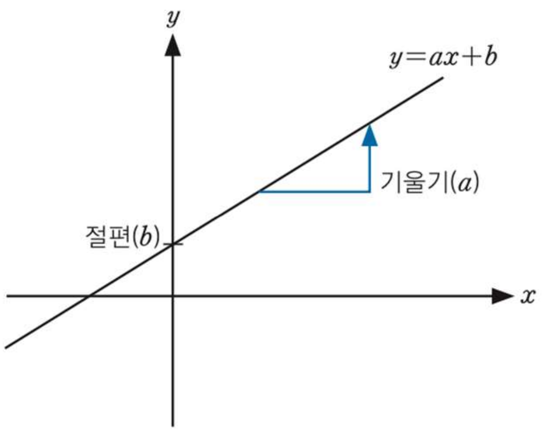
03 머신러닝의 기초를 다집니다
- 수치 예측

03-1 선형 회귀에 대해 알아보고 데이터를 준비합니다

1차 함수로 이해하는 선형 회귀

$$y = ax + b$$



선형 회귀는 기울기와 절편을 찾아줍니다

학교에서 배울 때는 기울기와 절편보다 x, y 값에 관심을 기울입니다.

1차 함수 문제 기울기가 7이고 절편이 4인 1차 함수 $y=7x+4$가 있습니다. x가 10이면 y는 얼마인가요?

① 74
② 72
③ 71

머신러닝은 x, y가 주어질 때 기울기와 절편을 구합니다.

선형 회귀 문제 x가 3일 때 y는 25, x가 4일 때 y는 32, x가 5일 때 y는 39라면 기울기와 절편의 값으로 적절한 것은 무엇인가요?

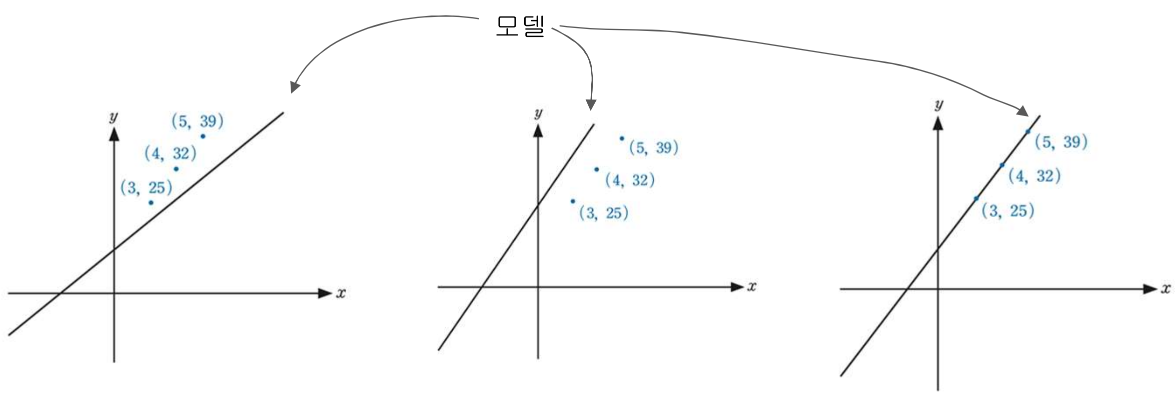
① 기울기는 6, 절편은 4
② 기울기는 7, 절편은 5
③ 기울기는 7, 절편은 4

입력 타깃

$$1.5\times x+0.1=y$$

가중치 절편

그래프를 통해 선형 회귀의 문제 해결 과정을 이해합니다



최적의 모델
x = 6일 때는? (46)

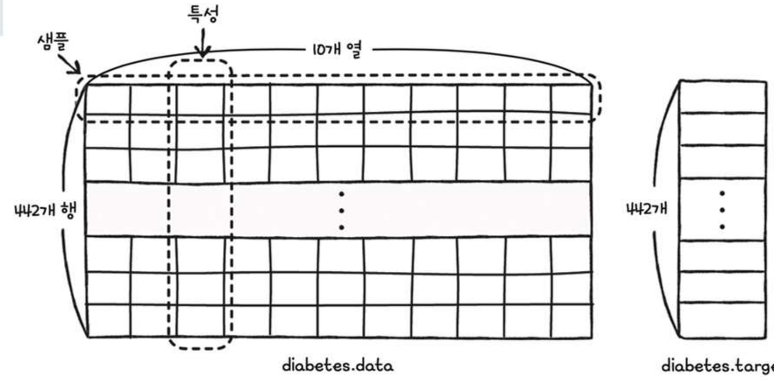
문제 해결을 위해 당뇨병 환자의 데이터 준비하기

```
from sklearn.datasets import load_diabetes
diabetes = load_diabetes( )
```

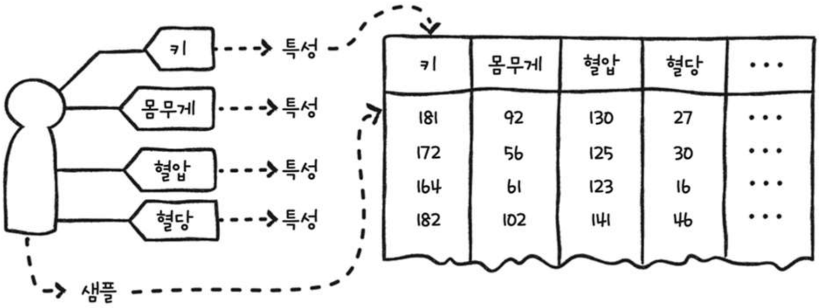
```
print(diabetes.data.shape, diabetes.target.shape)
(442, 10) (442,)
```

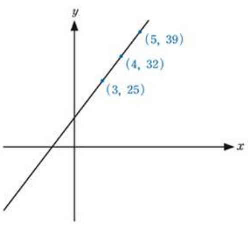
넘파이 배열

Bunch 클래스



샘플과 특성





여기에서는 특성 하나만 사용했습니다.

- 특성 2개를 사용하면?
- 3개를 사용하면?

입력 데이터와 타깃 데이터 자세히 보기

```
diabetes.data[0:3]
array([[ 0.03807591,  0.05068012,  0.06169621,  0.02187235, -0.0442235 ,
        -0.03482076, -0.04340085, -0.00259226,  0.01990842, -0.01764613],
       [-0.00188202, -0.04464164, -0.05147406, -0.02632783, -0.00844872,
        -0.01916334,  0.07441156, -0.03949338, -0.06832974, -0.09220405],
       [ 0.08529891,  0.05068012,  0.04445121, -0.00567061, -0.04559945,
        -0.03419447, -0.03235593, -0.00259226,  0.00286377, -0.02593034]])
```

네 번째 특성의 값입니다.

첫 번째 샘플입니다.

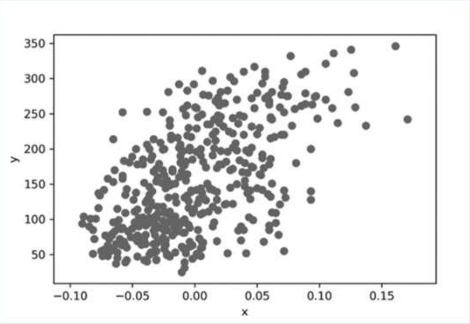
```
diabetes.target[:3]
array([151., 75., 141.])
```

도메인 지식이 필요합니다

실전에서는 지도 학습 데이터를 만드
는 데 많은 노력을 기울입니다

당뇨병 환자 데이터 시각화하기

```
import matplotlib.pyplot as plt
plt.scatter(diabetes.data[:, 2], diabetes.target)
plt.xlabel('x')
plt.ylabel('y')
plt.show( )
```

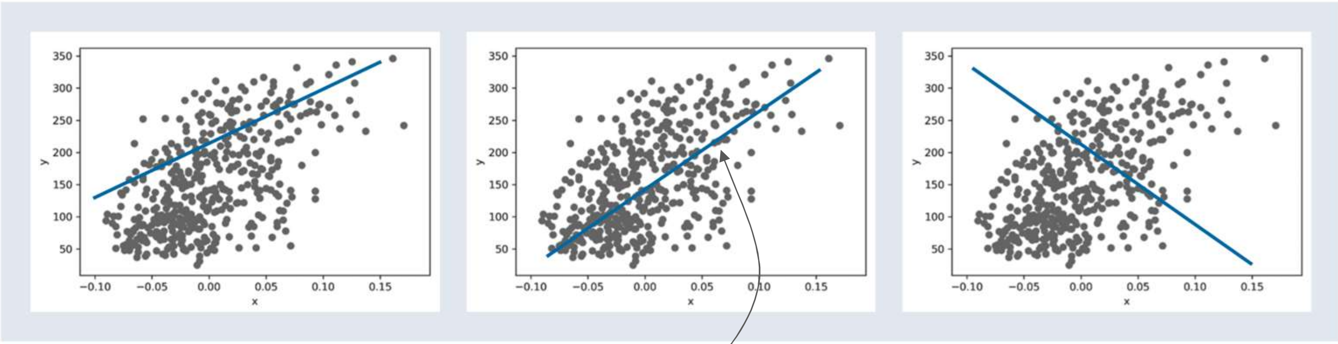


이후 코드를 간단하게
쓰기 위해

```
x = diabetes.data[:, 2]
y = diabetes.target
```

03-2 경사 하강법으로 학습하는 방법을 알아봅니다

어떤 직선이 가장 잘 표현하고 있나요?



경사 하강법으로 찾기!

회귀 문제를 풀기 위해 경사 하강법 외에 다른 방법은 없나요?

52

타깃과 예측값

타깃 기울기

$$y = ax + b$$

절편

$$\hat{y} = wx + b$$

예측값

가중치

53

훈련 데이터에 잘 맞는 w와 b를 찾는 방법

① 무작위로 **w**와 **b**를 정합니다(무작위로 모델 만들기).
② x에서 샘플 하나를 선택하여 $\hat{y}$을 계산합니다(무작위로 모델 예측하기).
③ $\hat{y}$과 선택한 샘플의 진짜 y를 비교합니다(예측한 값과 진짜 정답 비교하기, 틀릴 확률 99%).
④ $\hat{y}$이 y와 더 가까워지도록 **w**, **b**를 조정합니다(모델 조정하기).
⑤ 모든 샘플을 처리할 때까지 다시 ②~④ 항목을 반복합니다.

54

실제로 훈련 데이터에 맞는 w와 b 찾아보기

```
w = 1.0
b = 1.0
```

임의의 값으로 시작

```
y_hat = x[0] * w + b
print(y_hat)
1.0616962065186886
```

첫 번째 샘플에 대한 예측 만들기

어떻게 이 차이를 좁힐 수 있을까요?

```
print(y[0])
151.0
```

첫 번째 샘플의 실제 타깃

55

w 값을 조절해 예측값을 바꾸어 보죠

w를 0.1만큼 증가시켜 봅니다

```
w_inc = w + 0.1
y_hat_inc = x[0] * w_inc + b
print(y_hat_inc)
1.0678658271705574
```

이전보다 증가했습니다.

```
y_hat = x[0] * w + b
print(y_hat)
1.0616962065186886
```

타깃에 조금 더 가까워졌습니다

```
print(y[0])
151.0
```

w를 0.1만큼 증가시킨 것은 올바른 결정이었네요!

56

그럼 얼마만큼 증가했나요?

```
w_rate = (y_hat_inc - y_hat) / (w_inc - w)
print(w_rate)
0.061696206518688734
```

변화율

$$w\_rate = \frac{y\_hat\_inc - y\_hat}{w\_inc - w} = \frac{(x[0] * w\_inc + b) - (x[0] * w + b)}{w\_inc - w}$$

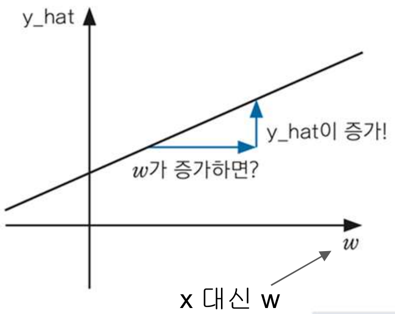
$$= \frac{(x[0] * (w + 0.1) - w)}{(w + 0.1) - w} = x[0]$$

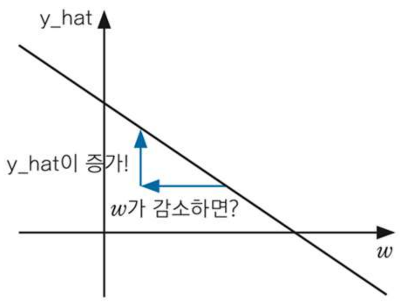
변화율은 x[0] 자체입니다

변화율을 보고 w를 어떻게 바꾸어야 할지 알 수 있나요?

57

변화율 부호에 따라 가중치를 업데이트하는 방법






```
w_new = w + w_rate
print(w_new)
1.0616962065186888
```

변화율을 더하면 예측값이 증가!

58

변화율로 절편 업데이트하기

```
b_inc = b + 0.1
y_hat_inc = x[0] * w + b_inc
print(y_hat_inc)
1.1616962065186887

b_rate = (y_hat_inc - y_hat) / (b_inc - b)
print(b_rate)
1.0
```

$$b\_rate = \frac{y\_hat\_inc - y\_hat}{b\_inc - b} = \frac{(x[0]*w + b\_inc) - (x[0]*w + b)}{b\_inc - b}$$

$$= \frac{(b+0.1) - b}{(b+0.1) - b} = 1$$

```
b_new = b + 1
print(b_new)
2.0
```

59

이 방식에 어떤 문제점이 있을까요?

- y_hat이 y에 한참 미치지 못 하는 값인 경우, w와 b를 더 큰 폭으로 수정할 수 없습니다(앞에서 변화율 만큼 수정을 했지만 특별한 기준을 정하기가 어렵습니다).
- y_hat이 y보다 커지면 y_hat을 감소시키지 못 합니다.

y_hat 과 y 의 차이가 크면 w와 b를 그에 비례해서 바꿔야 됩니다. → 빠르게 솔루션에 수렴

y_hat이 y 보다 크면 w와 b를 감소시켜야 합니다. → y_hat과 y 값에 능동적으로 대처

60

오차 역전파로 가중치와 절편을 업데이트합니다

오차와 변화율을 곱하여 가중치를 업데이트합니다

```
err = y[0] - y_hat
w_new = w + w_rate * err
b_new = b + 1 * err
print(w_new, b_new)
10.250624555904514 150.9383037934813
```

가중치와 절편이 큰 폭으로 바뀌었습니다

```
w_new = w + w_rate
print(w_new)
1.0616962065186888
```

```
b_new = b + 1
print(b_new)
2.0
```

61

두 번째 샘플을 사용하여 w와 b를 계산합니다

```
y_hat = x[1] * w_new + b_new
err = y[1] - y_hat
w_rate = x[1]
w_new = w_new + w_rate * err
b_new = b_new + 1 * err
print(w_new, b_new)
14.132317616381767 75.52764127612664
```

두 번째 샘플의 변화율은 샘플 값 그 자체입니다

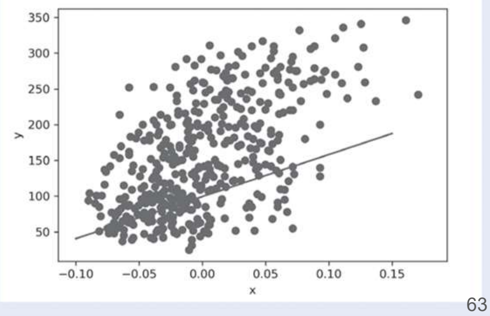
이런 식으로 모든 샘플에 대해 처리해 볼까요?

62

전체 샘플을 반복하여 가중치와 절편을 조정하기

```
for x_i, y_i in zip(x, y):
    y_hat = x_i * w + b
    err = y_i - y_hat
    w_rate = x_i
    w = w + w_rate * err
    b = b + 1 * err
print(w, b)
587.8654539985689 99.40935564531424
```

```
plt.scatter(x, y)
pt1 = (-0.1, -0.1 * w + b)
pt2 = (0.15, 0.15 * w + b)
plt.plot([pt1[0], pt2[0]], [pt1[1], pt2[1]])
plt.xlabel('x')
plt.ylabel('y')
plt.show( )
```



63

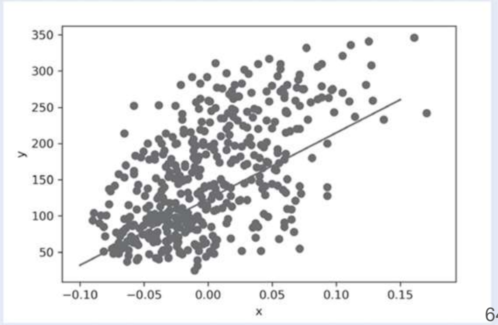
여러 에포크를 반복하기

```
for i in range(1, 100):
    for x_i, y_i in zip(x, y):
        y_hat = x_i * w + b
        err = y_i - y_hat
        w_rate = x_i
        w = w + w_rate * err
        b = b + 1 * err
print(w, b)
913.5973364345905 123.39414383177204
```

경사 하강법으로 찾은 선형 회귀 모델

$$\hat{y} = 913.6x + 123.4$$

```
plt.scatter(x, y)
pt1 = (-0.1, -0.1 * w + b)
pt2 = (0.15, 0.15 * w + b)
plt.plot([pt1[0], pt2[0]], [pt1[1], pt2[1]])
plt.xlabel('x')
plt.ylabel('y')
plt.show( )
```



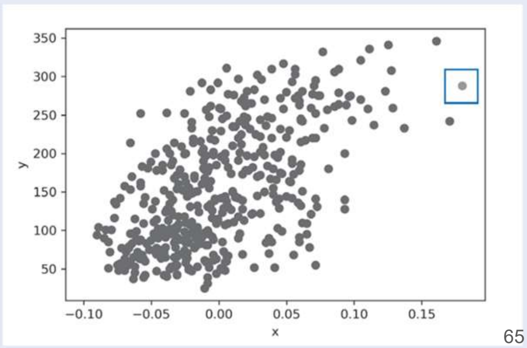
64

모델로 예측하기

$$\hat{y} = 913.6x + 123.4$$

```
x_new = 0.18
y_pred = x_new * w + b
print(y_pred)
287.8416643899983
```

```
plt.scatter(x, y)
plt.scatter(x_new, y_pred)
plt.xlabel('x')
plt.ylabel('y')
plt.show( )
```



65

지금까지 실습 내용을 정리해 보죠

지금까지는 모델을 이렇게 만들었습니다

1. w와 b를 임의의 값(1.0, 1.0)으로 초기화하고 훈련 데이터의 샘플을 하나씩 대입하여 y와 $\hat{y}$의 오차를 구합니다.
2. 1에서 구한 오차를 w와 b의 변화율에 곱하고 이 값을 이용하여 w와 b를 업데이트합니다.
3. 만약 $\hat{y}$이 y보다 커지면 오차는 음수가 되어 자동으로 w와 b가 줄어드는 방향으로 업데이트됩니다.
4. 반대로 $\hat{y}$이 y보다 작으면 오차는 양수가 되고 w와 b는 더 커지도록 업데이트됩니다.

66

03-3 손실 함수와 경사 하강법의 관계를 알아봅니다

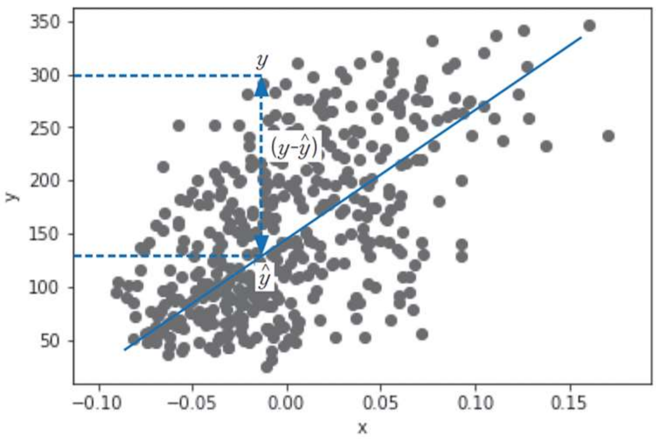
- 손실 함수는 예측한 값과 실제 타깃값의 차이를 측정합니다
- 손실 함수의 차이를 줄이는 방법으로 경사 하강법을 사용합니다
- 대표적인 회귀, 분류 등에는 널리 사용하는 손실 함수가 있습니다
- 복잡한 다른 문제에서는 자신만의 손실 함수를 정의하여 사용하기도 합니다

67

회귀의 손실 함수

제곱 오차(squared error)

$$SE=(y-\hat{y})^2$$



68

손실 함수의 기울기를 찾기 위해 미분합니다

가중치에 대하여 제곱 오차 미분하기

$$\frac{\partial SE}{\partial w}=\frac{\partial}{\partial w}(y-\hat{y})^2=2(y-\hat{y})(-\frac{\partial}{\partial w}\hat{y})=2(y-\hat{y})(-x)=-2(y-\hat{y})x$$

$$\frac{\partial}{\partial w}(w\times x+b)=x$$

$SE=\frac{1}{2}(y-\hat{y})^2$ 라면 $\frac{\partial SE}{\partial w}=-(y-\hat{y})x$

69

미분 결과를 가중치에서 빼면 손실 함수의 낮은 쪽으로 이동

$$w = w - \frac{\partial SE}{\partial w} = w + (y - \hat{y})x$$

앞서 직관으로 계산한 오차 역전파가 제곱 오차를 미분한 것과 결과가 같군요!

```
y_hat = x_i * w + b
err = y_i - y_hat
w_rate = x_i
w = w + w_rate * err
```

70

절편에 대해 미분하고 업데이트하기

$$\frac{\partial SE}{\partial b} = \frac{\partial}{\partial b}\frac{1}{2}(y - \hat{y})^2 = (y - \hat{y})(-\frac{\partial}{\partial b}\hat{y}) = (y - \hat{y})(-1) = -(y - \hat{y})1$$

$$\frac{\partial}{\partial w}(w \times x + b) = x$$

그레이디언트(gradient)

$$b = b - \frac{\partial SE}{\partial b} = b + (y - \hat{y})$$

```
err = y_i - y_hat
b = b + 1 * err
```

71

03-4 선형 회귀를 위한 뉴런을 만듭니다

Neuron 클래스 만들기

```
class Neuron:

    def __init__(self):
    # 초기화 작업을 수행합니다.
    ...

    # 필요한 메서드를 추가합니다.
    ...
```

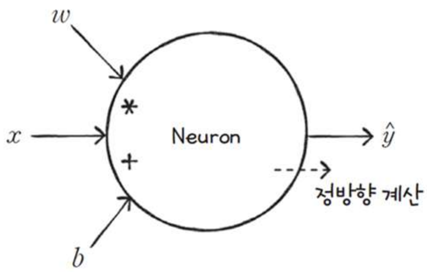
__init__() 메서드 작성하기

```
def __init__(self):
    self.w = 1.0
    self.b = 1.0
```

정방향 계산 만들기

오차를 계산하기 위 $\hat{y}=w\times x+b$ 을 먼저 구해야 합니다

```
def forpass(self, x):
    y_hat = x * self.w + self.b        # 직선 방정식을 계산합니다.
    return y_hat
```



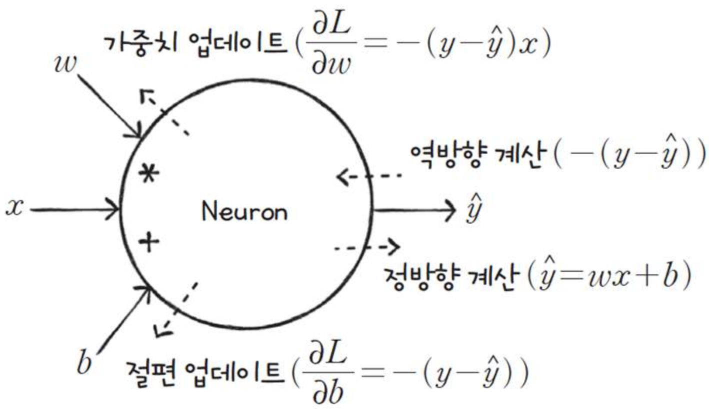
역방향 계산 만들기

손실 함수(제곱 오차 함수)

$$\frac{\partial L}{\partial w}=-(y-\hat{y})x$$

$$\frac{\partial L}{\partial b}=-(y-\hat{y})$$

```
def backprop(self, x, err):
    w_grad = x * err
    b_grad = 1 * err
    return w_grad, b_grad
```



훈련을 위한 fit() 메서드 구현

```
def fit(self, x, y, epochs=100):
    for i in range(epochs):                 # 에포크만큼 반복합니다.
        for x_i, y_i in zip(x, y):          # 모든 샘플에 대해 반복합니다.
            y_hat = self.forpass(x_i)  # 정방향 계산
            err = -(y_i - y_hat)            # 오차 계산
            w_grad, b_grad = self.backprop(x_i, err) # 역방향 계산
            self.w -= w_grad                # 가중치 업데이트
            self.b -= b_grad                # 절편 업데이트
```

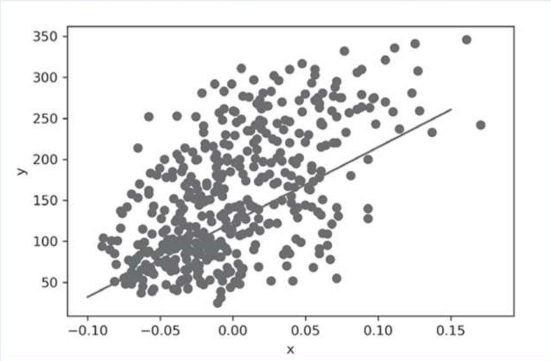
Neuron 클래스

```
class Neuron:

    def __init__(self):
        self.w = 1.0                    # 가중치를 초기화합니다.
        self.b = 1.0                    # 절편을 초기화합니다.

    def forpass(self, x):
        y_hat = x * self.w + self.b     # 직선 방정식을 계산합니다.
        return y_hat

    def backprop(self, x, err):
        w_grad = x * err                # 가중치에 대한 그레이디언트를 계산합니다.
        b_grad = 1 * err                # 절편에 대한 그레이디언트를 계산합니다.
        return w_grad, b_grad

    def fit(self, x, y, epochs=100):
        for i in range(epochs):         # 에포크만큼 반복합니다.
            for x_i, y_i in zip(x, y):  # 모든 샘플에 대해 반복합니다.
                y_hat = self.forpass(x_i) # 정방향 계산
                err = -(y_i - y_hat)    # 오차 계산
                w_grad, b_grad = self.backprop(x_i, err) # 역방향 계산
                self.w -= w_grad        # 가중치 업데이트
                self.b -= b_grad        # 절편 업데이트
```

뉴런 훈련

```
neuron = Neuron( )
neuron.fit(x, y)
```

```
plt.scatter(x, y)
pt1 = (-0.1, -0.1 * neuron.w + neuron.b)
pt2 = (0.15, 0.15 * neuron.w + neuron.b)
plt.plot([pt1[0], pt2[0]], [pt1[1], pt2[1]])
plt.xlabel('x')
plt.ylabel('y')
plt.show( )
```



04 분류하는 뉴런을 만듭니다

- 이진 분류(binary classification)

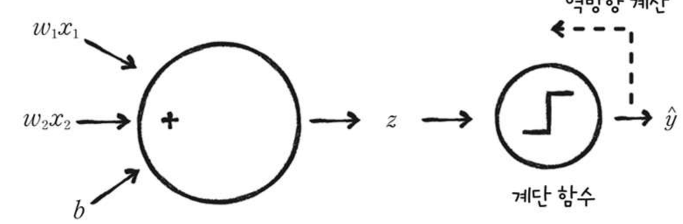
04-1 초기 인공지능 알고리즘과 로지스틱 회귀

이진 분류는 True(1) / False(0 or -1)로 구분하는 문제

퍼셉트론(Perceptron): 1957년 프랑크 로젠블라트가 발표.

여기에서 다층 퍼셉트론 이름이 유래됨

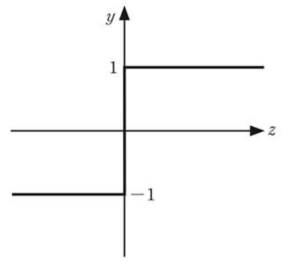
3장보다 입력이 1개 늘어남



$$w_1x_1+w_2x_2+b=z$$

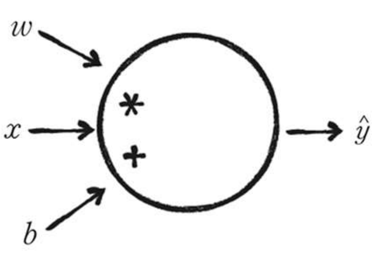
$$y=\begin{cases} 1 & (z>0) \\ -1 & \end{cases}$$

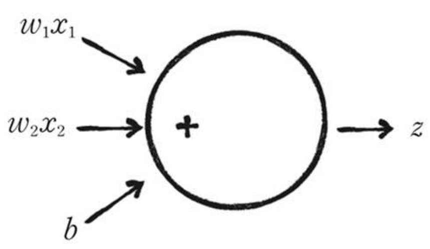
양성 클래스
(positive class)



음성 클래스
(negative class)

여러 개의 특성을 표현하는 방법





$$z=w_1x_1+w_2x_2+b$$

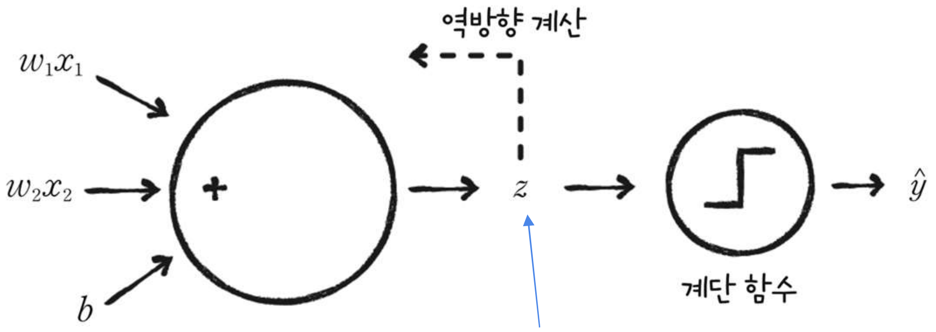
1번째 특성의 가중치와 입력

$$z=w_1x_1+w_2x_2+\cdots+w_nx_n+b$$

$$z=b+\sum_{i=1}^{n}w_ix_i$$

아달린(Adaline)에 대해 알아 봅니다

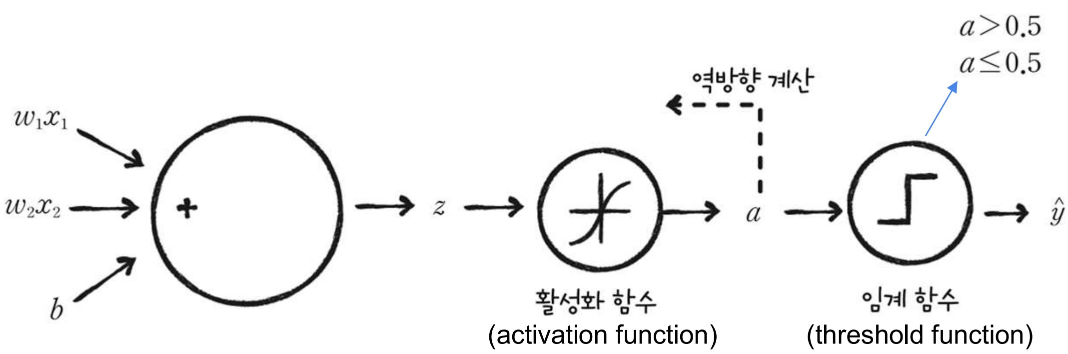
1960년 버나드 위드로우 & 테드 호프 적응형 선형 뉴런(Adaptive Linear Neuron) 발표



역방향 계산이 일어나는 위치가
퍼셉트론과 다름

로지스틱 회귀에 대해 알아봅니다

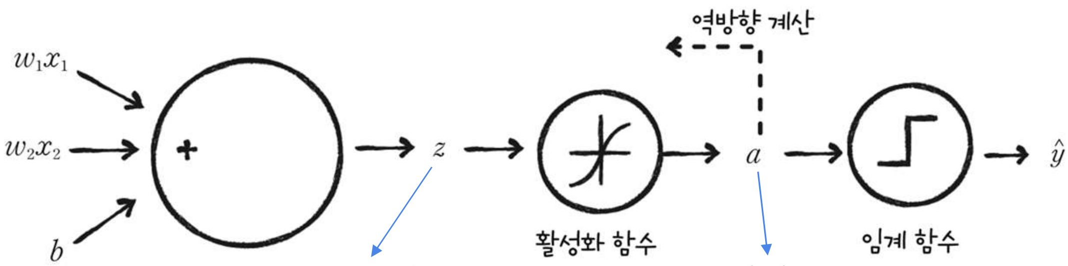
로지스틱 회귀는 분류 알고리즘입니다



시그모이드 함수
(왜 비선형 함수를 사용할까요?)

$$p=\frac{1}{1+e^{-z}}$$

04-2 시그모이드 함수로 확률을 만듭니다



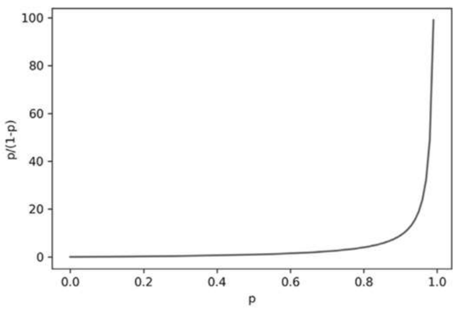
$$z=b+\sum_{i=1}^{n}w_ix_i$$

0~1
a > 0.5 : 양성 클래스
a <= 0.5 : 음성 클래스

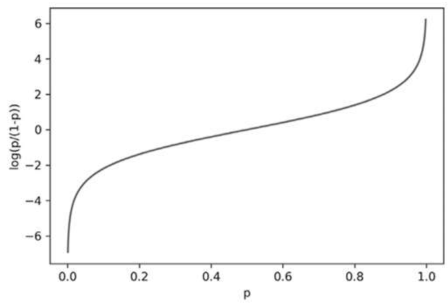
시그모이드 함수

오즈 비(odds ratio) → 로짓 함수(logit funtion) → 시그모이드 함수

$$OR(odds\ ratio)=\frac{p}{1-p}(p=\text{성공 확률})$$



$$logit(p)=log(\frac{p}{1-p})$$



로지스틱 함수

로짓 함수를 확률 p에 대해 정리한 것입니다

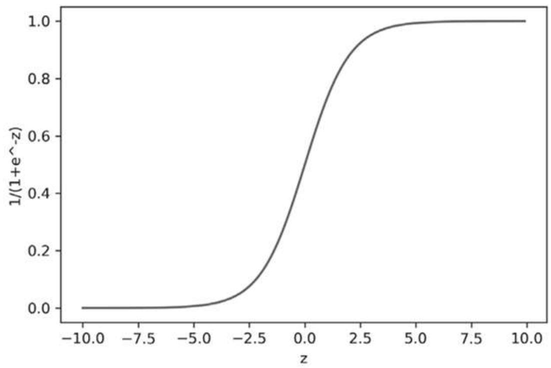
시그모이드(sigmoid) 함수라고도 부릅니다

$$log(\frac{p}{1-p})=z$$

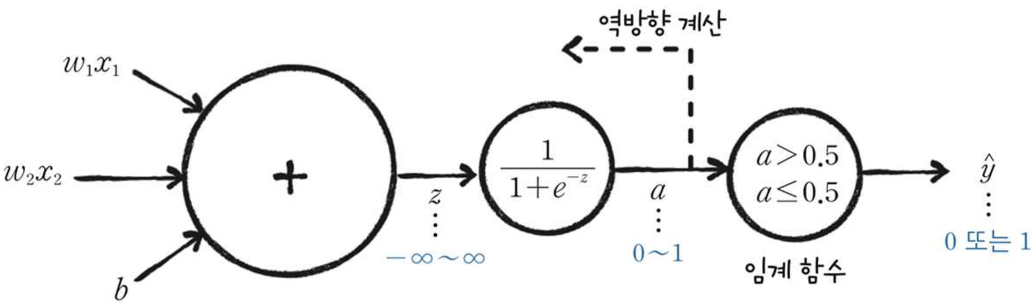
$$\frac{p}{1-p}=e^z$$

$$p(1+e^z)=e^z$$

$$p=\frac{e^z}{1+e^z}=\frac{1}{1+e^{-z}}$$



로지스틱 회귀 중간 정리하기



04-3 로지스틱 손실 함수를 경사 하강법에 적용합니다

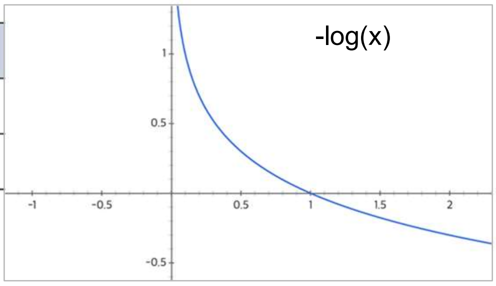
분류의 정확도는 미분 가능한 함수가 아닙니다

대신 이진 크로스 엔트로피(binary cross entropy)
또는 로지스틱(logistic) 손실 함수를 사용합니다

$$L = -(y\,log(a) + (1-y)\,log(1-a))$$

y: 타깃값
a: 활성화 함수의 출력

	L
y가 1인 경우(양성 클래스)	$-log(a)$
y가 0인 경우(음성 클래스)	$-log(1-a)$



로지스틱 손실 함수 미분하기

결과부터 보기! :)

	제곱 오차의 미분	로지스틱 손실 함수의 미분
가중치에 대한 미분	$\frac{\partial SE}{\partial w} = -(y-\hat{y})x$	$\frac{\partial}{\partial w_i} L = -(y-a)x_i$
절편에 대한 미분	$\frac{\partial SE}{\partial b} = -(y-\hat{y})1$	$\frac{\partial}{\partial b} L = -(y-a)1$

미분의 연쇄 법칙(Chain Rule)

합성 함수의 도함수를 구하는 방법

$$y=f(u),\ u=g(x) \qquad y=f(g(x)) \qquad \frac{\partial y}{\partial x}=\frac{\partial y}{\partial u}\frac{\partial u}{\partial x} \qquad \frac{\partial L}{\partial w_i}=? \qquad \frac{\partial L}{\partial b}=?$$

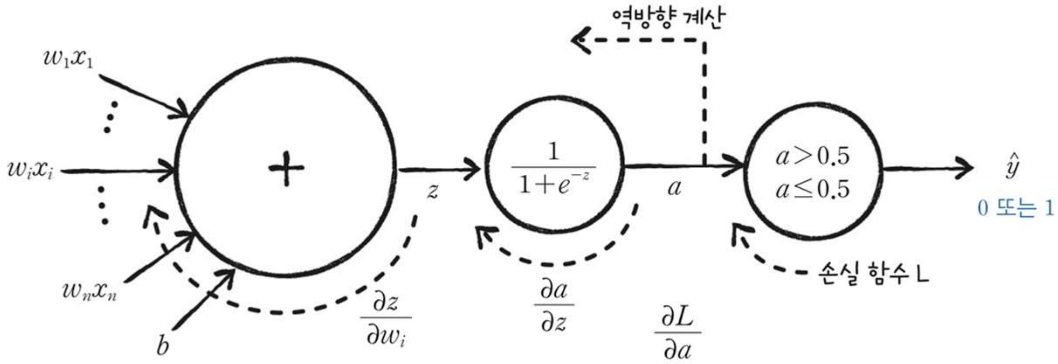
제곱 오차의 미분에 이미 적용해 보았습니다

$$SE=(y-\hat{y})^2,\ \hat{y}=w\times x+b$$

$$\frac{\partial SE}{\partial w}=\frac{\partial SE}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w}=\frac{\partial (y-\hat{y})^2}{\partial \hat{y}}\times\frac{\partial \hat{y}}{\partial w}=-2(y-\hat{y})\times x$$

연쇄 법칙을 뉴런 그림에 나타내 봅니다

$$\frac{\partial L}{\partial w_i}=\frac{\partial L}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial w_i}$$



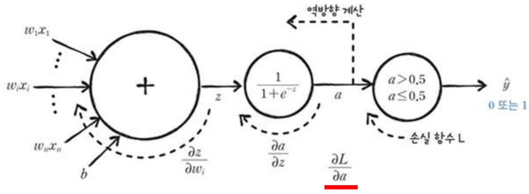
로지스틱 손실 함수를 a에 대해 미분하기

타깃값 활성화 함수의 출력

$$\frac{\partial L}{\partial a}=\frac{\partial}{\partial a}(-(ylog(a)+(1-y)log(1-a)))$$

$$=-(y\frac{\partial}{\partial a}log(a)+(1-y)\frac{\partial}{\partial a}log(1-a))$$

$$\frac{\partial L}{\partial a}=-(y\frac{1}{a}-(1-y)\frac{1}{1-a})$$



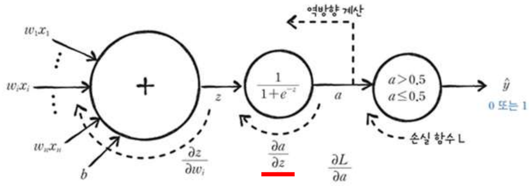
a를 z에 대해 미분하기

$$\frac{\partial e^{-z}}{\partial z}=-e^{-z}$$

$$\frac{\partial a}{\partial z}=\frac{\partial}{\partial z}(\frac{1}{1+e^{-z}})=\frac{\partial}{\partial z}(1+e^{-z})^{-1}$$

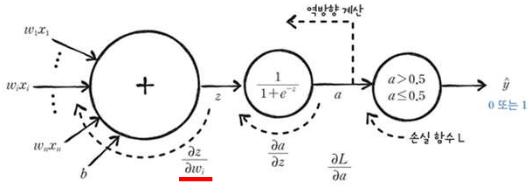
$$=-(1+e^{-z})^{-2}\frac{\partial}{\partial z}(e^{-z})=-(1+e^{-z})^{-2}(-e^{-z})=\frac{e^{-z}}{(1+e^{-z})^{2}}$$

$$\frac{\partial a}{\partial z}=\frac{1}{1+e^{-z}}\frac{e^{-z}}{1+e^{-z}}=\frac{1}{1+e^{-z}}(1-\frac{1}{1+e^{-z}})=a(1-a)$$



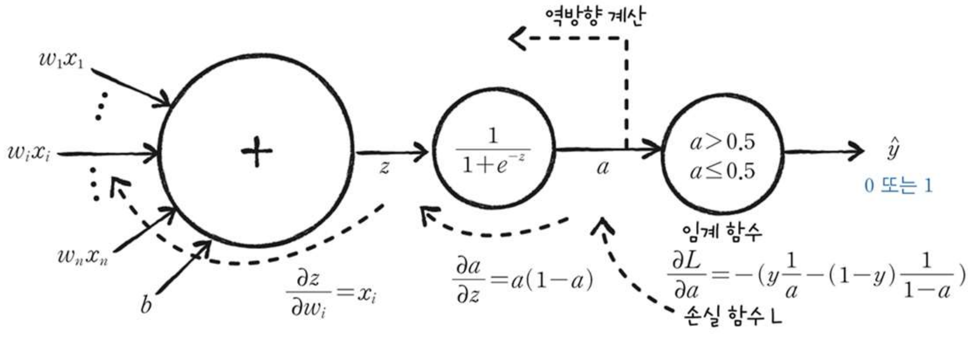
z를 w에 대해 미분하기

$$z = b + \sum_{i=1}^{n} w_i x_i$$



$$\frac{\partial z}{\partial w_i} = x_i$$

전체 미분 과정을 정리하기

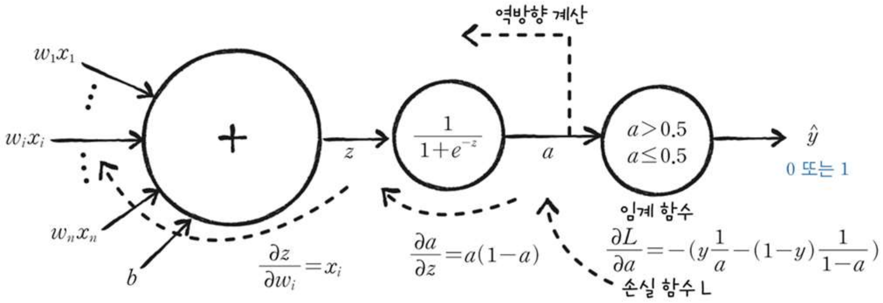


$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial w_i}$$

$$= -(y\frac{1}{a} - (1-y)\frac{1}{1-a})a(1-a)x_i = -(y(1-a) - (1-y)a)x_i \quad \Rightarrow \quad w_i = w_i - \frac{\partial L}{\partial w_i} = w_i + (y-a)x_i$$

$$= -(y - ya - a + ya)x_i = -(y-a)x_i$$

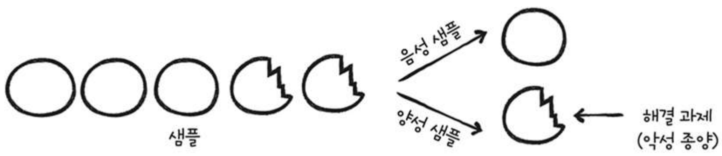
절편에 대한 도함수를 구해 봅니다



$\frac{\partial L}{\partial z}=\frac{\partial L}{\partial a}\frac{\partial a}{\partial z}=-(y-a)$ 이므로 $\frac{\partial L}{\partial b}=\frac{\partial L}{\partial z}\frac{\partial z}{\partial b}=-(y-a)\frac{\partial}{\partial b}(b+\sum_{i=1}^{n}w_ix_i)=-(y-a)1$

$\Rightarrow$ $b=b-\frac{\partial L}{\partial b}=b+(y-a)1$

04-4 분류용 데이터 세트를 준비합니다



	의학	이진 분류
좋음	양성 종양(정상 종양)	음성 샘플
나쁨	악성 종양	양성 샘플 ← 해결 과제

데이터 세트 준비하기

```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer( )
```

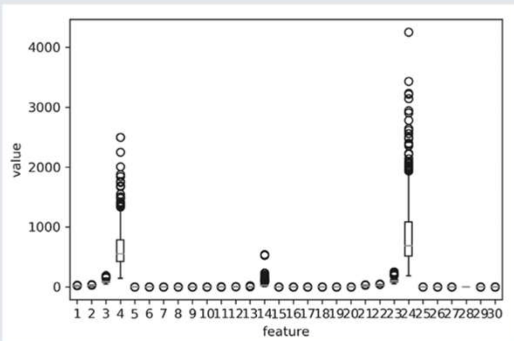
```
print(cancer.data.shape, cancer.target.shape)
(569, 30) (569,)
```

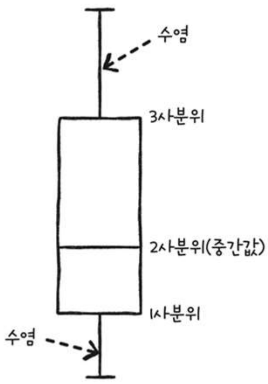
```
cancer.data[:3]
array([[1.799e+01, 1.038e+01, 1.228e+02, 1.001e+03, 1.184e-01, 2.776e-01,
        3.001e-01, 1.471e-01, 2.419e-01, 7.871e-02, 1.095e+00, 9.053e-01,
        8.589e+00, 1.534e+02, 6.399e-03, 4.904e-02, 5.373e-02, 1.587e-02,
        3.003e-02, 6.193e-03, 2.538e+01, 1.733e+01, 1.846e+02, 2.019e+03,
        1.622e-01, 6.656e-01, 7.119e-01, 2.654e-01, 4.601e-01, 1.189e-01],
       [2.057e+01, 1.777e+01, 1.329e+02, 1.326e+03, 8.474e-02, 7.864e-02,
        8.690e-02, 7.017e-02, 1.812e-01, 5.667e-02, 5.435e-01, 7.339e-01,
        3.398e+00, 7.408e+01, 5.225e-03, 1.308e-02, 1.860e-02, 1.340e-02,
        1.389e-02, 3.532e-03, 2.499e+01, 2.341e+01, 1.588e+02, 1.956e+03,
        1.238e-01, 1.866e-01, 2.416e-01, 1.860e-01, 2.750e-01, 8.902e-02],
       [1.969e+01, 2.125e+01, 1.300e+02, 1.203e+03, 1.096e-01, 1.599e-01,
        1.974e-01, 1.279e-01, 2.069e-01, 5.999e-02, 7.456e-01, 7.869e-01,
        4.585e+00, 9.403e+01, 6.150e-03, 4.006e-02, 3.832e-02, 2.058e-02,
        2.250e-02, 4.571e-03, 2.357e+01, 2.553e+01, 1.525e+02, 1.709e+03,
        1.444e-01, 4.245e-01, 4.504e-01, 2.430e-01, 3.613e-01, 8.758e-02]])
```

샘플

박스 플롯(상자 수염 그래프) 그려서 데이터 파악하기

```
plt.boxplot(cancer.data)
plt.xlabel('feature')
plt.ylabel('value')
plt.show( )
```





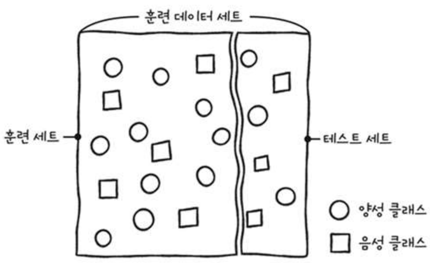
타깃 데이터 확인하고 훈련 데이터 준비하기

```
np.unique(cancer.target, return_counts=True)
(array([0, 1]), array([212, 357]))
```

```
x = cancer.data
y = cancer.target
```

04-5 로지스틱 회귀를 위한 뉴런을 만듭니다

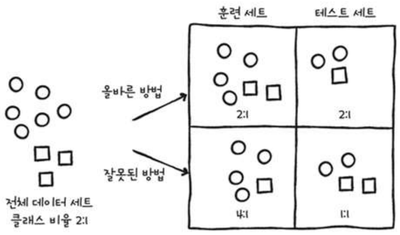
일반화 성능을 평가하기 위해 훈련 세트와 테스트 세트로 나눕니다



훈련 데이터 세트를 훈련 세트와 테스트 세트로 나누는 규칙

- 훈련 데이터 세트를 나눌 때는 테스트 세트보다 훈련 세트가 더 많아야 합니다.
- 훈련 데이터 세트를 나누기 전에 양성, 음성 클래스가 훈련 세트나 테스트 세트의 어느 한쪽에 몰리지 않도록 골고루 섞어야 합니다.

훈련 세트와 테스트 세트 나누기




```
from sklearn.model_selection import train_test_split
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, stratify=y, test_size=0.2,
random_state=42)
```

분할 결과 확인

```
print(x_train.shape, x_test.shape)
(455, 30) (114, 30)
```

```
np.unique(y_train, return_counts=True)
(array([0, 1]), array([170, 285]))
```

로지스틱 뉴런 구현하기

Neuron 클래스와 비슷

가중치와 절편을
미리 초기화 하지 않습니다

```
class LogisticNeuron:

    def __init__(self):
        self.w = None
        self.b = None

    def forpass(self, x):
        z = np.sum(x * self.w) + self.b    # 직선 방정식을 계산합니다.
        return z

    def backprop(self, x, err):
        w_grad = x * err                   # 가중치에 대한 그레이디언트를 계산합니다.
        b_grad = 1 * err                   # 절편에 대한 그레이디언트를 계산합니다.
        return w_grad, b_grad
```

```
a = np.array([1,2,3])
b = np.array([3,4,5])
print(a + b)
array([4, 6, 8])
print(a * b)
array([ 3,  8, 15])
```

```
np.sum(a * b)
26
```

나머지 메서드 구현하기

```
def fit(self, x, y, epochs=100):
    self.w = np.ones(x.shape[1])      # 가중치를 초기화합니다.
    self.b = 0                        # 절편을 초기화합니다.
    for i in range(epochs):           # epochs만큼 반복합니다.
        for x_i, y_i in zip(x, y):    # 모든 샘플에 대해 반복합니다.
            z = self.forpass(x_i)     # 정방향 계산
            a = self.activation(z)    # 활성화 함수 적용
            err = -(y_i - a)          # 오차 계산
            w_grad, b_grad = self.backprop(x_i, err) # 역방향 계산
            self.w -= w_grad          # 가중치 업데이트
            self.b -= b_grad          # 절편 업데이트
```

```
def activation(self, z):
    a = 1 / (1 + np.exp(-z))  # 시그모이드 계산
    return a
```

```
def predict(self, x):
    z = [self.forpass(x_i) for x_i in x]     # 선형 함수 적용
    a = self.activation(np.array(z))         # 활성화 함수 적용
    return a > 0.5                           # 계단 함수 적용
```

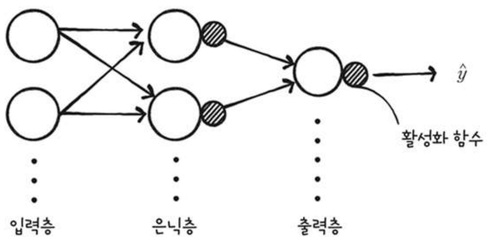
모델 훈련하고 결과 확인하기

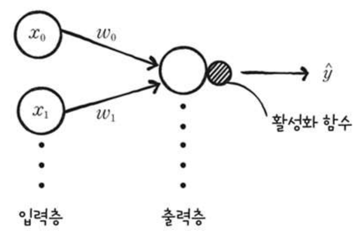
```
neuron = LogisticNeuron( )
neuron.fit(x_train, y_train)
```

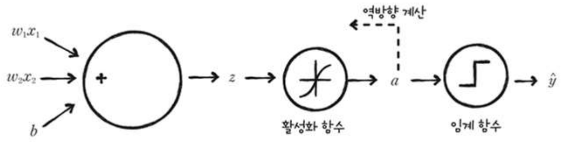
```
np.mean(neuron.predict(x_test) == y_test)
0.8245614035087719
```

04-6 로지스틱 회귀 뉴런으로 단일층 신경망을 만듭니다

이미 단일층 신경망을 구현했습니다! :D



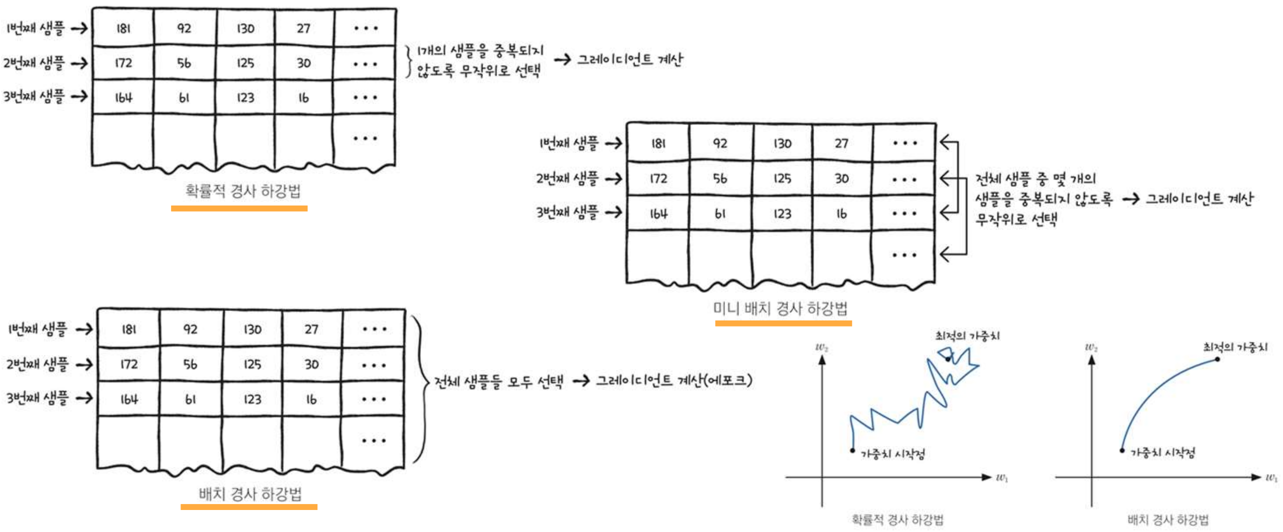




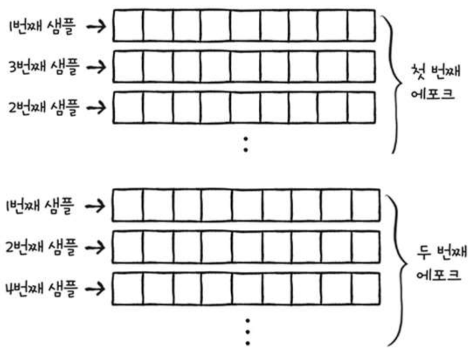
손실 함수 결괏값 저장 기능 추가하기

```
def __init__(self):
    self.w = None
    self.b = None
    self.losses = []
    ...

def fit(self, x, y, epochs=100):
        ...                                    # 이 부분은 잠시 후에 설명합니다.

        for i in index:                        # 모든 샘플에 대해 반복합니다.
            z = self.forpass(x[i])             # 정방향 계산
            a = self.activation(z)             # 활성화 함수 적용
            err = -(y[i]  -  a)                # 오차 계산
            w_grad, b_grad = self.backprop(x[i], err)  # 역방향 계산
            self.w -= w_grad                   # 가중치 업데이트
            self.b -= b_grad                   # 절편 업데이트
            # 안전한 로그 계산을 위해 클리핑한 후 손실을 누적합니다.
            a = np.clip(a, 1e-10, 1-1e-10)
            loss += -(y[i]*np.log(a)+(1-y[i])*np.log(1-a))
                                               # 에포크마다 평균 손실을 저장합니다.
        self.losses.append(loss/len(y))
```

여러가지 경사 하강법



에포크마다 훈련 샘플 섞기




```
def fit(self, x, y, epochs=100):
    self.w = np.ones(x.shape[1])                # 가중치를 초기화합니다.
    self.b = 0                                  # 절편을 초기화합니다.
    for i in range(epochs):                     # epochs만큼 반복합니다.
        loss = 0
        indexes = np.random.permutation(np.arange(len(x)))  # 인덱스를 섞습니다.
        for i in indexes:                       # 모든 샘플에 대해 반복합니다.
            z = self.forpass(x[i])              # 정방향 계산
            a = self.activation(z)              # 활성화 함수 적용
            err = -(y[i] - a)                   # 오차 계산
            w_grad, b_grad = self.backprop(x[i], err) # 역방향 계산
            self.w -= w_grad                    # 가중치 업데이트
            self.b -= b_grad                    # 절편 업데이트
            a = np.clip(a, 1e-10, 1-1e-10)  # 안전한 로그 계산을 위해 클리핑한 후 손실을 누적합니다.
            loss += -(y[i]*np.log(a)+(1-y[i])*np.log(1-a)) # 에포크마다 평균 손실을 저장합니다.
        self.losses.append(loss/len(y))
```

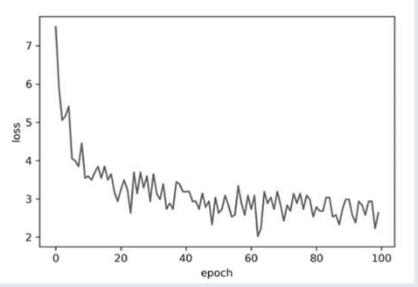
score() 메서드 추가하고 단일층 신경망 훈련하기

활성화 함수를 뺐습니다

```
def predict(self, x):
    z = [self.forpass(x_i) for x_i in x]    # 정방향 계산
    return np.array(z) > 0                  # 계단 함수 적용

def score(self, x, y):
    return np.mean(self.predict(x) == y)
```

```
layer = SingleLayer( )
layer.fit(x_train, y_train)
layer.score(x_test, y_test)
0.9298245614035088
```

```
plt.plot(layer.losses)
plt.xlabel('epoch')
plt.ylabel('loss')
plt.show( )
```



04-7 사이킷런으로 로지스틱 회귀를 수행합니다

로지스틱 손실 함수 지정

```
sgd = SGDClassifier(loss='log', max_iter=100, tol=1e-3, random_state=42)
```

회귀는 SGDRegressor

```
sgd.fit(x_train, y_train)
sgd.score(x_test, y_test)
0.8333333333333334
```

```
sgd.predict(x_test[0:10])
array([0, 1, 0, 0, 0, 0, 1, 0, 0, 0])
```

05 훈련 노하우를 배웁니다

05-1 검증 세트를 나누고 전처리 과정을 배웁니다

테스트 세트로 모델을 튜닝합니다(올바른 일반화 성능을 추정하기 어렵습니다)

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
cancer = load_breast_cancer( )
x = cancer.data
y = cancer.target
x_train_all, x_test, y_train_all, y_test = train_test_split(x, y, stratify=y,
                                                            test_size=0.2, random_state=42)
```

하이퍼파라미터

```
from sklearn.linear_model import SGDClassifier
sgd = SGDClassifier(loss='log', random_state=42)
sgd.fit(x_train_all, y_train_all)
sgd.score(x_test, y_test)
0.8333333333333334
```

```
from sklearn.linear_model import SGDClassifier
sgd = SGDClassifier(loss='hinge', random_state=42)
sgd.fit(x_train, y_train)
sgd.score(x_test, y_test)
0.9385964912280702
```

서포트 벡터 머신(SVM)

검증 세트를 준비합니다

혹은 개발 세트

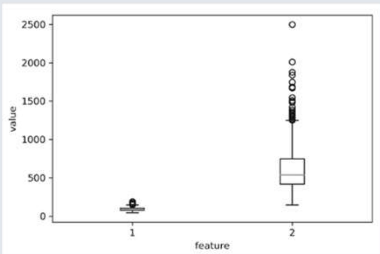

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
cancer = load_breast_cancer( )
x = cancer.data
y = cancer.target
x_train_all, x_test, y_train_all, y_test = train_test_split(x, y, stratify=y,
                                                test_size=0.2, random_state=42)
```

```
x_train, x_val, y_train, y_val = train_test_split(x_train_all, y_train_all,
stratify=y_train_all, test_size=0.2, random_state=42)
print(len(x_train), len(x_val))
364 91
```

데이터 전처리와 특성의 스케일을 알아봅니다

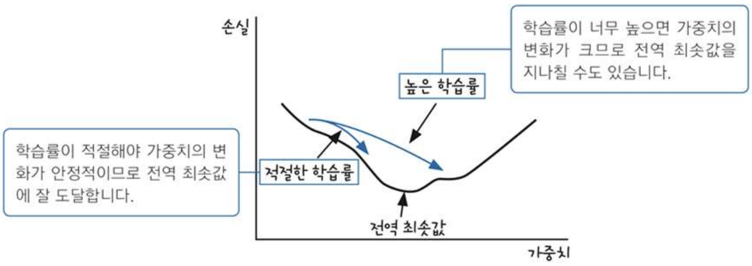
	당도	무게
사과 1	4	540
사과 2	8	700
사과 3	2	480

```
print(cancer.feature_names[[2,3]])
plt.boxplot(x_train[:, 2:4])
plt.xlabel('feature')
plt.ylabel('value')
plt.show( )
['mean perimeter' 'mean area']
```

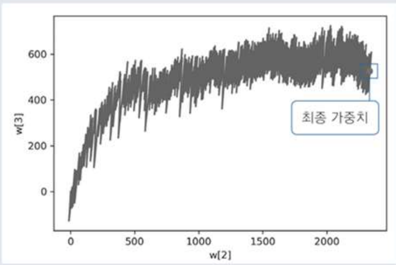


$$w_i = w_i - \frac{\partial L}{\partial w_i} = w_i + (y - a)x_i$$

가중치를 기록하고 학습률을 적용하기




```
w2 = []
w3 = []
for w in layer1.w_history:
    w2.append(w[2])
    w3.append(w[3])
plt.plot(w2, w3)
plt.plot(w2[-1], w3[-1], 'ro')
plt.xlabel('w[2]')
plt.ylabel('w[3]')
plt.show( )
```

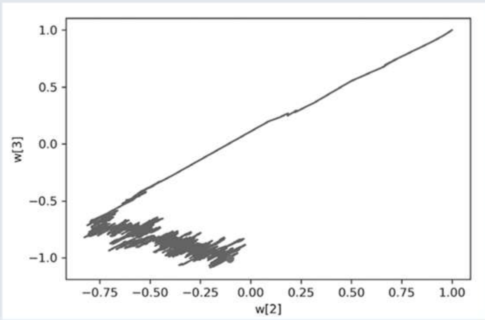


스케일을 조정해 모델을 훈련합니다

표준화(standardization): 평균 0, 분산 1 $z=\frac{x-\mu}{s}$ $s=\sqrt{\frac{1}{m}\sum_{i=0}^{m}(x_i-\mu)^2}$

```
train_mean = np.mean(x_train, axis=0)
train_std = np.std(x_train, axis=0)
x_train_scaled = (x_train - train_mean) / train_std
```

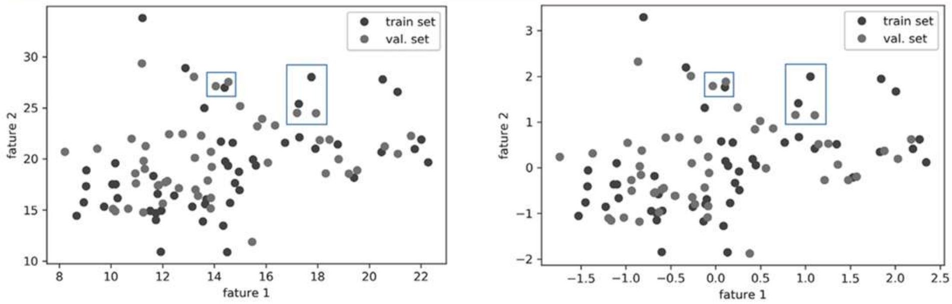
```
layer2 = SingleLayer( )
layer2.fit(x_train_scaled, y_train)
w2 = []
w3 = []
for w in layer2.w_history:
    w2.append(w[2])
    w3.append(w[3])
plt.plot(w2, w3)
plt.plot(w2[-1], w3[-1], 'ro')
plt.xlabel('w[2]')
plt.ylabel('w[3]')
plt.show( )
```



검증 세트로 모델 성능 평가하기

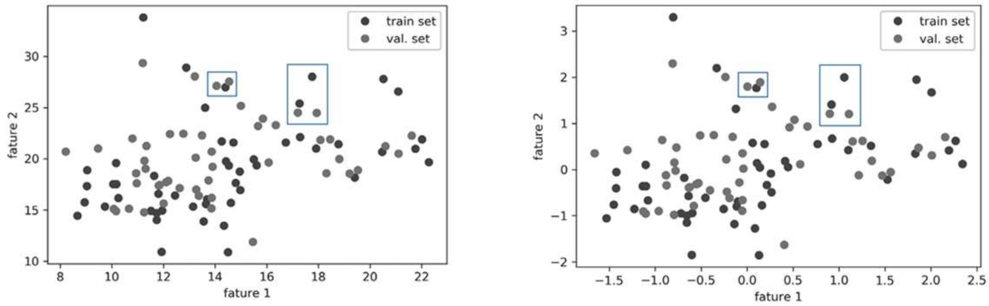
검증 세트의 평균, 표준 편차로 전처리해도 괜찮을까요?

```
val_mean = np.mean(x_val, axis=0)
val_std = np.std(x_val, axis=0)
x_val_scaled = (x_val - val_mean) / val_std
layer2.score(x_val_scaled, y_val)
0.967032967032967
```



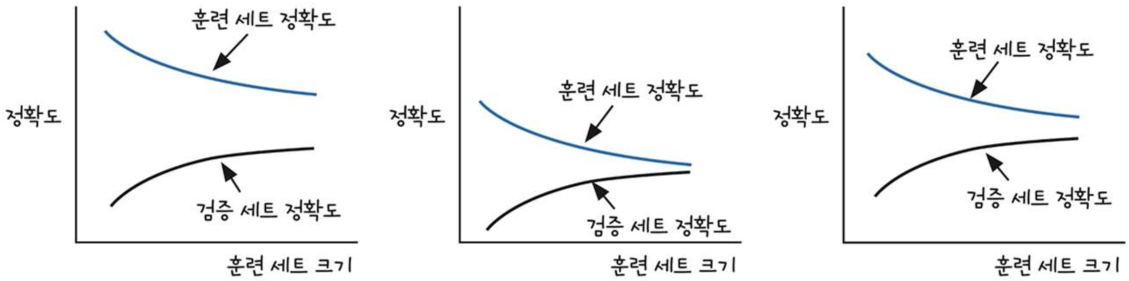
올바르게 검증 세트 전처리하기

```
x_val_scaled = (x_val - train_mean) / train_std
plt.plot(x_train_scaled[:50, 0], x_train_scaled[:50, 1], 'bo')
plt.plot(x_val_scaled[:50, 0], x_val_scaled[:50, 1], 'ro')
plt.xlabel('fature 1')
plt.ylabel('fature 2')
plt.legend(['train set', 'val. set'])
plt.show( )
```



05-2 과대적합과 과소적합을 알아봅니다

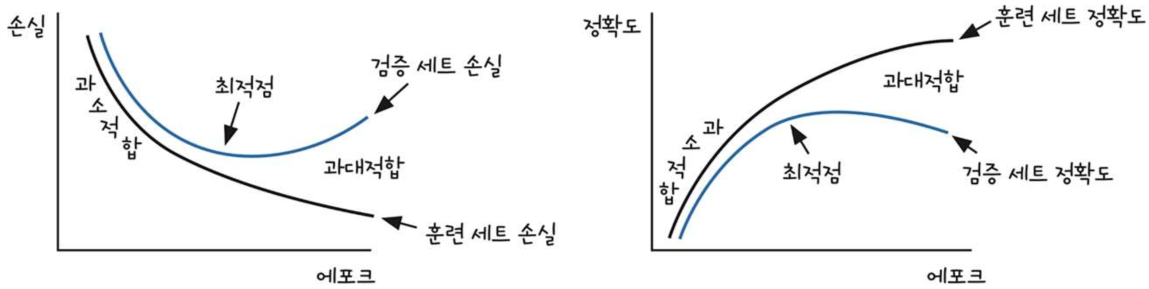
훈련 세트 vs 정확도



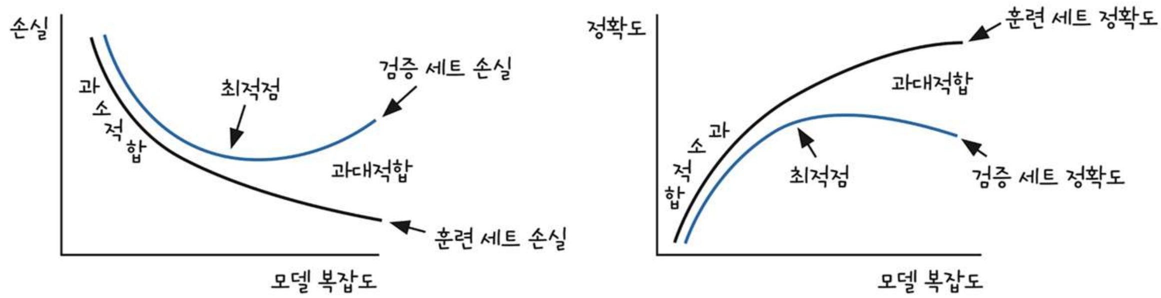
과대적합: 분산이 큼

과소적합: 편향이 큼

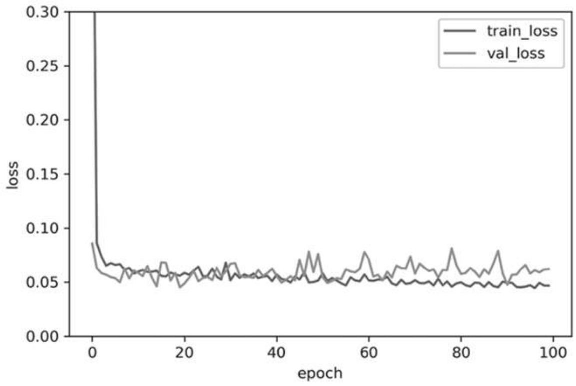
에포크 vs 손실 함수



모델 복잡도 vs 손실 함수



적절한 편향-분산 트레이드오프를 선택합니다

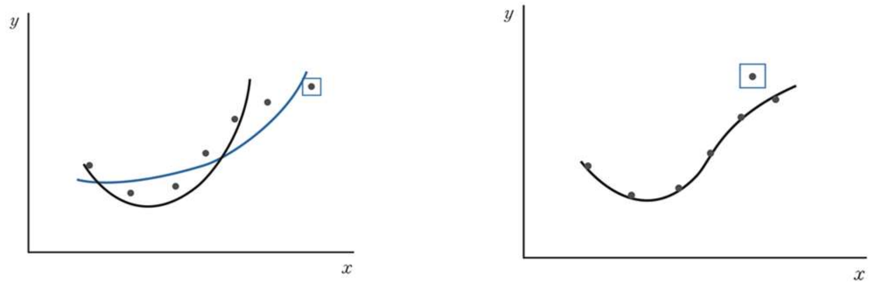


조기 종료

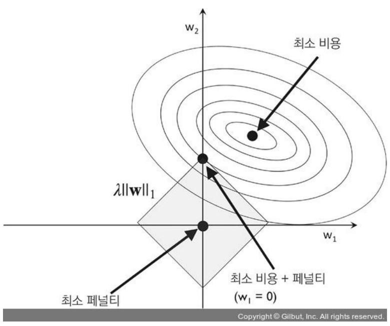
```
layer4 = SingleLayer( )
layer4.fit(x_train_scaled, y_train, epochs=20)
layer4.score(x_val_scaled, y_val)
0.978021978021978
```

05-3 규제 방법을 배우고 단일층 신경망에 적용합니다

어떤 모델이 좋은 모델일까요?



L1 규제를 알아봅니다



손실 함수 + L1 노름(norm)

$$||w||_1 = \sum_{i=1}^{n} |w_i|$$

하이퍼파라미터

$$L = -(y\log(a) + (1-y)\log(1-a)) + \alpha\sum_{i=1}^{n}|w_i|$$

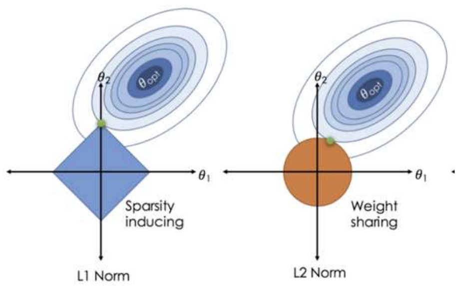
$$\frac{\partial}{\partial w}L = -(y-a)x + \alpha \times sign(w)$$

```
w_grad += alpha * np.sign(w)
```

선형 회귀 + L1 규제 : 라쏘(Lasso)

L2 규제를 알아봅니다

손실 함수 + L2 노름(norm)



$$||w||_2=\sqrt{\sum_{i=1}^{n}|w_i|^2}$$

하이퍼파라미터

$$L=-(ylog(a)+(1-y)log(1-a))+\frac{1}{2}\alpha\sum_{i=1}^{n}|w_i|^2$$

$$\frac{\partial}{\partial w}L=-(y-a)x+\alpha\times w$$

```
w_grad += alpha * w
```

선형 회귀 + L2 규제 : 릿지(Ridge)

L1 규제와 L2 규제 정리

L1 규제

그레이디언트에서 `alpha`에 가중치의 부호를 곱하여 그레이디언트에 더합니다.

```
w_grad += alpha * np.sign(w)
```

L2 규제

그레이디언트에서 `alpha`에 가중치를 곱하여 그레이디언트에 더합니다.

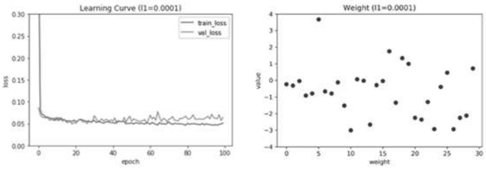
```
w_grad += alpha * w
```

```
# 그레이디언트에서 페널티 항의 미분값을 더합니다.
w_grad += self.l1 * np.sign(self.w) + self.l2 * self.w
self.w -= self.lr * w_grad          # 가중치 업데이트
self.b -= b_grad                    # 절편 업데이트
```

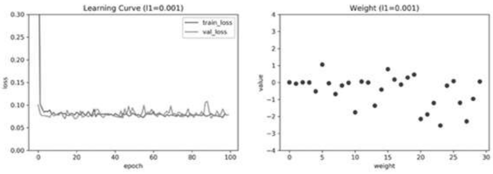
```
def reg_loss(self):
    return self.l1 * np.sum(np.abs(self.w)) + self.l2 / 2 * np.sum(self.w**2)
```

로지스틱 회귀 + L1 규제

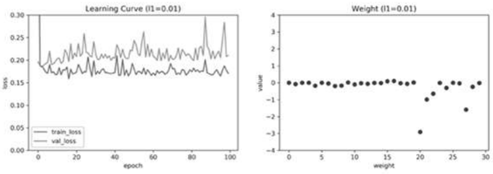
l1=0.0001



l1=0.001

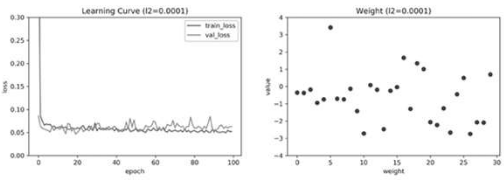


l1=0.01

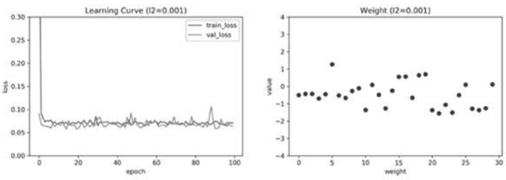


로지스틱 회귀 + L2 규제

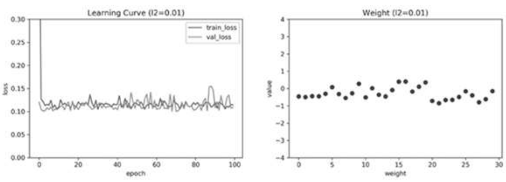
l2=0.0001



l2=0.001

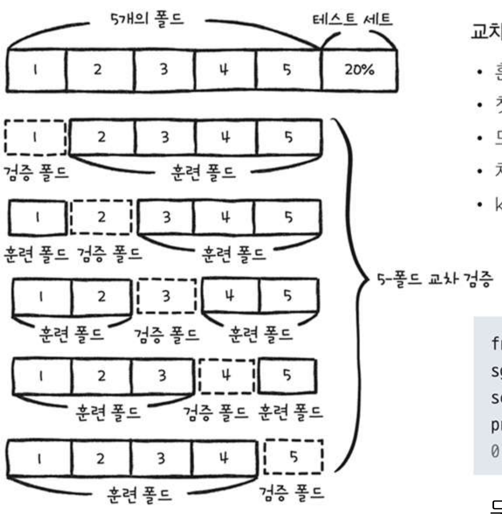


l2=0.01



05 훈련 노하우를 배웁니다

05-4 교차 검증을 알아보고 사이킷런으로 수행해 봅니다



교차 검증 과정

- 훈련 세트를 k개의 폴드(fold)로 나눕니다.
- 첫 번째 폴드를 검증 세트로 사용하고 나머지 폴드(k-1개)를 훈련 세트로 사용합니다.
- 모델을 훈련한 다음에 검증 세트로 평가합니다.
- 차례대로 다음 폴드를 검증 세트로 사용하여 반복합니다.
- k개의 검증 세트로 k번 성능을 평가한 후 계산된 성능의 평균을 내어 최종 성능을 계산합니다.

```
from sklearn.model_selection import cross_validate
sgd = SGDClassifier(loss='log', penalty='l2', alpha=0.001, random_state=42)
scores = cross_validate(sgd, x_train_all, y_train_all, cv=10)
print(np.mean(scores['test_score']))
0.850096618357488
```

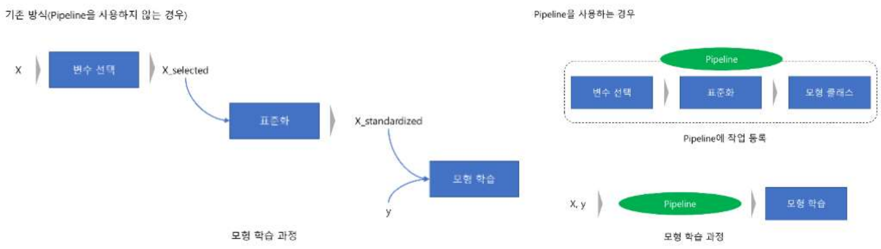
무엇이 잘못 되었을까요?

전처리 단계를 포함해 교차 검증을 수행합니다

검증 세트를 따로 나누지 않고 훈련 세트 전체를 사용합니다 (차라리 Val 사용안함)

전처리와 모델을 하나의 파이프라인으로 정의합니다

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
pipe = make_pipeline(StandardScaler( ), sgd)
scores = cross_validate(pipe, x_train_all, y_train_all, cv=10, return_train_score=True)
print(np.mean(scores['test_score']))
0.9694202898550724
```



Pipeline을 사용하지 않는 경우(왼쪽)과 Pipeline을 사용하는 경우(오른쪽) 모형 학습 과정