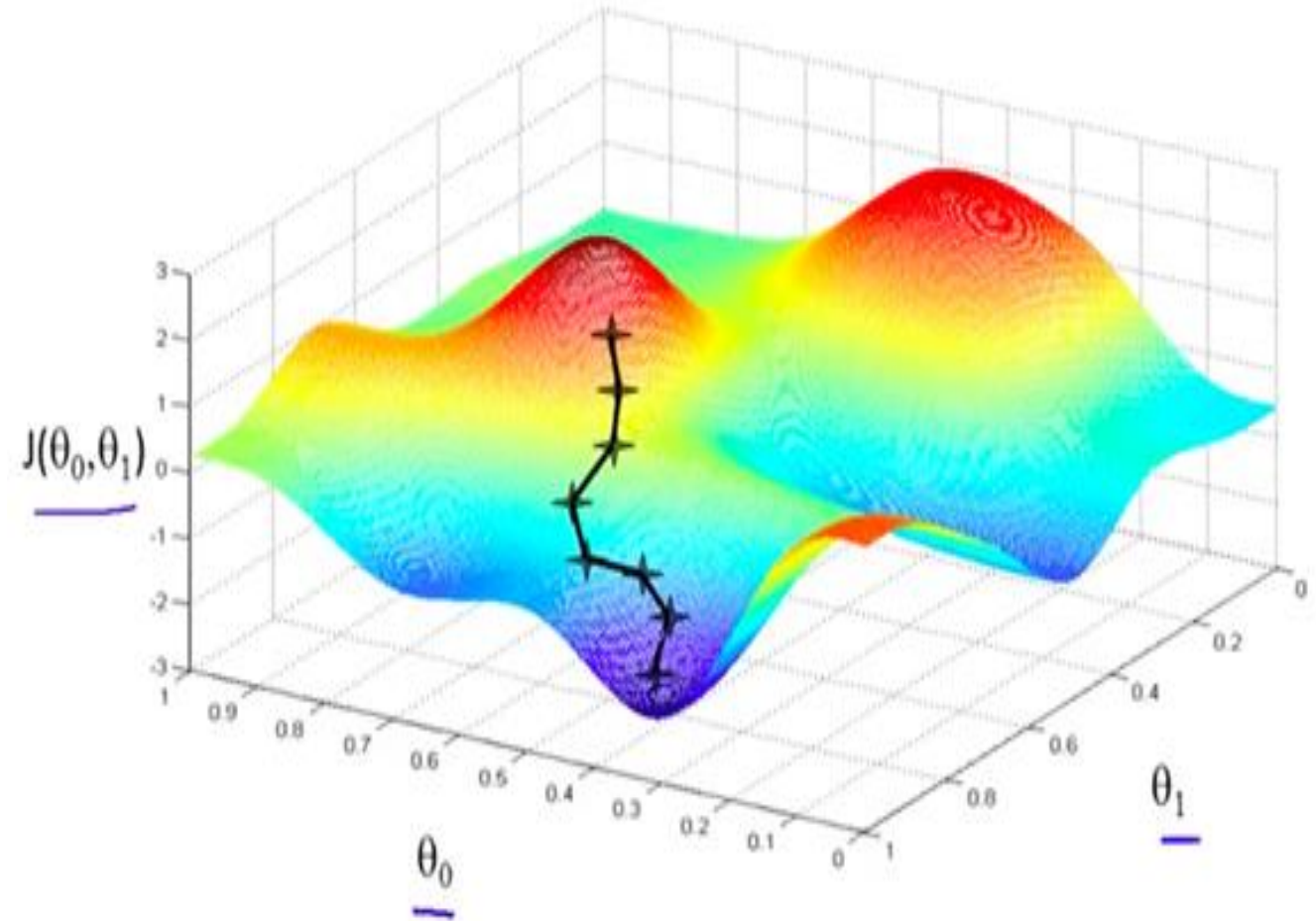


With different starting point

Convex Loss Function



Non-Convex Loss Function



Linear regression with one variable

Hypothesis:

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

Parameters:

$$\theta_0, \theta_1$$

Cost Function:

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

Goal: minimize $J(\theta_0, \theta_1)$
 θ_0, θ_1

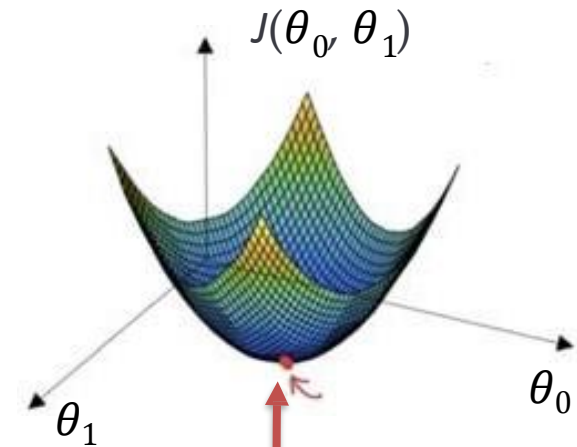
How Gradient Descent works

$$h(x) = \theta_0 x_0 + \theta_1 x_1$$

Imagine you have a machine learning problem and want to train your algorithm with **gradient descent** to minimize your cost-function $J(\theta_0, \theta_1)$ and reach its local minimum by tweaking its parameters (θ_0 and θ_1).

The horizontal axes represent the parameters (θ_0, θ_1), while the **cost function** $J(\theta_0, \theta_1)$ is represented on the vertical axes.

How we can find the optimal values of θ_0 and θ_1 that correspond to the minimum of the cost function

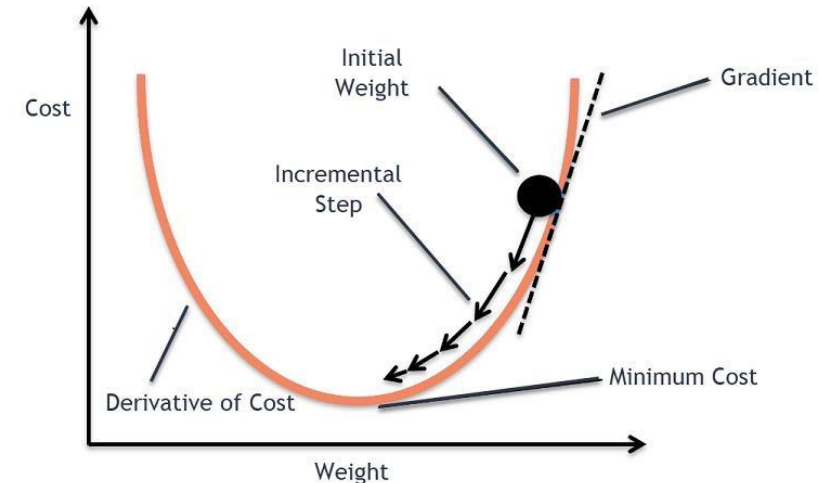
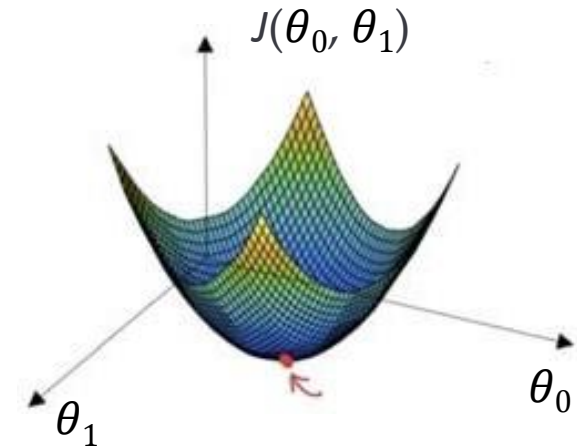


How Gradient Descent works

How we can find the optimal values of θ_0 and θ_1 that correspond to the minimum of the cost function

To find the values of θ_0 and θ_1 that correspond to the minimum of the cost function (marked with the red arrow).

- Start by defining the initial parameter's values θ_0 and θ_1 with some random numbers.
- Gradient descent then starts at that point (somewhere around the top of our illustration), and it takes one step after another in the steepest downside direction until it reaches the point where the **cost function** is as small as possible.



PROBLEM SETUP

Have some function $J(\theta_0, \theta_1)$

Want $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$

Outline:

- Start with some θ_0, θ_1
- Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$
until we hopefully end up at a minimum

Gradient Descent

It is iterative process and used to update the parameters of the model.

Learning-based Gradient Descent steps:

1. Initialize random variables to θ .
2. Keep change θ to minimize $j(\theta)$.

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

Where,

$j = 0, 1, \dots, n$

γ : Learning rate = very small number

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h(x^{(i)}) - y^{(i)})^2$$

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{\partial}{\partial \theta_j} \frac{1}{2} (h(x) - y)^2$$

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{\partial}{\partial \theta_j} \frac{1}{2} (h(x) - y)^2$$

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{2} \cdot 2 (h(x) - y) \cdot \frac{\partial}{\partial \theta_j} (h(x) - y)$$

$$h(x) = (\theta_0 x_0 + \theta_1 x_1 + \dots + \theta_n x_n)$$

$$\frac{\partial}{\partial \theta_j} J(\theta) = (h(x) - y) \cdot \frac{\partial}{\partial \theta_j} (\theta_0 x_0 + \theta_1 x_1 + \dots + \theta_n x_n - y)$$

$$\frac{\partial}{\partial \theta_j} J(\theta) = (h(x) - y) \cdot x_j$$

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^m (h(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$$

Partial derivative

الاشتقاق الجزئي

$$\frac{\partial}{\partial x} f(x)$$

$$f(x) = x^2 + t^3$$

$$\frac{\partial}{\partial x} f(x) = 2x + 3t^2 \cdot \frac{\partial}{\partial x} (t)$$

Gradient Descent Formalization

The Learning Rate (γ)

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

Initialized value



$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^m (h(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$$

Learning rate = very small number

The size of this step

$$\theta_{new} = ??$$

(Initialized
value)

$$\theta_{old} = 10$$

$$\frac{\partial}{\partial \theta_j} J(\theta) = 6$$

$$\gamma = 1$$

$$\begin{aligned} \theta_{new} &= 10 - 1 * 6 \\ &= 4 \end{aligned}$$

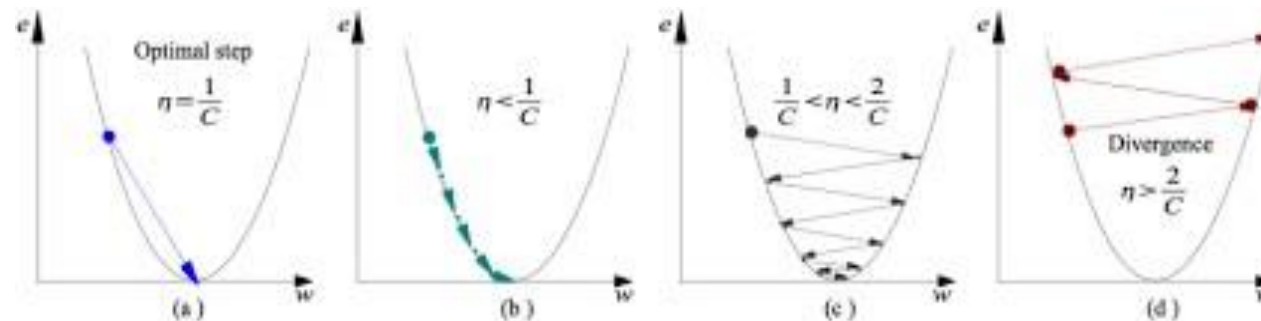
$$\gamma = 0.1$$

$$\begin{aligned} \theta_{new} &= 10 - 0.1(6) \\ &= 9.4 \end{aligned}$$

The Learning Rate

it must be chosen carefully to end up with local minima.

- If the learning rate is too high, we might **OVERSHOOT** the minima and keep bouncing, without reaching the minima
- If the learning rate is too small, the training might turn out to be too long (**more precise**, but it is time-consuming)



- Learning rate is optimal, model converges to the minimum
- Learning rate is too small, it **takes more time** but converges to the minimum
- Learning rate is higher than the optimal value, it **overshoots** but converges ($1/C < \eta < 2/C$)
- Learning rate is very large, it **overshoots** and diverges, moves away from the minima, performance decreases on learning

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

$$J(\theta) = (h(x) - y)^2$$

$$J(\theta) = (\theta x - 4)^2$$

Let's say $x = 1$

$$\begin{aligned} \text{Find } \frac{\partial}{\partial \theta_j} J(\theta) \\ = 2(\theta x - 4)(1) \end{aligned}$$

set $\gamma = 0.1$

#step1: Initialize random variable

$$\theta_1 = 6$$

#step 2: perform iteration

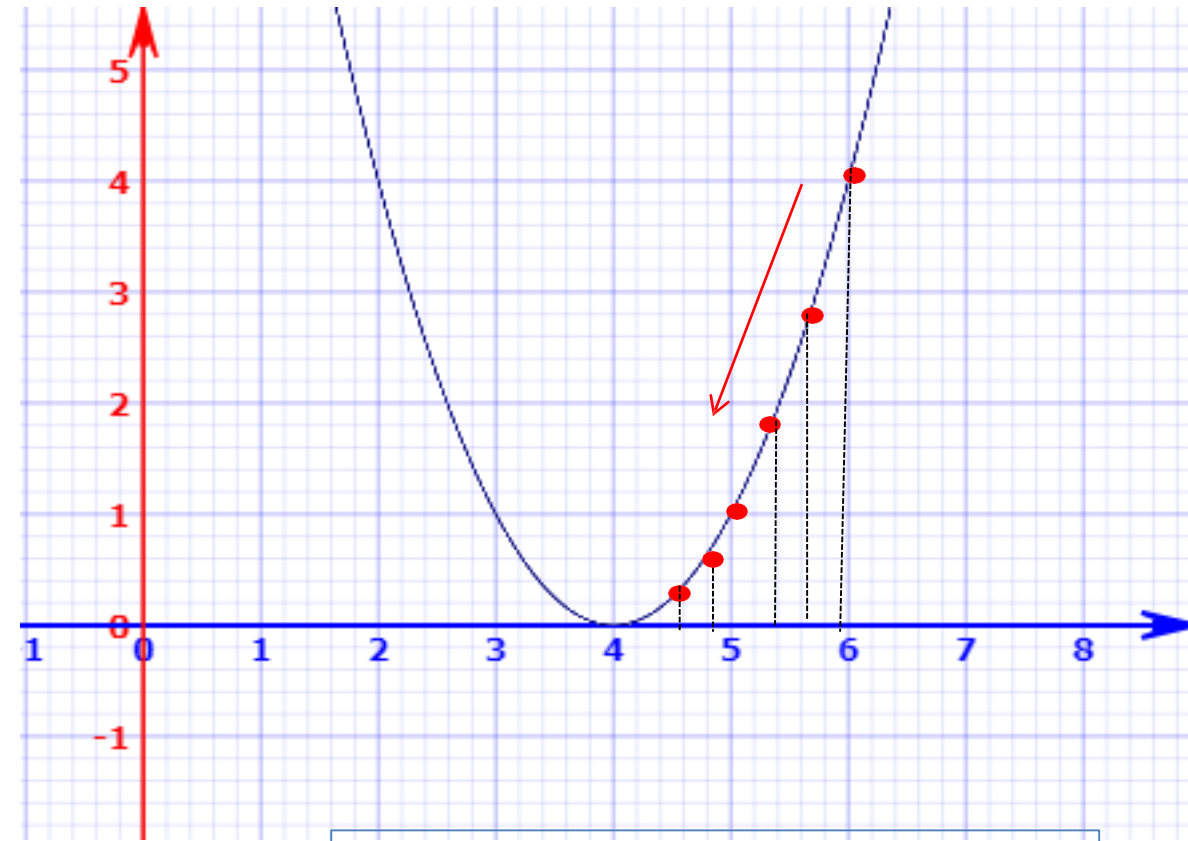
Iter1

$$\begin{aligned} \theta_1 &= 6 - 0.1 * (2(6 * 1 - 4)) \\ &= 6 - 0.1 * (4) \\ &= 6 - 0.4 \\ &= 5.6 \end{aligned}$$

θ_1	$J(\theta)$
6	4
5.6	2.56
5.48	2.19

iter2

$$\begin{aligned} \theta_1 &= 5.6 - 0.1 * (2(5.6 * 1 - 4)) \\ &= 5.6 - 0.1 * (1.2) \\ &= 5.6 - 0.12 \\ &= 5.48 \end{aligned}$$

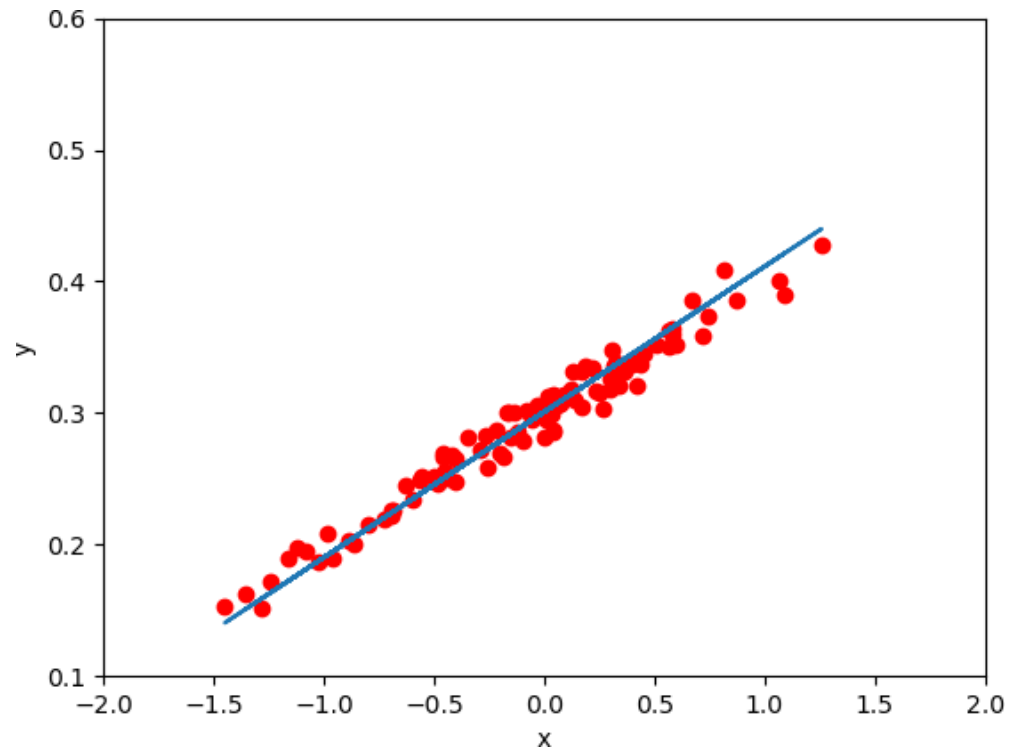


Repeat until convergence.

for $j = 0, 1, \dots, n$

Example after implement some iterations using gradient descent


Iteration



GRADIENT DESCENT FOR MULTIPLE VARIABLE

MULTI-VARIABLE

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

$x_1^{(4)} = 852$

$m=47$

Notation:

n = number of features $n=4$

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

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

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

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

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 {

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

$$\underbrace{\hspace{10em}}_{\frac{\partial}{\partial \theta_0} J(\theta)}$$

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

(simultaneously update θ_0, θ_1)

}

New algorithm ($n \geq 1$):

Repeat {

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

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

}

$$x_0 = 1$$

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

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

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

GRADIENT
DESCENT

V/S

NORMAL
EQUATION

Normal Equation

Gradient Descent

- ❑ In mathematics gradient descent (also often called steepest descent) is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function.
- ❑ The idea is to take repeated steps in the opposite direction of the gradient (or approximate gradient) of the function at the current point, because this is the direction of steepest descent. Conversely, stepping in the direction of the gradient will lead to a local maximum of that function; the procedure is then known as gradient ascent.

Normal Equation :

Normal Equation, an [analytical approach](#) used for optimization.

It is an alternative for Gradient descent. Normal equation performs minimization without iteration.

Normal equations are equations obtained by setting equal to zero the partial derivatives of the [sum of squared errors](#) or [cost function](#); normal equations allow one to estimate the parameters of multiple linear regression.

Normal equation

Another algorithm that solve for the optimal value of θ .

$$\frac{\partial}{\partial \theta_j} J(\theta) = 0$$

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

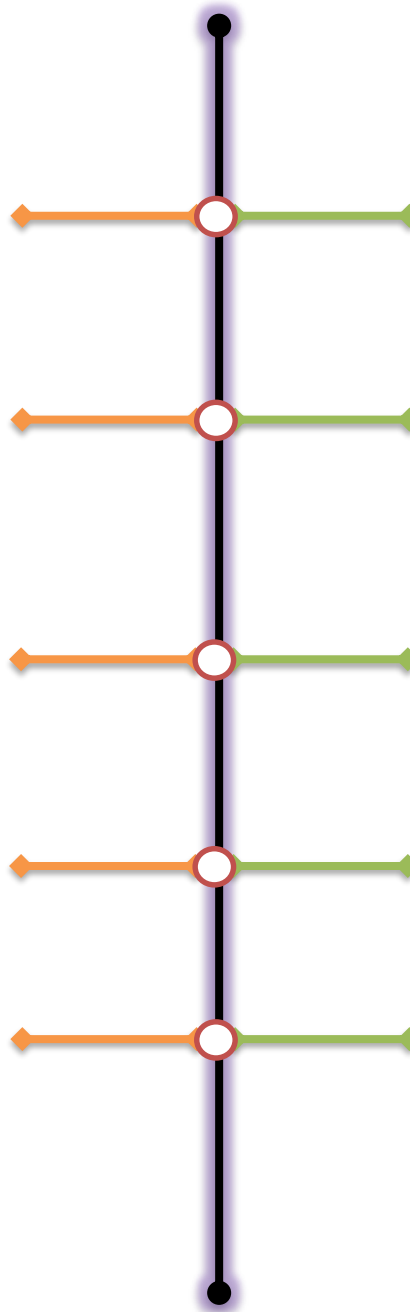
Where:

X = input features value

y = output value

One analytical step

No need to choose learning rate.



Gradient Descent

An optimization algorithm used to find the optimal values of θ .

$$\frac{\partial}{\partial \theta_j} J(\theta)$$

$$\theta_j := \theta_j - \gamma \sum_{i=1}^m (h(x^{(i)}) - y^{(i)}) . x_j^{(i)}$$

Iterative process

We need to choose learning rate.

Example:

$n = 4$

Size	Room#	Age	Floor	Price
90	3	3	1	50
120	5	15	3	70
150	4	10	2	100
200	6	5	4	150

$m = 4$

$$\mathbf{X} = \begin{matrix} & x_0 & x_1 & x_2 & x_3 & x_4 \\ \begin{bmatrix} 1 & 90 & 3 & 3 & 1 \\ 1 & 120 & 5 & 15 & 3 \\ 1 & 150 & 4 & 10 & 2 \\ 1 & 200 & 6 & 5 & 4 \end{bmatrix} & & & & \end{matrix} \quad \mathbf{y} = \begin{matrix} y \\ \begin{bmatrix} 50 \\ 70 \\ 100 \\ 150 \end{bmatrix} \end{matrix}$$

$m \times n$ m

Example:

Let's take the feature **Room#** to predict **Price**

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

Step1

$$X = \begin{matrix} & x_0 & x_1 \\ \begin{bmatrix} 1 & 3 \\ 1 & 5 \\ 1 & 4 \\ 1 & 6 \end{bmatrix} & y = \begin{bmatrix} 50 \\ 70 \\ 100 \\ 150 \end{bmatrix} \end{matrix}$$

Step2

$$X^T = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 3 & 5 & 4 & 6 \end{bmatrix}$$

Step3

$$X^T \cdot X = \begin{bmatrix} 4 & 18 \\ 18 & 86 \end{bmatrix}$$

$$h(x) = \theta_0 x_0 + \theta_1 x_1$$

Step4

$$(X^T \cdot X)^{-1} = \begin{bmatrix} 4.3 & -.9 \\ -0.9 & 0.2 \end{bmatrix}$$

Step5

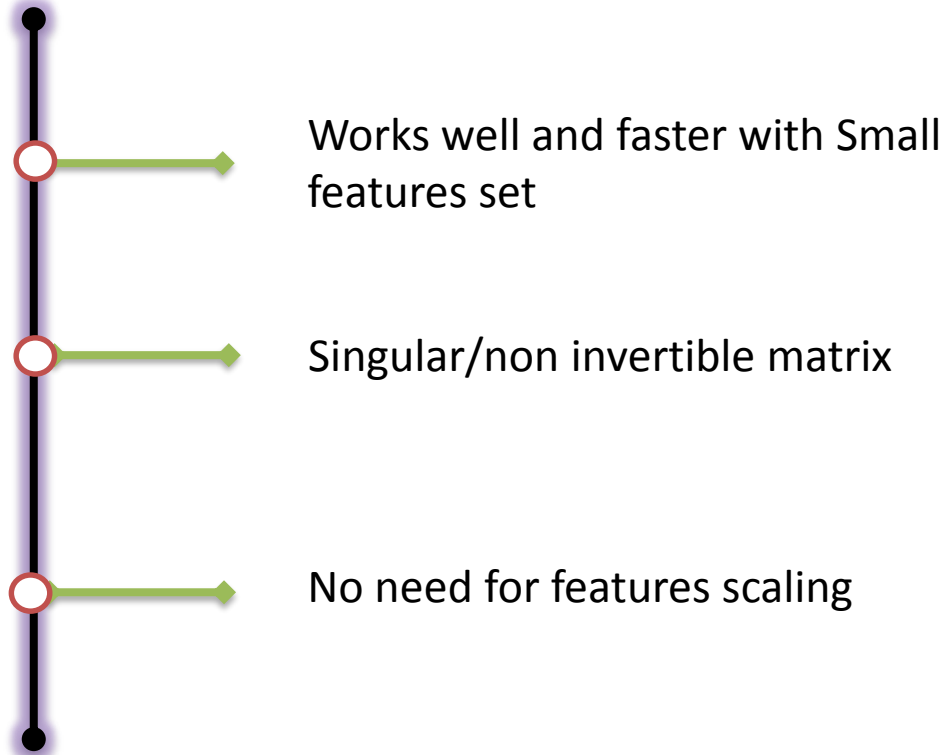
$$(X^T \cdot X)^{-1} \cdot X^T = \begin{bmatrix} 1.6 & -0.2 & 0.7 & -1.1 \\ -0.3 & 0.1 & -0.1 & 0.3 \end{bmatrix}$$

Step6

$$(X^T \cdot X)^{-1} \cdot X^T \cdot y = \begin{bmatrix} -29 \\ 27 \end{bmatrix} \begin{matrix} \leftarrow \theta_0 \\ \leftarrow \theta_1 \end{matrix}$$

$$h(x) = -29x_0 + 27x_1$$

Room#	Gradient Descent	Normal Equation	Actual Price
	$y = -28x_0 + 27x_1$	$y = -29x_0 + 27x_1$	
3	53	52	50
5	107	106	70
4	80	79	100
6	134	133	150



Features Scaling

Features Scaling

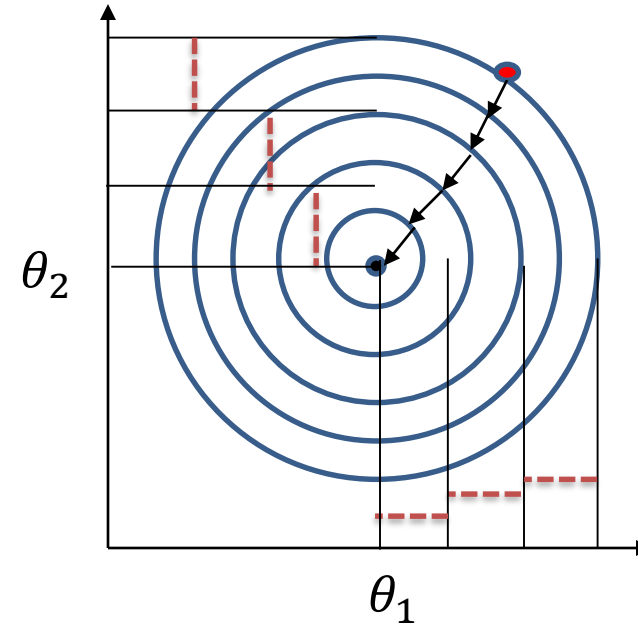
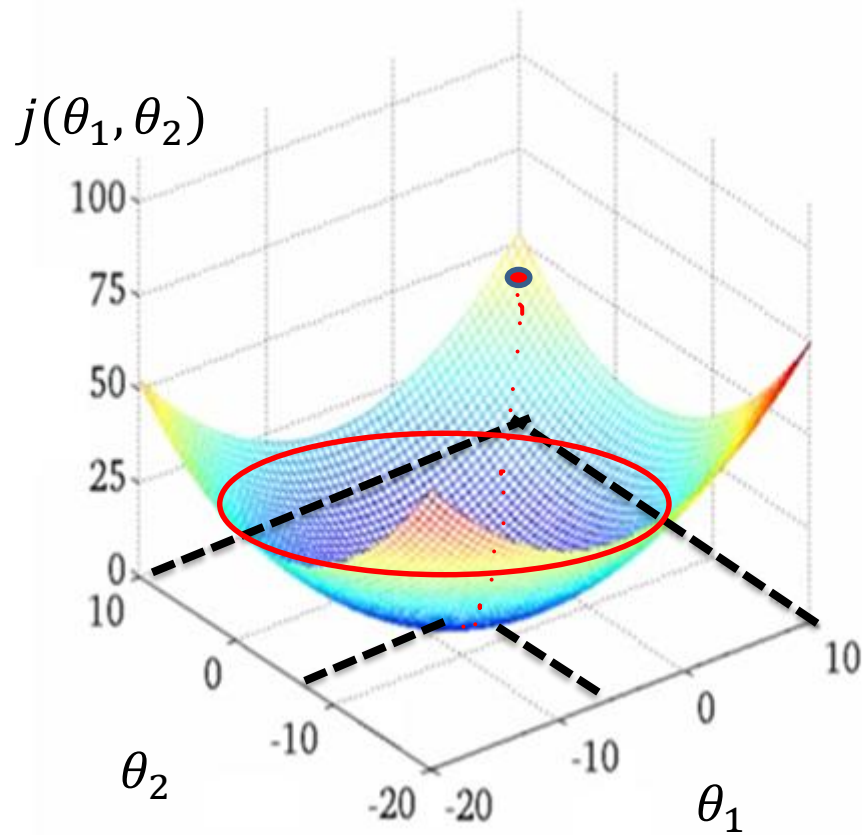
$$y = \theta_1 x_1 + \theta_2 x_2$$

$$2 \leq x_1 \leq 7$$

Room #

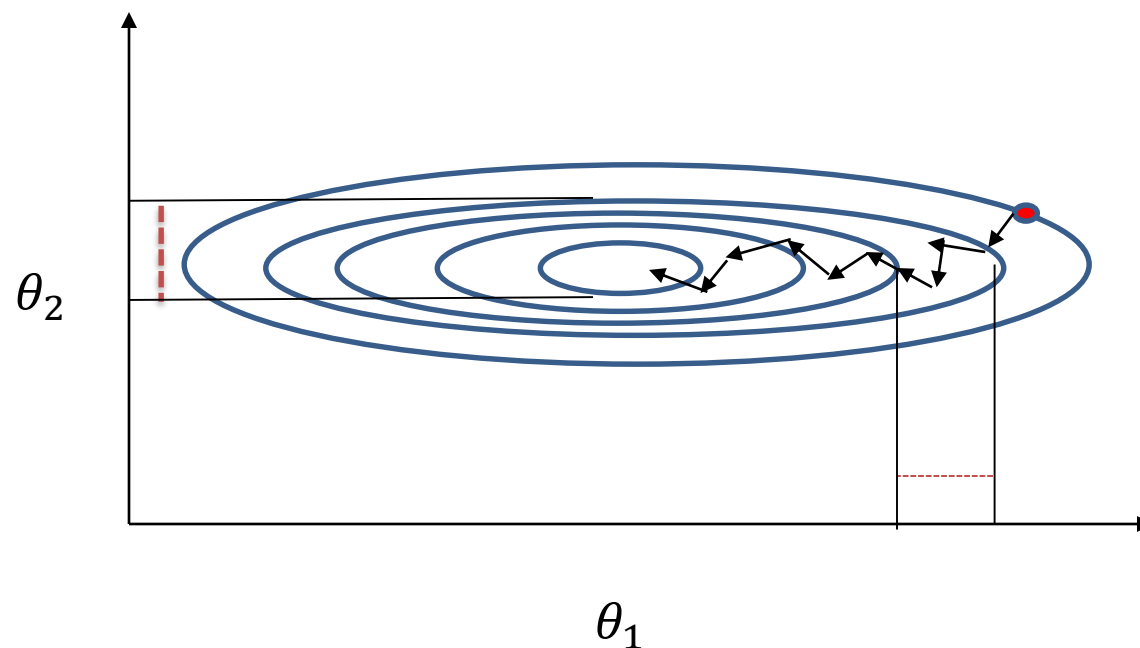
$$65 \leq x_2 \leq 300$$

Size



Speed (KM)	Height (M)	Price
200	2.20	100,000
160	1.90	75,000
180	2.10	80,000
240	2.00	150,000

$$price = \theta_1(speed) + \theta_2(height)$$



- *overshooting*
- Longtime

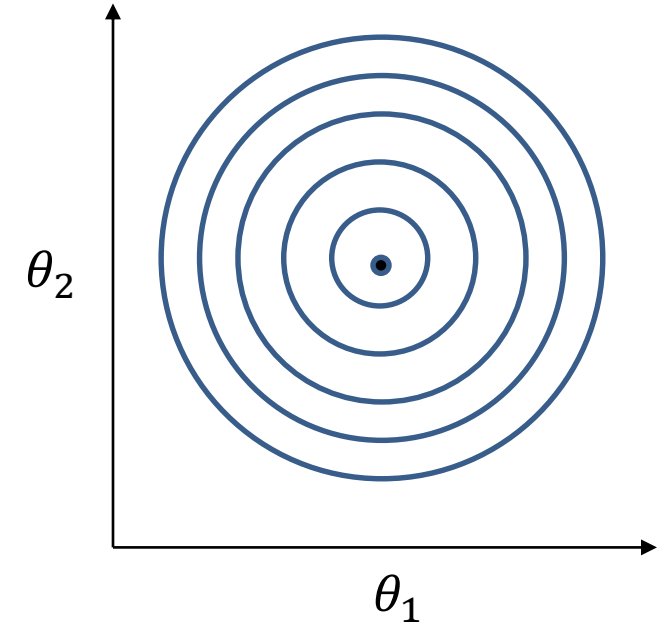
Speed (KM)	Height (M)
200	2.20
160	1.90
180	2.10
240	2.00

Feature scaling

Speed (KM)	Height (M)
0.5	1
0	0
0.25	0.66
1	0.33

$$0 \leq x \leq 1$$

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}$$



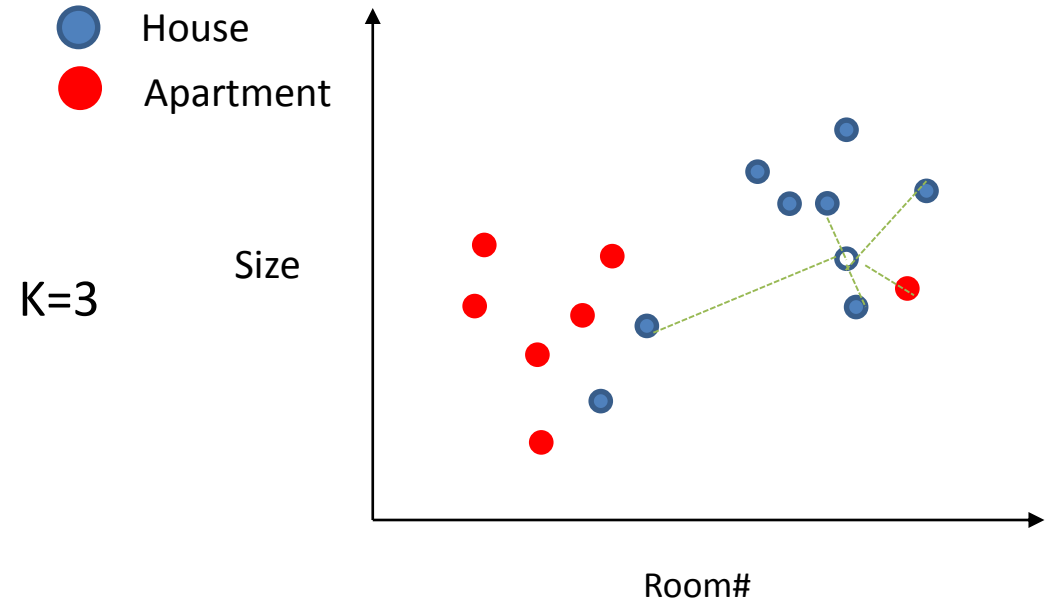
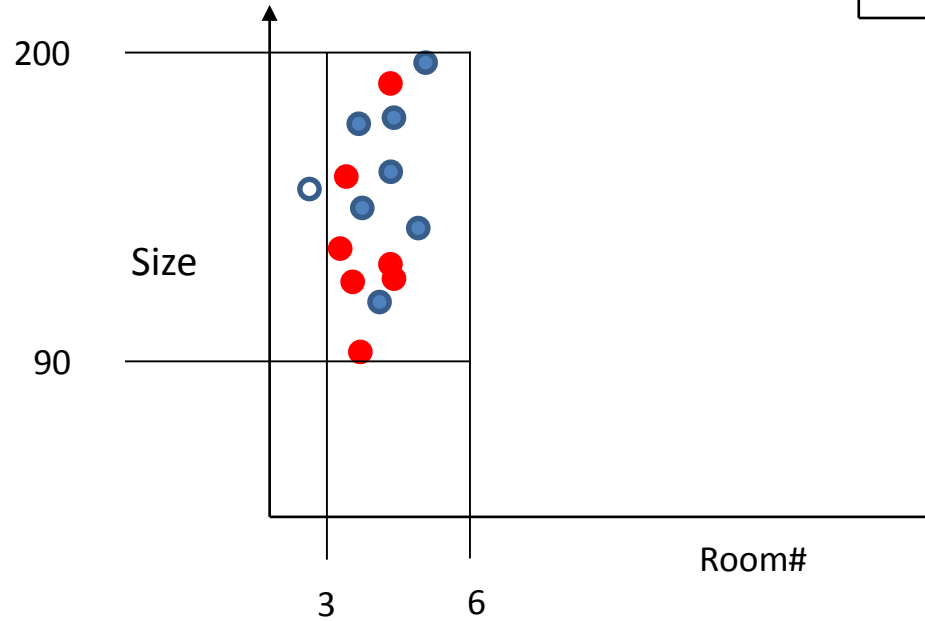
Features Scaling

Used to unify the features ranges to be on a similar scale.

Euclidian Distance $\sqrt{(\Delta x_1)^2 + (\Delta x_2)^2}$

SVM, K-Means, K-NN

Size	Room#
90	3
120	5
150	4
200	6



Features Scaling Methods

1. Min-Max Scaling

Rescaling the values into a range [0, 1]

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}$$

2. Standardization (z-score)

Rescaling the values to have a mean of 0 and a standard deviation of 1.

$$x' = \frac{x - \mu}{\sigma}$$

Size	Room#
90	3
120	5
150	4
200	6

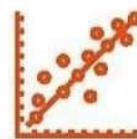
$\mu = 140$	$\mu = 4.5$
$\sigma = 40.62$	$\sigma = 1.118$

Size	Room#
0	0
0.27	0.66
0.54	0.33
1	1

Size	Room#
-1.23	-1.34
-0.49	0.44
.24	-0.44
1.47	1.34

The End

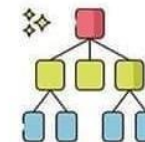
Questions?



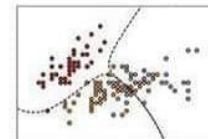
Linear
Regression



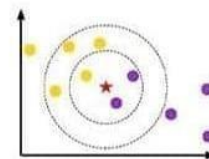
Logistic
Regression



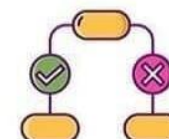
CART
Algorithm



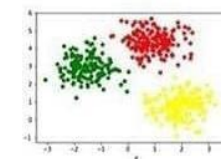
Naïve
Bayes



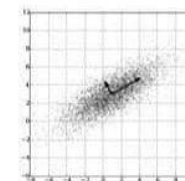
KNN
Algorithm



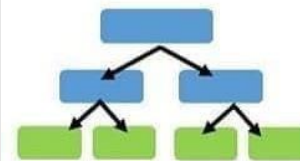
Apriori



K-Means



PCA



Random Forest
Classification

TN	FP	TN
FN	TP	FN
TN	FP	TN

AdaBoost