CHAPTER 07

Linear Regression Basics

01
선형회귀의 이해

1. 선형회귀의 개념

- 선형회귀(Linear Regression) : 종속변수 y와 한 개 이상의 독립변수 x와의 선형 상관관계를 모델링하는 회귀분석 기법
- 기존 데이터를 활용해 연속형 변수값을 예측
- $y = ax + b$ 꼴의 수식을 만들고 a와 b의 값을 찾아냄

01 선형회귀의 이해

- 앞으로 개봉할 영화 예상 관객 수 y를 예측하는 문제

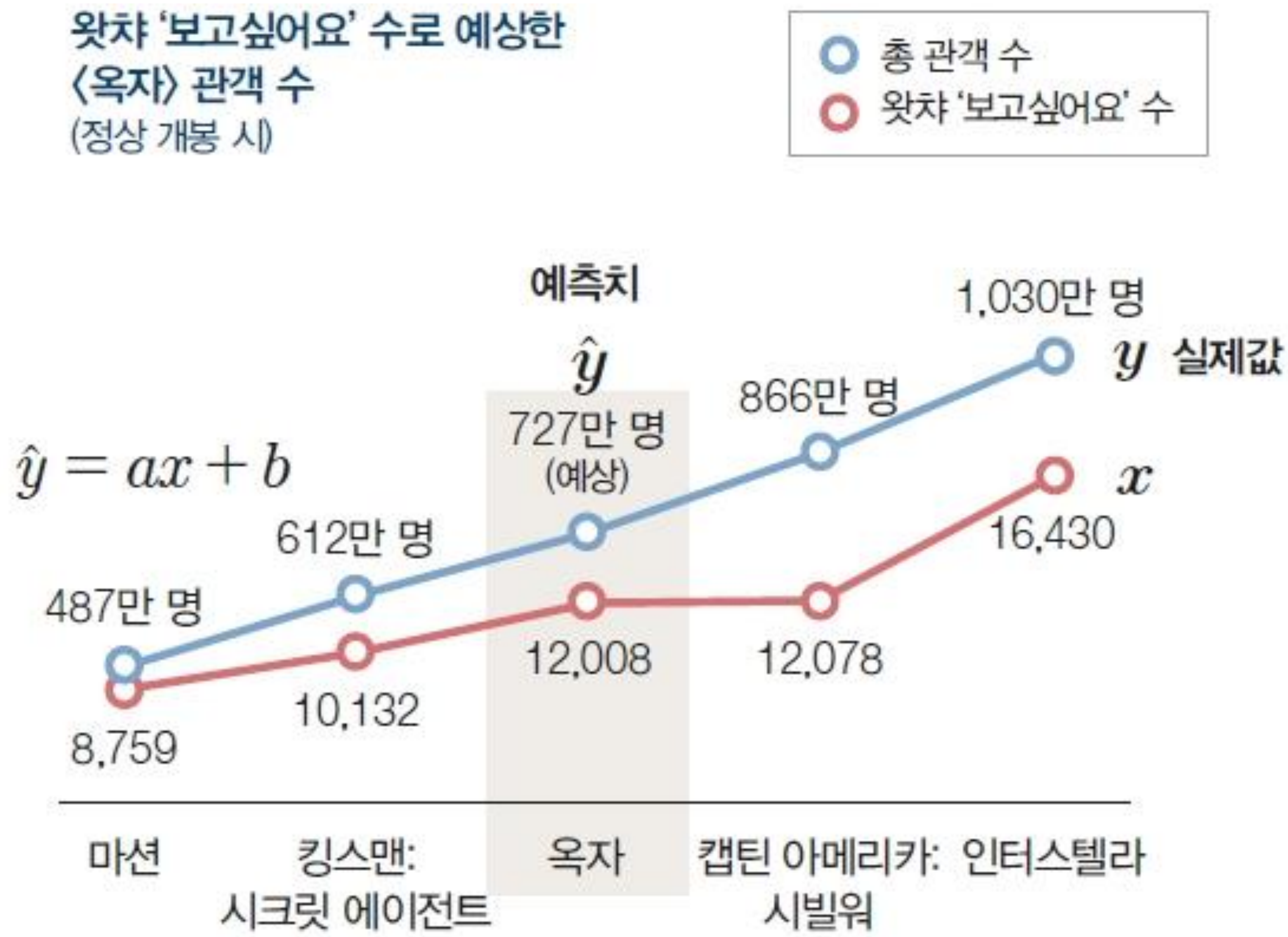


그림 7-1 왓챠 '보고싶어요' 수로 예상한 '옥자' 관객 수

01 선형회귀의 이해

- 실제 관객 수를 y로 표현하여 좌표평면 상에 나타냄

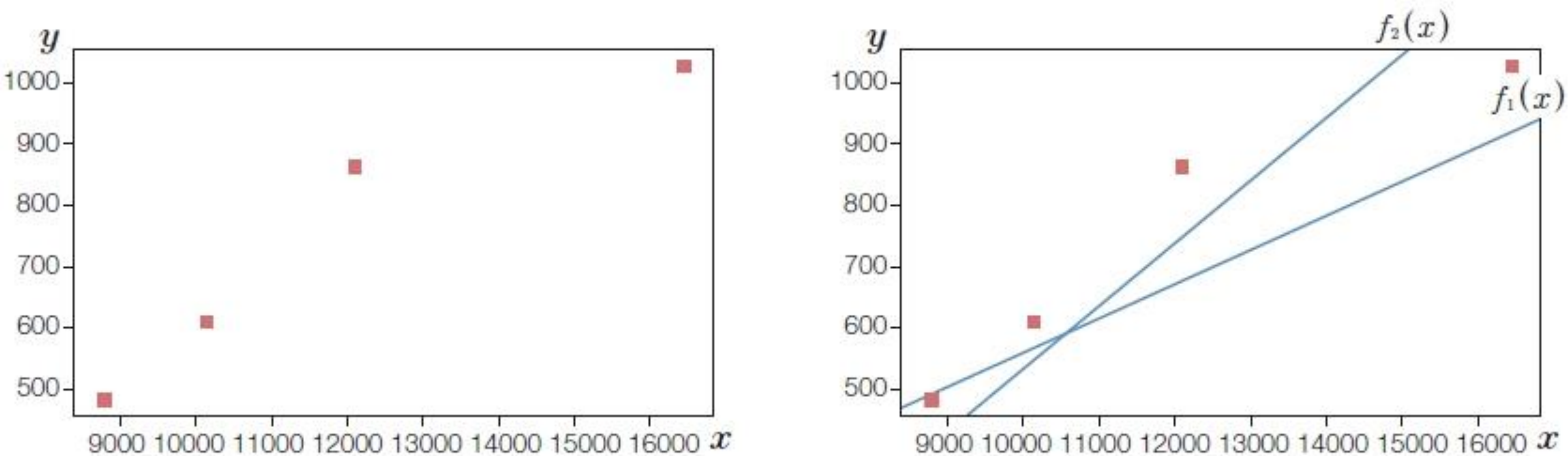


그림 7-2 좌표평면에 작성한 2개의 직선, $f_1(x)$와 $f_2(x)$

01 선형회귀의 이해

- 두 그래프 중 어떤 것이 기존 데이터를 '잘 표현하는가'
- 예측값이 실제값 대비 차이가 많이 나지 않는 그래프

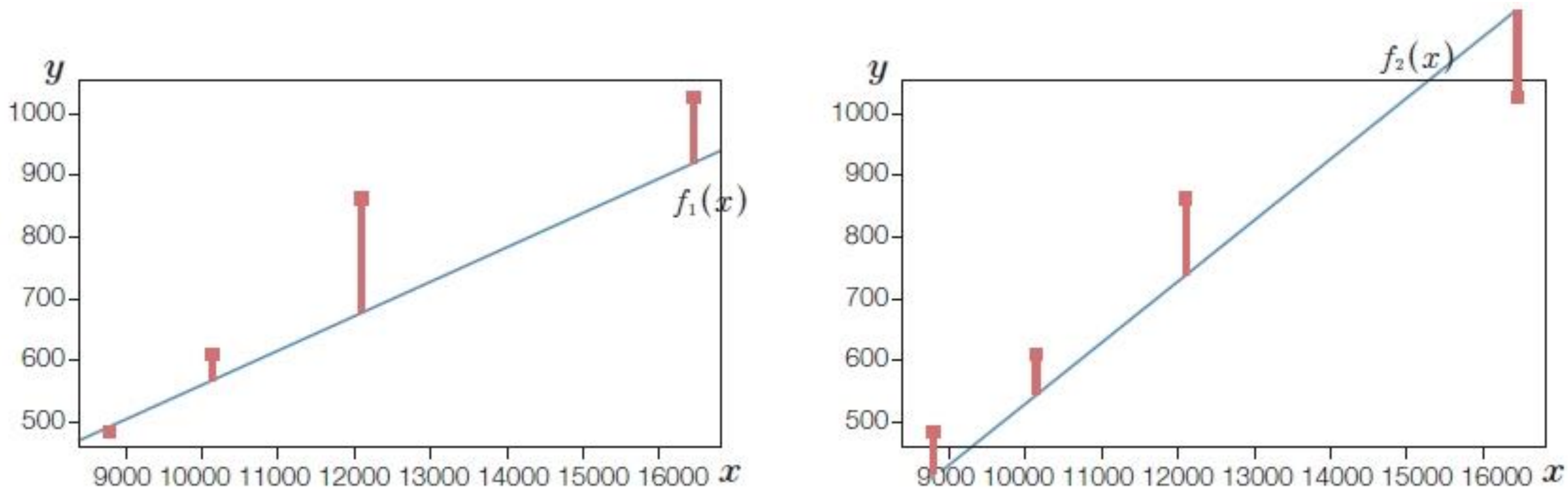


그림 7-3 $f_2(x)$가 $f_1(x)$보다 좀 더 데이터를 잘 표현하고 있다.

2. 예측 함수와 실제값 간의 차이

- 예측 함수는 예측값과 실제값 간의 차이를 최소화하는 방향으로
- 데이터 n개 중 i번째 데이터의 y값에 대한 실제값과 예측값의 차이
- 데이터가 5개 있는 $\hat{y}^i - y^i$개 데이터의 오차의 합

$$(\hat{y}^{(1)} - y^{(1)}) + (\hat{y}^{(2)} - y^{(2)}) + (\hat{y}^{(3)} - y^{(3)}) + (\hat{y}^{(4)} - y^{(4)}) + (\hat{y}^{(5)} - y^{(5)})$$

  - 오차 값들이 음수와 양수로 나왔을 때 값들 간의 차이가 상쇄되어 0으로 계산될 수 있음

01 선형회귀의 이해

- 값의 제곱을 사용하여 오차의 합을 표현

$$(\hat{y}^{(1)}-y^{(1)})^2+(\hat{y}^{(2)}-y^{(2)})^2+(\hat{y}^{(3)}-y^{(3)})^2+(\hat{y}^{(4)}-y^{(4)})^2+(\hat{y}^{(5)}-y^{(5)})^2$$

$$\sum_{i=1}^{n}(\hat{y}^{(i)}-y^{(i)})^2$$

- 같은 식을 행렬로 표현

$$\hat{y}=\begin{bmatrix} w_1\times 8759+w_0 \\ w_1\times 10132+w_0 \\ w_1\times 12078+w_0 \\ w_1\times 16430+w_0 \end{bmatrix} \qquad y=\begin{bmatrix} 487 \\ 612 \\ 866 \\ 1030 \end{bmatrix}$$

$$(\hat{y}-y)^2=\begin{bmatrix} (w_1\times 8759+w_0-487)^2 \\ (w_1\times 10132+w_0-612)^2 \\ (w_1\times 12078+w_0-866)^2 \\ (w_1\times 16430+w_0-1030)^2 \end{bmatrix}$$

01 선형회귀의 이해

- 제곱 오차(square error) : $(\hat{y} - y)^2$ 로 예측값과 실제값의 제곱을 표시하여 오차를 나타냄

- 제곱 오차를 최소화하는 w_0 와 w_1 을 찾아야 함

$$\sum_{i=1}^{n} \left(w_i x^{(i)} + w_0 \times 1 - y^{(i)} \right)^2$$

02
선형회귀의 기초 수식

02 선형회귀의 기초 수식

1. 비용함수의 개념

- 비용함수(cost function) : 머신러닝에서 최소화해야 할 예측값과 실제값의 차이
- 가설함수(hypothesis function) : 예측값을 예측하는 함수

$$f(x) = h_\theta(x)$$

- 함수 입력값은 x이고
  함수에서 결정할 것은 θ로, 가중치(weight) 값인 w_n

02 선형회귀의 기초 수식

- 비용함수가 두 개의 가중치 값으로 결정됨

$$COST(w,b) = \frac{1}{m} * \sum_{i=1}^{m} (wx_i + b - y_i)^2$$

- 잔차의 제곱합(Error sum of squares) : 예측값인 가설함수와 실제값인 y값 간의 차이를 제곱해서 모두 합함
  - 총 데이터는 m개가 존재하고 각 데이터의 예측값과 실제값을 뺀 후 제곱한 값들을 모두 합한 값
- 손실함수(loss function) : 비용함수에서 잔차의 제곱합 부분

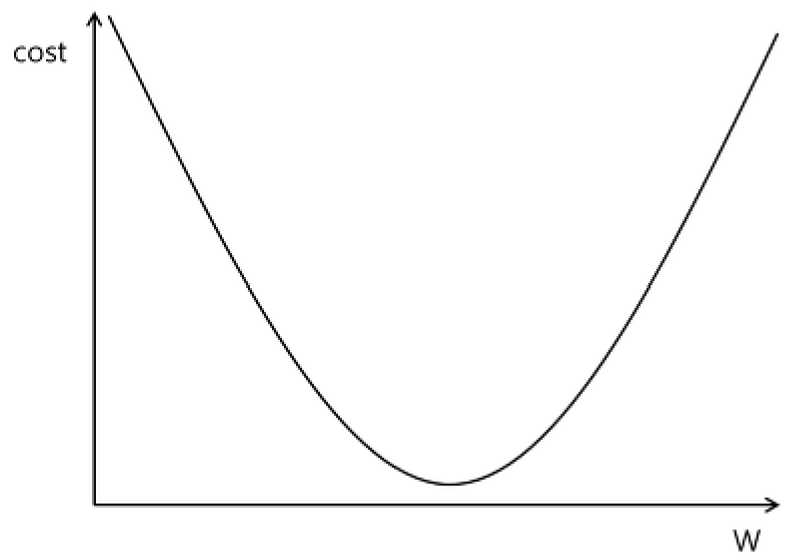
- 평균 제곱 오차(mean squared error, MSE) : 잔차의 제곱합을 m으로 나눈 값

02 선형회귀의 기초 수식

- 비용함수를 수식의 목적이 나타나도록 표기

$$COST(w,b) = \frac{1}{m} * \sum_{i=1}^{m} (wx_i + b - y_i)^2$$

- 비용함수 그래프



03
최소 제곱법 VS 경사하강법

1. 최소제곱법으로 풀기

- 최소제곱법(least square method) : 선형대수의 표기법을 사용하여 방정식으로 선형회귀 문제를 푸는 방법

  - 비용함수를 미분

$$Min\ \ S^2 = Min\sum_{i=1}^{n}\varepsilon_i^2 = Min\sum_{i=1}^{n}(y_i - b_0 - b_1x_i)^2$$

03 최소자승법으로 선형회귀 풀기

1. 최소제곱법으로 풀기

- 일차 함수의 기울기 a와 b(y절편)을 구하는 방법
  - 기울기 a

$$a = \frac{(x - x\text{ 평균})(y - y\text{ 평균})\text{의 합}}{(x - x\text{ 평균})\text{의 합의 제곱}}$$

$$a = \frac{\sum_{i=1}^{n}(x - mean(x))(y - mean(y))}{\sum_{i=1}^{n}(x - mean(x))^2}$$

  - y절편 b

$$b = y\text{의 평균} - (x\text{의 평균} \times \text{기울기 } a)$$

$$b = mean(y) - (mean(x) * a)$$

2. 최소제곱법의 활용

- 데이터의 개수가 피쳐의 개수보다 많은 경우가 대부분이라서 자주 사용됨

- 최소자승법의 장점 : 반복(iteration)과 사용자가 지정하는 하이퍼 매개변수(hyper parameter)가 존재하지 않아서 데이터만 있으면 쉽게 해를 구할 수 있음

- 단점 : 피쳐가 늘어나면 속도가 느려짐
  - 현재 컴퓨터의 연산속도로는 다른 알고리즘에 비해 느린 것이 아님

04 경사하강법으로 선형회귀 풀기

1. 경사하강법의 개념

- 경사하강법(gradient descent) : 경사를 하강하면서 수식을 최소화하는 매개변수의 값을 찾아내는 방법

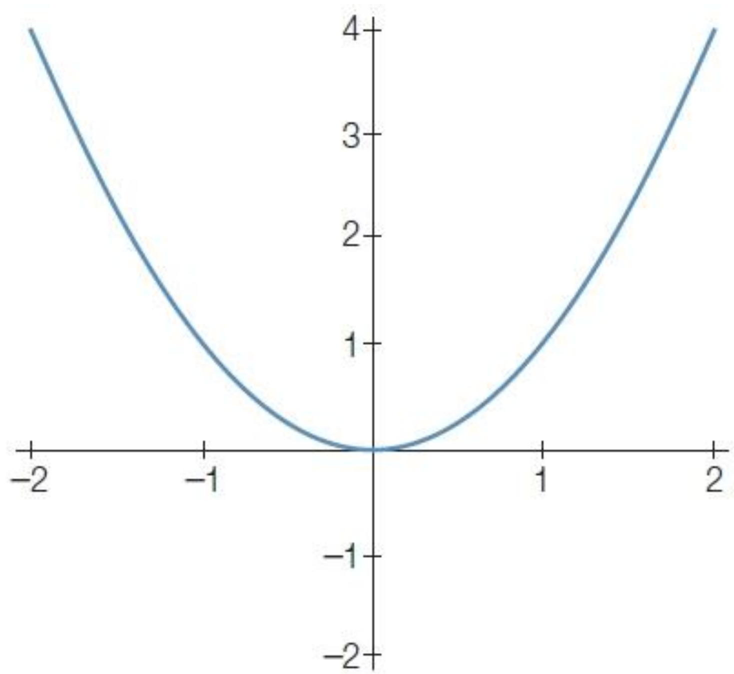


그림 7-5 $y = x^2$ 그래프

04 경사하강법으로 선형회귀 풀기

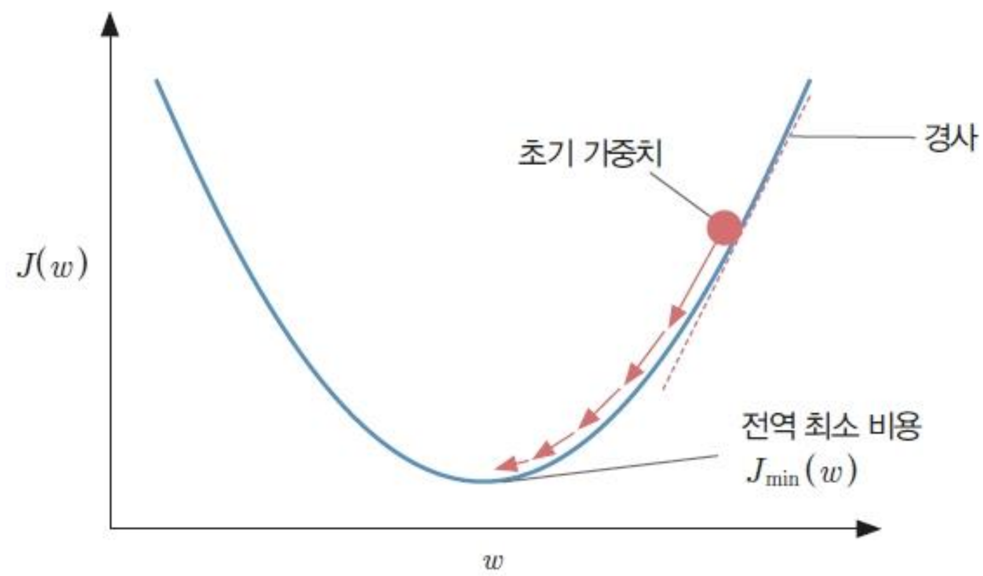
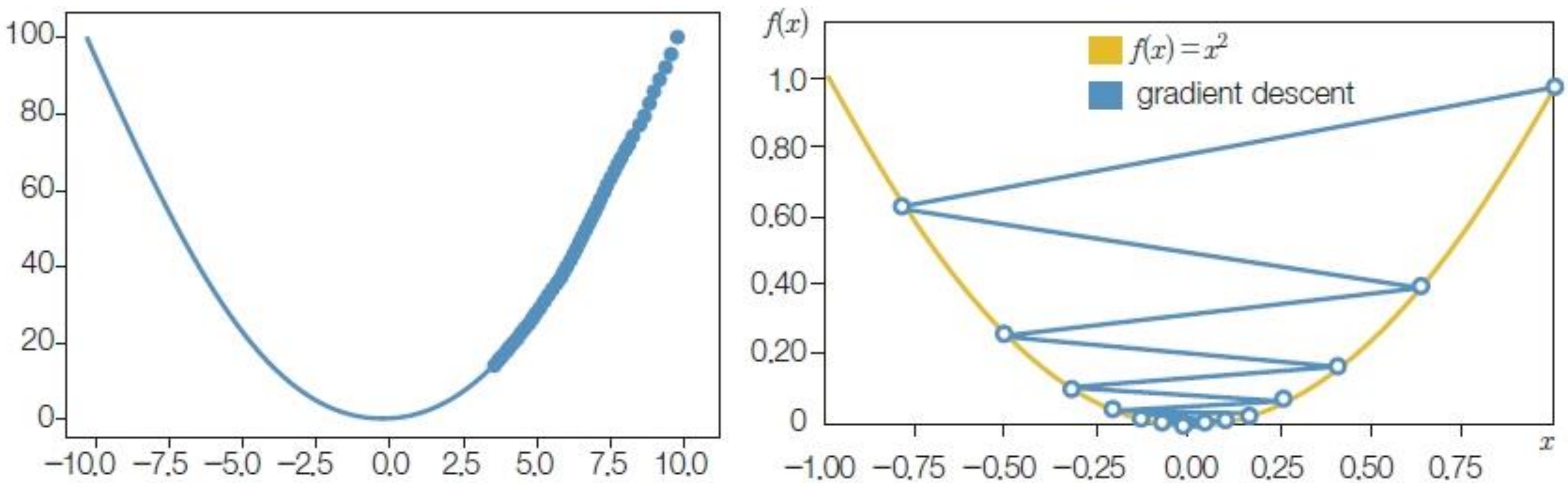


그림 7–6 $y = x^2$ 그래프에 경사하강법 적용

- 점이 최솟값을 달성하는 방향으로 점점 내려감
  - 몇 번 적용할 것인가? : 많이 실행할수록 최솟값에 가까워짐
  - 한 번에 얼마나 많이 내려갈 것인가? : 한 번에 얼마나 많은 공간을 움직일지를 기울기, 즉 경사라고 부름
    - 경사(gradient) : 경사하강법의 하이퍼 매개변수

04 경사하강법으로 선형회귀 풀기



(a) 학습률이 작아 최솟값을 수렴하지 못하는 경우

(b) 학습률이 커서 발산한 경우

그림 7-7 **학습률이 너무 크거나 작을 때 발생하는 문제**

04 경사하강법으로 선형회귀 풀기

- w 변수가 두 개이기 때문에 3차원 그래프로 J를 표현

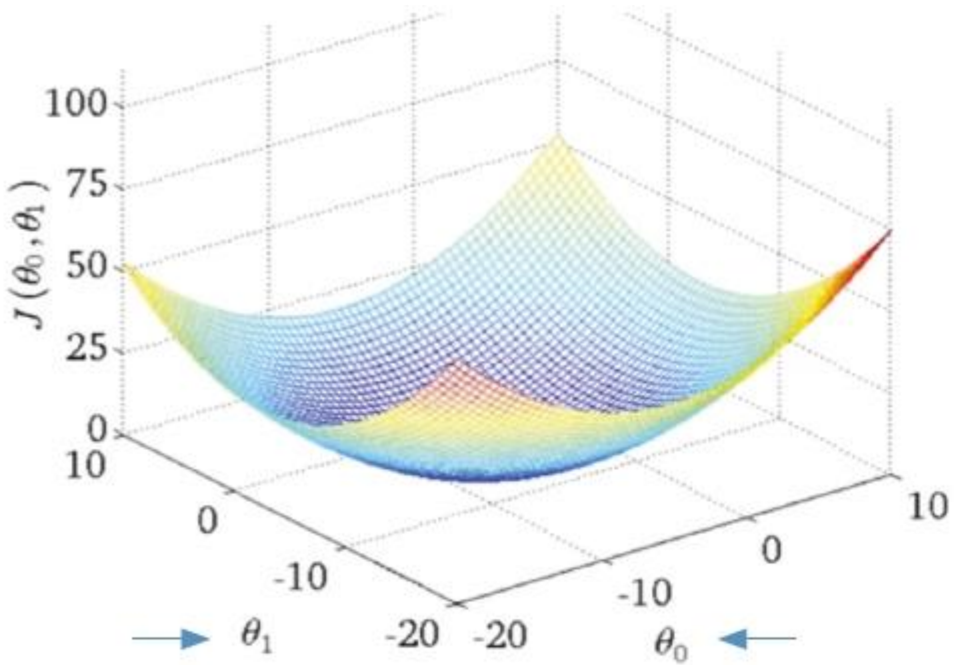


그림 7-9 비용함수 J 그래프 © Andrew Ng

04
선형회귀 성능 측정하기

05 선형회귀 성능 측정하기

1. 훈련/테스트 분할

- 훈련/테스트 분할(train/test split) : 머신러닝에서 데이터를 학습을 하기 위한 학습 데이터셋(train dataset)과 학습의 결과로 생성된 모델의 성능을 평가하기 위한 테스트 데이터셋(test dataset)으로 나눔
- 모델이 새로운 데이터셋에도 일반화(generalize)하여 처리할 수 있는지를 확인

05 선형회귀 성능 측정하기

- 모델이 데이터에 과다적합(over-fit)된 경우 :
  생성된 모델이 특정 데이터에만 잘 맞아서 해당 데이터셋에 대해서는 성능을 발휘할 수 있지만 새로운 데이터셋에서는 전혀 성능을 낼 수 없다
- 모델이 데이터에 과소적합(under-fit)된 경우 :
  기존 학습 데이터를 제대로 예측하지 못함

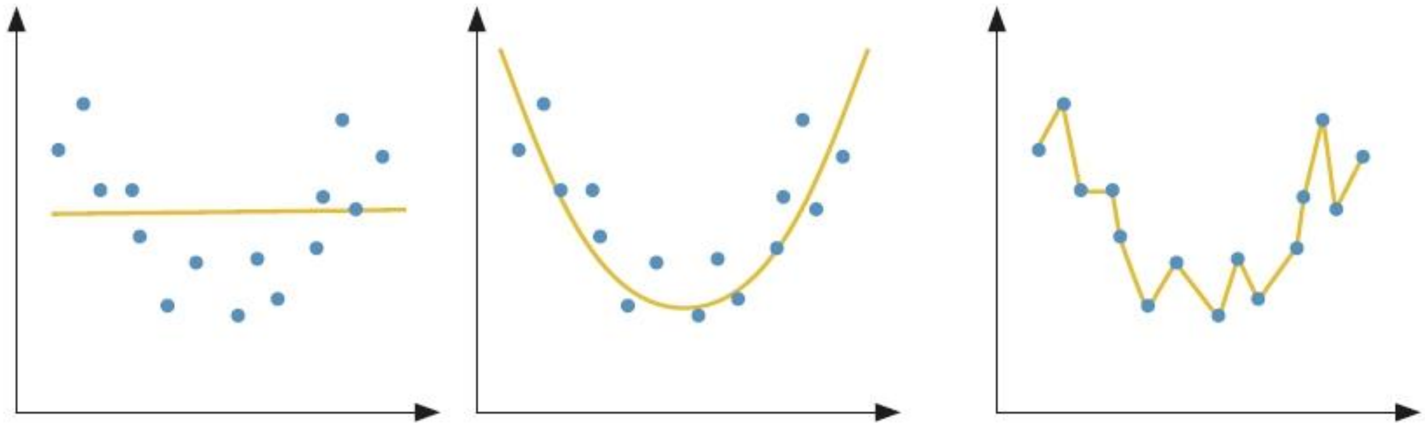


그림 7-10 다양한 데이터셋과 성능 파악

05 선형회귀 성능 측정하기

- 홀드아웃 메서드(hold-out method) : 전체 데이터셋에서 일부를 학습 데이터와 테스트 데이터로 나누는 일반적인 데이터 분할 기법
  - 전체 데이터에서 랜덤하게 학습 데이터셋과 테스트 데이터셋을 나눔
  - 일반적으로 7:3 또는 8:2 정도의 비율

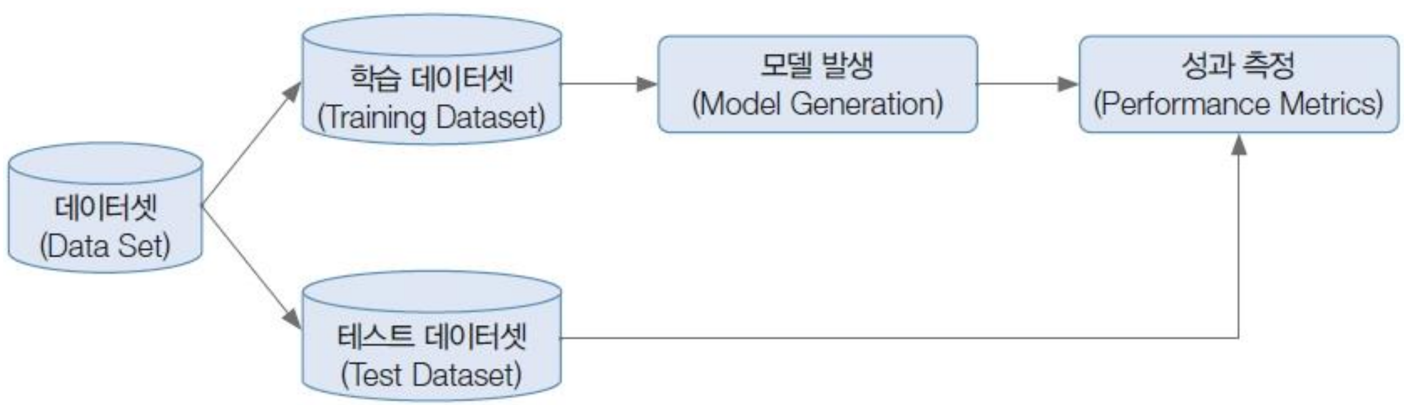


그림 7-11 홀드아웃 메서드(hold-out method) 기법

05 선형회귀 성능 측정하기

- sklearn 모듈이 제공하는 train_test_split 함수 사용
  - X와 y 벡터 값을 각각 넣고
  - 매개변수 test_size에 테스트 데이터로 사용할 데이터의 비율을 지정
  - random_state는 랜덤한 값을 기준으로 임의로 지정하는 값

In [1]:
```
import numpy as np
from sklearn.model_selection import train_test_split

X, y = np.arange(10).reshape((5, 2)), range(5)

X_train, X_test, y_train, y_test = train_test_split(
 X, y, test_size=0.33, random_state=42)
```

2. 선형회귀의 성능 측정 지표

2.1 MAE

- MAE(Mean Absolute Error) : 평균 절대 잔차
- 모든 테스트 데이터에 대해 예측값과 실제값의 차이에 대해 절댓값을 구하고, 이 값을 모두 더한 후에 데이터의 개수만큼 나눈 결과

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

- 직관적으로 예측값과 실측값의 차이를 알 수 있음

05 선형회귀 성능 측정하기

- sklearn 모듈에서는 median_absolute_error 함수로 MAE를 구함

In [2]:	`from sklearn.metrics import median_absolute_error`
	`y_true = [3, -0.5, 2, 7]`
	`y_pred = [2.5, 0.0, 2, 8]`
	`median_absolute_error(y_true, y_pred)`

Out [2]:	0.5

05 선형회귀 성능 측정하기

2.2 RMSE

- RMSE(Root Mean Squared Error) : 평균제곱근 오차
- 오차에 대해 제곱을 한 다음 모든 값을 더하여 평균을 낸 후 제곱근을 구함
- MAE에 비해 상대적으로 값의 차이가 더 큼
- 차이가 크게 나는 값에 대해서 페널티를 주고 싶다면 RMSE 값을 사용

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

05 선형회귀 성능 측정하기

- sklearn 모듈에서 RMSE를 직접적으로 지원하지는 않고 mean_squared_error만 지원

In [3]:
```python
from sklearn.metrics import mean_squared_error
y_true = [3, -0.5, 2, 7]
y_pred = [2.5, 0.0, 2, 8]
mean_squared_error(y_true, y_pred)
```

Out [3]:
```
0.375
```

2.3 결정계수

- 결정계수(R-squared) : 두 개의 값의 증감이 얼마나 일관성을 가지는지 나타내는 지표
- 예측값이 크면 클수록 실제값도 커지고, 예측값이 작으면 실제값도 작아짐
- 두 개의 모델 중 어떤 모델이 조금 더 상관성이 있는지를 나타낼 수 있지만, 값의 차이 정도가 얼마인지는 나타낼 수 없다는 한계가 있음

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \mu)^2}$$

05 선형회귀 성능 측정하기

- sklearn 모듈에서 r2_score 사용

In [4]:
```
from sklearn.metrics import r2_score
y_true = [3, -0.5, 2, 7]
y_pred = [2.5, 0.0, 2, 8]
r2_score(y_true, y_pred)
```

Out [4]:
```
0.9486081370449679
```

05
코드로
선형회귀 구현하기

06 코드로 선형회귀 구현하기

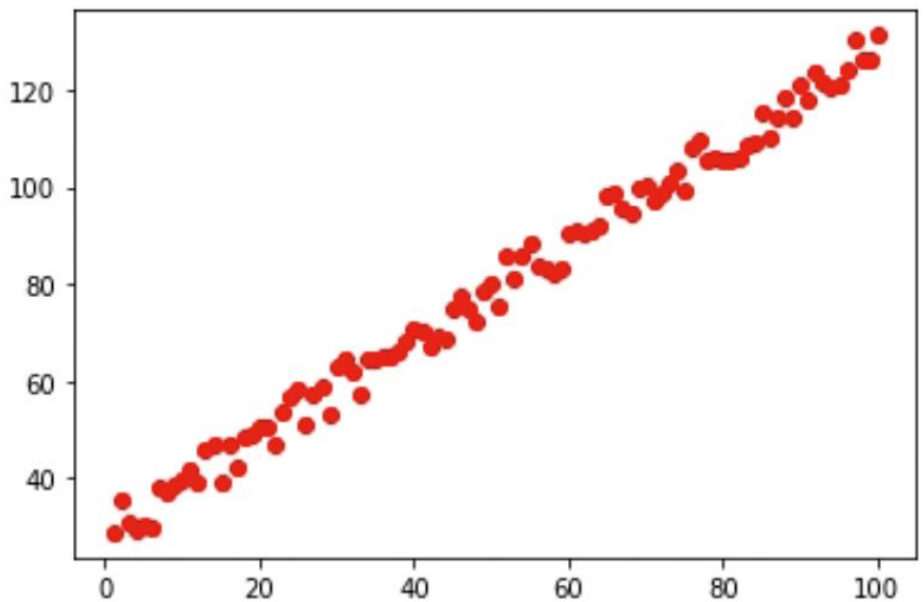
- 경사하강법을 선형회귀로 구현
  - 데이터 생성

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
import random

def gen_data(numPoints, bias, variance):
    x = np.zeros(shape=(numPoints, 2))
    y = np.zeros(shape=numPoints)

    for i in range(0, numPoints):
        x[i][0] = 1 # (2) 데이터 x의 상수항에는 1
        x[i][1] = i # (3) 데이터 x 값은 1씩 증가시킴
        y[i] = (i+bias) + random.uniform(0, 1) * variance
        # (4) 데이터 y에 bias 생성
    return x, y
x, y = gen_data(100, 25, 10) # (1) 100개의 데이터 생성
```

06 코드로 선형회귀 구현하기

```
plt.plot(x[:,1]+1,y,"ro") # (5) 데이터 x와 y의 상관관계 그래프 작성
plt.show()
```

Out [1]:



06 코드로 선형회귀 구현하기

- 생성된 데이터에 경사하강법 적용

```
In [2]: def gradient_descent(x, y, theta, alpha, m,
        numIterations):
            xTrans = x.transpose() # (6)
            theta_list = [] # (7)
            cost_list = [] # (8)
            for i in range(0, numIterations): # (9)
                hypothesis = np.dot(x, theta) # (10)
                loss = hypothesis - y # (11)
                cost = np.sum(loss ** 2) / (2 * m) # (12)
                gradient = np.dot(xTrans, loss) / m # (13)
                theta = theta - alpha * gradient # (14)
                if i % 250 == 0: # (15)
                    theta_list.append(theta)
                    cost_list.append(cost)
            return theta,np.array(theta_list), cost_list # (16)
```

06 코드로 선형회귀 구현하기

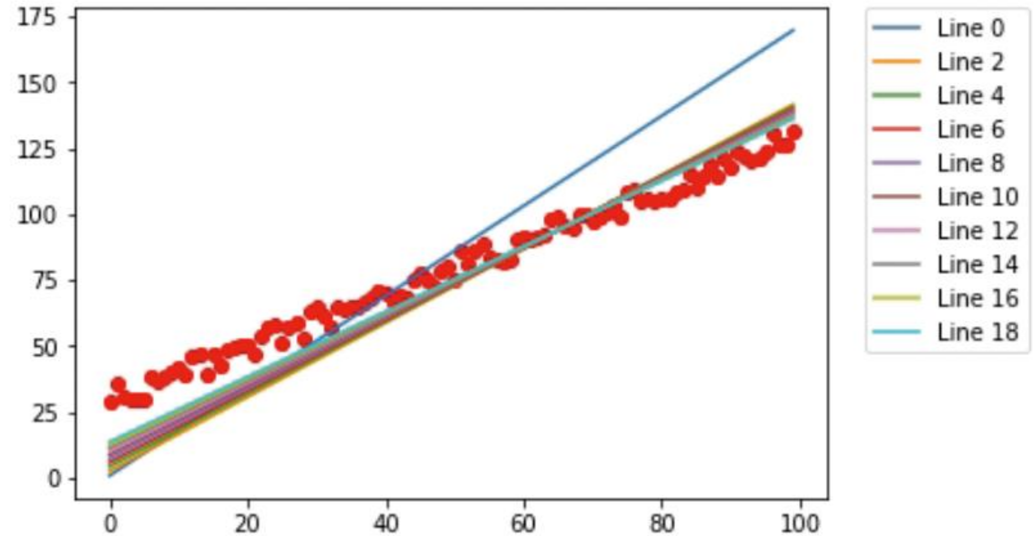
```
m, n = np.shape(x) # (1)
numIterations= 5000 # (2)
alpha = 0.0005 # (3)
theta = np.ones(n) # (4)

theta,theta_list, cost_list = gradient_descent(x, y,
theta, alpha, m, numIterations) # (5)
```

06 코드로 선형회귀 구현하기

In [3]:

```
y_predict_step= np.dot(x, theta_list.transpose())
plt.plot(x[:,1],y,"ro")
for i in range (0,20,2):
    plt.plot(x[:,1],y_predict_step[:,i],
label='Line %d'%i)

plt.legend(bbox_to_anchor=(1.05, 1), loc=2,
borderaxespad=0.)
plt.show()
```

Out [3]:

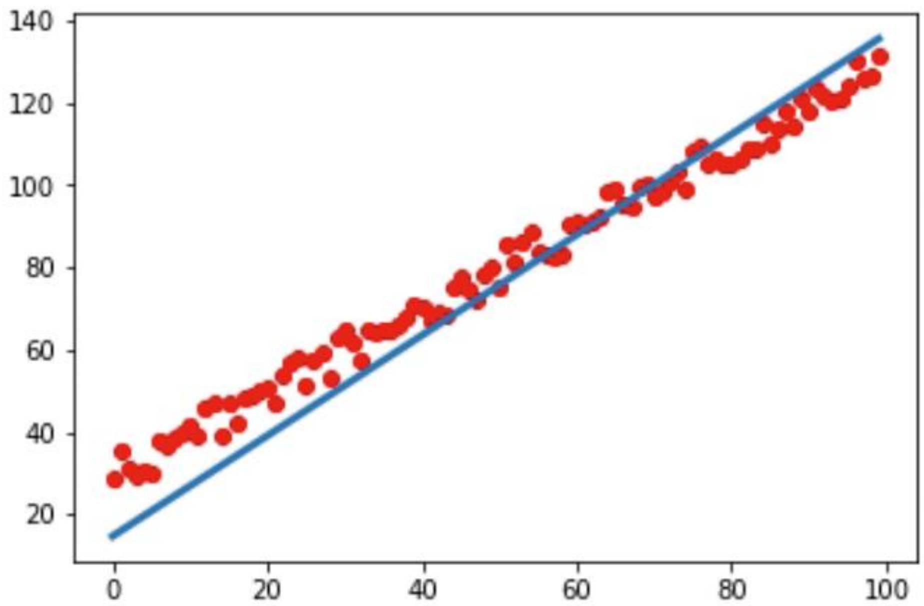


06 코드로 선형회귀 구현하기

In [4]:

```
y_predict= np.dot(x, theta)
plt.plot(x[:,1],y,"ro")
plt.plot(x[:,1],y_predict, lw=3)
plt.show()
```

Out [4]:



06 코드로 선형회귀 구현하기

In [5]:

```
iterations = range(len(cost_list))

plt.scatter(iterations,cost_list)
plt.show()
```

Out [5]: