4. 가장 훌륭한 예측선 긋기: 선형 회귀

서론

- 딥러닝은 자그마한 통계의 결과들이 무수히 얽히고설켜 이루어지는 복잡한 연산의 결정체입니다.
- 우리 몸을 이해하려면 몸을 구성하는 기본 단위인 세포의 역할을 알아야 하듯 딥러닝을 이해하려면 딥러닝의 가장 말단에서 이루어지는 가장 기본적인 두가지 계산원리를 알아야 합니다.
- 바로 선형회귀와 로지스틱회귀 입니다.
- 이를 알고 나면 딥러닝을 구동시키는 원리에 한 걸음 다가설 수 있습니다.

서론

- 가장 훌륭한 예측선 긋기란 통계학 용어인 선형 회귀(linear regression)를 쉽게 풀어서 쓴 것입니다.
- 머신러닝은 제대로 된 선을 긋는 작업부터 시작됩니다.
- 선의 방향을 잘 정하면 그 선을 따라가는 것만으로도 지금은 보이지 않는 미래의 것을 예측할 수 있기 때문이지요.

선형 회귀의 정의

- “학생들의 중간고사 성적이 다 다르다.”
- 네, 다르겠죠.
- 그런데 위 문장이 나타낼 수 있는 정보는 너무 제한적입니다
- 학급의 학생마다 제각각 성적이 다르다는 당연한 사실 외에는 알 수 있는 게 없습니다.
- 이번에는 다음 문장을 보겠습니다.

선형 회귀의 정의

- “학생들의 중간고사 성적이 [ ]에 따라 다 다르다.”
- 이 문장은 정보가 담길 여지를 열어 놓고 있습니다.
- [ ] 부분에 시험 성적을 좌우할 만한 여러 가지 것이 들어간다면 좀 더 많은 사실을 전달할 수 있습니다.
- 예를들면 공부한 시간, 시험 당일의 컨디션, 사교육비 지출액 등이 들어갈 수 있겠지요.
- 무엇이 들어가든지 해당 성적의 이유를 나름대로 타당하게 설명할 수 있습니다.
- 따라서 이 문장이 중간고사 성적의 차이와 이유를 나타낼 때 더욱 효과적입니다.

선형 회귀의 정의

- 여기서 [ ] 에 들어갈 내용을 '정보'라고 합니다.
- 머신러닝과 딥러닝은 이 정보가 필요합니다.
- 정보를 정확히 준비해 놓기만 하면 성적을 예측하는 방정식을 만들 수도 있습니다.
- 이 단순한 정의를 이번에는 좀 더 수학적인 언어로 표현해 보겠습니다.
- 성적을 변하게 하는 '정보' 요소를 x라고 하고, 이 x 값에 의해 변하는 '성적'을 y라 합시다.

선형 회귀의 정의

- x 값이 변함에 따라 y 값도 변한다는 이 정의 안에서, 독립적으로 변할 수 있는 값 x를 독립 변수라고 합니다.
- 또한, 이 독립 변수에 따라 종속적으로 변하는 y를 종속 변수라고 합니다.
- 선형 회귀란 독립 변수 x를 시용해 종속 변수 y의 움직임을 예측하고 설명하는 작업을 말합니다.
- 독립 변수가 x 하나뿐이어서 이것만으로는 정확히 설명할 수 없을 때는 x_1, x_2, x_3 등 x 값을 여러 개 준비해 놓을 수도 있습니다.
- 하나의 x 값만으로도 y 값을 설명할 수 있을 때 이를 단순 선형 회귀 (simple linear regression)라고 합니다.
- 또한, x 값이 여러 개 필요할 때는 다중 선형 회귀 (muliple linear regression) 라고 합니다.

가장 훌륭한 예측선이란?

- 우선 독립 변수가 하나뿐인 단순 선형 회귀의 예를 공부해 봅시다.
- 성적을 결정하는 여러 요소중에 '공부한 시간' 한 가지만 놓고 생각해 보겠습니다.
- 중간고사를 본 4 명의 학생에게 각각 공부한 시간을 물어보고 이들의 중간고사 성적을 다음 표와 같이 정리했다고 해 봅시다.

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

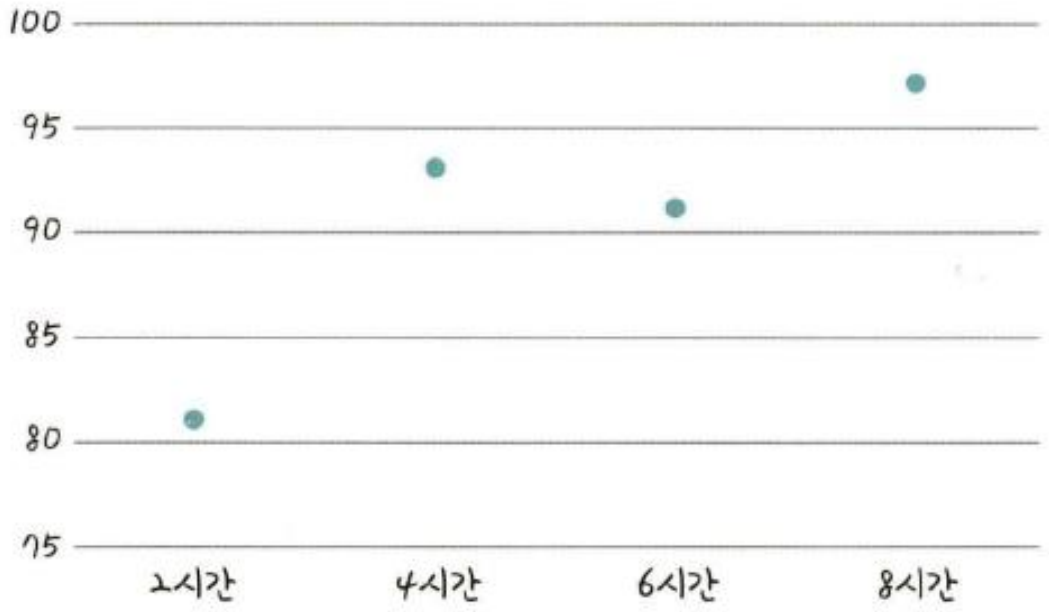
가장 훌륭한 예측선이란?

- 여기서 공부한 시간을 x라 하고 성적을 y라 할 때 집합 x와 집합 y를 다음과 같이 표현할 수 있습니다.

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

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

- 이를 좌표 평면에 나타내면 그림과 같습니다.



가장 훌륭한 예측선이란?

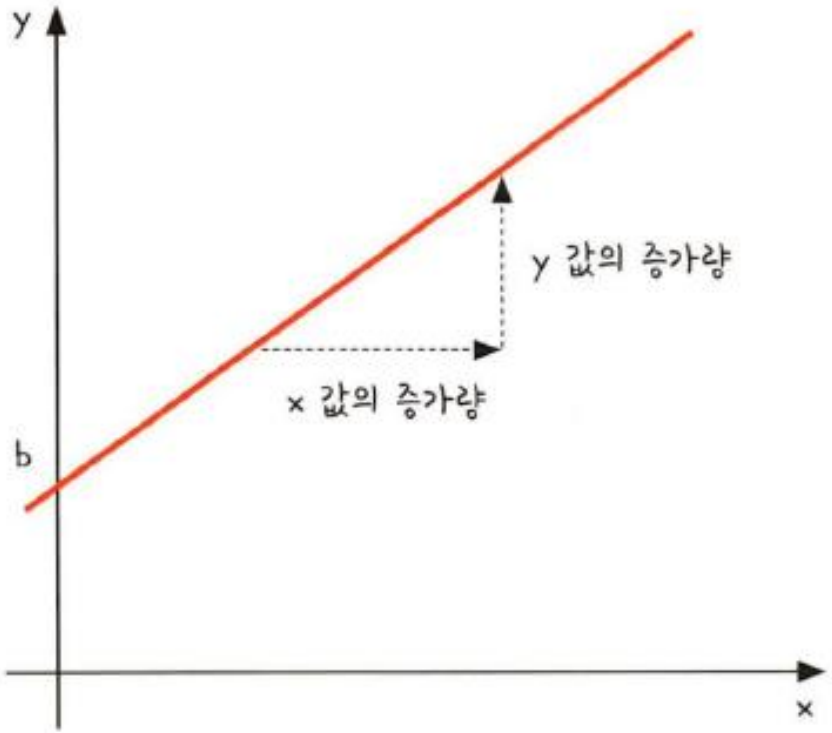
- 좌표 평면에 나타내 놓고 보니 왼쪽이 아래로 향하고 오른쪽이 위를 향하는 일종의 '선형(직선으로 표시될 만한 형태)'을 보입니다.
- 선형 회귀를 공부하는 과정은 이 점들의 특정을 가장 잘 나타내는 선을 그리는 과정과 일치합니다.
- 여기에서 선은 직선이므로 곧 일차함수그래프입니다.

가장 훌륭한 예측선이란?

- 일차함수 그래프는 다음과 같은 식으로 표현할 수 있습니다.

$$y = ax + b$$

  - a는 직선의 기울기이고, b는 y축을 지나는 값인 'y 절편'이 되겠지요.



가장 훌륭한 예측선이란?

- 여기서 x 값은 독립 변수이고 y 값은 종속변수입니다
- 즉, x 값에 따라 y 값은 반드시 달라집니다.
- 다만, 정확하게 계산하려면 상수 a와 b의 값을 알아야 합니다.
- 따라서 이 직선을 훌륭하게 그으려면 직선의 기울기 a값과 y 절편 b값을 정확히 예측해 내야 하는 것입니다.

가장 훌륭한 예측선이란?

- 앞서, 선형 회귀는 곧 정확한 직선을 그려내는 과정이라고 했습니다.
- 바꿔 말하면, 선형 회귀는 결국 최적의 a값과 b값을 찾아내는 작업이라고 할 수 있습니다.
- 직선을 잘 긋는 것이 어째서 중요할까요?
- 잘 그어진 직선을 통해 우리는 앞에서 본 표의 공부한 시간과 중간고사 성적 데이터에 들어 있지 않은 여러 가지 내용을 유추할 수 있기 때문입니다.

가장 훌륭한 예측선이란?

- 예를 들어, 표에 나와 있지 않은 또 다른 학생의 성적을 예측하고 싶다고 합시다.
- 이때 정확한 직선을 그어 놓았다면 이 학생이 몇 시간을 공부했는지만 물어보면 됩니다
- 정확한 a와 b의 값을 따라 움직이는 직선에 학생이 공부한 시간인 x 값을 대입하면 예측 성적인 y 값을 구할 수 있는 것입니다.

가장 훌륭한 예측선이란?

- 딥러닝을 포함한 머신러닝의 예측은 결국 이러한 기본 접근 방식과 크게 다르지 않습니다.
- 기존 데이터 (정보)를 가지고 어떤 선이 그려질지를 예측한 뒤, 아직 답이 나오지 않은 그 무언가를 그 선에 대입해 보는 것이지요
- 따라서 선형 회귀의 개념을 이해하는 것은 딥러닝을 이해하는 중요한 첫걸음입니다.

최소 제곱법

- 이제 우리의 목표는 가장 정확한 직선을 긋는 것입니다.
- 더 구체적으로는 정확한 기울기 a와 정확한 y 절편의 값 b를 알아내면 된다고 했습니다.
- 그런데 만일 우리가 최소 제곱법 (method of least squares)이라는 공식을 알고 적용한다면, 최소 제곱법을 통해 일차함수의 기울기 a와 절편 b를 바로 구할 수 있습니다.
- 최소 제곱법이란 회귀 분석에서 사용되는 표준 방식입니다.
- 실험이나 관찰을 통해 얻은 데이터를 분석하여 미지의 상수를 구할 때 사용되는 공식이지요.

최소 제곱법

- 지금 가진 정보가 x 값(입력 값, 여기서는 '공부한 시간'과 y 값(출력 값, 여기서는 '성적')일 때 이를 이용해 기울기 a를 구하는 방법은 다음과 같습니다.

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

- 이것이 바로 최소 제곱법입니다.
- 뭔가 복잡해 보이지만, 의미는 어렵지 않습니다.

최소 제곱법

- 각 x와 y의 편차를 곱해서 이를 합한 값을 구합니다.
- 그리고 이를 x편차 제곱의 합으로 나누면 우리가 원하는 기울기를 구할 수 있습니다.
- 이를 식으로 표현하면 다음과 같습니다.

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

- 이 공식이 맞는지 우리가 가진 성적 (y)과 공부한 시간(x) 데이터를 넣어 기울기 a를 구해봅시다.

최소 제곱법

- 먼저 x값의 평균과 y 값의 평균을 구해 보면 다음과 같습니다.
  - 공부한 시간(x) 평균: (2+4+6+8)/4 = 5
  - 성적(y) 평균: (81+93+91+97)/4 = 90.5
- 이를 위 식에 대입하면 다음과 같습니다.

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

$$= \frac{46}{20} = 2.3$$

최소 제곱법

- 기울기 a는 2.3 이 나왔네요!
- 다음은 y절편인 b를 구하는 공식입니다.
  - b = y의 평균 – (x의 평균 * 기울기 a)
- y의 평균에서 x의 평균과 기울기의 곱을 빼면 b의 값이 나온다는 의미입니다.
- 이를 식으로 표현하면 다음과 같습니다.

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

최소 제곱법

- 우리는 이미 y평균, X평균, 그리고 조금 전 구한 기 울기 x까지, 이 식을 풀기 위해 필요한 모든 변수를 알고 있습니다.
- 이를 식에 대입해 보겠습니다.

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

- y 절편 b는 79가 나왔습니다.
- 이제 다음과 같이 예측 값을 구하기 위한 직선의 방정식이 완성되었습니다.

$$y = 2.3x + 79$$

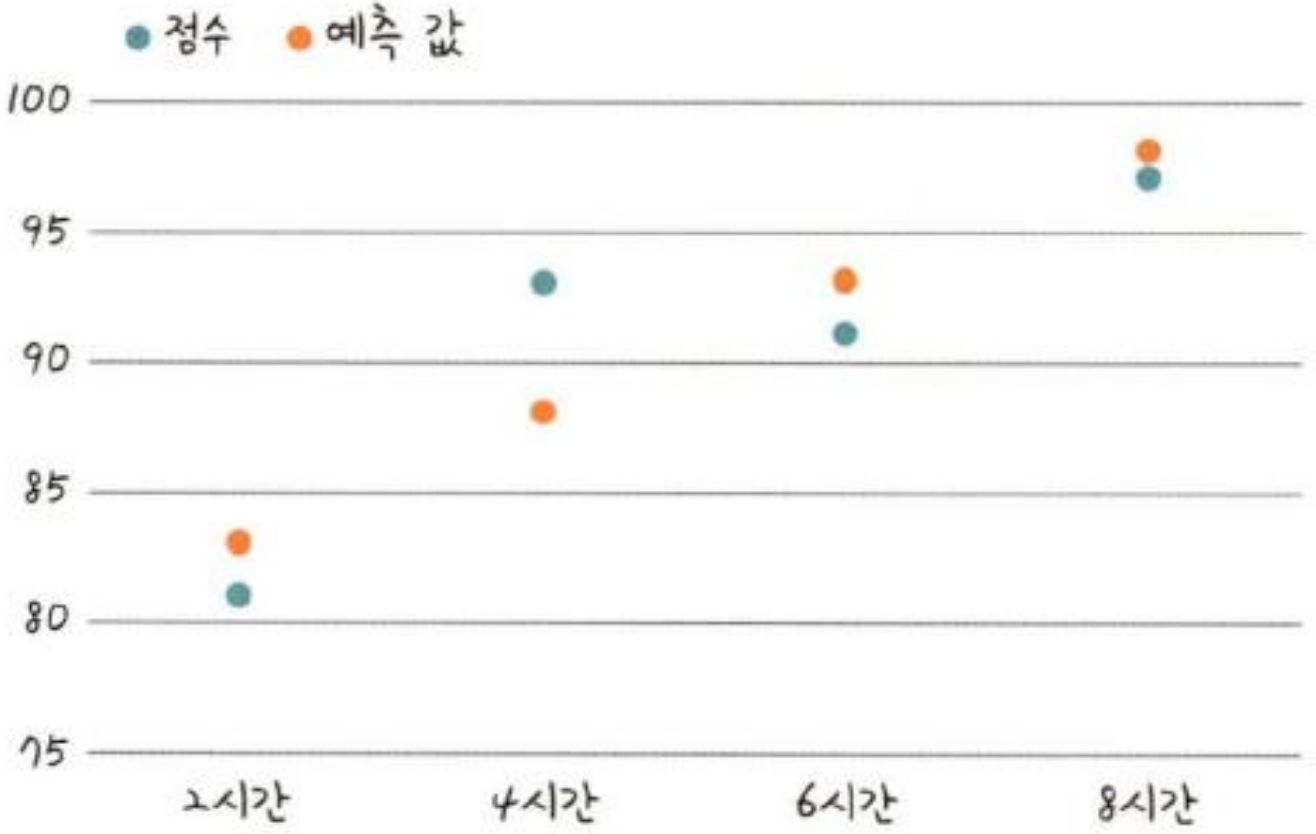
최소 제곱법

- 이 식에 우리가 가진 데이터를 대입해 보겠습니다.
- x를 대입했을 때 나오는 y 값을 '예측 값'이라고 부르겠습니다.

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

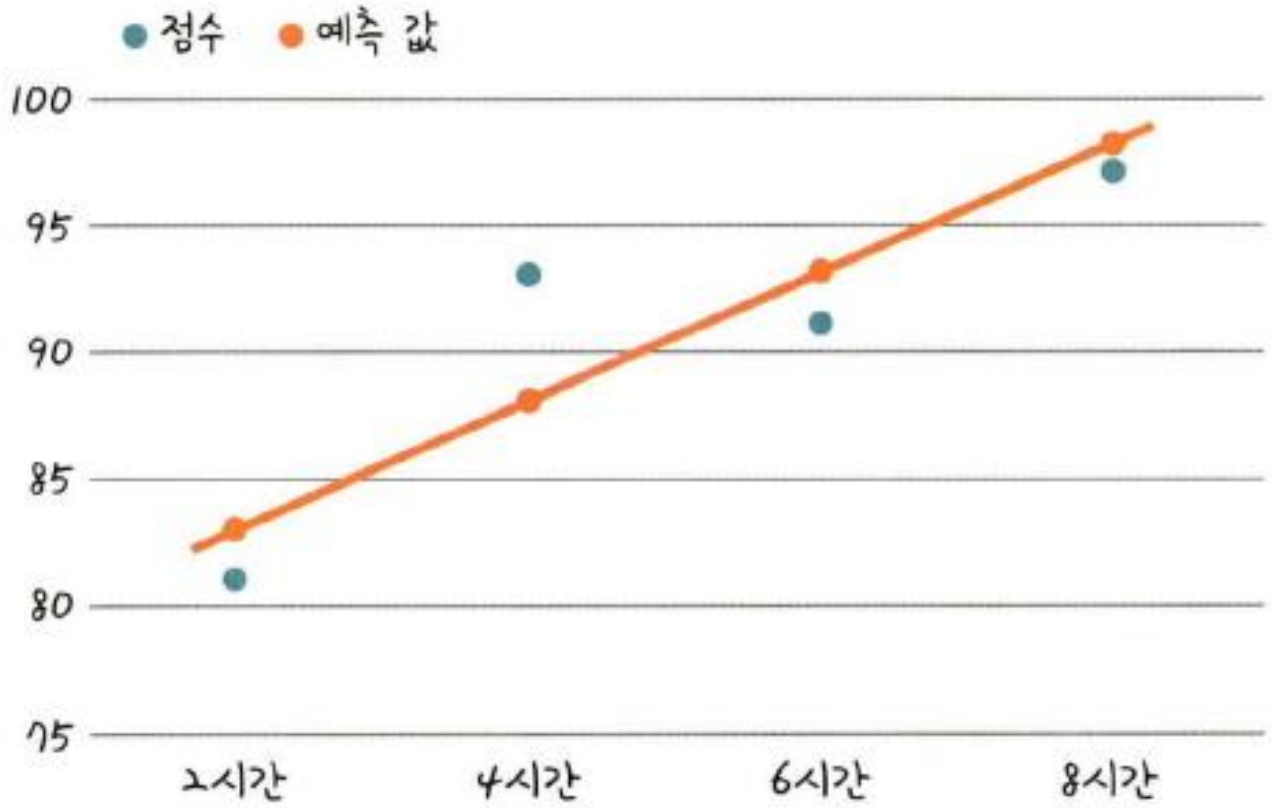
최소 제곱법

- 좌표 평면에 이 예측 값을 찍어 보겠습니다.



최소 제곱법

- 예측한 점들을 연결해 직선을 그으면 그림과 같습니다.



최소 제곱법

- 이것이 비로 오차가 가장 적은, 주어진 좌표의 특성을 가장 잘 나타내는 직선입니다.
- 우리가 원하는 예측 직선이지요.
- 이 직선에 우리는 다른 x 값(공부한 시간)을 집어 넣어서 '공부량에 따른 성적을 예측'할 수 있습니다.

코딩으로 확인하는 최소 제곱

- 우리가 이론을 배우는 목적은 딥러닝을 구현하기 위해서 입니다.
- 지금까지 공부한 내용을 코딩으로 구현해 보겠습니다.
- 먼저 넘파이 라이브러리를 불러와 간단히 np라는 이름으로 사용할 수 있게 설정하겠습니다.
- 그리고 앞서 나온 데이터 값을 '리스트' 형식 으로 다음과 같이 x와 y로 정의합니다.

```
import numpy as np

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

코딩으로 확인하는 최소 제곱

- 이제 최소 제곱근 공식에 의해 기울기 a나 절편 b의 값을 구해 보겠습니다.
- x의 모든 원소의 평균을 구하는 넘파이 함수는 mean ( )입니다
- mx라는 변수에 x 원소들의 평균값을, my에 y 원소들의 평균값을 입력합니다.

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

코딩으로 확인하는 최소 제곱

- 이제 앞서 살펴본 최소 제곱근 공식 중 분모의 값, 즉 'x의 평균값과 x의 각 원소들의 차를 제곱하라'는 파이썬 명령을 만들 차례입니다.
- 해당 식을 옮겨 보면 다음과 같습니다.

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

- 다음과 같이 divisor라는 변수를 만들어 위 식을 파이썬으로 구현해 저장하도록 하겠습니다.

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

코딩으로 확인하는 최소 제곱

- 이제 분자에 해당하는 부분을 구하겠습니다.

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

- 다음과 같이 새로운 함수를 정의하여 dividend 변수에 분자의 값을 저장합니다.

```
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)
```

코딩으로 확인하는 최소 제곱

- 임의의 변수 d의 초깃값을 0으로 설정한 뒤 x의 개수만큼 실행합니다
- d에 x의 각 원소와 평균의 차, y의 각 원소와 평균의 차를 곱해서 차례로 더하는 최소 제곱법을 그대 로 구현합니다.
- def는 함수를 만들 때 사용하는 예약어입니다.
- 여기서는 top이라는 함수를 새롭게 만들었고, 그 안에 최소 제곱법의 분자식을 그대로 가져와 구현하였습니다.

코딩으로 확인하는 최소 제곱

- 이제 위에서 구한 분모와 분자를 계산하여 기울기 a를 구하겠습니다.

```
a = dividend / divisor
```

- a를 구하고 나면 y 절편을 구하는 공식을 이용해 b를 구할 수 있습니다.

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

```
b = my – (mx*a)
```

코딩으로 확인하는 최소 제곱

- 이를 하나의 파일로 정리해 보면 다음과 같습니다.

```
import numpy as np

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

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

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

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

코딩으로 확인하는 최소 제곱

- 이를 하나의 파일로 정리해 보면 다음과 같습니다.

```
dividend = top(x, mx, y, my)

print("분모:", divisor)
print("분자:", dividend)

a = dividend / divisor
b = my - (mx * a)

print("기울기 a = ", a)
print("y 절편 b = ", b)
```

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

평균 제곱근 오차

- 앞서 우리는 최소 제곱법을 이용해 가장 훌륭한 직선을 그렸습니다.
- 공식에 따라 기울기와 절편을 편리하게 구했지만 이 공식만으로 앞으로 만나게 될 모든 상황을 해결하기는 어렵습니다.
- 여러 개의 입력을 처리하기에는 무리가 있기 때문입니다.
- 딥러닝은 대부분 입력 값이 여러 개인 상황에서 이 를 해결하기 위해 실행됩니다.
- 따라서 복잡한 연산 과정이 따를 수밖에 없습니다.

평균 제곱근 오차

- 이렇게 여러 개의 입력 값을 계산할 때는 임의의 선을 그리고 난 후, 이 선이 얼마나 잘 그려졌는지를 평가하여 조금씩 수정해 가는 방법을 사용합니다.
- 이를 위해 주어진 선의 오차를 평가하는 오차 평가 알고리즘이 필요합니다.
- 지금부터 오차를 평가하는 방법 중 가장 많이 사용되는 평균 제곱근 오차(root mean square error)에 대해 배워보겠습니다.

잘못 그은 선 바로잡기

- 우리는 앞서 기울기 a와 y 절편 b를 찾기 위해 최소 제곱법을 사용했습니다.
- 하지만 이는 변수가 '공부한 시간' 하나뿐일 때였습니다
- 2장에서 살펴본 폐암 수술 환자의 생존율 데이터를 보면, 입력 데이터의 종류가 17개나 됩니다.
- 모든 딥러닝 프로젝트는 이처럼 여러 개의 입력 변수를 다룹니다.
- 따라서 기울기 a와 절편 b를 찾아내는 다른 방법이 필요합니다.

잘못 그은 선 바로잡기

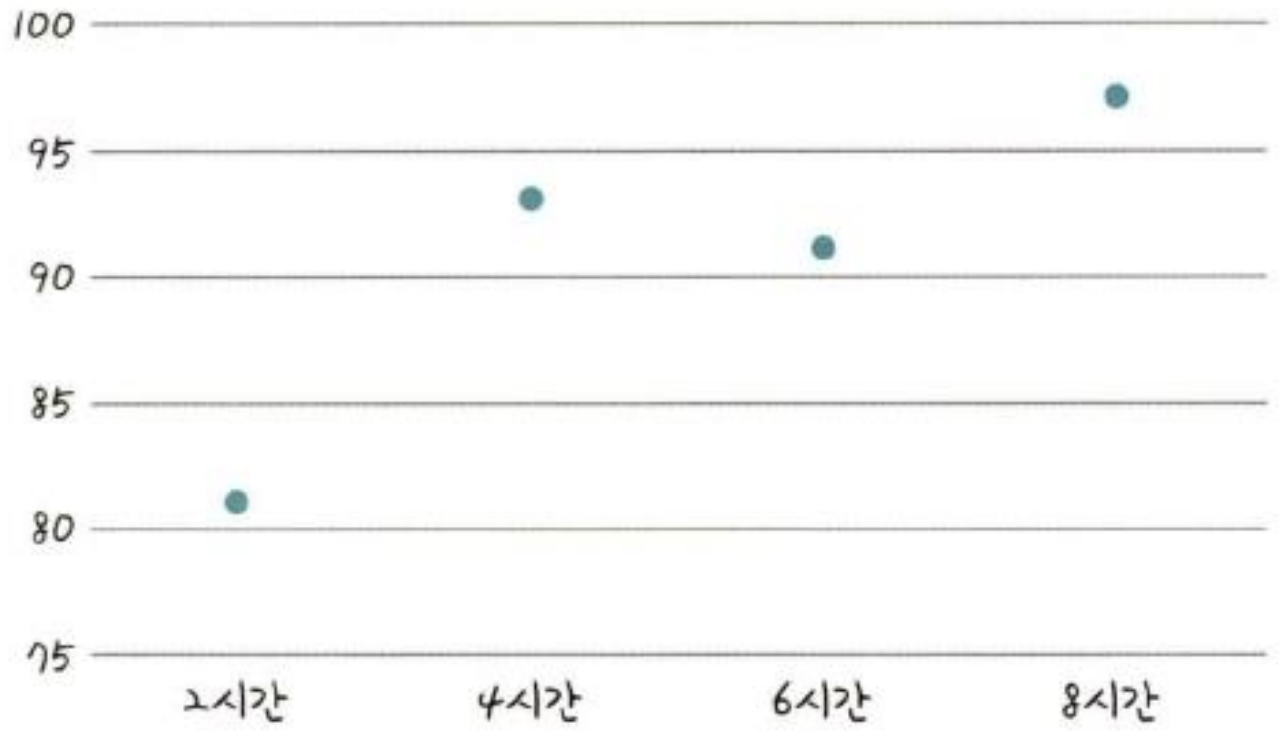
- 가장 많이 사용하는 방법은 '일단 그리고 조금씩 수정해 나가기' 방식입니다.
- 가설을 하나 세운 뒤 이 값이 주어진 요건을 충족하는지 판단하여 조금씩 변화를 주고, 이 변화가 긍정적이면 오차가 최소가 될 때까지 이 과정을 계속 반복하는 방법입니다.
- 이는 딥러닝을 가능하게 해주는 가장 중요한 원리 중 하나입니다.

잘못 그은 선 바로잡기

- 그런데 선을 긋고 나서 수정하는 과정에서 빠지면 안 되는 것이 있습니다.
- 나중에 그린 선이 먼저 그린 선보다 더 좋은지 나쁜지를 판단하는 방법입니다.
- 즉, 각 선의 오차를 계산할 수 있어야 하고, 오차가 작은 쪽으로 바꾸는 알고리즘이 필요한 것이지요.
- 그럼 지금부터 오차를 계산하는 방법을 알아보겠습니다.

잘못 그은 선 바로잡기

- 앞서 나온 공부한 시간과 성적의 관계도를 다시 한번 볼까요?

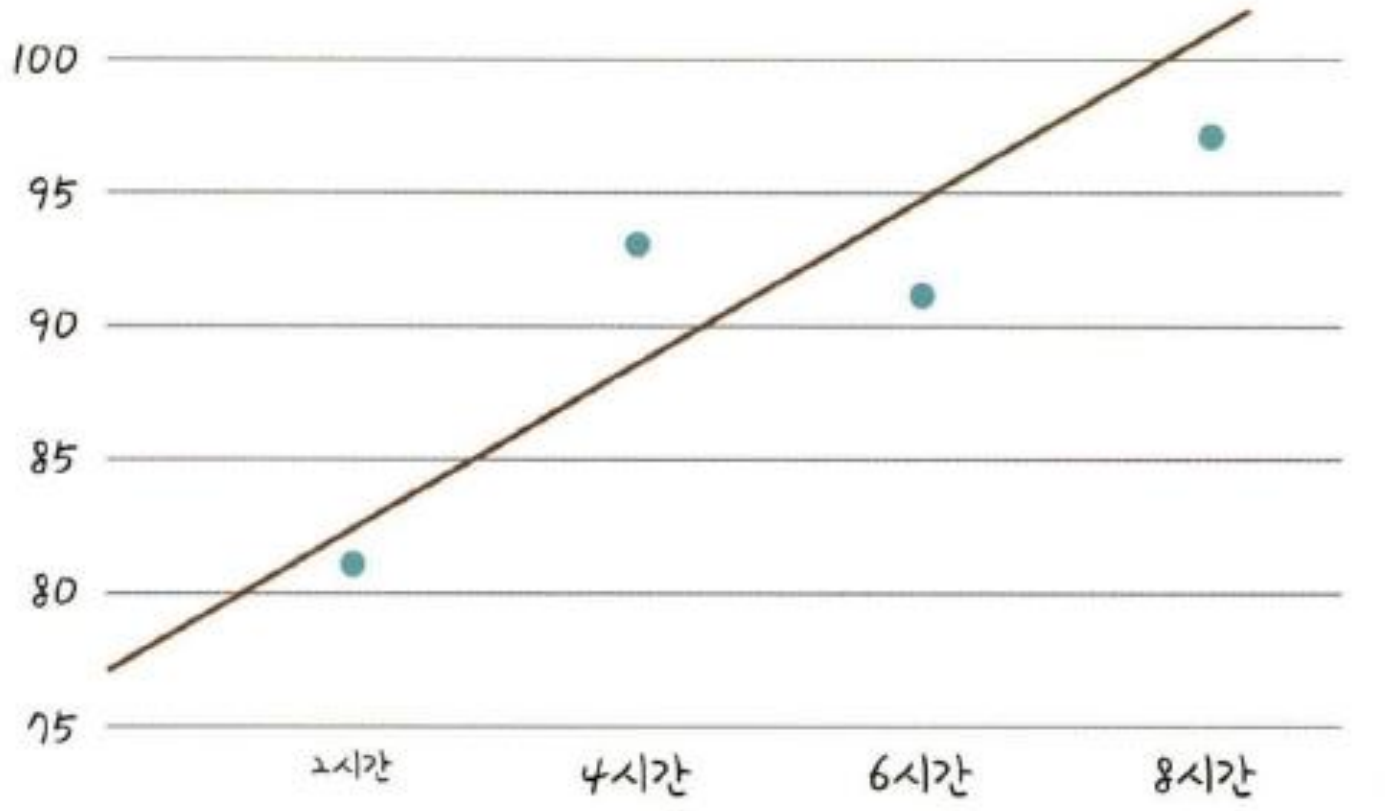


잘못 그은 선 바로잡기

- 우리는 앞에서 최소 제곱법을 통해 점들의 특성을 가장 잘 나타내는 최적의 직선이 $y = 2.3x + 79$ 임을 구했지만, 이번에는 최소 제곱법을 사용하지 않고 아무 값이나 a와 b에 대입해 보겠습니다.
- 임의의 값을 대입한 뒤 오차를 구하고 이 오차를 최소화하는 방식을 사용해서 최종 a와 최종 b의 값을 구해 보겠습니다.

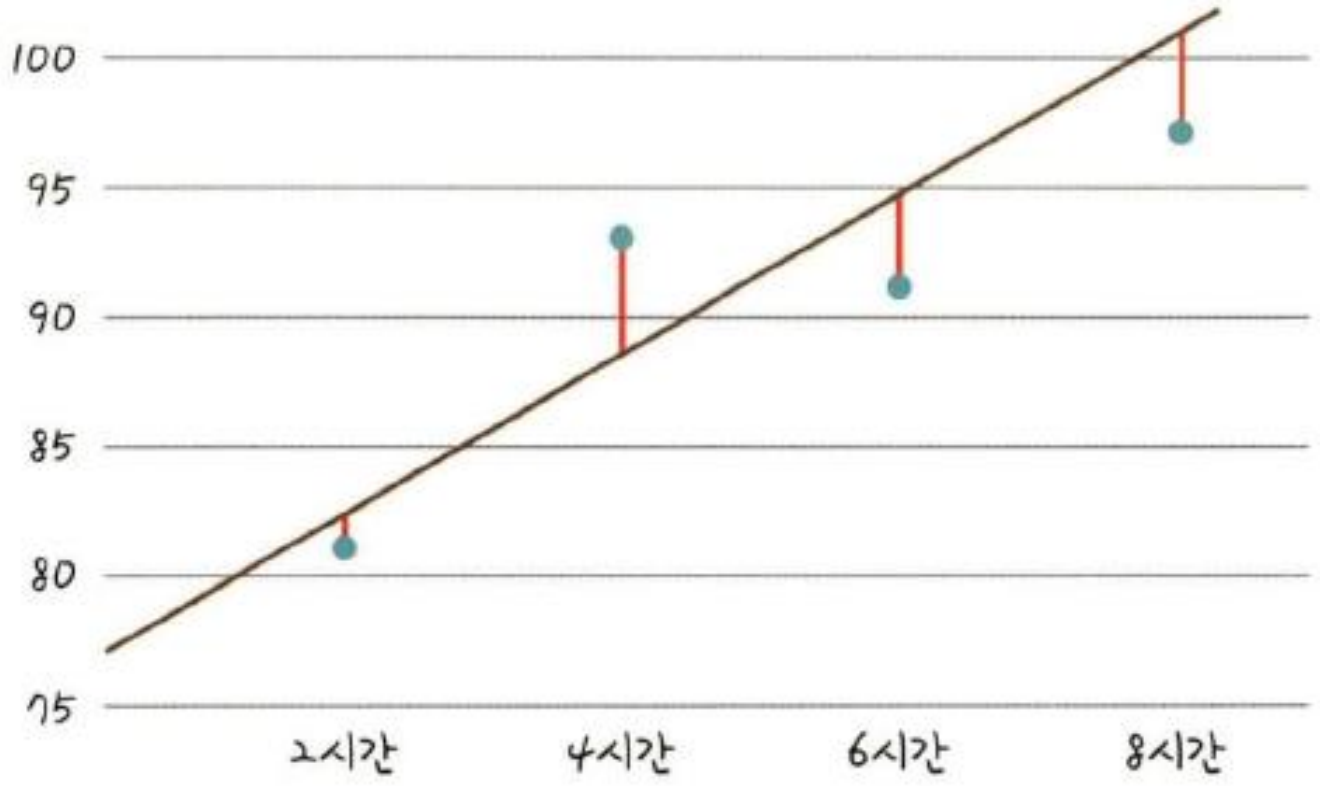
잘못 그은 선 바로잡기

- 먼저 대강 선을 그어보기 위해서 기울기 a와 절편 b를 임의의 수 3과 76이라고 가정해 보겠습니다.
- y = 3x + 76 인 선을 그려보면 그림과 같습니다.



잘못 그은 선 바로잡기

- 그림과 같이 그린 임의의 직선이 어느 정도의 오차가 있는지를 확인하려면 각 점과 그래프 사이의 거리를 재면 됩니다.

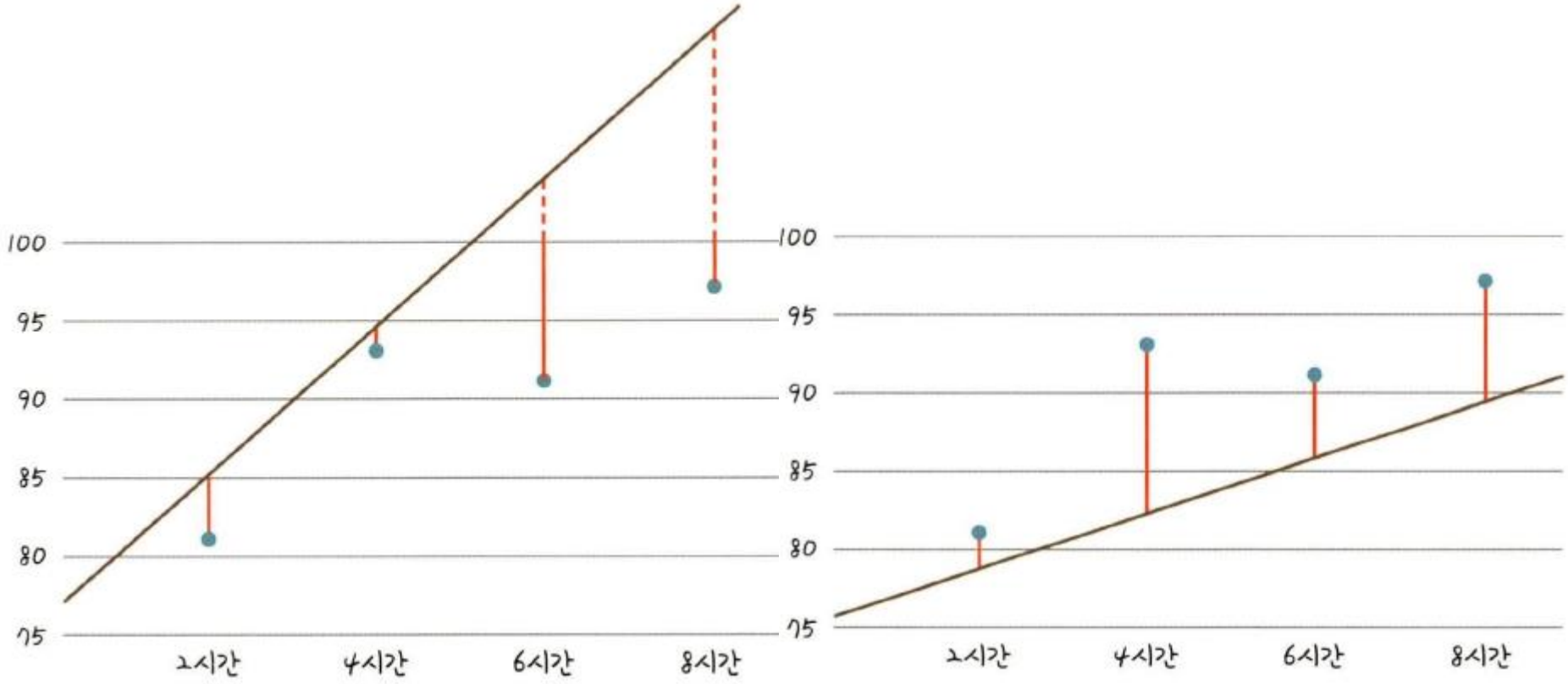


잘못 그은 선 바로잡기

- 그림에서 볼 수 있는 빨간색 선은 직선이 잘 그어졌는지를 나타냅니다.
- 이 직선들의 함이 작을수록 잘 그어진 직선이고 이 직선들의 합이 클수록 잘못 그어진 직선이 됩니다.

잘못 그은 선 바로잡기

- 예를 들어, 기울기 값을 각각 다르게 설정한 그래프를 볼까요?



잘못 그은 선 바로잡기

- 그래프의 기울기가 잘못되었을수록 빨간색 선의 거리의 합, 즉 오차의 합도 커집니다.
- 만약 기울기가 무한대로 커지면 오차도 무한대로 커지는 상관관계가 있는 것을 알 수 있습니다.
- 빨간색 선의 거리의 합을 실제로 계산해 보겠습니다.
- 거리는 입력 데이터에 나와 있는 y의 '실제 값'과 x를 y = 3x + 76의 식에 대입해서 나오는 '예측 값'과의 차이를 통해 구할 수 있습니다.

잘못 그은 선 바로잡기

- 예를 들어, 2시간 공부했을 때의 실제 나온 점수(81점)와 그래프 y = 3x + 76식에 x= 2를 대입했을 때 (82점)의 차이가 곧 오차입니다.
- 따라서 오차를 구하는 방정식은 다음과 같습니다.
  - 오차 = 실제 값 – 예측 값
- 이 식에 주어 진 데이터를 대입하여 얻을 수 있는 모든 오차의 값을 정리하면 표와 같습니다.

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

잘못 그은 선 바로잡기

- 이렇게 해서 구한 오차를 모두 더하면 1 + (-5) + 3 + 3 = 2 가 됩니다.
- 그런데 이 값은 오차가 실제로 얼마나 큰지를 가늠하기에는 적합하지 않습니다.
- 왜냐하면 오차에 양수와 음수가 섞여 있기 때문에 오차를 단순히 더해 버리면 합이 0 이 될 수도 있기 때문입니다.
- 부호를 없애야 정확한 오차를 구할 수 있습니다
- 따라서 오차의 합을 구할 때는 각 오차의 값을 제곱해 줍니다.

잘못 그은 선 바로잡기

- 이를 식으로 표현하면 다음과 같습니다.

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

- 여기서 i는 x가 나오는 순서를, n은 x 원소의 총 개수를 의미합니다.
- p_i는 x_i에 대응하는 '실제 값'이고 y_i는 x_i가 대입되었을 때 직선의 방정식(여기서는 y = 3x + 76) 이 만드는 '예측 값'입니다
- 이 식에 의해 오차의 합을 다시 계산하면 1 + 25 + 9 + 9 = 44 입니다.

잘못 그은 선 바로잡기

- 오차의 합에 이어 각 x 값의 평균 오차를 이용합니다.
- 위에서 구한 값을 n으로 나누면 오차 합의 평균을 구할 수 있습니다.
- 이를 평균 제곱 오차(Mean Squared Error, MSE) 라고 부릅니다.

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

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

잘못 그은 선 바로잡기

- 그런데 때로 평균 제곱 오차가 너무 커서 쓰기가 불편할 때가 있습니다.
- 각 오차를 제곱한 값을 사용하므로 대용량 데이터 를 이용할 때는 계산 속도가 느려질 수 있습니다.
- 이럴 때는 여기에 다시 제곱근을 씌워 줍니다
- 이를 평균 제곱근 오차(Root Mean Squared Error, RMSE)라고 합니다.

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

- 앞서 그은 직선의 평균 제곱근 오차는 $\sqrt{11} = 3.3116\cdots$ 이 됩니다.

잘못 그은 선 바로잡기

- 평균 제곱 오차 또는 평균 제곱근 오차는 오차를 계산해서 앞선 추론이 잘 되었는지 평가하는 대표적인 공식입니다.
- 이제 우리가 하는 작업은 '평균 제곱근 오차의 계산 결과가 가장 작은 선을 찾는 작업이라고 다시 설명할 수 있습니다.
- 선형 회귀란 임의의 직선을 그어 이에 대한 평균 제곱근 오차를 구하고, 이 값을 가장 작게 만들어 주는 a와 b 값을 찾아가는 작업입니다.

코딩으로 확인하는 평균 제곱근 오차

- 이제 앞서 알아본 평균 제곱근 오차를 파이썬으로 구현해 보겠습니다.
- 임의로 정한 기울기 a와 y 절편 b의 값이 각각 3과 76이라고 할 때 리스트 'ab'를 만들어 여기에 이 값을 저장합니다.

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

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

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

코딩으로 확인하는 평균 제곱근 오차

- 다음은 내부 함수를 만들 차례 입니다.
- predict() 라는 함수를 사용해 일차 방정식 y= ax + b를 구현합니다.

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

코딩으로 확인하는 평균 제곱근 오차

- 평균 제곱근 공식을 그대로 파이썬 함수로 옮기면 다음과 같습니다.

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

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(p_i - y_i)^2}$$

- np.sqrt() 은 제곱근을 **2는 제곱을 구하라는 뜻이 고, mean( )은 평균값을 구하라는 뜻입니다
- 예측값과 실제 값을 각각 rmse() 라는 함수의 p와 a 자리에 입력해서 평균 제곱근을 구합니다.

코딩으로 확인하는 평균 제곱근 오차

- 이제 rmse( ) 함수에 데이터를 대입하여 최종값을 구하는 함수 rmse_val( )을 만듭니다.

```
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 들어갑니다.
- 이 값과 y 값이 각각 예측 값과 실제 값으로 rmse () 함수 안에 들어가게 됩니다.

코딩으로 확인하는 평균 제곱근 오차

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

```
predict_result = []

for i in range(len(x)):
    predict_result.append(predict(x[i]))
    print("공부한 시간 = %.f, 실제 점수 = %.f, 예측 점수 = %.f" %(x[i], y[i], predict(x[i])))
```

코딩으로 확인하는 평균 제곱근 오차

- 이제 하나로 정리해 볼까요?

```
import numpy as np

ab = [3, 76]

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

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


def rmse(p, a):
  return np.sqrt(((p - 1)**2).mean())

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

코딩으로 확인하는 평균 제곱근 오차

- 이제 하나로 정리해 볼까요?

```
predict_result = []

for i in range(len(x)):
  predict_result.append(predict(x[i]))
  print("공부한 시간 = %.f, 실제 점수 = %.f, 예측 점수 = %.f" %(x[i], y[i], predict(x[i])))

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

```
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: 90.24965373894794
```

코딩으로 확인하는 평균 제곱근 오차

- 이를 통해 우리가 처음 가정한 a = 3, b = 76은 오차가 약 3.3166이라는 것을 알게 되었습니다.
- 이제 남은 것은 이 오차를 줄이면서 새로운 선을 긋는 것입니다.
- 이를 위해서는 a와 b의 값을 적절히 조절하면서 오차의 변화를 살펴보고, 그 오차가 최소화되는 a와 b의 값을 구해야 합니다.
- 다음 장에서는 오차를 줄이는 방법인 '경사 하강법'에 대해 알아보겠습니다.

Q&A