9주차

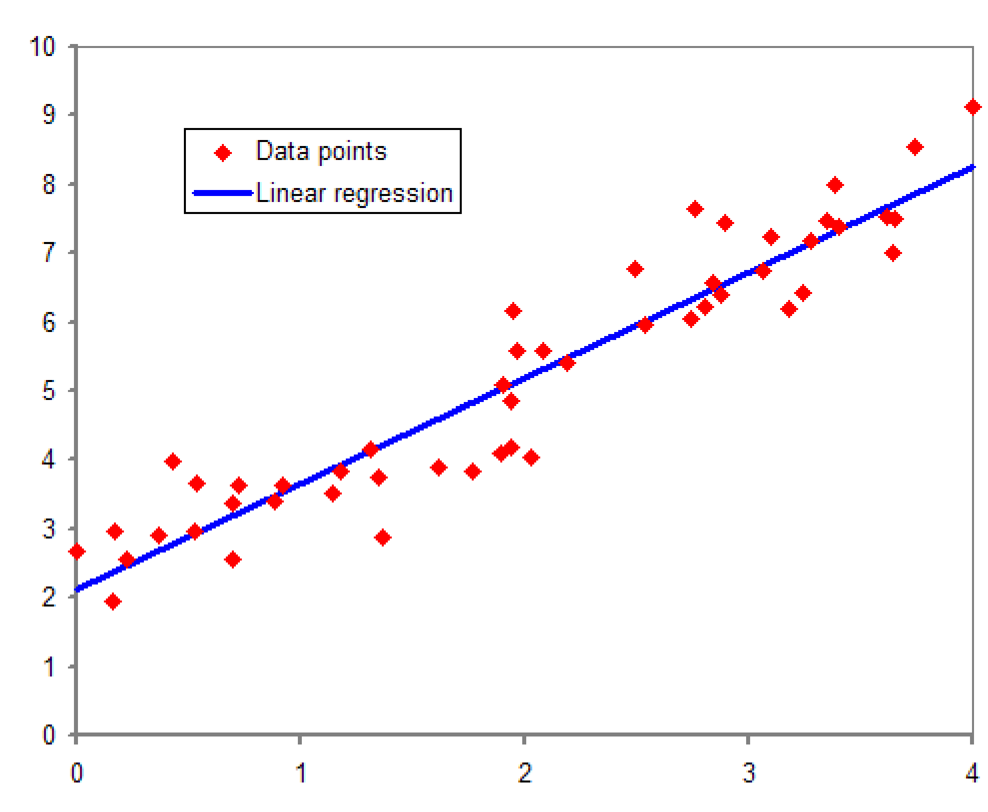
가장 훌륭한 예측선 긋기: 선형회귀

모두를 위한 인공지능 활용

한동대학교
김영섭 교수

선형 회귀의 정의

- 선형 회귀(**linear regression**) **-** 가장 훌륭한 예측선 긋기
  - 머신러닝은 제대로 된 선을 긋는 작업부터 시작됨
  - 미래를 예측하는 것도 가능



선형 회귀의 정의

- **"**학생들의 중간고사 성적이 **[   ]**에 따라 다 다르다.**"**
  - 이 문장은 정보가 담길 여지를 열어 놓고 있음

- 여기서 **[   ]**에 들어갈 내용을 '정보'라고 함
  - 머신러닝과 딥러닝은 이 정보가 필요함
  - 많은 정보가 더 정확한 예측을 가능케하며, 이때의 '많은 정보' 가 곧 '빅데이터'

선형 회귀의 정의

- 성적을 변하게 하는 ‘정보’ 요소를 **x**라고 하고, 이 **x** 값에 의해 변하는 ‘성적’을 **y**라 할 때, **x** 값이 변함에 따라 **y** 값도 변한다
  - 독립적으로 변할 수 있는 값 **x**를 독립 변수
  - 이 독립 변수에 따라 종속적으로 변하는 **y**를 종속 변수
- **선형 회귀**란 독립 변수 **x**를 사용해 종속 변수 **y**의 움직임을 예측하는 작업

선형 회귀의 정의

- 독립 변수 **x** 하나만으로는 정확히 설명할 수 없을 때는 **x** 값을 여러 개**(x1, x2, x3** 등**)** 준비한다.
  - 하나의 **x** 값만으로도 **y** 값을 설명 할 수 있을 때 이를 **단순 선형 회귀**
  - **x** 값이 여러 개 필요할 때는 **다중 선형 회귀**라고 함

가장 훌륭한 예측선이란?

- 독립 변수가 하나뿐인 단순 선형 회귀의 예
  - 성적을 결정하는 여러 요소 중에 '공부한 시간' 한 가지만 놓고 예측하는 경우
  - 중간고사를 본 **4**명의 학생에게 각각 공부한 시간을 물어보고 이들의 중간고사 성적을 표 **3-1**과 같이 정리했을 때
  - 공부한 시간을 **x**라 하고 성적을 **y**라 할때 집합 **x**와 집합 **y**를 다음과 같이 표현

$$x = \{2,\ 4,\ 6,\ 8\}$$

$$y = \{81,\ 93,\ 91,\ 97\}$$

공부한 시간	2시간	4시간	6시간	8시간
성적	81점	93점	91점	97점

표 3-1 공부한 시간과 중간고사 성적 데이터

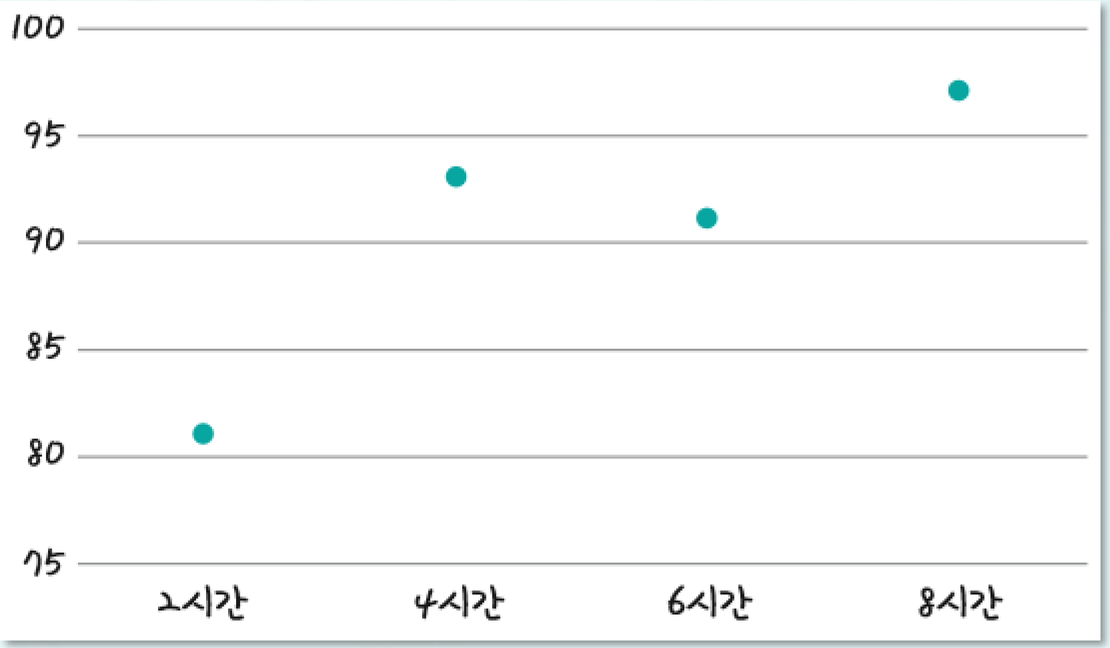


그림 3-1 표 3-1의 공부한 시간과 성적을 좌표로 표현

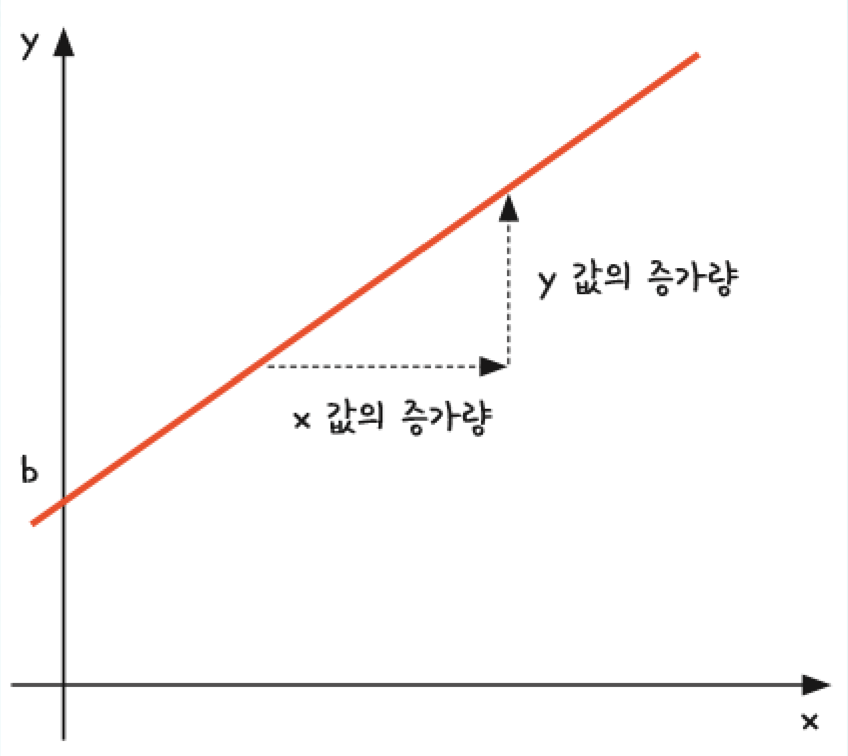
가장 훌륭한 예측선이란?

- 좌표 평면에 나타내 놓고 보니, 왼쪽이 아래로 향하고 오른쪽이 위를 향하는 일종의 '선형(직선으로 표시될 만한 형태)'을 보임
  - 선형 회귀는 이 점들의 특징을 가장 잘 나타내는 선을 그리는 과정
  - 여기에서 선은 직선이므로 곧 일차 함수 그래프이므로,

$$y = ax + b$$

- a는 직선의 기울기, $\frac{y\text{값의 증가량}}{x\text{값의 증가량}}$이고,

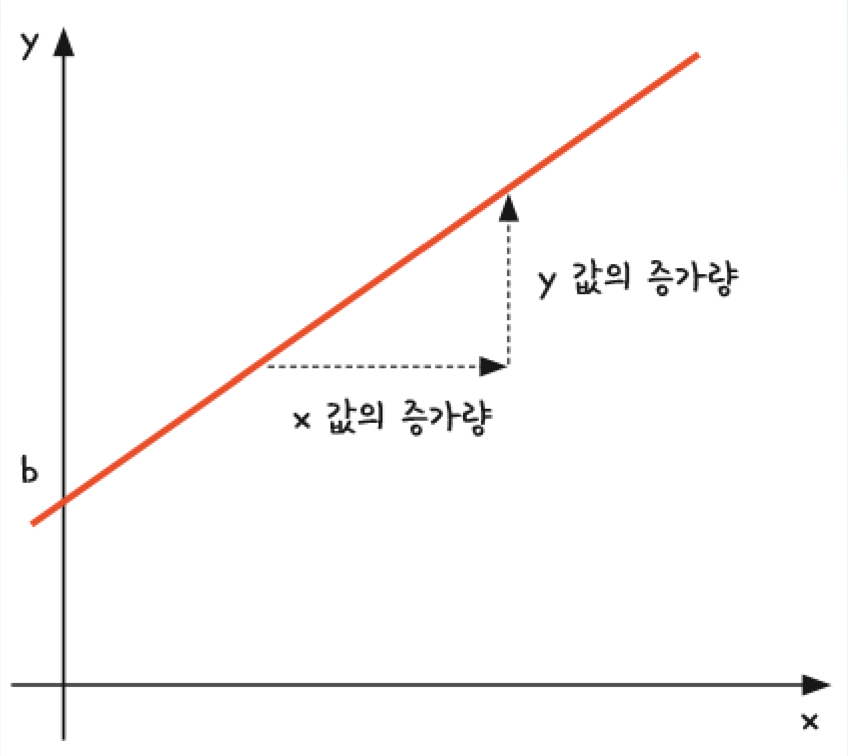
- b는 y축을 지나는 값인 'y 절편'이 됨



가장 훌륭한 예측선이란?

- 여기서 **x** 값은 독립 변수이고 **y** 값은 종속 변수
- 즉, **x** 값에 따라 **y** 값은 반드시 달라짐

- 다만, 정확하게 계산하려면 상수 **a**와 **b**의 값을 알아야 함
- 따라서 이 직선을 훌륭하게 그으려면
  - 직선의 기울기 **a** 값과
  - **y** 절편 **b** 값을 정확히 예측해 내야 한다



가장 훌륭한 예측선이란?

- 선형 회귀는 곧 정확한 직선을 그려내는 과정
  - 최적의 **a** 값과 **b** 값을 찾아내는 작업**!!**

- 예측선을 그리는 이유
  - 잘 그어진 직선을 통해 우리는 표 **3-1**의 공부한 시간과 중간고사 성적 데이터에 들어 있지 않은 여러 가지 내용을 유추할 수 있다.
  - 예를 들어**,** 표 **3-1**에 나와 있지 않은 또 다른 학생의 성적을 예측하고 싶을 때**,** 정확한 직선을 그어 놓았다면 이 학생이 몇 시간을 공부했는지만 물어보면 됨
  - 정확한 **a**와 **b**의 값을 따라 움직이는 직선에 학생이 공부한 시간인 **x** 값을 대입하면 예측 성적인 **y** 값을 구할 수 있는 것

가장 훌륭한 예측선이란?

- 딥러닝과 머신러닝의 '예측' 이란?
  - 기존 데이터(정보)를 가지고 어떤 선이 그려질지를 예측한 뒤,
  - 아직 답이 나오지 않은 그 무언가를 그 선에 대입해 보는 것

- 선형 회귀의 개념을 이해하는 것은 딥러닝을 이해하는 중요한 첫걸음

최소 제곱법

- 정확한 기울기 **a**와 정확한 **y** 절편의 값 **b**를 알아내는 간단한 방법
- 최소 제곱법이 : 회귀 분석에서 사용되는 표준 방식으로, 실험이나 관찰을 통해 얻은 데이터를 분석하여 미지의 상수를 구할 때 사용되는 공식
- 최소 제곱법 공식을 알고 적용한다면 일차 함수의 기울기 **a**와 **y** 절편 **b**를 바로 구할 수 있음

최소 제곱법

- 지금 가진 정보가 **x** 값(입력 값, 여기서는 '공부한 시간')과 **y** 값(출력 값, 여기서는 '성적')일 때 최소 제곱법을 이용해 기울기 **a**를 구하는 방법
  - 각 **x**와 **y**의 편차를 곱해서 이를 합한 값을 구함
  - 그리고 이를 **x** 편차 제곱의 합으로 나눔

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

  - 식으로 표현하면

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

최소 제곱법

- 성적(y)과 공부한 시간(x)을 가지고 최소 제곱법으로 기울기 **a**를 구하려면
  1. **x** 값의 평균과 **y** 값의 평균을 각각 구한다.
     - 공부한 시간(x) 평균: (2 + 4 + 6 + 8) ÷ 4 = 5
     - 성적(y) 평균: (81+ 93 + 91 + 97) ÷ 4 = 90.5
  2. 이를 식에 대입한다.

→ 기울기는 2.3!

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

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

$$= 2.3$$

최소 제곱법

- 다음은 **y** 절편인 **b**를 구하는 공식

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

- **y**의 평균에서 **x**의 평균과 기울기의 곱을 빼면 **b**의 값이 나온다는 의미식으로 표현하면 다음과 같음

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

최소 제곱법

- 우리는 이미 **y**평균, **x**평균, 그리고 조금 전 구한 기울기 **x**까지, 이 식을 풀기 위해 필요한 모든 변수를 알고 있음

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

→ y 절편 b는 79!

- 이를 식에 대입해 보면, $y = 2.3x + 79$
- 이제 다음과 같이 예측 값을 구하기 위한 직선의 방정식이 완성됨

최소 제곱법

- 이 식에 **x**를 대입했을 때 나오는 **y** 값을 '예측 값'으로 정리하면

공부한 시간	2	4	6	8
성적	81	93	91	97
예측 값	**83.6**	**88.2**	**92.8**	**97.4**

표 3-2 최소 제곱법 공식으로 구한 성적 예측 값

- 좌표 평면에 이 예측 값을 찍어 보자

그림 3-3 공부한 시간, 성적, 예측 값을 좌표로 표현

최소 제곱법

- 예측한 점들을 연결해 직선을 그려보자

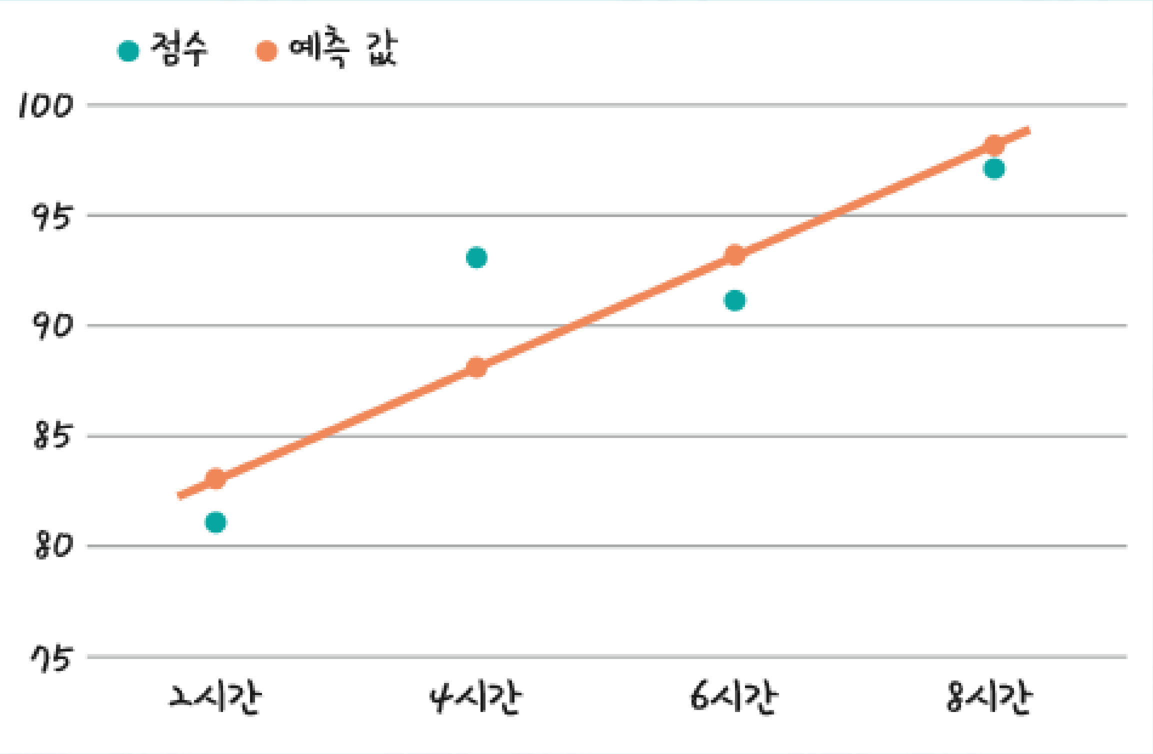


그림 3-4 오차가 최저가 되는 직선의 완성

최소 제곱법

- 이것이 바로 오차가 가장 적은, 주어진 좌표의 특성을 가장 잘 나타내는 직선
  → 우리가 원하는 예측 직선
- 이 직선에 우리는 다른 **x** 값(공부한 시간)을 집어넣어서 '공부량에 따른 성적을 예측'할 수 있음

코딩으로 확인하는 최소 제곱

- 넘파이 라이브러리는 **np**라는 이름으로 사용할 수 있도록 설정

```
import numpy as np
x = [2, 4, 6, 8]
y = [81, 93, 91, 97]
```

- 앞서 나온 데이터 값을 '리스트' 형식으로 다음과 같이 **x**와 **y**로 정의

- 넘파이 함수 **mean()**로
  x, y의 모든 원소들의 평균을 구함
- **mx**에 **x** 원소들의 평균값을,
  my에 **y** 원소들의 평균값을 저장

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

코딩으로 확인하는 최소 제곱

- 앞서 살펴본 최소 제곱근 공식 중 분모의 값, 즉 '**x**의 평균값과 **x**의 각 원소들의 차를 제곱하라'

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

- **divisor**라는 변수를 만들어 위 식을 파이썬으로 구현해 저장

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

코딩으로 확인하는 최소 제곱

- 이제 분자에 해당하는 부분을 구해보자
- **top** 함수에 **dividend** 변수에 분자의 값을 저장

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

```
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)
```

- 임의의 변수 **d**의 초깃값을 **0**으로 설정한 뒤 **x**의 개수만큼 실행
- **d**에 **x**의 각 원소와 평균의 차, **y**의 각 원소와 평균의 차를 곱해서 차례로 더하는 최소 제곱법을 그대로 구현

코딩으로 확인하는 최소 제곱

- 앞에서 구한 분모와 분자를 계산하여 기울기 **a**를 구함

```
a = dividend / divisor
```

- **a**를 구하고 나면 **y** 절편을 구하는 공식을 이용해 **b**를 구할 수 있음

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

```
b = my - (mx*a)
```

코딩으로 확인하는 최소 제곱

```python
import numpy as np

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

# x와 y의 평균값
mx = np.mean(x)
my = np.mean(y)
print("x의 평균값:", mx)
print("y의 평균값:", my)

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

```python
# 기울기 공식의 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)
print("분모:", divisor)
print("분자:", dividend)
# 기울기와 y 절편 구하기
a = dividend / divisor
b = my - (mx*a)
# 출력으로 확인
print("기울기 a =", a)
print("y 절편 b =", b)
```

코딩으로 확인하는 최소 제곱

- 실행결과

```
x의 평균값: 5.0
y의 평균값: 90.5
분모: 20.0
분자: 46.0
기울기 a = 2.3
y 절편 b = 79.0
```

- 기울기 a의 값과 y 절편 b의 값이 각각 2.3과79임을 알 수 있다.

잘못 그은 선 바로잡기

- 최소 제곱법의 한계
- '여러 개의 입력(x)'값이 있는 경우 이 공식만으로 처리할 수 없다.

- 딥러닝은 대부분 입력 값이 여러 개인 상황에서 이를 해결해야 함!

- 여러 개의 입력 값을 계산하는 방법
  - 임의의 선을 그리고 난 후
  - 이 선이 얼마나 잘 그려졌는지를 평가하여
  - 조금씩 수정해 가는 방법을 사용

- 이를 위해 주어진 선의 오차를 평가하는 오차 평가 알고리즘이 필요

  → 가장 많이 사용되는 방법: 평균 제곱근 오차**(root mean square error)**

잘못 그은 선 바로잡기

- ‘일단 그리고 조금씩 수정해 나가기’ 방식에 대하여
  - 가설을 하나 세운 뒤 이 값이 주어진 요건을 충족하는지를 판단하여 조금씩 변화를 주고, 이 변화가 긍정적이면 오차가 최소가 될 때까지 이 과정을 계속 반복하는 방법

- 나중에 그린 선이 먼저 그린 선보다 더 좋은지 나쁜지를 판단하기 위해 필요한 것은?
  - 각 선의 오차를 계산할 수 있어야 한다.
  - 이 오차가 작은 쪽으로 바꾸는 알고리즘이 필요하다.

잘못 그은 선 바로잡기

- 대강의 선을 긋기 위해 기울기 **a**와 **y** 절편 **b**를 임의의 수 **3**과 **76**이라고 가정하면

  y = 3x + 76 직선을 그을 수 있다.

- 임의의 직선이 어느 정도의 오차가 있는지를 확인하려면 각 점과 그래프 사이의 거리를 재면 됨
- 이 거리들의 합이 작을수록 잘 그어진 직선이고, 이 직선들의 합이 클수록 잘못 그어진 직선이 됨

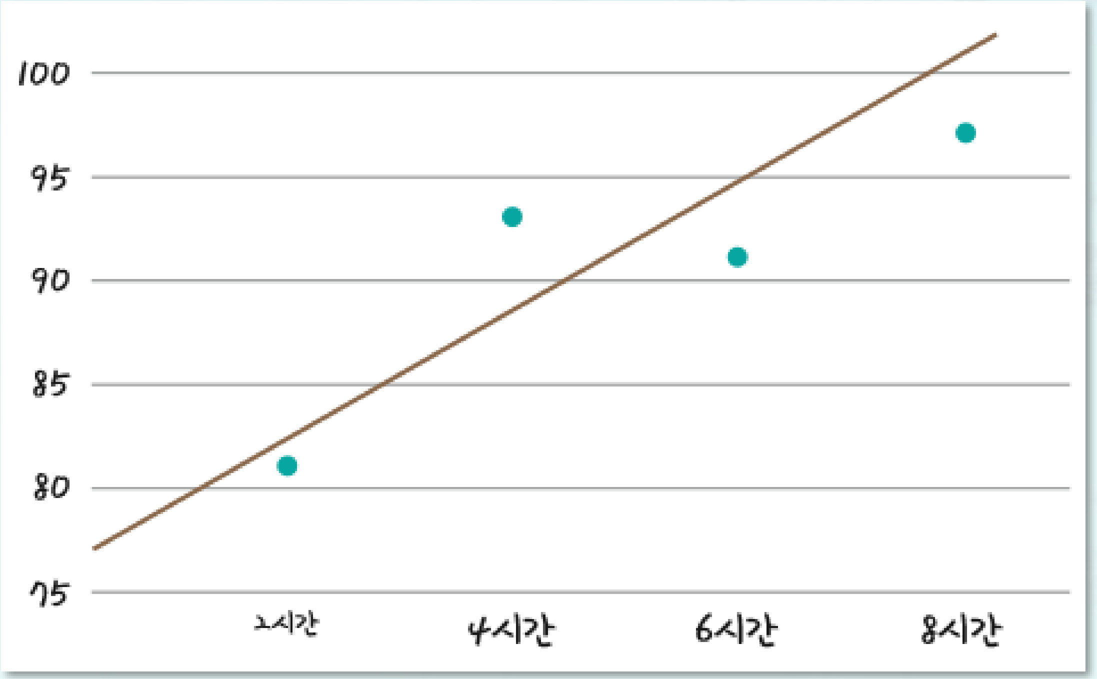


그림 3-6 임의의 직선 그려보기

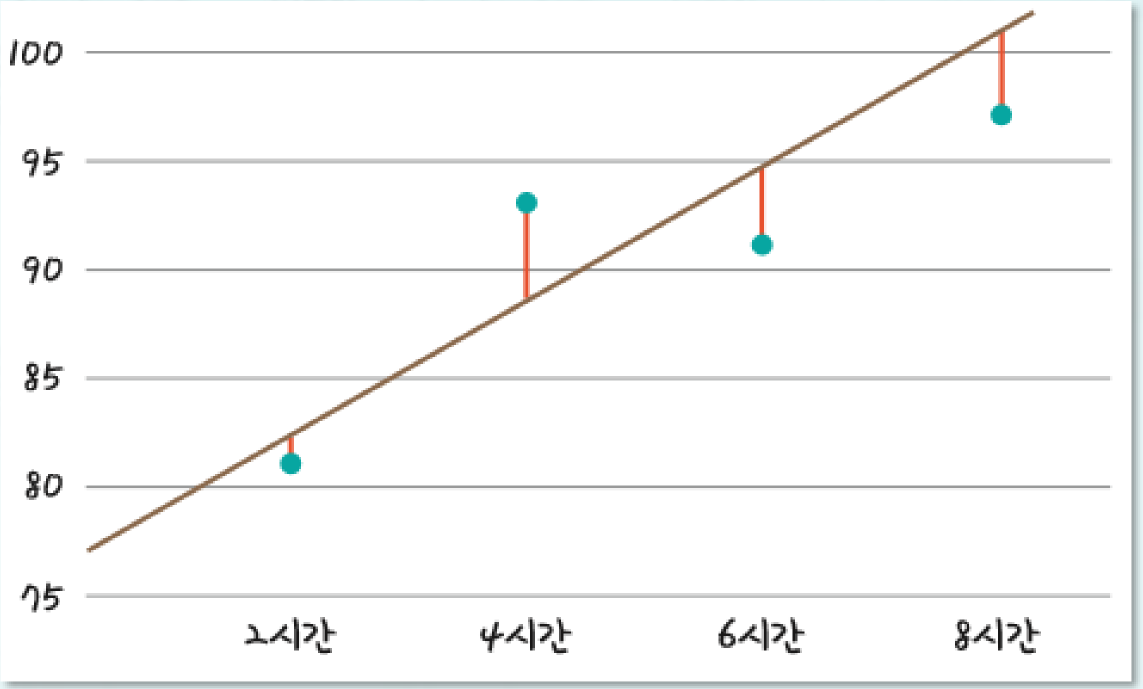


그림 3-7 임의의 직선과 실제 값 사이의 거리

잘못 그은 선 바로잡기

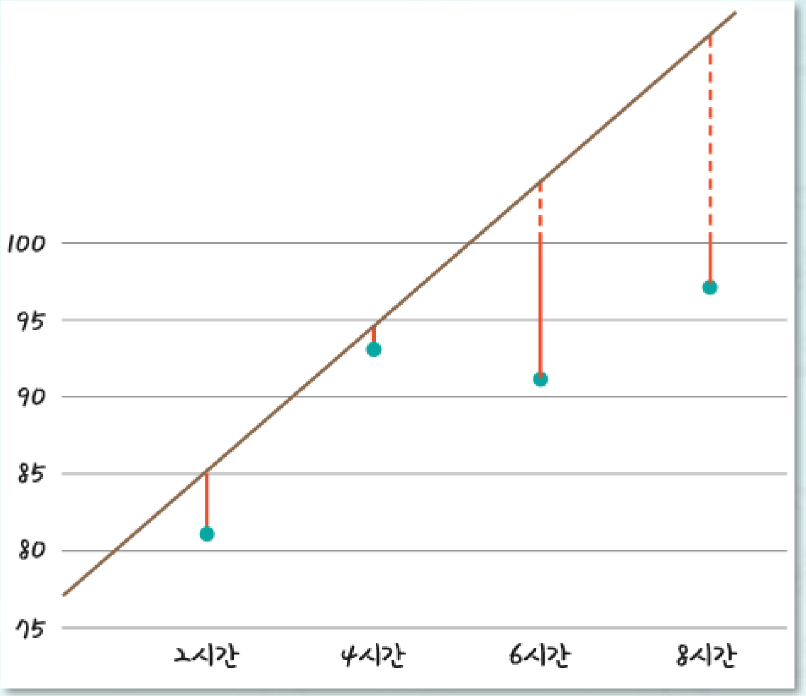


그림 3-8 기울기를 너무 크게 잡았을 때의 오차

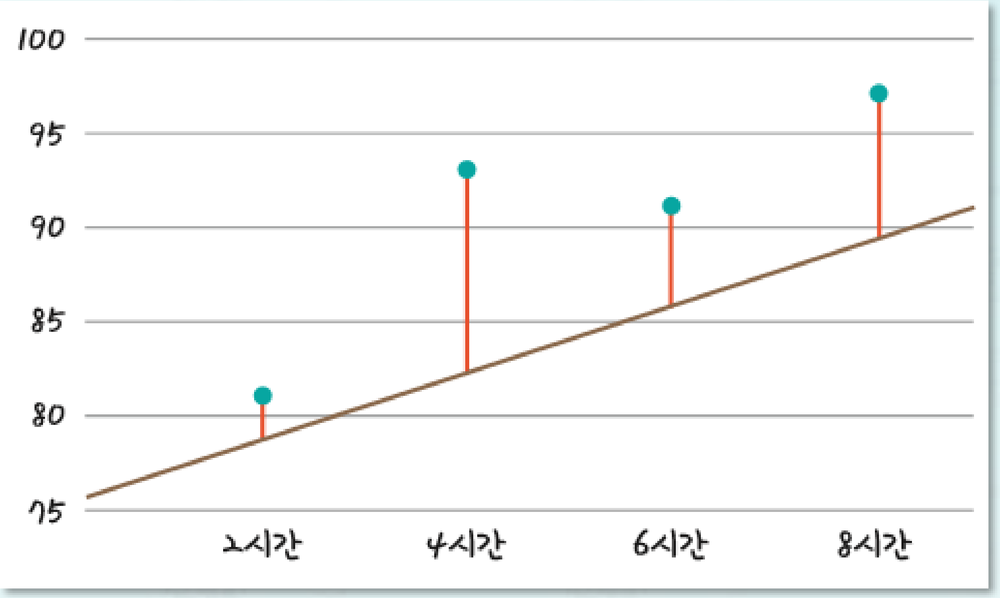


그림 3-9 기울기를 너무 작게 잡았을 때의 오차

- 그래프의 기울기가 잘못 되었을 수록 빨간색 선의 거리의 합, 즉 오차의 합도 커짐
  → 만약 기울기가 무한대로 커지면 오차도 무한대로 커지는 상관관계가 있다!

잘못 그은 선 바로잡기

- 빨간색 선의 거리의 합을 실제로 계산해 보자
- 거리는 입력 데이터에 나와 있는 **y**의 '실제 값'과 **x**를 $y = 3x + 76$의 식에 대입해서 나오는 '예측 값'과의 차이를 통해 구할 수 있음

- 예. **2**시간 공부했을 때의 실제 나온 점수(**81**점)와 그래프 $y = 3x + 76$식에 $x = 2$를 대입했을 때(**82**점)의 차이가 곧 오차

  → 오차를 구하는 방정식 (오차 = 실제 값 – 예측 값)

잘못 그은 선 바로잡기

- 이 식에 주어진 데이터를 대입하여 얻을 수 있는 모든 오차의 값을 정리하면

공부한 시간(x)	2	4	6	8
성적(실제 값, y)	81	93	91	97
예측 값	82	88	94	100
오차	1	−5	3	3

표 3-3 주어진 데이터에서 오차 구하기

- 부호를 없애야 정확한 오차를 구할 수 있음. 따라서 오차의 합을 구할 때는 각 오차의 값을 제곱해 준다

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

- 여기서 **i**는 **x**가 나오는 순서를, **n**은 **x** 원소의 총 개수를 의미
- **pi**는 **xi**에 대응하는 '실제 값'이고 **yi**는 **xi**가 대입되었을 때 직선의 방정식(여기서는 **y = 3x + 76)**이 만드는 '예측 값'
- 이 식에 의해 오차의 합을 다시 계산하면 **1 + 25 + 9 + 9 = 44**

잘못 그은 선 바로잡기

- 오차의 합을 **n**으로 나누면 오차 합의 평균을 구할 수 있음
  → 평균 제곱 오차**(Mean Squared Error, MSE)**

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

- 이 식은 앞으로 머신러닝과 딥러닝을 공부할 때 자주 등장하는 중요한 식**!**
- 이 식에 따라 우리가 앞서 그은 임의의 직선은 **44/4 = 11**의 평균 제곱 오차를 갖는 직선이라고 말할 수 있음

잘못 그은 선 바로잡기

- 여기에 다시 제곱근을 씌워 주면, 평균 제곱근 오차**(Root Mean Squared Error, RMSE)**라고 함

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

- 앞서 그은 직선의 평균 제곱근 오차는 $\sqrt{11}$ **= 3.3166…**이 된다.

- 평균 제곱 오차 또는 평균 제곱근 오차는 오차를 계산해서 앞선 추론이 잘 되었는지 평가하는 대표적인 공식

잘못 그은 선 바로잡기

- 잘못 그은 선 바로잡기는 곧 '평균 제곱근 오차'의 계산 결과가 가장 작은 선을 찾는 작업

- 선형 회귀란?
  → 임의의 직선을 그어 이에 대한 평균 제곱근 오차를 구하고
  → 이 값을 가장 작게 만들어 주는 **a**와 **b** 값을 찾아가는 작업!

잘못 그은 선 바로잡기

- 평균 제곱근 오차를 파이썬으로 구현해 보자

- 먼저 임의로 정한 기울기 **a**와 **y** 절편 **b**의 값이 각각 **3**과 **76**이라고 할 때 리스트 '**ab**'를 만들어 여기에 이 값을 저장

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

- '**data**'라는 리스트를 만들어 공부한 시간과 이에 따른 성적을 각각 짝을 지어 저장
- 그리고 **x** 리스트와 **y** 리스트를 만들어
  첫 번째 값을 **x** 리스트에 저장하고 두 번째 값을 **y** 리스트에 저장

```
data = [[2, 81], [4, 93], [6, 91], [8, 97]]
x = [i[0] for i in data]
y = [i[1] for i in data]
```

잘못 그은 선 바로잡기

- **predict()**라는 함수를 사용해 일차 방정식 **y = ax + b**를 구현

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

- 평균 제곱근 공식 $\sqrt{\frac{1}{n}\sum_{i=1}^{n}(p_i - y_i)^2}$ 을 그대로 파이썬 함수로 옮기면 다음과 같음

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

- **np.sqrt()**은 제곱근을, ****2**는 제곱을 구하라는 뜻
- **mean()**은 평균값을 구하라는 뜻
- 예측 값과 실제 값을 각각 **rmse()**라는 함수의 **p**와 **a** 자리에 입력해서 평균 제곱근을 구함

잘못 그은 선 바로잡기

- 이제 **rmse()** 함수에 데이터를 대입하여 최종값을 구하는 함수 **rmse_val()**을 만들어 보자

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

- **predict_result**에는 앞서 만든 일차 방정식 함수 **predict()**의 결과값이 들어감
- 이 값과 **y** 값이 각각 예측 값과 실제 값으로 **rmse()** 함수 안에 들어가게 됨

잘못 그은 선 바로잡기

- 이제 모든 **x** 값을 **predict()** 함수에 대입하여 예측 값을 구하고, 이 예측 값과 실제 값을 통해 최종값을 출력하는 코드를 다음과 같이 작성

```python
# 예측 값이 들어갈 빈 리스트를 만든다.
predict_result = []

# 모든 x 값을 한 번씩 대입하여
for i in range(len(x)):
    #그 결과 predict_result 리스트 완성한다.
    predict_result.append(predict(x[i]))
    print("공부한 시간 = %.f, 실제 점수 = %.f,
예측 점수 = %.f" % (x[i], y[i],
    predict(x[i])))
```

```python
import numpy as np

# 기울기 a와 y 절편 b
ab = [3, 76]
# x, y의 데이터 값
data = [[2, 81], [4, 93], [6, 91], [8, 97]]
x = [i[0] for i in data]
y = [i[1] for i in data]
# y = ax + b에 a와 b 값을 대입하여 결과를 출력하
는 함수
def predict(x):
    return ab[0]*x + ab[1]
```

잘못 그은 선 바로잡기

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

# RMSE 함수를 각 y 값에 대입하여 최종 값을 구하는 함수
def rmse_val(predict_result,y):
    return rmse(np.array(predict_result), np.array(y))

# 예측 값이 들어갈 빈 리스트
predict_result = []

# 모든 x 값을 한 번씩 대입하여
for i in range(len(x)):
    # predict_result 리스트를 완성한다.
    predict_result.append(predict(x[i]))
    print("공부한 시간 = %.f, 실제 점수 = %.f, 예측 점수 = %.f" % (x[i], y[i], predict(x[i])))

# 최종 RMSE 출력
print("rmse 최종값: " + str(rmse_val(predict_result,y)))
```

잘못 그은 선 바로잡기

- 실행 결과

```
공부한 시간=2, 실제 점수=81, 예측 점수=82
공부한 시간=4, 실제 점수=93, 예측 점수=88
공부한 시간=6, 실제 점수=91, 예측 점수=94
공부한 시간=8, 실제 점수=97, 예측 점수=100
rmse 최종값: 3.31662479036
```

잘못 그은 선 바로잡기

- 이를 통해 우리가 처음 가정한 **a = 3, b = 76**은 오차가 약 **3.3166**이라는 것을 알게 됨
- 이제 남은 것은 이 오차를 줄이면서 새로운 선을 긋는 것
- 이를 위해서는 **a**와 **b**의 값을 적절히 조절하면서 오차의 변화를 살펴보고**,** 그 오차가 최소화되는 **a**와 **b**의 값을 구해야 함