머신러닝

machine
learning

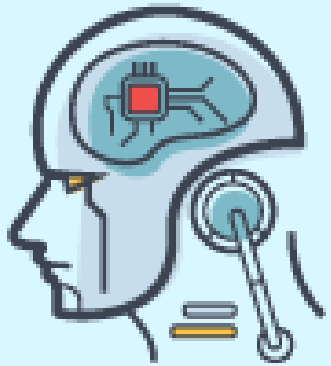
목차

AI, 머신러닝, 딥러닝

Artificial Intelligence

인공지능

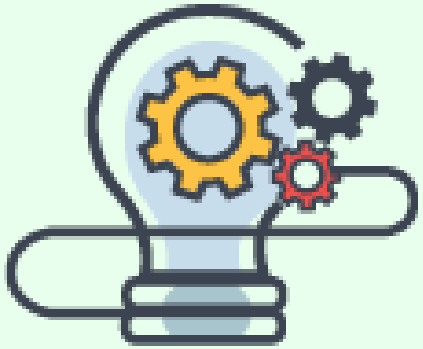
사고나 학습등 인간이 가진
지적 능력을 컴퓨터를 통해
구현하는 기술



Machine Learning

머신러닝

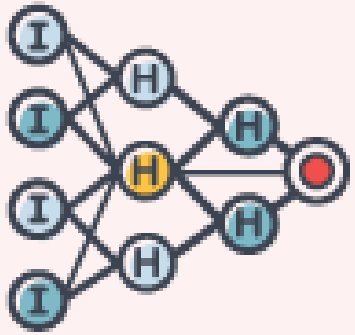
컴퓨터가 스스로 학습하여
인공지능의 성능을
향상 시키는 기술 방법



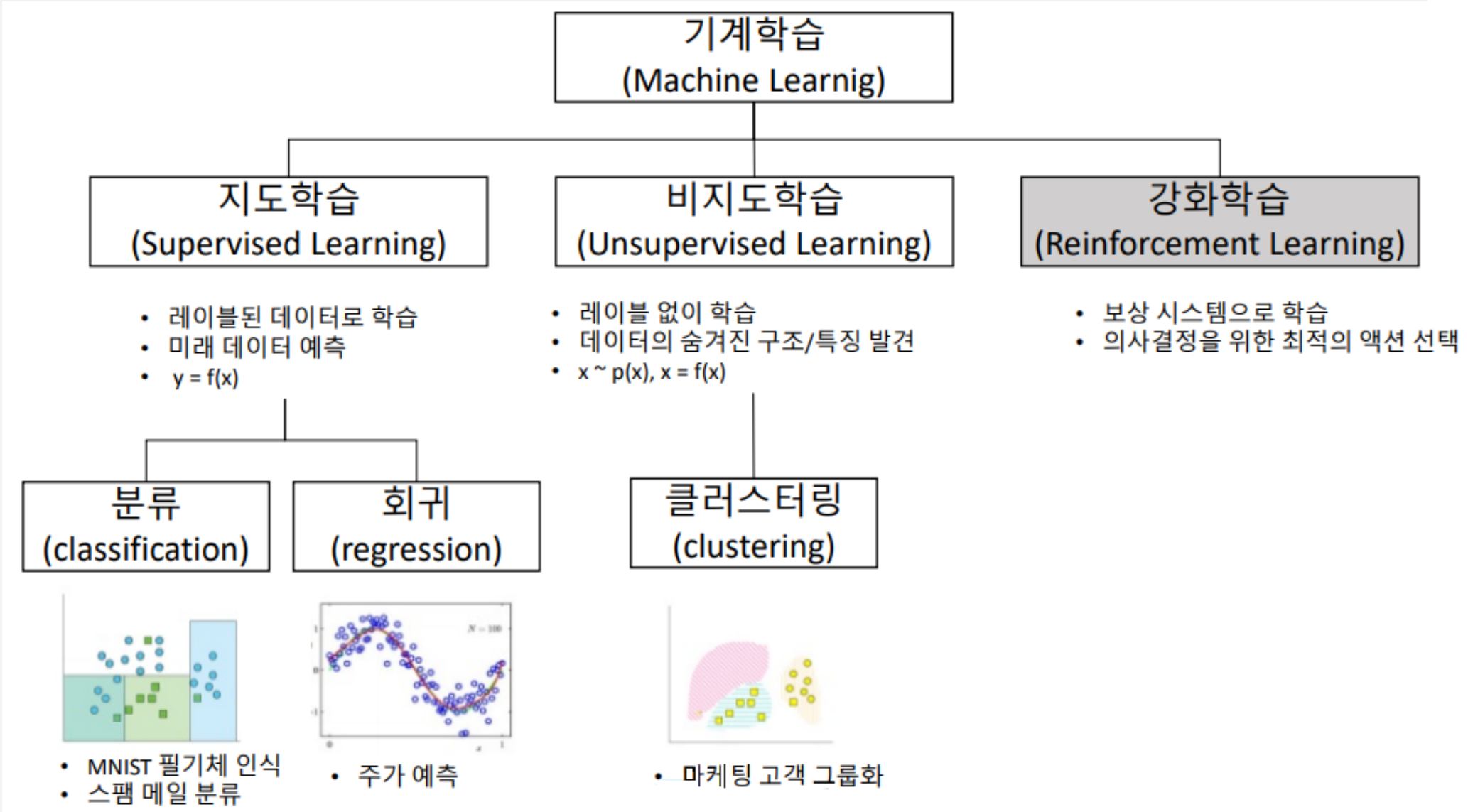
Deep Learning

딥러닝

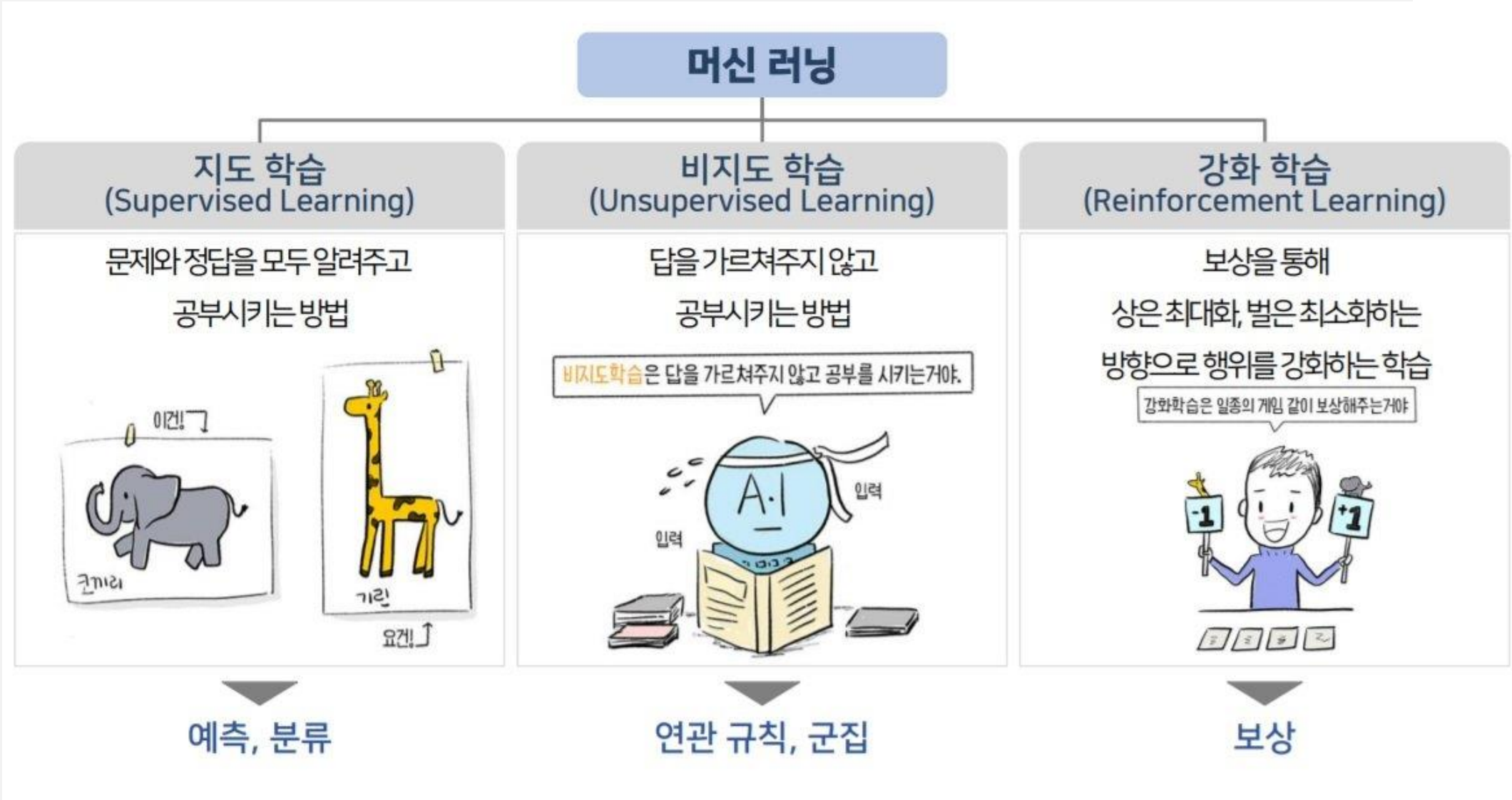
인간의 뉴런과 비슷한
인공신경망 방식으로
정보를 처리



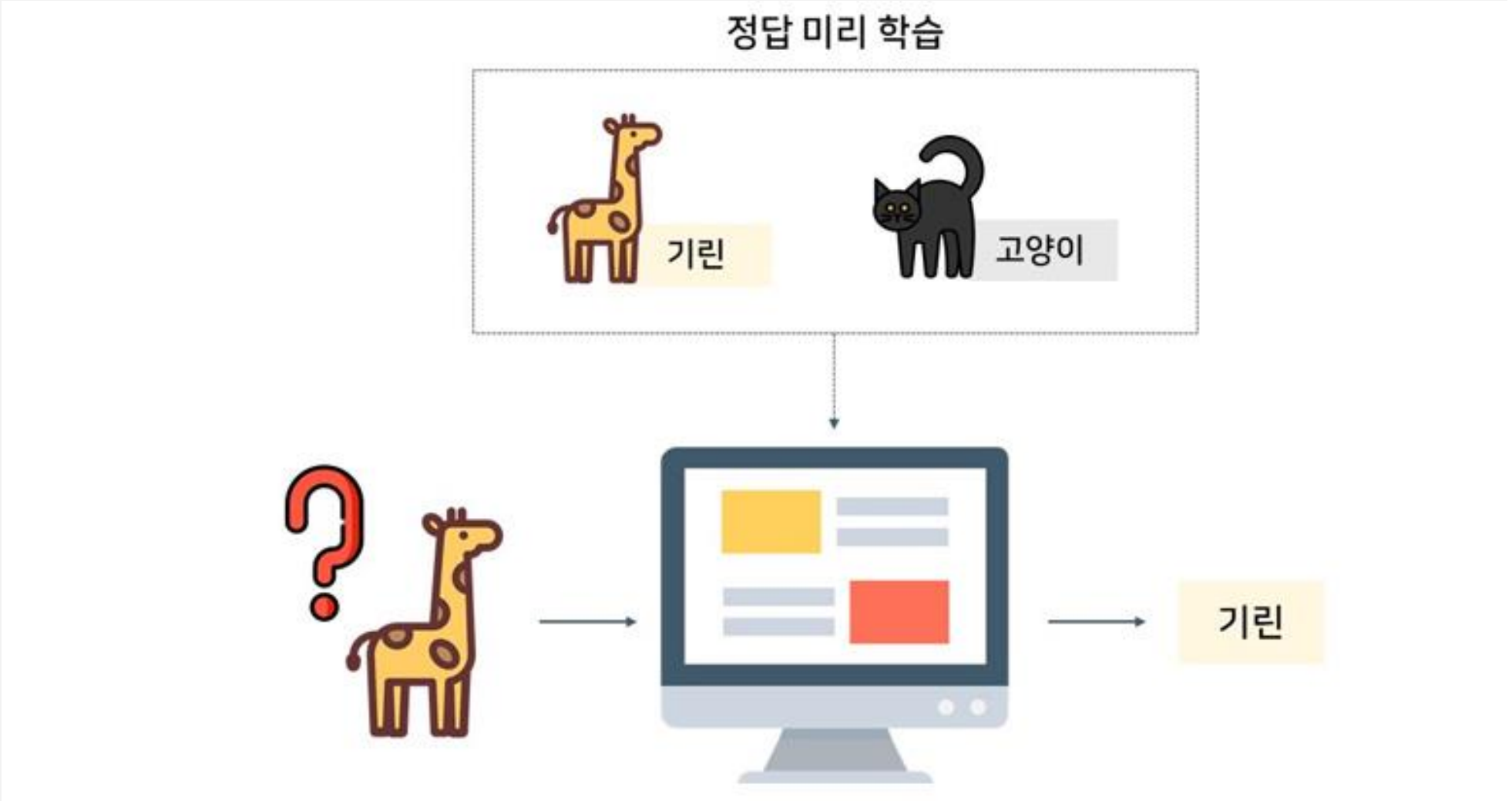
지도학습, 비지도학습, 강화학습



지도학습, 비지도학습, 강화학습



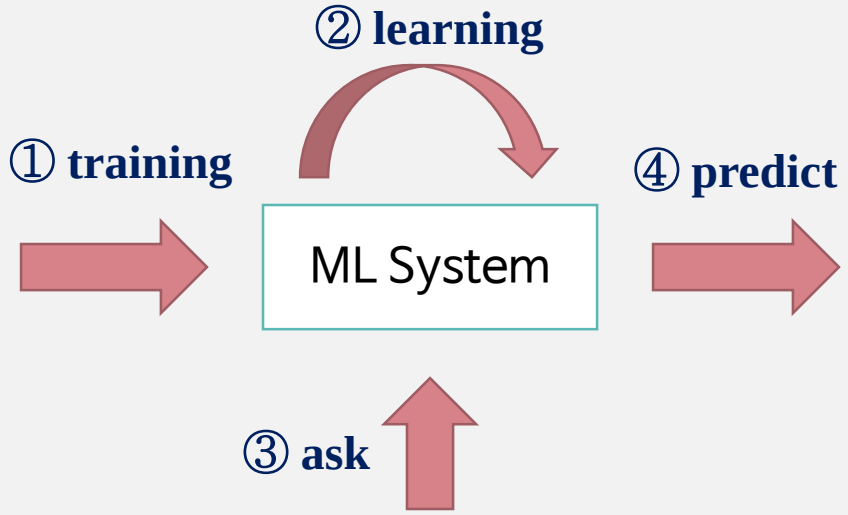
지도학습



비지도학습

지도학습(Supervised Learning)

공부시간(x)	시험성적(t)
9	74
14	81
21	86
27	90
32	88
37	92



공부시간-시험성적 학습 결과를 바탕으로 입력 데이터에 없는 공부시간에 대한 성적 예측

공부시간이 43시간일 경우 시험성적은?

Test Data : 43시간

지도학습은 입력 값(x)과 정답(t, label)을 포함하는 Training Data를 이용하여 학습하고, 그 학습된 결과를 바탕으로 미지의 데이터(Test Data)에 대해 미래 값을 예측(Predict)하는 방법 -〉 대부분 머신러닝 문제는 지도학습에 해당

예1) 시험 공부 시간(입력)과 Pass/Fail(정답)을 이용하여 당락 여부 예측

예2) 집 평수(입력)과 가격 데이터(정답) 이용하여 임의의 평수 가격 예측

지도학습(Supervised Learning)- 회귀(Regression), 분류(Classification)

지도학습은 학습결과를 바탕으로 미래의 무엇을 예측하느냐에 따라 회귀(Regression), 분류(Classification)등으로 구분할 수 있음

- 회귀(Regression)는 Training Data를 이용하여 연속적인(숫자)값을 예측하는 것

  예) 집 평수와 가격관계, 공부시간과 시험성적 등의 관계

- 분류(Classification)는 Training Data를 이용하여 주어진 입력 값이 어떤 종류인지 구별하는 것

  예) 공부시간과 합격여부

공부시간(x)	시험성적(t)
9	74
14	81
21	86
27	90
32	88
37	92

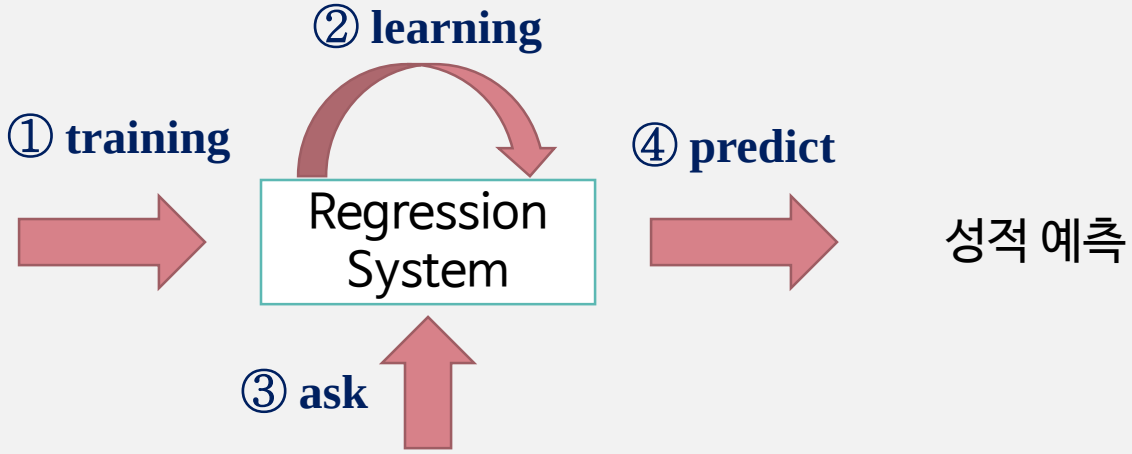
집 평수(x)	가격(t)
20	98
25	119
30	131
40	133
50	140
55	196

공부시간(x)	시험성적(t)
9	fail
14	fail
21	pass
27	pass
32	pass
37	pass

집 평수(x)	가격(t)
20	low
25	low
30	medium
40	medium
50	medium
55	high

지도학습(Supervised Learning)- 회귀(Regression), 분류(Classification)

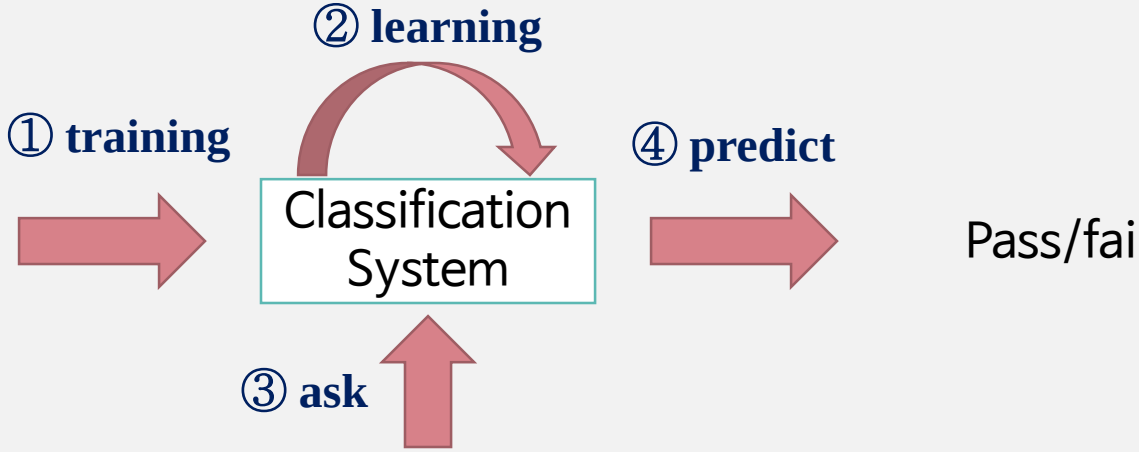
공부시간(x)	시험성적(t)
9	74
14	81
21	86
27	90
32	88
37	92



성적 예측

Test Data : 공부시간이 43시간일 경우 시험성적은?

공부시간(x)	시험성적(t)
9	fail
14	fail
21	pass
27	pass
32	pass
37	pass



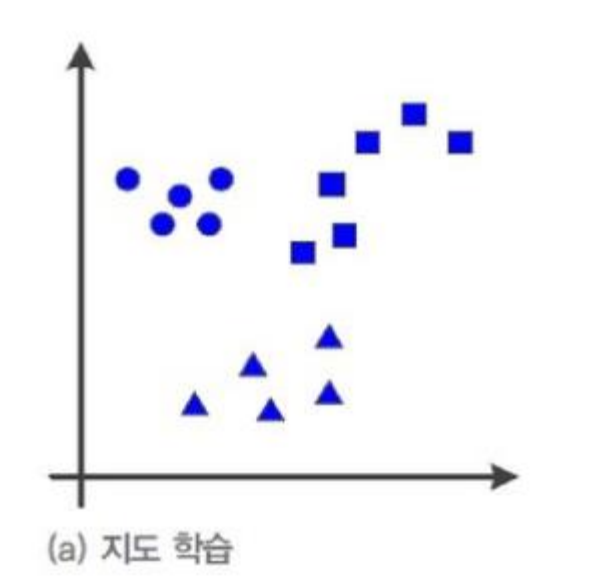
Pass/fail 예측

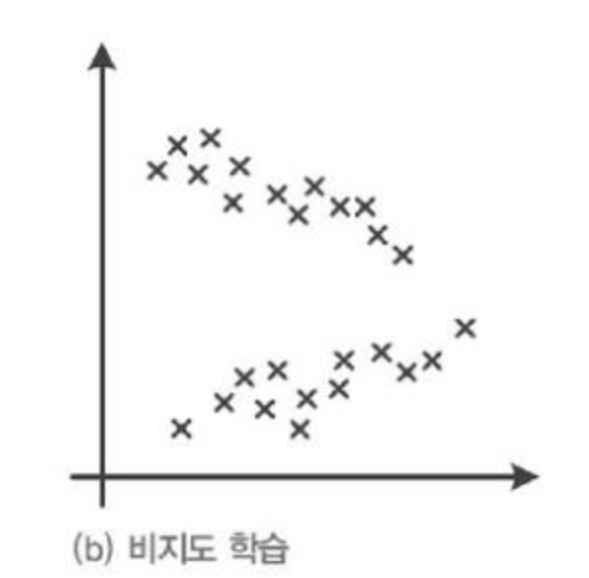
Test Data : 공부시간이 43시간일 경우 pass?

비지도학습(Unsupervised Learning)

비지도학습(Unsupervised Learning)은 트레이닝 데이터에 정답은 없고 입력 데이터만 있기 때문에 입력에 대한 정답을 찾는 것이 아닌 패턴, 특성 등을 학습을 통해 발견하는 방법을 말함.

예) 군집화 알고리즘을 이용한 뉴스 그룹핑, 백화점 상품 추천 시스템 등





지도학습(예측) – 선형 회귀(Linear Regression)

- Training Data를 이용하여 데이터의 특성과 상관관계 등을 파악하고,

그 결과를 바탕으로 Training Data에 없는 미지의 데이터가 주어졌을 경우에

그 결과를 연속적인 (숫자)값으로 예측하는 것

- 주어진 데이터를 이용해 일차방정식을 수정해 나가는 것

학습을 거쳐서 가장 합리적인 선을 찾아내는 것

학습을 많이 해도 '완벽한' 식을 찾아 내지 못할 수 있다.

가장 근사치의 값을 찾는 것

지도 학습 – 선형 회귀(Linear Regression)

[step1] analyze training data

학습데이터(training data)는 입력 (x)인 공부시간에 비례해서 출력(y)인 시험성적도 증가하는 경향이 있음

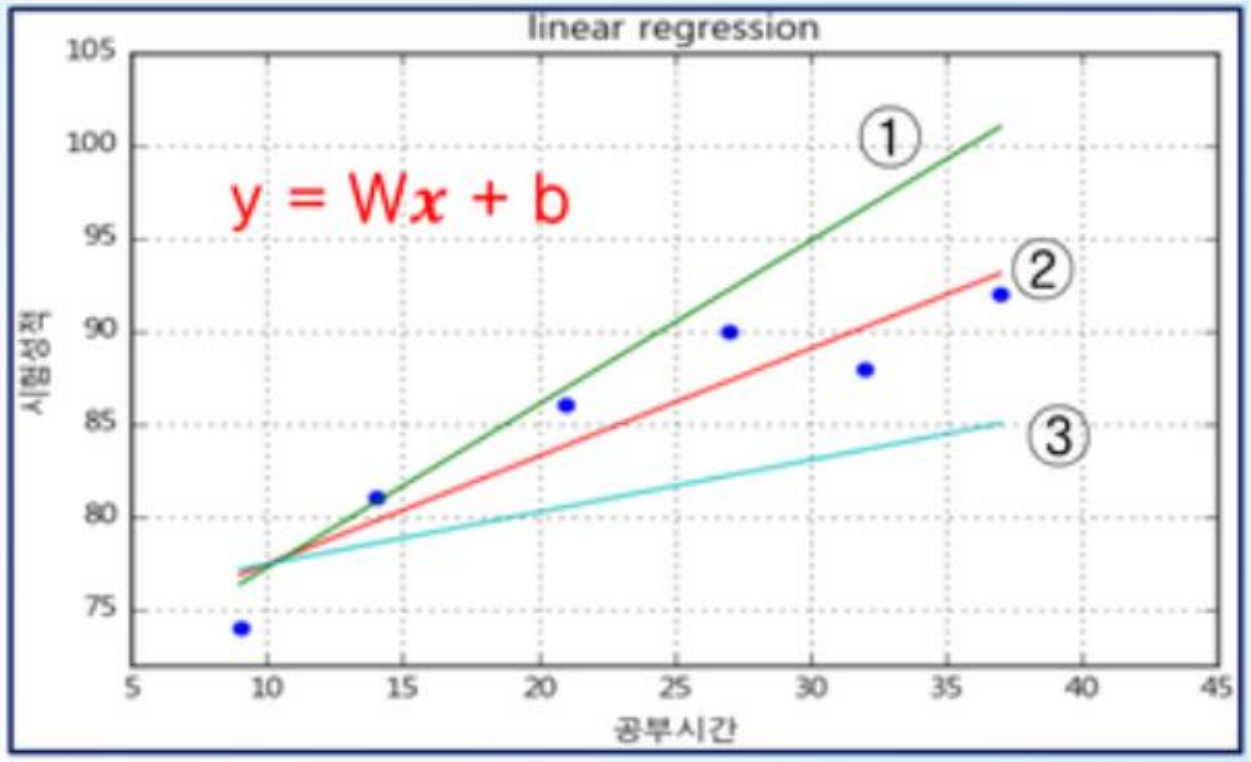
즉, 입력(x)과 출력(y) 은 $y = Wx + b$ 형태로 나타낼 수 있음

공부시간(x)	시험성적(t)
9	74
14	81
21	86
27	90
32	88
37	92

지도 학습 – 선형 회귀(Linear Regression)

[step2] find W and b

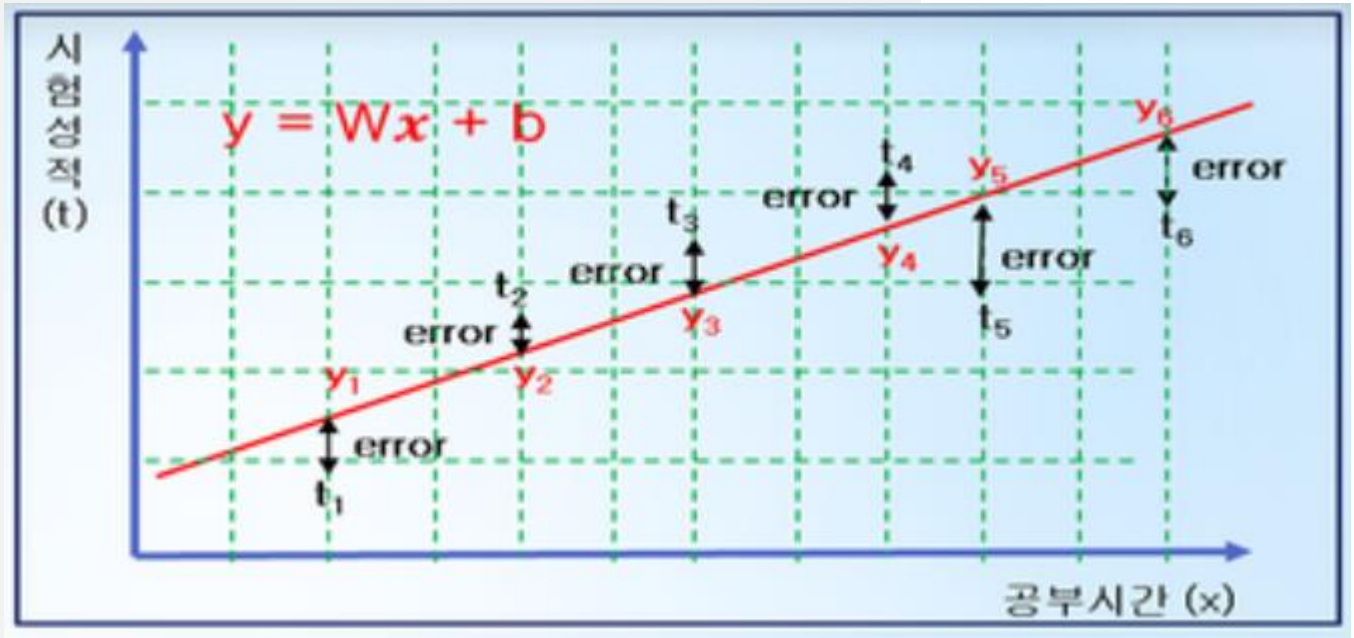
- ①②③,… 등의 다향한 $y = Wx + b$ 직선 中
- training data의 특성을 가장 잘 표현할 수 있는 가중치 W(기울기), 바이어스 b (y절편)를 찾는 것이 학습(Learning) 개념임



지도 학습 – 선형 회귀(Linear Regression)

공부시간(x)	시험성적(t)
9	74
14	81
21	86
27	90
32	88
37	92

오차(error)=t-y



Training data의 정답(t)과 직선 y=Wx+b 값의 차이인 오차(error)는
오차(error)=t-y=t-(Wx+b)으로 계산되며
오차가 크다면, 우리가 임의로 설정한 직선의 가중치와 바이어스 값이 잘못된 것이고
오차가 작다면 직선의 가중치와 바이어스 값이 잘 된 것이기 때문에
미래 값 예측도 정확할 수 있다고 예상할 수 있음.

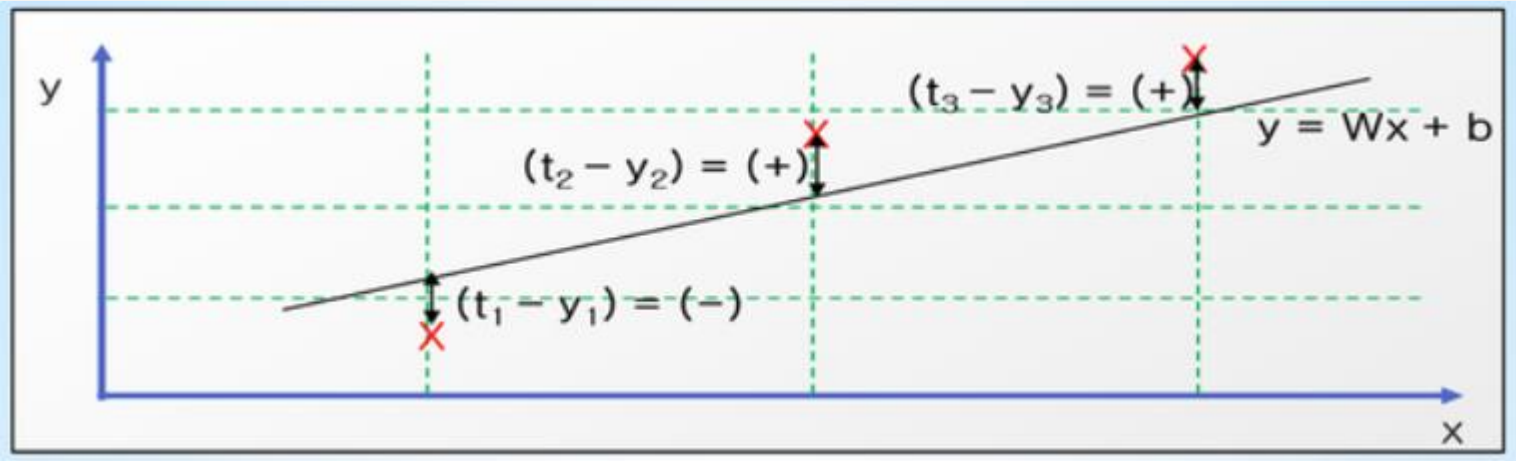
지도 학습 – 선형 회귀(Linear Regression)

머신러닝 regression 시스템은

모든 데이터의 오차(error) = t−y=t−(Wx+b)의 합이 최소가 되어서
미래 값을 잘 예측할 수 있는 가중치 W와 바이어스 b값을 찾아야 함.

선형 회귀(Linear Regression)-손실함수(loss function)

손실함수(loss function 또는 cost function)는
training data의 정답(t)과 입력(x)에 대한 계산 값 y 의 차이를 모두 더해 수식으로 나타낸 것



각각의 오차인(t-y)를 모두 더해서 손실함수(loss function)을 구하면 각각의 오차가(+),(-)등이 동시에 존재하기 때문에 오차의 합이 0이 나올 수도 있음.
즉, 0이라는 것이 최소 오차 값인지 아닌지를 판별하는 것이 어려움.

손실함수에서 오차(error)를 계산할 때는 $(t-y)^2 = (t-[Wx+b])^2$ 을 사용함.
즉 오차는 언제나 양수이며, 제곱을 하기 때문에 정답과 계산 값 차이가 크다면
제곱에 의해 오차는 더 큰 값을 가지게 되어 머신 러닝 학습에 있어 장점을 가짐.

선형 회귀(Linear Regression)-손실함수(loss function)

$$y = Wx + b$$

$$\text{loss function} = E(W,b) = \frac{1}{n}\sum_{i=1}^{n}[t_i - y_i]^2 = \frac{1}{n}\sum_{i=1}^{n}[t_i - (Wx_i + b)]^2$$

X와 t는 training data에서 주어지는 값이므로
손실함수(loss function)인 E(W,b)는 결국 W와 b에 영향을 받는 함수임.

- E(W,b)값이 작다는 것은 정답(t , target)과 y=Wx+b에 의해 계산된 값의 평균 오차가 작다는 의미

- 평균 오차가 작다는 것은 미지의 데이터 x가 주어질 경우,
  확률적으로 미래의 결과값도 오차가 작을 것이라고 추측할 수 있음.

- 이처럼 training data를 바탕으로 손실 함수 E(W,b)가 최소값을 갖도록
  (W , b)를 구하는 것이 (linear) regression model의 최종 목적임

선형 회귀(Linear Regression)-경사하강법

손실 함수 E(W,b)가 최소값을 갖도록 (W , b)를 구하는 것이 (linear) regression model의 최종 목적

경사하강법
(gradient decent algorithm)

선형 회귀(Linear Regression)-경사하강법

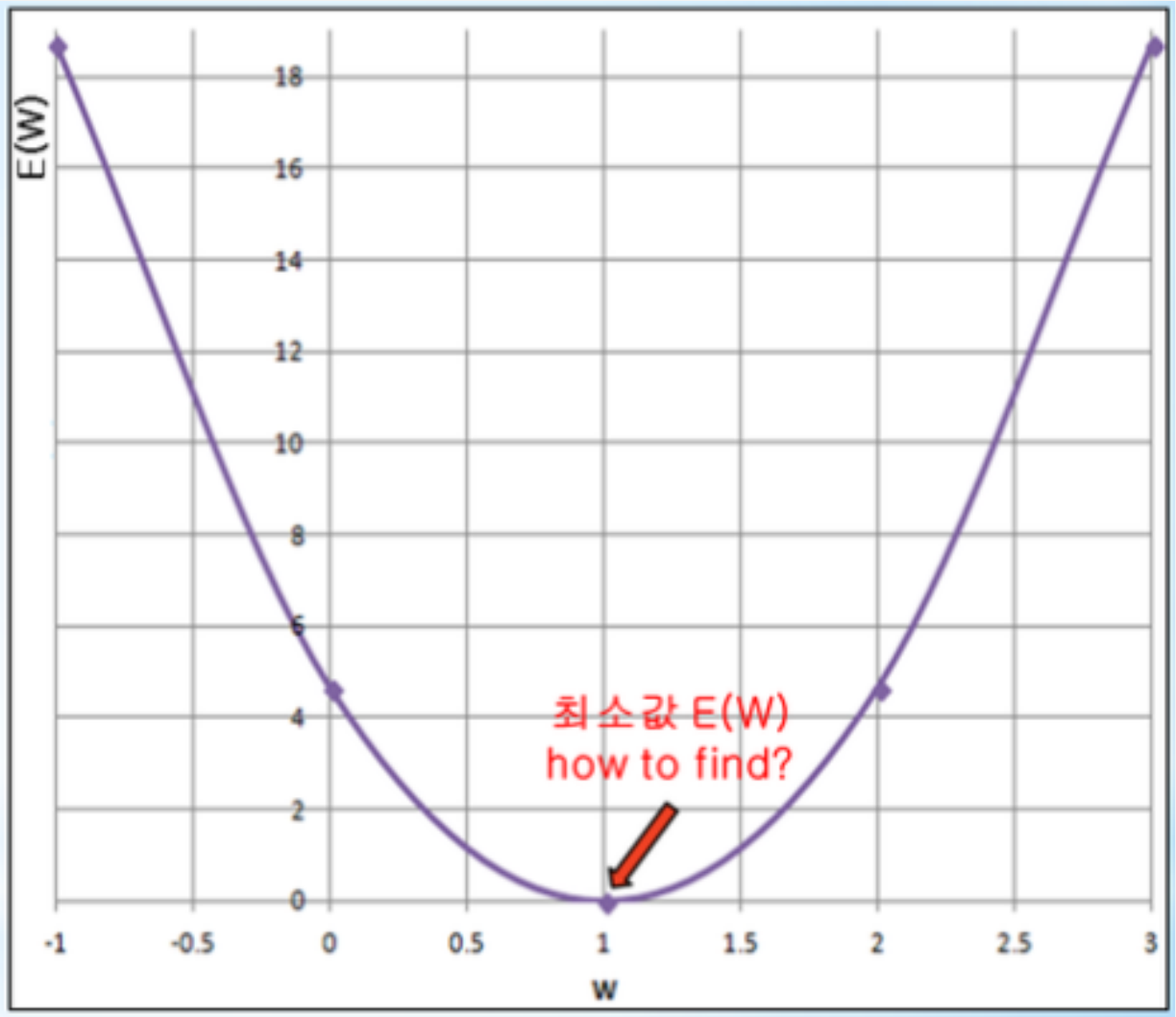
계산을 쉽게 하고 손실함수의 모양을 파악하기 위해 E(W,b)에서 바이어스 b=0을 가정

x	t
1	1
2	2
3	3

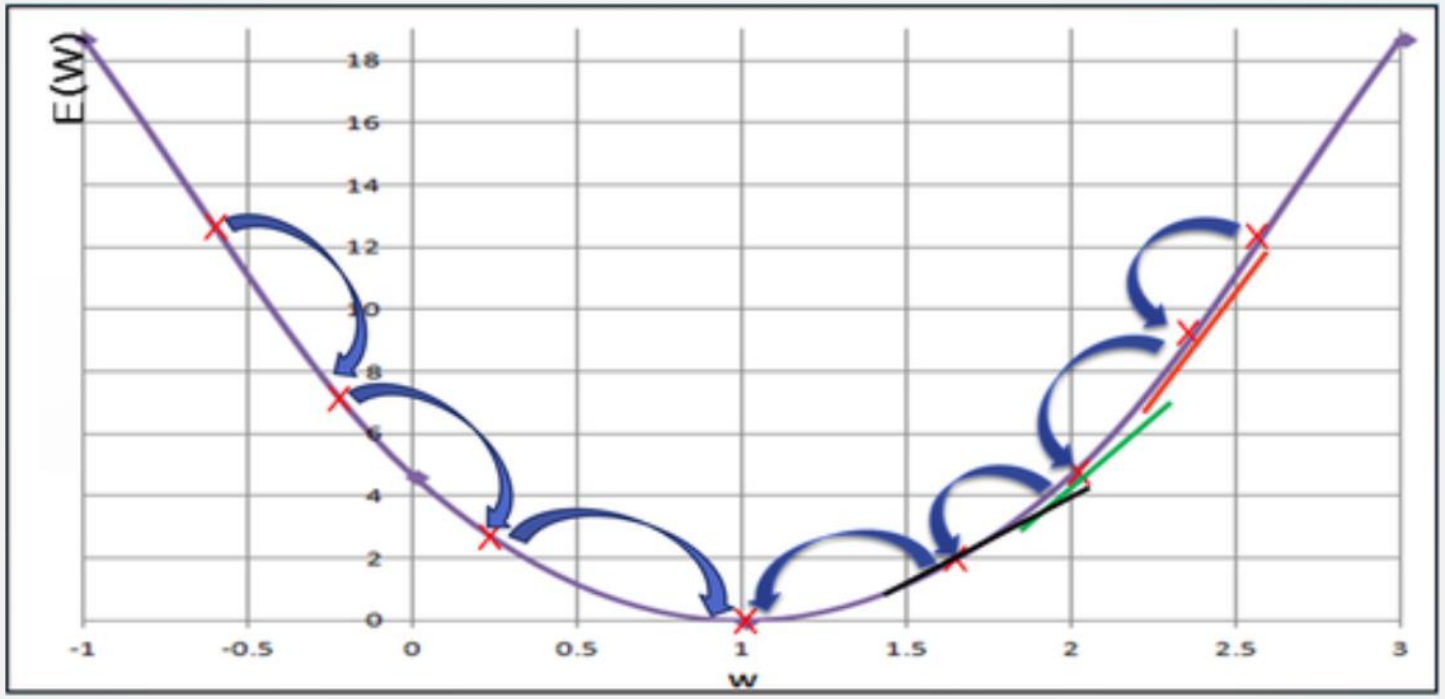
W = -1	$E(-1,0)=\frac{1}{3}\sum_{i=1}^{3}[t_i-(-1*x_i+0)]^2=\frac{[1-(-1*1+0)]^2+[2-(-1*2+0)]^2+[3-(-1*3+0)]^2}{3}=18.7$
W = 0	$E(0,0)=\frac{1}{3}\sum_{i=1}^{3}[t_i-(0*x_i+0)]^2=\frac{[1-(0*1+0)]^2+[2-(0*2+0)]^2+[3-(0*3+0)]^2}{3}=4.67$
W = 1	$E(1,0)=\frac{1}{3}\sum_{i=1}^{3}[t_i-(1*x_i+0)]^2=\frac{[1-(1*1+0)]^2+[2-(1*2+0)]^2+[3-(1*3+0)]^2}{3}=0$
W = 2	$E(2,0)=\frac{1}{3}\sum_{i=1}^{3}[t_i-(2*x_i+0)]^2=\frac{[1-(2*1+0)]^2+[2-(2*2+0)]^2+[3-(2*3+0)]^2}{3}=4.67$
W = 3	$E(3,0)=\frac{1}{3}\sum_{i=1}^{3}[t_i-(3*x_i+0)]^2=\frac{[1-(3*1+0)]^2+[2-(3*2+0)]^2+[3-(3*3+0)]^2}{3}=18.7$

선형 회귀(Linear Regression)-경사하강법

w	E(w)
-1	18.7
0	4.67
1	0
2	4.67
3	18.7



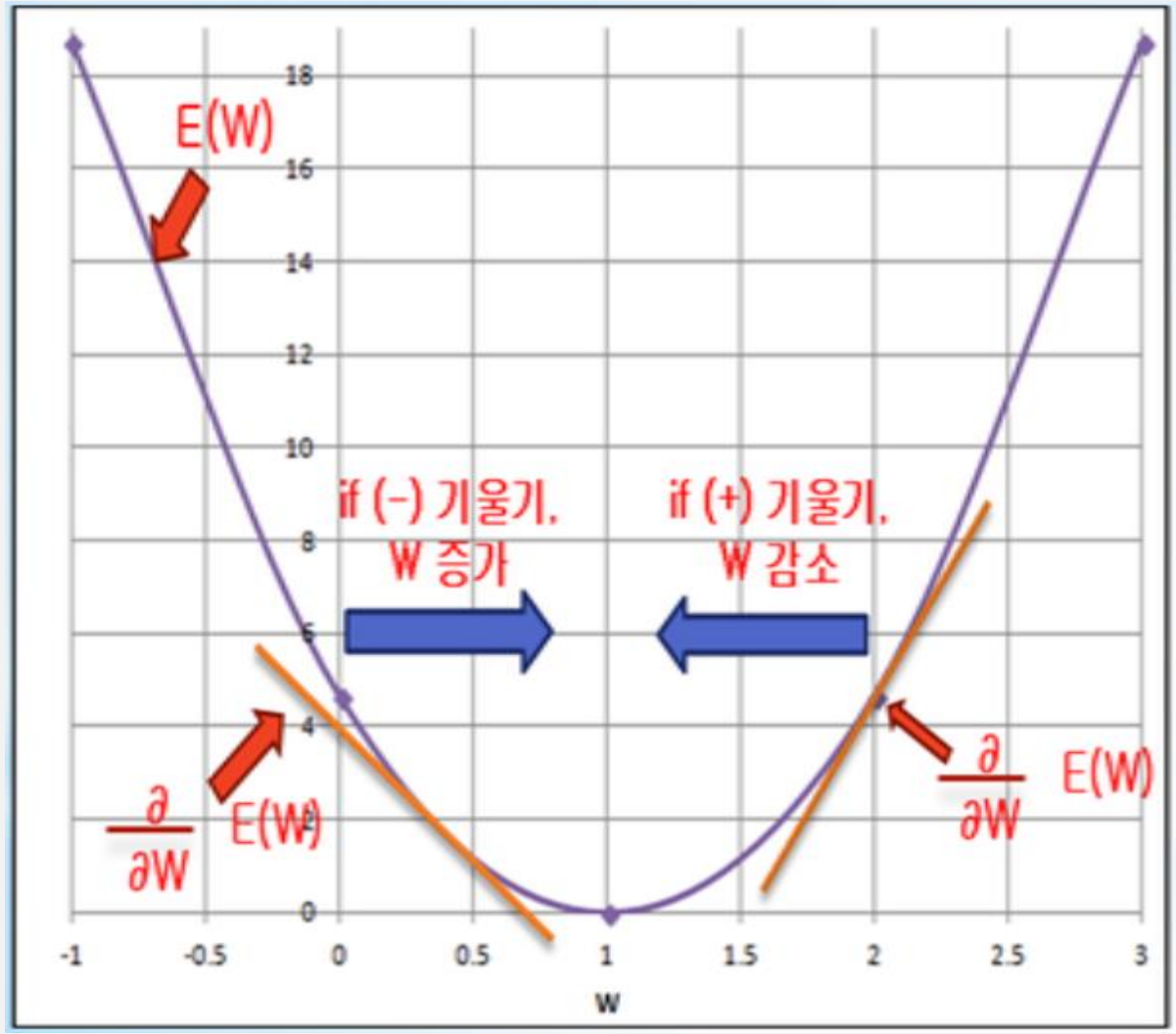
선형 회귀(Linear Regression)-경사하강법



① 임의의 가중치 W 선택 ② 그 W에서의 직선의 기울기를 나타내는 미분 값(해당W에서의 미분, $\frac{\partial E(W)}{\partial W}$ 을 구함
③ 그 미분 값이 작아지는 방향으로 W감소(또는 증가)시켜 나가면
④ 최종적으로 기울기가 더 이상 작아지지 않는 곳을 찾을 수 있는데
그 곳이 손실함수 E(W) 최소값임을 알 수 있음

이처럼 W에서의 직선의 기울기인 미분 값을 이용하여 그 값이 작아지는 방향으로 진행하여
손실함수 최소값을 찾는 방법을 경사하강법(gradient decent algorithm)이라고 함.

선형 회귀(Linear Regression)-경사하강법 W값 구하기



W에서의 편미분 $\frac{\partial E(W)}{\partial W}$ 해당 W에서 기울기(slope)를 나타냄

-〉 $\frac{\partial E(W)}{\partial W}$ 양수(+) 값을 갖는다면
W는 왼쪽으로 이동시켜야만(감소)
손실함수 E(W) 최소값 찾음.

-〉 $\frac{\partial E(W)}{\partial W}$ 양수(-) 값을 갖는다면
W는 오른쪽으로 이동시켜야만(증가)
손실함수 E(W) 최소값 찾음.

$$W = W - a\frac{\partial E(W,b)}{\partial W}$$

a는 학습률(learning rate)이라고 부르며,
W 값의 감소 또는 증가 되는 비율을 나타냄

선형 회귀(Linear Regression)-경사하강법 손실함수E(W,b) 최소값이 되는 W,b

Linear regression 목표는 training data 특성/분포를 가장 잘 나타내는 임의의 직선 $y = Wx + b$ 에서의 [W,b]를 구하는 것

$$y = Wx + b$$

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

손실함수 E(W,b) 최소값을 갖는 W

$$W = W - a\frac{\partial E(W,b)}{\partial W}$$

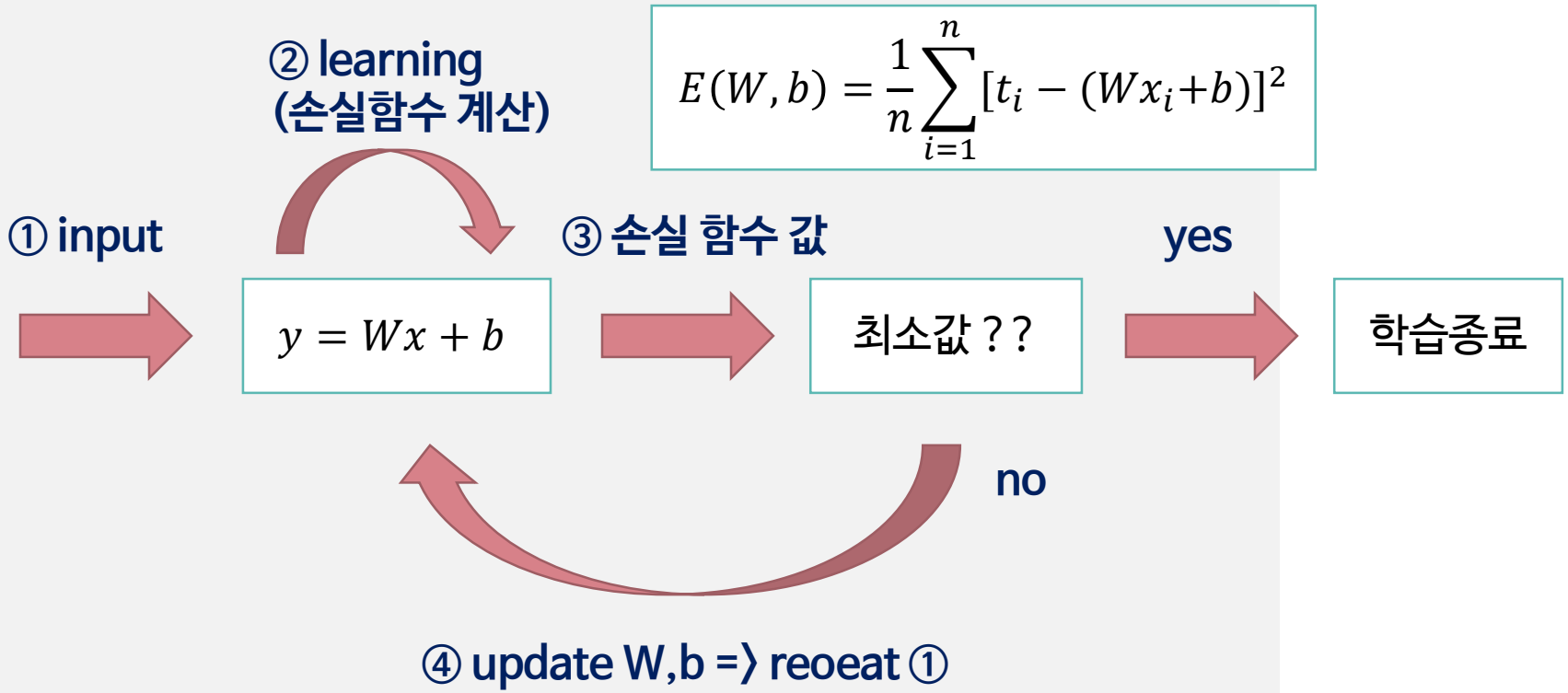
손실함수 E(W,b) 최소값을 갖는 b

$$\mathrm{b} = b - a\frac{\partial E(W,b)}{\partial b}$$

a는 학습률(learning rate)이라고 부르며, W 값의 감소 또는 증가 되는 비율을 나타냄

선형 회귀(Linear Regression)- [W,b] 계산 프로세스

입력(x)	정답(t)
x_1	t_1
x_2	t_2
…	…
…	…
x_n	t_n



$$W = W - a\frac{\partial E(W,b)}{\partial W}$$

$$\mathrm{b} = b - a\frac{\partial E(W,b)}{\partial b}$$

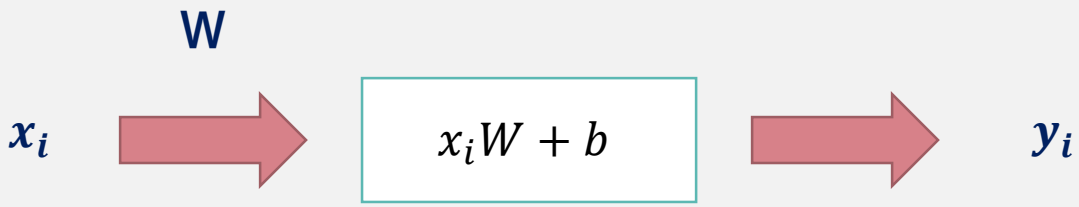
파이썬으로 구현하는 선형회귀

순서	데이터 및 수식	Python 구현
1		슬라이싱(slicing)또는 list comprehension등을 이용하는 입력 x와 정답 t를 numpy 데이터형으로 분리
2	$y = Wx + b$	W = numpy.random.rand(…) b = numpy.random.rand(…)
3	$E(W,b) = \frac{1}{n}\sum_{i=1}^{n}[t_i - (Wx_i+b)]^2$	def loss_func(…): y=numpy.dot(X,W) + b return (numpy.sum((t-y)**2))/(len(x))
4	학습률 a	Learning_rate = 1e-3 or 1e-4 or 1e-5 …
5	$W = W - a\frac{\partial E(W,b)}{\partial W}$ $b = b - a\frac{\partial E(W,b)}{\partial b}$	f = lambda x : loss_func(…) for step in range(6000): # 6000은 임의값 W -= learning_rate * numerical_derivative(f,W) b -= learning_rate * numerical_derivative(f,b)

파이썬으로 입력변수가 1개인 단순 선형회귀 구현

Training data

입력(x)	정답(t)
1	2
2	3
3	4
4	5
5	6



$$X * W + b = Y$$

$$x_1W + b = y_1$$
$$x_2W + b = y_2$$
$$x_3W + b = y_3$$
$$x_4W + b = y_4$$
$$x_5W + b = y_5$$

행렬변환

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} * (W) + b = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{pmatrix}$$

$$(5 * 1) * (1 * 1) = (5 * 1)$$

오차를 계산하기 위해서는 traning data의 모든 입력 x에 대해 각각의 y=Wx+b 계산해야 함
이때 입력 x, 정답 t, 가중치 W 모두를 행렬로 나타낸 후에, 행렬 곱(dot product)을 이용하면 계산 값 y 또한 행렬로 표시되어 모든 입력 데이터에 대해 한번에 쉽게 계산되는 것을 알 수 있음.

파이썬으로 입력변수가 1개인 단순 선형회귀 구현

[1] 학습데이터(Training Data)준비

입력(x)	정답(t)
1	2
2	3
3	4
4	5
5	6

```
import numpy as np

x_data = np.array([1,2,3,4,5]).reshape(5,1)
y_data = np.array([2,3,4,5,6]).reshape(5,1)
```

[2] 임의의 직선 y = Wx + b 정의(임의의 값으로 가중치 W, 바이어스 b 초기화)

$$y = Wx + b$$

```
W = np.random.rand(1,1)
b = np.random.rand(1)
print('W=',W,'W.shape=',W.shape,'b=',b,'b.shape=',b.shape)
```

```
W= [[0.08355463]] W.shape= (1, 1) b= [0.22921152] b.shape= (1,)
```

[3] 손실함수 E(W,b) 정의

$$E(W,b) = \frac{1}{n}\sum_{i=1}^{n}[t_i - (Wx_i + b)]^2$$

```
def loss_func(x,t):
    y = np.dot(x,W) + b
    return (np.sum((t-y)**2))/(len(x))
```

파이썬으로 입력변수가 1개인 단순 선형회귀 구현

[4] 수치미분 numerical_derivative 및 utility 함수 정의

```python
def numerical_derivative(f, x):
    delta_x = 1e-4 # 0.0001
    grad = np.zeros_like(x)

    it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite'])

    while not it.finished:
        idx = it.multi_index
        tmp_val = x[idx]
        x[idx] = float(tmp_val) + delta_x
        fx1 = f(x) # f(x+delta_x)

        x[idx] = tmp_val - delta_x
        fx2 = f(x) # f(x-delta_x)
        grad[idx] = (fx1 - fx2) / (2*delta_x)

        x[idx] = tmp_val
        it.iternext()

    return grad
```

```python
# 손실함수 값 계산 함수
# 입력변수 x, t : numpy type
def error_val(x, t):
    y = np.dot(x,W) + b

    return ( np.sum( (t - y)**2 ) ) / ( len(x) )

# 학습을 마친 후, 임의의 데이터에 대해 미래 값 예측 함수
# 입력변수 x : numpy type
def predict(x):
    y = np.dot(x,W) + b

    return y
```

파이썬으로 입력변수가 1개인 단순 선형회귀 구현

[5] 학습율 (learning rate) 초기화 및 손실함수가 최소가 될 때까지 W, b 업데이트

$$W = W - a\frac{\partial E(W,b)}{\partial W}$$

$$b = b - a\frac{\partial E(W,b)}{\partial b}$$

```
learning_rate = 1e-5  # 1e-2, 1e-3 은 손실함수 값 발산

f = lambda x : loss_func(x_data,t_data)

print("Initial error value = ", error_val(x_data, t_data), " Initial W = ", W.tolist(), ", b = ", b)

for step in  range(10001):

    W -= learning_rate * numerical_derivative(f, W)

    b -= learning_rate * numerical_derivative(f, b)

    if (step % 400 == 0):
        print("step = ", step,'\t', "error value = ", error_val(x_data, t_data),'\t', "W = ", W.tolist(),'\t', " b = ",b )
```

파이썬으로 입력변수가 1개인 단순 선형회귀 구현

[6] 학습결과 및 입력 43에 대한 미래 값 예측

```
Initial error value =  14.531497341731026 Initial W =  [[0.06127899]] , b =  [0.24277486]
step =  0 error value =  8.567424324572617 W =  [[0.31323112]] , b =  [0.2991255]
step =  400 error value =  0.0031556619621834585 W =  [[1.03647892]] , b =  [0.86833176]
step =  800 error value =  0.00020134988937508912 W =  [[1.00921451]] , b =  [0.96674082]
step =  1200 error value =  1.2847313317205923e-05 W =  [[1.00232757]] , b =  [0.99159879]
step =  1600 error value =  8.197345426051741e-07 W =  [[1.00058794]] , b =  [0.99787787]
step =  2000 error value =  5.230391006658732e-08 W =  [[1.00014851]] , b =  [0.99946395]
step =  2400 error value =  3.3372986815317288e-09 W =  [[1.00003751]] , b =  [0.9998646]
step =  2800 error value =  2.129393859004111e-10 W =  [[1.00000948]] , b =  [0.9999658]
step =  3200 error value =  1.358679171083491e-11 W =  [[1.00000239]] , b =  [0.99999136]
step =  3600 error value =  8.669176353436028e-13 W =  [[1.0000006]] , b =  [0.99999782]
step =  4000 error value =  5.531447034379199e-14 W =  [[1.00000015]] , b =  [0.99999945]
step =  4400 error value =  3.529390218991019e-15 W =  [[1.00000004]] , b =  [0.99999986]
step =  4800 error value =  2.251959625896013e-16 W =  [[1.00000001]] , b =  [0.99999996]
step =  5200 error value =  1.4368836600933026e-17 W =  [[1.]] , b =  [0.99999999]
step =  5600 error value =  9.168174860014702e-19 W =  [[1.]] , b =  [1.]
step =  6000 error value =  5.849842421894404e-20 W =  [[1.]] , b =  [1.]
step =  6400 error value =  3.732523315847544e-21 W =  [[1.]] , b =  [1.]
step =  6800 error value =  2.381576491942223e-22 W =  [[1.]] , b =  [1.]
step =  7200 error value =  1.5196071454178155e-23 W =  [[1.]] , b =  [1.]
step =  7600 error value =  9.71206142905894e-25 W =  [[1.]] , b =  [1.]
step =  8000 error value =  6.227887241625266e-26 W =  [[1.]] , b =  [1.]
```

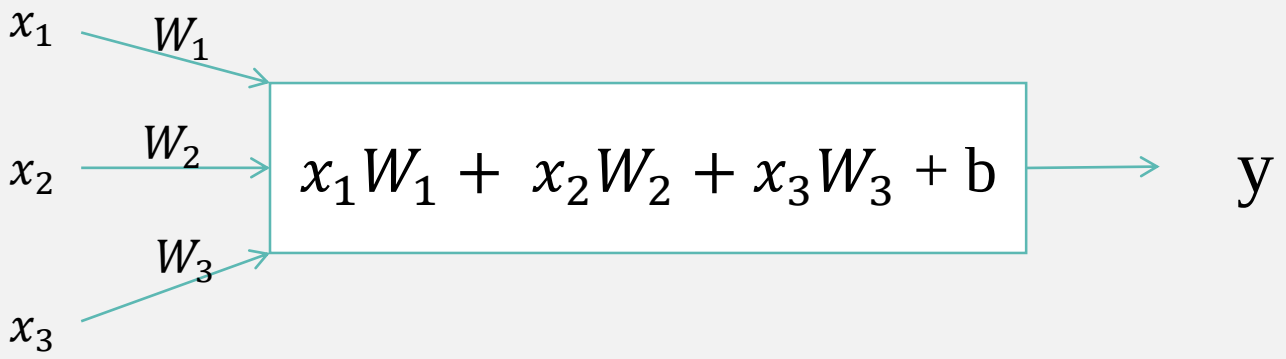
```
predict(43)
```

```
array([[44.]])
```

Multi-variable regression -concept

```
[[ 73.,  80.,  75., 152.],
 [ 93.,  88.,  93., 185.],
 [ 89.,  91.,  90., 180.],
 [ 96.,  98., 100., 196.],
 [ 73.,  66.,  70., 142.],
 [ 53.,  46.,  55., 101.],
 [ 69.,  74.,  77., 149.],
 [ 47.,  56.,  60., 115.],
 [ 87.,  79.,  90., 175.],
 [ 79.,  70.,  88., 164.],
 [ 69.,  70.,  73., 141.],
 [ 70.,  65.,  74., 141.],
 [ 93.,  95.,  91., 184.],
 [ 79.,  80.,  73., 152.],
 [ 70.,  73.,  78., 148.],
 [ 93.,  89.,  96., 192.],
 [ 78.,  75.,  68., 147.],
 [ 81.,  90.,  93., 183.],
 [ 88.,  92.,  86., 177.],
 [ 78.,  83.,  77., 159.],
 [ 82.,  86.,  90., 177.],
 [ 86.,  82.,  89., 175.],
 [ 78.,  83.,  85., 175.],
 [ 76.,  83.,  71., 149.],
 [ 96.,  93.,  95., 192.]]
```

(25*4)



$$X * W + b = Y$$

```
73.,  80.,  75.,
93.,  88.,  93.,
89.,  91.,  90.,
96.,  98., 100.,
73.,  66.,  70.,
53.,  46.,  55.,
69.,  74.,  77.,
47.,  56.,  60.,
87.,  79.,  90.,
```

(25*3)

$$* \begin{bmatrix} W_1 \\ W_2 \\ W_3 \end{bmatrix} + b =$$

(3*1)

```
152.
185.
180.
196.
142.
101.
149.
115.
175.
```

(25*1)

Multi-variable regression -example

[1] 학습데이터(Training Data)준비

x_1	x_2	x_3	t
73	80	75	152
93	88	93	185
…	…	…	…
96	93	95	192

```
import numpy as np

loaded_data = np.loadtxt('./data-01-test-score.csv', delimiter=',', dtype=np.float32)

x_data = loaded_data[ :, 0:-1]
t_data = loaded_data[ :, [-1]]

# 데이터 차원 및 shape 확인
print("x_data.ndim = ", x_data.ndim, ", x_data.shape = ", x_data.shape)
print("t_data.ndim = ", t_data.ndim, ", t_data.shape = ", t_data.shape)
```

[2] 임의의 직선 $y = x_1W_1 + x_2W_2 + x_3W_3 + \mathrm{b}$ 정의

$$y = x_1W_1 + x_2W_2 + x_3W_3 + \mathrm{b}$$

```
W = np.random.rand(3,1)  # 3X1 행렬
b = np.random.rand(1)
print("W = ", W.tolist(), ", W.shape = ", W.shape, ", b = ", b, ", b.shape = ", b.shape)
```

```
W =  [[0.2685320323868211], [0.860709913848709], [0.17834996688706506]] , W.shape =  (3, 1) , b =  [0.35656829] , b.shape =  (1,)
```

Multi-variable regression -example

[3] 손실함수 E(W,b) 정의

$$E(W,b) = \frac{1}{n}\sum_{i=1}^{n}[t_i - (Wx_i + b)]^2$$

```
def loss_func(x, t):
    y = np.dot(x,W) + b

    return ( np.sum( (t - y)**2 ) ) / ( len(x) )
```

[4] 수치미분 numerical_derivative 및 utility 함수 정의

```
def numerical_derivative(f, x):
    delta_x = 1e-4 # 0.0001
    grad = np.zeros_like(x)

    it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite'])

    while not it.finished:
        idx = it.multi_index
        tmp_val = x[idx]
        x[idx] = float(tmp_val) + delta_x
        fx1 = f(x) # f(x+delta_x)

        x[idx] = tmp_val - delta_x
        fx2 = f(x) # f(x-delta_x)
        grad[idx] = (fx1 - fx2) / (2*delta_x)

        x[idx] = tmp_val
        it.iternext()

    return grad
```

```
# 손실함수 값 계산 함수
# 입력변수 x, t : numpy type
def error_val(x, t):
    y = np.dot(x,W) + b

    return ( np.sum( (t - y)**2 ) ) / ( len(x) )

# 학습을 마친 후, 임의의 데이터에 대해 미래 값 예측 함수
# 입력변수 x : numpy type
def predict(x):
    y = np.dot(x,W) + b

    return y
```

Multi-variable regression -example

[5] 학습율 (learning rate) 초기화 및 손실함수가 최소가 될 때까지 W, b 업데이트

$$W = W - a\frac{\partial E(W,b)}{\partial W}$$

$$\mathrm{b} = b - a\frac{\partial E(W,b)}{\partial b}$$

```
learning_rate = 1e-5  # 1e-2, 1e-3 은 손실함수 값 발산

f = lambda x : loss_func(x_data,t_data)

print("Initial error value = ", error_val(x_data, t_data),'\t', " Initial W = ", W.tolist(),'\t', " b = ", b)

for step in  range(10001):

    W -= learning_rate * numerical_derivative(f, W)

    b -= learning_rate * numerical_derivative(f, b)

    if (step % 400 == 0):
        print("step = ", step,'\t', "error value = ", error_val(x_data, t_data),'\t', "W = ", W.tolist(),'\t', " b = ",b )
```

Multi-variable regression -example

[6] 학습결과 및 입력 [100,98,81]에 대한 미래 값 예측

```
Initial error value =  6.126013071497288          Initial W =  [[0.3559861173736011], [0.5299252179276218], [1.1256768052774448]]        b =  [0.15076713]
step =  0       error value =  6.126012772088335        W =  [[0.35598611735513086], [0.5299252227840896], [1.1256768213842834]]        b =  [0.1507654]
step =  400     error value =  6.125893027016563        W =  [[0.35598610996720864], [0.5299271652211962], [1.1256832636214402]]        b =  [0.15007334]
step =  800     error value =  6.125773318881061        W =  [[0.35598610258040714], [0.5299291073587217], [1.1256897048649372]]        b =  [0.14938139]
step =  1200    error value =  6.125653647670549        W =  [[0.35598609519475233], [0.529931049196677], [1.1256961451149372]]         b =  [0.14868955]
step =  1600    error value =  6.1255340133734615       W =  [[0.3559860878102312], [0.5299329907351272], [1.125702584371587]]          b =  [0.14799781]
step =  2000    error value =  6.125414415978546        W =  [[0.35598608042685564], [0.5299349319741105], [1.125709022635039]]         b =  [0.14730618]
step =  2400    error value =  6.125294855474409        W =  [[0.35598607304462626], [0.5299368729136645], [1.1257154599054517]]        b =  [0.14661465]
step =  2800    error value =  6.125175331849651        W =  [[0.35598606566353974], [0.5299388135538572], [1.1257218961829583]]        b =  [0.14592323]
step =  3200    error value =  6.125055845092866        W =  [[0.35598605828357804], [0.5299407538947284], [1.1257283314677344]]        b =  [0.14523192]
step =  3600    error value =  6.12493639519272         W =  [[0.3559860509047483], [0.5299426939363145], [1.1257347657599441]]         b =  [0.14454072]
step =  4000    error value =  6.124816982137837        W =  [[0.3559860435270664], [0.5299446336786706], [1.1257411990597044]]         b =  [0.14384962]
step =  4400    error value =  6.1246976059168325       W =  [[0.3559860361505258], [0.5299465731218279], [1.125747631367201]]          b =  [0.14315863]
step =  4800    error value =  6.124578266518383        W =  [[0.35598602877511865], [0.5299485122658613], [1.1257540626825546]]        b =  [0.14246774]
step =  5200    error value =  6.124458963931095        W =  [[0.3559860214008706], [0.5299504511107783], [1.1257604930059564]]         b =  [0.14177696]
step =  5600    error value =  6.124339698143642        W =  [[0.3559860140277222], [0.5299523896566547], [1.125766922337538]]          b =  [0.14108629]
step =  6000    error value =  6.124220469144617        W =  [[0.35598600665572766], [0.5299543279035286], [1.1257733506774519]]        b =  [0.14039573]
step =  6400    error value =  6.124101276922732        W =  [[0.3559859992848653], [0.5299562658514443], [1.1257797780258585]]         b =  [0.13970527]
step =  6800    error value =  6.123982121466631        W =  [[0.35598599191512004], [0.5299582035004611], [1.1257862043829119]]        b =  [0.13901492]
step =  7200    error value =  6.123863002764961        W =  [[0.35598598454654384], [0.5299601408505863], [1.1257926297487584]]        b =  [0.13832467]
step =  7600    error value =  6.123743920806396        W =  [[0.3559859771791076], [0.5299620779018871], [1.1257990541235496]]         b =  [0.13763453]
step =  8000    error value =  6.1236248755795915       W =  [[0.35598596981279573], [0.529964014654438], [1.125805477507432]]          b =  [0.1369445]
step =  8400    error value =  6.123505867073229        W =  [[0.35598596244763553], [0.5299659511082634], [1.1258118999005455]]        b =  [0.13625457]
step =  8800    error value =  6.1233868952759485       W =  [[0.35598595508361525], [0.5299678872633975], [1.125818321303077]]         b =  [0.13556475]
step =  9200    error value =  6.1232679601764595       W =  [[0.3559859477207163], [0.5299698231199035], [1.1258247417151703]]         b =  [0.13487504]
step =  9600    error value =  6.123149061763428        W =  [[0.3559859403589675], [0.5299717586778214], [1.1258311611369616]]         b =  [0.13418543]
step =  10000   error value =  6.1230302000255765       W =  [[0.35598593299832126], [0.5299736939372174], [1.1258375795686277]]        b =  [0.13349593]
```

```
test_data = np.array([100, 98, 81])

predict(test_data)
```

```
array([178.86235518])
```

Logistic Regression - Classification

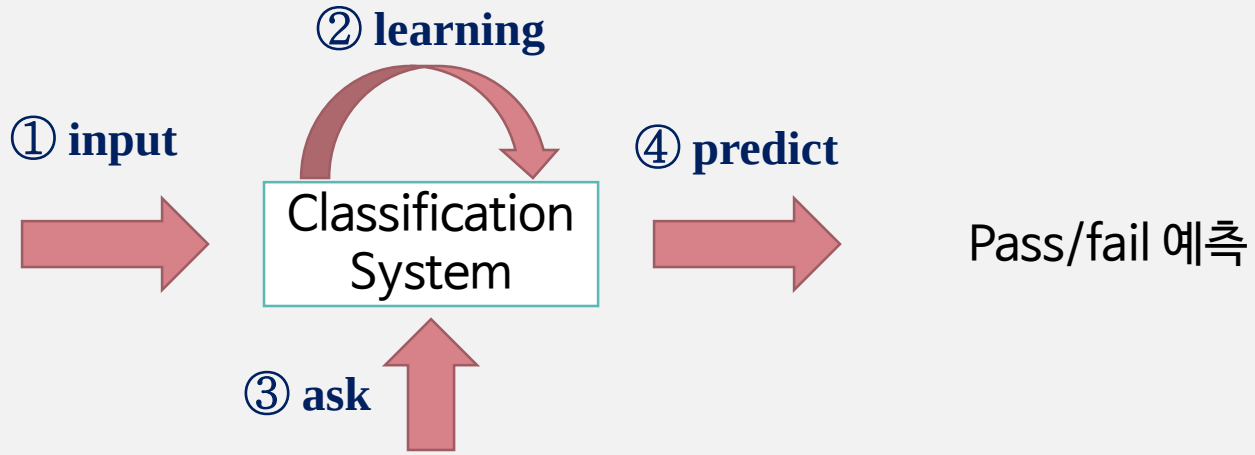
분류(Classification)

- Training Data 특성과 관계 등을 파악 한 후에 미지의 입력 데이터에 대해서 결과가 어떤 종류의 값으로 분류 될 수 있는지 예측하는 것

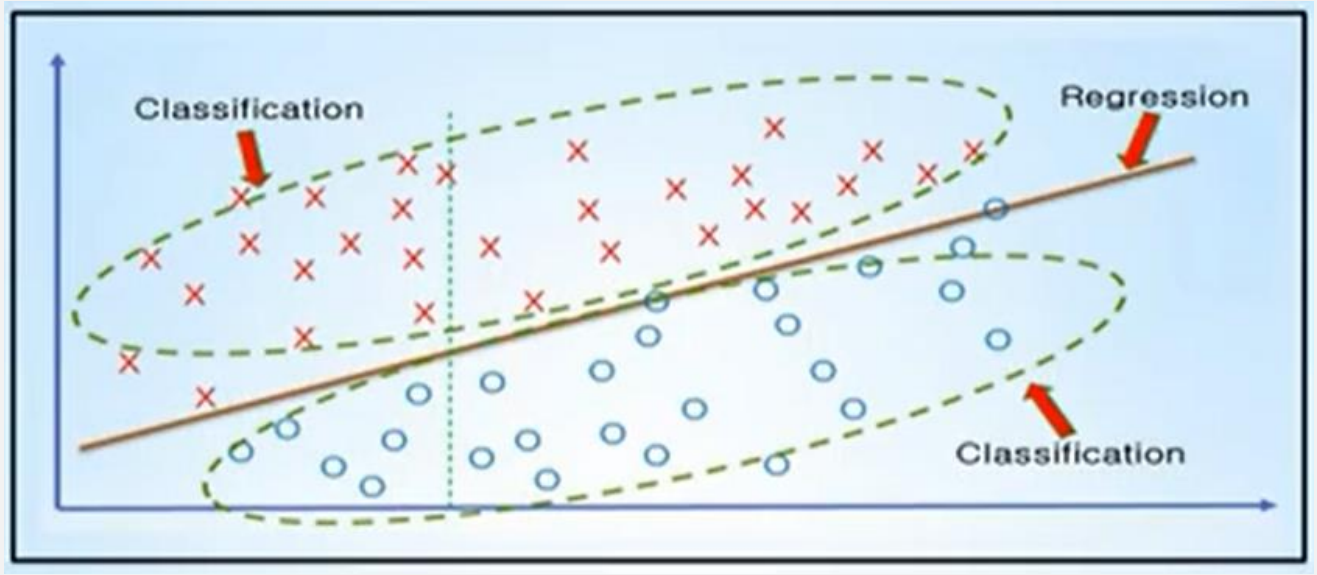
(예: 스팸문자 분류 [spam(1) or ham(0)], 암 판별 [악성종양(1) or 종양(0)]

공부시간 (x)	시험성적(t)
9	fail
14	fail
21	pass
27	pass
32	pass
37	pass

Training Data



Test Data : 공부시간이 43시간일 경우 pass?



Logistic Regression 알고리즘은

Traning Data 특성고 분포를 나타내는 최적의 직선을 찾고(Linear Regression)

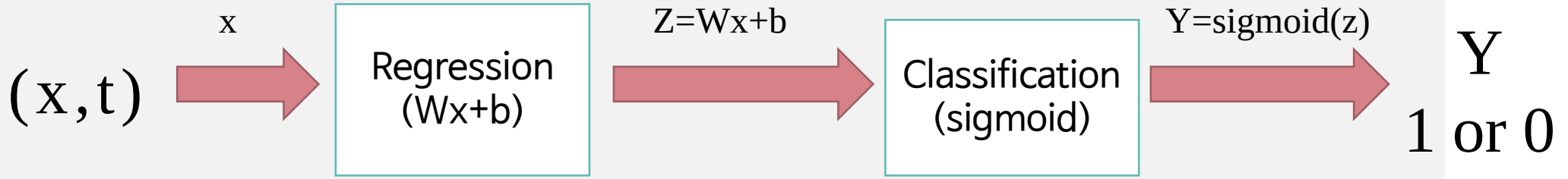
그 직선을 기준으로 데이터를 위(1) 또는 아래(0) 등으로 분류(Classification)해주는 알고리즘

-〉 이러한 Logistic Regression은 Classification 알고리즘 중에서도 정확도가 높은 알고리즘으로 알려져 있어서

Deep learning에서 기본 Component로 사용되고 있음.

(x, t) → Regression System → Classification System → True or False

Classification - sigmoid function



출력값 y가 1 또는 0만을 가져야만 하는 분류(classification)시스템에서, 함수 값으로 0-1사이의 값을 가지는 Sigmoid 함수를 사용할 수 있음

-〉 즉, linear regression 출력 Wx+b가 어떤 값을 갖더라도, 출력 함수로 sigmoid를 사용해서

① sigmoid 계산 값이 0.5보다 크면 결과로 1이 나올 확률이 높다는 것이기 때문에 출력 값 y는 1을 정의하고

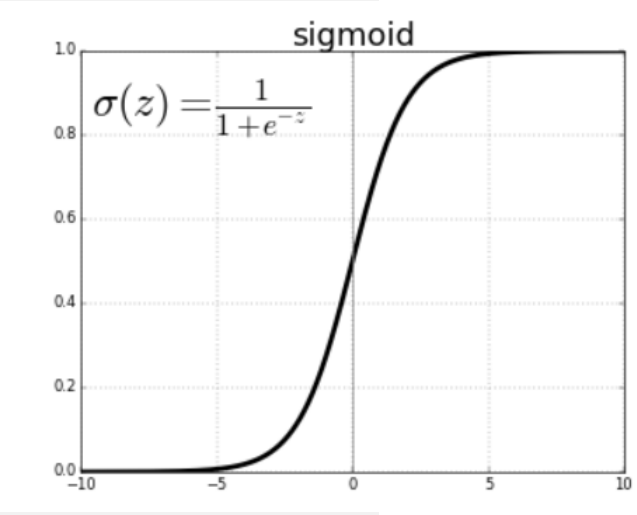
② sigmoid 계산 값이 0.5미만이면 결과로 0이 나올 확률이 높다는 것이므로 출력 값 y는 0정의하여 classification 시스템을 구현할 수 있음.

* sigmoid 함수의 실제 계산 값 sigmoid(z)는 결과가 나타날 확률을 의미함

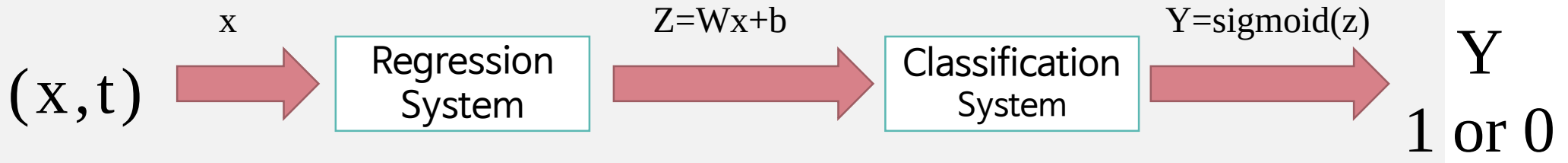
$$y = \frac{1}{1 + e^{-(Wx+b)}}$$

$$z = Wx + b$$

$$y = sigmoid(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$



Classification - 손실함수(loss function) W, b



분류시스템(classification) 최종 출력 값 y는 sigmoid함수에 의해 논리적으로 1 또는 0 값을 가지기 때문에 연속값을 갖는 선형회귀 때와는 다른 손실함수 필요함

$$y = \frac{1}{1 + e^{-(Wx+b)}} \ , t_i = 0 \ or \ 1$$

$$E(W,b) = -\sum_{i=1}^{n} \{t_i log y_i + (1 - t_i) \log(1 - y_i)\}^2$$

손실함수 E(W,b) 최소값을 갖는 W

$$W = W - a\frac{\partial E(W,b)}{\partial W}$$

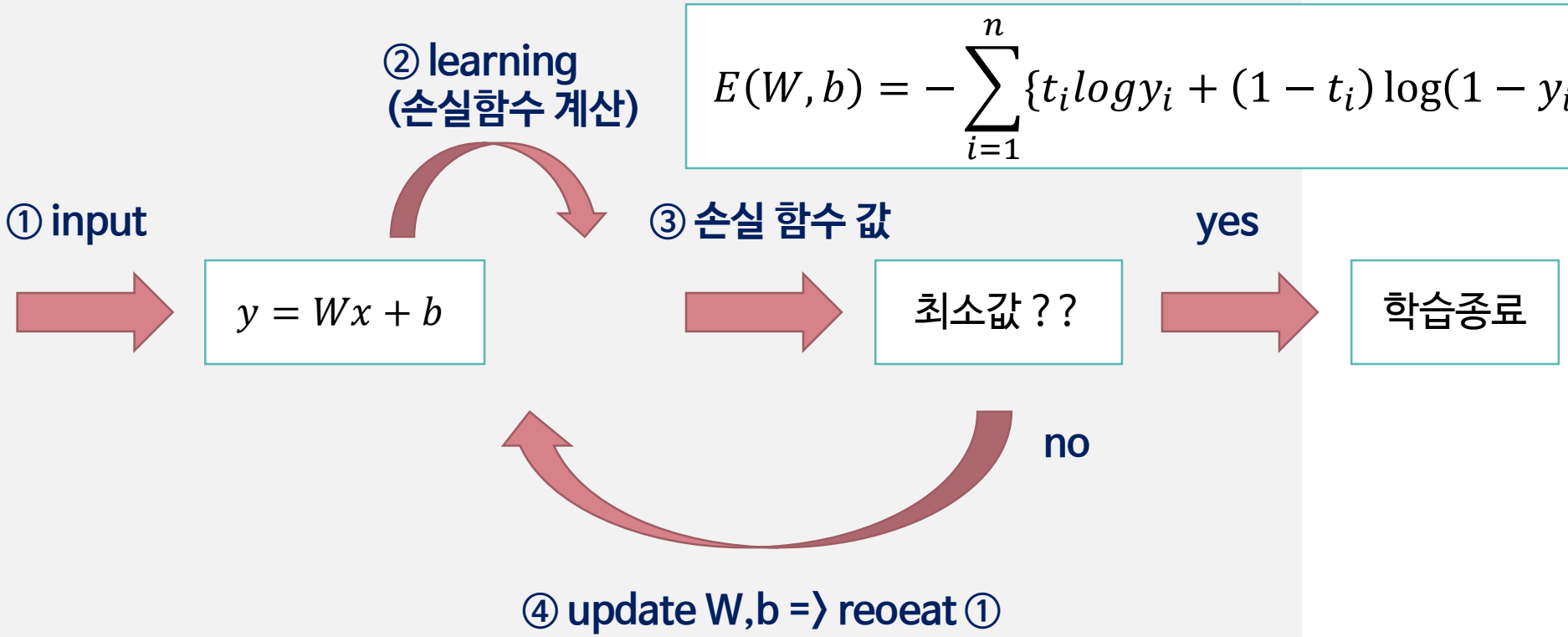
손실함수 E(W,b) 최소값을 갖는 b

$$\mathrm{b} = b - a\frac{\partial E(W,b)}{\partial b}$$

a는 학습률(learning rate)이라고 부르며, W 값의 감소 또는 증가 되는 비율을 나타냄

Classification 에서의 [W, b] 계산 프로세스

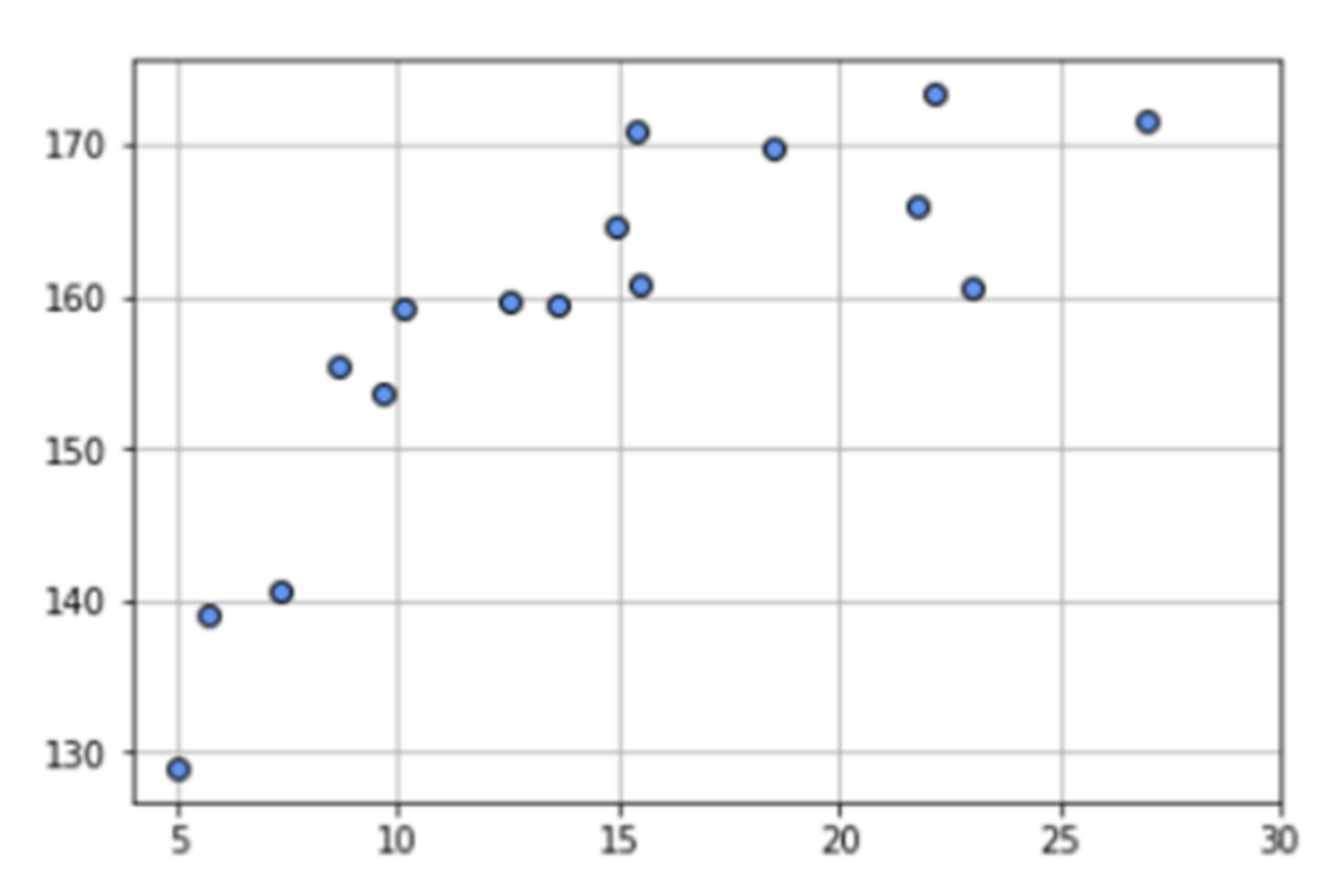
입력(x)	정답(t)
x_1	t_1
x_2	t_2
…	…
…	…
x_n	t_n



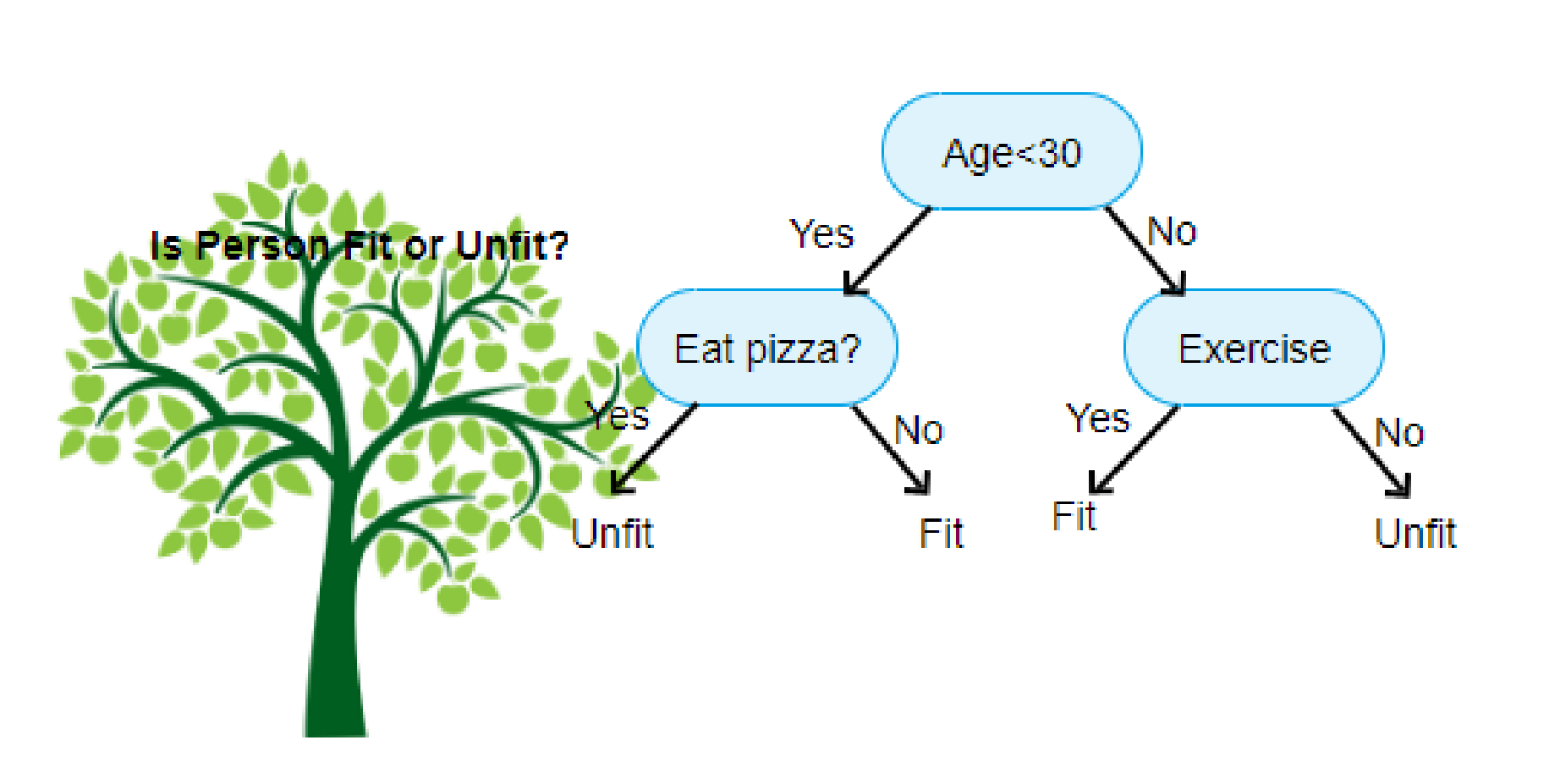
$$W = W - a\frac{\partial E(W,b)}{\partial W}$$

$$\mathrm{b} = b - a\frac{\partial E(W,b)}{\partial b}$$

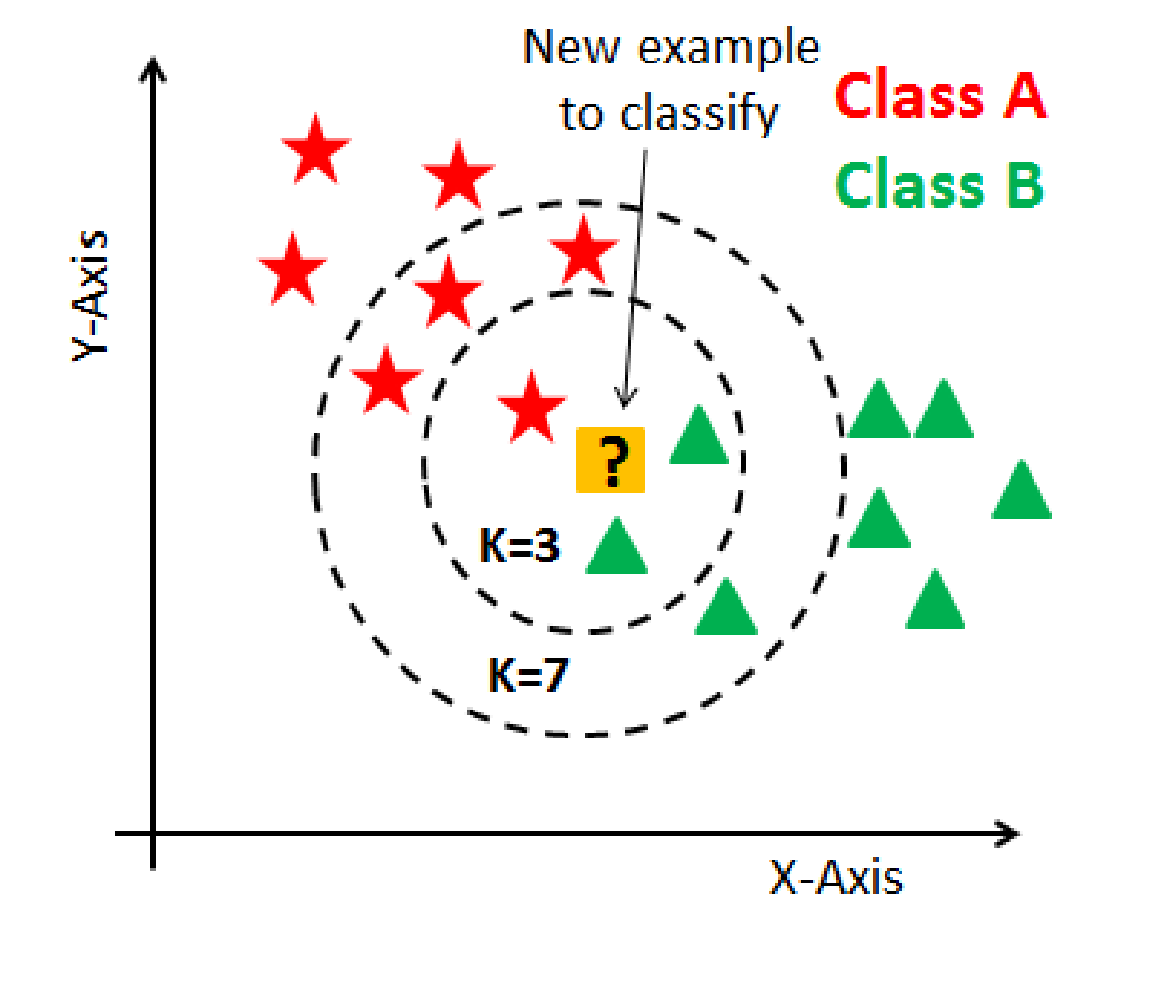
지도학습(회귀)



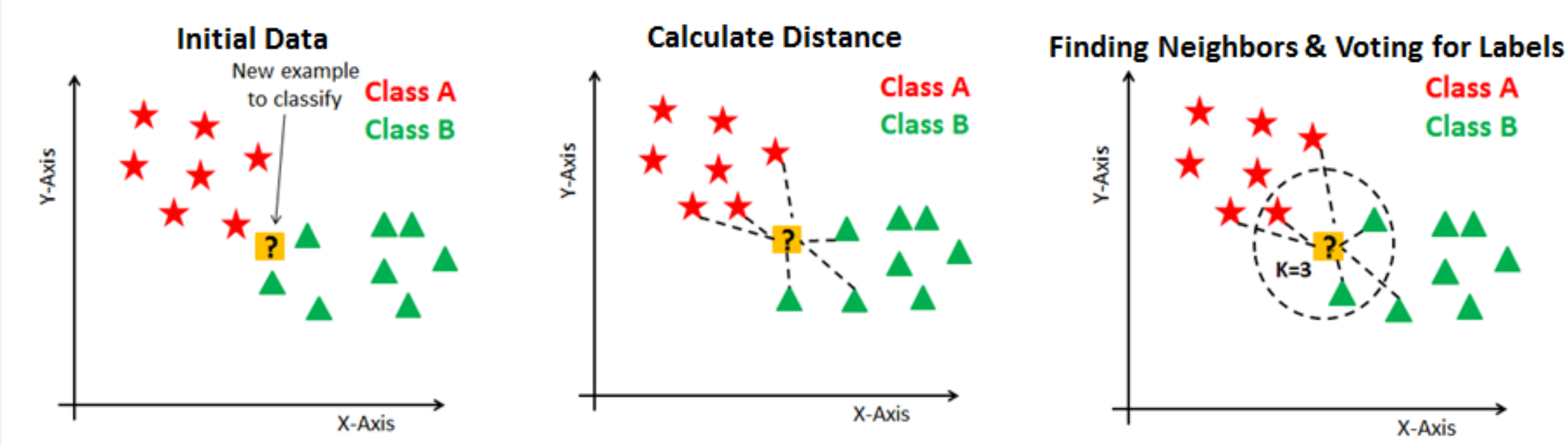
지도학습(분류) – 의사결정 트리(Decision Tree)



지도학습(분류) - K-nearest neighbor classifier(k-최근접 이웃 분류기)



지도학습(분류) - K-nearest neighbor classifier(k-최근접 이웃 분류기)



비지도학습(군집) - K-means Clustering(평균 군집)

KMeans 클러스터링 알고리즘은 n개의 중심점을 찍은 후에, 이 중심점에서 각 점간의 거리의 합이 가장 최소화가 되는 중심점 n의 위치를 찾고, 이 중심점에서 가까운 점들을 중심점을 기준으로 묶는 클러스터링 알고리즘이다.