선형 회귀 Linear Regression - 연속
로지스틱 회귀 logistic regression - 카테고리

명지대학교
MYONGJI UNIVERSITY

학습 목표

- 회귀의 개념을 이해한다.
- 경사 하강법을 이해한다.
- 과잉 적합과 과소 적합을 이해한다.
- 파이썬과 sklearn을 이용하여 회귀를 구현해본다.

회귀(regression)와 분류(classification)

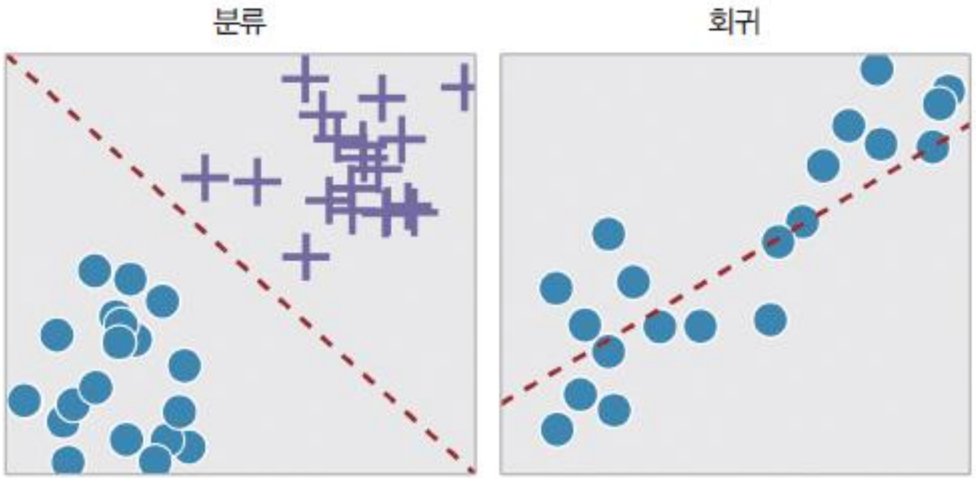


그림 4-1 회귀와 분류

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 y가 실수이고 입력 x도 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**
  - 기울기->가중치
  - 절편->바이어스

선형 회귀 예제

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

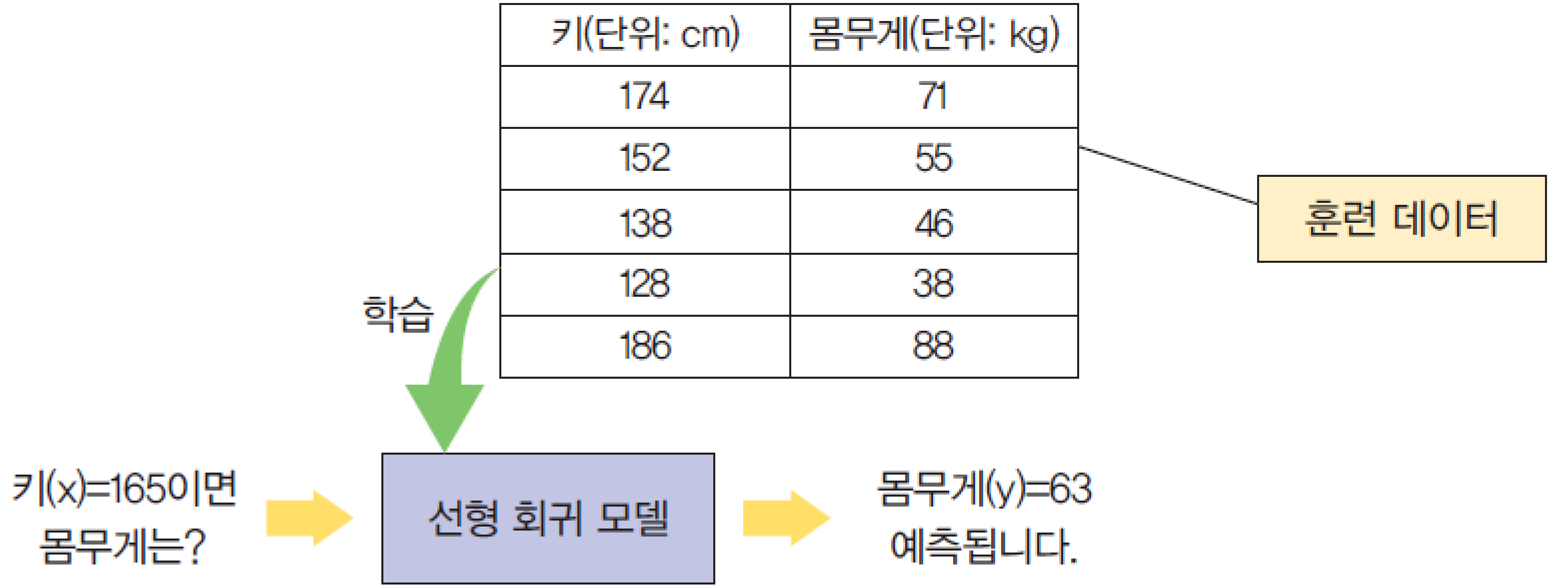


그림 4-3 선형 회귀의 예제

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

$$f(x, y, z) = w_0 + w_1 x + w_2 y + w_3 z$$

$$\text{매출} = w_0 + w_1 \times \text{인터넷 광고} + w_2 \times \text{신문광고} + w_3 \times TV\text{광고}$$

선형 회귀의 원리

x	y
1	2
2	5
3	6

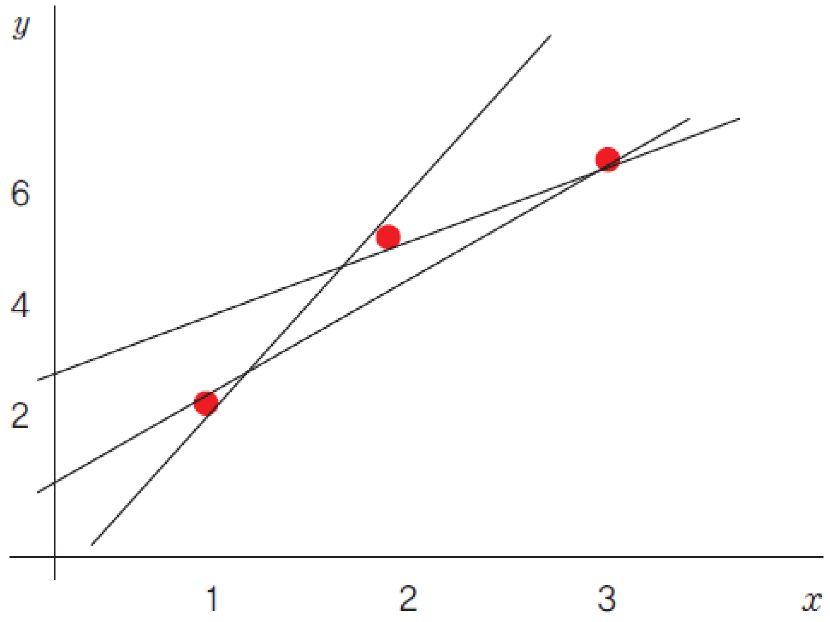
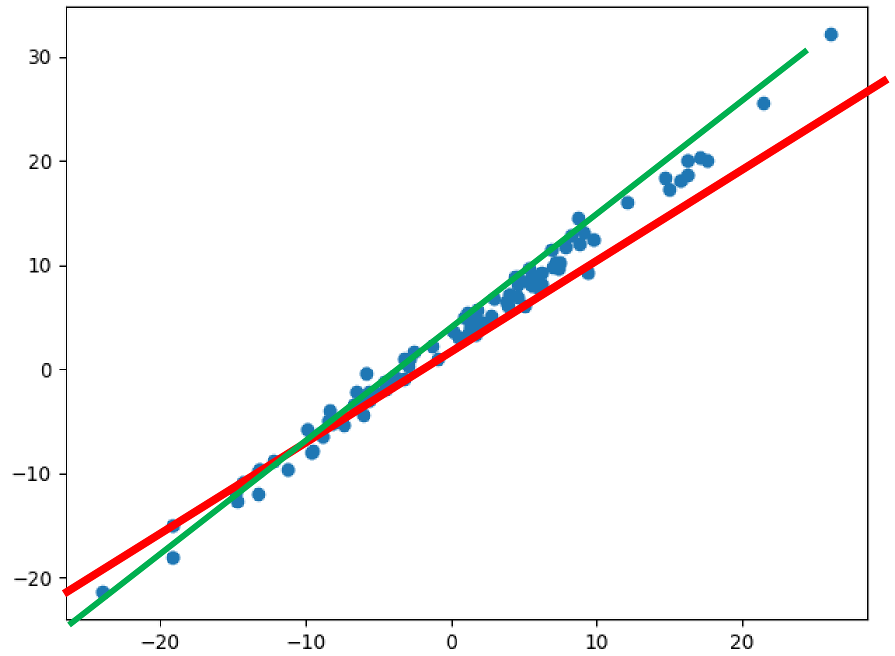
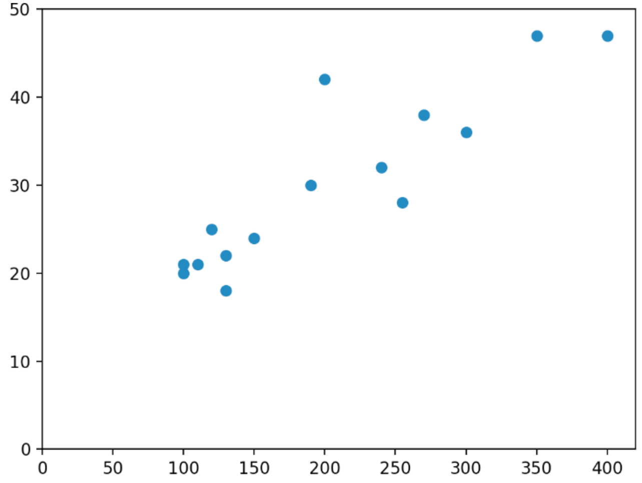


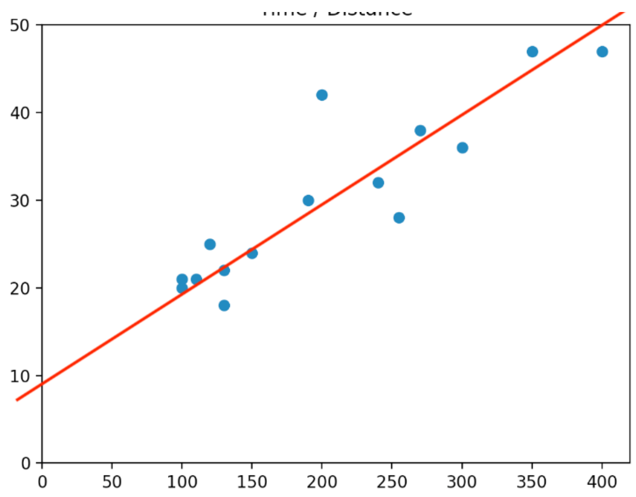
그림 4-4 데이터와 직선

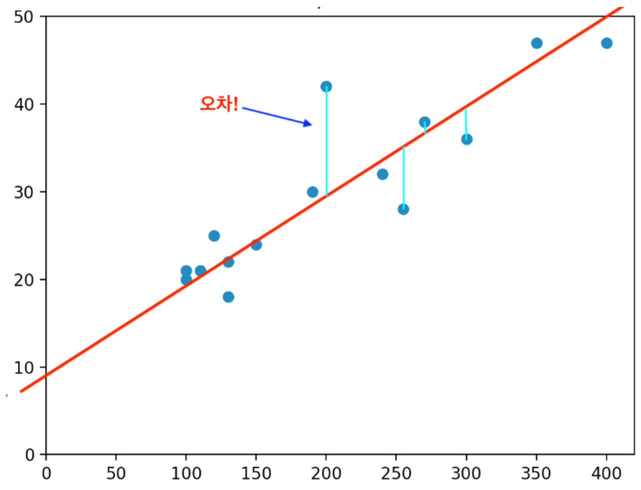
```
import numpy as np
from sklearn.linear_model import LinearRegression

X = 10 * np.random.randn(100,1)
y = 3 + X +np.random.randn(100,1)

plt.plot(X, y, 'o')
plt.show()
```



최선의 것은?







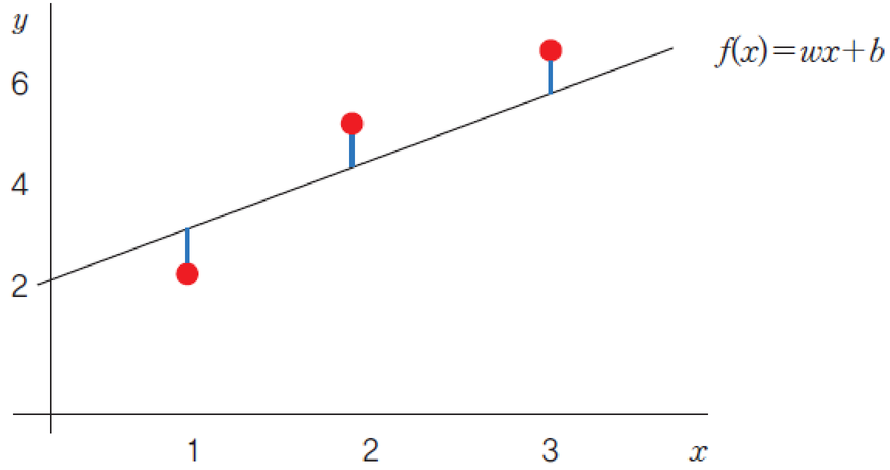


그림 4-5 데이터와 직선 간의 거리

손실함수

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

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

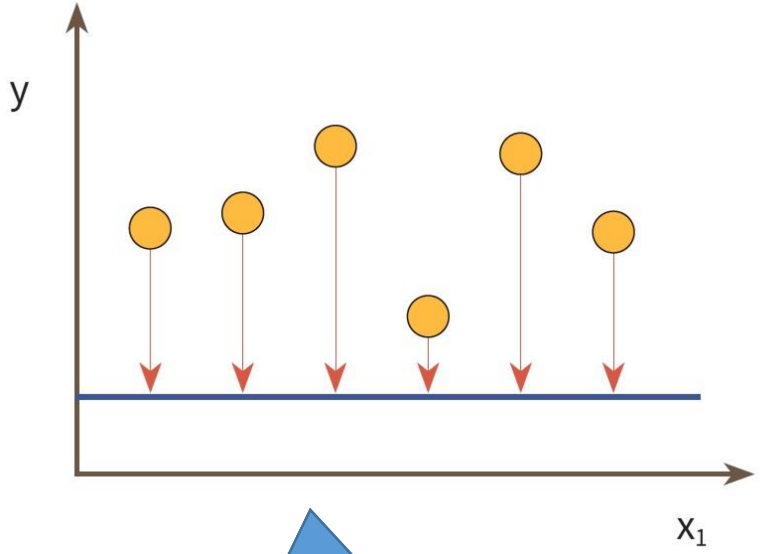
일반화

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

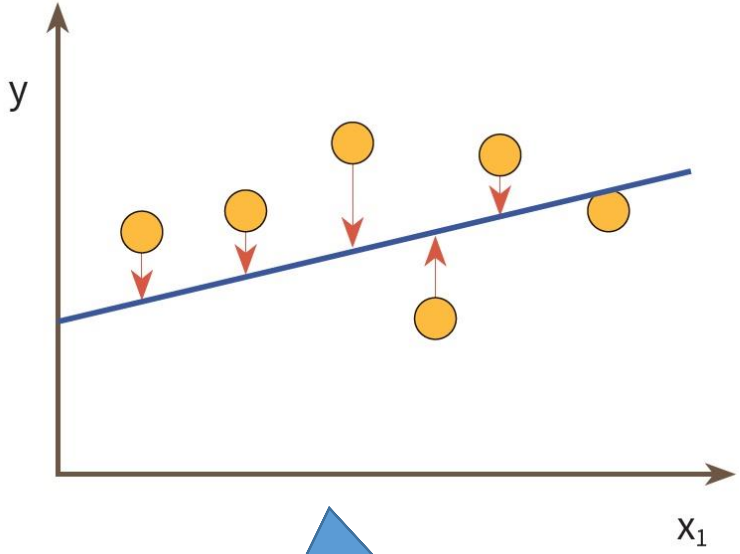
우리가 찾는 것

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

손실 함수



손실이 큰 경우



손실이 작은 경우

선형 회귀에서 손실 함수 최소화 방법

분석적인 방법: 독립 변수와 종속 변수가 각각 하나인 선형 회귀

$$cost(W) = \frac{1}{m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})^2$$

x	Y
1	1
2	2
3	3

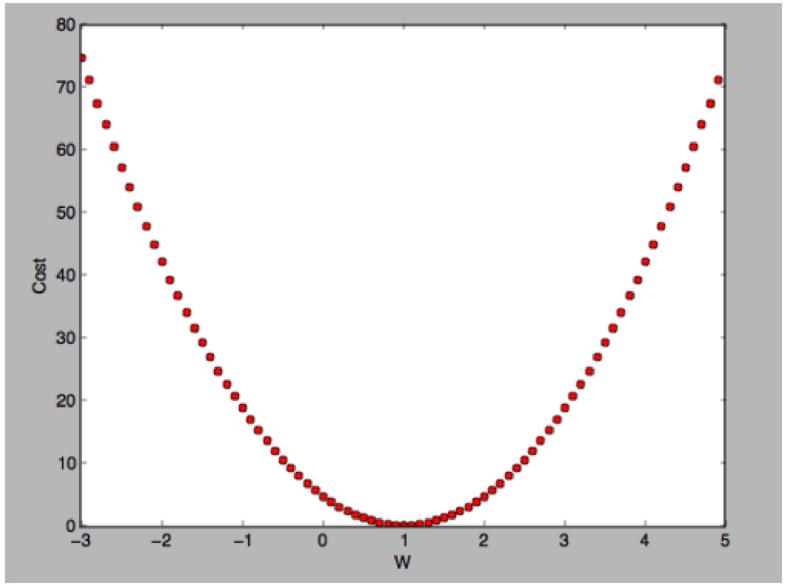
- W=1, cost(W)=0

  $\frac{1}{3}((1*1-1)^2 + (1*2-2)^2 + (1*3-3)^2)$

- W=0, cost(W)=4.67

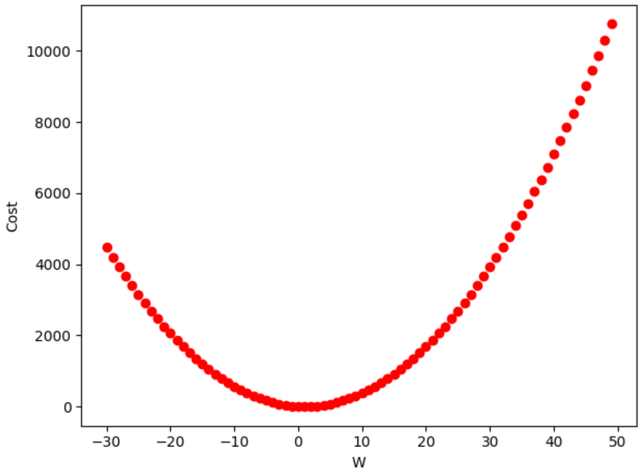
  $\frac{1}{3}((0*1-1)^2 + (0*2-2)^2 + (0*3-3)^2)$

- W=2, cost(W)=?



경사 하강법(gradient descent method):

- cost함수 그래프를 그리면 위와 같은 2차 포물선



- 경사 하강법은 손실 함수가 어떤 형태이라도, 또 매개 변수가 아무리 많아도 적용할 수 있는 일반적인 방법이다.
- 점진적인 학습이 가능하다

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
def forward(wa, x):
  return wa * x

X = np.array([1., 2., 3.])
Y = np.array([1., 2., 3.])
W_val = []
cost_val = []
m = n_samples = len(X)
for i in range(-30, 50):
    W_val.append(i )
    print(forward(i ,X))
    cost_val.append(tf.reduce_sum(tf.pow(forward(i ,X)-Y, 2))/(m))
plt.plot(W_val, cost_val, 'ro')
plt.ylabel('Cost')
plt.xlabel('W')
plt.show()
```

경사하강법

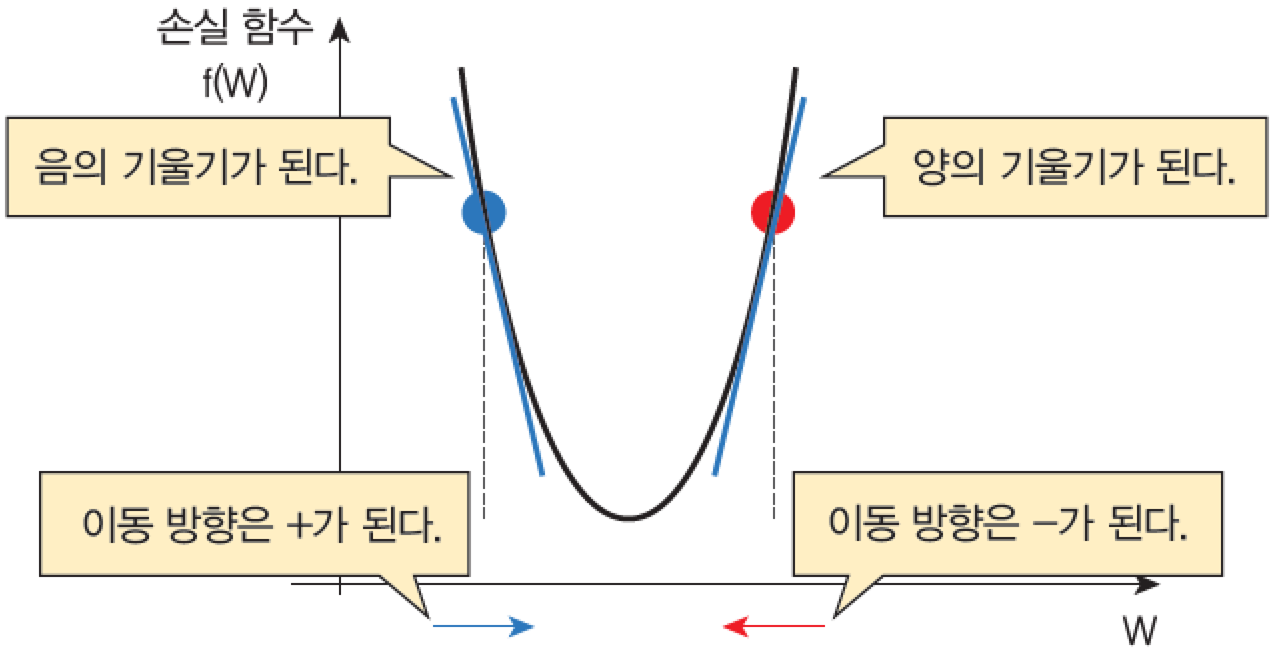
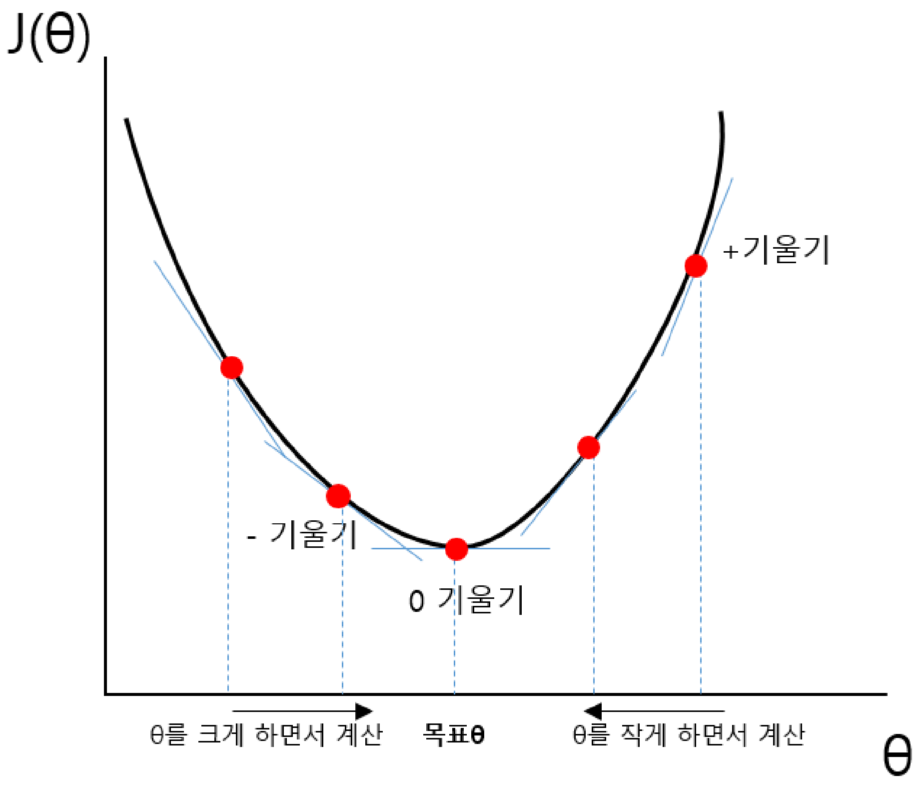
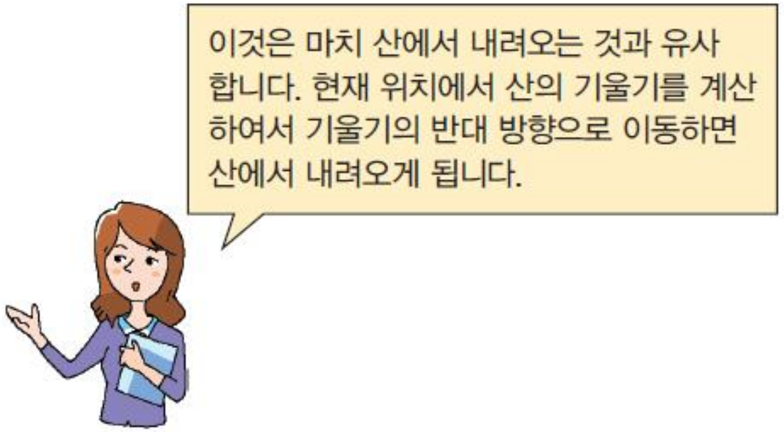


그림 4-7 경사 하강법의 이해



경사하강법


기울기가 음수이니
오른쪽으로 많이 이동
기울기가 음수이지만 작으므로
오른쪽으로 천천히 이동
목표 도달!


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

학습률

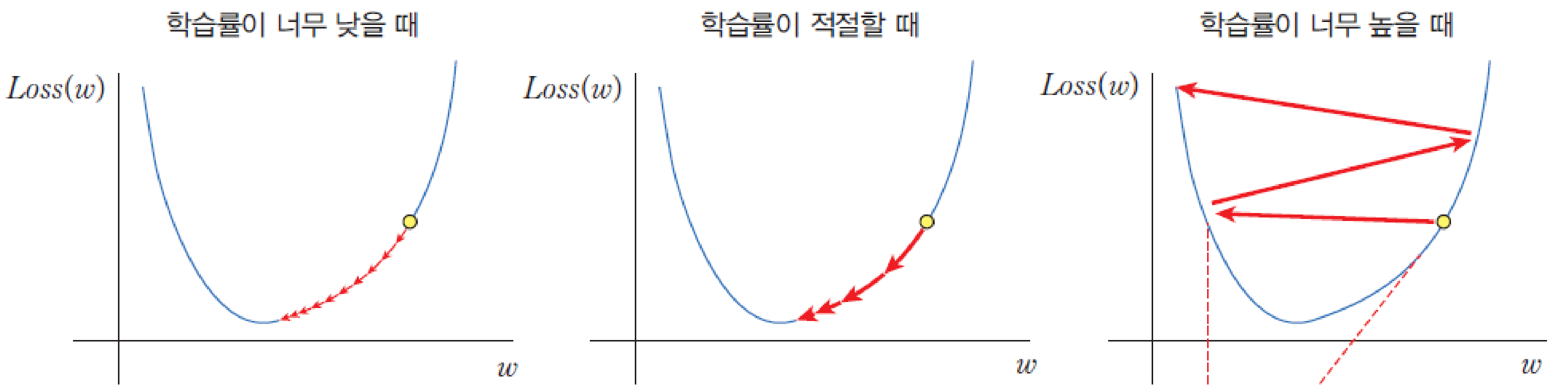


그림 4-9 학습률

지역최소값 문제

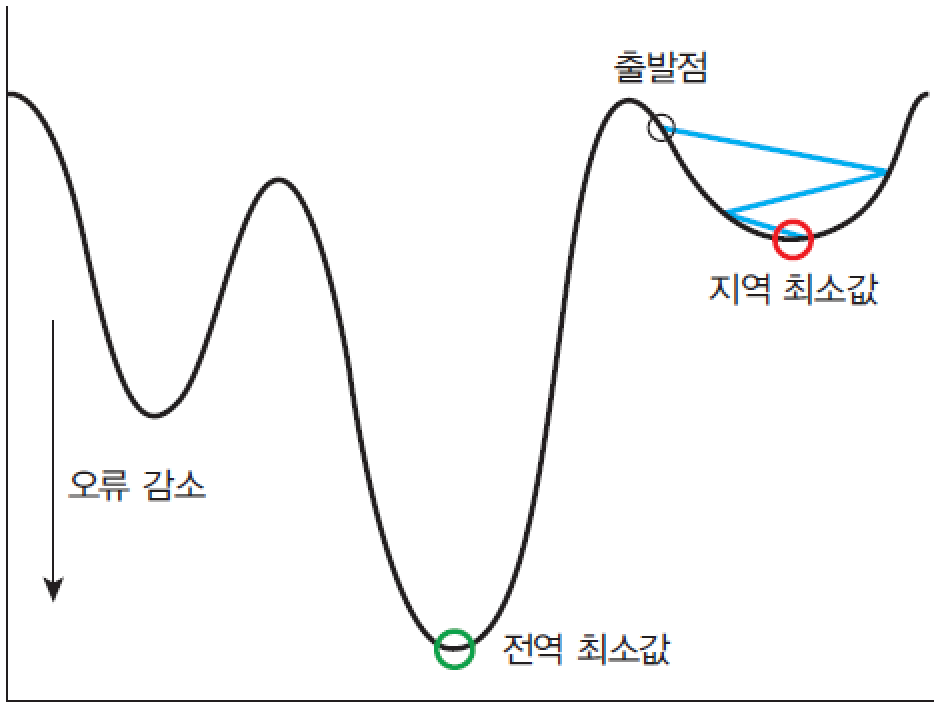


그림 4-10 지역 최소값과 전역 최소값

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

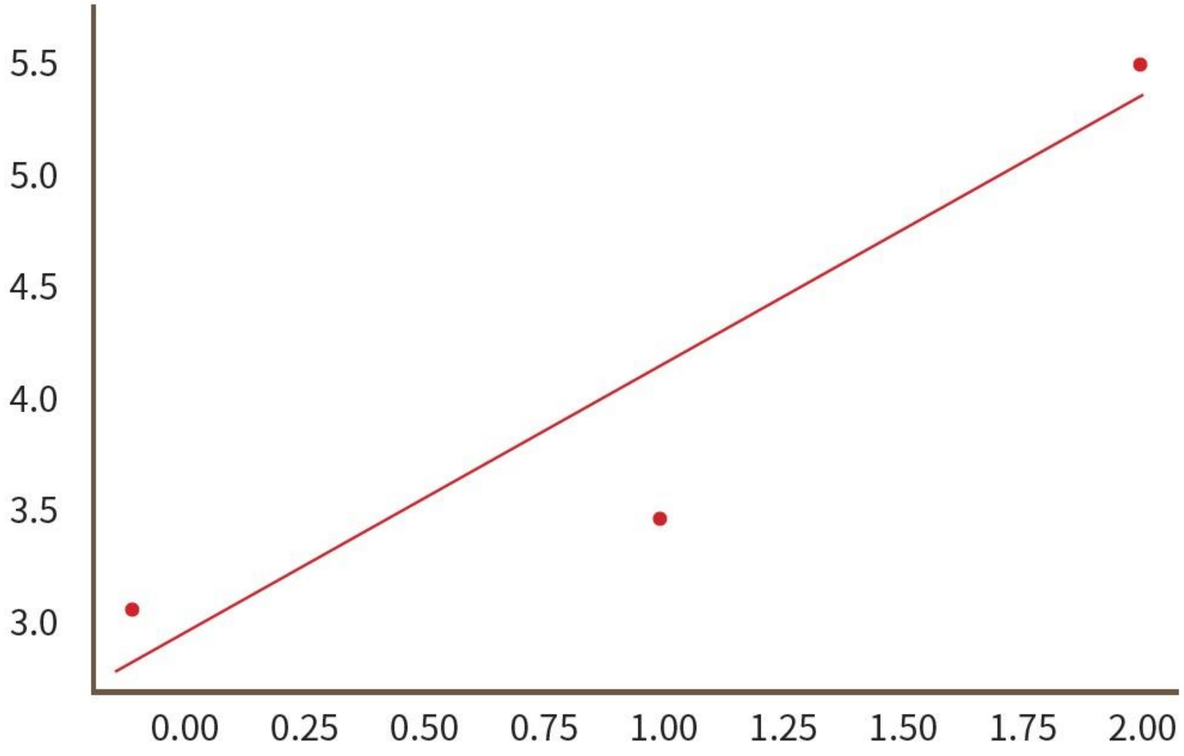
```
import numpy as np
import matplotlib.pyplot as plt

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

W = 0        # 기울기
b = 0        # 절편

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()
```

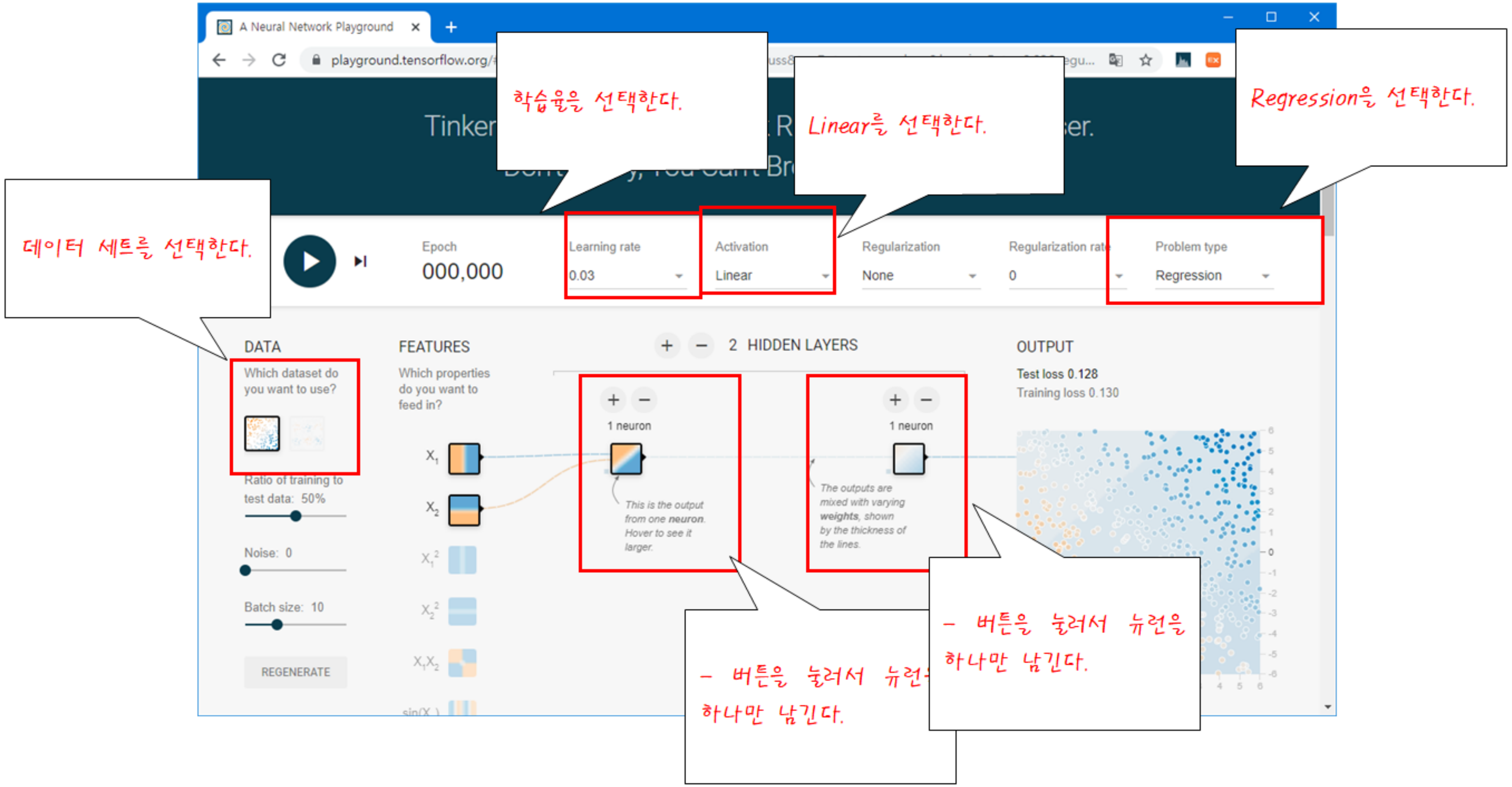
```
1.2532418085611319 2.745502230882486
```

경사 하강법 구현

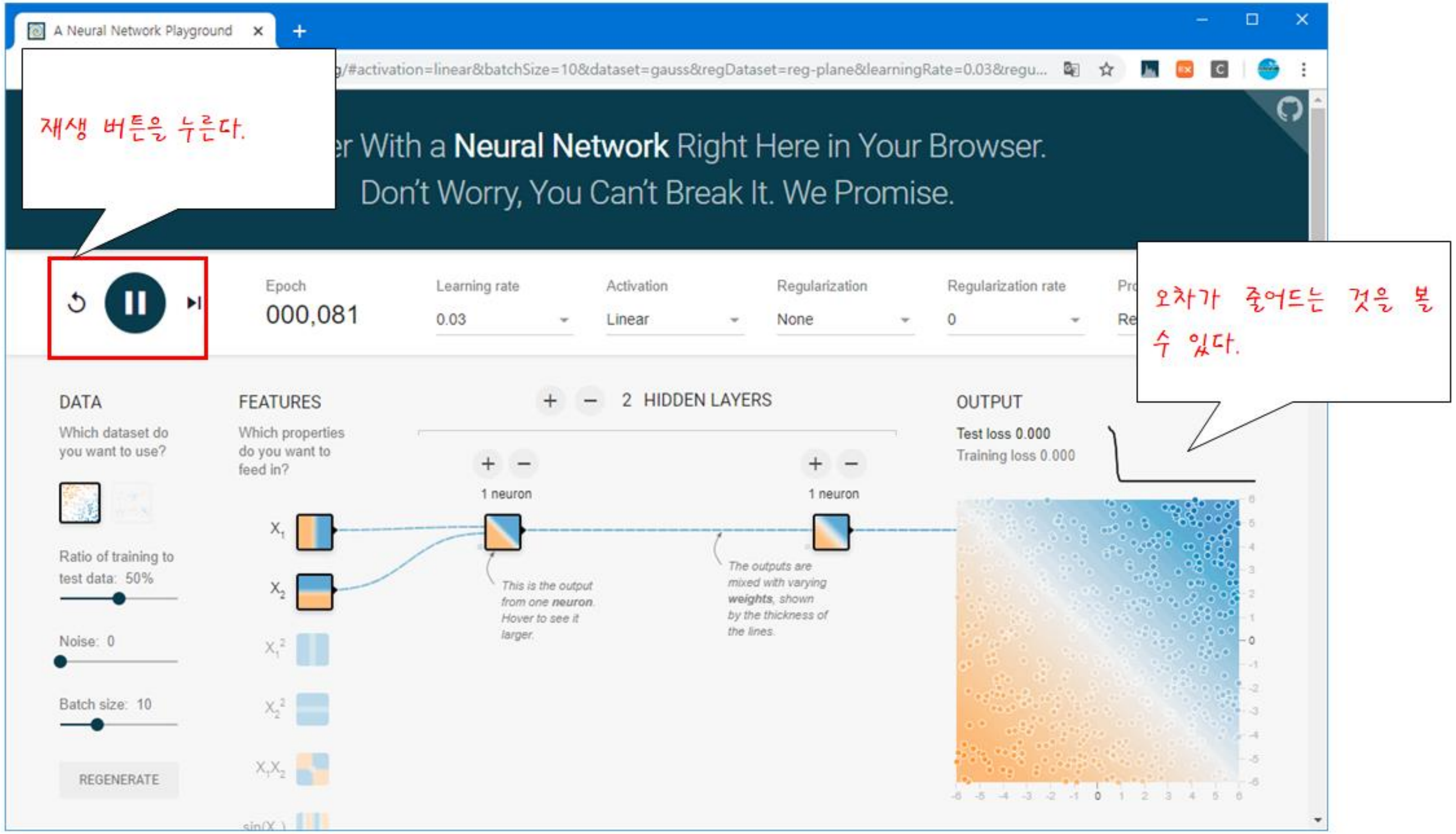


Lab: 학습률 실습

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



Lab: 학습률 실습


A Neural Network Playground
/#activation=linear&batchSize=10&dataset=gauss&regDataset=reg-plane&learningRate=0.03&regu...
재생 버튼을 누른다.
er 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

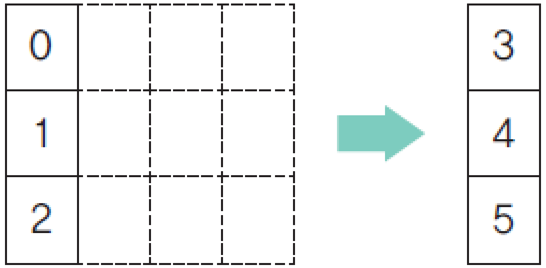
선형 회귀 예제

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

```
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)
```

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.**



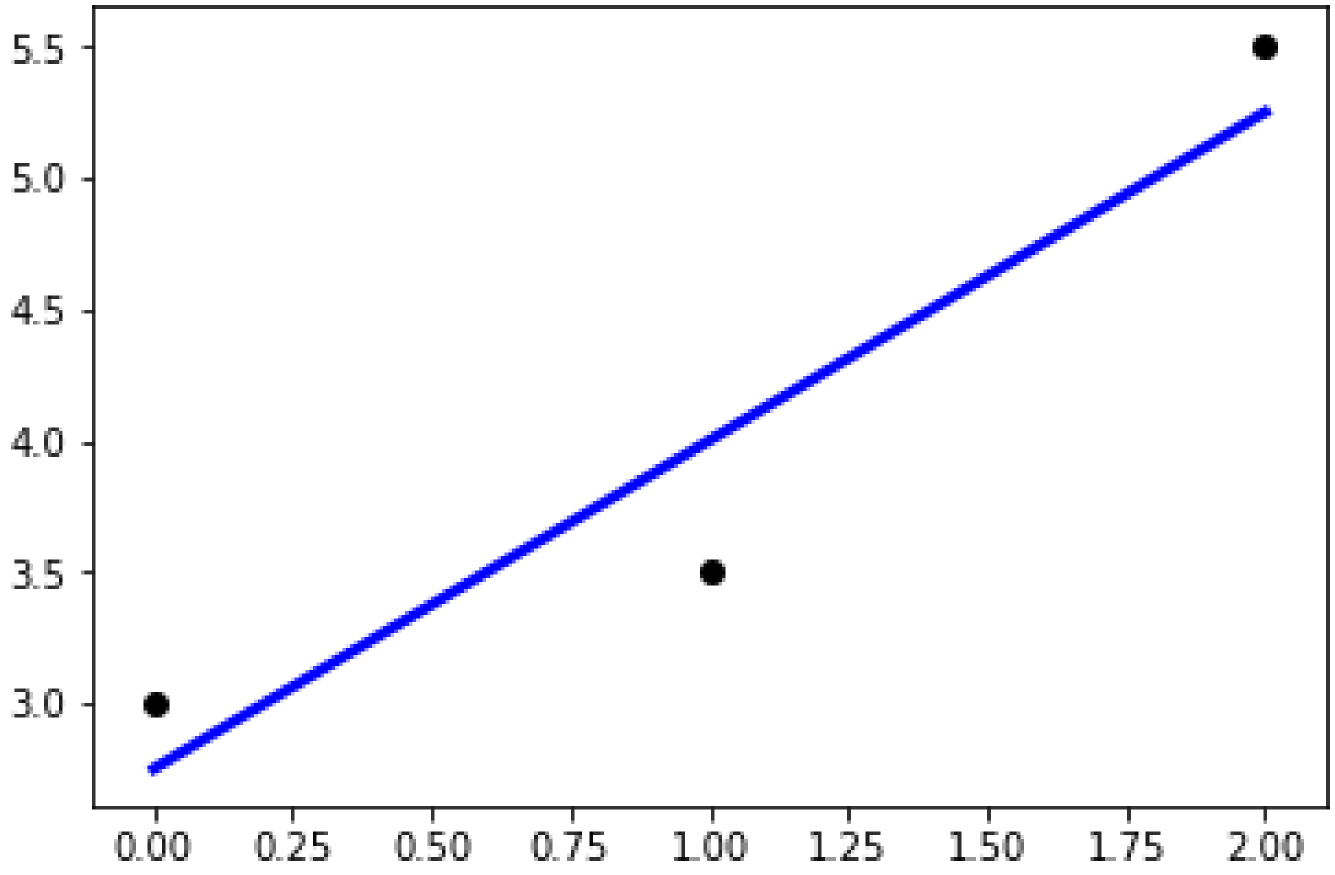
선형 회귀 예제

```
>>> 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.])
```

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()
```

실행 결과



Lab: 선형 회귀 실습

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

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

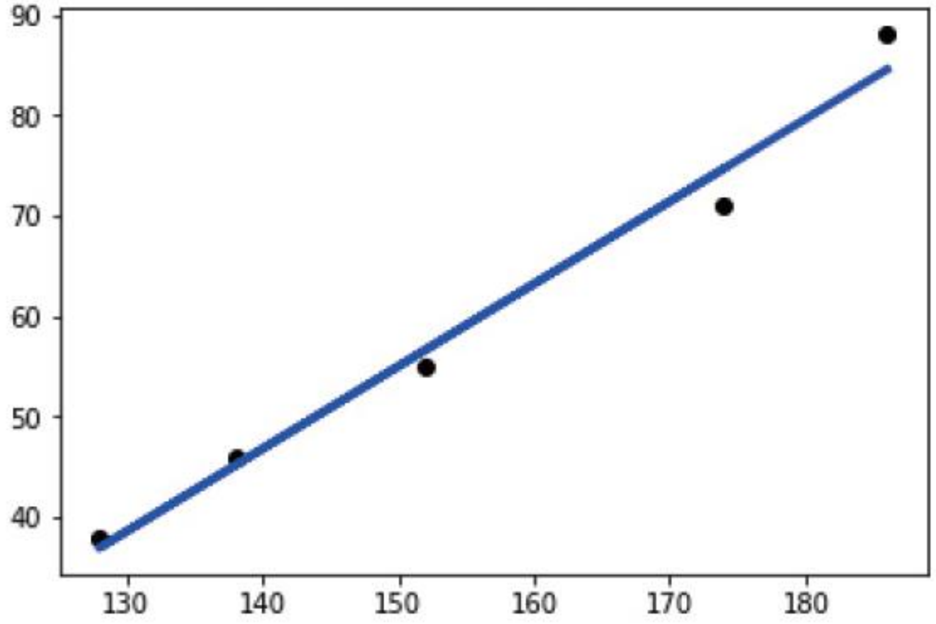
학습 데이터

Sol.

```
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()
```

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 적합지 않은 경우가 많다. 더 나은 모델을 찾아야 한다.

과소적합과 과잉적합

[그림 1.13]의 1차 모델은 과소적합

- 모델의 '용량이 작아' 오차가 클 수밖에 없는 현상

비선형 모델을 사용하는 대안

- [그림 1-13]의 2차, 3차, 4차, 12차는 다항식 곡선을 선택한 예
- 1차(선형)에 비해 오차가 크게 감소함

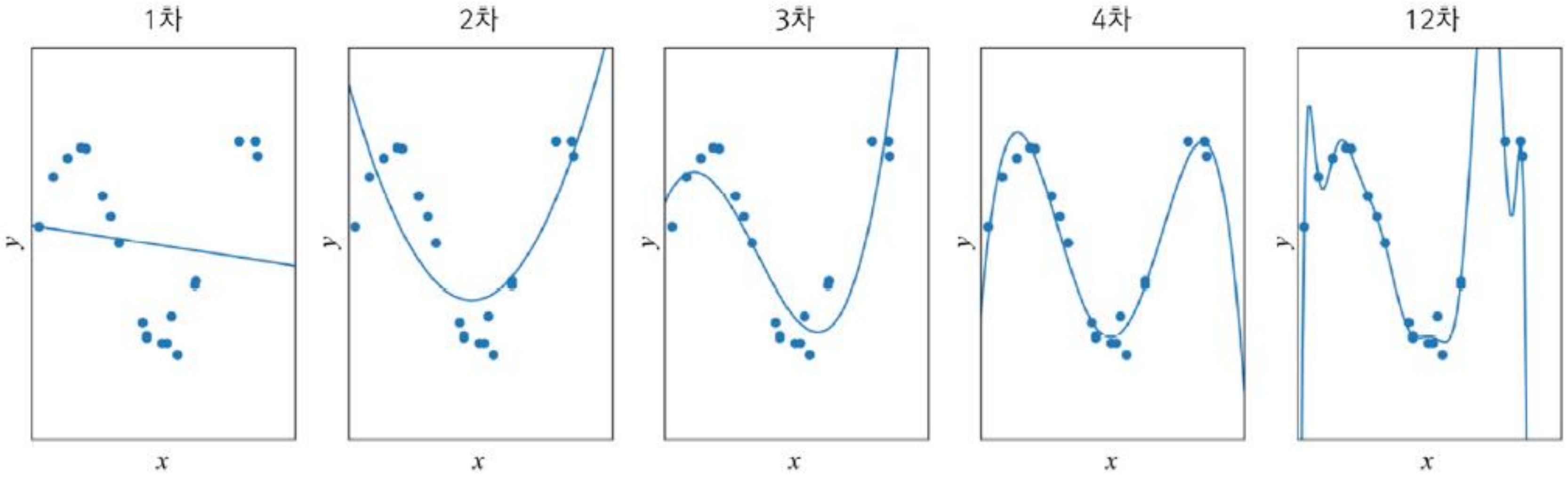
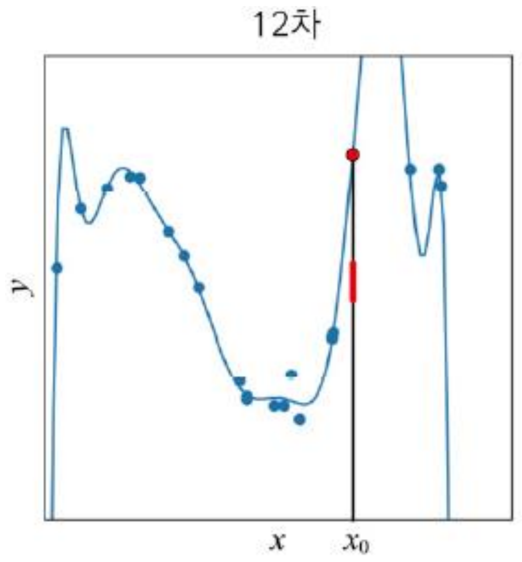


그림 1-13 과소적합과 과잉적합 현상

과잉적합

과잉적합

- 12차 다항식 곡선을 채택한다면 훈련집합에 대해 거의 완벽하게 근사화함
- 하지만 '새로운' 데이터를 예측한다면 큰 문제 발생
    - x_0에서 빨간 막대 근방을 예측해야 하지만 빨간 점을 예측
- 이유는 '용량이 크기' 때문. 학습 과정에서 잡음까지 수용 → 과잉적합 현상
- 적절한 용량의 모델을 선택하는 모델 선택 작업이 필요함



과잉 적합 vs 과소 적합

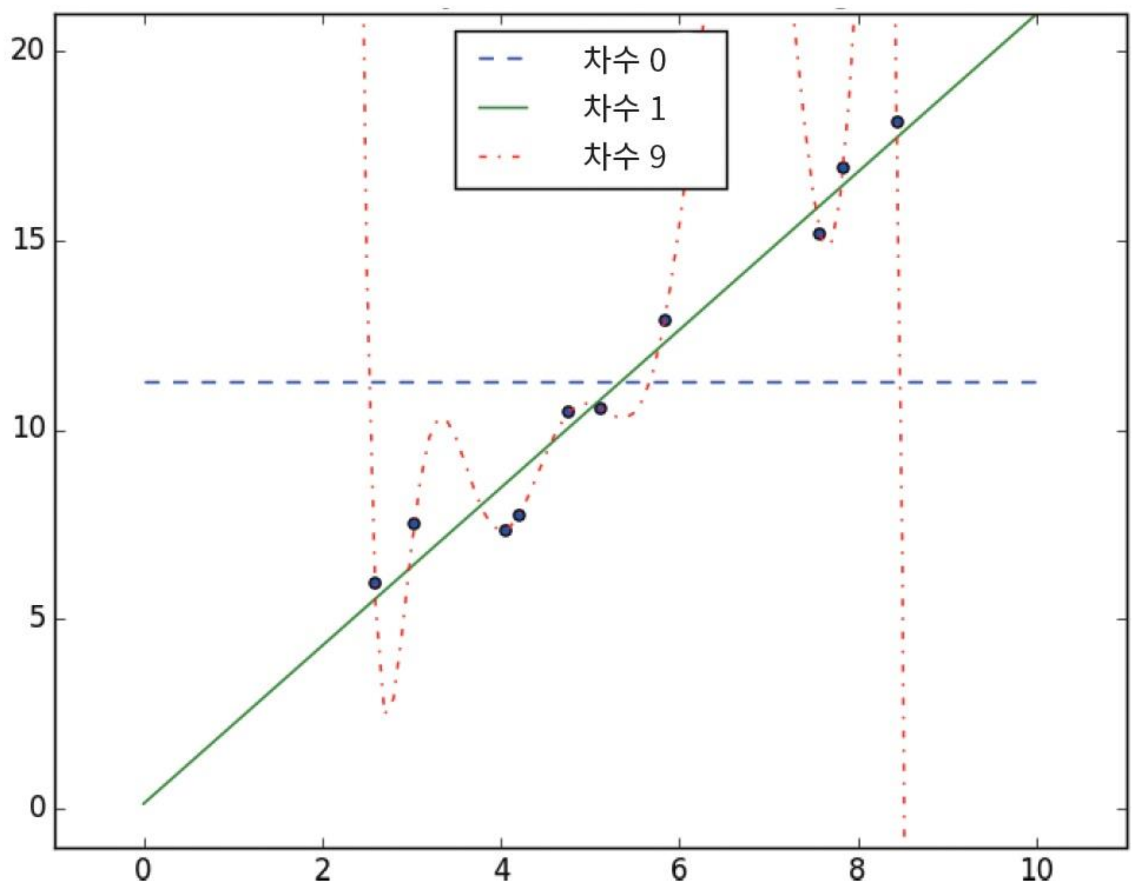
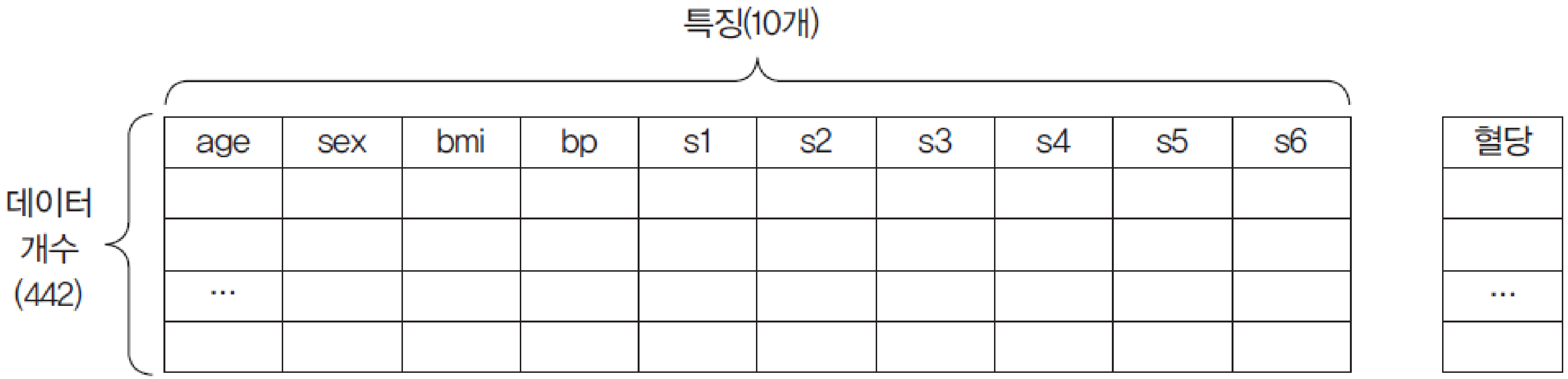


그림 10-5 회귀에서 과잉 적합의 예

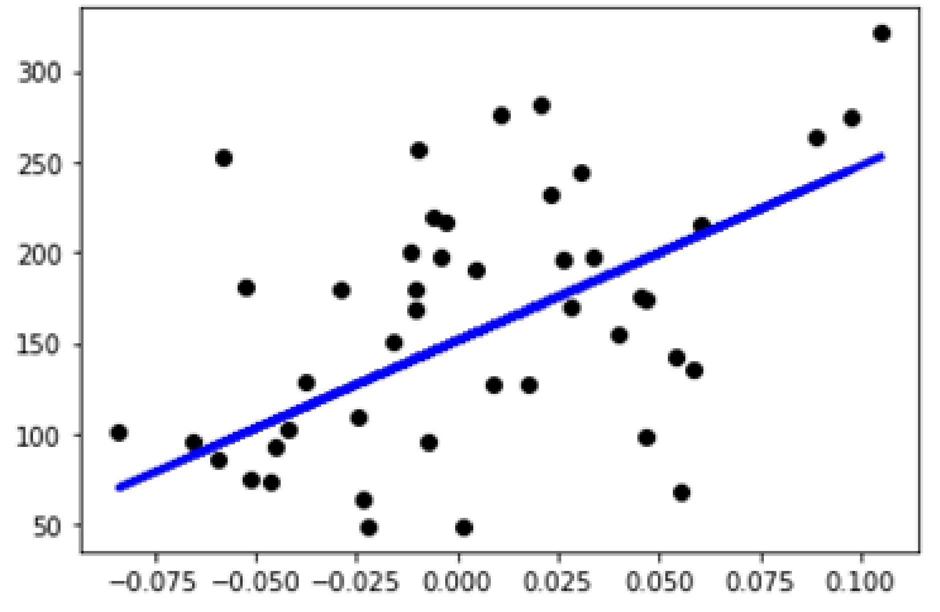
Lab: 당뇨병 예제

- **sklearn 라이브러리에는 당뇨병 환자들의 데이터가 기본적으로 포함되어 있다.**



선형 회귀 예제

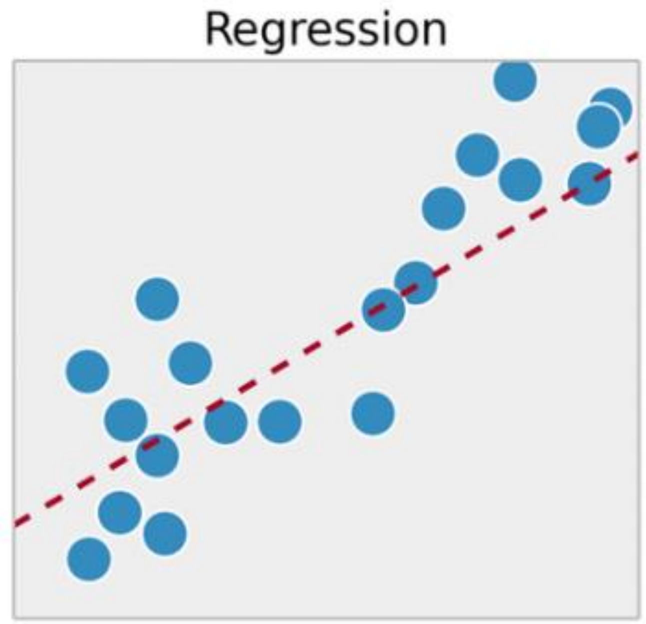
```
import matplotlib.pylab as plt
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn import datasets

# 당뇨병 데이터 세트를 적재한다.
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)

# 하나의 특징(BMI)만 추려내서 2차원 배열로 만든다. BMI 특징의 인덱스가 2이다.
diabetes_X_new = diabetes_X[:, np.newaxis, 2]

# 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_X_new, diabetes_y, test_size=0.1, random_state=0)
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.xlim([-0.13,0.15])
plt.scatter(X_test, y_test,  color='black')
plt.plot(X_test, y_pred, color='blue', linewidth=3)
```

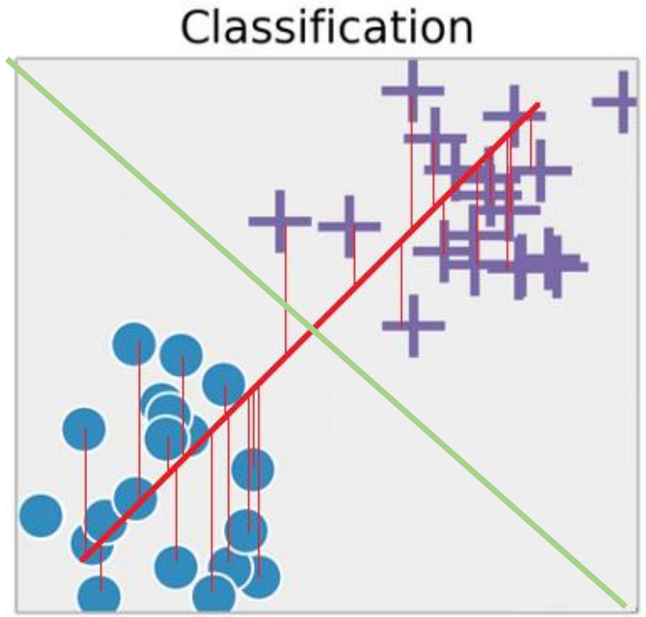


Logistic Regression

Linear Regression vs Logistic regression

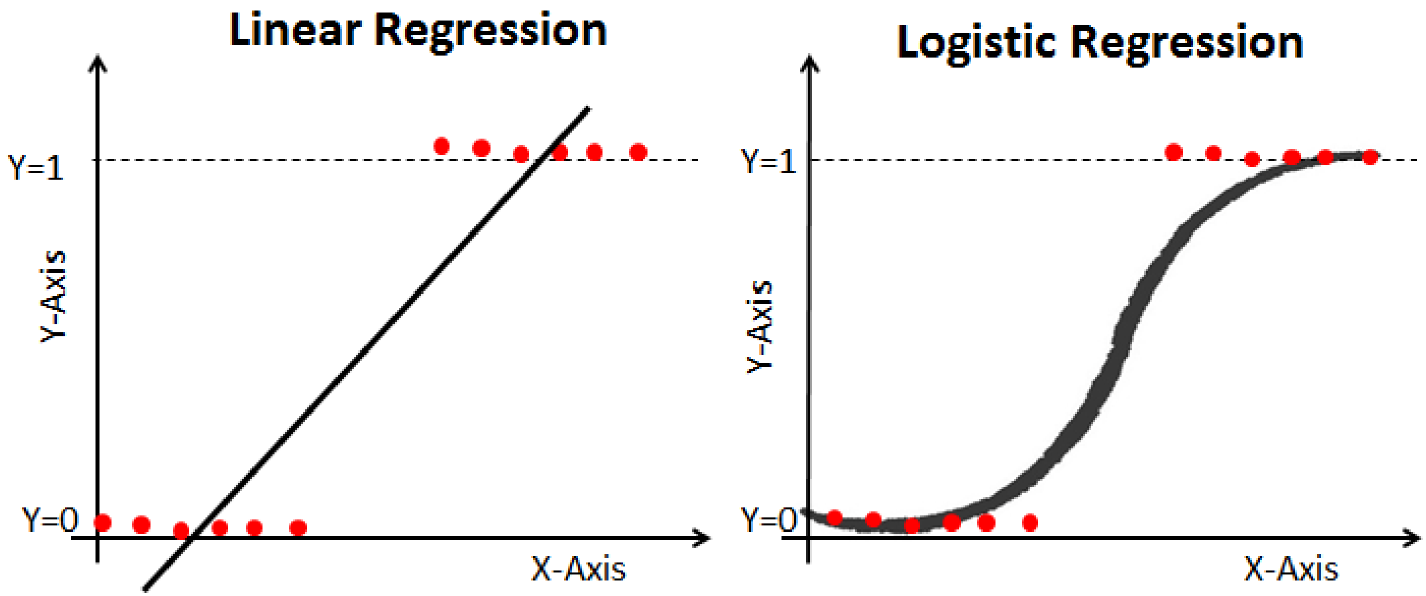
- 선형회귀의 경우는 MSE(Mean Squared Error)라는 Loss 함수를 사용했습니다. MSE는 직선과 데이터의 차이를 제곱한 값의 평균, 즉 분산을 최소화시키는 방향으로 학습
- 분류는??
  - 둘 사이를 가로 질러야…





Logistic regression

- 로지스틱회귀의 분류모델은 이진분류 모델로 클래스가 딱 2개 있을 때만 사용 가능
- 선형 회귀 다음으로 간단한 분류, 회귀 알고리즘
- 데이터 샘플에 맞는 최적의 로지스틱 함수를 구하고 이를 통해 (데이터 특성으로) 예측값을 추출하는 알고리즘

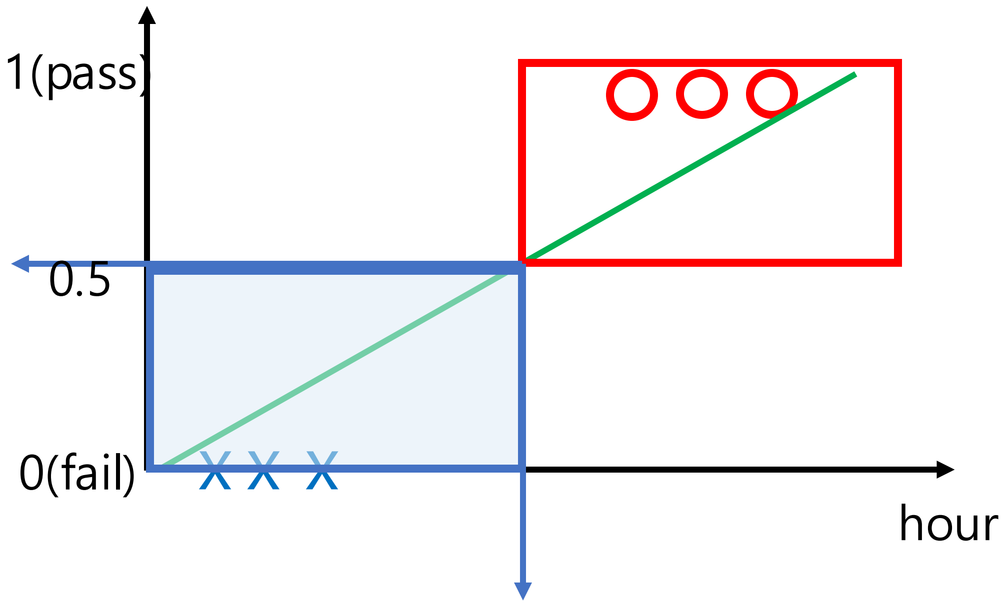


Logistic regression

- 선형회귀모델 => 연속성
- 선형회귀의 출력값이 전체의 절반을 넘었으면 그냥 반올림해서 출력을 1로, 절반을 못넘겼으면 출력을 0으로 생각해서 분류

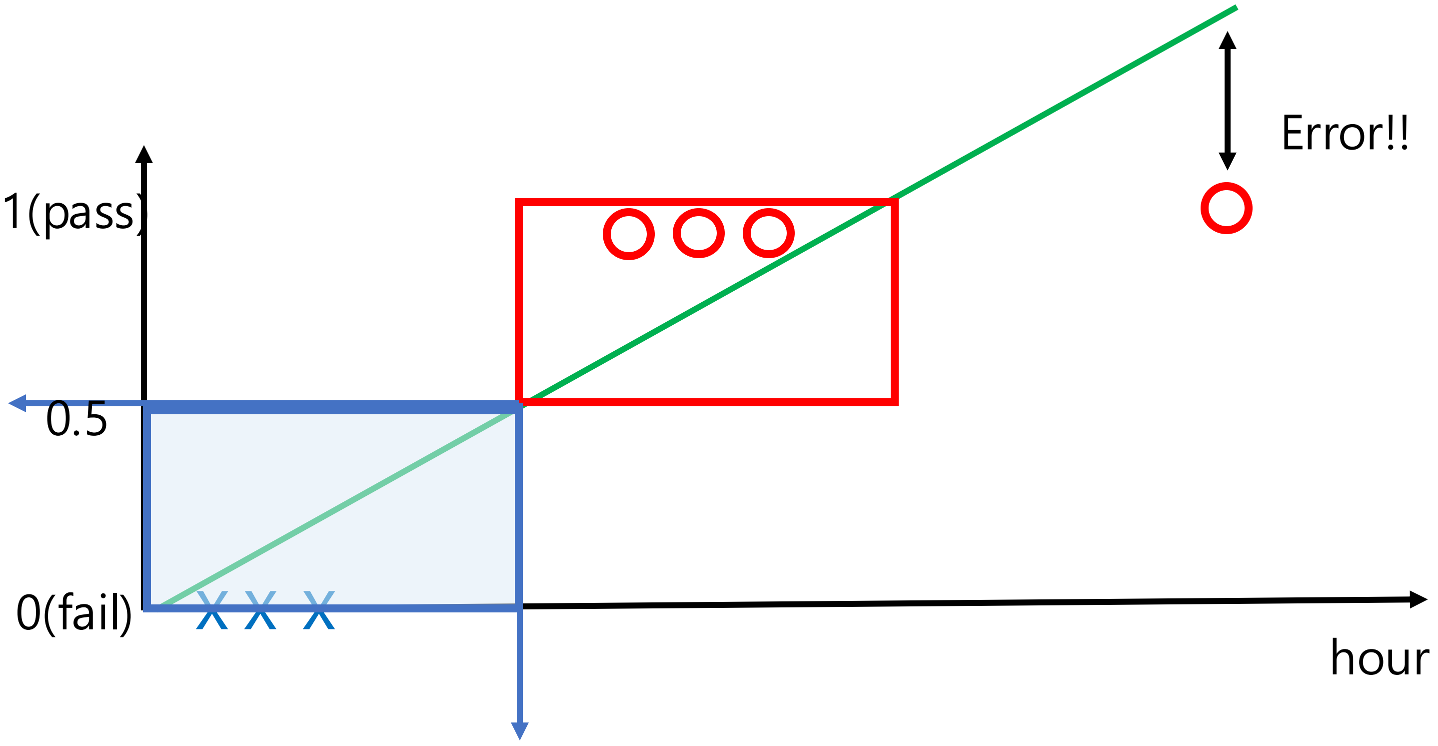
Logistic regression

- 직선의 중간쯤을 넘어가면 불합격
- 직선의 중간쯤을 넘어가면 합격



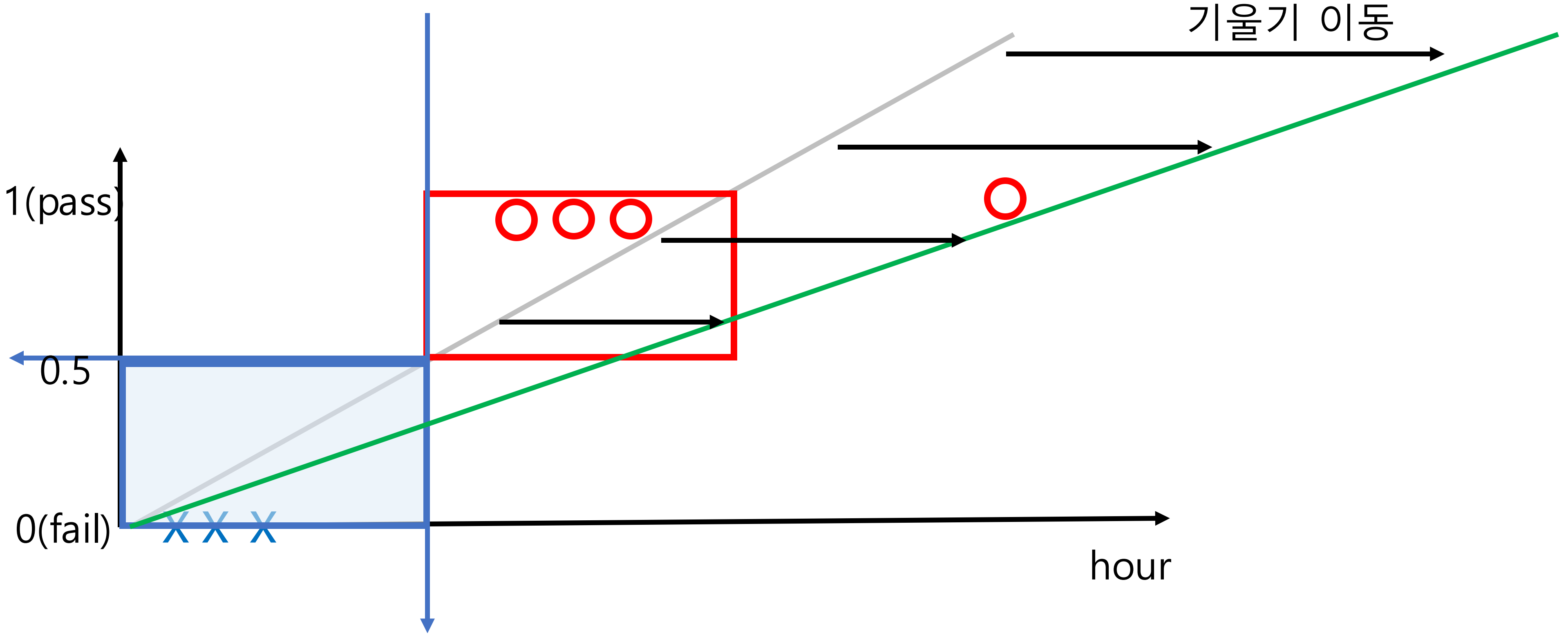
Logistic regression ?

새로운 데이터의 MSE 에러



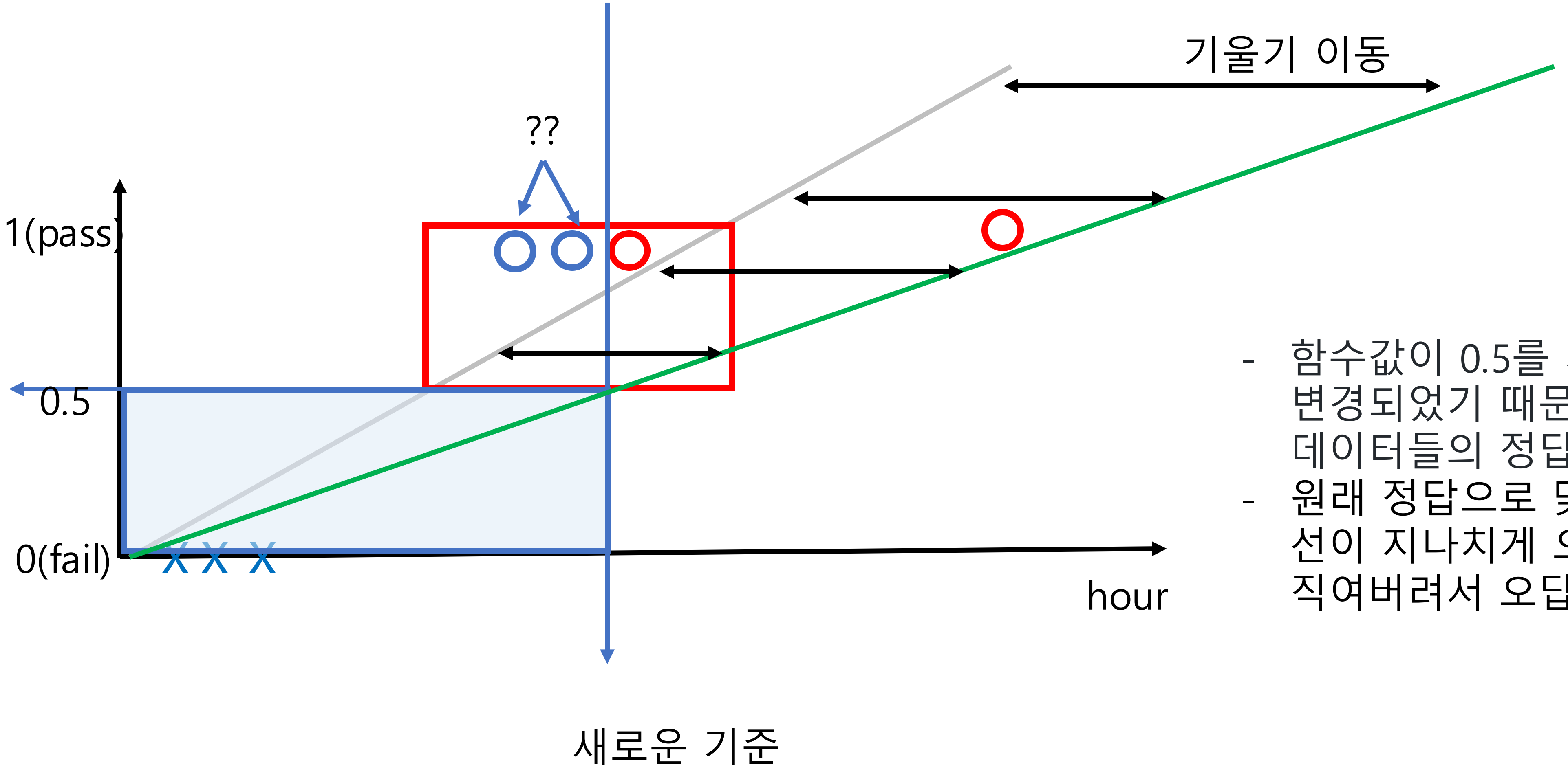
Logistic regression ?

데이터(y)와 직선(^y)간의 평균을 최소화 하기 위해 직선의 기울기가 움직임



Logistic regression ?

기울기 y = wx +b 에서 w의 변화에 따른 0.5 의 기준 변화

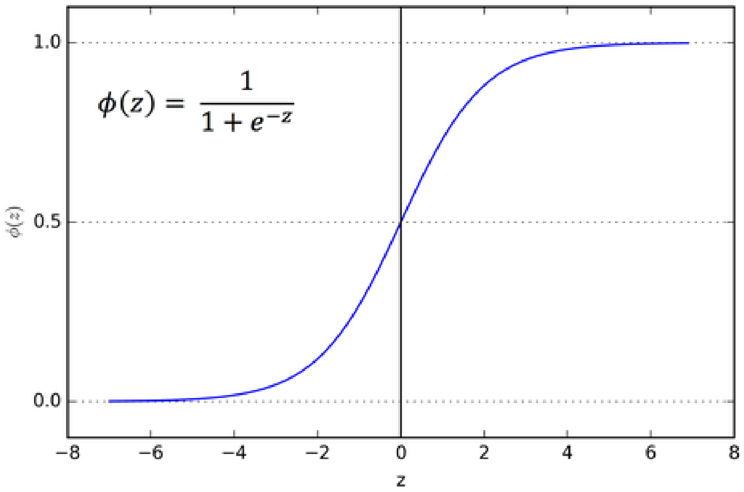


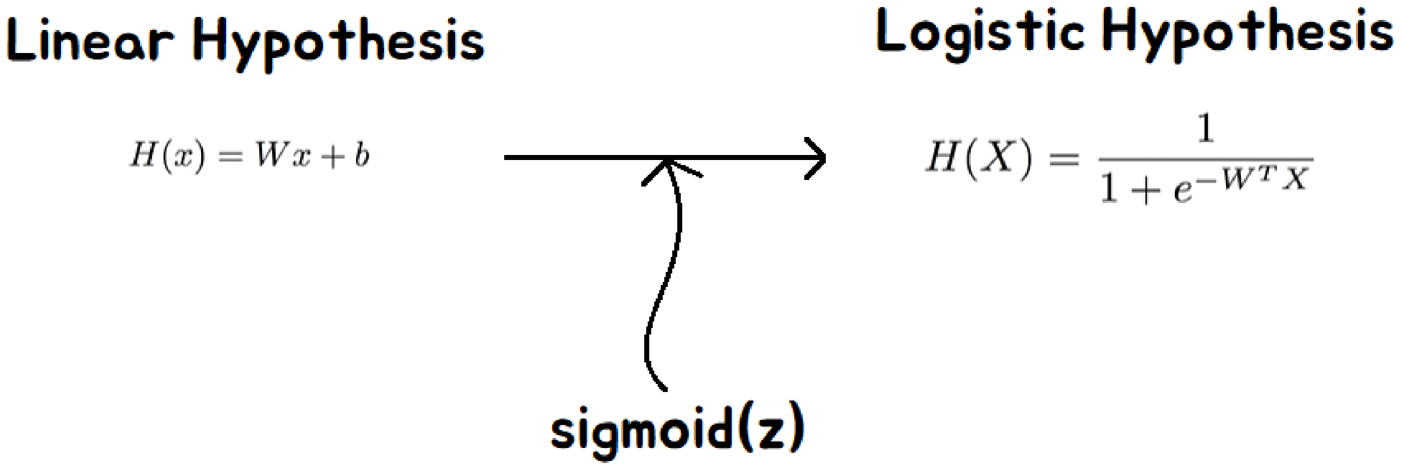
- 함수값이 0.5를 지나는 구간이 변경되었기 때문에 이에 다른 데이터들의 정답 여부가 바뀜!!!
- 원래 정답으로 맞췄지만 기준선이 지나치게 오른쪽으로 움직여버려서 오답

시그모이드 함수 (Sigmoid)

Logistic regression 에서는 Linear Regression의 결과값을 인풋값으로 사용

- 시그모이드 함수는 곧 베이즈 정리
- 기존에 사용했던 선형회귀 모델의 출력을 그대로 시그모이드 함수의 입력으로 넣으면 0 혹은 1의 값을 출력하게 되고, 이 값으로 예측을 수행하면 됩니다. 기존의 출력은 y=wx+b였습니다. 이 출력을 시그모이드 함수 sig $\frac{1}{1+e^{-(wx+b)}}$ 입력으로 넣으면 y = 가 되고, 이 값은 0 또는 1에 가까운 분류값이

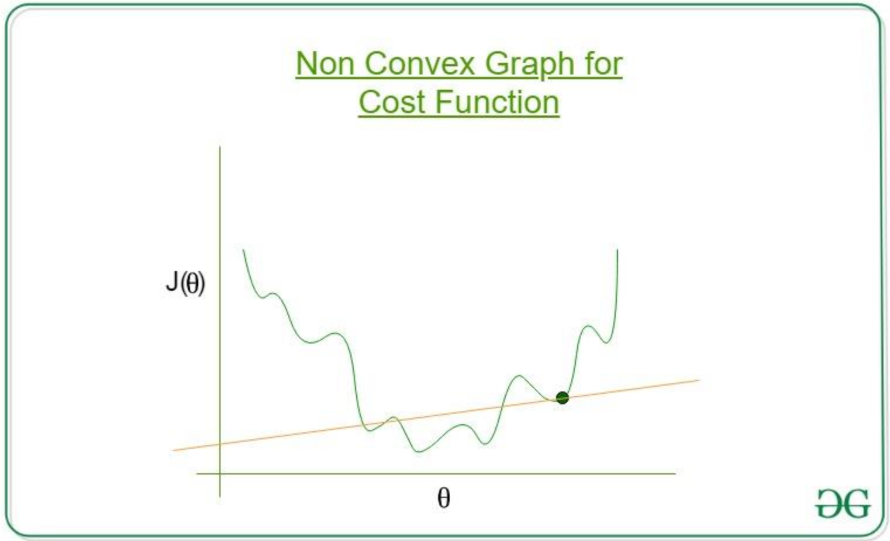


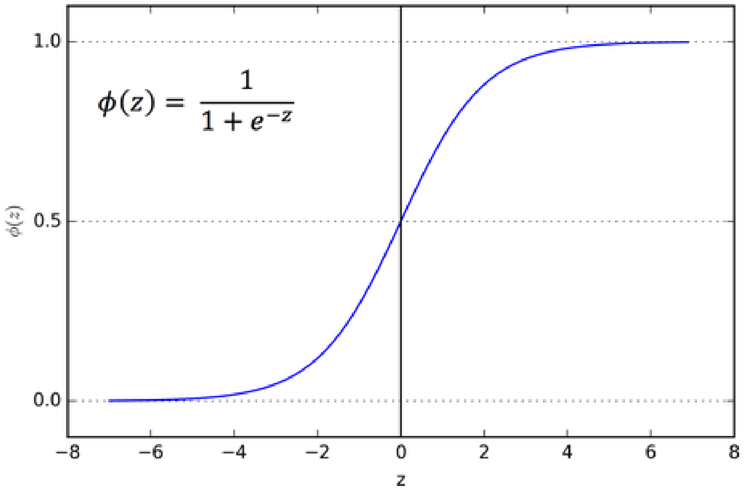


시그모이드 함수 (Sigmoid)

어떻게 가장 좋은 가중치 W 값을 구할 것인가?

- Cost function을 최소화
- Linear regression은MSE 로 최소값 가능

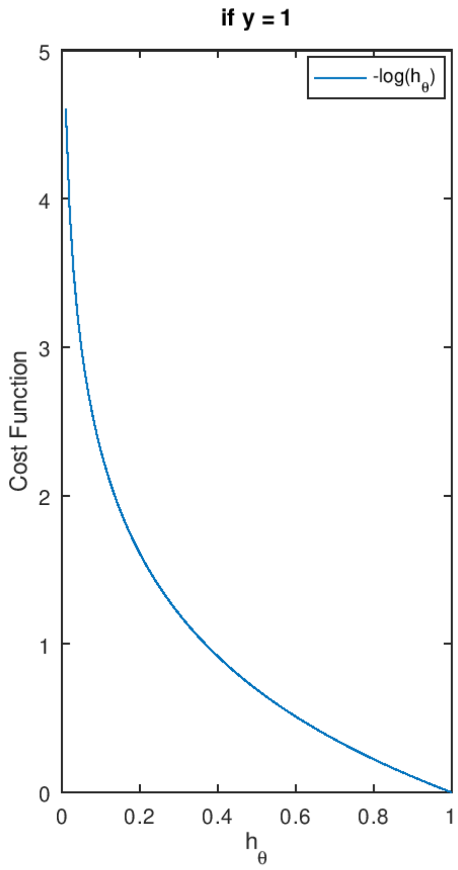


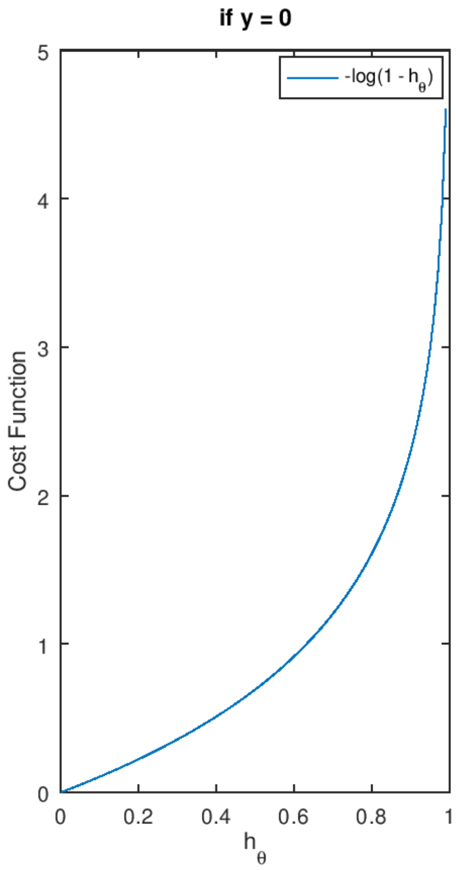


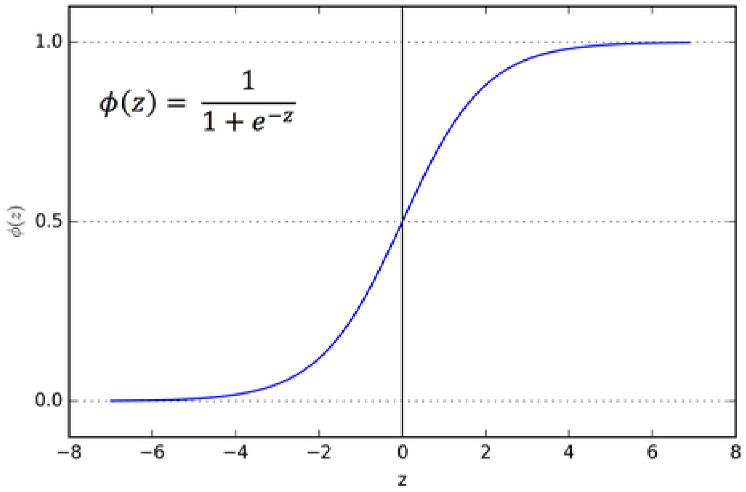
- 하지만, Logistic regression은 불가! ➔ Log loss (Cross entropy 사용)

시그모이드 함수 (Sigmoid)

Cross entropy







Cross Entropy Loss 함수

- 로지스틱 모델의 예측분포는 sigmoid(wx+b)이고,

$$\begin{cases} -log(sigmoid(wx+b)), & \text{if y} = 1 \\ -log(1-sigmoid(wx+b)), & \text{if y} = 0 \end{cases}$$ undefined

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)})$$

$$H_\theta\big(X(i)\big) = wxi + b$$

$$J(\theta) = \frac{1}{m}[\sum_{i=1}^{m} -y^{(i)}log(h_\theta(x^{(i)})) + (1-y^{(i)})log(1-h_\theta(x^{(i)}))]$$

$$m = number\ of\ samples$$

y=1 $$J(\theta) = \frac{1}{m}[\sum_{i=1}^{m} -y^{(i)}log(h_\theta(x^{(i)})) + (1-\cancel{y^{(i)})log}(1-h_\theta(x^{(i)}))]$$

y=0 $$J(\theta) = \frac{1}{m}[\sum_{i=1}^{m} -y^{(i)}\cancel{log(h}_\theta(x^{(i)})) + (1-y^{(i)})log(1-h_\theta(x^{(i)}))]$$

로지스틱 회귀

로지스틱 회귀 장점

- CrossEntropy 로스함수를 적용한 알고리즘으로, CrossEntropy 로스함수를 가장 처음으로 이해하기 좋은 알고리즘입니다.
- 선형적인 문제를 간단하게 풀 때 매우 효과적이기 때문에 전세계적으로 가장 사랑받는 머신러닝 알고리즘입니다.

로지스틱 회귀 단점

- 이진분류만 가능합니다. (클래스가 딱 2개 있을 때 만 사용가능)
- 복잡한 비선형 문제를 해결하는데 큰 어려움을 겪습니다.
- 다중공선성과 같은 문제를 자체적으로 해결하지 못합니다.

예제

유방암 분류

```python
import pandas as pd

# 0 : 양성, 1 : 악성
from sklearn import datasets
from sklearn.linear_model import LogisticRegression
cancer_ds = datasets.load_breast_cancer()

clf = LogisticRegression(multi_class = 'ovr', solver='liblinear') #one-vs-rest (OvR)
clf.fit(cancer_ds.data , cancer_ds.target)
clf.predict(cancer_ds.data)- cancer_ds.target

clf.score(cancer_ds.data, cancer_ds.target)
```

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression

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 제곱한 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