선형 회귀
Linear Regression

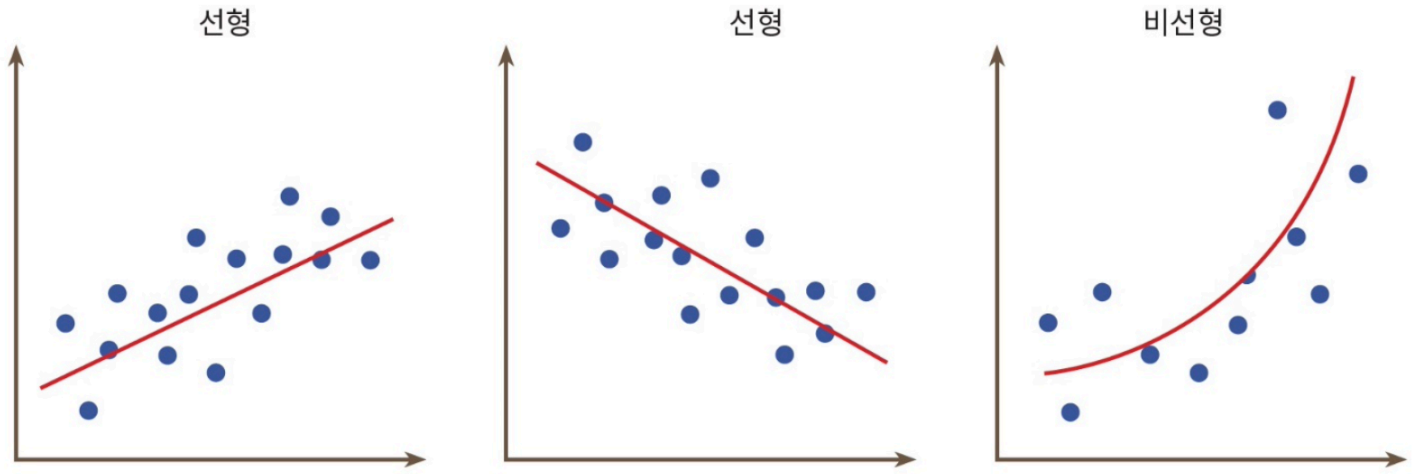
내용

- 회귀의 개념
- 손실 함수 (loss function)
- 과잉 적합 (overfitting)
- 과소 적합 (underfitting)
- 구현

선형 회귀

- 회귀란 데이터들의 추이를 가장 잘 설명하는 선형식 또는 비선형식을 찾는 문제
- y = f(x)에서 출력 y가 실수이고 입력 x도 실수일 때 함수 f(x)를 예측

$$y = f(x),\ x \in \mathrm{R} \text{ and } y \in \mathrm{R}$$



$f(x) = ax + b, a \neq 0$

$f(x) = ax^2 + bx + c, a \neq 0$

선형 회귀의 예

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

선형 회귀 소개

- 직선의 방정식: f(x) = wx+b, w는 0이 아님
- 선형 회귀는 입력 데이터를 가장 잘 설명하는 기울기와 절편값을 찾는 문제
- 선형 회귀의 기본식: f(x) = wx+b
  - 기울기, 가중치 (weight)
  - 절편, 바이어스 (bias)

선형 회귀 예제

키(단위: cm)	몸무게(단위: kg)
174	71
152	55
138	46
128	38
186	88

학습 데이터

학습

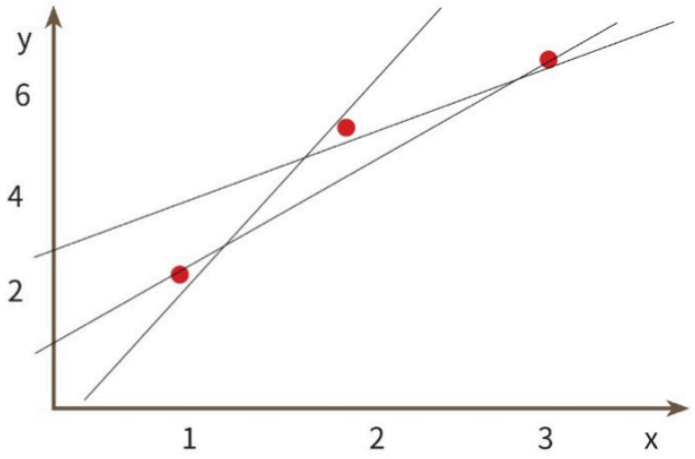
키(x)=165이면
몸무게는?

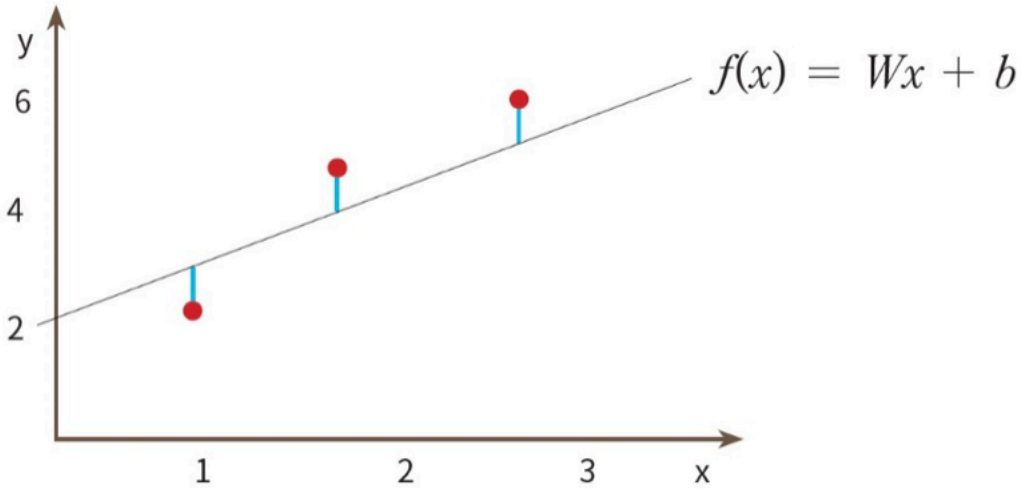
선형 회귀 모델

몸무게(y)=63으로
예측됩니다.

선형 회귀의 원리

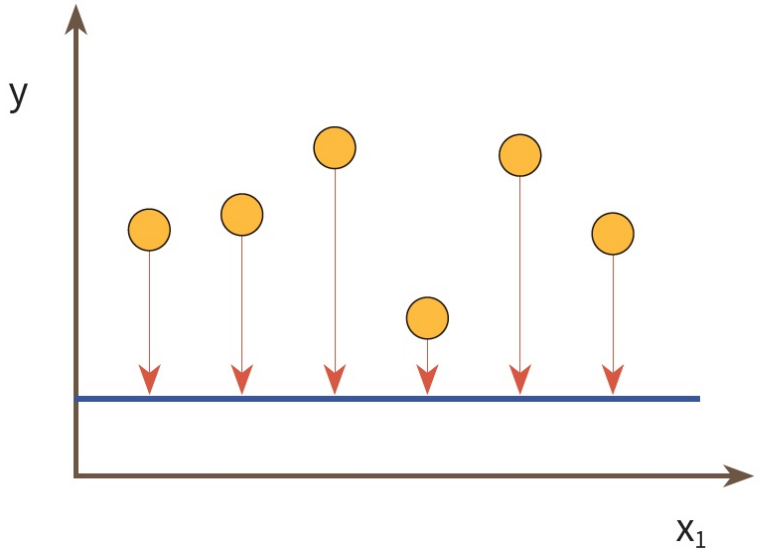
x	y
1	2
2	5
3	6

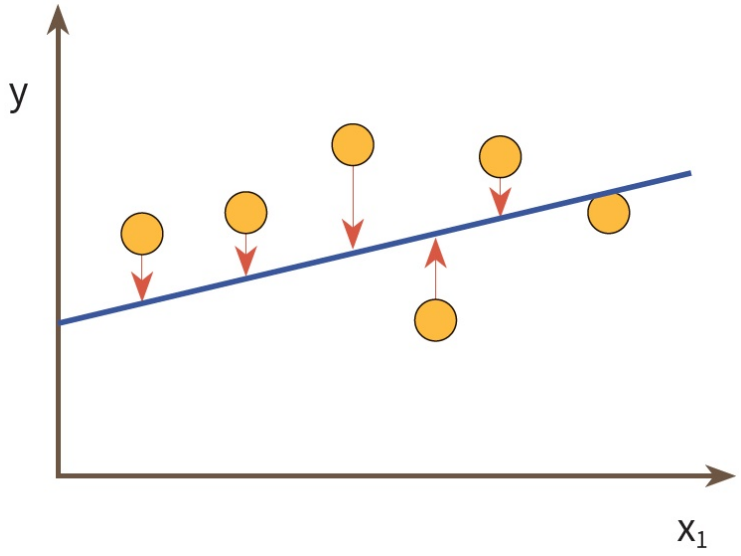




$$f(x) = Wx + b$$

손실 함수 (Loss Function)





손실함수

- 직선과 데이터 사이의 간격을 제곱하여 합한 값을 손실 함수(loss function)
- 비용 함수(cost function)

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

일반화

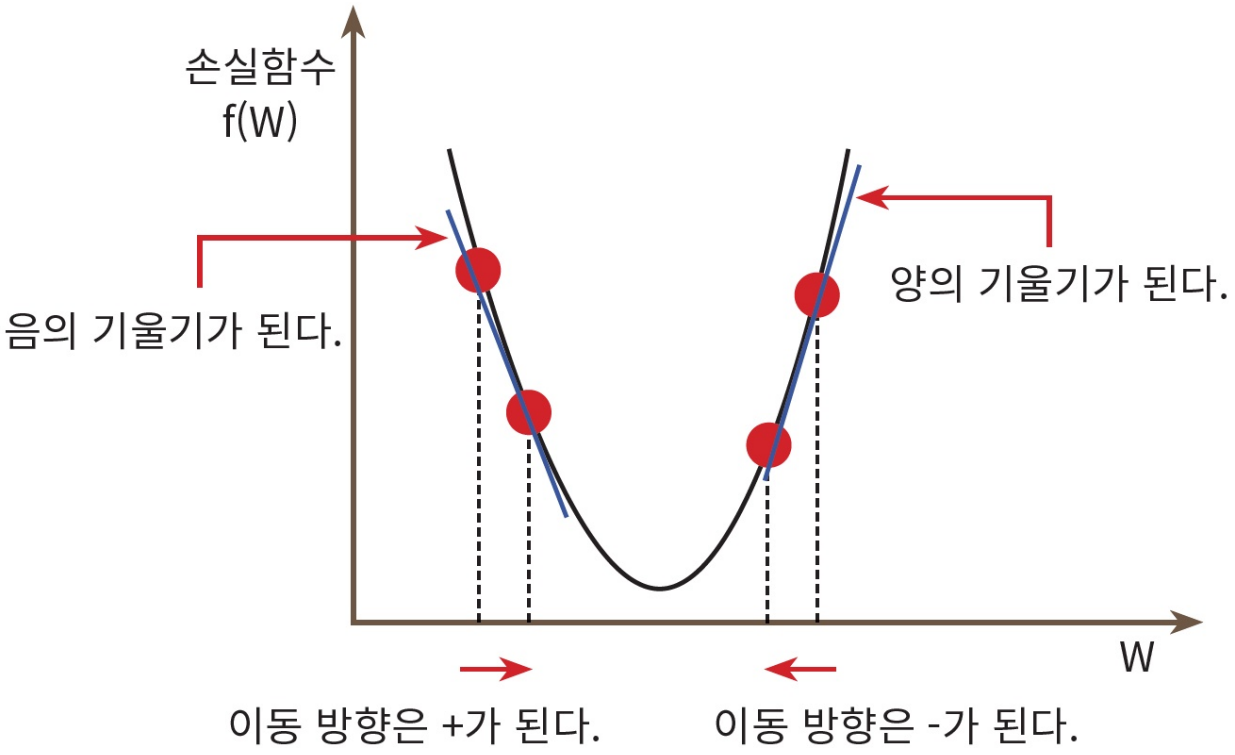
$$Loss(W, b) = \frac{1}{n}\sum_{i=1}^{n} (f(x_i) - y_i)^2 = \frac{1}{n}\sum_{i=1}^{n} ((Wx_i + b) - y_i)^2$$

우리가 찾는 것

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

손실 함수 최소화

- 경사 하강법(gradient descent method)



선형 회귀에서 경사하강법

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

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

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

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

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

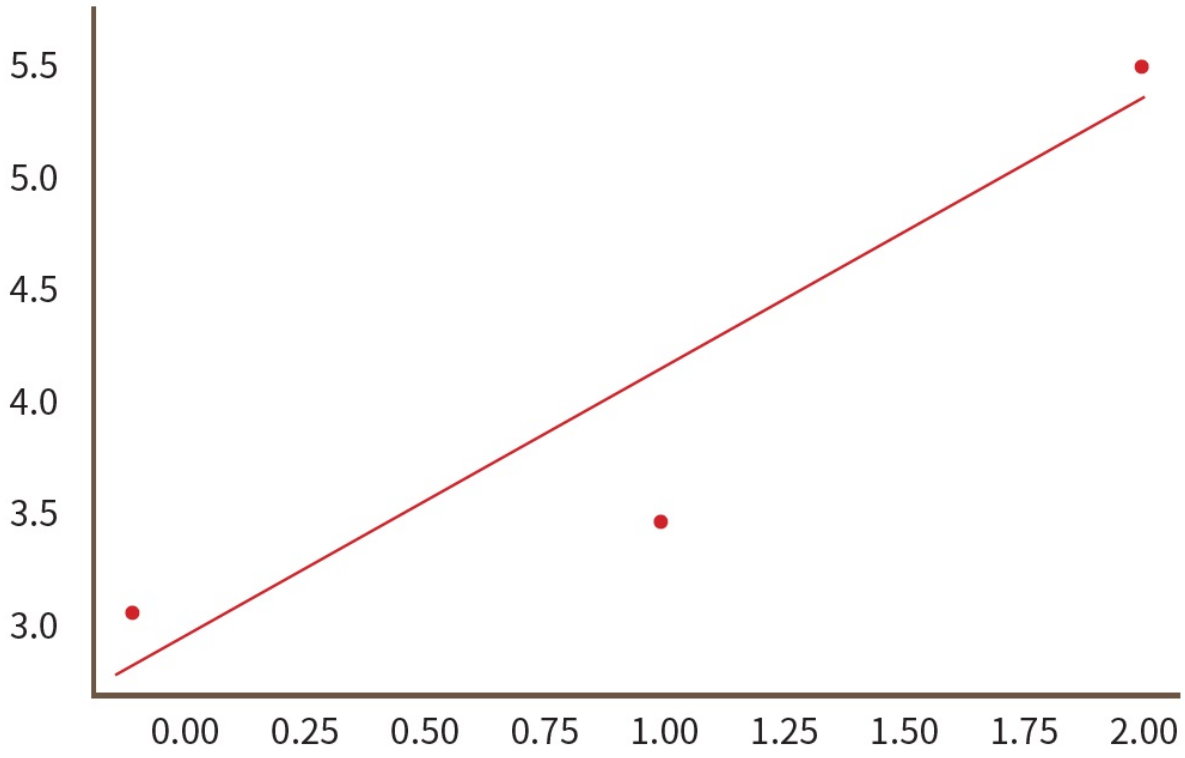
경사 하강법 구현

```
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

lrate = 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 - lrate * dW  # 기울기 수정
    b = b - lrate * db  # 절편 수정
```

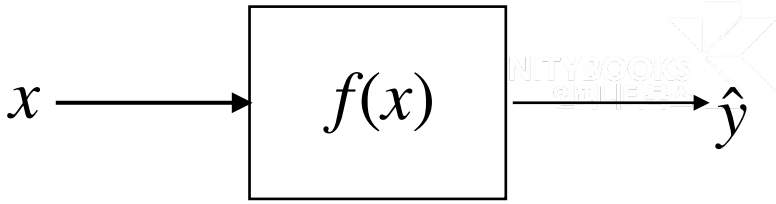
경사 하강법 구현


```
# 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()
```

경사 하강법 구현

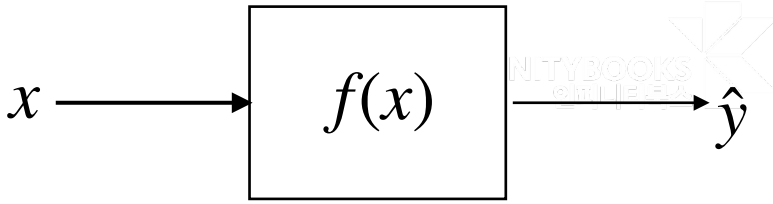


선형회귀 평가



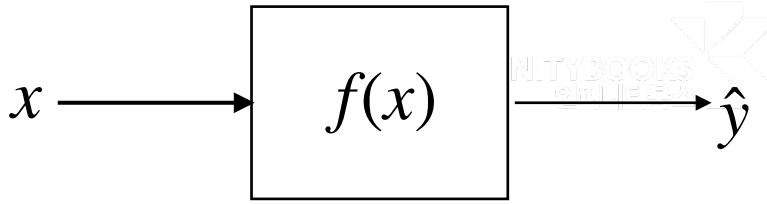
- 데이터셋 $D = \{(x_i, y_i) \mid i = 1, \ldots, N\}$
  - 입력값 $x_i \in \mathbb{R}$
  - 실제값 $y_i \in \mathbb{R}$
- 학습된 모델 $\hat{y} = f(x)$
  - 입력값 $x \in \mathbb{R}$
  - 출력값 $\hat{y} \in \mathbb{R}$
- 데이터 (x_i, y_i)의 평가
  - 모델 $\hat{y}_i = f(x_i)$
  - 만약 $\hat{y}_i \simeq y_i$ 않으면 오류가 발생

선형회귀 평가



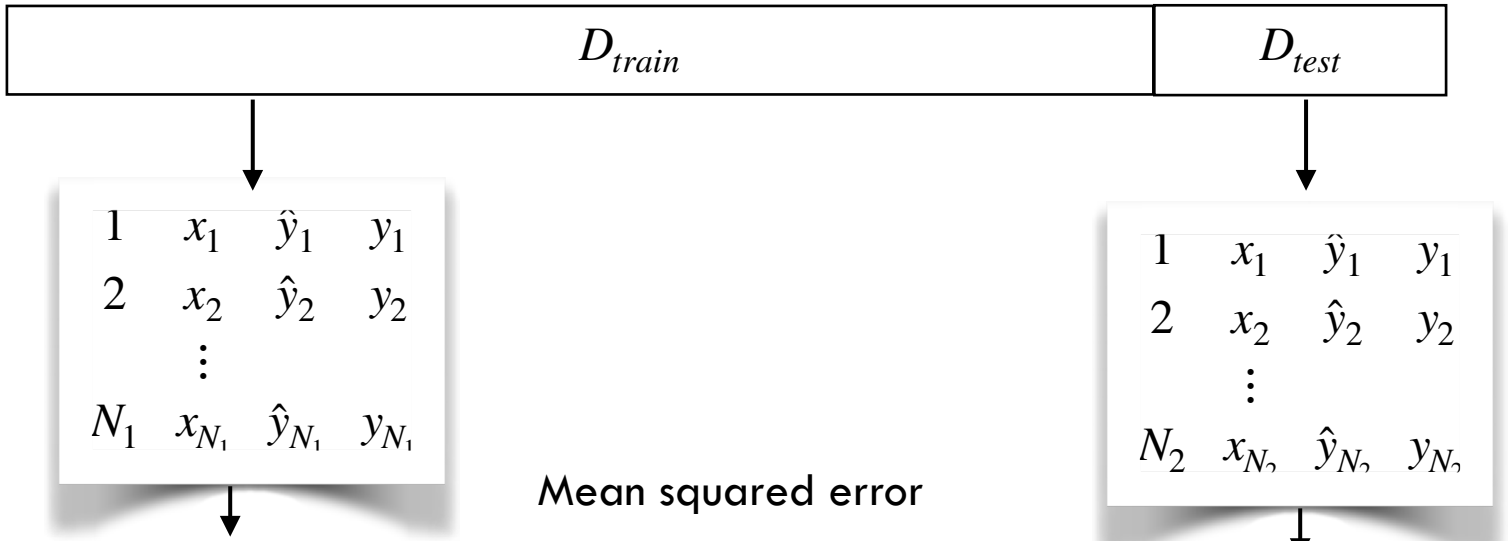
- 객관적 평가를 여러 경우에 모델을 평가
- 데이터셋을 훈련과 테스트 데이터셋으로 분리 $D = D_{train} \cup D_{test}$
  - 훈련 데이터셋 $D_{train}, N_1 = |D_{train}|$
  - 테스트 데이터셋 $D_{test}, N_2 = |D_{test}|$
  - 나누는 비율은 8:2가 보편적으로 선택. 그러나 데이터셋 크기에 따라 다름
  - 임의 선택을 이용하여 데이터셋을 나눔
- 주어진 데이터셋에서 반복적으로 평가를 진행
  - 훈련과 테스트 데이터셋에 대해 동일하게 평가
  - k-way 교차 평가: 훈련과 테스트 데이터셋으로 나누어 k번 평가 진행
  - k-way 검증 교차 평가: 훈련, 검증 과 테스트 데이터셋으로 나누어 k번 평가 진행

5-way 선형회귀 평가



$$D = \{(x_i, y_i) \mid i = 1, \ldots, N\}$$

$$D = D_{train} \cup D_{test}$$

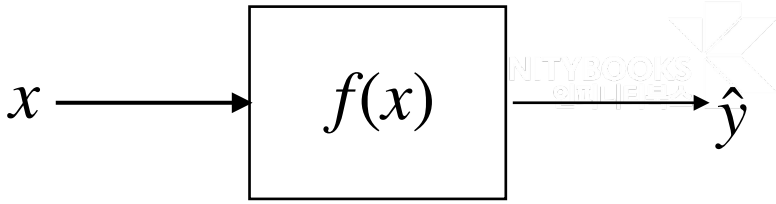


Mean squared error

$$\text{MSE}_{train} = \frac{\sum_{i=1}^{N_1} (\hat{y}_i - y_i)^2}{N_1}$$

$$\text{MSE}_{test} = \frac{\sum_{i=1}^{N_2} (\hat{y}_i - y_i)^2}{N_2}$$

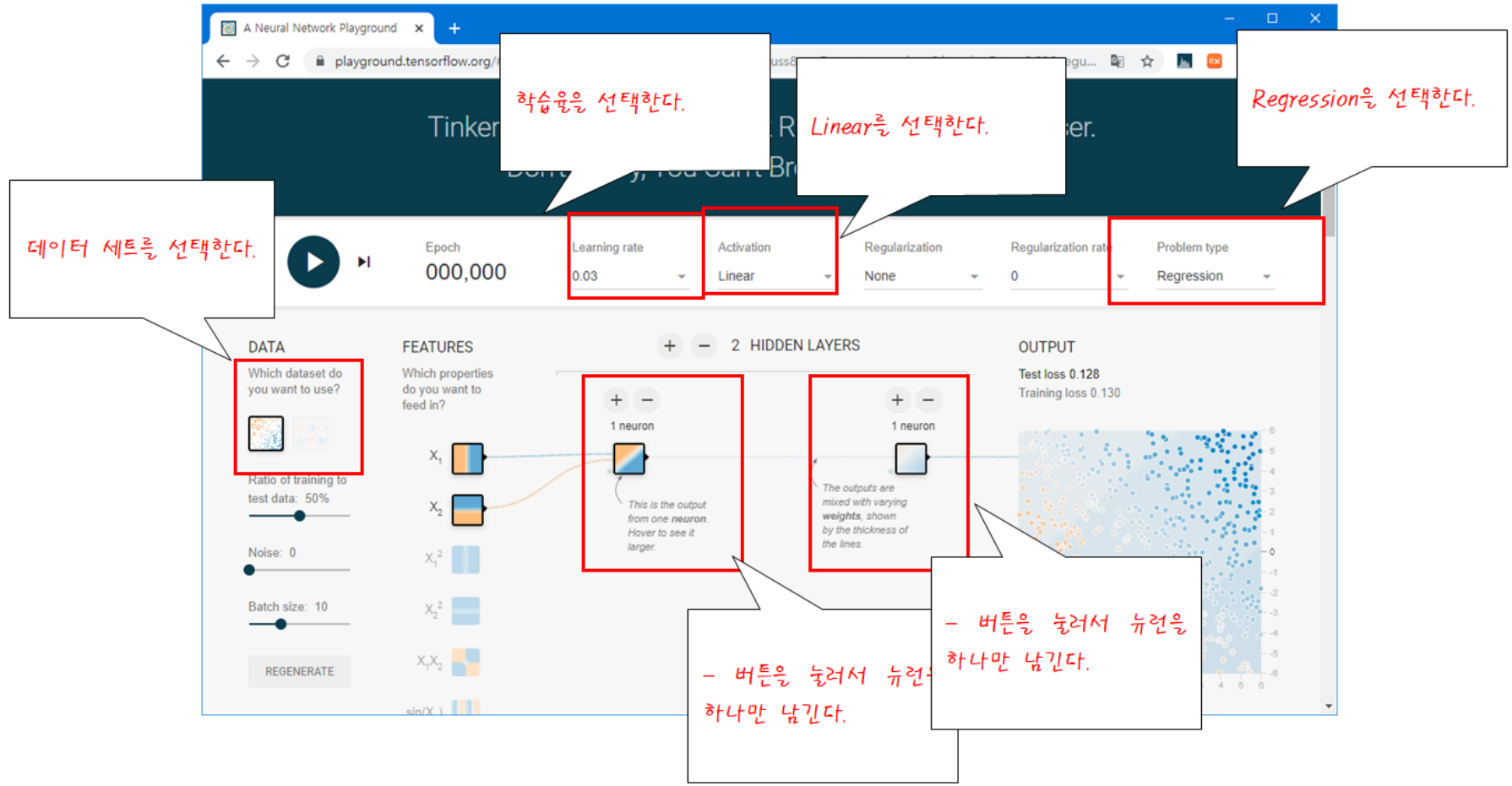
5-way 선형회귀 평가



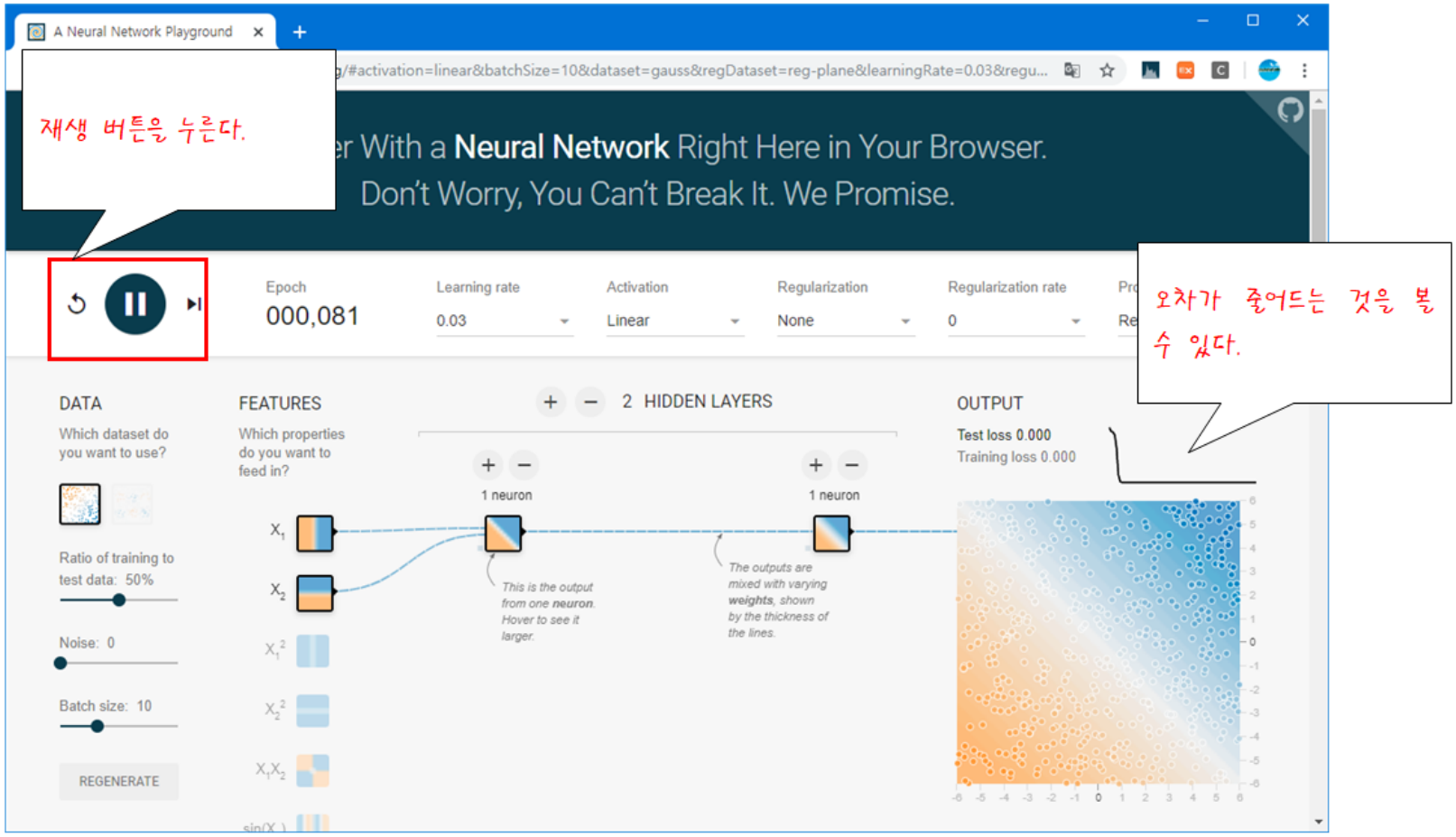
- 테스트셋 선택이 가능한 모든 조합에 대해 평가 (가능한 모든 경우의 수는?)
- 모델 최종 평가는 모든 평가의 평균임
- 5-way 교차 검증의 경우 훈련 데이터셋에서 검증 데이터셋을 일부 선택
  - 훈련 모델은 검증 데이터셋을 제외한 데이터로 구성
  - 훈련 모델을 검증 데이터셋으로 평가 후 가장 성능이 높은 훈련 모델을 선택하여 테스트셋을 평가

예: 학습률 실습

- 구글의 텐서 플로우 플레이그라운드는 이주 유용한 사이트(https://playground.tensorflow.org)이다.



예: 학습률 실습


A Neural Network Playground
재생 버튼을 누른다.
With a Neural Network Right Here in Your Browser.
Don't Worry, You Can't Break It. We Promise.
Epoch
000,081
Learning rate
0.03
Activation
Linear
Regularization
None
Regularization rate
0
오차가 줄어드는 것을 볼 수 있다.
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?
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

선형 회귀 예제

- Scikit-Learn 라이브러리를 사용하여 회귀 함수를 구현

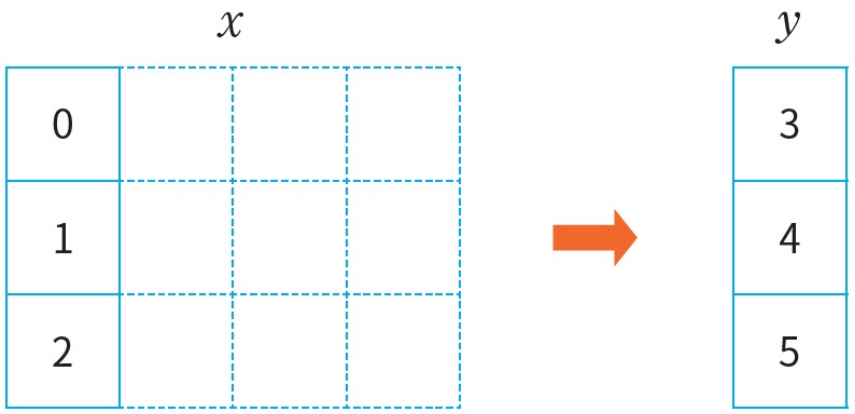
```
import matplotlib.pylab as plt
from sklearn import linear_model

# 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)
```

선형 회귀 예제


```
>>> reg.coef_                    # 직선의 기울기
array([1.25])

>>> reg.intercept_               # 직선의 y-절편
2.7500000000000004

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

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

학습 데이터 만들기

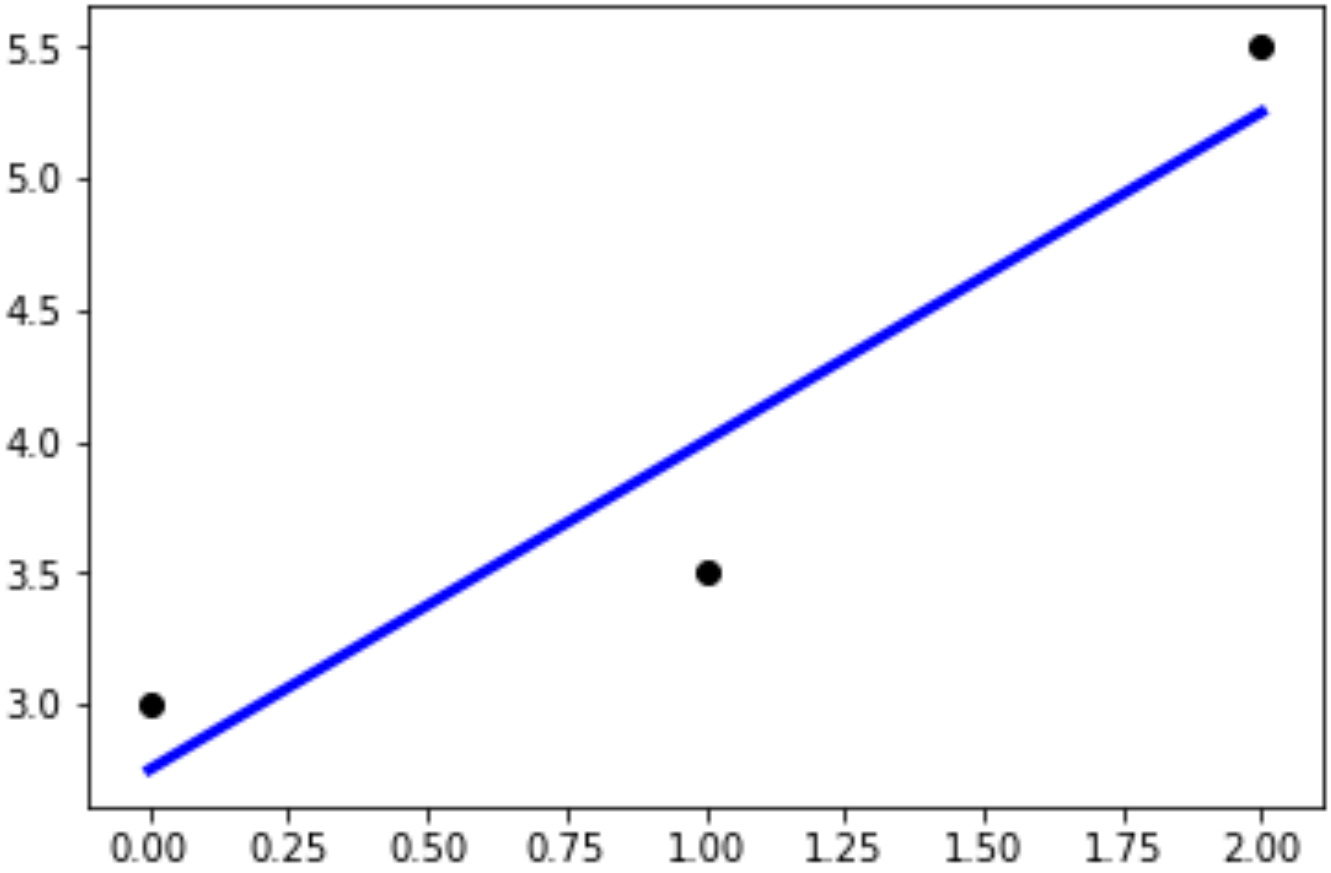
- 학습 데이터는 2차원로 구성



시각화


```
# 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()
```

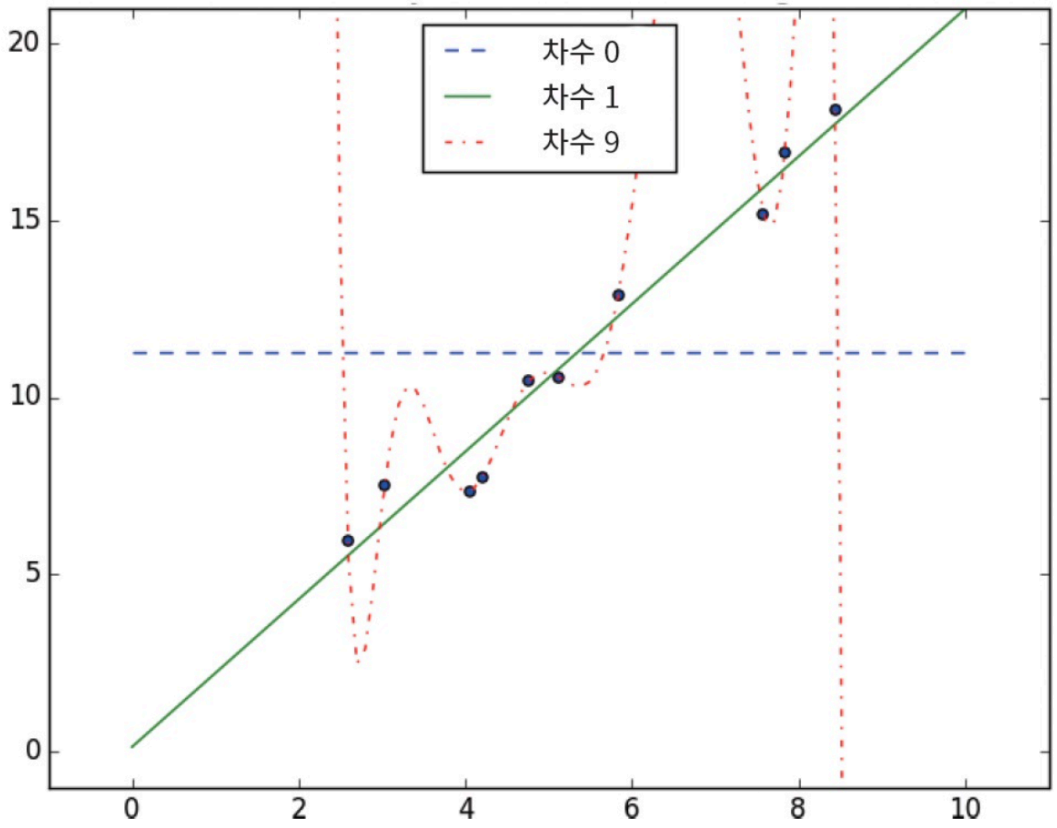
실행 결과



과잉 적합 vs 과소 적합

- 과잉 적합(overfitting)
  - 학습하는 데이터에서는 성능이 뛰어나지만 새로운 데이터(일반화)에 대해서는 성능이 잘 나오지 않는 모델을 생성
- 과소 적합(underfitting)
  - 학습 데이터에서도 성능이 좋지 않은 경우
  - 모델 자체가 적합지 않은 경우

과잉 적합 vs 과소 적합



예: 선형 회귀 실습

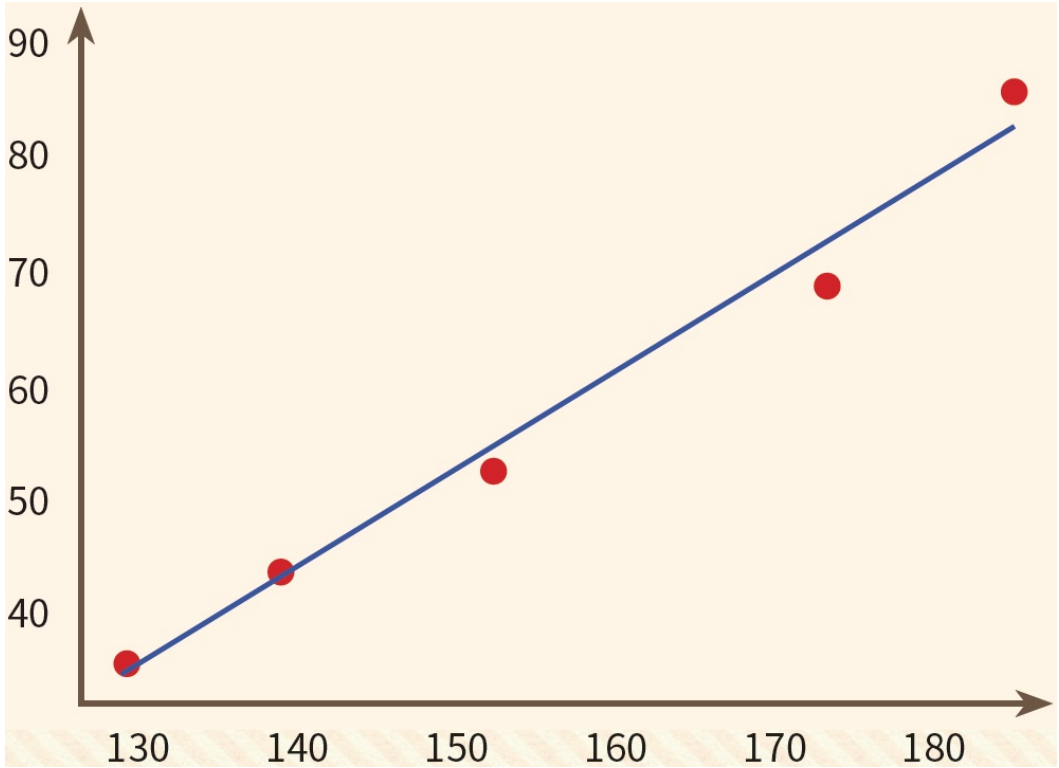
- 인간의 키와 몸무게는 어느 정도 비례할 것으로 예상
- 선형 회귀를 이용하여 학습시키고 키가 165cm일 때의 예측값을 추정

예: 선형 회귀 실습

```
import matplotlib.pylab 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()
```

선형 회귀 예제


```
[67.30998637]
```



예: 당뇨병 예제

- sklearn 라이브러리에는 당뇨병 환자들의 데이터 이용

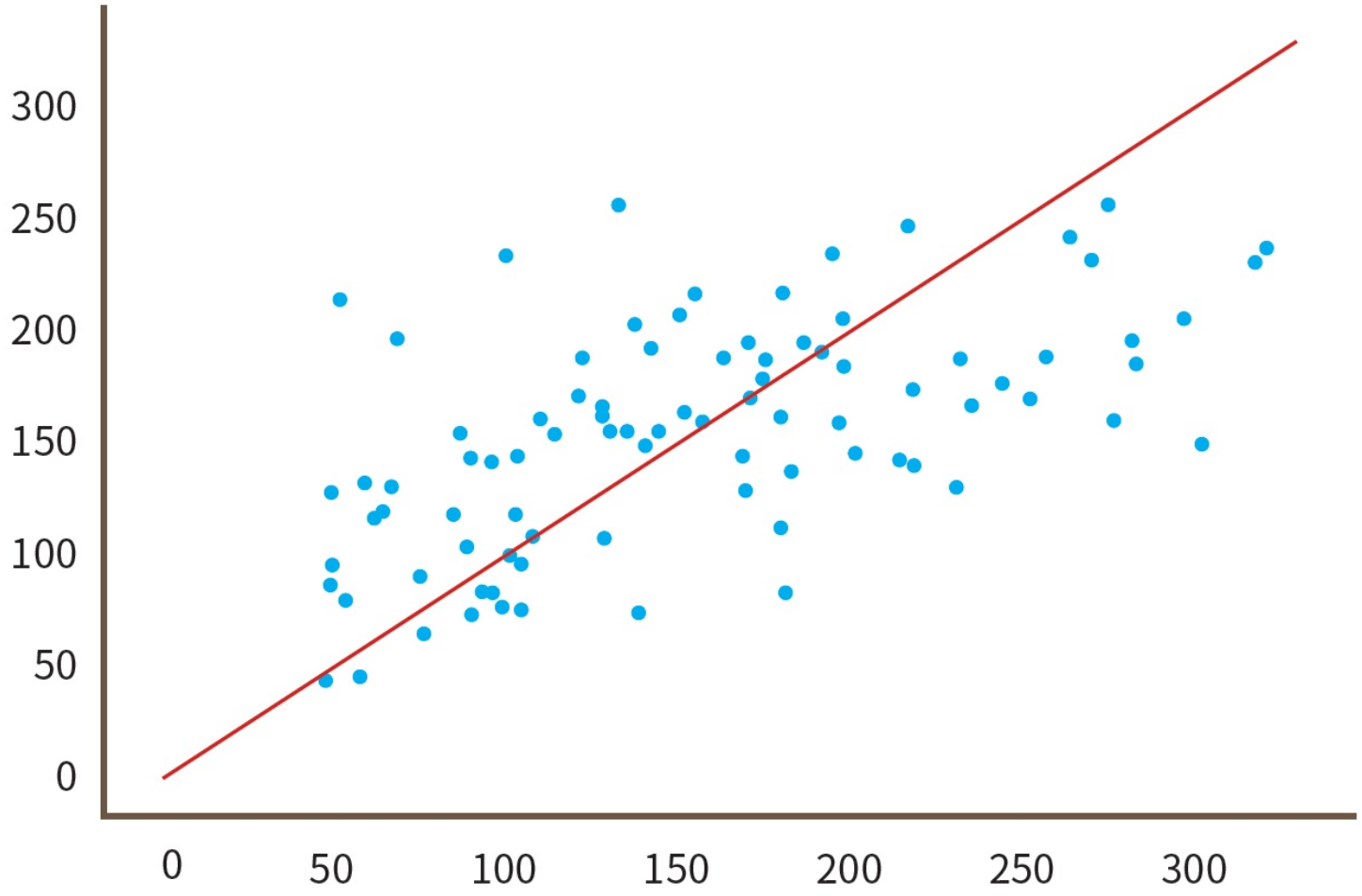
예: 당뇨병 예제

```
import matplotlib.pylab as plt
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn import datasets

# 당뇨병 데이터 세트를 적재한다.
diabetes = datasets.load_diabetes()

# 학습 데이터와 테스트 데이터를 분리한다.
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(diabetes.data, diabetes.target,
test_size=0.2, random_state=0)

선형 회귀 모델로 학습을 수행한다.
model = LinearRegression()
model.fit(X_train, y_train)
```

예: 당뇨병 예제

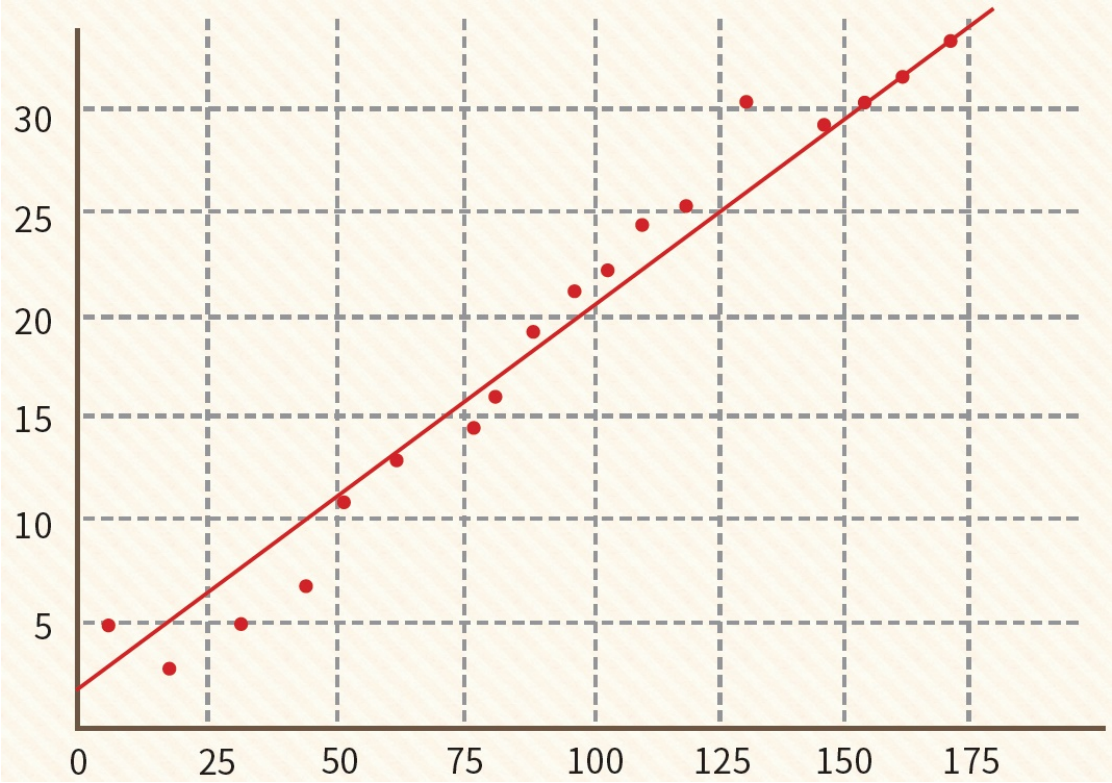

```
# 테스트 데이터로 예측해보자.
y_pred = model.predict(X_test)

# 실제 데이터와 예측 데이터를 비교해보자.
plt.plot(y_test, y_pred, '.')

# 직선을 그리기 위하여 완벽한 선형 데이터를 생성한다.
x = np.linspace(0, 330, 100)
y = x
plt.plot(x, y)
plt.show()
```

실행 결과



예: 면적에 따른 집값 예측

- 사용자가 아파트 면적을 입력하면 아파트의 가격이 출력되는 시스템을 구현
- 아파트 면적과 가격은 모두 실수이므로 기계 학습의 방법 중에서 선형 회귀를 사용



Summary

- 지도 학습에는 회귀(regression)와 분류(classification)가 있음
- 선형 회귀는 입력 데이터를 가장 잘 설명하는 직선의 기울기와 절편값을 찾는 문제
- 손실 함수(loss function)란 실제 데이터와 직선 간의 차이를 제곱한 값
- 손실 함수의 값이 작아지는 방향을 알려면 일반적으로 경사 하강법(gradient descent method)을 사용