# 3장. 가장 훌륭한 예측선 긋기: 선형 회귀

2018.05.25 딥메니아
By. 김학영
cs7610@naver.com

# 1. 선형 회귀의 정의

"학생들의 중간고사 성적이 다 다르다."

➔ 당연한 사실 외에는 알 수 있는 것이 없음

"학생들의 중간고사 성적이 [ 공부시간 ]에 따라 다 다르다."

➔ 해당 성적의 이유를 타당하게 설명할 수 있음

➔ [ ]에 들어갈 내용을 **'정보'**라고 하며, 머신러닝과 딥러닝은 정보가 필요함

# 1. 선형 회귀의 정의

“학생들의 중간고사 성적이 [ 공부시간 ]에 따라 다 다르다.”

- **x:** 성적을 변하게 하는 ‘정보’ 요소 = **독립 변수**
- **y:** x값에 의해 변하는 ‘성적’ = **종속 변수**

- **선형 회귀:** 독립변수 x를 이용해 종속변수 y의 움직임을 예측하고 설명하는 작업

1. **단순 선형 회귀(simple linear regression):** 하나의 x값만으로 y값을 설명할 수 있을 때
   y=ax+b

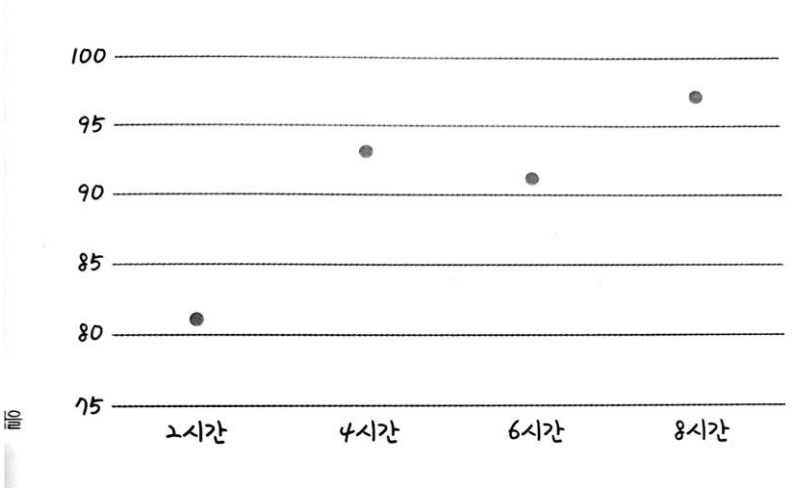
1. **다중 선형 회귀(multiple linear regression):** x값이 여러 개 필요할 때
   y=ax1+bx2+cx3+b

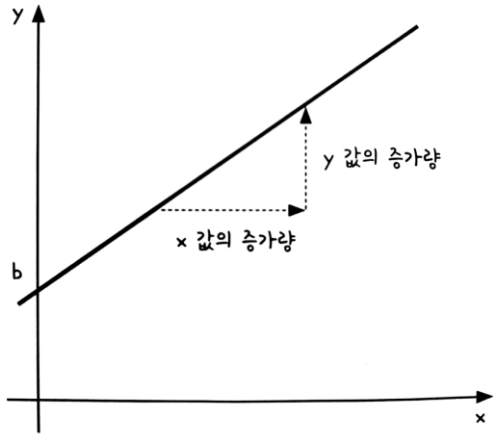
ex) “학생들의 중간고사 성적이 [ 공부시간, 컨디션, 사교육비 ]에 따라 다 다르다.”

# 2. 가장 훌륭한 예측선이란?

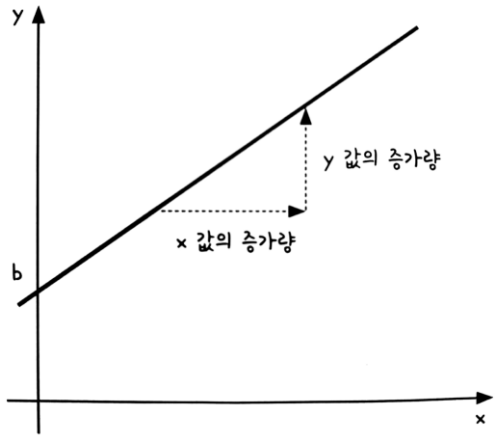
“학생들의 중간고사 성적이 [ 공부시간 ]에 따라 다 다르다.”

| 공부한 시간(x) | 2시간 | 4시간 | 6시간 | 8시간 |
|---|---|---|---|---|
| 성적(y) | 81점 | 93점 | 91점 | 97점 |



# 2. 가장 훌륭한 예측선이란?




"학생들의 중간고사 성적이 [ 공부시간 ]에 따라 다 다르다."

y= ax+b

a: y값의 증가량 / x값의 증가량 (직선의 기울기)

b: y절편

- 선형 회귀는 결국 최적의 a값과 b값을 찾아내는 작업!

- 데이터에 들어 있지 않은 학생의 성적을 예측할 때, 공부시간(x)만 알면 됨
- 정확한 a와 b값을 따라 움직이는 직선에 공부시간인 x값을 대입하기만 하면, 예측성적인 y값을 구할 수 있음

- 머신러닝은 결국 기존 데이터(정보)를 가지고 어떤 선이 그려질지를 예측한 뒤, 아직 답이 나오지 않은 그 무언가를 그 선에 대입해보는 것

# 3. 최소 제곱법(method of least squares)

최적의 a값과 b값을 찾아내는 공식!

"학생들의 중간고사 성적이 [ 공부시간 ]에 따라 다 다르다."

| 공부한 시간(x) | 2시간 | 4시간 | 6시간 | 8시간 |
|---|---|---|---|---|
| 성적(y) | 81점 | 93점 | 91점 | 97점 |

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

$$a = \frac{(2-5)(81-90.5)+(4-5)(93-90.5)+(6-5)(91-90.5)+(8-5)(97-90.5)}{(2-5)^2+(4-5)^2+(6-5)^2+(8-5)^2}$$

$$= \frac{46}{20}$$

$$= 2.3$$

# 3. 최소 제곱법(method of least squares)

“학생들의 중간고사 성적이 [ 공부시간 ]에 따라 다 다르다.”

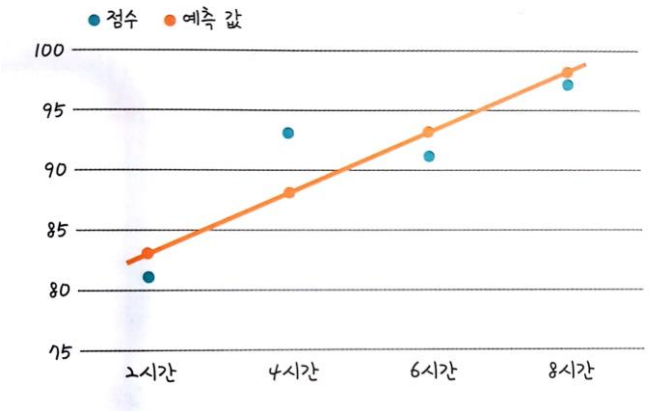
| 공부한 시간(x) | 2시간 | 4시간 | 6시간 | 8시간 |
|---|---|---|---|---|
| 성적(y) | 81점 | 93점 | 91점 | 97점 |
| 예측값( $\hat{y}$ ) | 83.6 | 88.2 | 92.8 | 97.4 |

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

$$b = 90.5 - (2.3 \times 5)$$
$$= 79$$

**y= 2.3x + 79**

위 직선에 다른 x 값(공부한 시간)을 집어넣어서 ‘공부량에 따른 성적을 예측’할 수 있음



# 4. 코딩으로 확인하는 최소제곱

import numpy as np

-> numpy 라이브러리를 불러와 np라는 이름으로 사용

x=[2, 4, 6, 8]
y=[81, 93, 91, 97]

-> 데이터 값을 '리스트' 형식으로 x와 y로 정의
리스트 이름=[요소1, 요소2, 요소3….]

**1) x와 y의 평균값**

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

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

mx = np.mean(x)
my = np.mean(y)

-> mx라는 변수에 x원소들의 평균값을 입력
-> my라는 변수에 y원소들의 평균값을 입력

**mean():** 모든 원소의 평균을 구하는 numpy 함수

# 4. 코딩으로 확인하는 최소제곱

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

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

## 2) 기울기 공식(a)의 분모

```
divisor = sum([(mx - i)**2 for i in x])
```

<- x의 평균값과 x의 각 원소들의 차를 제곱하라

**sum()**: ∑

****2:** 제곱을 구하라

**for i in x:** x의 각 원소를 한 번씩 i 자리에 대입하라

## 3) 기울기 공식(a)의 분자

```
def top(x, mx, y, my):
    d = 0
    for i in range(len(x)):
        d += (x[i] - mx) * (y[i] - my)
    return d
dividend = top(x, mx, y, my)
```

<- top이라는 함수를 새롭게 만들겠다
(**def:** 함수를 만들 때 사용하는 예약어)

<- 임의의 변수 d의 초기값을 0으로 설정한 뒤 x의 개수만큼 실행

<- d에 각 원소와 평균의 차, y의 각 원소와 평균의 차를 곱해서 차례로 더하라

# 4. 코딩으로 확인하는 최소제곱

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

**4) 기울기와 y 절편 구하기**

a = dividend / divisor

b = my - (mx*a)

# 5. 평균 제곱근 오차

- BUT 실제로는 훨씬 많은 변수를 다루게 되며, 복잡한 연산 과정을 거치게 됨
- 임의의 선을 그리고 난 후, 얼마나 잘 그려졌는지 평가하여 조금씩 수정해가는 방법 사용

- **평균 제곱근 오차(root mean square error):** 주어진 선의 오차를 평가하는 오차 평가 알고리즘

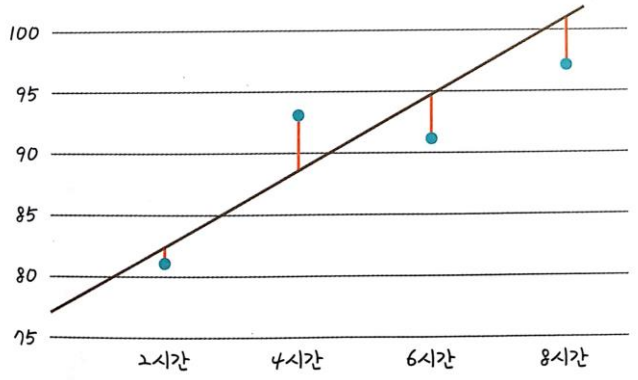
# 6. 잘못 그은 선 바로잡기

"학생들의 중간고사 성적이 [ 공부시간 ]에 따라 다 다르다."

| 공부한 시간(x) | 2시간 | 4시간 | 6시간 | 8시간 |
|---|---|---|---|---|
| 성적(y) | 81점 | 93점 | 91점 | 97점 |
| 예측값( $\hat{y}$ ) | 83.6 | 88.2 | 92.8 | 97.4 |
| 오차 | 1 | -5 | 3 | 3 |

**오차** = 실제 값(y) - 예측 값( $\hat{y}$ )
**실제 값:** y의 실제 값
**예측 값:** x를 식에 대입해서 나오는 값

빨간색 직선(오차)의 합이 작을수록 잘 그어진 직선이고, 직선들의 합이 클수록 잘못 그어진 직선



# 6. 잘못 그은 선 바로잡기

- 오차에는 양수와 음수가 섞여 있기 때문에 단순히 더해 버리면 합이 0이 될 수도 있음
- + - 부호를 없애기 위해서, 각 오차의 값을 제곱해줌

  $p_i$ 실제 값

  $y_i$ 예측 값

$$\text{오차의 합} = \sum_{i=1}^{n} (p_i - y_i)^2$$

$$\text{평균 제곱 오차(MSE)} = \frac{1}{n}\sum_{i=1}^{n} (p_i - y_i)^2$$

$$\text{평균 제곱근 오차(RMSE)} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} (p_i - y_i)^2}$$

- **선형회귀**란 임의의 직선을 그어 이에 대한 평균 제곱근 오차를 구하고, 이 값을 가장 작게 만들어주는 a와 b값을 찾아가는 작업

# 7. 코딩으로 확인하는 평균 제곱근 오차

**1) 기울기 a와 y 절편 b**

```
ab=[3,76]
```

**2) x,y의 데이터 값**

```
data = [[2, 81], [4, 93], [6, 91], [8, 97]]
```
<- [공부시간, 성적]

```
x = [i[0] for i in data]
```
<- 첫 번째 값을 x리스트에 저장

```
y = [i[1] for i in data]
```
<- 두 번째 값을 y리스트에 저장

**3) y=ax + b에 a,b 값 대입하여 결과를 출력하는 함수**

```
def predict(x):
    return ab[0]*x + ab[1]
```

# 7. 코딩으로 확인하는 평균 제곱근 오차

**4) RMSE 함수**

```
def rmse(p, a):
    return np.sqrt(((p - a) ** 2).mean())
```

**np.sqrt():** 제곱근

****2:** 제곱을 구하라

**mean():** 평균값을 구하라

$$\text{평균 제곱근 오차(RMSE)} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(p_i - y_i)^2}$$

**5) RMSE 함수를 각 y값에 대입하여 최종 값을 구하는 함수**

```
def rmse_val(predict_result,y):
    return rmse(np.array(predict_result), np.array(y))
```

# 7. 코딩으로 확인하는 평균 제곱근 오차

**6) 예측값이 들어갈 빈 리스트**

```
predict_result = []
```

**7) 모든 x값을 한 번씩 대입하여 predict_result 리스트완성.**

```
for i in range(len(x)):
    predict_result.append(predict(x[i]))
    print("공부시간=%.f, 실제점수=%.f, 예측점수=%.f" % (x[i], y[i], predict(x[i])))
```

**8) 최종 RMSE 출력**

```
print("rmse 최종값: " + str(rmse_val(predict_result,y)))
```