그래프 그리기

Contents

CHAPTER 03 그래프 그리기

머신러닝을 이해하는 데 필요한 최소한의 프로그래밍 지식을 알아본다

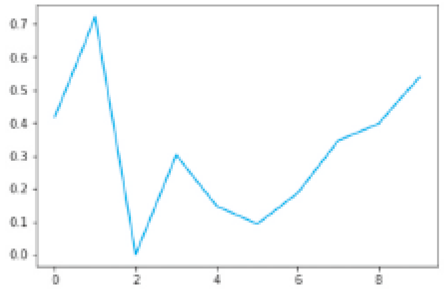
1.1 임의의 그래프 그리기

- matplotlib의 pyplot 라이브러리를 import하고, 이를 plt라는 별칭을 만들어 사용함.
- 주피터 노트북에서 그래프를 표시하기 위해 %matplotlib inline 명령을 추가함.

In

```
# 리스트 1-(1)
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# data 작성
np.random.seed(1) # 난수를 고정
x = np.arange(10)
y = np.random.rand(10)

# 그래프 표시
plt.plot(x, y) # 꺾은선 그래프를 등록
plt.show() # 그래프 그리기
```

Out



1.2 프로그램 리스트 규칙

- 리스트 번호는 1-(1), 1-(2), 1-(3), 2-(1)과 같음.
- 괄호 앞의 숫자가 같은 리스트는 변수를 공유
- 1-(1)에서 만든 변수와 함수는 1-(2)와 1-(3)에서 사용할 수 있음.
- 이력을 메모리에서 삭제하려면 다음과 같은 명령을 입력, 실행함.

In
```
%reset
```

Out
```
Once deleted, variables cannot be recovered. Proceed (y/[n])?
```

1.3 3차 함수 f (x) = (x-2) x (x+2) 그리기

- 먼저 함수 f (x)를 정의함

In
```
# 리스트 2-(1)
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
def f(x):
    return (x - 2) * x * (x + 2)
```

- 함수의 x에 숫자를 넣으면 다음처럼 대응하는 f의 값이 돌아옴

In
```
# 리스트 2-(2)
print(f(1))
```

Out
```
-3
```

- x는 ndarray 배열이며 각각에 대응한 f를 한꺼번에 ndarray로 돌려줌

In
```
# 리스트 2-(3)
print(f(np.array([1, 2, 3])))
```

Out
```
[-3 0 15]
```

1.4 그리는 범위를 결정하기

- 그래프를 그리는 x의 범위를 -3에서 3까지로 하고, 그 범위에서 계산한 x를 간격 0.5로 정의함.

In
```
# 리스트 2-(4)
x = np.arange(-3, 3.5, 0.5)
print(x)
```

Out
```
[-3.  -2.5 -2.  -1.5 -1.  -0.5  0.   0.5  1.   1.5  2.   2.5  3. ]
```

- linspace(n1, n2, n)하면 범위 n1에서 n2 사이를 일정 간격 n개의 구간으로 나눈 값을 돌려줌.

In
```
# 리스트 2-(5)
x = np.linspace(-3, 3, 10)
print(np.round(x, 2))
```

Out
```
[-3.   -2.33 -1.67 -1.   -0.33  0.33  1.    1.67  2.33  3.  ]
```

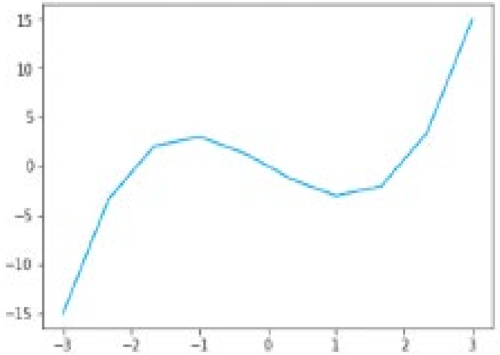
1.5 그래프 그리기

- x를 사용하여 f(x)의 그래프를 그려보면 다음처럼 실행 결과가 나타남.

In

```
# 리스트 2-(6)
plt.plot(x, f(x))
plt.show()
```

Out



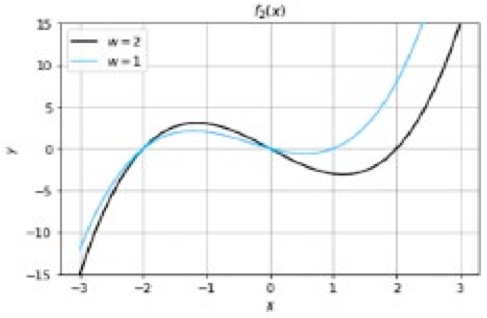
1.6 그래프를 장식하기

- 그래프를 조금 더 손질하여, 매끄럽고 부드럽게 그려봄.

In

```
# 리스트 2-(7)
# 함수를 정의

def f2(x, w):
    return (x - w) * x * (x + 2) # (A) 함수 정의


# x를 정의
x = np.linspace(-3, 3, 100) # (B) x를 100 분할하기

# 차트 묘사
plt.plot(x, f2(x, 2), color='black', label='$w=2$') #(C)
plt.plot(x, f2(x, 1), color='cornflowerblue',
         label='$w=1$')    #(D)
plt.legend(loc="upper left") # (E) 범례 표시
plt.ylim(-15, 15)         # (F) y 축의 범위
plt.title('$f_2(x)$')     # (G) 제목
plt.xlabel('$x$')         # (H) x 라벨
plt.ylabel('$y$')         # (I) y 라벨
plt.grid(True)            # (J) 그리드
plt.show()
```

Out



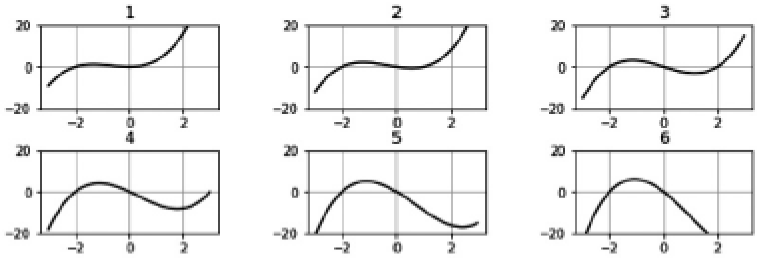
1.7 그래프를 여러 개 보여주기

- 여러 그래프를 나란히 표시하려면 plt.subplot(n1, n2, n)를 사용하여 전체를 세로 n1, 가로 n2로 나눈 n번째에 그래프가 그려짐.

In

```
# 리스트 2-(9)
plt.figure(figsize=(10, 3)) # (A) figure 지정
plt.subplots_adjust(wspace=0.5, hspace=0.5) # (B) 그래프의 간격을 지정
for i in range(6):
    plt.subplot(2, 3, i + 1) # (C) 그래프 묘사의 위치를 지정
    plt.title(i + 1)
    plt.plot(x, f2(x, i), 'k')
    plt.ylim(-20, 20)
    plt.grid(True)
plt.show()
```

Out



2.1 이변수 함수

- 두 변수의 함수, 즉 이변수 함수를 그림으로 나타내기.

$$f(x_0, x_1) = (2x_0^2 + x_1^2)\exp(-(2x_0^2 + x_1^2))$$

- 먼저 앞의 함수를 f3로 정의함. 그리고 다양한 x0과 x1 값에 대한 f3 의 값을 계산함.

In
```
# 리스트 3-(1)
import numpy as np
import matplotlib.pyplot as plt

# 함수 f3을 정의
def f3(x0, x1):
    r = 2 * x0**2 + x1**2
    ans = r * np.exp(-r)
    return ans

# x0, x1에서 각각 f3을 계산
xn = 9
x0 = np.linspace(-2, 2, xn) # (A)
x1 = np.linspace(-2, 2, xn) # (B)
y = np.zeros((len(x0), len(x1))) # (C)
for i0 in range(xn):
    for i1 in range(xn):
        y[i1, i0] = f3(x0[i0], x1[i1]) # (D)
```

SECTION 02 3차원 그래프 그리기

- xn = 9이므로 다음 명령을 실행하면 x0은 9개의 요소로 구성. x1도 x0과 같음.

In
```
# 리스트 3-(2)
print(x0)
```

Out
```
[-2. -1.5 -1 -0.5 0. 0.5 1. 1.5 2.]
```

- 행렬 y는 print(y)로 표시할 수 있지만 복잡해 보임, 그래서 넘파이 함수 round를 사용함.

In
```
# 리스트 3-(3)
print(np.round(y, 1))
```

Out
```
[[0.  0.  0.  0.  0.1 0.  0.  0.  0. ]
 [0.  0.  0.1 0.2 0.2 0.2 0.1 0.  0. ]
 [0.  0.  0.1 0.3 0.4 0.3 0.1 0.  0. ]
 [0.  0.  0.2 0.4 0.2 0.4 0.2 0.  0. ]
 [0.  0.  0.3 0.3 0.  0.3 0.3 0.  0. ]
 [0.  0.  0.2 0.4 0.2 0.4 0.2 0.  0. ]
 [0.  0.  0.1 0.3 0.4 0.3 0.1 0.  0. ]
 [0.  0.  0.1 0.2 0.2 0.2 0.1 0.  0. ]
 [0.  0.  0.  0.  0.1 0.  0.  0.  0. ]]
```

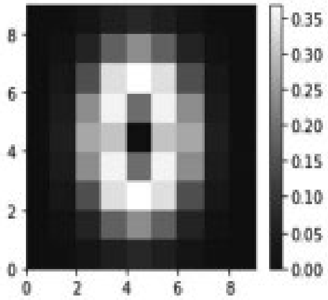
2.2 수치를 색으로 표현하기: pcolor

- 2차원 행렬의 요소를 색상으로 표현함.
- plt.pcolor(2차원ndarray) 명령을 사용하여 실행 결과를 확인 함.
- (A)는 색상을 회색 음영, (B)는 행렬을 색상, (C)는 행렬 옆에 컬러 바를 나타내는 명령문 임.

In

```
# 리스트 3-(4)
plt.figure(figsize=(3.5, 3))
plt.gray() # (A)
plt.pcolor(y) # (B)
plt.colorbar() # (C)
plt.show()
```

Out



SECTION 02 3차원 그래프 그리기

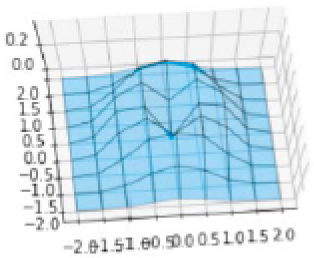
2.3 함수의 표면을 표시: surface

- surface (서피스)로 불리는 3차원의 입체 그래프로 표시하는 방법임.

In

```
#리스트 3-(5)
from mpl_toolkits.mplot3d import Axes3D # (A)

xx0, xx1 = np.meshgrid(x0, x1) # (B)

plt.figure(figsize=(5, 3.5))
ax = plt.subplot(1, 1, 1, projection='3d') # (C)
ax.plot_surface(xx0, xx1, y, rstride=1, cstride=1, alpha=0.3,
                color='blue', edgecolor='black') # (D)
ax.set_zticks((0, 0.2)) # (E)
ax.view_init(75, -95) # (F)
plt.show()
```

Out



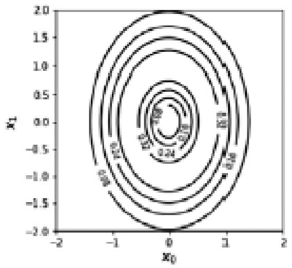
2.4 등고선으로 표시: contour

- 함수의 높이를 알아보려면 등고선 플롯이 편리함.

In

```
# 리스트 3-(9)
xn = 50
x0 = np.linspace(-2, 2, xn)
x1 = np.linspace(-2, 2, xn)

y = np.zeros((len(x0), len(x1)))
for i0 in range(xn):
    for i1 in range(xn):
        y[i1, i0] = f3(x0[i0], x1[i1])

xx0, xx1 = np.meshgrid(x0, x1) # (A)

plt.figure(1, figsize=(4, 4))
cont = plt.contour(xx0, xx1, y, 5, colors='black') # (B)
cont.clabel(fmt='%3.2f', fontsize=8) # (C)
plt.xlabel('$x_0$', fontsize=14)
plt.ylabel('$x_1$', fontsize=14)
plt.show()
```

Out



- 해상도를 50×50으로 하여 xx0, xx1, y를 생성함.
- plt.contour (xx0, xx1, y, 5, colors='black')로 등고선 플롯을 작성함.
- plt.contour의 반환값을 cont에 저장하고,
- cont.clabel(fmt='%3.2f', fontsize=8)로 설정하여,
  각 등고선에 숫자를 넣을 수 있음.
- fmt='%3.2f'로 숫자 형식을 지정함. fontsize 옵션은 문자의 크기를 결정함.

지도 학습: 회귀

Contents

Contents

CHAPTER 05 지도 학습: 회귀

SECTION 01 1차원 입력 직선 모델

- 나이 x와 키 t가 세트로 된 데이터를 생각함.
- 이를 묶어 다음과 같이 세로 벡터로 나타냄.

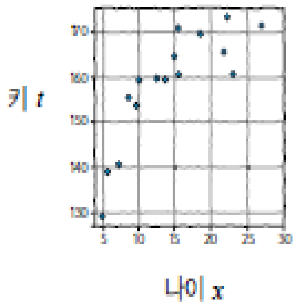
In

```
# 리스트 5-1-(5)
# 데이터 그래프 -----------------------------
plt.figure(figsize=(4, 4))
plt.plot(X, T, marker='o', linestyle='None',
         markeredgecolor='black', color='cornflowerblue')
plt.xlim(X_min, X_max)
plt.grid(True)
plt.show()
```

Out

```
# 실행 결과는 [그림 5-1]을 참조
```

그림 5-1 나이와 키의 인공 데이터(16인분)



나이 x

```
[15.43 23.01  5.00 12.56  8.67  7.31 9.66 13.64
 14.92 18.47 15.48 22.13 10.11 26.95 5.68 21.76]
```

키 t

```
[170.91 160.68 129.00 159.70 155.46 140.56 153.65 159.43
 164.70 169.65 160.71 173.29 159.31 171.52 138.96 165.87]
```

리스트 5-1-(1)

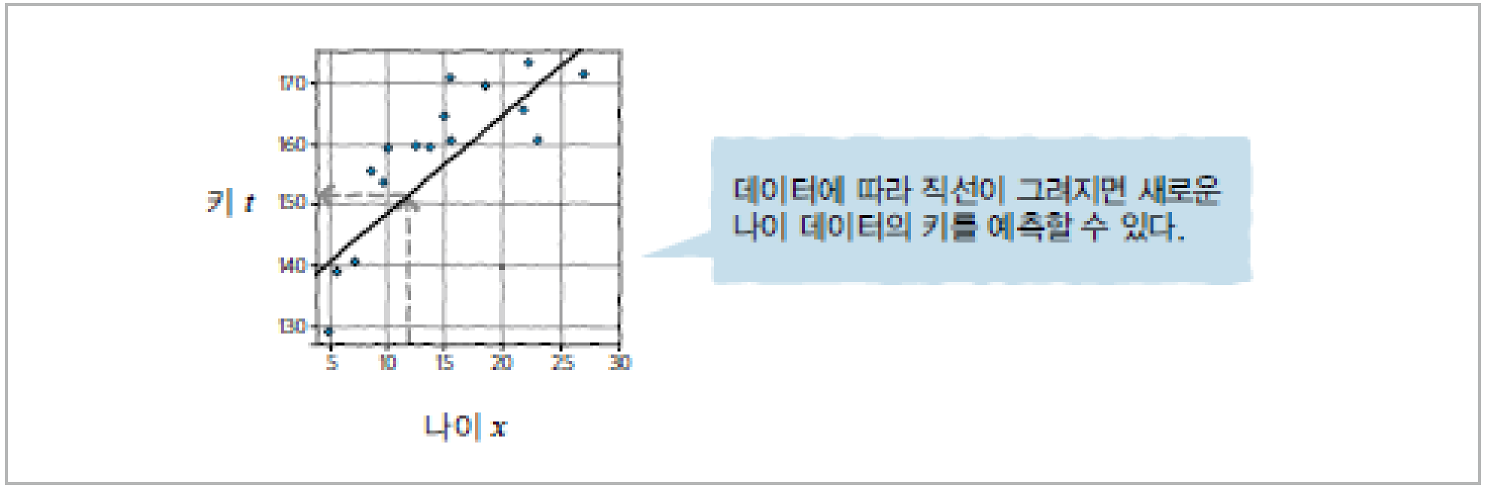
SECTION 01 1차원 입력 직선 모델

1.1 직선 모델

- 직선의 방정식은 다음과 같이 나타낼 수 있습니다.
- 기울기를 나타내는 w0과 절편을 나타내는 w1에 적당한 값을 넣으면, 다양한 위치와 기울기의 직선을 만들 수 있음.
- 이 수식은 입력 x에 y (x)를 출력하는 함수로 볼 수 있으므로, y (x)는 x에 대한 t의 예측치로 간주할 수 있음.

$$y(x) = w_0x + w_1$$ → [직선의 방정식]

그림 5-2 데이터에 따라 직선을 긋다

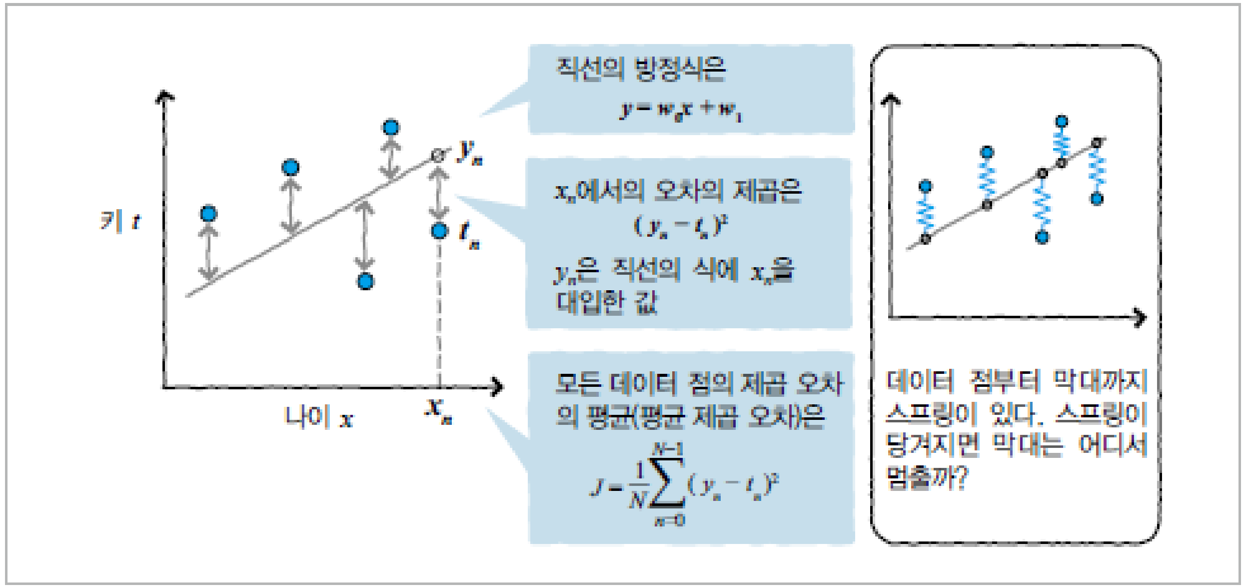


1.2 제곱 오차 함수

- ‘데이터에 부합하도록’ 다음과 같이 오차 J를 정의함.
- J는 평균 제곱 오차(mean square error, MSE)로, 직선과 데이터 점의 차의 제곱의 평균.

$$J = \frac{1}{N}\sum_{n=0}^{N-1}(y_n - t_n)^2$$

그림 5-3 평균 제곱 오차



SECTION 01 1차원 입력 직선 모델

- 평균 제곱오차와 매개 변수의 관계

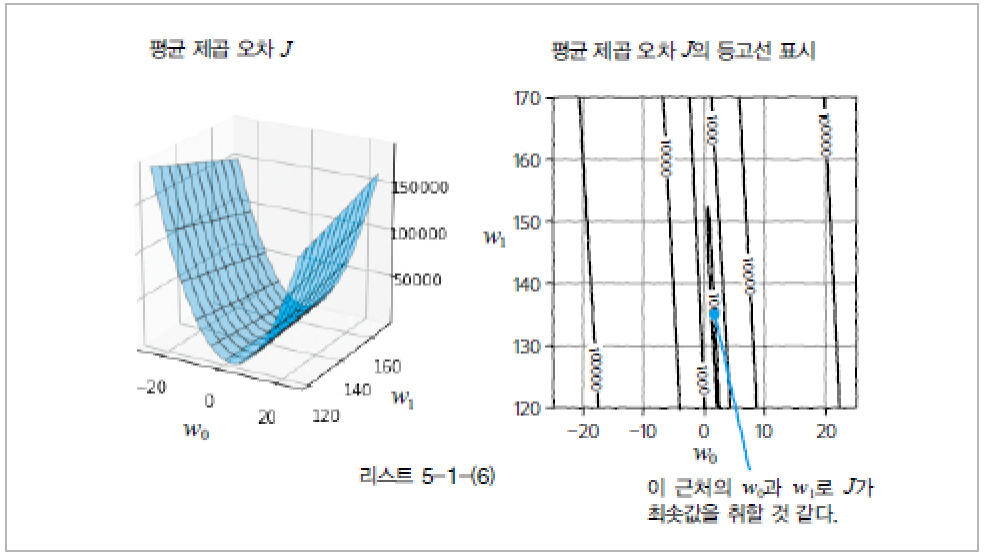
In

```
# 리스트 5-1-(6)
from mpl_toolkits.mplot3d import Axes3D
# 평균 오차 함수 ---------------------------------
def mse_line(x, t, w):
    y = w[0] * x + w[1]
    mse = np.mean((y - t)**2)
    return mse

# 계산 ---------------------------------------
xn = 100 # 등고선 표시 해상도
w0_range = [-25, 25]
w1_range = [120, 170]
x0 = np.linspace(w0_range[0], w0_range[1], xn)
x1 = np.linspace(w1_range[0], w1_range[1], xn)
xx0, xx1 = np.meshgrid(x0, x1)
J = np.zeros((len(x0), len(x1)))
for i0 in range(xn):
    for i1 in range(xn):
        J[i1, i0] = mse_line(X, T, (x0[i0], x1[i1]))

# 표시 ---------------------------------------
plt.figure(figsize=(9.5, 4))
plt.subplots_adjust(wspace=0.5)

ax = plt.subplot(1, 2, 1, projection='3d')
ax.plot_surface(xx0, xx1, J, rstride=10, cstride=10, alpha=0.3,
                color='blue', edgecolor='black')
ax.set_xticks([-20, 0, 20])
ax.set_yticks([120, 140, 160])
ax.view_init(20, -60)

plt.subplot(1, 2, 2)
cont = plt.contour(xx0, xx1, J, 30, colors='black',
                   levels=[100, 1000, 10000, 100000])
cont.clabel(fmt='%1.0f', fontsize=8)
plt.grid(True)
plt.show()
```

Out

```
# 실행 결과는 [그림 5-4]를 참조
```

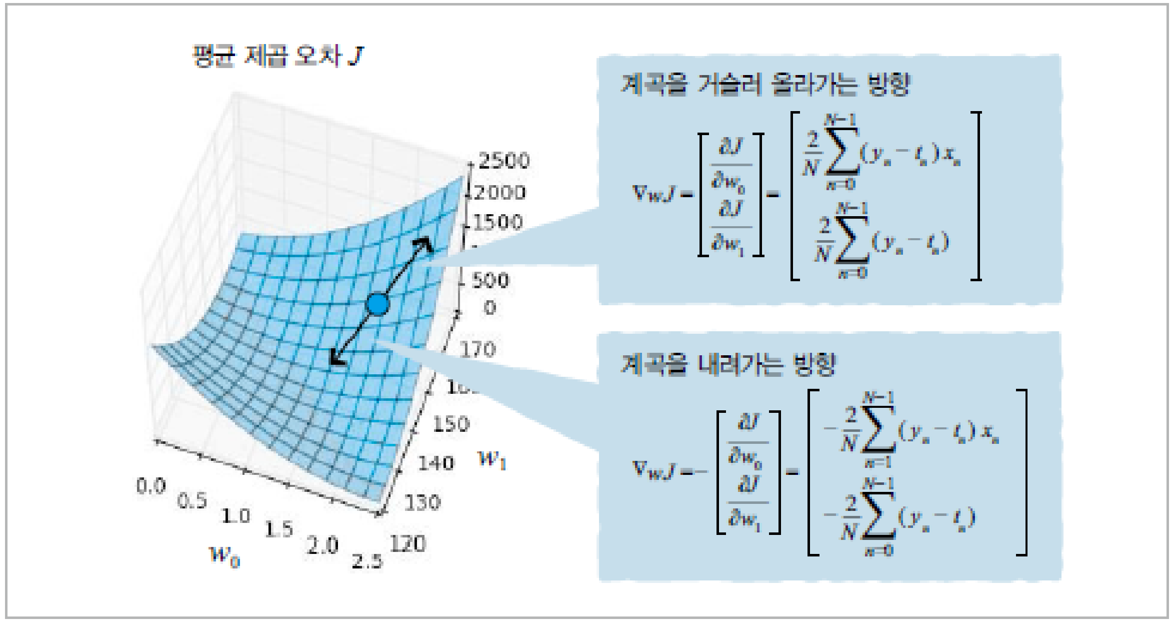
그림 5-4 평균 제곱 오차와 매개 변수의 관계



1.3 매개 변수 구하기(경사 하강법)

- J가 가장 작아지는 w0과 w1은 구하는데 가장 간단하고 기본적인 방법은 경사 하강법임.
- 경사 하강법의 w0과 w1에 대한 J의 지형
- 경사 하강법 또는 구배법(勾配法)은 1차 근삿값 발견용 최적화 알고리즘임.
- 기본 아이디어는 함수의 기울기(경사)를 구하여 기울기가 낮은 쪽으로 계속 이동시켜서 극값에 이를 때까지 반복시키는 것임.

그림 5-5 기울기(gradient)



SECTION 01 1차원 입력 직선 모델

- 경사 하강법은 반복 계산에 의해 근사값을 구하는 수치 계산법임.
- 프로그램을 실행하면 마지막으로 얻어진 w 값 등을 표시하고 w의 갱신 내역을 그래프로 표시함.

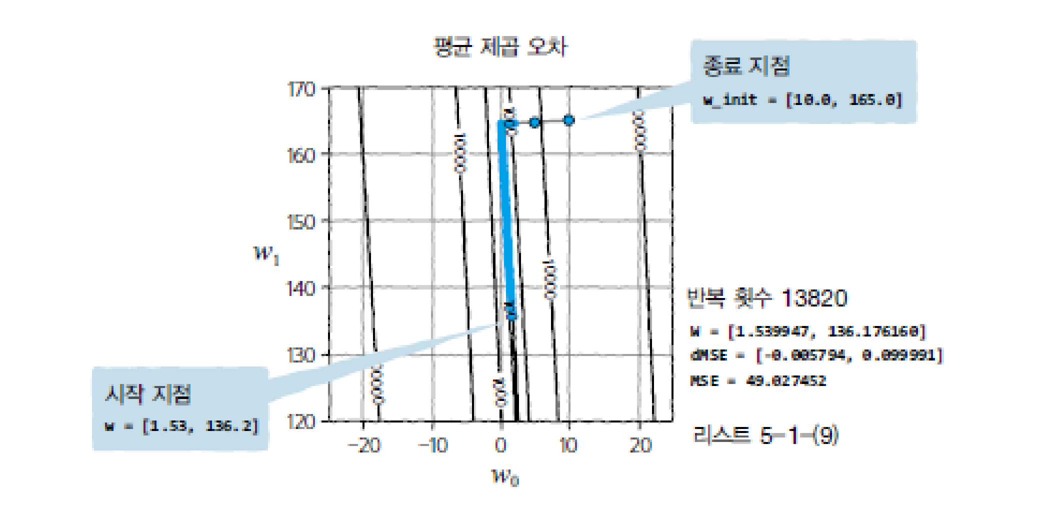
In

```
# 리스트 5-1-(9)
# 경사 하강법 ------------------------------------
def fit_line_num(x, t):
    w_init = [10.0, 165.0] # 초기 매개 변수
    alpha = 0.001 # 학습률
    i_max = 100000 # 반복의 최대 수
    eps = 0.1 # 반복을 종료 기울기의 절대값의 한계
    w_i = np.zeros([i_max, 2])
    w_i[0, :] = w_init
    for i in range(1, i_max):
        dmse = dmse_line(x, t, w_i[i - 1])
        w_i[i, 0] = w_i[i - 1, 0] - alpha * dmse[0]
        w_i[i, 1] = w_i[i - 1, 1] - alpha * dmse[1]
        if max(np.absolute(dmse)) < eps: # 종료판정, np.absolute는 절대값
            break
    w0 = w_i[i, 0]
    w1 = w_i[i, 1]
    w_i = w_i[:i, :]
    return w0, w1, dmse, w_i


# 메인 ------------------------------------
plt.figure(figsize=(4, 4)) # MSE의 등고선 표시
xn = 100 # 등고선 해상도
w0_range = [-25, 25]
w1_range = [120, 170]
x0 = np.linspace(w0_range[0], w0_range[1], xn)
x1 = np.linspace(w1_range[0], w1_range[1], xn)
xx0, xx1 = np.meshgrid(x0, x1)
J = np.zeros((len(x0), len(x1)))
for i0 in range(xn):
    for i1 in range(xn):
        J[i1, i0] = mse_line(X, T, (x0[i0], x1[i1]))
cont = plt.contour(xx0, xx1, J, 30, colors='black',
                   levels=(100, 1000, 10000, 100000))
cont.clabel(fmt='%1.0f', fontsize=8)
plt.grid(True)
# 경사 하강법 호출
W0, W1, dMSE, W_history = fit_line_num(X, T)
# 결과보기
print('반복 횟수 {0}'.format(W_history.shape[0]))
print('W=[{0:.6f}, {1:.6f}]'.format(W0, W1))
print('dMSE=[{0:.6f}, {1:.6f}]'.format(dMSE[0], dMSE[1]))
print('MSE={0:.6f}'.format(mse_line(X, T, [W0, W1])))
plt.plot(W_history[:, 0], W_history[:, 1], '.-',
         color='gray', markersize=10, markeredgecolor='cornflowerblue')
plt.show()
```

Out

```
# 실행 결과는 [그림 5-6]을 참조
```

그림 5-6 경사 하강법



SECTION 01 1차원 입력 직선 모델

- 직선 식에 대입하여 데이터 분포에 겹쳐서 그려봄.
- 언제나 전체의 최소값으로 수렴함.

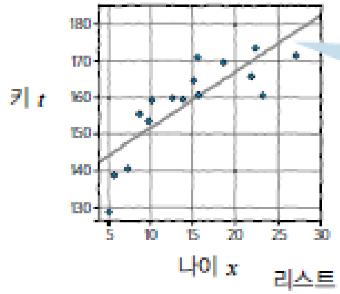
In

```
# 리스트 5-1-(10)
# 선 표시 ----------------------------------
def show_line(w):
    xb = np.linspace(X_min, X_max, 100)
    y = w[0] * xb + w[1]
    plt.plot(xb, y, color=(.5, .5, .5), linewidth=4)


# 메인 ------------------------------------
plt.figure(figsize=(4, 4))
W=np.array([W0, W1])
mse = mse_line(X, T, W)
print("w0={0:.3f}, w1={1:.3f}".format(W0, W1))
print("SD={0:.3f} cm".format(np.sqrt(mse)))
show_line(W)
plt.plot(X, T, marker='o', linestyle='None',
         color='cornflowerblue', markeredgecolor='black')
plt.xlim(X_min, X_max)
plt.grid(True)
plt.show()
```

Out

```
# 실행 결과는 [그림 5-7]을 참조
```

그림 5-7 경사하강법에 의한 직선 모델의 피팅fitting 결과



직선의 식은

$y = w_0 x + w_1$

$w_0 = 1.5, w_1 = 136.1$

이때,

평균 제곱 오차 MSE= 49.03cm^2

표준 편차 SD = 7.00cm

리스트 5-1-(10)

1.4 선형 모델 매개 변수의 해석해

- 직선 모델의 경우에서는 근사적인 해석이 아니라 방정식을 해결하여 정확한 해를 구할 수 있음.
- 해석해(解析解)란 해석적(analytic)으로 풀이가 가능한 해, 다시 말해, 해석적 해를 말함.

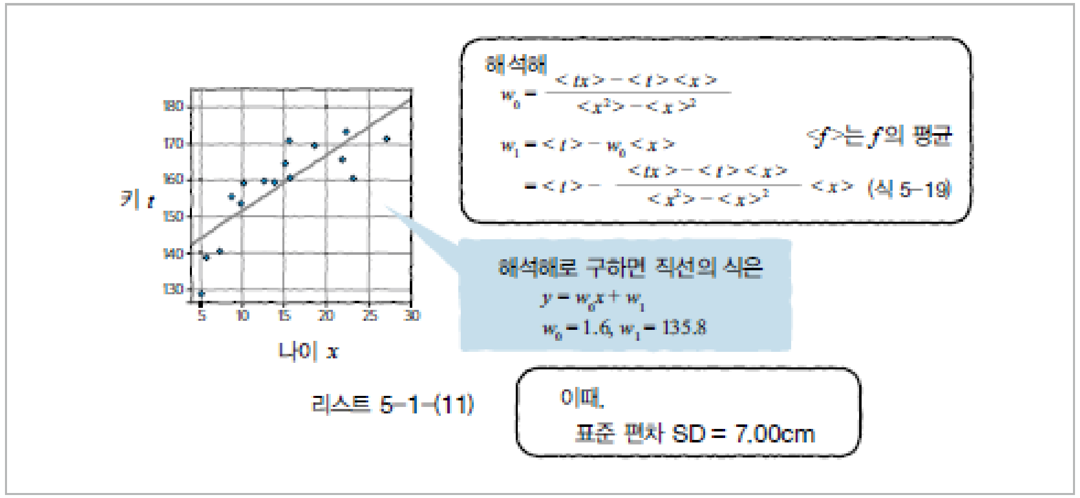
In

```python
# 리스트 5-1-(11)
# 해석해 ------------------------------------
def fit_line(x, t):
    mx = np.mean(x)
    mt = np.mean(t)
    mtx = np.mean(t * x)
    mxx = np.mean(x * x)
    w0 = (mtx - mt * mx) / (mxx - mx**2)
    w1 = mt - w0 * mx
    return np.array([w0, w1])

# 메인 ------------------------------------
W = fit_line(X, T)
print("w0={0:.3f}, w1={1:.3f}".format(W[0], W[1]))
mse = mse_line(X, T, W)
print("SD={0:.3f} cm".format(np.sqrt(mse)))
plt.figure(figsize=(4, 4))
show_line(W)
plt.plot(X, T, marker='o', linestyle='None',
         color='cornflowerblue', markeredgecolor='black')
plt.xlim(X_min, X_max)
plt.grid(True)
plt.show()
```

Out

```python
# 실행 결과는 [그림 5-8]을 참조
```

그림 5-8 해석해 의한 선형 모델의 피팅 결과



SECTION 02 2차원 입력면 모델

- 몸무게 x와 키 t가 세트로 된 데이터를 생각함.
- 이를 묶어 다음과 같이 세로 벡터로 나타냄.

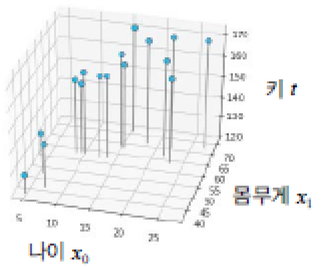
In

```python
# 리스트 5-1-(14)
# 2차원 데이터의 표시 -------------------------
def show_data2(ax, x0, x1, t):
    for i in range(len(x0)):
        ax.plot([x0[i], x0[i]], [x1[i], x1[i]],
                [120, t[i]], color='gray')
        ax.plot(x0, x1, t, 'o',
                color='cornflowerblue', markeredgecolor='black',
                markersize=6, markeredgewidth=0.5)
        ax.view_init(elev=35, azim=-75)

# 메인 ------------------------------------
plt.figure(figsize=(6, 5))
ax = plt.subplot(1,1,1,projection='3d')
show_data2(ax, X0, X1, T)
plt.show()
```

Out

```
# 실행 결과는 [그림 5-9]를 참조
```

그림 5-9 나이와 몸무게와 키의 인공 데이터



리스트 5-1-(13), 5-1-(14)

```
X0=
[15.43 23.01  5.   12.56  8.67  7.31 9.66 13.64
 14.92 18.47 15.48 22.13 10.11 26.95 5.68 21.76]

X1=
[70.43 58.15 37.22 56.51 57.32 40.84 57.79 56.94
 63.03 65.69 62.33 64.95 57.73 66.89 46.68 61.08]

T=
[170.91 160.68 129.    159.7  155.46 140.56 153.65
 159.43 164.7  169.65 160.71 173.29 159.31 171.52
 138.96 165.87]
```

2.1 데이터의 표시 방법

- 수식을 작성할 때의 데이터 표시법을 정리.
- 데이터의 번호는 n으로, 벡터의 요소(0=나이, 1=몸무게 등) 번호는 m으로 나타내도록 함.

$$\mathbf{x}_n = [x_{n,0}, x_{n,1}]$$

- 데이터 번호 n의 모든 x의 요소를 쓸 때는 볼드체

$$\mathbf{x}_n = [x_{n,0}, x_{n,1}, \cdots, x_{n,M-1}]$$

- xn이 2차원이 아니라 M차원일 경우

$$\mathbf{X} = \begin{bmatrix} x_{0,0} & x_{0,1} & \cdots & x_{0,M-1} \\ x_{1,0} & x_{1,1} & \cdots & x_{1,M-1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N-1,0} & x_{N-1,1} & \cdots & x_{N-1,M-1} \end{bmatrix}$$

- 모든 데이터 n을 보여주는 경우

$$\mathbf{x}_m = \begin{bmatrix} x_{0,m} \\ x_{1,m} \\ \vdots \\ x_{N-1,m} \end{bmatrix}$$

- 차원 m으로 정리하고 싶을 경우

$$\mathbf{t} = \begin{bmatrix} t_0 \\ t_1 \\ \vdots \\ t_{N-1} \end{bmatrix}$$

- t에 대해서도 모든 N으로 정리할 경우

SECTION 02 2차원 입력면 모델

2.2 면 모델

- 공간에 많은 점을 찍을 수 있으며, 이 점의 집합이 '평평한 표면을 형성하는' 것임.

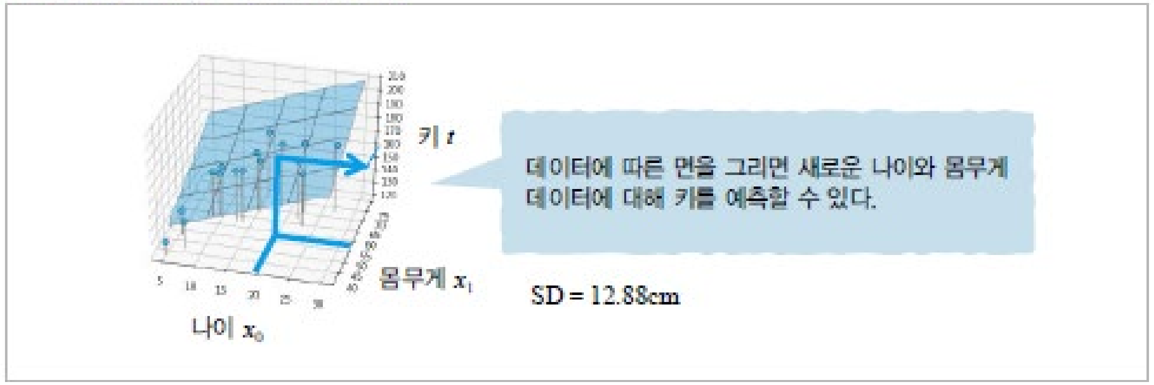
In

```
# 리스트 5-1-(15)
#면의 표시 ------------------------------------
def show_plane(ax, w):
    px0 = np.linspace(X0_min, X0_max, 5)
    px1 = np.linspace(X1_min, X1_max, 5)
    px0, px1 = np.meshgrid(px0, px1)
    y = w[0]*px0 + w[1] * px1 + w[2]
    ax.plot_surface(px0, px1, y, rstride=1, cstride=1, alpha=0.3,
                    color='blue', edgecolor='black')

#면의 MSE ------------------------------------
def mse_plane(x0, x1, t, w):
    y = w[0] * x0 + w[1] * x1 + w[2] # (A)
    mse = np.mean((y - t)**2)
    return mse

# 메인 ------------------------------------
plt.figure(figsize=(6, 5))
ax = plt.subplot(1, 1, 1, projection='3d')
W = [1.5, 1, 90]
show_plane(ax, W)
show_data2(ax, X0, X1, T)
mse = mse_plane(X0, X1, T, W)
print("SD={0:.3f} cm".format(np.sqrt(mse)))
plt.show()
```

Out

```
# 실행 결과는 [그림 5-10]을 참조
```

그림 5-10 데이터에 따라 면을 대응시키기



2.3 매개 변수의 해석해

2차원 면 모델의 경우에도 1차원의 선 모델과 마찬가지로 평균 제곱 오차를 정의할 수 있음.

In

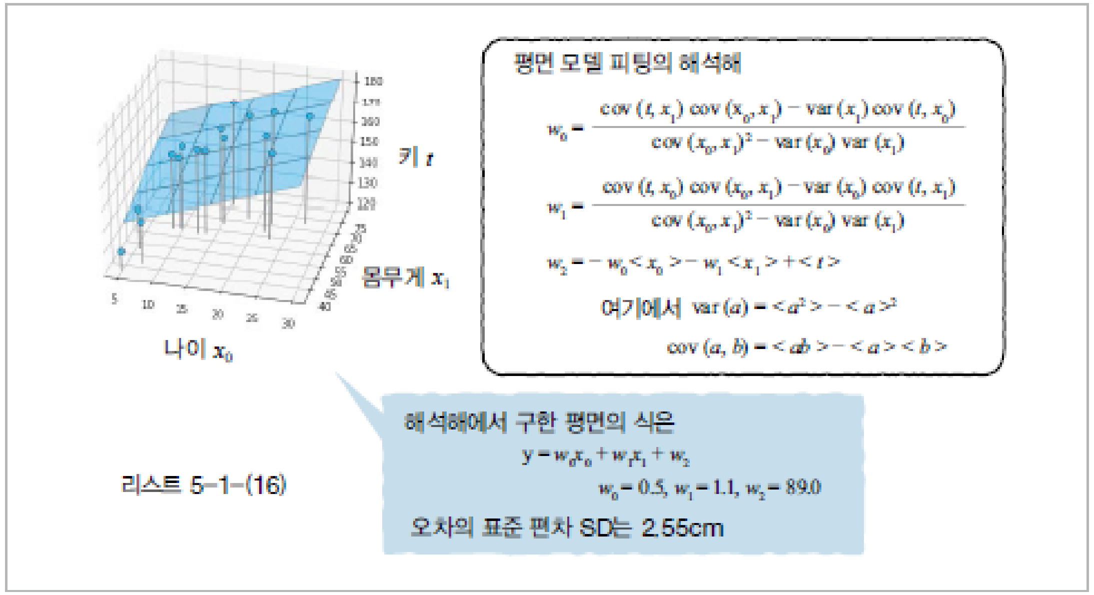
```
# 리스트 5-1-(16)
# 해석해 ------------------------------------
def fit_plane(x0, x1, t):
    c_tx0 = np.mean(t * x0) - np.mean(t) * np.mean(x0)
    c_tx1 = np.mean(t * x1) - np.mean(t) * np.mean(x1)
    c_x0x1 = np.mean(x0 * x1) - np.mean(x0) * np.mean(x1)
    v_x0 = np.var(x0)
    v_x1 = np.var(x1)
    w0 = (c_tx1 * c_x0x1 - v_x1 * c_tx0) / (c_x0x1**2 - v_x0 * v_x1)
    w1 = (c_tx0 * c_x0x1 - v_x0 * c_tx1) / (c_x0x1**2 - v_x0 * v_x1)
    w2 = -w0 * np.mean(x0) - w1 * np.mean(x1) + np.mean(t)
    return np.array([w0, w1, w2])


# 메인 ------------------------------------
plt.figure(figsize=(6, 5))
ax = plt.subplot(1, 1, 1, projection='3d')
W = fit_plane(X0, X1, T)
print("w0={0:.1f}, w1={1:.1f}, w2={2:.1f}".format(W[0], W[1], W[2]))
show_plane(ax, W)
show_data2(ax, X0, X1, T)
mse = mse_plane(X0, X1, T, W)
print("SD={0:.3f} cm".format(np.sqrt(mse)))
plt.show()
```

Out

```
# 실행 결과는 [그림 5-11]을 참조
```

$$J = \frac{1}{N}\sum_{n=0}^{N-1}(y(x_n) - t_n)^2 = \frac{1}{N}\sum_{n=0}^{N-1}(w_0 x_{n,0} + w_1 x_{n,1} + w_2 - t_n)^2$$

그림 5-11 해석해에 의한 평면 모델의 피팅 결과



3.1 D차원 선형 회귀 모델

공식	설명
$y(x) = w_0x_0 + w_1x_1 + \cdots + w_{D-1}x_{D-1} + w_D$ (식 5-37)	• 직선 모델, 면 모델은 모두 선형 회귀 모델이라는 같은 종류의 모델임. • 일반적으로 [식 5-37]과 같이 나타냄.
$y(x) = w_0x_0 + w_1x_1 + \cdots + w_{D-1}x_{D-1}$ (식 5-38)	• 마지막 wD는 절편을 나타내고, x가 곱해지지 않은 점에 주의. • 절편의 항을 포함하지 않는 모델로 생각함(식 5-38).
$y(x) = w_0x_0 + w_1x_1 + \cdots + w_{D-1}x_{D-1} = [w_0 \cdots w_{D-1}]\begin{bmatrix} x_0 \\ \vdots \\ x_{D-1} \end{bmatrix} = \mathbf{w}^{\mathrm{T}}\mathbf{x}$ (식 5-39)	• 절편 wD가 모델에 포함되지 않으면 어떤 w에서도 원점 x =[0, 0, ..., 0]을 대입하면 y가 0이 됨. • 즉, 이 모델은 어떤 w더라도 원점을 지나는 평면(고차원 공간의 면과 같은 것)임. • 이 모델을 행렬 표기법을 사용하여 짧게 정리하면, [식 5-39]의 오른쪽처럼 wTx로 나타낼 수 있음.
$\mathbf{w} = \begin{bmatrix} x_0 \\ \vdots \\ x_{D-1} \end{bmatrix}$	• 즉 w는 다음과 같음.

3.2 매개 변수의 해석해

공식	설명
$$J(\mathbf{w})=\frac{1}{N}\sum_{n=0}^{N-1}(y(x_n)-t_n)^2=\frac{1}{N}\sum_{n=0}^{N-1}(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n-t_n)^2$$ (식 5-40)	• 평균 제곱 오차 J를 [식 5-40]처럼 나타냄
$$\frac{\partial J}{\partial w_i}=\frac{1}{N}\sum_{n=0}^{N-1}\frac{\partial}{\partial w_i}(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n-t_n)^2=\frac{2}{N}\sum_{n=0}^{N-1}(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n-t_n)\,x_{n,i}$$ (식 5-41)	• 연쇄 법칙을 사용하여 wi로 미분하면 [식 5-41]과 같음.
$$\frac{2}{N}\sum_{n=0}^{N-1}(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n-t_n)\,x_{n,i}=0$$ (식 5-42)	• wTxn= w0 xn,0 +…+ wn,D−1 xn,D−1을 wi로 미분하면 xn,i만 남게 되므로 주의. • J를 최소로 만드는 w는 모든 wi 방향에 대한 기울기가 0인, 즉 편미분(식 5-41)이 0이 되므로, [식 5-42]는 i = 0~D – 1에서 성립됨.
$$\sum_{n=0}^{N-1}(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n-t_n)\,x_{n,i}=0$$ (식 5-43)	• 즉, 이 D개의 연립 방정식을 각 wi에 대해 풀면 해답을 얻을 수 있음. • 먼저 양변을 N/2배 하여 약간 간단하게 만든 [식 5-43]을 생각해 봄.
$$\sum_{n=0}^{N-1}(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n-t_n)\,x_{n,0}=0$$ $$\sum_{n=0}^{N-1}(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n-t_n)\,x_{n,1}=0$$ $$\vdots$$ $$\sum_{n=0}^{N-1}(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n-t_n)\,x_{n,D-1}=0$$ (식 5-44)	• 행렬을 사용하면 D는 D인 채로 답을 낼 수 있음. • 우선 [식5-43] 전체를 벡터 형식으로 정리함. • [식 5-43]은 모든 i에서 성립되므로 각각을 정성스럽게 써 나가면 [식 5-44]와 같음.

3.2 매개 변수의 해석해

공식	설명
$\sum_{n=0}^{N-1}(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n - t_n)[x_{n,0}, x_{n,1}, \cdots, x_{n,D-1}] = [0 \quad 0 \quad \cdots \quad 0]$ (식 5–45)	• 마지막 x의 첨자만 0에서 D-1까지 변하고 있음. • 이러한 식을 벡터 하나로 묶어서 [식5-45]와 같이 나타낼 수 있음.
$\sum_{n=0}^{N-1}(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n - t_n)\mathbf{x}_n^{\mathrm{T}} = [0 \quad 0 \quad \cdots \quad 0]$ (식 5–46)	• 그리고 $[x_{n,0}, x_{n,1}, \cdots, x_{n,D-1}]$은 $\mathbf{x}_n^{\mathrm{T}}$ [식 5-46]과 같음.
$\sum_{n=0}^{N-1}(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n - t_n\mathbf{x}_n^{\mathrm{T}}) = [0 \quad 0 \quad \cdots \quad 0]$ (식 5–47)	• 이제 [식 5-43]을 벡터 형식으로 변환. • 행렬도 (a + b)c = ac + bc의 분배 법칙이 성립되므로 [식 5-47]과 같이 확장할 수 있음.
$\mathbf{w}^{\mathrm{T}}\sum_{n=0}^{N-1}\mathbf{x}_n\mathbf{x}_n^{\mathrm{T}} - \sum_{n=0}^{N-1}t_n\mathbf{x}_n^{\mathrm{T}} = [0 \quad 0 \quad \cdots \quad 0]$ (식 5–48)	• 합을 분해하면 [식 5-48]과 같음.
$\mathbf{w}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{X} - \mathbf{t}^{\mathrm{T}}\mathbf{X} = [0 \quad 0 \quad \cdots \quad 0]$ (식 5–49)	• 이 좌변은 [식 5-49]와 같이 행렬의 식으로 나타낼 수 있음.

3.2 매개 변수의 해석해

공식	설명
$\mathbf{X} = \begin{bmatrix} x_{0,0} & x_{0,1} & \cdots & x_{0,D-1} \\ x_{1,0} & x_{1,1} & \cdots & x_{0,D-1} \\ \vdots & \vdots & \vdots & \vdots \\ x_{N-1,0} & x_{N-1,1} & \cdots & x_{N-1,D-1} \end{bmatrix}$	• X는 모든 데이터를 하나의 행렬로 나타낸 행렬인 [식 5-50]임.
$\sum_{n=0}^{N-1} \mathbf{x}_n \mathbf{x}_n^{\mathrm{T}} = \mathbf{X}^{\mathrm{T}}\mathbf{X}$ (식 5-51) $\sum_{n=0}^{N-1} t_n \mathbf{x}_n^{\mathrm{T}} = \mathbf{t}^{\mathrm{T}}\mathbf{X}$ (식 5-52)	• [식 5-48]에서 [식 5-49]로 변환하려면 [식 5-51], [식 5-52]를 사용함.
$\begin{bmatrix} \sum_{n=0}^{N-1} x_{n,0}^2 & \sum_{n=0}^{N-1} x_{n,0}x_{n,1} & \cdots & \sum_{n=0}^{N-1} x_{n,0}x_{n,D-1} \\ \sum_{n=0}^{N-1} x_{n,1}x_{n,0} & \sum_{n=0}^{N-1} x_{n,1}^2 & \cdots & \sum_{n=0}^{N-1} x_{n,1}x_{n,D-1} \\ \vdots & \vdots & \vdots & \vdots \\ \sum_{n=0}^{N-1} x_{n,D-1}x_{n,0} & \sum_{n=0}^{N-1} x_{n,D-1}x_{n,1} & \cdots & \sum_{n=0}^{N-1} x_{n,D-1}^2 \end{bmatrix}$ (식 5-53)	식 5-51]은 좌변과 우변을 성분 표기의 행렬로 하면 모두 [식 5-53]처럼 되기 때문에 등호가 성립된다고 볼 수 있음. N = 2, D = 2를 상정하면 확인하기도 편리함.
$\begin{bmatrix} \sum_{n=0}^{N-1} t_n x_{n,0} & \sum_{n=0}^{N-1} t_n x_{n,1} & \cdots & \sum_{n=0}^{N-1} t_n x_{n,D-1} \end{bmatrix}$ (식 5-54)	[식 5-52]도 왼쪽과 오른쪽을 성분 표기하면 둘 다 [식 5-54]와 같이 되기 때문에 등호가 성립 된다고 볼 수 있음
$(\mathbf{w}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{X} - \mathbf{t}^{\mathrm{T}}\mathbf{X})^{\mathrm{T}} = [0 \quad 0 \quad \cdots \quad 0]^{\mathrm{T}}$ (식 5-55)	그런데 여기에서는 [식 5-49]를 변형하여 w =의 형태로 가져가는 것을 생각. 먼저 양변을 전치하면 [식 5-55]와 같음.

SECTION.03 D차원 선형 회귀 모델

3.2 매개 변수의 해석해

공식	설명
$(\mathbf{w}^T\mathbf{X}^T\mathbf{X})^T - (\mathbf{t}^T\mathbf{X})^T = [0 \quad 0 \quad \cdots \quad 0]^T$ (식 5-56)	• 위의 좌변 두 항에 외부의 T를 작용시키면 [식 5-56]과 같음. • $(\mathbf{A}+\mathbf{B})^T = \mathbf{A}^T + \mathbf{B}^T$ 관계식을 사용했음.
$(\mathbf{X}^T\mathbf{X})^T(\mathbf{w}^T)^T - \mathbf{X}^T\mathbf{t} = [0 \quad 0 \quad \cdots \quad 0]^T$ (식 5-57)	• 또한 $(\mathbf{A}^T)^T = \mathbf{A}$라는 관계식과 $(\mathbf{AB})^T = \mathbf{B}^T\mathbf{A}^T$(4.6.7절)을 사용하여 [식 5-57]을 얻음.
$(\mathbf{X}^T\mathbf{X})\mathbf{w} - \mathbf{X}^T\mathbf{t} = [0 \quad 0 \quad \cdots \quad 0]^T$ (식 5-58)	• 좌변의 1항에서는, $\mathbf{w}^T = \mathbf{A}$, $\mathbf{X}^T\mathbf{X} = \mathbf{B}$ 생각함. • 또한 좌변의 1항은 [식 5-58]과 같이 정리할 수 있음.
$(\mathbf{X}^T\mathbf{X})\mathbf{w} = \mathbf{X}^T\mathbf{t}$ (식 5-59)	• 위 $\mathbf{X}^T\mathbf{t}$를 우변으로 이동하면 [식 5-59]와 같음.
$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{t}$ (식 5-60)	• 마지막으로 왼쪽의 $(\mathbf{X}^T\mathbf{X})$를 지우기 위해서 $(\mathbf{X}^T\mathbf{X})^{-1}$을 양변에 왼쪽에서 곱하면 [식 5-60]과 같이 해석 해를 얻게 됨.

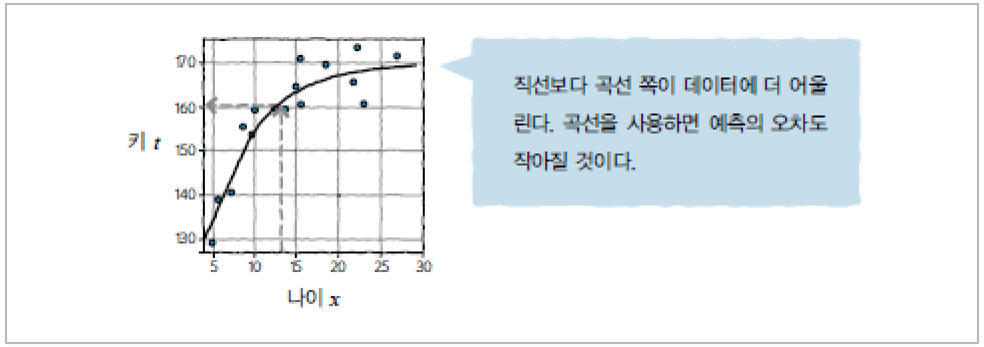
3.3 원점을 지나지 않는 면에 대한 확장

공식	설명
$y(\mathbf{x}) = w_0x_0 + w_1x_1$ (식 5-61)	• 원점에 고정된 면의 방정식은 입력 데이터가 2차원의 경우, [식 5-61]과 같음.
$y(\mathbf{x}) = w_0x_0 + w_1x_1 + w_2$ (식 5-62)	• 여기에 세 번째 매개 변수 w2를 더하면 면을 위아래로 이동할 수 있기 때문에, 원점을 지나지 않는 면을 표현할 수 있음(식 5-62).
$y(\mathbf{x}) = w_0x_0 + w_1x_1 + w_2x_2 = w_0x_0 + w_1x_1 + w_2$ (식 5-63)	• x는 2차원 벡터였지만, 항상 1을 얻는 3차원의 요소 x2 = 1을 추가하여 x를 3차원 벡터라고 생각함. • 그러면 [식 5-63]과 같이 원점에 얽매이지 않는 면을 표현할 수 있게 됨.

SECTION.04 선형 기저 함수 모델

- 기저 함수(basis function) 또는 바탕 함수란 함수 공간의 기저인 함수를 말함.
- 모든 연속함수들은 기저 함수들의 선형결합으로 표시할 수 있음.

그림 5-12 곡선에 의한 피팅



- 기저 함수는 $\phi_j(x)$로 나타냄. 'ϕ는 [illegible]'이라는 그리스 문자.
- 기저 함수는 여러 세트에서 사용되기 때문에 그 번호를 나타내는 J에는 인덱스가 붙어 있음.

$$\phi_j(x) = \exp\left\{-\frac{(x-\mu_j)^2}{2s^2}\right\}$$

SECTION.04 선형 기저 함수 모델

- 기저 함수(basis function) 또는 바탕 함수란 함수 공간의 기저인 함수를 말함.
- 모든 연속함수들은 기저 함수들의 선형결합으로 표시할 수 있음.
- 곡선에 의한 피팅

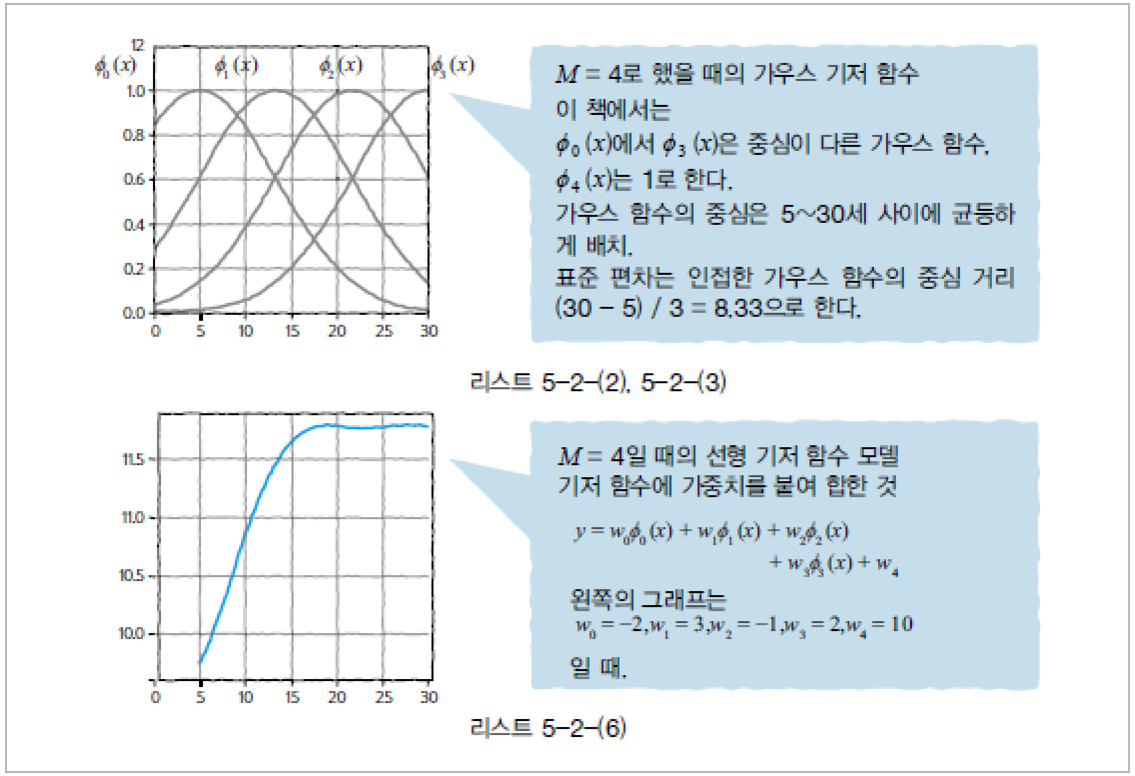
In

```
# 리스트 5-2-(3)
# 메인 ------------------------------------
M = 4
plt.figure(figsize=(4, 4))
mu = np.linspace(5, 30, M)
s = mu[1] - mu[0] # (A)
xb = np.linspace(X_min, X_max, 100)
for j in range(M):
    y = gauss(xb, mu[j], s)
    plt.plot(xb, y, color='gray', linewidth=3)
plt.grid(True)
plt.xlim(X_min, X_max)
plt.ylim(0, 1.2)
plt.show()
```

Out

```
# 실행 결과는 [그림 5-13]을 참조
```

그림 5-13 곡선에 의한 피팅



SECTION.04 선형 기저 함수 모델

- 기저 함수 모델의 피팅 결과

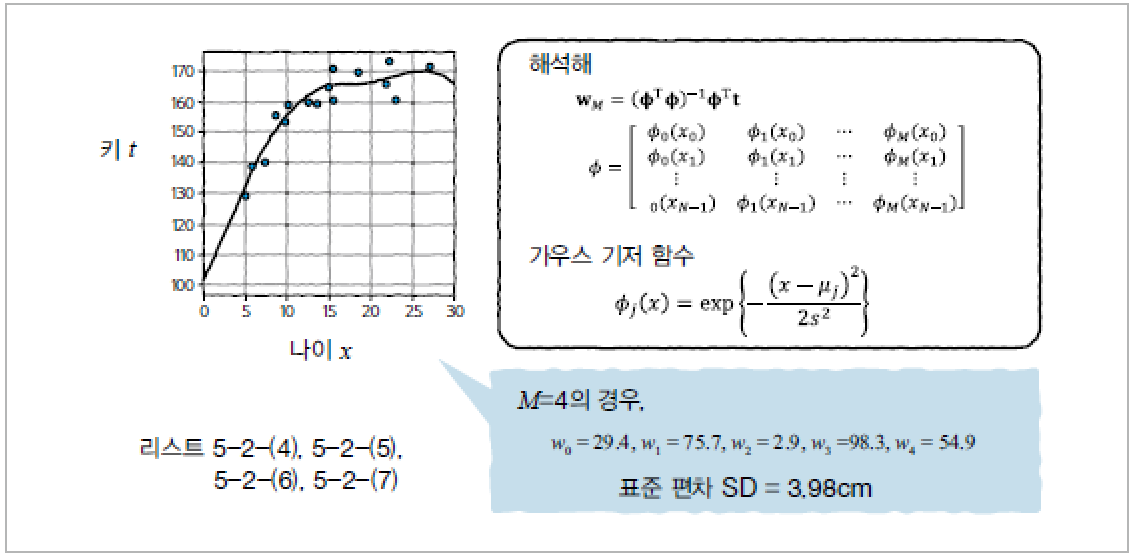
In

```
# 리스트 5-2-(7)
# 가우스 기저 함수 표시 -----------------------
def show_gauss_func(w):
    xb = np.linspace(X_min, X_max, 100)
    y = gauss_func(w, xb)
    plt.plot(xb, y, c=[.5, .5, .5], lw=4)


# 메인 ----------------------------------
plt.figure(figsize=(4, 4))
M = 4
W = fit_gauss_func(X, T, M)
show_gauss_func(W)
plt.plot(X, T, marker='o', linestyle='None',
         color='cornflowerblue', markeredgecolor='black')
plt.xlim(X_min, X_max)
plt.grid(True)
mse = mse_gauss_func(X, T, W)
print('W='+ str(np.round(W,1)))
print("SD={0:.2f} cm".format(np.sqrt(mse)))
plt.show()
```

Out

```
# 실행 결과는 [그림 5-14]를 참조
```

그림 5-14 선형 기저 함수 모델의 피팅 결과



SECTION.05 오버피팅의 문제

- 선형 기저 함수 모델을 사용하여 피팅을 시도함.

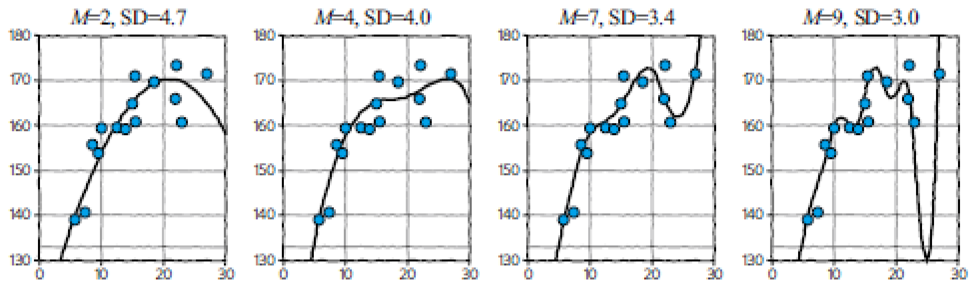
In

```
# 리스트 5-2-(8)
plt.figure(figsize=(10, 2.5))
plt.subplots_adjust(wspace=0.3)
M = [2, 4, 7, 9]
for i in range(len(M)):
    plt.subplot(1, len(M), i + 1)
    W = fit_gauss_func(X, T, M[i])
    show_gauss_func(W)
    plt.plot(X, T, marker='o', linestyle='None',
             color='cornflowerblue', markeredgecolor='black')
    plt.xlim(X_min, X_max)
    plt.grid(True)
    plt.ylim(130, 180)
    mse = mse_gauss_func(X, T, W)


    plt.title("M={0:d}, SD={1:.1f}".format(M[i], np.sqrt(mse)))
plt.show()
```

Out

```
# 실행 결과는 [그림 5-15]를 참조
```

그림 5-15 M=2, 4, 7, 9의 경우 선형 기저 함수 모델에 의한 피팅



M이 늘어나면 오차의 표준 편차 SD는 점점 줄어들지만,
가우스 기저 함수는 흐물흐물하게 변한다!?
이렇게 해서는 새로운 입력에 대한 예측이 잘 되지 않는다.
이 현상을 과적합이라고 한다.

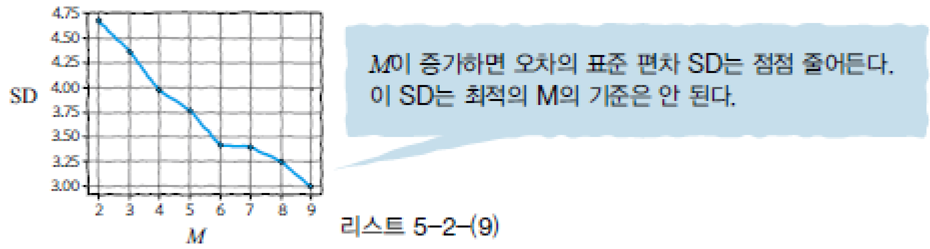
리스트 5-2-(8)

SECTION.05 오버피팅의 문제

- 좀 더 정량적으로 봄.
- M이 증가할수록 선형 기저 함수 모델은 작은 곡선도 표현할 수 있게 되므로 곡선은 데이터 점에 근접하게 되고, 오차(SD)는 점점 감소함.

```
# 리스트 5-2-(9)
plt.figure(figsize=(5, 4))
M = range(2, 10)
mse2 = np.zeros(len(M))
for i in range(len(M)):

    W = fit_gauss_func(X, T, M[i])
    mse2[i] = np.sqrt(mse_gauss_func(X, T, W))
plt.plot(M, mse2, marker='o',
         color='cornflowerblue', markeredgecolor='black')
plt.grid(True)
plt.show()
```

그림 5-16 선형 기저 함수 모델 M과 SD의 관계



리스트 5-2-(9)

SECTION.05 오버피팅의 문제

- 선형 기저 함수 모델 M과 SD의 관계

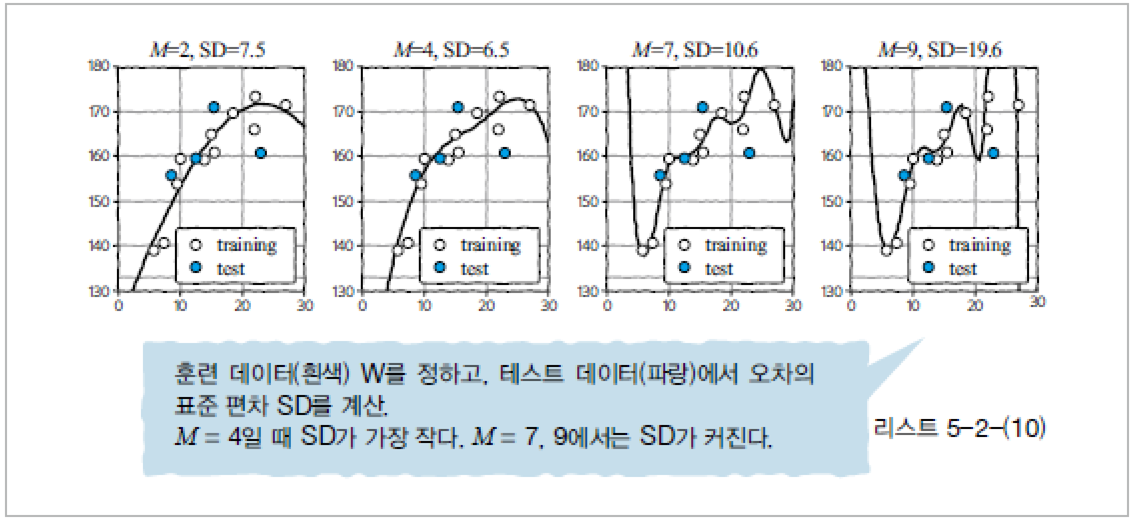
In

```
# 리스트 5-2-(10)
# 훈련 데이터와 테스트 데이터 ------------------
X_test = X[:int(X_n / 4 + 1)]
T_test = T[:int(X_n / 4 + 1)]
X_train = X[int(X_n / 4 + 1):]
T_train = T[int(X_n / 4 + 1):]
# 메인 ----------------------------------
plt.figure(figsize=(10, 2.5))


plt.subplots_adjust(wspace=0.3)
M = [2, 4, 7, 9]
for i in range(len(M)):
    plt.subplot(1, len(M), i + 1)
    W = fit_gauss_func(X_train, T_train, M[i])
    show_gauss_func(W)
    plt.plot(X_train, T_train, marker='o',
             linestyle='None', color='white',
             markeredgecolor='black', label='training')
    plt.plot(X_test, T_test, marker='o', linestyle='None',
             color='cornflowerblue',
             markeredgecolor='black', label='test')
    plt.legend(loc='lower right', fontsize=10, numpoints=1)
    plt.xlim(X_min, X_max)
    plt.ylim(130, 180)
    plt.grid(True)
    mse = mse_gauss_func(X_test, T_test, W)
    plt.title("M={0:d}, SD={1:.1f}".format(M[i], np.sqrt(mse)))
plt.show()
```

Out

```
# 실행 결과는 [그림 5-17]을 참조
```

그림 5-17 선형 기저 함수 모델 M과 SD의 관계



SECTION.05 오버피팅의 문제

- 홀드 아웃에 의한 선형 기저 함수 모델의 훈련 데이터, 테스트 데이터의 SD

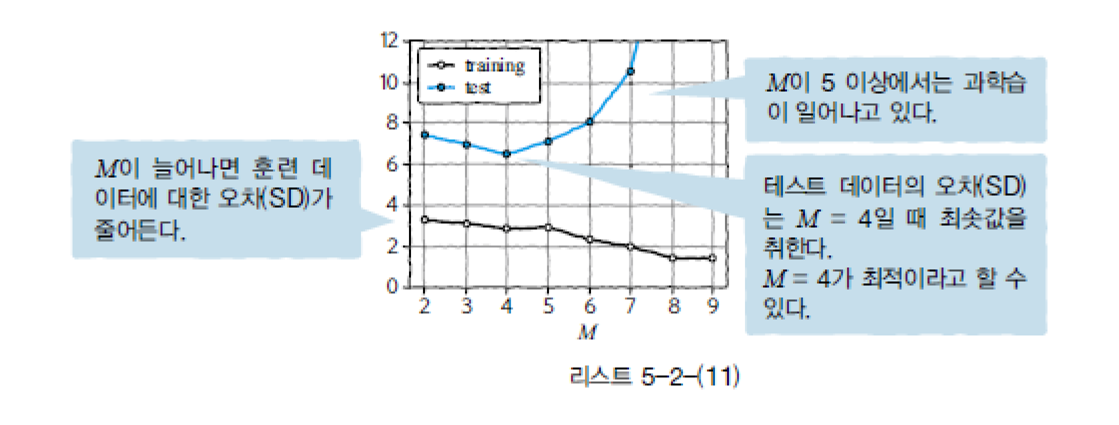
In

```
# 리스트 5-2-(11)
plt.figure(figsize=(5, 4))
M = range(2, 10)
mse_train = np.zeros(len(M))
mse_test = np.zeros(len(M))
for i in range(len(M)):
    W = fit_gauss_func(X_train, T_train, M[i])
    mse_train[i] = np.sqrt(mse_gauss_func(X_train, T_train, W))
    mse_test[i] = np.sqrt(mse_gauss_func(X_test, T_test, W))
plt.plot(M, mse_train, marker='o', linestyle='-',
         markerfacecolor='white', markeredgecolor='black',
         color='black', label='training')
plt.plot(M, mse_test, marker='o', linestyle='-',
         color='cornflowerblue', markeredgecolor='black',
         label='test')
plt.legend(loc='upper left', fontsize=10)
plt.ylim(0, 12)
plt.grid(True)
plt.show()
```

Out

```
# 실행 결과는 [그림 5-18]을 참조
```

그림 5-18 홀드 아웃에 의한 선형 기저 함수 모델의 훈련 데이터, 테스트 데이터의 SD'



SECTION.05 오버피팅의 문제

- 데이터의 분류법에 따라 홀드 아웃의 결과가 달라짐
- 데이터를 분할하는 종류의 개수로 K겹교차 검증(K-hold cross-validation)으로 부르기도 함.

그림 5-19 데이터의 분류법에 따라 홀드 아웃의 결과가 달라짐

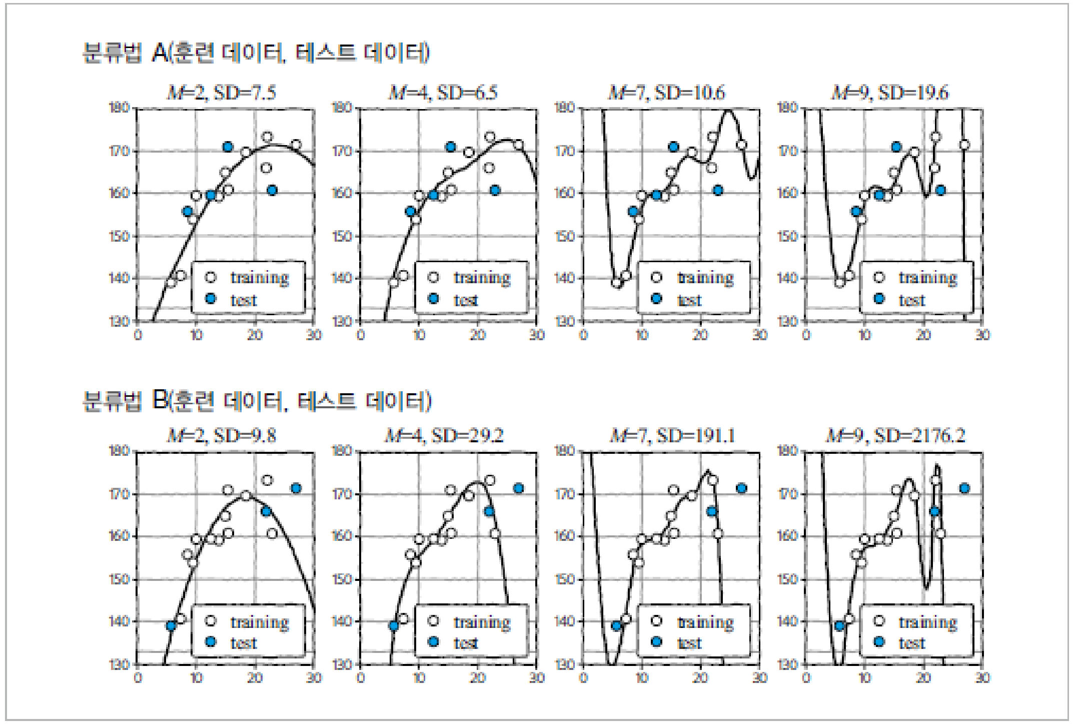


그림 5-20 K겹 교차 검증(K-fold cross validation) 방법

데이터 X와 t를 K개로 분할한다. 이 중 하나를 테스트 데이터로 삼아 나머지를 훈련 데이터로 한다. 훈련 데이터를 사용해 모델 매개 변수를 최적화하고, 테스트 데이터의 평균 제곱 오차를 계산한다. 테스트 데이터를 바꾸어 이 절차를 K회 반복한다. 마지막으로 K개의 평균 제곱 오차의 평균을 취하여 모델의 평가치로 한다.

SECTION.05 오버피팅의 문제

- 선형 기저 함수 모델의 LOOCV

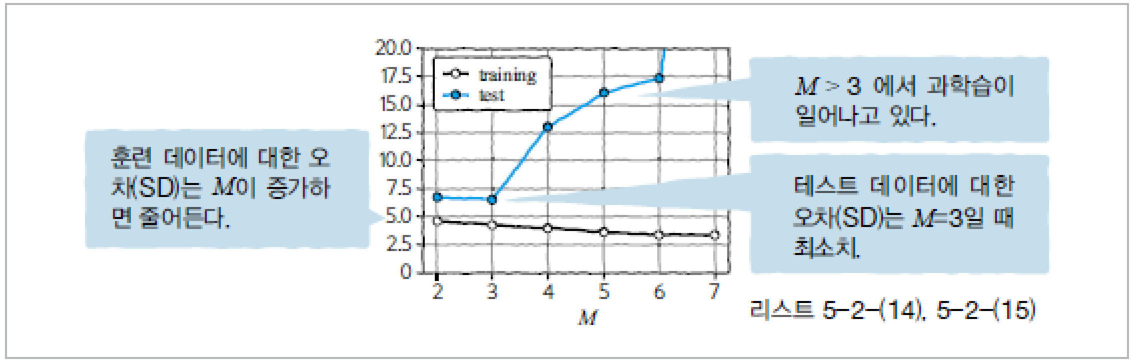
In

```
# 리스트 5-2-(15)
M = range(2, 8)
K = 16
Cv_Gauss_train = np.zeros((K, len(M)))
Cv_Gauss_test = np.zeros((K, len(M)))
for i in range(0, len(M)):
    Cv_Gauss_train[:, i], Cv_Gauss_test[:, i] =\
                kfold_gauss_func(X, T, M[i], K)
mean_Gauss_train = np.sqrt(np.mean(Cv_Gauss_train, axis=0))
mean_Gauss_test = np.sqrt(np.mean(Cv_Gauss_test, axis=0))


plt.figure(figsize=(4, 3))
plt.plot(M, mean_Gauss_train, marker='o', linestyle='-',
       color='k', markerfacecolor='w', label='training')
plt.plot(M, mean_Gauss_test, marker='o', linestyle='-',
       color='cornflowerblue', markeredgecolor='black', label='test')
plt.legend(loc='upper left', fontsize=10)
plt.ylim(0, 20)
plt.grid(True)
plt.show()
```

Out

```
# 실행 결과는 [그림 5-21]을 참조
```

그림 5-21 선형 기저 함수 모델의 LOOCV



SECTION.05 오버피팅의 문제

- LOOCV에서 얻은 M=3의 선형 기저 함수 모델의 피팅

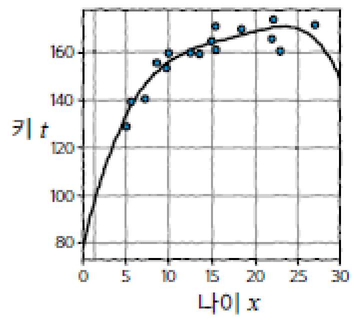
In

```
# 리스트 5-2-(16)
M = 3
plt.figure(figsize=(4, 4))
W = fit_gauss_func(X, T, M)
show_gauss_func(W)
plt.plot(X, T, marker='o', linestyle='None',
         color='cornflowerblue', markeredgecolor='black')
plt.xlim([X_min, X_max])
plt.grid(True)
mse = mse_gauss_func(X, T, W)
print("SD={0:.2f} cm".format(np.sqrt(mse)))
plt.show()
```

Out

```
# 실행 결과는 [그림 5-22]를 참조
```

그림 5-22 LOOCV에서 얻은 M=3의 선형 기저 함수 모델의 피팅



이때,

오차의 표준 편차 SD = 4.37cm

LOOCV에서, 기저의 수 M = 3의 경우에 예측 오차가 최소화되었다.

테스트 데이터의 오차의 표준 편차 SD = 6.51cm

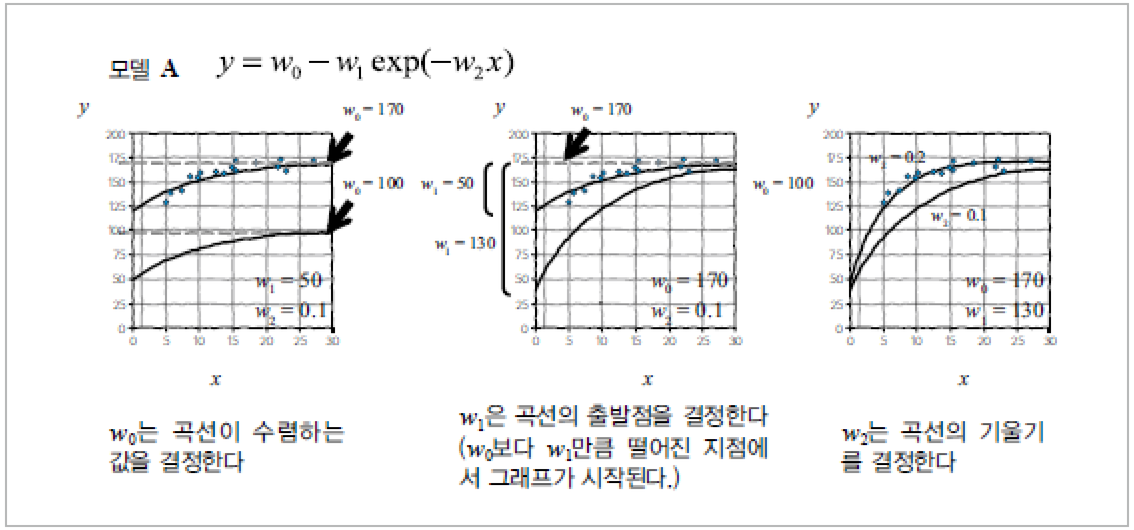
리스트 5-2-(16)

- ‘키는 나이가 들면서 점차 커지고 일정한 곳에서 수렴한다’는 지식을 모델에 추가

$$y(x) = w_0 - w_1 \exp(-w_2 x) \qquad \text{(식 5–71)}$$

- 함수의 성질을 그림으로 나타냄.

그림 5-23 새로운 모델 A



SECTION.06 새로운 모델의 생성

• 데이터에 맞는 매개 변수를 구하기 위해 평균 제곱 오차 J가 최소가 되도록 w0, w1, w2를 선택함.

$$J = \frac{1}{N}\sum_{n=0}^{N-1}(y_n - t_n)^2 \qquad \text{(식 5-72)}$$

• 모델 A의 정의

```
# 리스트 5-2-(17)
# 모델 A -----------------------------------
def model_A(x, w):
    y = w[0] - w[1] * np.exp(-w[2] * x)
    return y


# 모델 A 표시 ------------------------------
def show_model_A(w):
    xb = np.linspace(X_min, X_max, 100)
    y = model_A(xb, w)
    plt.plot(xb, y, c=[.5, .5, .5], lw=4)

# 모델 A의 MSE ------------------------------
def mse_model_A(w, x, t):
    y = model_A(x, w)
    mse = np.mean((y - t)**2)
    return mse
```

• 매개 변수의 최적화

```
# 리스트 5-2-(18)
from scipy.optimize import minimize


# 모델 A의 매개 변수 최적화 -----------------
def fit_model_A(w_init, x, t):
    res1 = minimize(mse_model_A, w_init, args=(x, t), method="powell")
    return res1.x
```

SECTION.06 새로운 모델의 생성

- '키는 나이가 들면서 점차 커지고 일정한 곳에서 수렴한다'는 지식을 모델에 추가

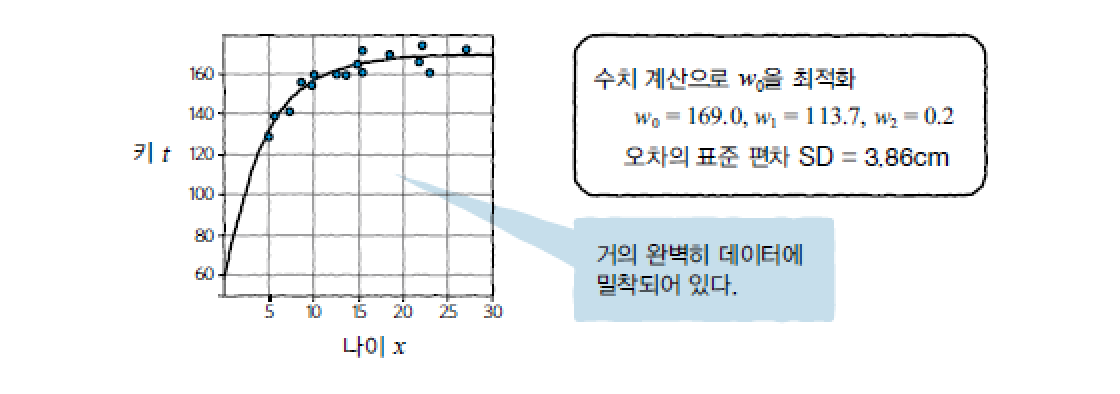
In

```
# 리스트 5-2-(19)
# 메인 ------------------------------------
plt.figure(figsize=(4, 4))
W_init=[100, 0, 0]
W = fit_model_A(W_init, X, T)
print("w0={0:.1f}, w1={1:.1f}, w2={2:.1f}".format(W[0], W[1], W[2]))
show_model_A(W)
plt.plot(X, T, marker='o', linestyle='None',
         color='cornflowerblue',markeredgecolor='black')
plt.xlim(X_min, X_max)
plt.grid(True)
mse = mse_model_A(W, X, T)
print("SD={0:.2f} cm".format(np.sqrt(mse)))
plt.show()
```

Out

```
# 실행 결과는 [그림 5-24]를 참조
```

그림 5-24 모델 A에 의한 피팅



SECTION.07 모델의 선택

- 홀드 아웃 검증과 교차 검증 모델로 모델의 좋고 나쁨을 평가할 수 있음.
- 모델 간의 비교

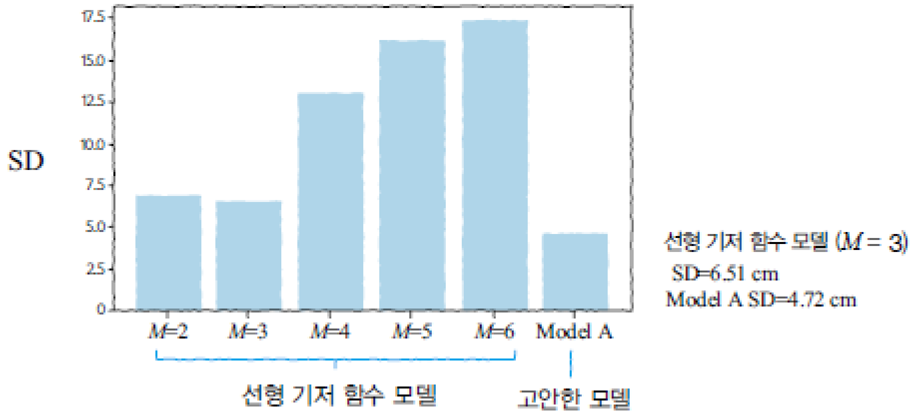
In

```
# 리스트 5-2-(20)
# 교차 검증 model_A ----------------------------
def kfold_model_A(x, t, k):
    n = len(x)
    mse_train = np.zeros(k)
    mse_test = np.zeros(k)
    for i in range(0, k):
        x_train = x[np.fmod(range(n), k) != i]
        t_train = t[np.fmod(range(n), k) != i]
        x_test = x[np.fmod(range(n), k) == i]
        t_test = t[np.fmod(range(n), k) == i]
        wm = fit_model_A(np.array([169, 113, 0.2]), x_train, t_train)
        mse_train[i] = mse_model_A(wm, x_train, t_train)
        mse_test[i] = mse_model_A(wm, x_test, t_test)
    return mse_train, mse_test


# 메인 ------------------------------------
K = 16
Cv_A_train, Cv_A_test = kfold_model_A(X, T, K)
mean_A_test = np.sqrt(np.mean(Cv_A_test))
print("Gauss(M=3) SD={0:.2f} cm".format(mean_Gauss_test[1]))
print("Model A SD={0:.2f} cm".format(mean_A_test))
SD = np.append(mean_Gauss_test[0:5], mean_A_test)
M = range(6)
label = ["M=2", "M=3", "M=4", "M=5", "M=6", "Model A"]
plt.figure(figsize=(5, 3))
plt.bar(M, SD, tick_label=label, align="center",
facecolor="cornflowerblue")
plt.show()
```

Out

```
# 실행 결과는 [그림 5-25]를 참조
```

그림 5-25 선형 기저 함수 모델과 모델 A의 LOOCV의 비교

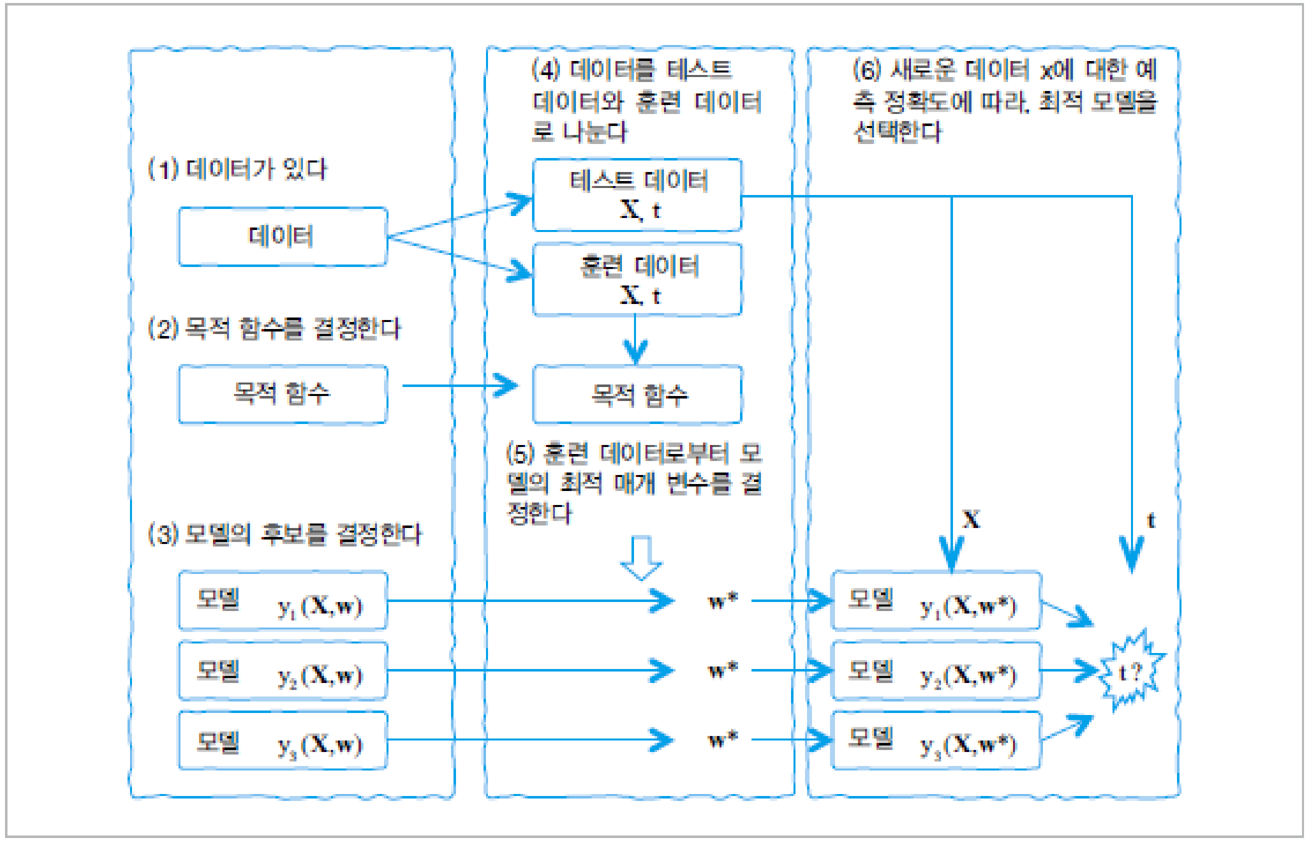


LOOCV 결과, 새로 생각한 모델 A의 테스트 데이터의 오차(SD: 평균 제곱 오차의 평균 제곱근)는 4.72cm였다. 가장 오차가 작았던 $M = 3$의 선형 기저 함수 모델의 SD는 6.51cm이라는 점에서 모델 A는 가우스 기저 함수보다 데이터에 적합한 모델이라는 결론을 지을 수 있다.

SECTION.08 정리

• 지도 학습의 회귀 문제의 해결법 정리.

그림 5-26 지도 학습에 의한 해석(모델 선택)의 흐름



SECTION.08 정리

- 입력 변수와 목표 변수의 데이터가 있음(1)
- 무엇을 가지고 예측의 정확도를 높일지,목적 함수를 결정함(2).
- 모델의 후보를 생각함(3).
- 도입한 모델을 고안할 수 있는지 등을 생각함.
- 홀드 아웃 검증을 한다면, 데이터를 테스트 데이터와 훈련 데이터로 나누어 둠(4).
- 훈련 데이터를 사용하여 원하는 함수가 최소(또는 최대)가 되도록 각 모델의 매개 변수를 결정함(5).
- 모델 매개 변수를 사용하여 가장 오차가 적은 모델을 선택함(6).
- 모델이 결정되면 보유한 데이터를 모두 사용하여 모델 매개 변수를 최적화함.
- 최적화된 모델이 미지의 입력에 대해 가장 유력한 예측 모델이 됨.

지도 학습: 분류

Contents

Contents

지도 학습: 분류

1.1 문제 설정

- 가장 간단한 입력 정보가 1차원이고, 분류할 클래스가 두 가지인 경우를 생각함.
- 데이터를 기초로 무게를 통해 성별을 예측하는 모델을 만드는 것이 목적임.

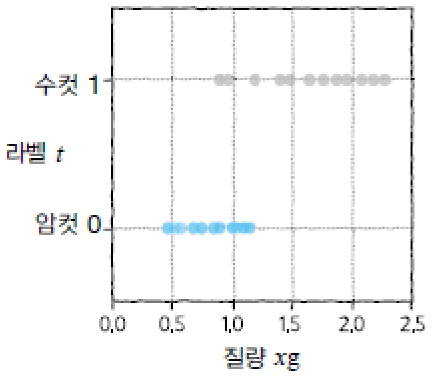
In

```
# 리스트 6-1-(1)
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
# 데이터 생성 --------------------------------
np.random.seed(seed=0) # 난수를 고정
X_min = 0
X_max = 2.5
X_n = 30
X_col = ['cornflowerblue', 'gray']
X = np.zeros(X_n) # 입력 데이터
T = np.zeros(X_n, dtype=np.uint8) # 목표 데이터
Dist_s = [0.4, 0.8] # 분포의 시작 지점
Dist_w = [0.8, 1.6] # 분포의 폭
Pi = 0.5 # 클래스 0의 비율
for n in range(X_n):
    wk = np.random.rand()
    T[n] = 0 * (wk < Pi) + 1 * (wk >= Pi) # (A)
    X[n] = np.random.rand() * Dist_w[T[n]] + Dist_s[T[n]] # (B)
# 데이터 표시 --------------------------------
print('X=' + str(np.round(X, 2)))
print('T=' + str(T))
```

Out

```
X=[1.94 1.67 0.92 1.11 1.41 1.65 2.28 0.47 1.07 2.19 2.08 1.02 0.91 1.16
 1.46 1.02 0.85 0.89 1.79 1.89 0.75 0.9  1.87 0.5  0.69 1.5  0.96 0.53
 1.21 0.6 ]
T=[1 1 0 0 1 1 1 0 0 1 1 0 0 0 1 0 0 0 1 1 0 1 1 0 0 1 1 0 1 0]
```

그림 6-1 어떤 곤충의 질량과 자웅(수컷, 암컷)의 인공 데이터(30마리 분)



질량

```
X= [1.94  1.67  0.92  1.11  1.41
    1.65  2.28  0.47  1.07  2.19
    2.08  1.02  0.91  1.16  1.46
    1.02  0.85  0.89  1.79  1.89
    0.75  0.99  1.87  0.50  0.69
    1.50  0.96  0.53  1.21  0.60]
```

암수

```
T = [1  1  0  0  1
     1  1  0  0  1
     1  0  0  0  1
     0  0  0  1  1
     0  1  1  0  0
     1  1  0  1  0 ]
```

데이터 X, T는 벡터(지면 사정으로 6×3으로 정리함)

1: 수컷
0: 암컷

SECTION.01 1차원 입력 2클래스 분류

• 문제를 푸는 방침은 수컷과 암컷을 분리하는 경계선을 결정하는 것임.

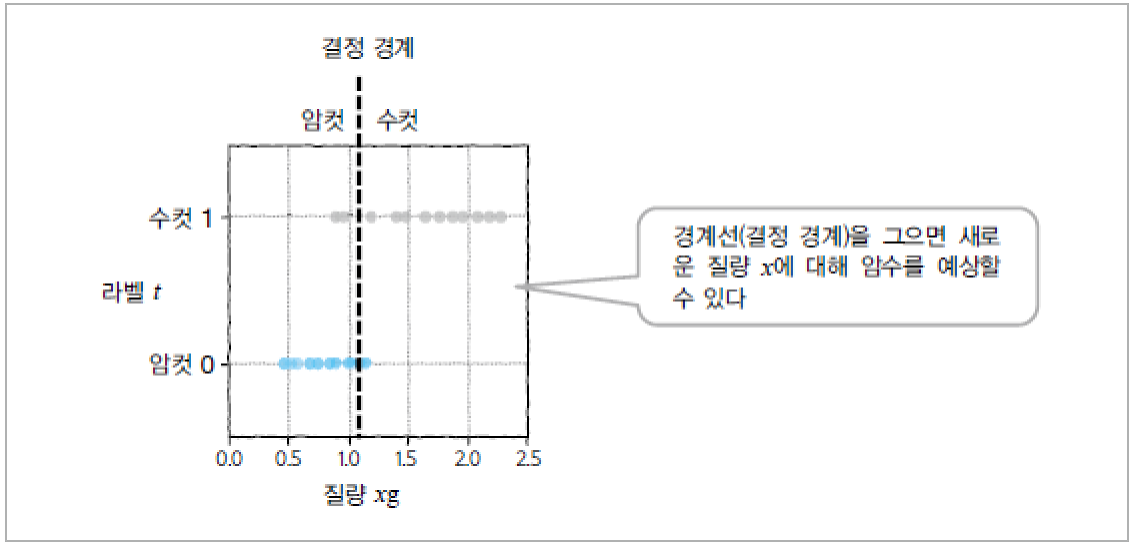
In

```
# 리스트 6-1-(2)
# 데이터 분포 표시 ----------------------------
def show_data1(x, t):
    K = np.max(t) + 1
    for k in range(K): # (A)
        plt.plot(x[t == k], t[t == k], X_col[k], alpha=0.5,
                 linestyle='none', marker='o') # (B)
        plt.grid(True)
        plt.ylim(-.5, 1.5)
        plt.xlim(X_min, X_max)
        plt.yticks([0, 1])
# 메인 ------------------------------------
fig = plt.figure(figsize=(3, 3))
show_data1(X, T)
plt.show()
```

Out

```
# 실행 결과는 [그림 6-2]를 참조
```

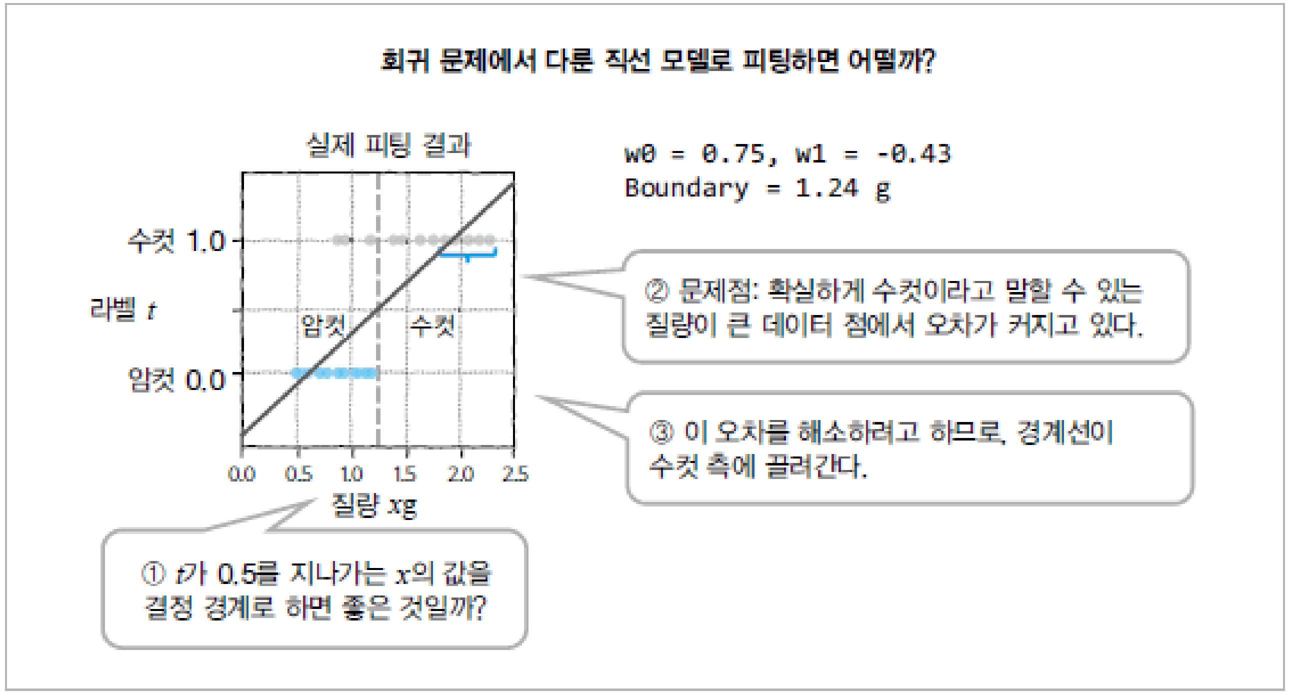
그림 6-2 문제를 풀기 위한 방침



SECTION.01 1차원 입력 2클래스 분류

- 어떻게 결정 경계를 결정하면 좋을까?
- 우선 선형 회귀 모델을 사용하는 것임.

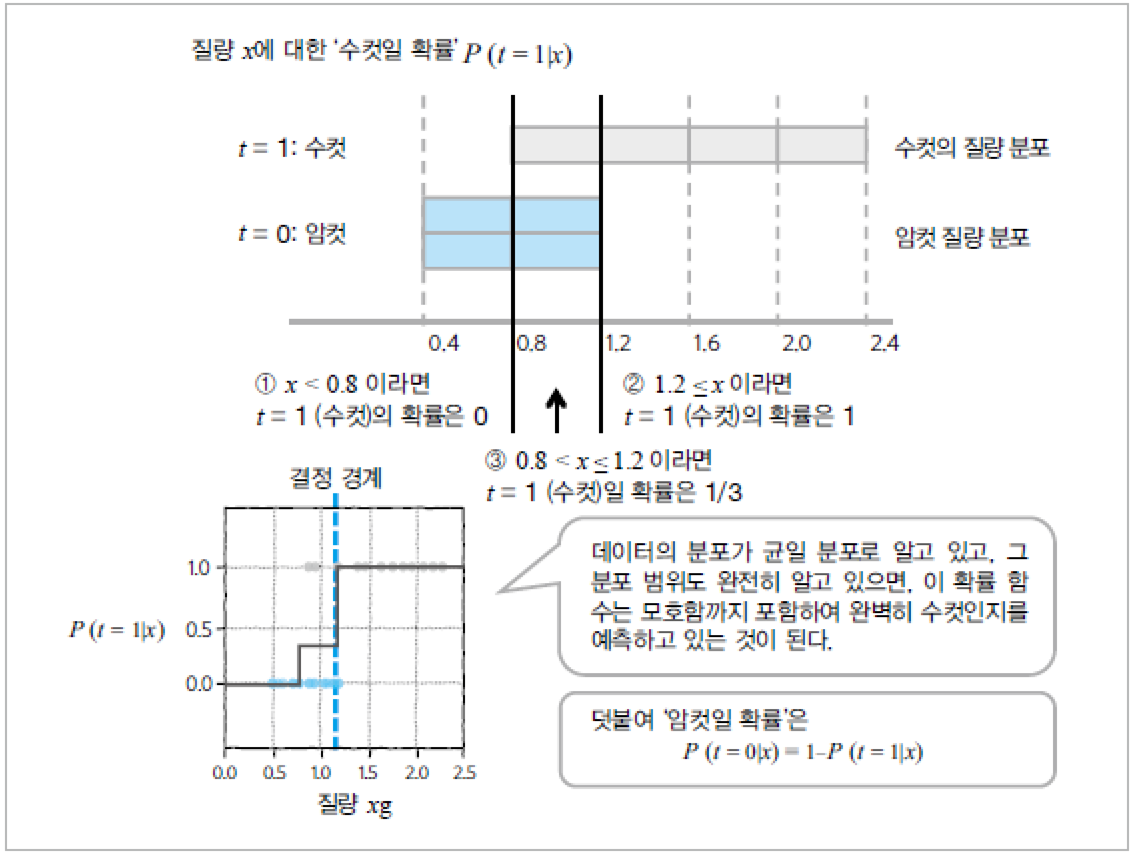
그림 6-3 선형 회귀 모델로 분류 문제를 풀다



1.2 확률로 나타내는 클래스 분류

- ‘확률의 세계’에 들어감.
- “수컷일 확률은 1/3이다.”처럼 모호성을 확률로 포함한 예측은 가능함

그림 6-4 수컷일 확률은



1.3 최대가능도법

- 최대가능도방법(最大可能度方法, maximum likelihood method) 또는 최대우도법(最大尤度法)은 어떤 확률변수에서 표집한 값들을 토대로 그 확률변수의 모수를 구하는 방법임.
- 어떤 모수가 주어졌을 때, 원하는 값들이 나올 가능도를 최대로 만드는 모수를 선택하는 방법임.

그림 6-5 최대가능도법의 개념

문제

t는 0 또는 1의 값을 갖는 데이터이다. 어떤 x의 범위에 주목하여,

"처음 3회가 $t = 0$이고, 4회째는 $t = 1$이다." 라고 하면
$t = 1$이 될 확률은 얼마나 될까?

$P(t = 1|x) = w$로 하여, w를 구하자.

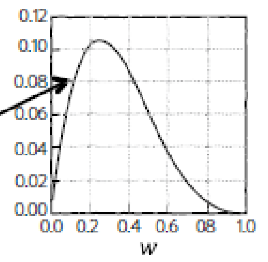
고려할 방식(최대가능도법)
주어진 입력 데이터 x에 대해 라벨 데이터 t가 생성될 확률(가능도)이 가장 커지는 w를 추정치로 한다.

풀이

$t = 0$이 될 확률은 $(1 - w)$이고, $t = 1$이 될 확률은 w이다.
$t = 0$이 3회, $t = 1$이 1회 나올 확률(가능도)는

$$P(\mathbf{T} = 0, 0, 0, 1|x) = (1 - w)(1 - w)(1 - w)w$$

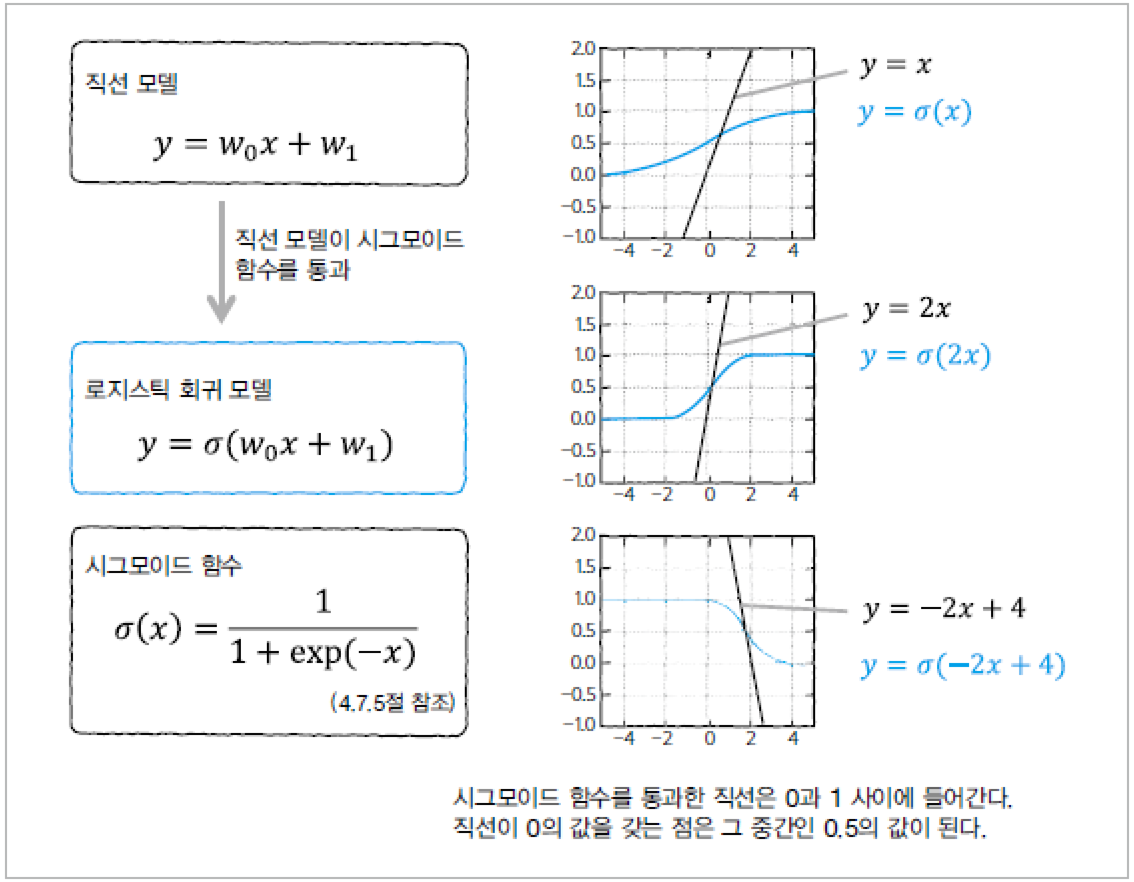
위 식이 최대가 되는 w를 구하면 된다.
답 $w = 0.25$



1.4 로지스틱 회귀 모델

- 로지스틱 회귀(logistic regression)는 D.R.Cox가 1958년에 제안한 확률 모델로서 독립 변수의 선형 결합을 이용하여 사건의 발생 가능성을 예측하는 데 사용되는 통계 기법임.

그림 6-6 로지스틱 회귀 모델



SECTION.01 1차원 입력 2클래스 분류

- 프로그래밍으로 알아보면

[로지스틱 회귀 모델을 정의]

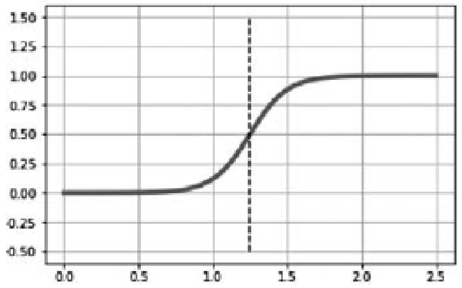
In

```
# 리스트 6-1-(3)
def logistic(x, w):
    y = 1 / (1 + np.exp(-(w[0] * x + w[1])))
    return y
```

In

```
# 리스트 6-1-(4)
def show_logistic(w):
    xb = np.linspace(X_min, X_max, 100)
    y = logistic(xb, w)
    plt.plot(xb, y, color='gray', linewidth=4)
    # 결정 경계
    i = np.min(np.where(y > 0.5)) # (A)
    B = (xb[i - 1] + xb[i]) / 2 # (B)
    plt.plot([B, B], [-.5, 1.5], color='k', linestyle='--')
    plt.grid(True)
    return B


# test
W = [8, -10]
show_logistic(W)
```

Out



1.5 교차 엔트로피 오차

- 로지스틱 회귀 모델의 평균 교차 엔트로피 오차 함수

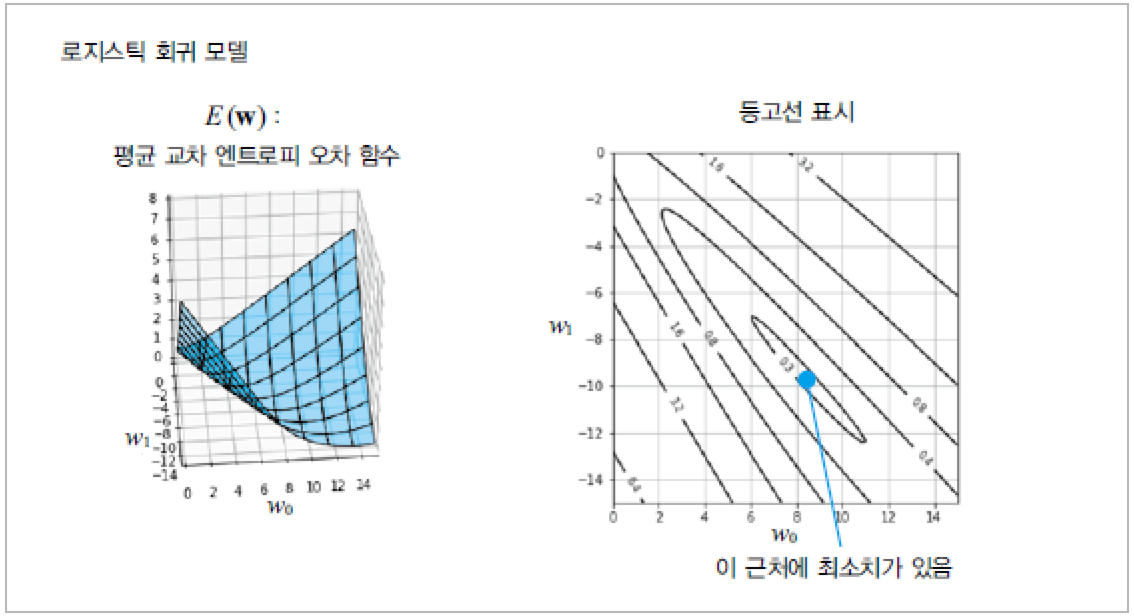
In

```
# 리스트 6-1-(6)
from mpl_toolkits.mplot3d import Axes3D

# 계산 ----------------------------------------
xn = 80 # 등고선 표시 해상도
w_range = np.array([[0, 15], [-15, 0]])
x0 = np.linspace(w_range[0, 0], w_range[0, 1], xn)
x1 = np.linspace(w_range[1, 0], w_range[1, 1], xn)
xx0, xx1 = np.meshgrid(x0, x1)
C = np.zeros((len(x1), len(x0)))
w = np.zeros(2)
for i0 in range(xn):
    for i1 in range(xn):
        w[0] = x0[i0]
        w[1] = x1[i1]
        C[i1, i0] = cee_logistic(w, X, T)


# 표시 ----------------------------------------
plt.figure(figsize=(12, 5))
#plt.figure(figsize=(9.5, 4))
plt.subplots_adjust(wspace=0.5)
ax = plt.subplot(1, 2, 1, projection='3d')
ax.plot_surface(xx0, xx1, C, color='blue', edgecolor='black',
            rstride=10, cstride=10, alpha=0.3)
ax.set_xlabel('$w_0$', fontsize=14)
ax.set_ylabel('$w_1$', fontsize=14)
ax.set_xlim(0, 15)
ax.set_ylim(-15, 0)
ax.set_zlim(0, 8)
ax.view_init(30, -95)


plt.subplot(1, 2, 2)
cont = plt.contour(xx0, xx1, C, 20, colors='black',
              levels=[0.26, 0.4, 0.8, 1.6, 3.2, 6.4])
cont.clabel(fmt='%1.1f', fontsize=8)
plt.xlabel('$w_0$', fontsize=14)
plt.ylabel('$w_1$', fontsize=14)
plt.grid(True)
plt.show()
```

Out

```
# 실행 결과는 [그림 6-7]을 참조
```

그림 6-7 로지스틱 회귀 모델의 평균 교차 엔트로피 오차 함수



1.6 학습 규칙의 도출

- 로지스틱 회귀 모델의 학습

In

```
# 리스트 6-1-(7)
# 평균 교차 엔트로피 오차의 미분 --------------
def dcee_logistic(w, x, t):
    y = logistic(x, w)
    dcee = np.zeros(2)
    for n in range(len(y)):
        dcee[0] = dcee[0] + (y[n] - t[n]) * x[n]
        dcee[1] = dcee[1] + (y[n] - t[n])
    dcee = dcee / X_n
    return dcee


# --- test
W=[1, 1]
dcee_logistic(W, X, T)
```

Out

```
array([0.30857905, 0.39485474])
```

그림 6-8 로지스틱 회귀 모델의 학습

로지스틱 회귀 모델

$$y_n = \sigma(a_n) = \frac{1}{1+\exp(-a_n)} \qquad a_n = w_0 x + w_1 \qquad \text{(식 6-21, 6-22)}$$

평균 교차 엔트로피 오차 함수

$$E(\mathbf{w}) = -\frac{1}{N}\sum_{n=0}^{N-1}\{t_n \log y_n + (1-t_n)\log(1-y_n)\} \qquad \text{(식 6-17)}$$

학습 규칙에 사용하는 편미분

$$\frac{\partial E}{\partial w_0} = \frac{1}{N}\sum_{n=0}^{N-1}(y_n - t_n)x_n \qquad \frac{\partial E}{\partial w_1} = \frac{1}{N}\sum_{n=0}^{N-1}(y_n - t_n) \qquad \text{(식 6-32, 식 6-33)}$$

1.7 경사 하강법에 의한 해

- 경사 하강법으로 로지스틱 회귀 모델의 매개 변수를 찾아봄.

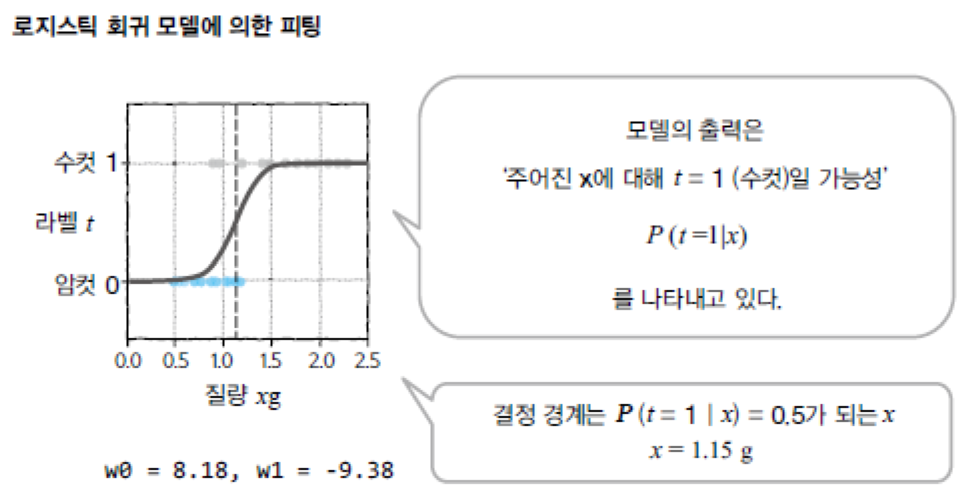
In

```
# 리스트 6-1-(8)
from scipy.optimize import minimize


# 매개 변수 검색
def fit_logistic(w_init, x, t):
    res1 = minimize(cee_logistic, w_init, args=(x, t),
                    jac=dcee_logistic, method="CG") # (A)
    return res1.x


# 메인 ------------------------------------
plt.figure(1, figsize=(3, 3))
W_init=[1,-1]
W = fit_logistic(W_init, X, T)
print("w0 = {0:.2f}, w1 = {1:.2f}".format(W[0], W[1]))
B=show_logistic(W)
show_data1(X, T)
plt.ylim(-.5, 1.5)
plt.xlim(X_min, X_max)
cee = cee_logistic(W, X, T)
print("CEE = {0:.2f}".format(cee))
print("Boundary = {0:.2f} g".format(B))
plt.show()
```

Out

```
# 실행 결과는 [그림 6-9]를 참조
```

그림 6-9 로지스틱 회귀 모델에 의한 피팅 결과




```
w0 = 8.18, w1 = -9.38
CEE = 0.25
Boundary = 1.15 g
```

2.1 문제 설정

- 2클래스의 분류와 3클래스의 분류 데이터를 함께 만듬.

In

```
# 리스트 6-2-(1)
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline


# 데이터 생성 ---------------------------------
np.random.seed(seed=1)  # 난수를 고정
N = 100 # 데이터의 수
K = 3 # 분포 수
T3 = np.zeros((N, 3), dtype=np.uint8)
T2 = np.zeros((N, 2), dtype=np.uint8)
X = np.zeros((N, 2))
X_range0 = [-3, 3] # X0 범위 표시용
X_range1 = [-3, 3] # X1 범위 표시용
Mu = np.array([[-.5, -.5], [.5, 1.0], [1, -.5]]) # 분포의 중심
Sig = np.array([[.7, .7], [.8, .3], [.3, .8]]) # 분포의 분산
Pi = np.array([0.4, 0.8, 1]) # (A) 각 분포에 대한 비율 0.4 0.8 1
for n in range(N):
    wk = np.random.rand()
    for k in range(K): # (B)
        if wk < Pi[k]:
            T3[n, k] = 1
            break
    for k in range(2):
        X[n, k] = (np.random.randn() * Sig[T3[n, :] == 1, k]
                   + Mu[T3[n, :] == 1, k])
T2[:, 0] = T3[:, 0]
T2[:, 1] = T3[:, 1] | T3[:, 2]
```

SECTION.02 2차원 입력 2클래스 분류

- 입력 데이터 X의 첫 5개를 살펴봄.

In
```
# 리스트 6-2-(2)
print(X[:5,:])
```

Out
```
[[-0.14173827  0.86533666]
 [-0.86972023 -1.25107804]
 [-2.15442802  0.29474174]
 [ 0.75523128  0.92518889]
 [-1.10193462  0.74082534]]
```

- 클래스 데이터 T2의 처음 5개는 다음과 같음.

In
```
# 리스트 6-2-(3)
print(T2[:5,:])
```

Out
```
[[0 1]
 [1 0]
 [1 0]
 [0 1]
 [1 0]]
```

SECTION.02 2차원 입력 2클래스 분류

• 클래스 데이터 T3의 처음 5개는 다음과 같음.

In

```
# 리스트 6-2-(4)
print(T3[:5,:])
```

Out

```
[[0 1 0]
 [1 0 0]
 [1 0 0]
 [0 1 0]
 [1 0 0]]
```

SECTION.02 2차원 입력 2클래스 분류

- T2와 T3를 그림으로 그리기.
- 2차원 입력에 대한 인공 데이터

In

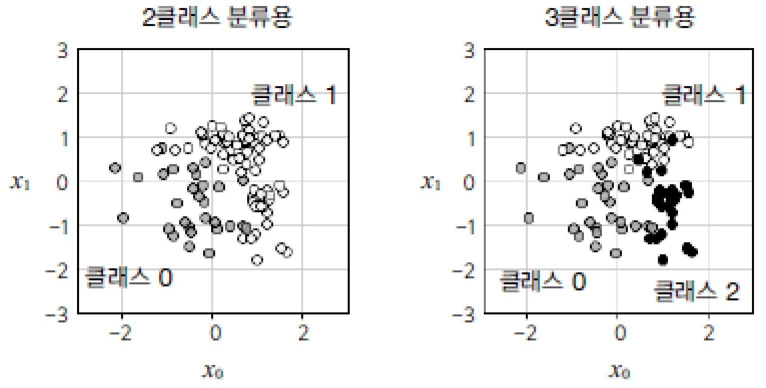
```
# 리스트 6-2-(5)
# 데이터 표시 --------------------------
def show_data2(x, t):
    wk, K = t.shape
    c = [[.5, .5, .5], [1, 1, 1], [0, 0, 0]]
    for k in range(K):
        plt.plot(x[t[:, k] == 1, 0], x[t[:, k] == 1, 1],
                 linestyle='none', markeredgecolor='black',
                 marker='o', color=c[k], alpha=0.8)
        plt.grid(True)


# 메인 ------------------------------
plt.figure(figsize=(7.5, 3))
plt.subplots_adjust(wspace=0.5)
plt.subplot(1, 2, 1)
show_data2(X, T2)
plt.xlim(X_range0)
plt.ylim(X_range1)


plt.subplot(1, 2, 2)
show_data2(X, T3)
plt.xlim(X_range0)
plt.ylim(X_range1)
plt.show()
```

Out

```
# 실행 결과는 [그림 6-10]을 참조
```

그림 6-10 2차원 입력에 대한 인공 데이터

데이터 수
$N = 100$



입력 데이터

```
X=[[-0.14  0.87]
   [-0.87 -1.25]
   [-2.15  0.29]
   [ 0.76  0.93]
   [-1.10  0.74]
   [ 0.24  0.88]
   [ 1.22 -0.20]
    ...         ]
```

클래스 데이터

```
T2=[[0 1]
    [1 0]
    [1 0]
    [0 1]
    [1 0]
    [0 1]
    [0 1]
    ... ]
```

1-of-K 부호화

클래스 데이터

```
T3=[[0 1 0]
    [1 0 0]
    [1 0 0]
    [0 1 0]
    [1 0 0]
    [0 1 0]
    [0 0 1]
     ... ]
```

1-of-K 부호화

SECTION.02 2차원 입력 2클래스 분류

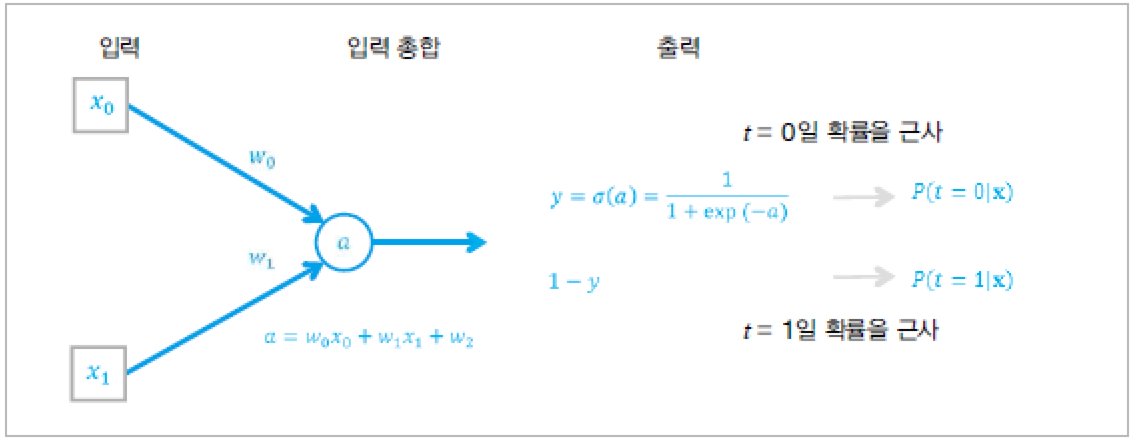
2.2 로지스틱 회귀 모델

- 로지스틱 회귀 모델은 1차원 입력 버전에서 간단히 2차원 입력 버전으로 확장할 수 있음.

$$y = \sigma(a) \longrightarrow a = w_0 x_0 + w_1 x_1 + w_2$$

- 2차원 입력 2클래스 분류의 로지스틱 회귀 모델

그림 6-11 2차원 입력 2클래스 분류의 로지스틱 회귀 모델



SECTION.02 2차원 입력 2클래스 분류

• 모델을 정의함.

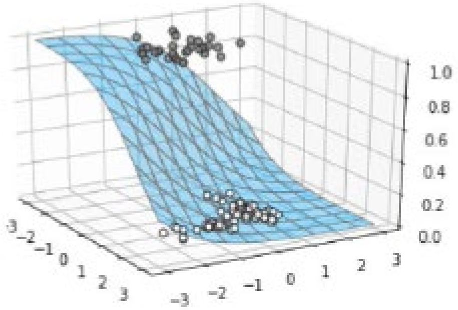
```
In   # 리스트 6-2-(6)
     # 로지스틱 회귀 모델 -----------------
     def logistic2(x0, x1, w):
         y = 1 / (1 + np.exp(-(w[0] * x0 + w[1] * x1 + w[2])))
         return y
```

• 로지스틱 회귀 모델과 데이터를 3차원으로 표시.

```
In   # 리스트 6-2-(7)
     # 모델 3D보기 ------------------------------
     from mpl_toolkits.mplot3d import axes3d
     def show3d_logistic2(ax, w):
         xn = 50
         x0 = np.linspace(X_range0[0], X_range0[1], xn)
         x1 = np.linspace(X_range1[0], X_range1[1], xn)
         xx0, xx1 = np.meshgrid(x0, x1)
         y = logistic2(xx0, xx1, w)
         ax.plot_surface(xx0, xx1, y, color='blue', edgecolor='gray',
                         rstride=5, cstride=5, alpha=0.3)


     def show_data2_3d(ax, x, t):
         c = [[.5, .5, .5], [1, 1, 1]]
         for i in range(2):
             ax.plot(x[t[:, i] == 1, 0], x[t[:, i] == 1, 1], 1 - i,
                     marker='o', color=c[i], markeredgecolor='black',
                     linestyle='none', markersize=5, alpha=0.8)
         Ax.view_init(elev=25, azim=-30)
```

```
     # test ---
     Ax = plt.subplot(1, 1, 1, projection='3d')
     W=[-1, -1, -1]
     show3d_logistic2(Ax, W)
     show_data2_3d(Ax,X,T2)
```

Out



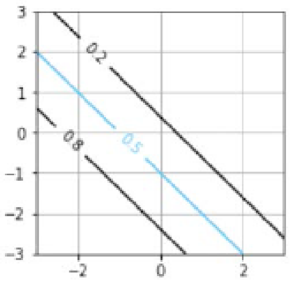
SECTION.02 2차원 입력 2클래스 분류

• 로지스틱 회귀 모델의 출력이 등고선으로 표시.

In

```
# 리스트 6-2-(8)
# 모델 등고선 2D 표시 ------------------------
def show_contour_logistic2(w):
    xn = 30 # 매개 변수의 분할 수
    x0 = np.linspace(X_range0[0], X_range0[1], xn)
    x1 = np.linspace(X_range1[0], X_range1[1], xn)
    xx0, xx1 = np.meshgrid(x0, x1)
    y = logistic2(xx0, xx1, w)
    cont = plt.contour(xx0, xx1, y, levels=(0.2, 0.5, 0.8),
                       colors=['k', 'cornflowerblue', 'k'])
    cont.clabel(fmt='%1.1f', fontsize=10)
    plt.grid(True)


# test ---
plt.figure(figsize=(3,3))
W=[-1, -1, -1]
show_contour_logistic2(W)
```

Out



SECTION.02 2차원 입력 2클래스 분류

- 로지스틱 회귀 모델의 매개 변수를 구하고, 결과를 표시

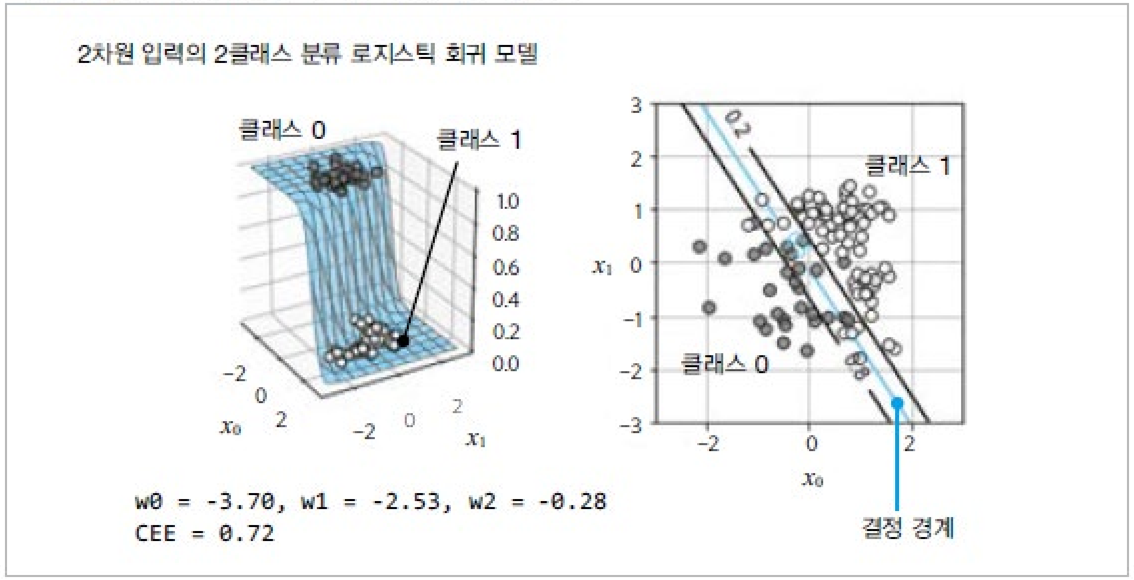
In

```
# 리스트 6-2-(11)
from scipy.optimize import minimize


# 로지스틱 회귀 모델의 매개 변수 검색 -----
def fit_logistic2(w_init, x, t):
    res = minimize(cee_logistic2, w_init, args=(x, t),
                   jac=dcee_logistic2, method="CG")
    return res.x


# 메인 ------------------------------------
plt.figure(1, figsize=(7, 3))
plt.subplots_adjust(wspace=0.5)


Ax = plt.subplot(1, 2, 1, projection='3d')
W_init = [-1, 0, 0]
W = fit_logistic2(W_init, X, T2)
print("w0 = {0:.2f}, w1 = {1:.2f}, w2 = {2:.2f}".format(W[0], W[1], W[2]))
show3d_logistic2(Ax, W)


show_data2_3d(Ax, X, T2)
cee = cee_logistic2(W, X, T2)
print("CEE = {0:.2f}".format(cee))


Ax = plt.subplot(1, 2, 2)
show_data2(X, T2)
show_contour_logistic2(W)
plt.show()
```

Out

```
# 실행 결과는 [그림 6-12]를 참조
```

그림 6-12 2차원 입력의 로지스틱 회귀 모델에 의한 피팅 결과

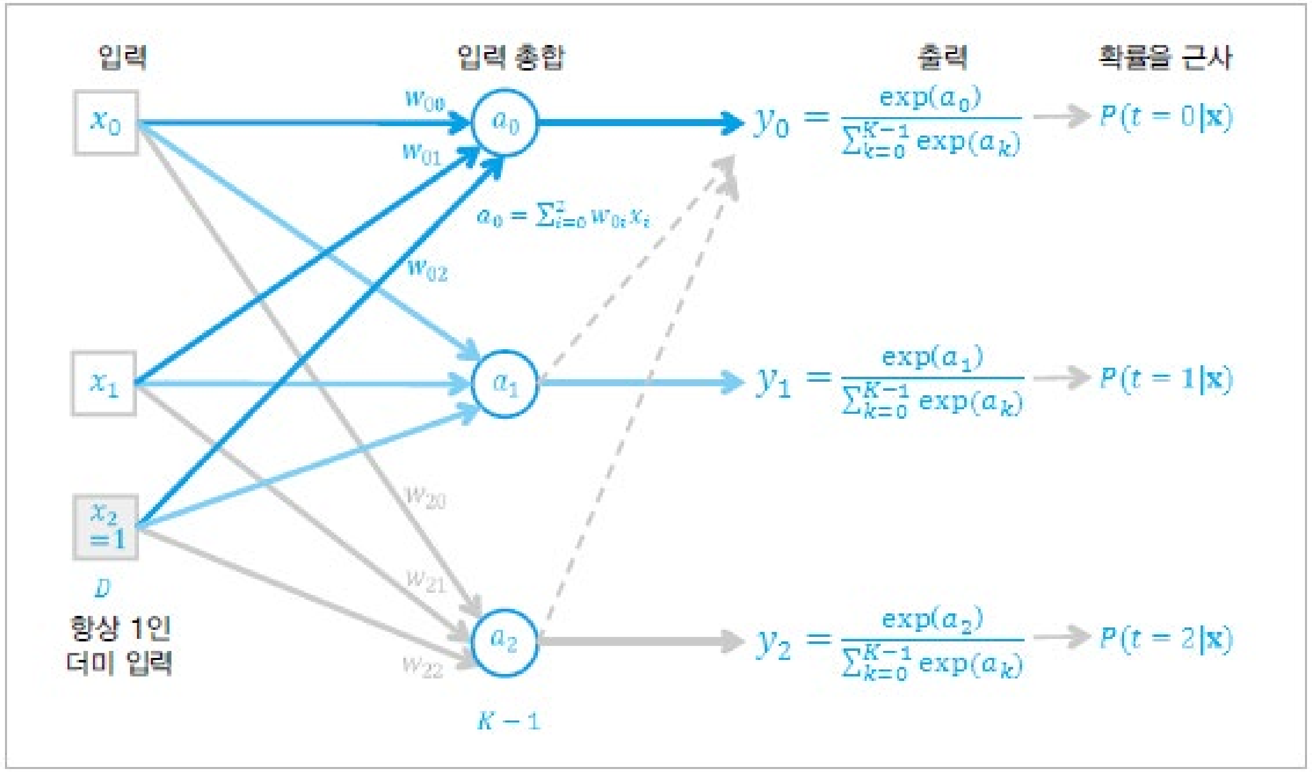


SECTION.03 2차원 입력 3클래스 분류

3.1 3클래스 분류 로지스틱 회귀 모델

- 3클래스 이상의 클래스 분류에 대응 가능함.

그림 6-13 2차원 입력 3클래스 분류의 로지스틱 회귀 모델



SECTION.03 2차원 입력 3클래스 분류

- 3클래스용 로지스틱 회귀 모델 logistic3을 구현함.

In

```
# 리스트 6-2-(12)
# 3클래스용 로지스틱 회귀 모델 ------------------
def logistic3(x0, x1, w):
    K = 3
    w = w.reshape((3, 3))
    n = len(x1)
    y = np.zeros((n, K))
    for k in range(K):
        y[:, k] = np.exp(w[k, 0] * x0 + w[k, 1] * x1 + w[k, 2])
    wk = np.sum(y, axis=1)
    wk = y.T / wk
    y = wk.T
    return y


# test ---
W = np.array([1, 2, 3, 4 ,5, 6, 7, 8, 9])
y = logistic3(X[:3, 0], X[:3, 1], W)
print(np.round(y, 3))
```

Out

```
[[0.    0.006 0.994]
 [0.965 0.033 0.001]
 [0.925 0.07  0.005]]
```

3.2 교차 엔트로피 오차

- 교차 엔트로피 오차를 계산하는 함수인 cee_logistic3을 정의해 봄.

In

```
# 리스트 6-2-(13)
# 교차 엔트로피 오차 ------------
def cee_logistic3(w, x, t):
    X_n = x.shape[0]
    y = logistic3(x[:, 0], x[:, 1], w)
    cee = 0
    N, K = y.shape
    for n in range(N):
        for k in range(K):
            cee = cee - (t[n, k] * np.log(y[n, k]))
    cee = cee / X_n
    return cee


# test ----
W = np.array([1, 2, 3, 4 ,5, 6, 7, 8, 9])
cee_logistic3(W, X, T3)
```

Out

```
3.9824582404787288
```

3.3 경사 하강법에 의한 해

- 각 매개 변수에 대한 미분값을 출력하는 함수 dcee_logistic3임.

In
```
# 리스트 6-2-(14)
# 교차 엔트로피 오차의 미분 ------------
def dcee_logistic3(w, x, t):
    X_n = x.shape[0]
    y = logistic3(x[:, 0], x[:, 1], w)
    dcee = np.zeros((3, 3)) # (클래스의 수 K) x (x의 차원 D+1)
    N, K = y.shape
    for n in range(N):
        for k in range(K):
            dcee[k, :] = dcee[k, :] - (t[n, k] - y[n, k])* np.r_[x[n, :], 1]
    dcee = dcee / X_n
    return dcee.reshape(-1)


# test ----
W = np.array([1, 2, 3, 4 ,5, 6, 7, 8, 9])
dcee_logistic3(W, X, T3)
```

Out
```
array([ 0.03778433,  0.03708109, -0.1841851 , -0.21235188, -0.44408101,
       -0.38340835,  0.17456754,  0.40699992,  0.56759346])
```

SECTION.03 2차원 입력 3클래스 분류

- minimize ()에 전달하여 매개 변수 검색을 수행하는 함수를 만듬.

```
In  # 리스트 6-2-(15)
    # 매개 변수 검색 -----------------
    def fit_logistic3(w_init, x, t):
        res = minimize(cee_logistic3, w_init, args=(x, t),
                       jac=dcee_logistic3, method="CG")
        return res.x
```

- 등고선에 결과를 표시하는 함수 show_contour_logistic3도 만듬.

```
In  # 리스트 6-2-(16)
    # 모델 등고선 2D 표시 --------------------
    def show_contour_logistic3(w):
        xn = 30 # 매개 변수의 분할 수
        x0 = np.linspace(X_range0[0], X_range0[1], xn)
        x1 = np.linspace(X_range1[0], X_range1[1], xn)

        xx0, xx1 = np.meshgrid(x0, x1)
        y = np.zeros((xn, xn, 3))
        for i in range(xn):
        wk = logistic3(xx0[:, i], xx1[:, i], w)
        for j in range(3):
            y[:, i, j] = wk[:, j]
    for j in range(3):
        cont = plt.contour(xx0, xx1, y[:, :, j],
                           levels=(0.5, 0.9),
                           colors=['cornflowerblue', 'k'])
        cont.clabel(fmt='%1.1f', fontsize=9)
    plt.grid(True)
```