파이썬으로 배우는
머신러닝의 교과서

Contents

Contents

CHAPTER 09 비지도 학습

‘비지도 학습’의 클러스터링 알고리즘을 설명.

SECTION.01 2차원 입력 데이터

- 비지도 학습, 또는 자율 학습(Unsupervised Learning)은 머신러닝의 일종으로, 데이터가 어떻게 구성되었는지를 알아내는 문제에 속함.
- 이 방법은 지도 학습(Supervised Learning)과는 달리 입력값에 대한 목표치가 주어지지 않음.
- 클래스 정보 없이, 입력 데이터가 비슷한 것끼리 클래스로 나누는 것이 클러스터링 임.

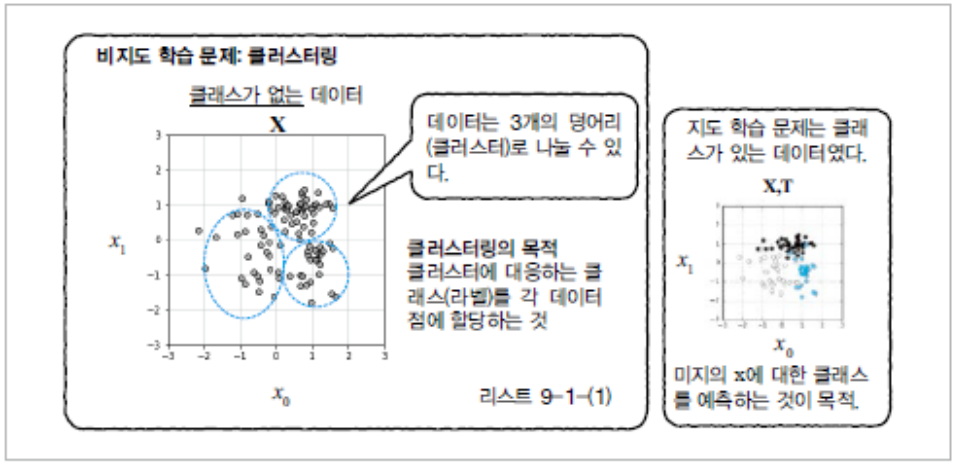
In

```
# ------------------ 리스트 9-1-(1)
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# 데이터 생성 ---------------------------------
np.random.seed(1)
N = 100
K = 3
T3 = np.zeros((N, 3), dtype=np.uint8)
X = np.zeros((N, 2))
X_range0 = [-3, 3]
X_range1 = [-3, 3]
X_col = ['cornflowerblue', 'black', 'white']
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]) # 누적 확률
for n in range(N):
    wk = np.random.rand()
    for k in range(K):
        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])

# 데이터를 그리기 ------------------------------
def show_data(x):
    plt.plot(x[:, 0], x[:, 1], linestyle='none',
             marker='o', markersize=6,
             markeredgecolor='black', color='gray', alpha=0.8)
    plt.grid(True)

# 메인 ----------------------------------------
plt.figure(1, figsize=(4, 4))
show_data(X)
plt.xlim(X_range0)
plt.ylim(X_range1)
plt.show()
np.savez('data_ch9.npz', X=X, X_range0=X_range0,
         X_range1=X_range1)
```

Out

```
# 실행 결과는 [그림 9-1]을 참조
```

그림 9-1 클러스터링

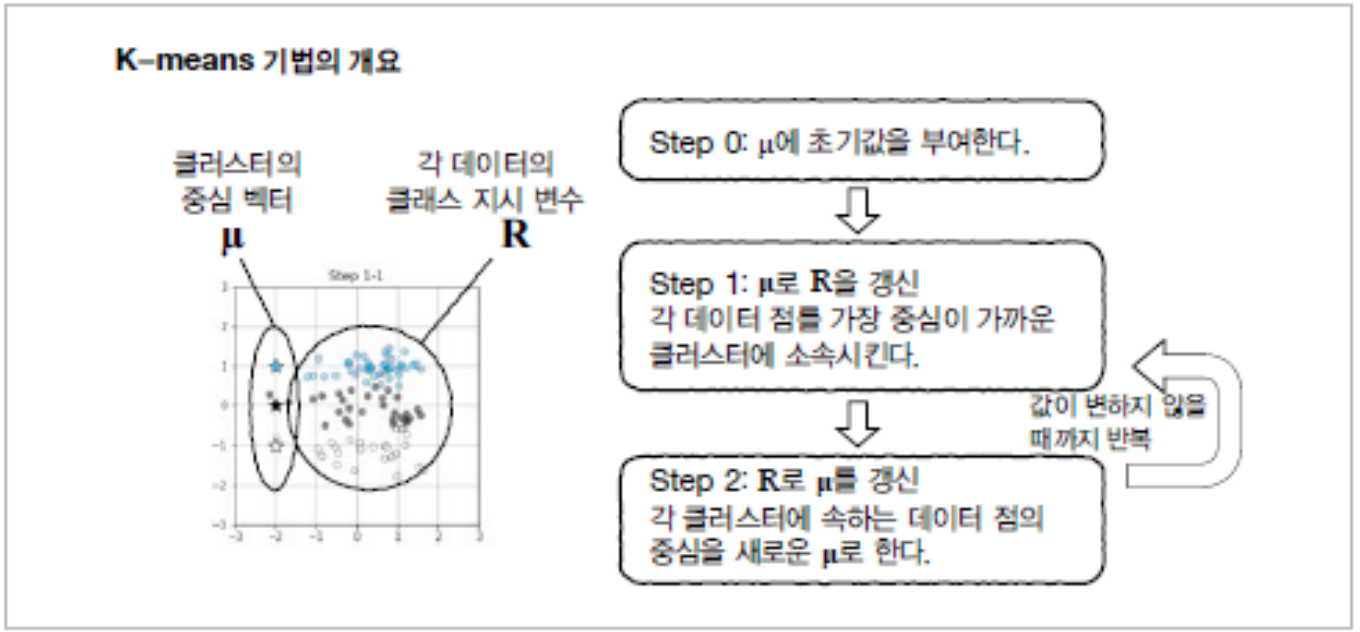


SECTION.02 K-means 기법

2.1 K-means 기법의 개요

- 단계를 순서대로 설명

그림 9-2 K-means 기법



2.2 Step 0: 변수의 준비와 초기화

- 프로그램으로 구현, 실행해서 확인

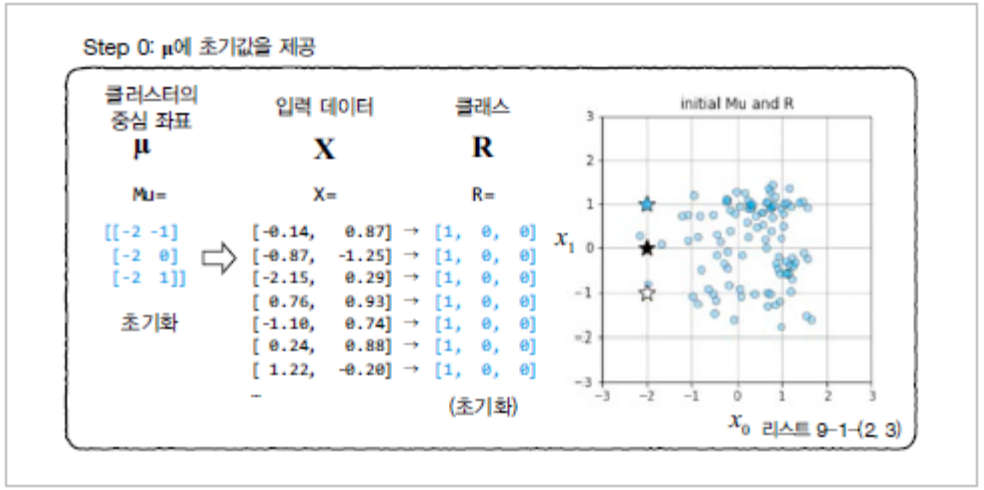
In

```
# ---------------- 리스트 9-1-(3)
# 데이터를 그리는 함수 --------------------------
def show_prm(x, r, mu, col):
    for k in range(K):
        # 데이터 분포의 묘사
        plt.plot(x[r[:, k] == 1, 0], x[r[:, k] == 1, 1],
                 marker='o',
                 markerfacecolor=X_col[k], markeredgecolor='k',
                 markersize=6, alpha=0.5, linestyle='none')
        # 데이터의 평균을 '별표'로 묘사
        plt.plot(mu[k, 0], mu[k, 1], marker='*',
                 markerfacecolor=X_col[k], markersize=15,
                 markeredgecolor='k', markeredgewidth=1)
    plt.xlim(X_range0)
    plt.ylim(X_range1)
    plt.grid(True)


# ----------------------------
plt.figure(figsize=(4, 4))
R = np.c_[np.ones((N, 1)), np.zeros((N, 2))]
show_prm(X, R, Mu, X_col)
plt.title('initial Mu and R')
plt.show()
```

Out

```
# 실행 결과는 [그림 9-3]을 참조
```

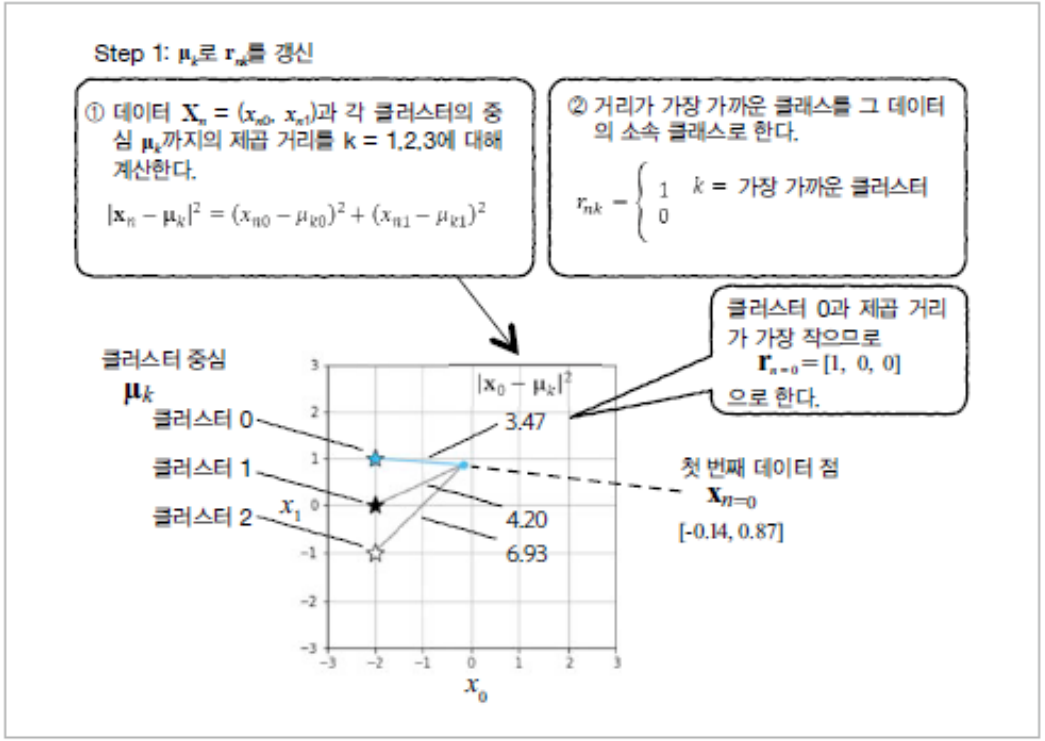
그림 9-3 Step 0: 매개 변수의 초기화



2.3 Step 1: R의 갱신

- 갱신 방법은 "각 데이터 점을 가장 중심이 가까운 클러스터에 넣는다." 임.

그림 9-4 Step 1: 첫 번째 데이터 ($n=0$)의 $\mathbf{r}_n$을 갱신



SECTION.02 K-means 기법

- 모든 데이터에 대해 수행함.

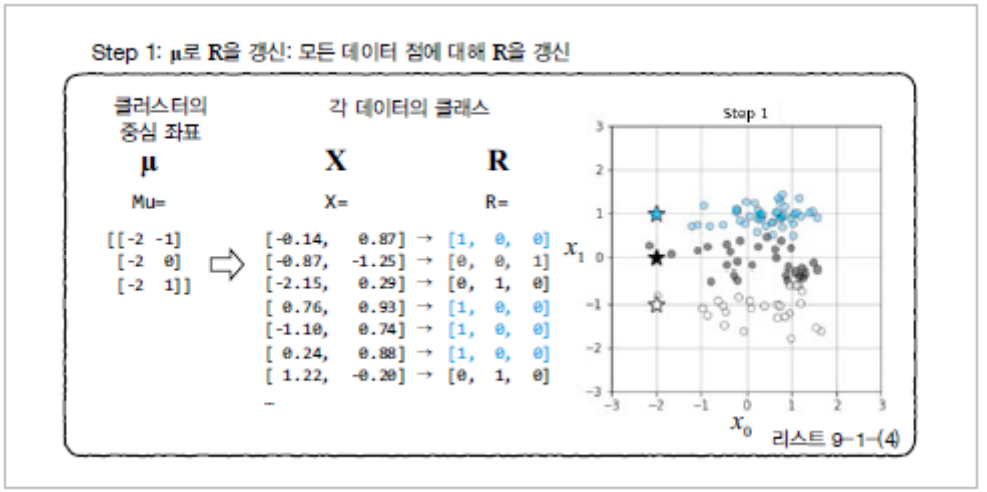
In

```
# ---------------- 리스트 9-1-(4)
# r을 정한다 (Step 1) ----------

def step1_kmeans(x0, x1, mu):
    N = len(x0)
    r = np.zeros((N, K))
    for n in range(N):
        wk = np.zeros(K)
        for k in range(K):
            wk[k] = (x0[n] - mu[k, 0])**2 + (x1[n] - mu[k, 1])**2
        r[n, np.argmin(wk)] = 1
    return r


# ----------------------------
plt.figure(figsize=(4, 4))
R = step1_kmeans(X[:, 0], X[:, 1], Mu)
show_prm(X, R, Mu, X_col)
plt.title('Step 1')
plt.show()
```

Out

```
# 실행 결과는 [그림 9-5]를 참조
```

그림 9-5 Step 1: 모든 데이터에 대해 R을 갱신



2.4 Step 2: μ의 갱신

- 갱신 방법은 "각 클러스터에 속하는 데이터 점의 중심을 새로운 μ로 한다." 임.

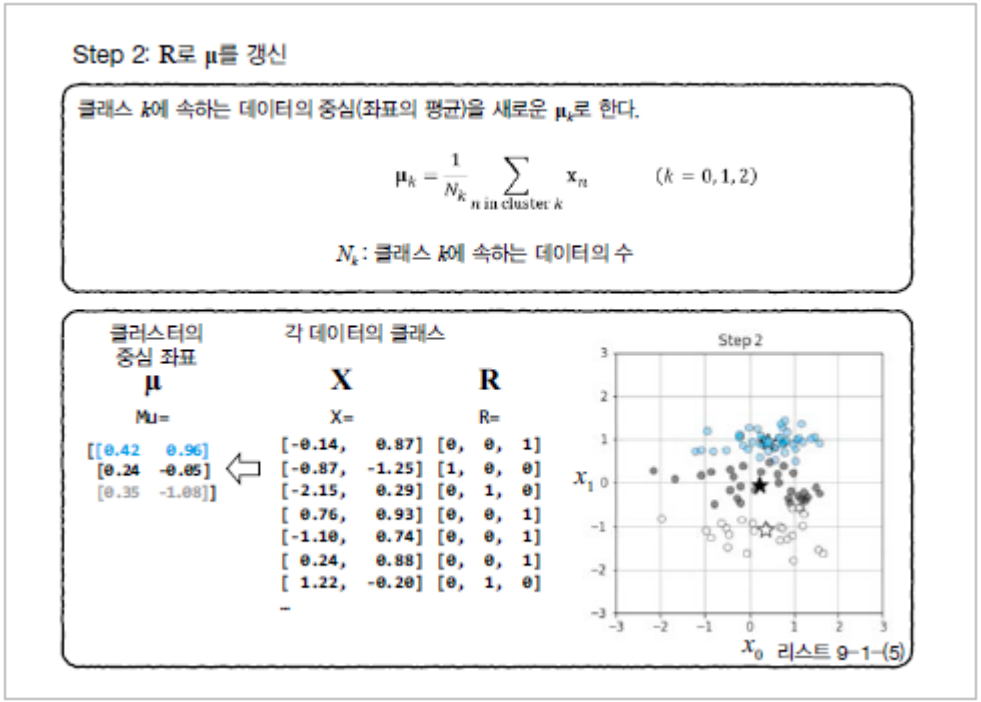
In

```
# ---------------- 리스트 9-1-(5)
# Mu 결정 (Step 2) ----------
def step2_kmeans(x0, x1, r):
    mu = np.zeros((K, 2))
    for k in range(K):
        mu[k, 0] = np.sum(r[:, k] * x0) / np.sum(r[:, k])
        mu[k, 1] = np.sum(r[:, k] * x1) / np.sum(r[:, k])
    return mu


# ------------------------------
plt.figure(figsize=(4, 4))
Mu = step2_kmeans(X[:, 0], X[:, 1], R)
show_prm(X, R, Mu, X_col)

plt.title('Step2')
plt.show()
```

Out

```
# 실행 결과는 [그림 9-6] 참조
```

그림 9-6 Step 2: μ의 갱신



2.4 Step 2: μ의 갱신

- K-means 기법에 의한 클러스터링 과정
- Step 1과 Step 2의 절차를 반복할 뿐임. 그리고 변수의 값이 변화하지 않으면 프로그램을 종료함.

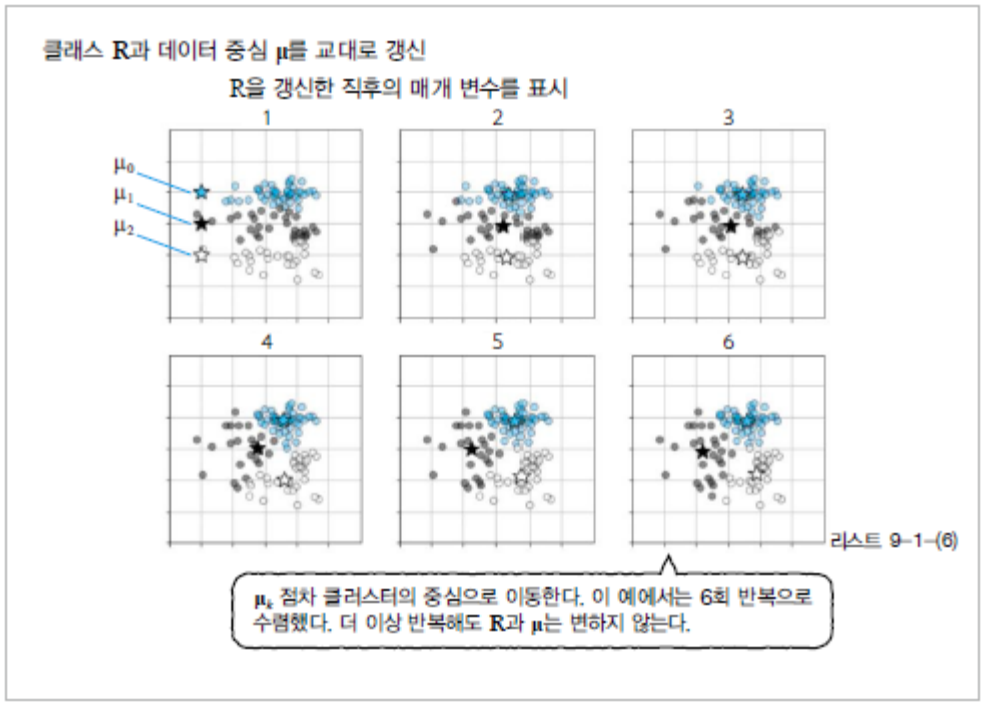
In

```
# ------------------ 리스트 9-1-(6)
plt.figure(1, figsize=(10, 6.5))

Mu = np.array([[-2, 1], [-2, 0], [-2, -1]])
max_it = 6 # 반복 횟수
for it in range(0, max_it):
    plt.subplot(2, 3, it + 1)
    R = step1_kmeans(X[:, 0], X[:, 1], Mu)
    show_prm(X, R, Mu, X_col)
    plt.title("{0:d}".format(it + 1))
    plt.xticks(range(X_range0[0], X_range0[1]), "")
    plt.yticks(range(X_range1[0], X_range1[1]), "")
    Mu = step2_kmeans(X[:, 0], X[:, 1], R)
plt.show()
```

Out

```
# 실행 결과는 [그림 9-7]을 참조
```

그림 9-7 K-means 기법에 의한 클러스터링 과정



2.5 왜곡 척도

- 프로그램을 실행시키면, 초기값의 왜곡 척도가 표시됨.
- 이 함수로 K-means 기법의 반복에 의한 왜곡 척도를 계산함.

In

```
# ------------------ 리스트 9-1-(7)
# 목적 함수 ----------------------------------
def distortion_measure(x0, x1, r, mu):
    # 입력은 2차원으로 제한하고 있다
    N = len(x0)
    J = 0
    for n in range(N):
        for k in range(K):
            J = J + r[n, k] * ((x0[n] - mu[k, 0])**2

                               + (x1[n] - mu[k, 1])**2)
    return J


# ---- test
# ---- Mu와 R의 초기화
Mu = np.array([[-2, 1], [-2, 0], [-2, -1]])
R = np.c_[np.ones((N, 1), dtype=int), np.zeros((N, 2), dtype=int)]
distortion_measure(X[:, 0], X[:, 1], R, Mu)
```

Out

```
771.70911703348781
```

In

```
# ------------------ 리스트 9-1-(8)
# Mu와 R의 초기화
N=X.shape[0]
K=3
Mu = np.array([[-2, 1], [-2, 0], [-2, -1]])
R = np.c_[np.ones((N, 1), dtype=int), np.zeros((N, 2), dtype=int)]
max_it = 10
it = 0
DM = np.zeros(max_it) # 왜곡 척도의 계산 결과를 넣는다
for it in range(0, max_it): # K-means 기법
    R = step1_kmeans(X[:, 0], X[:, 1], Mu)
    DM[it] = distortion_measure(X[:, 0], X[:, 1], R, Mu) # 왜곡 척도
    Mu = step2_kmeans(X[:, 0], X[:, 1], R)
print(np.round(DM, 2))
plt.figure(2, figsize=(4, 4))
plt.plot(DM, color='black', linestyle='-', marker='o')
plt.ylim(40, 80)
plt.grid(True)
plt.show()
```

Out

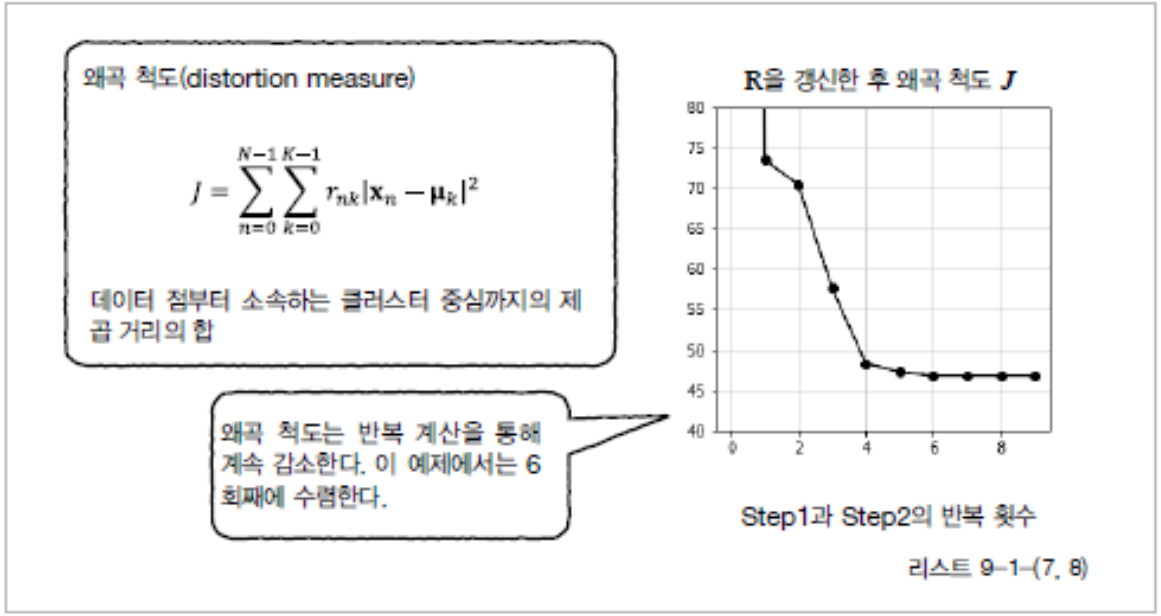
```
# 실행 결과는 [그림 9-8]을 참조
```

SECTION.02 K-means 기법

• 다양한 μ에서 시작하여 얻은 결과 중에 가장 왜곡 척도가 작은 결과를 사용하는 방법이 사용됨.

•

그림 9-8 왜곡 척도



SECTION.03 가우시안 혼합 모델

3.1 확률적 클러스터링

- 가우시안 혼합 모델(Gaussian Mixture Model; GMM)은 이름 그대로 Gaussian 분포가 여러 개 혼합된 클러스터링(clustering) 알고리즘임.
- 확률의 개념을 도입하면 좋음.
- 확률적 클러스터링은 데이터의 배후에 숨어 있는 잠재 변수 z를 확률적으로 γ으로 추정하는 것임.

그림 9-9 확률 모델로 확장

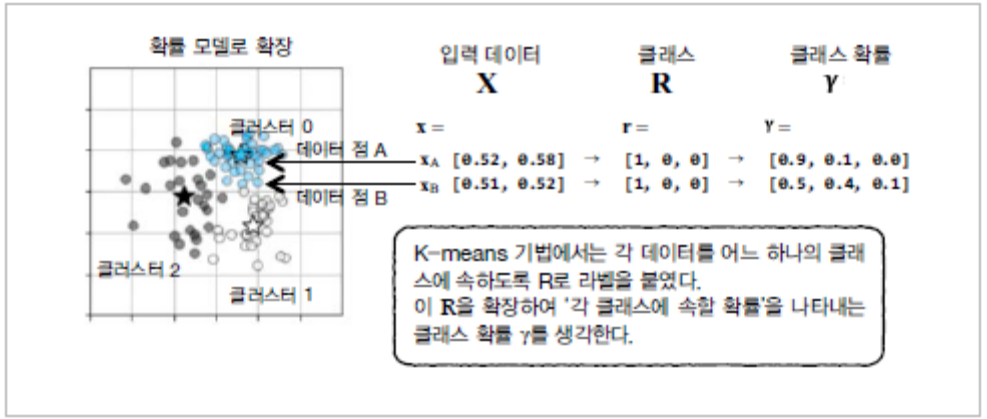
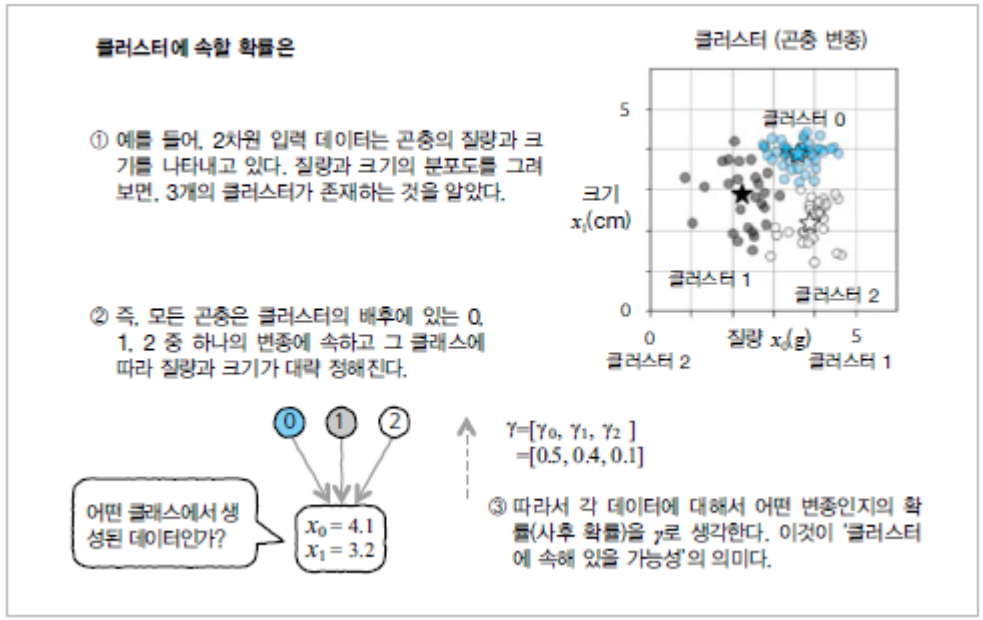


그림 9-10 클러스터에 속할 확률



SECTION.03 가우시안 혼합 모델

3.2 가우시안 혼합 모델

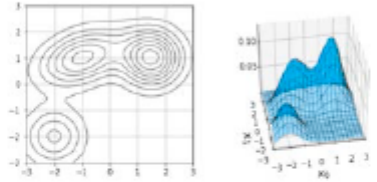
- 부담률 γ를 구하기 위해 가우시안 혼합 모델이라는 확률 모델을 소개함.
- 가우시안 혼합 모델은 2차원 가우스 함수 여러 개를 합친 것임.

그림 9-11 가우시안 혼합 모델

가우시안 혼합 모델

$$p(x) = \sum_{k=0}^{K-1} \pi_k N(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

K개의 가우스 분포를 모두 합해 다양한 분포를 표현할 수 있다.



리스트 9-2-(1, 2, 4, 6, 7)

분포의 모양을 결정하는 매개 변수(입력 x가 2차원인 경우)

중심 벡터: 각 가우스 분포 k의 중심

$$\boldsymbol{\mu}_k = (\mu_{k0}, \mu_{k1})$$

공분산 행렬: 각 가우스 분포 k의 퍼짐

$$\boldsymbol{\Sigma}_k = \begin{bmatrix} \sigma_{k00} & \sigma_{k01} \\ \sigma_{k10} & \sigma_{k11} \end{bmatrix} \qquad \sigma_{k00} > 0, \sigma_{k11} > 0, \sigma_{k01} = \sigma_{k10}$$

혼합 계수: 각각의 가우스 분포의 크기의 비율

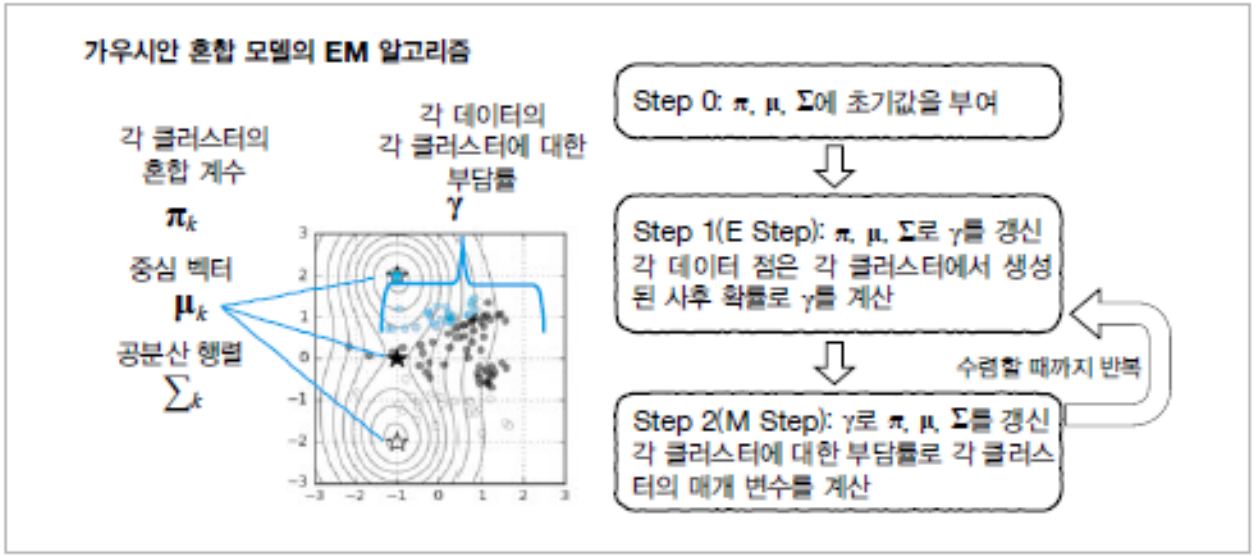
$$\pi_k \qquad 0 \le \pi_k \le 1, \sum_{k=0}^{K-1} \pi_k = 1$$

SECTION.03 가우시안 혼합 모델

3.3 EM 알고리즘의 개요

- EM(expectation-maximization algorithm) 즉, 알고리즘 기댓값 최대화 알고리즘은 관측되지 않는 잠재변수에 의존하는 확률 모델에서 최대가능도(maximum likelihood)나 최대사후확률(maximum a posteriori; MAP)을 갖는 모수의 추정값을 찾는 반복적인 알고리즘 임.
- EM 알고리즘을 사용하여 가우시안 혼합 모델을 데이터에 피팅해보고, 부담률 γ를 구하는 방법을 설명함.

그림 9-12 가우시안 혼합 모델의 EM 알고리즘: 개요



SECTION.03 가우시안 혼합 모델

3.4 Step 0: 변수의 준비 및 초기화

- 변수의 초기화와 매개 변수를 그림으로 그리는 부분을 프로그램으로 구현 함.

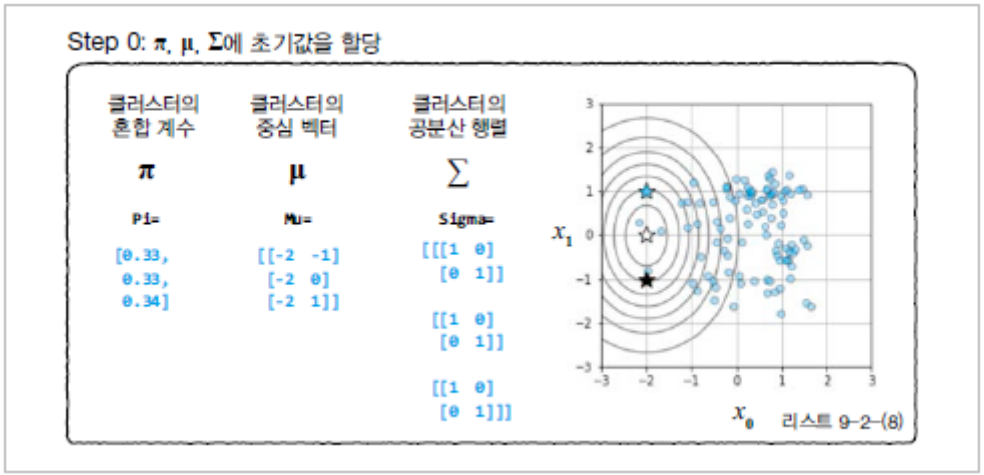
In

```
# ------------------ 리스트 9-2-(8)
# 초기 설정 ------------------------------------
N = X.shape[0]
K = 3
Pi = np.array([0.33, 0.33, 0.34])
Mu = np.array([[-2, 1], [-2, 0], [-2, -1]])
Sigma = np.array([[[1, 0], [0, 1]], [[1, 0], [0, 1]], [[1, 0], [0, 1]]])
Gamma = np.c_[np.ones((N, 1)), np.zeros((N, 2))]


X_col=np.array([[0.4, 0.6, 0.95], [1, 1, 1], [0, 0, 0]])


# 데이터를 그리기 ------------------------------
def show_mixgauss_prm(x, gamma, pi, mu, sigma):
    N, D = x.shape
    show_contour_mixgauss(pi, mu, sigma)
    for n in range(N):
        col=gamma[n,0]*X_col[0]+gamma[n,1]*X_col[1]+gamma[n,2]*X_col[2]
        plt.plot(x[n, 0], x[n, 1], 'o',
                 color=tuple(col), markeredgecolor='black',
                 markersize=6, alpha=0.5)
    for k in range(K):
        plt.plot(mu[k, 0], mu[k, 1], marker='*',
                 markerfacecolor=tuple(X_col[k]), markersize=15,
                 markeredgecolor='k', markeredgewidth=1)


    plt.grid(True)


plt.figure(1, figsize=(4, 4))
show_mixgauss_prm(X, Gamma, Pi, Mu, Sigma)
plt.show()
```

Out

```
# 실행 결과는 [그림 9-13]을 참조
```

그림 9-13 가우시안 혼합 모델의 EM 알고리즘: 초기화



SECTION.03 가우시안 혼합 모델

3.5 Step 1(E Step): γ 갱신

- 가우시안 혼합 모델의 EM 알고리즘: Step 1(E Step)

In

```
# ------------------ 리스트 9-2-(9)
# gamma를 갱신(E Step) --------------------
def e_step_mixgauss(x, pi, mu, sigma):
    N, D = x.shape
    K = len(pi)
    y = np.zeros((N, K))
    for k in range(K):
        y[:, k] = gauss(x, mu[k, :], sigma[k, :, :]) # KxN
    gamma = np.zeros((N, K))
    for n in range(N):
        wk = np.zeros(K)
        for k in range(K):
            wk[k] = pi[k] * y[n, k]
        gamma[n, :] = wk / np.sum(wk)
    return gamma


# 메인 ----------------------------------
Gamma = e_step_mixgauss(X, Pi, Mu, Sigma)
```

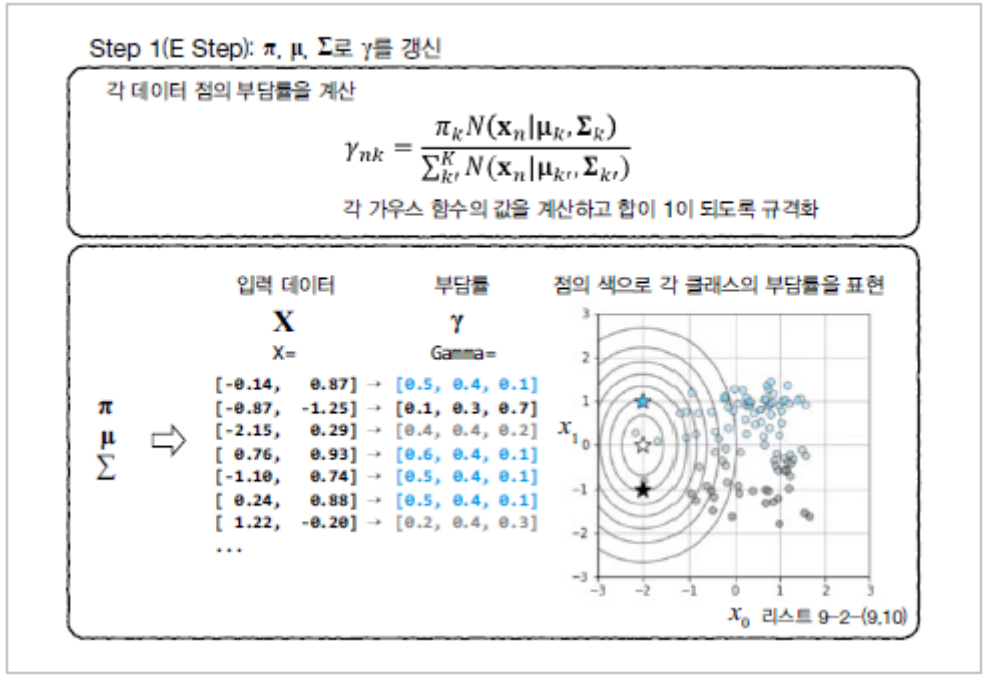
In

```
# ---------------- 리스트 9-2-(10)
# 표시 ----------------------------------

plt.figure(1, figsize=(4, 4))
show_mixgauss_prm(X, Gamma, Pi, Mu, Sigma)
plt.show()
```

Out

```
# 실행 결과는 [그림 9-14]를 참조
```

그림 9-14 가우시안 혼합 모델의 EM 알고리즘: Step 1(E Step)



SECTION.03 가우시안 혼합 모델

3.6 Step 2(M Step): π, μ, Σ의 갱신

- 가우시안 혼합 모델의 EM 알고리즘: Step 2(M Step)

In

```
# ----------------- 리스트 9-2-(11)
# Pi, Mu, Sigma 갱신 (M step) ------------
def m_step_mixgauss(x, gamma):
    N, D = x.shape
    N, K = gamma.shape
    # pi를 계산
    pi = np.sum(gamma, axis=0) / N
    # mu를 계산
    mu = np.zeros((K, D))
    for k in range(K):
        for d in range(D):
            mu[k, d] = np.dot(gamma[:, k], x[:, d]) / np.sum(gamma[:, k])
    # sigma를 계산
    sigma = np.zeros((K, D, D))
    for k in range(K):
        for n in range(N):
            wk = x - mu[k, :]
            wk = wk[n, :, np.newaxis]
            sigma[k, :, :] = sigma[k, :, :] + gamma[n, k] * np.dot(wk, wk.T)
        sigma[k, :, :] = sigma[k, :, :] / np.sum(gamma[:, k])
    return pi, mu, sigma


# 메인 ----------------------------------
Pi, Mu, Sigma = m_step_mixgauss(X, Gamma)
```

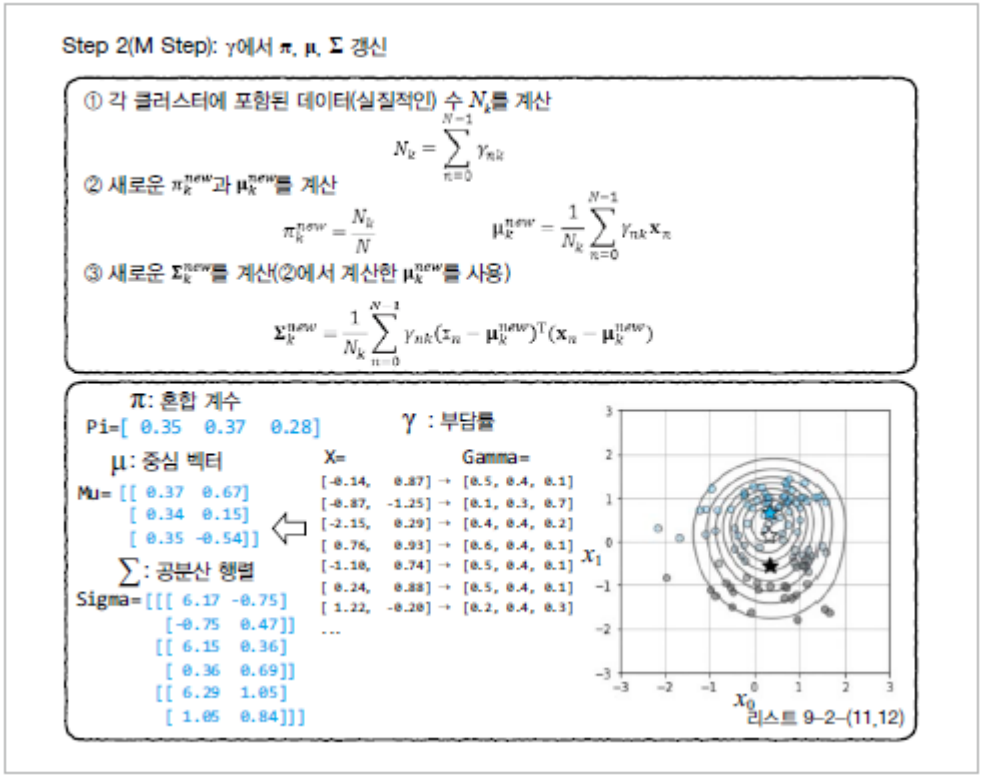
In

```
# ----------------- 리스트 9-2-(12)
# 표시 ----------------------------------
plt.figure(1, figsize=(4, 4))
show_mixgauss_prm(X, Gamma, Pi, Mu, Sigma)
plt.show()
```

Out

```
# 실행 결과는 [그림 9-15]를 참조
```

그림 9-15 가우시안 혼합 모델의 EM 알고리즘: Step 2(M Step)



SECTION.03 가우시안 혼합 모델

- 가우시안 혼합 모델의 EM 알고리즘의 수렴 과정

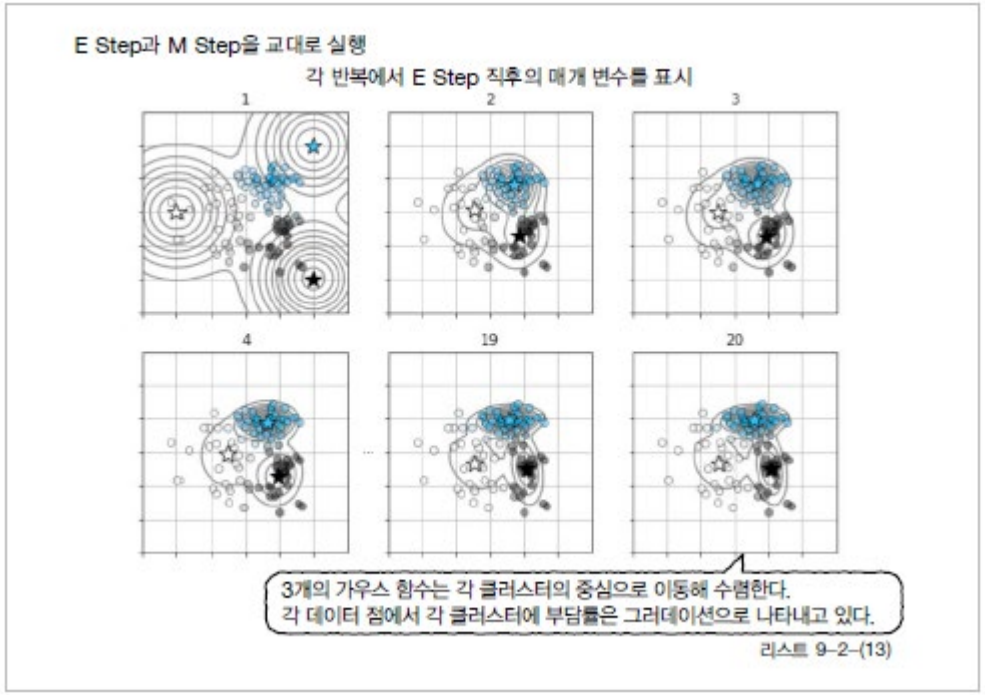
In

```
# ------------------ 리스트 9-2-(13)
Pi = np.array([0.3, 0.3, 0.4])
Mu = np.array([[2, 2], [-2, 0], [2, -2]])
Sigma = np.array([[[1, 0], [0, 1]], [[1, 0], [0, 1]], [[1, 0], [0, 1]]])
Gamma = np.c_[np.ones((N, 1)), np.zeros((N, 2))]
plt.figure(1, figsize=(10, 6.5))
max_it = 20 # 반복 횟수


i_subplot=1;
for it in range(0, max_it):
    Gamma = e_step_mixgauss(X, Pi, Mu, Sigma)
    if it<4 or it>17:
        plt.subplot(2, 3, i_subplot)
        show_mixgauss_prm(X, Gamma, Pi, Mu, Sigma)
        plt.title("{0:d}".format(it + 1))
        plt.xticks(range(X_range0[0], X_range0[1]), "")
        plt.yticks(range(X_range1[0], X_range1[1]), "")
        i_subplot=i_subplot+1
    Pi, Mu, Sigma = m_step_mixgauss(X, Gamma)
plt.show()
```

Out

```
# 실행 결과는 [그림 9-16]을 참조
```

그림 9-16 가우시안 혼합 모델의 EM 알고리즘의 수렴 과정



리스트 9-2-(13)

SECTION.03 가우시안 혼합 모델

3.7 가능도

- 가우시안 혼합 모델은 데이터의 분포 p (x)를 나타내는 모델임.
- 가능도는 모든 데이터 점 x가 모델에서 생성된 확률이므로 [식 9-21]에 할당됨.

$$p(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \prod_{n=0}^{N-1} \sum_{k=0}^{K-1} \pi_k N(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \qquad (식 9\text{–}21)$$

- 이 로그를 취한 로그 가능도는 [식 9-22]임.

$$\log p(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=0}^{N-1} \left\{ \log \sum_{k=0}^{K-1} \pi_k N(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\} \qquad (식 9\text{–}22)$$

- 가능도나 로그 가능도를 최적화시킬 때는 극대화하기 때문에 -1을 곱한 음의 로그 가능도를 오차 함수 E(π, μ, Σ)로 정의함.(식 9-23).

$$E(\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = -\log p(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = -\sum_{n=0}^{N-1} \left\{ \log \sum_{k=0}^{K-1} \pi_k N(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\} \qquad (식 9\text{–}23)$$

SECTION.03 가우시안 혼합 모델

- 오차 함수를 정의하고, 오차 함수의 변화를 그래프로 그림.

In

```
# ---------------- 리스트 9-2-(14)
# 혼합 가우스의 목적 함수 ---------------------
def nlh_mixgauss(x, pi, mu, sigma):
    # x: NxD
    # pi: Kx1
    # mu: KxD
    # sigma: KxDxD
    # output lh: NxK
    N, D = x.shape
    K = len(pi)
    y = np.zeros((N, K))
    for k in range(K):
        y[:, k] = gauss(x, mu[k, :], sigma[k, :, :]) # KxN
    lh = 0
    for n in range(N):
        wk = 0
        for k in range(K):
            wk = wk + pi[k] * y[n, k]
        lh = lh + np.log(wk)
    return -lh
```

In

```
# ---------------- 리스트 9-2-(15)
Pi = np.array([0.3, 0.3, 0.4])
Mu = np.array([[2, 2], [-2, 0], [2, -2]])
Sigma = np.array([[[1, 0], [0, 1]], [[1, 0], [0, 1]], [[1, 0], [0, 1]]])
Gamma = np.c_[np.ones((N, 1)), np.zeros((N, 2))]


max_it = 20
it = 0
Err = np.zeros(max_it) # distortion measure
for it in range(0, max_it):
    Gamma = e_step_mixgauss(X, Pi, Mu, Sigma)
    Err[it] = nlh_mixgauss(X,Pi,Mu,Sigma)
    Pi, Mu, Sigma = m_step_mixgauss(X, Gamma)
    print(np.round(Err, 2))
    plt.figure(2, figsize=(4, 4))
    plt.plot(np.arange(max_it) + 1,
    Err, color='k', linestyle='-', marker='o')
    #plt.ylim([40, 80])
    plt.grid(True)
    plt.show()
```

Out

```
# 실행 결과는 [그림 9-17] 바로 참조
```

SECTION.03 가우시안 혼합 모델

- 가우시안 혼합 모델의 EM 알고리즘: 음의 로그 가능도 실행 결과

그림 9-17 가우시안 혼합 모델의 EM 알고리즘: 음의 로그 가능도