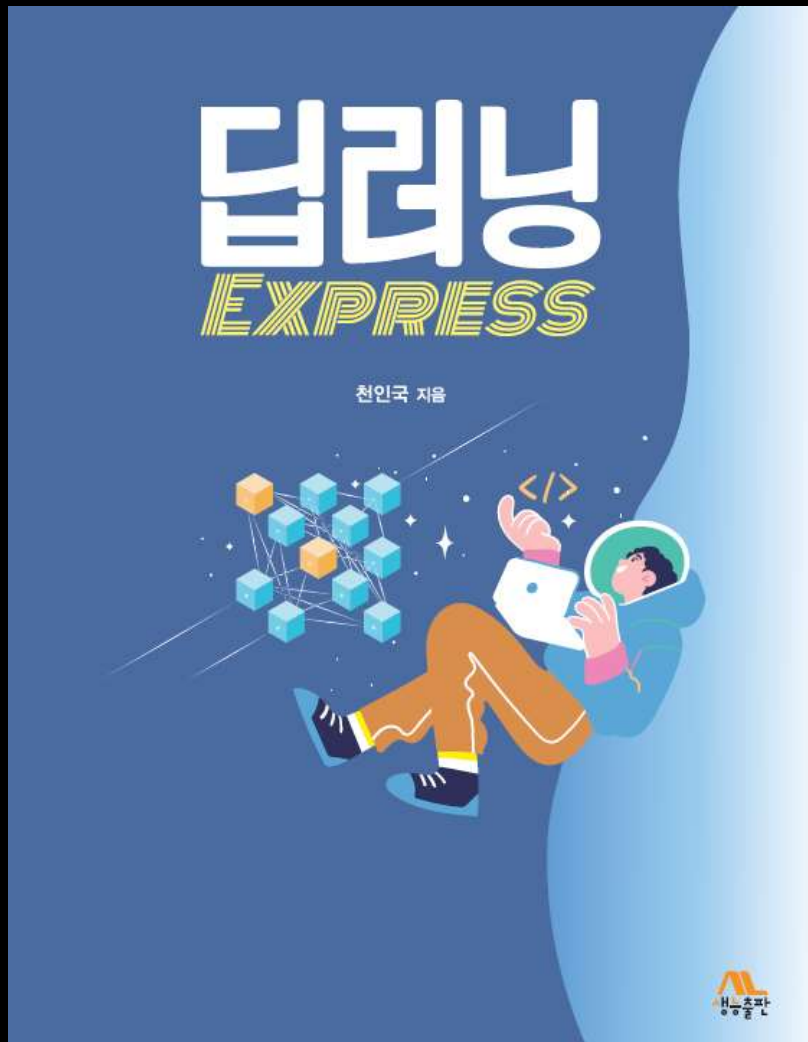


선형 회귀 분석

Linear Regression

교재/참고도서





학습 목표

- 회귀의 개념을 이해한다.
- 경사 하강법을 이해한다.
- 과잉 적합과 과소 적합을 이해한다.
- 파이썬과 **sklearn**을 이용하여 회귀를 구현해본다.





회귀(regression)와 분류(classification)

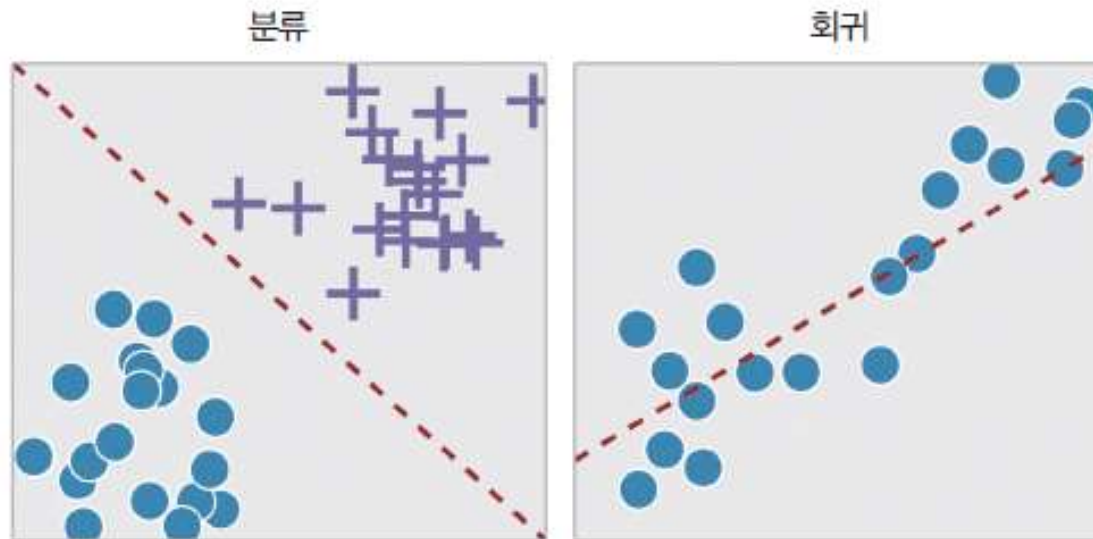


그림 4-1 회귀와 분류



선형 회귀

- 회귀란 일반적으로 데이터들을 2차원 공간에 찍은 후에 이들 데이터들을 가장 잘 설명하는 직선이나 곡선을 찾는 문제라고 할 수 있다.
- $y = f(x)$ 에서 출력 y 가 실수이고 입력 x 도 실수일 때 함수 $f(x)$ 를 예측하는 것이 회귀이다.

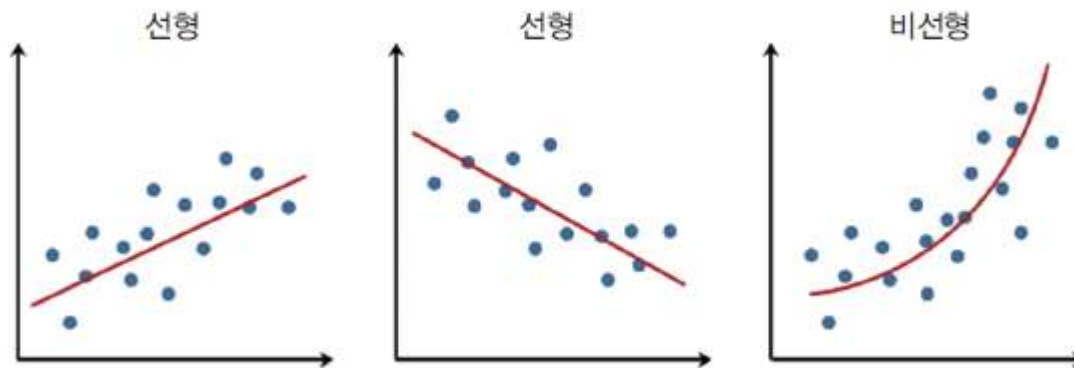


그림 4-2 회귀의 종류



선형 회귀의 예

- 부모의 키와 자녀의 키의 관계 조사
- 면적에 따른 주택의 가격
- 연령에 따른 실업율 예측
- 공부 시간과 학점 과의 관계
- CPU 속도와 프로그램 실행 시간 예측



선형 회귀 소개

- 직선의 방정식: $f(x) = mx+b$
- 선형 회귀는 입력 데이터를 가장 잘 설명하는 기울기와 절편값을 찾는 문제이다
- 선형 회귀의 기본식: $f(x) = Wx+b$
 - W: 기울기->가중치
 - B: 절편->바이어스



선형 회귀 예제



그림 4-3 선형 회귀의 예제



선형 회귀의 종류

- 단순 선형 회귀: 단순 선형 회귀는 독립 변수(x)가 하나인 선형 회귀이다.

$$f(x) = wx + b$$

- 다중 선형 회귀: 독립 변수가 여러 개인 경우

$$f(x, y, z) = w_0 + w_1x + w_2y + w_3z$$

$$\text{매출} = w_0 + w_1 \times \text{인터넷 광고} + w_2 \times \text{신문 광고} + w_3 \times \text{TV광고}$$



선형 회귀의 원리

x	y
1	2
2	5
3	6

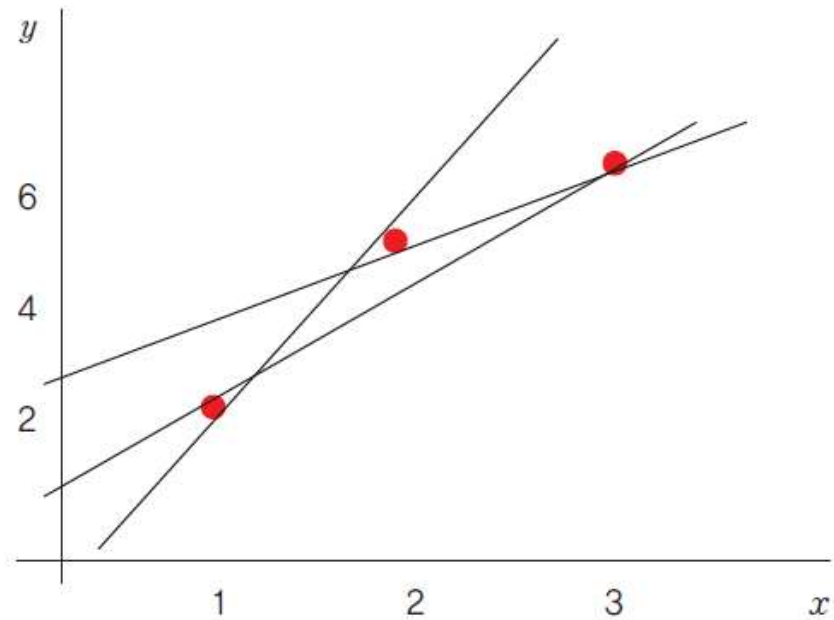


그림 4-4 데이터와 직선



직선과 데이터의 거리

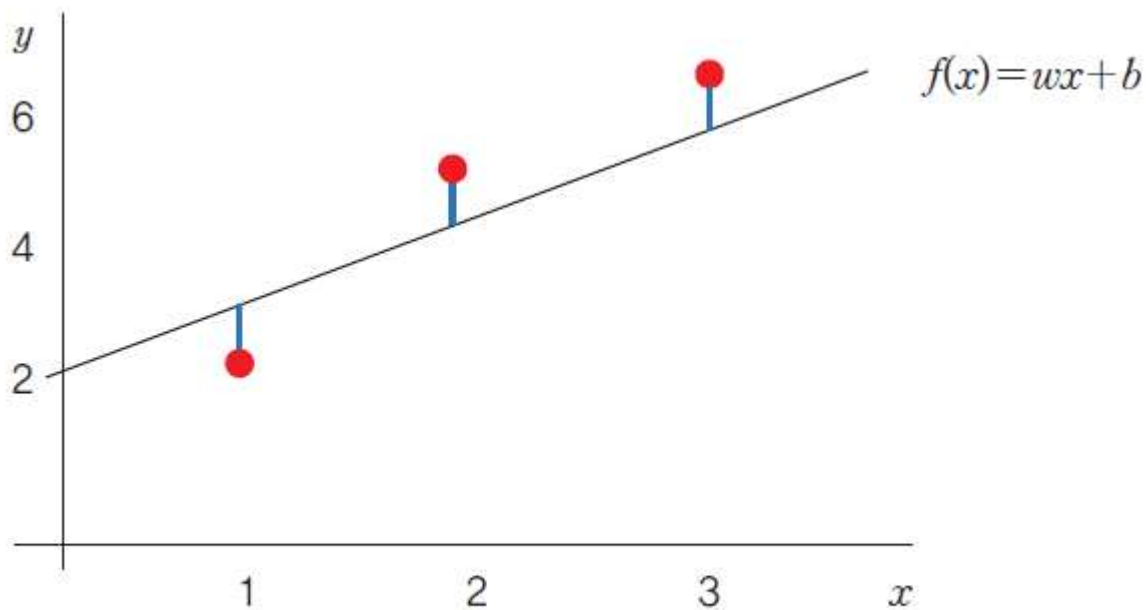


그림 4-5 데이터와 직선 간의 거리



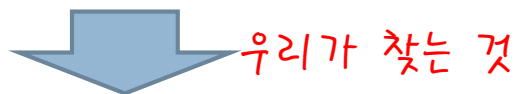
손실함수 (loss function)

- 직선과 데이터 사이의 간격을 제공하여 합한 값을 손실 함수(loss function) 또는 비용 함수(cost function)라고 한다.

$$Loss = \frac{1}{3} ((f(x_1) - y_1)^2 + (f(x_2) - y_2)^2 + (f(x_3) - y_3)^2)$$



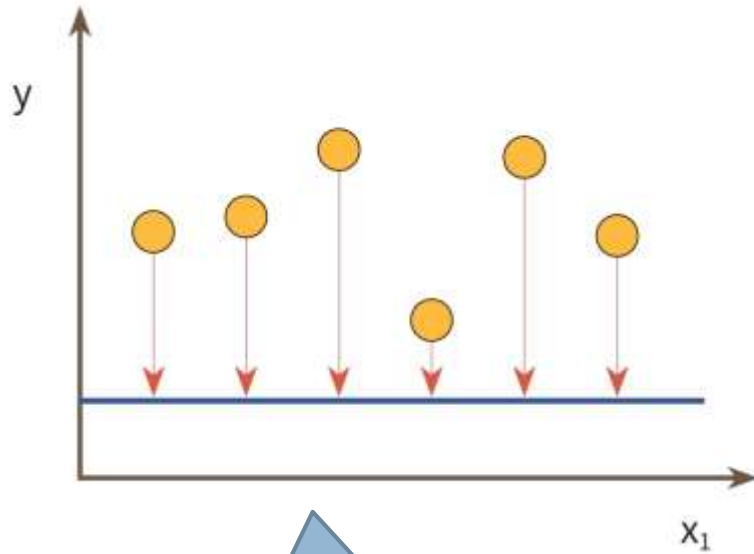
$$Loss(W, b) = \frac{1}{n} \sum_{i=1}^n (f(x_i) - y_i)^2 = \frac{1}{n} \sum_{i=1}^n ((Wx_i + b) - y_i)^2$$



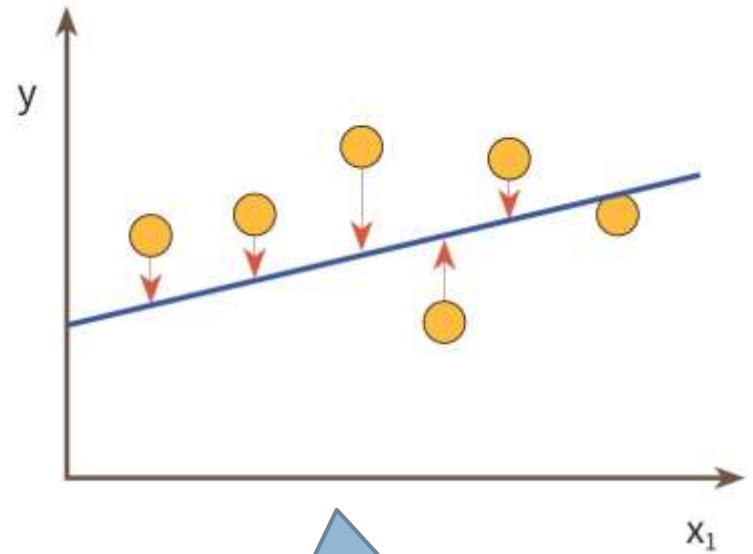
$$\underset{W, b}{\operatorname{argmin}} Loss(W, b)$$



손실 함수



손실이 큰 경우



손실이 작은 경우



선형 회귀에서 손실 함수 최소화 방법

- 분석적인 방법: 독립 변수와 종속 변수가 각각 하나인 선형 회귀
 - 최소제곱법 (<https://datalabbit.tistory.com/49>)

$$w = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2}, \quad b = \bar{y} - w\bar{x}$$

- 경사 하강법 (gradient descent method):
 - 경사 하강법은 손실 함수가 어떤 형태이러도, 또 매개 변수가 아무리 많아도 적용할 수 있는 일반적인 방법이다.
 - 점진적인 학습이 가능하다



경사하강법

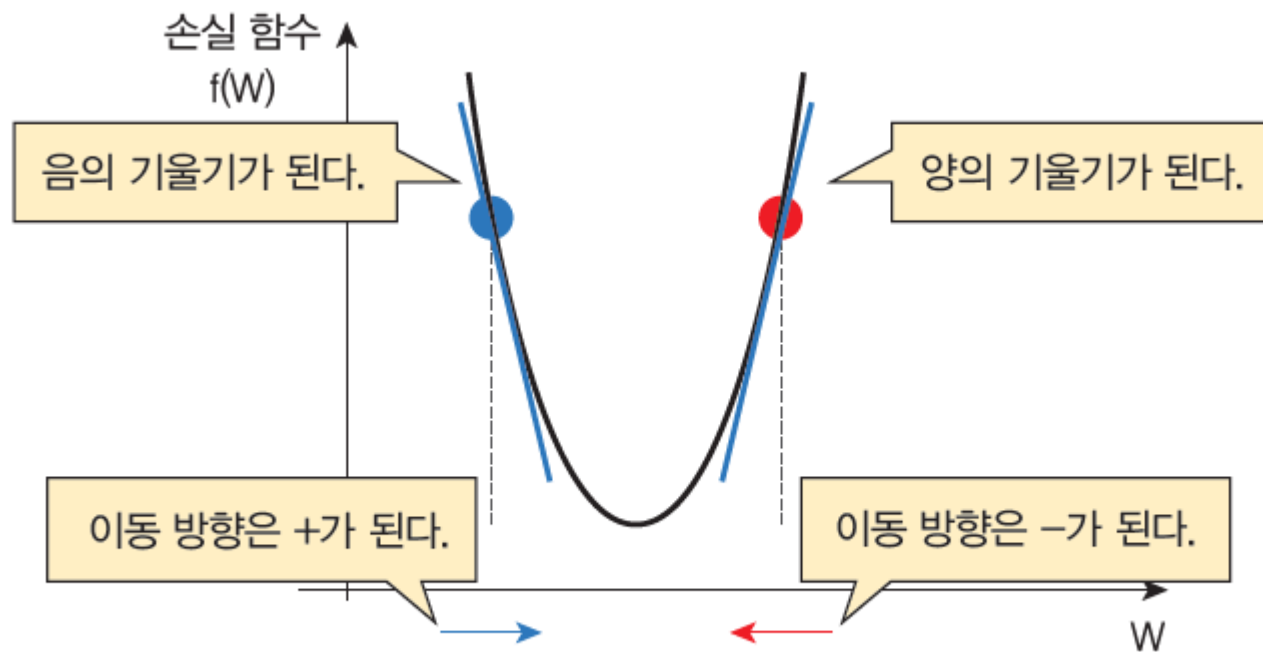


그림 4-7 경사 하강법의 이해



경사하강법



이것은 마치 산에서 내려오는 것과 유사합니다. 현재 위치에서 산의 기울기를 계산하여서 기울기의 반대 방향으로 이동하면 산에서 내려오게 됩니다.



학습률 (lr: learning rate)

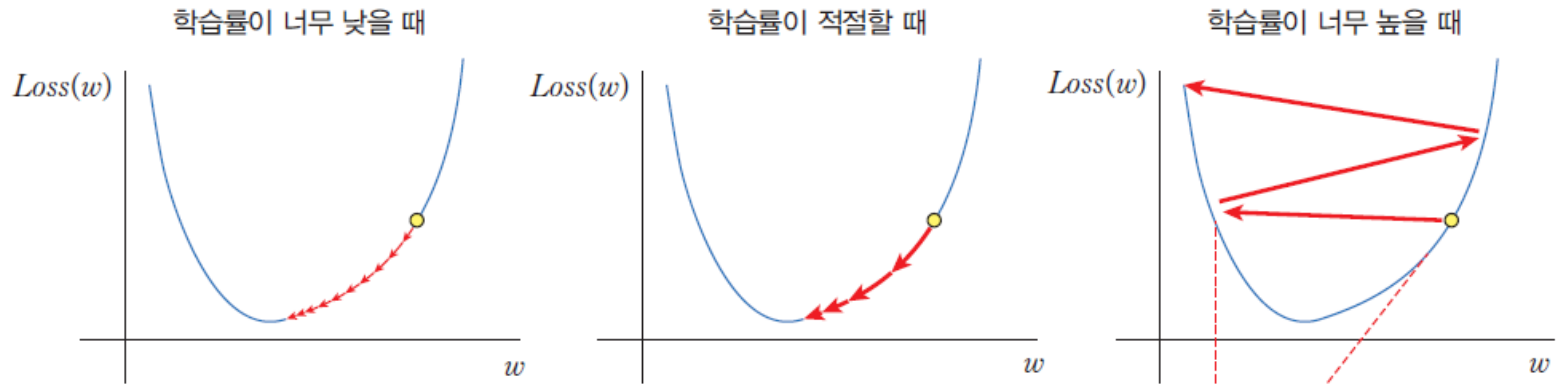


그림 4-9 학습률



지역최소값 문제

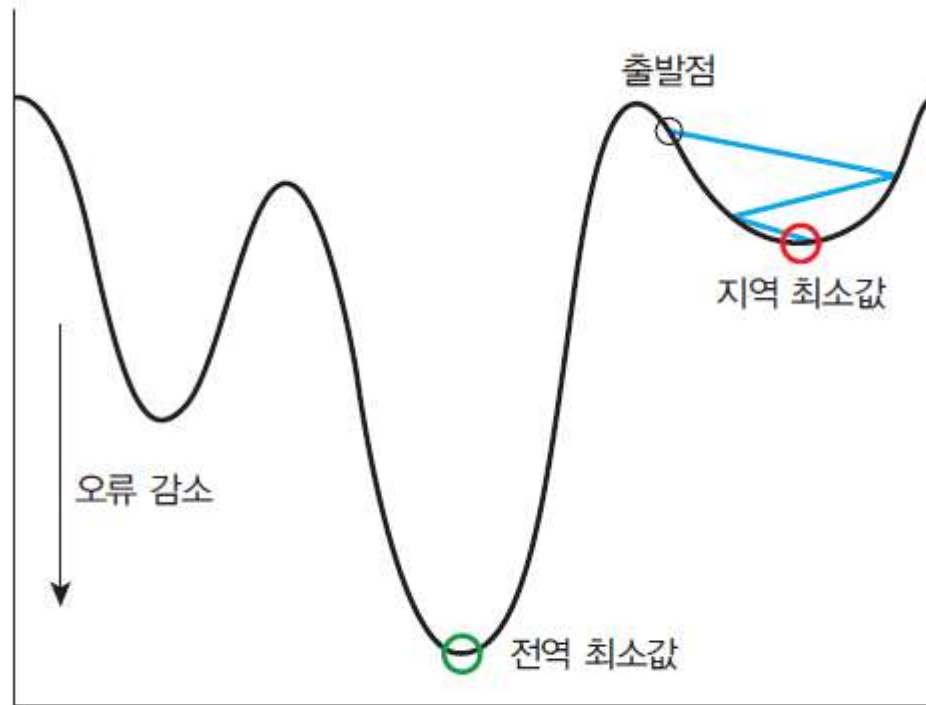


그림 4-10 지역 최소값과 전역 최소값



선형 회귀에서 경사하강법

$$Loss(W, b) = \frac{1}{n} \sum_{i=1}^n ((Wx_i + b) - y_i)^2$$

$$\frac{\partial Loss(W, b)}{\partial W} = \frac{1}{n} \sum_{i=1}^n 2((Wx_i + b) - y_i)(x_i) = \frac{2}{n} \sum_{i=1}^n x_i((Wx_i + b) - y_i)$$

$$\frac{\partial Loss(W, b)}{\partial b} = \frac{2}{n} \sum_{i=1}^n ((Wx_i + b) - y_i)$$

$$W = W - 0.01 * \frac{\partial Loss}{\partial W}$$

$$b = b - 0.01 * \frac{\partial Loss}{\partial b}$$



경사 하강법 구현

```
import numpy as np
import matplotlib.pyplot as plt

X = np.array([0.0, 1.0, 2.0])
y = np.array([3.0, 3.5, 5.5])

W = 0    # 기울기
b = 0    # 절편

lr = 0.01 # 학습률
epochs = 1000 # 반복 횟수

n = float(len(X)) # 입력 데이터의 개수

# 경사 하강법
for i in range(epochs):
    y_pred = W*X + b # 예측값
    dW = (2/n) * sum(X * (y_pred-y))
    db = (2/n) * sum(y_pred-y)
    W = W - lr * dW # 기울기 수정
    b = b - lr * db # 절편 수정
```



경사 하강법 구현

```
# 기울기와 절편을 출력한다.
```

```
print (W, b)
```

```
# 예측값을 만든다.
```

```
y_pred = W*X + b
```

```
# 입력 데이터를 그래프 상에 찍는다.
```

```
plt.scatter(X, y)
```

```
# 예측값은 선그래프로 그린다.
```

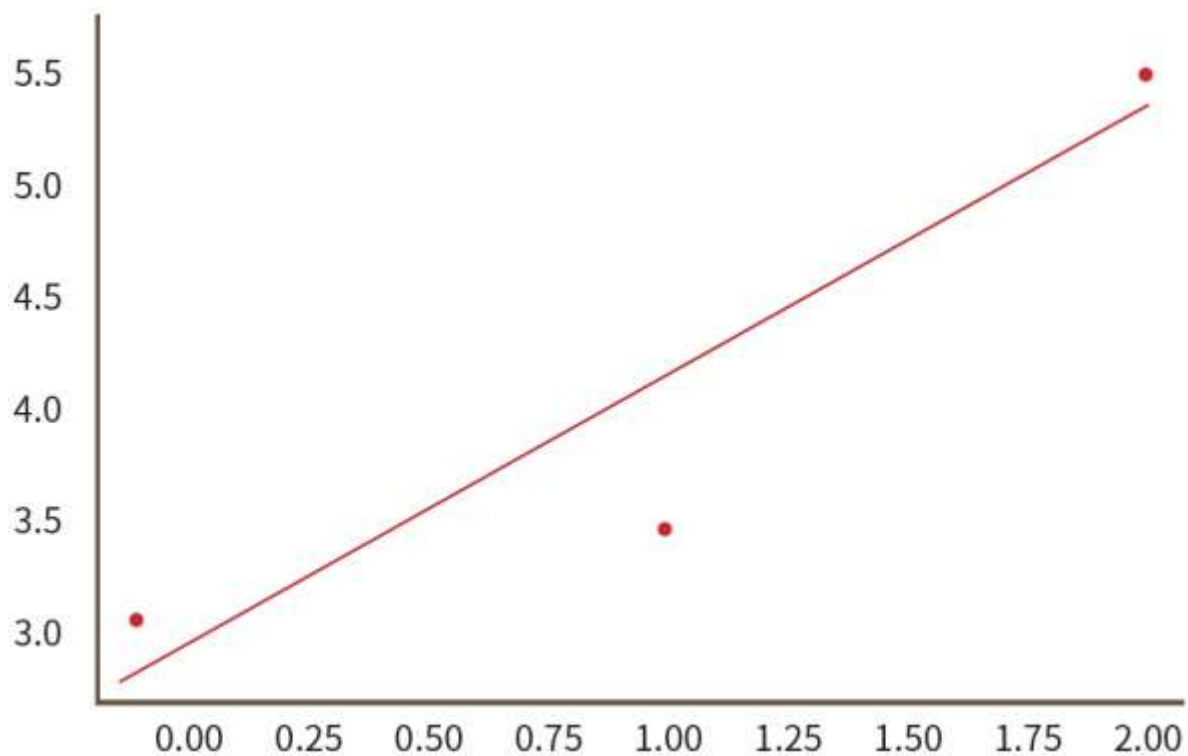
```
plt.plot([min(X), max(X)], [min(y_pred), max(y_pred)], color='red')
```

```
plt.show()
```

1.2532418085611319 2.745502230882486



경사 하강법 구현





Lab: 학습 실험

- 구글의 텐서 플로우 플레이그라운드(https://playground.tensorflow.org)는 이주 유용한 사이트이다.

The screenshot shows the TensorFlow Playground interface with several red boxes and callouts explaining key features:

- 데이터 세트를 선택한다.** (Select the dataset.) - Points to the 'DATA' section where a dataset is chosen.
- 학습률을 선택한다.** (Select the learning rate.) - Points to the 'Learning rate' dropdown menu.
- Linear를 선택한다.** (Select Linear.) - Points to the 'Activation' dropdown menu.
- Regression을 선택한다.** (Select Regression.) - Points to the 'Problem type' dropdown menu.
- 2 HIDDEN LAYERS** - Points to the hidden layer configuration section.
- 1 neuron** - Points to the first hidden layer neuron.
- 1 neuron** - Points to the second hidden layer neuron.
- 버튼을 눌러서 뉴런 하나만 남긴다.** (Press the button to leave only one neuron.) - Points to the minus buttons in the hidden layer configuration.
- 버튼을 눌러서 뉴런 하나만 남긴다.** (Press the button to leave only one neuron.) - Points to the minus buttons in the hidden layer configuration.

The interface also displays the following information:

- Epoch:** 000,000
- Learning rate:** 0.03
- Activation:** Linear
- Regularization:** None
- Regularization rate:** 0
- Problem type:** Regression
- DATA:** Which dataset do you want to use? (Ratio of training to test data: 50%, Noise: 0, Batch size: 10)
- FEATURES:** Which properties do you want to feed in? (X₁, X₂, X₁², X₂², X₁X₂, sin(X₁))
- OUTPUT:** Test loss 0.128, Training loss 0.130



Lab: 하스틱 시스

A Neural Network Playground

재생 버튼을 누른다.

Epoch: 000,081
Learning rate: 0.03
Activation: Linear
Regularization: None
Regularization rate: 0

DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE

FEATURES
Which properties do you want to feed in?
 X_1
 X_2
 X_1^2
 X_2^2
 $X_1 X_2$
 $\sin(X_1)$

2 HIDDEN LAYERS
1 neuron
1 neuron
This is the output from one neuron. Hover to see it larger.
The outputs are mixed with varying weights, shown by the thickness of the lines.

OUTPUT
Test loss 0.000
Training loss 0.000

오차가 줄어드는 것을 볼 수 있다.



선형 회귀 예제

- 이번 절에서는 아나콘다에 포함되어 있는 **Scikit-Learn** 라이브러리를 사용하여 회귀 함수를 구현하는 방법을 살펴본다.

```
import matplotlib.pyplot as plt
from sklearn import linear_model
```

```
# 선형 회귀 모델을 생성한다.
reg = linear_model.LinearRegression()
```

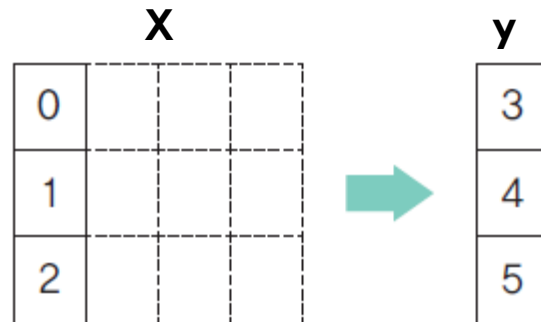
```
# 데이터는 파이썬의 리스트로 만들어도 되고 아니면 넘파이의 배열로 만들어도 됨
X = [[0], [1], [2]]          # 2차원으로 만들어야 함
y = [3, 3.5, 5.5]            #  $y = x + 3$ 
```

```
# 학습을 시킨다.
reg.fit(X, y)
```



학습 데이터 만들기

- 학습 데이터는 반드시 2차원 배열이어야 한다(한 열만 있어도 반드시 2차원 배열 형태로 만들어야 한다). 따라서 리스트의 리스트를 만들어서 다음과 같은 2차원 배열을 생성한다.



$X = [[0], [1], [2]]$

$y = [3, 3.5, 5.5]$



선형 회귀 예제

```
>>> reg.coef_  
array([1.25])
```

직선의 기울기

```
>>> reg.intercept_  
2.7500000000000004
```

직선의 y-절편

```
>>> reg.score(X, y)  
0.8928571428571429
```

R2 score

```
>>> reg.predict([[5]])  
array([8.])
```

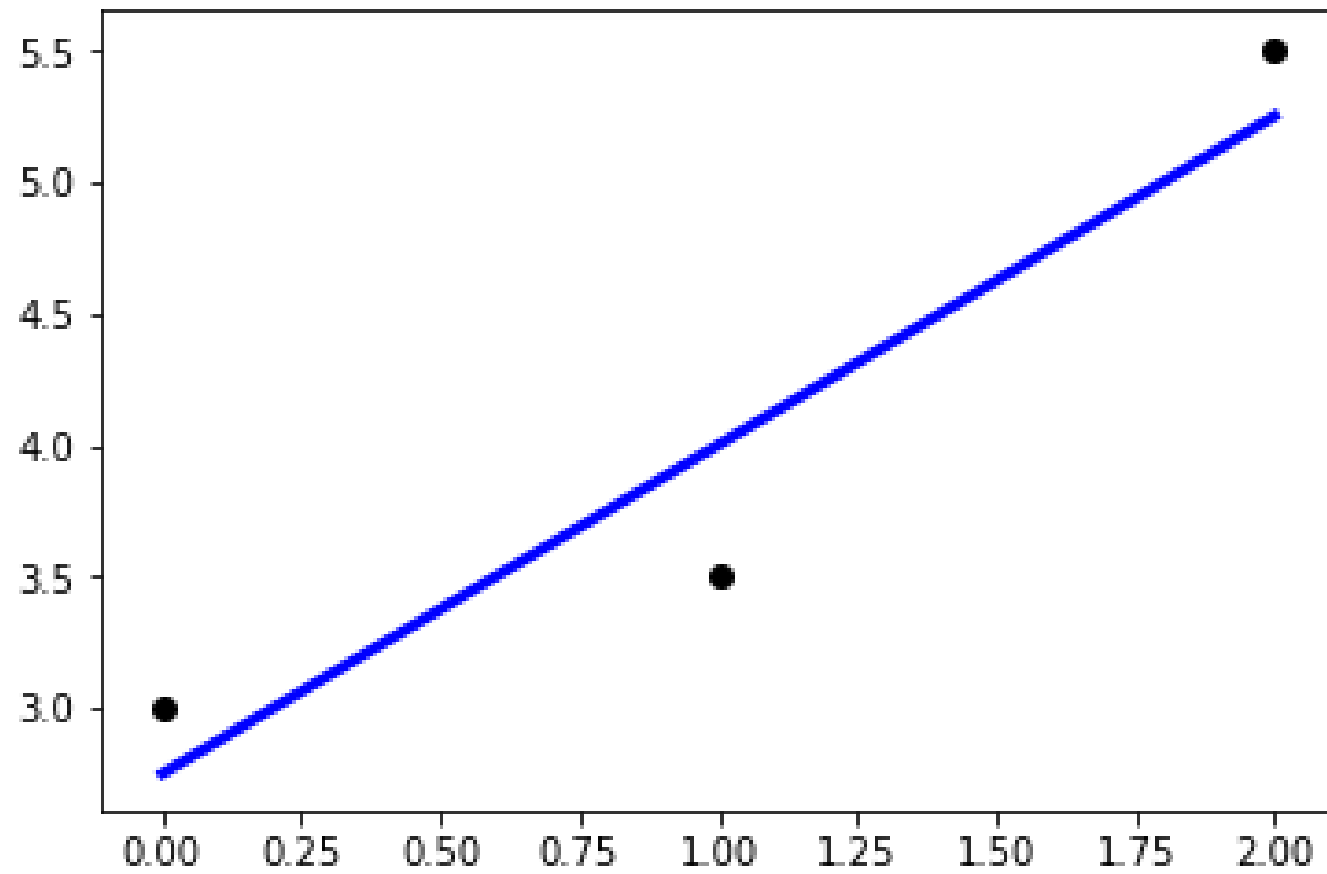


그래프를 그려보자.

```
# 학습 데이터와 y 값을 산포도로 그린다.  
plt.scatter(X, y, color='black')  
  
# 학습 데이터를 입력으로 하여 예측값을 계산한다.  
y_pred = reg.predict(X)  
  
# 학습 데이터와 예측값으로 선그래프로 그린다.  
# 계산된 기울기와 y 절편을 가지는 직선이 그려진다.  
plt.plot(X, y_pred, color='blue', linewidth=3)  
plt.show()
```



실험 결과





Lab: 선형 회귀 실습

- 인간의 키와 몸무게는 어느 정도 비례할 것으로 예상된다. 아래와 같은 데이터가 있을 때, 선형 회귀를 이용하여 학습시키고 키가 165cm 일 때의 예측값을 얻어보자. 파이썬으로 구현하여 본다.





Sol.

```
import matplotlib.pyplot as plt
from sklearn import linear_model

reg = linear_model.LinearRegression()

X = [[174], [152], [138], [128], [186]]
y = [71, 55, 46, 38, 88]

reg.fit(X, y)                                # 학습

print(reg.predict([[165]]))

# 학습 데이터와 y 값을 산포도로 그린다.
plt.scatter(X, y, color='black')

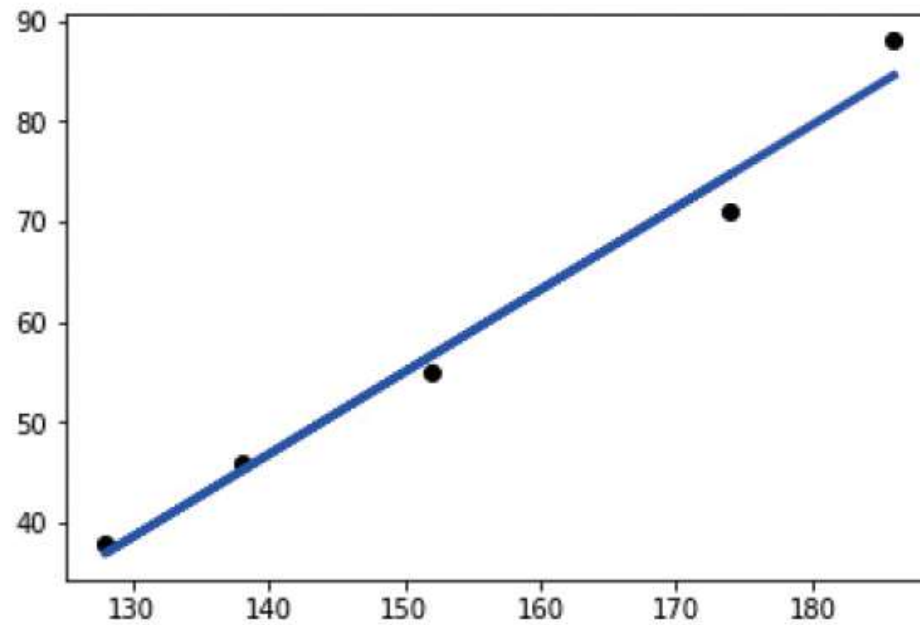
# 학습 데이터를 입력으로 하여 예측값을 계산한다.
y_pred = reg.predict(X)

# 학습 데이터와 예측값으로 선그래프로 그린다.
# 계산된 기울기와 y 절편을 가지는 직선이 그려진다.
plt.plot(X, y_pred, color='blue', linewidth=3)
plt.show()
```



Sol.

[67.30998637]





과잉 적합 vs 과소 적합

- 과잉 적합(**overfitting**)이란 학습하는 데이터에서는 성능이 뛰어나지만 새로운 데이터(일반화)에 대해서는 성능이 잘 나오지 않는 모델을 생성하는 것이다.
- 과소 적합(**underfitting**)이란 학습 데이터에서도 성능이 좋지 않은 경우이다. 이 경우에는 모델 자체가 적합하지 않은 경우가 많다. 더 나은 모델을 찾아야 한다.



과잉 적합 vs 과소 적합

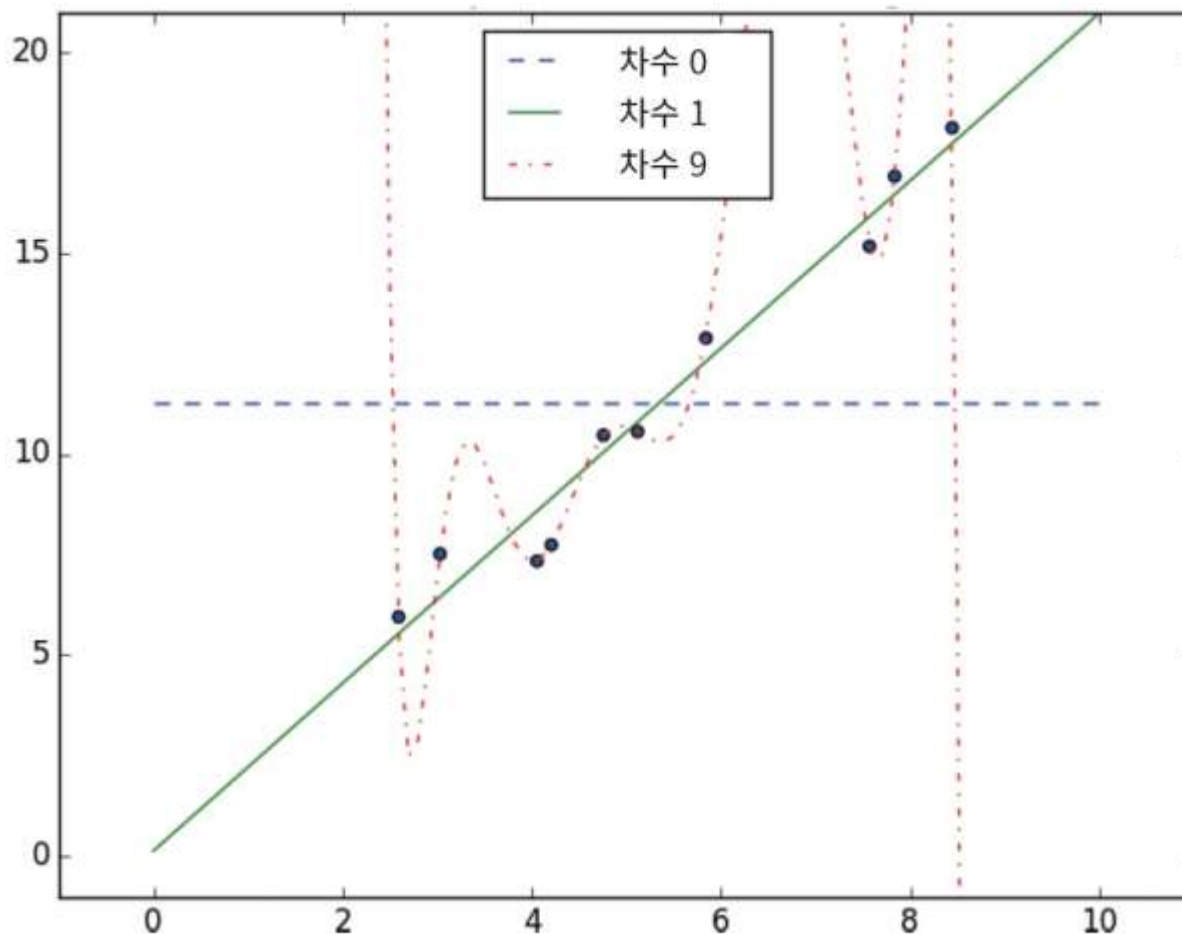


그림 10-5 회귀에서 과잉 적합의 예



Lab: 당뇨병 예제

- sklearn 라이브러리에는 당뇨병 환자들의 데이터가 기본적으로 포함되어 있다.

특징(10개)										데이터 개수 (442)	<table><tr><th>혈당</th></tr><tr><td></td></tr><tr><td></td></tr><tr><td>...</td></tr><tr><td></td></tr></table>	혈당			...	
혈당																
...																
age	sex	bmi	bp	s1	s2	s3	s4	s5	s6							
...																

sklearn.datasets.load_diabetes

```
sklearn.datasets.load_diabetes(*, return_X_y=False, as_frame=False, scaled=True)
```

Load and return the diabetes dataset (regression).

Samples total	442
Dimensionality	10
Features	real, $-0.2 < x < 0.2$
Targets	integer 25 - 346

[illegible]

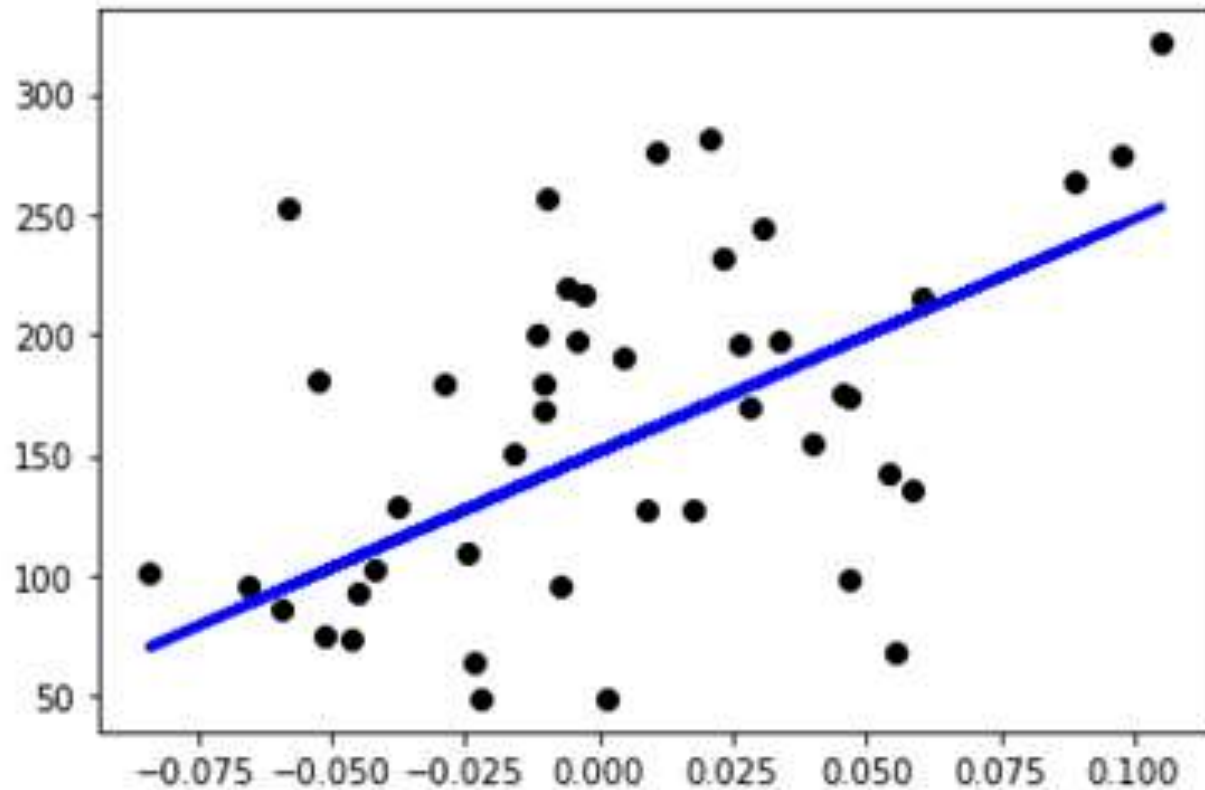


선형 회귀 예제

```
regr = linear_model.LinearRegression()  
regr.fit(X_train, y_train)  
  
# 테스트 데이터로 예측해보자.  
y_pred = model.predict(X_test)  
  
# 실제 데이터와 예측 데이터를 비교해보자.  
plt.plot(y_test, y_pred, '.')  
  
plt.scatter(X_test, y_test, color='black')  
plt.plot(X_test, y_pred, color='blue', linewidth=3)  
plt.show()
```

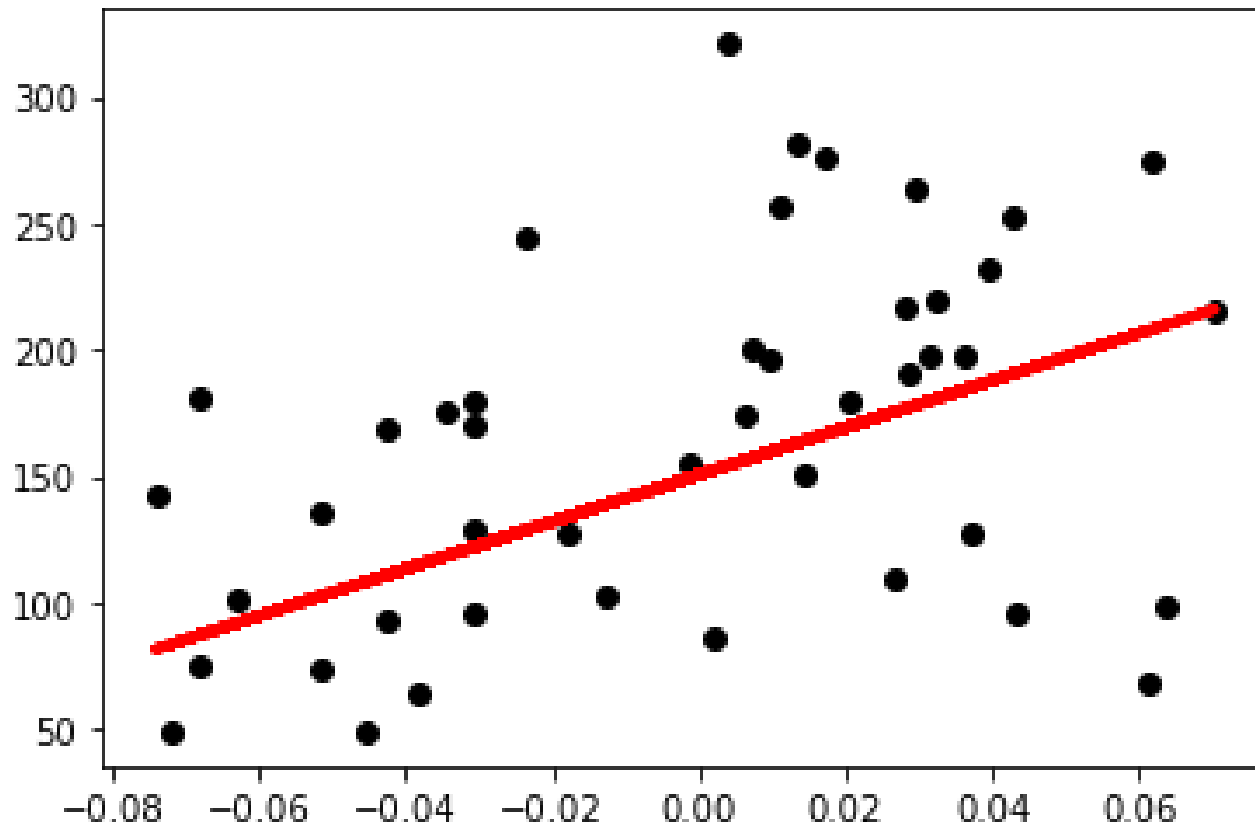


실행 결과: *bmi* vs. *target* ($r^2 = 0.334$)



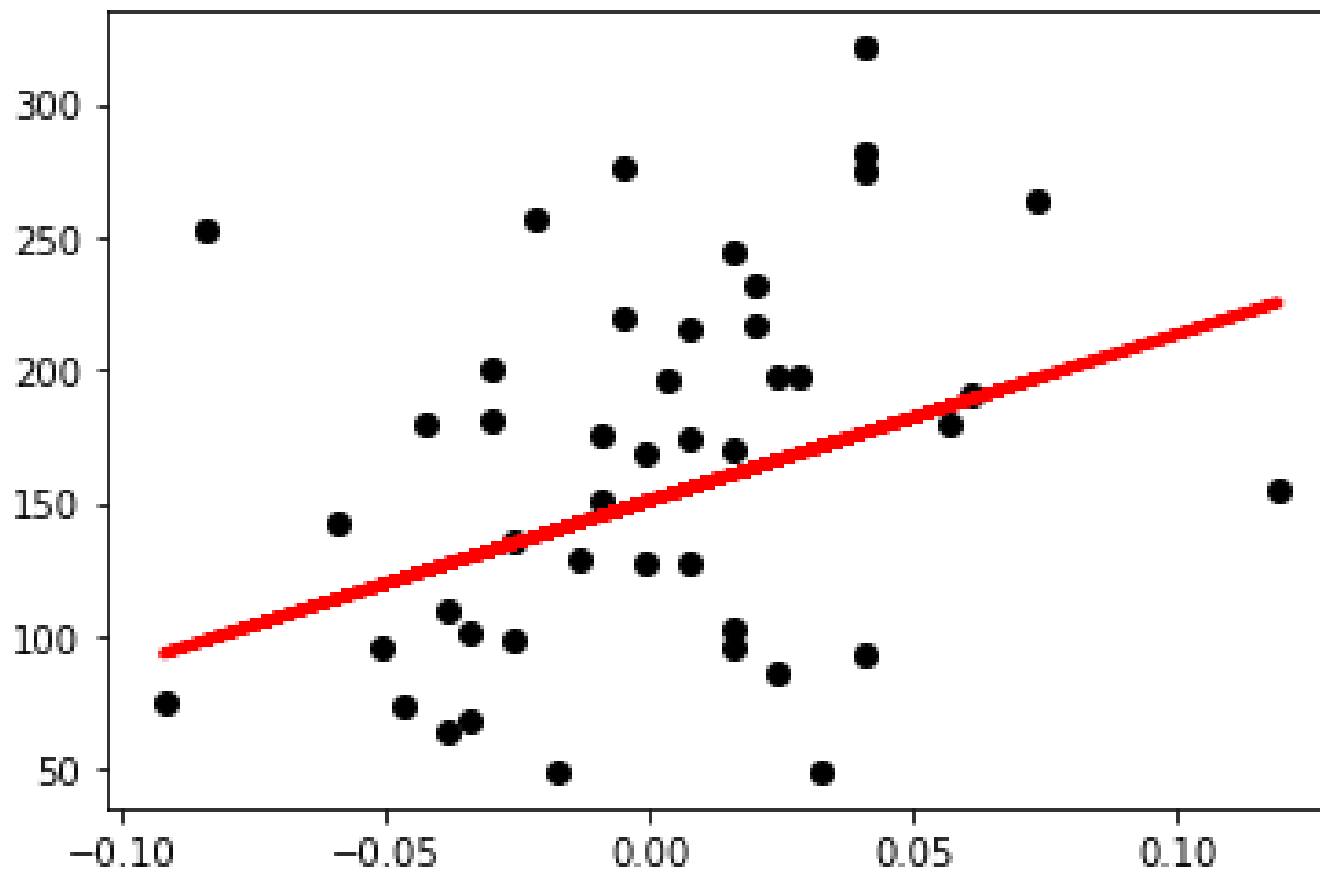


실험 결과: $s5(ltg)$ vs. $target$ ($r^2 = 0.335$)



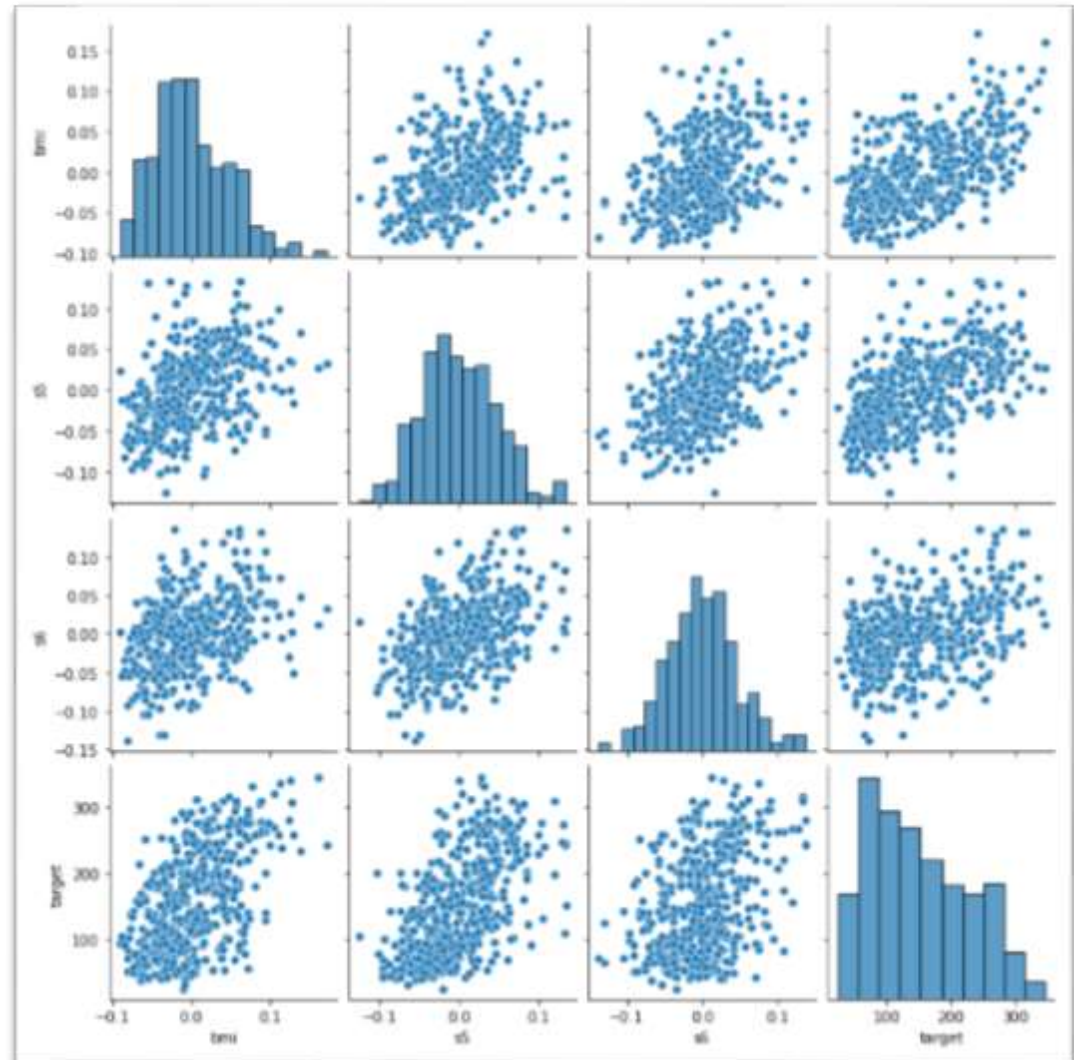
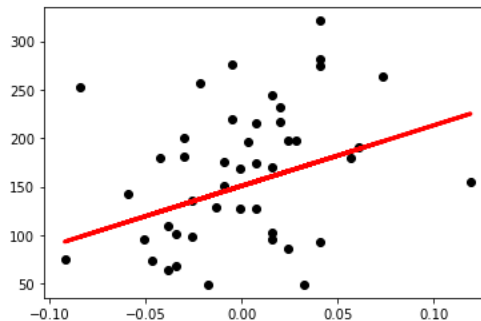
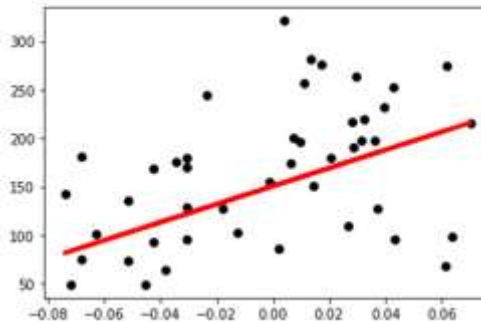
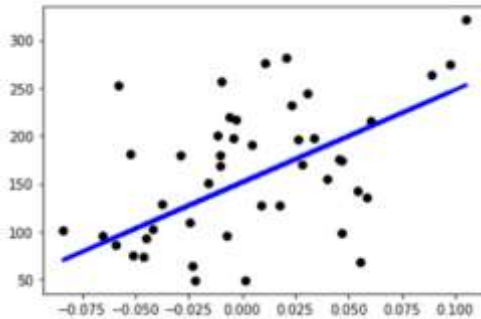


실험 결과: $s6(glu)$ vs. $target$ ($r^2 = 0.299$)





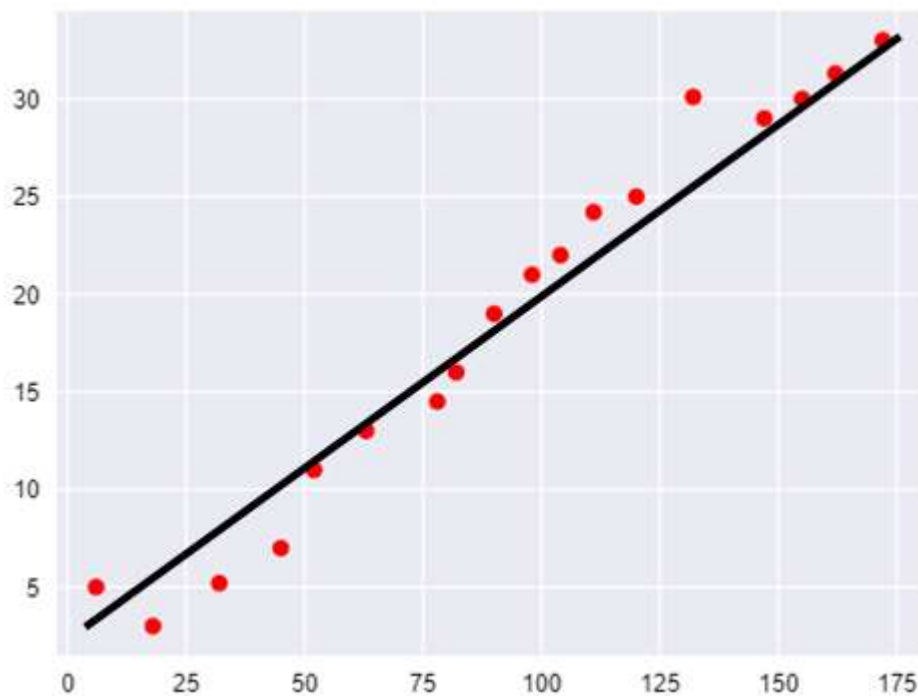
실행 결과: *bmi, s5, s6* vs. *target*





Mini Project: 면적에 따른 집값 예측

- 사용자가 아파트 면적을 입력하면 아파트의 가격이 출력되는 시스템을 만들어보자. 아파트 면적과 가격은 모두 실수이므로 기계 학습의 방법 중에서 선형 회귀를 사용하여야 한다





Summary

- 지도 학습에는 회귀(regression)와 분류(classification)가 있었다. 전자는 연속적인 값을 예측하고 후자는 입력을 어떤 카테고리 중의 하나로 예측한다.
- 선형 회귀는 입력 데이터를 가장 잘 설명하는 직선의 기울기와 절편값을 찾는 문제이다.
- 손실 함수(loss function)란 실제 데이터와 직선 간의 차이를 제공한 값이다.
- 회귀란 손실 함수를 최소로 하는 직선의 기울기와 절편값을 계산하는 것이다.
- 손실 함수의 값이 작아지는 방향을 알려면 일반적으로 경사 하강법(gradient descent method)과 같은 방법을 많이 사용한다.



Q & A

