Mixture of Gaussian 의 EM Algorithm 적용

바이오 인공지능융합학과 2021192861 김가연

1. Introduction

1.1 Mixture of Gaussian

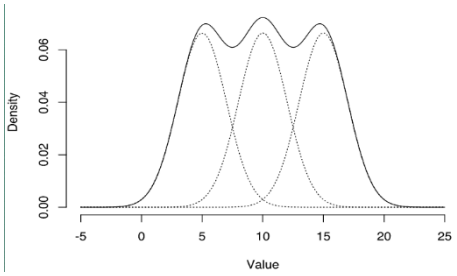


Figure 1 Gaussian Mixture Model

Gaussian Mixture Model, GMM 은 Gaussian 분포가 여러 개 혼합된 clustering 알고리즘이다.

왼쪽 그림처럼 현실에 존재하는 복잡한 형태의 확률 분포를 여러 개의 Gaussian 분포를 혼합하여 표현하자는 것이 GMM 의 기본 아이디어다.

$$p(\mathbf{x}|\mathbf{\Theta}) = \sum_{j=1}^{m} \alpha_j p_j(\mathbf{x}|\boldsymbol{\theta}_j)$$

Figure 2 GMM의 x가 발생할 확률

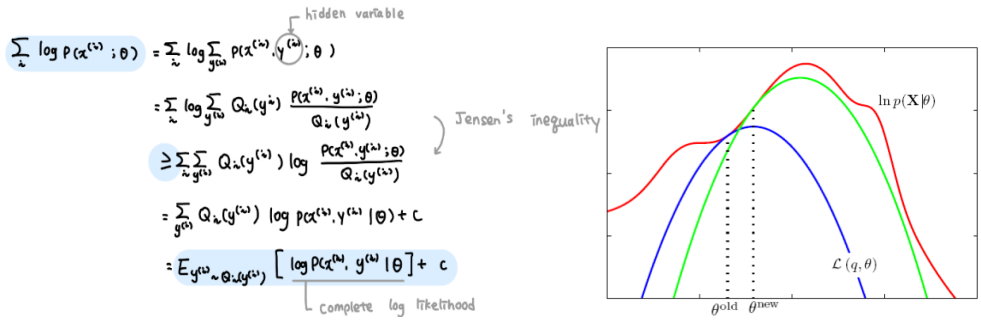
GMM 의 x 가 발생할 확률은 여러 Gaussian probability density function 의 합으로 표현된다. 위 식에서는 총 m 개의 components 가 있다. 여기서 α_j는 mixing probability 로 j 번째 component 가 선택될 확률을 나타내고 α_j는 0~1 사이의 값을 가지고 모든 α_j의 합은 1 이 된다. $p_j(x|\theta)$는 component density 로 j 번째 sample 의 likelihood 이다.

1.2 EM Algorithm (Expectation-Maximization Algorithm)

1.2.1 EM 알고리즘의 개념

EM 알고리즘은 관측되지 않은 잠재변수(latent variables)에 의존하는 확률 모델에서 maximum likelihood 를 갖는 최적의 매개변수 θ를 찾는 반복적인 알고리즘이다. EM 알고리즘은 2 가지 step, E-step, M-step 을 반복하면서 이루어지는데 먼저 E-step 은 주어진 임의의 파라미터 초기값에서 likelihood 와 최대한 근사한 likelihood 값을 계산한다. 그리고 M-step 은 E-step 에서 계산된 likelihood 를 최대화하는 새로운 파라미터 θ를 계산한다.

1.2.2 EM 알고리즘 프로세스

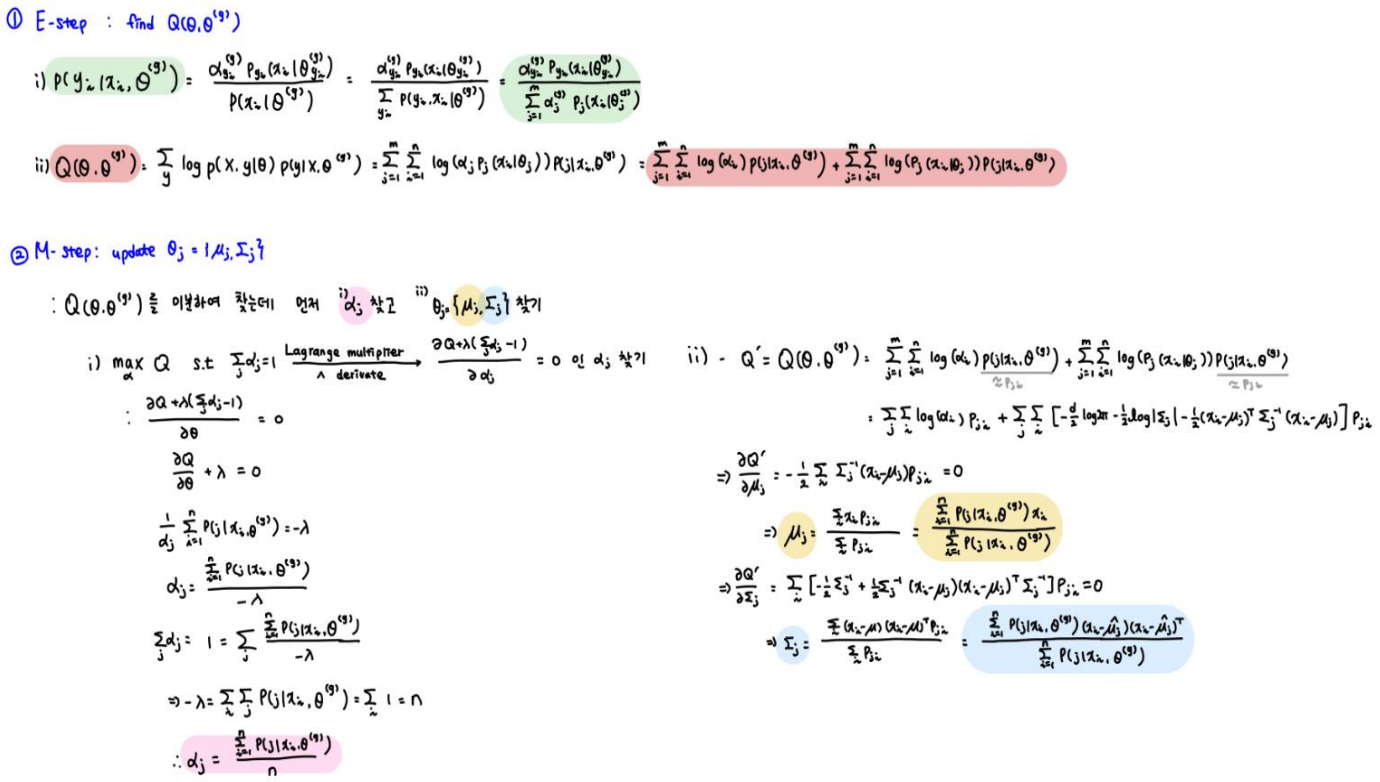


왼쪽 위의 그림처럼 우리가 maximization 하고싶은 likelihood $\sum_i \log p(x^{(i)}|\theta)$는 'Jensen's inequality'에 의해 complete log likelihood $E_{y^{(i)}\sim Q_i(y^{(i)})}[\log p(x^{(i)}, y^{(i)}|\theta]$ 이상으로 정의된다. 따라서 complete log likelihood 를 높이게 되면 이를 lower bound 로 가지는 maximization 하고싶은 likelihood 를 높일 수 있을 것이다. 즉 오른쪽 위 그림에서 볼 수 있듯이 EM 알고리즘은 실제 cost function $\sum_i \log p(x^{(i)}|\theta)$의 tight lower bound 인 $E_{y^{(i)}\sim Q_i(y^{(i)})}[\log p(x^{(i)}, y^{(i)}|\theta]$를 최대화하는 것이다.

위에서는 두 식의 관계가 이상으로 정의되었는데 만약 $Q_i(y^{(i)}) := \log p(y^{(i)}|x^{(i)}, \theta]$ 이라면 두 식은 같아지게 된다. 즉 EM 알고리즘의 E-step 은 $Q_i(y^{(i)}) := \log p(y^{(i)}|x^{(i)}, \theta]$인 $Q_i(y^{(i)})$를 찾고,

M-step 에서 $\theta := argmax_\theta \sum_i \sum_{y^{(i)}} Q_i(y^{(i)}) \log \frac{p(x^{(i)}, y^{(i)}; \theta)}{Q_i(y^{(i)})}$로 θ를 업데이트 하는 것이다.

1.3 MoG(Mixture of Gaussian)의 EM Algorithm 적용



2. Method

2.1 Data

2.1.1 평균과 분산

```
# 차원 수
Ndim = 2
# class 수
NClasses = 3
# nsample : 전체 데이터 갯수
Nsample = 150
# class 별 뽑히는 확률 같게 하려면 '1'로 설정
same_probability = 1
# Mean
Mu = [np.random.uniform(0.0, 50.0, Ndim) for i in range(1,NClasses+1)]

# Covariance
SDClass = np.random.uniform(1.0,10.0,NClasses)
#print('SDClass : ',SDClass)
Cov = [makeSDmatrix(Ndim,i) for i in SDClass]
```

```
Means of the classes
['Means for Class 1'] [28.78109855 12.60408777]
['Means for Class 2'] [15.8768149 37.8477567]
['Means for Class 3'] [16.31058868 35.0237212 ]

        Variances of the classes
        ['the variance for Class 1']
        [[1.98716156 0.39564416]
         [0.39564416 2.13130009]]
        ['the variance for Class 2']
        [[6.51461282 0.37277097]
         [0.37277097 6.49495126]]
        ['the variance for Class 3']
        [[2.60120821 0.80054451]
         [0.80054451 2.78075097]]
```

```
def makeSDmatrix(n,sd):
    M = np.matrix(np.random.rand(n,n))
    M = 0.5*(M + M.T)
    M = M + sd*np.eye(n)
    return M
```

2 차원 Gaussian Mixture Model 의 data 를 뽑기 위해 class 수 (NClasses)와 총 데이터 sample 수(Nsample)을 설정한다. 각 component 에서 sample 이 뽑히는 확률을 같게 하려면 same_probability 를 '1'로 설정한다.

3 개의 components 를 사용한다고 하면 평균 Mu 는 0 에서 50 사이에서 random 하게 3 개의 class 마다 평균 2 개, 총 6 개의 값을 뽑는다. 분산은 2 차원이므로 공분산의 형태일 것이고 마찬가지로 1 에서 10 사이에서 하나의 값을 random 하게 뽑고 그 값을 makeSDmatrix 함수에 넣어서 symmetric positive definite 행렬로 변환해 covariance 형태의 분산을 만든다.

2.1.2 각 component 에서 sample 을 뽑을 확률 α_i과 sample 개수 뽑기

```
# 모든 클래스가 선택될 확률을 동일하게
if same_probability==1:
    alpha = np.repeat(1.0/NClasses,NClasses)
# 클래스 별 선택될 확률을 다르게
else:
    a = np.ones(NClasses)
    n = 1
    p = len(a)
    rd = np.random.gamma(np.repeat(a,n),n,p)
    rd = np.divide(rd,np.repeat(np.sum(rd),p))
    alpha = rd
```

Figure 3 component별 선택될 확률

```
The probabilities of each classes from 1 to 3
[0.17322058 0.20811174 0.61866768]
```

Figure 4 확률이 다를 경우 예시

```
The probabilities of each classes from 1 to 3
[0.33333333 0.33333333 0.33333333]
```

Figure 3 확률이 같을 경우 예시

모든 component 에서 다른 확률로 sample 을 뽑는다면 gamma 분포를 통해 합이 1 이도록 확률을 class 개수만큼 분배하고, 같은 확률로 sample 을 뽑는다면 1 을 각 class 개수로 나누어 사용한다.

```
r = np.random.multinomial(Nsample,alpha)
```

Figure 4 확률에 따라 component별로 sample 개수선택

random.multinomial을 통해 alpha의 확률을 사용해 총 Nsample개의 데이터를 클래스 별로 나누어 준다.

```
data_sample = [np.random.multivariate_normal(Mu[i], Cov[i], r[i]) for i in range(0,NClasses)]
```

Figure 5 multivariate 정규 분포를 이용하여 각 클래스별로 sample을 random으로 뽑기

random.multivariate_normal 을 이용해 위에서 선택된 평균, 분산, 데이터 개수에 따라 2 차원 데이터를 class 수 만큼 뽑는다. Class 가 3 개이고 각 class 별로 44, 50, 56 개의 데이터가 있다면 [28.11044942, 9.37201796], 이런 식의 data 가 각 클래스별로 44, 50, 56 개 존재한다.

2.1.3 EM algorithm test 를 위해 데이터 섞기

```
# data를 array로
y = np.empty([1, Ndim])
for i in range(NClasses):
    y = np.vstack((y,data_sample[i]))
y = np.delete(y, 0, axis=0)

# 데이터들에 클래스 할당하기
v_true = np.zeros((1))
for i,j in enumerate(r):
    v_true = np.hstack((v_true, np.repeat(i+1, j)))

v_true = np.array([v_true[1:]])
y_true = np.concatenate((y, v_true.T), axis=1)

# 데이터 섞기
np.random.shuffle(y_true)
```

위에서 뽑은 data_sample 을 (150,2) 형태의 array 로 만든다.

다음으로 정확도 계산을 위해 data_sample array 에 열을 하나 추가하여 class 번호도 붙여준다. 예를 들어 [16.82792156 26.00812523 1.]는 1 번 class 에 속하는 (x,y) 2 차원 데이터를 의미한다.

마지막으로 EM algorithm 실행 전 데이터를 random 한 순서로 섞어서 데이터를 저장해 준다.

2.2 EM Algorithm

2.2.1 초기 평균, 분산, 확률 설정하기

```
initMu = np.empty([NClasses, Ndim])
initCov = np.empty([Ndim, Ndim, NClasses])

for j in range(NClasses):

    initMu[j,:] = np.random.uniform(0.0, 50.0, Ndim)
    sd = np.random.uniform(1.0,10.0)
    initCov[:,:,j] = makeSDmatrix(Ndim,sd)

a = np.ones(NClasses)
n = 1
p = len(a)
rd = np.random.gamma(np.repeat(a,n),n,p)
rd = np.divide(rd,np.repeat(np.sum(rd),p))
initW = rd
```

초기 평균(initMu), 분산(initCov), 각 component 에서 sample 이 뽑힐 확률(initW)을 위해 앞에서 data 를 뽑을 때와 동일하게 평균과 분산은 크기만큼 random 하게 만들어주고 확률은 Gamma 분포를 통해 합이 1 이 되도록 확률 값을 나누어 준다.

2.2.2 E-step

```
def EStep(y, w, mu, cov):

    r_ij = np.zeros((y.shape[0], mu.shape[0]))

    for Object in range(y.shape[0]):
        r_ij_Sumj = np.zeros(mu.shape[0])

        for jClass in range(mu.shape[0]):
            r_ij_Sumj[jClass] = w[jClass]
                    * sc.stats.multivariate_normal.pdf(y[Object,:], mu[jClass,:], cov[:,:,jClass])

        for jClass in range(r_ij_Sumj.shape[0]):
            r_ij[Object,jClass] =   r_ij_Sumj[jClass] / np.sum(r_ij_Sumj)

    return r_ij
```

$\boldsymbol{p(y_i|x_i,\theta^{(g)})} = \frac{\alpha_{y_i}^{(g)} p_{y_i}(x_i|\theta_{y_i}^{(g)})}{\sum_{j=1}^{m}\alpha_j^{(g)} p_j(x_i|\theta_j^{(g)})}$계산을 위해서 먼저 $\alpha_{y_i}^{(g)} p_{y_i}(x_i|\theta_{y_i}^{(g)})$인 r_ij_Sumj[jClass]를 보면 $\alpha_{y_i}^{(g)}$가 w[jClass]이고 현재 우리는 multivariate_normal 을 통해 sampling 을 하였으므로 $p_{y_i}(x_i|\theta_{y_i}^{(g)})$는 multivariate_normal.pdf 를 따를 것이다. 따라서 두 식의 곱인 $\alpha_{y_i}^{(g)} p_{y_i}(x_i|\theta_{y_i}^{(g)})$가 분포가 되고 이것을 분자로, class 개수만큼 j 에 대해 모두 더한 np.sum(r_ij_Sumj)를 분모에 주면 우리가 구할 $p(y_i|x_i,\theta^{(g)})$가 정의된다.

2.2.3 M-step

```
def MStep(r, y, mu, cov):

    N = y.shape[0]

    Allmu_j = np.zeros((N, mu.shape[0], mu.shape[1]))
    Allcov_j = np.zeros((N, cov.shape[0], cov.shape[1], cov.shape[2]))
    mu_j = np.zeros((mu.shape[0], mu.shape[1]))
    cov_j = np.zeros((cov.shape[0], cov.shape[1], cov.shape[2]))

    # weight 업데이트
    w_j = np.sum(r, axis=0) / N

    # mean : mu_j
    for Object in range(N):
        Allmu_j[Object,:,:]=np.asmatrix(r[Object,:]).T*y[Object,:]

    for j in range(cov.shape[2]):
        mu_j[j,:] = (np.sum(Allmu_j, axis=0)[j,:])/(np.sum(r, axis=0)[j])

    # covariance : cov_j
    for Object in range(N):
        for j in range(cov.shape[2]):
            Allcov_j[Object,:,:,j] = r[Object,j]
            *np.asmatrix(y[Object,:]-mu_j[j,:]).T*np.asmatrix(y[Object,:]-mu_j[j,:])

    for j in range(cov.shape[2]):
        cov_j[:,:,j] = (np.sum(Allcov_j, axis=0)[:,:,j])/(np.sum(r,axis=0)[j])

    return w_j,mu_j,cov_j
```

1.3 에서 보았듯이 $\boldsymbol{\alpha_j} = \frac{\sum_{i=1}^{n} p(j|x_i,\theta^{(g)})}{n}$, $\boldsymbol{\mu_j} = \frac{\sum_{i=1}^{n} p(j|x_i,\theta^{(g)}) x_i}{\sum_{i=1}^{n} p(j|x_i,\theta^{(g)})}$, $\boldsymbol{\Sigma_j} = \frac{\sum_{i=1}^{n} p(j|x_i,\theta^{(g)})(x_i-\widehat{\mu_j})(x_i-\widehat{\mu_j})^T}{\sum_{i=1}^{n} p(j|x_i,\theta^{(g)})}$[1] **로 업데이트**한다.

[1] MoG의 EM 알고리즘에서 사용할 수 있는 Σ_j는 총 5가지 ($\sigma^2 I$, $\sigma_j^2 I$, $diag(\{\sigma_{jk}^2\}_{k=1}^{d})$, Σ, arbitrary)가 있는데 본 실험에서는 가장 general한 방법인 'arbitrary' 를 사용하였다.

2.2.4 E-step, M-step 반복적으로 적용

```
EMiteration = 2000

# starting values
r_n = EStep(y, initW, initMu, initCov)
w_n,mu_n,cov_n = MStep(r_n, y, initMu, initCov)

logLH = -10000000000000

for i in range(EMiteration):
    # E step
    r_n = EStep(y, w_n, mu_n,cov_n)

    # M step
    w_n,mu_n,sigma_n = MStep(r_n, y, mu_n, cov_n)

    # log likelihood
    logLall = np.zeros((y.shape[0]))

    for Object in range(y.shape[0]):
        LH = np.zeros(NClasses)

        for jClass in range(NClasses):
            LH[jClass] = w_n[jClass]
            * sc.stats.multivariate_normal.pdf(y[Object,:], mu_n[jClass,:], cov_n[:,:,jClass])

        logLall[Object] = np.log(np.sum(LH))

    logL = np.sum(logLall)

    if logL > logLH:
        # 수렴조건을 둘 차이가 10^(-4) 미만일때로 설정
        if (logL-logLH<10**(-6)):
            print('둘 차이가 매우 적음')
            break
        else :
            logLH = logL
            print ('found larger: ', logLH)
            w_p = w_n
            mu_p = mu_n
            sigma_p = sigma_n
            r_p = r_n
print('final log likelihood : ', logLH)
```

EM Algorithm 은 E-step 과 M-step 을 반복적으로 실행하는데 stop condition 이 필요하다. 총 2 가지 ①현재 log likelihood 값과 직전의 log likelihood 값의 차이가 설정된 ε[2]보다 작을 때, 혹은 ② iteration 의 수를 정해주어 stop condition 을 정의한다. 본 실험에서는 ① 현재 log likelihood 값과 직전의 log likelihood 값의 차이가 설정된 ε보다 작을 때에 EM 알고리즘이 종료된다. 두 log likelihood 값의 차이가 ε보다 작지 않지만 log likelihood 값이 더 큰 것이 발견되었다면 log likelihood, α_j, μ_j,Σ_j, 결과 r_p 를 모두 현재 값으로 업데이트 해준다.

[2] 본 실험에서 ε은 10^{-6}으로 설정하였다.

2.2.5 정확도 계산

EM 알고리즘을 통해 clustering 을 하게 되면 원래의 class 번호에 따라 분류되는 것이 아니기 때문에 r_p 의 결과 (150*NClasses)에서 NClasses 인 열을 순열으로 모든 경우의 수를 따져 보아야 한다. 예를 들어 5 개의 class 를 사용한다면 결과 r_p 는 150*5 의 형태이고 이때 열 5 개를 모든 순열 5*4*3*2*1=120 가지의 경우의 수를 모두 따져 보아야한다.

```
a=list(range(NClasses))
permute = list(permutations(a,NClasses))
permute_=[]
acc_list=[]
permute_list=[]
r_list=[]
# 모든 permutation을 고려해 정확도 구하여서 가장 높은 정확도를 가지는 permutation 찾기
for i in range(len(permute)):

    Gorder = list(permute[i])
    permute_.append(Gorder)

    r_=[]
    reorder = r_p[:,Gorder]
    inf = np.argmax(reorder, axis=1)
    r_.append(inf+1)

    Clust=y_true[:,2]==inf+1
    now_acc=np.mean(Clust)
    acc_list=np.append(acc_list, now_acc)
    permute_list.append(list)
    r_list.append(r_)
acc=max(acc_list)
index=np.argmax(acc_list)
```

먼저 class 의 개수만큼 0 에서 NClasses-1 까지의 list 를 만들고 이 숫자들로 가능한 모든 순열을 만든다. 총 순열의 수만큼 반복하며 해당 permutation 을 이용해 r_p 를 재정렬(reorder)하고 np.argmax 를 통해 가장 높은 확률을 가지는 것을 찾는다(inf). 후에 inf 에 1 을 더하고 원래의 class 가 같은지 비교하여(Clust) np.mean 을 통해 accuracy 를 찾아낸다.

예를 들어 class 가 5 개라면 a={0,1,2,3,4}가 되고 permute={(0,1,2,3,4), (0,1,2,4,3), … }이며 permute 는 총 120 개의 원소가 있다. 이중 하나의 순열 {0,1,3,4,2} 이용해 r_p 를 재정렬하여 reoder 을 만들고 np.argmax(reorder)을 하여 재정렬한 후 확률이 가장 높은 값을 가지는 index 를 150 개 모두 찾아낸 후 값에 1 을 더해주면 이것이 우리가 예측한 class 번호 inf+1 이 된다. 마지막으로 원래의 답인 y_true[:,2]와 inf+1 를 비교하여 Clust 를 만들고 np.mean(Clust)를 통해 정확도를 계산한다.

for 문을 이용해 가능한 모든 순열의 정확도를 acc_list 에 저장해 두고 list 에서 가장 큰 값이 우리가 사용할 정확도이고 그때 사용된 순열이 가장 바르게 재정렬한 경우이다. 정확도를 acc 로 그때의 순열을 index 로 저장한다.

3. Result

비교를 위해 총 150 개의 데이터를 class 3 개, 5 개, 7 개로 나누어 실험을 진행하였다.

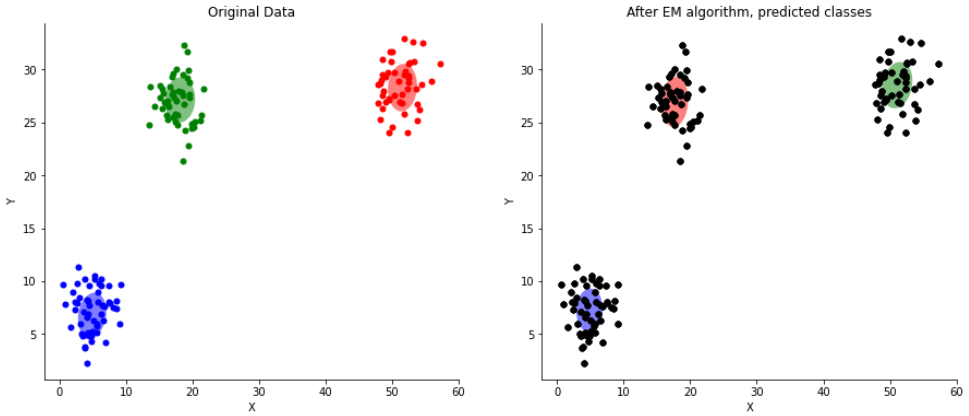
3.1 class 3 개

3.1.1 각 component 별 선택될 확률 α_i가 동일할 때 (α_i=0.333)

```
The probabilities of each classes from 1 to 3
[0.33333333 0.33333333 0.33333333]
The number of objects in each classes from 1 to 3
[54 49 47]
```

각 component 별로 선택될 확률 α_i가 동일하므로 각각의 component 들은 모두 33.33%의 확률로 데이터가 선택될 것이다. 위 확률을 바탕으로 데이터 150 개는 각각 class 에 59, 49, 47 개로 나누어 졌다.

① 원래 데이터 분포와 clustering 이후의 분포 비교



실험 데이터를 확인해 보면 왼쪽 위의 그림과 같이 분포하는 것을 확인할 수 있다. 다음으로 EM 알고리즘을 반복적으로 실행하여 clustering 한 결과 오른쪽 위의 그림과 같이 데이터가 잘 clustering 된 것을 확인할 수 있다.

② log likelihood값과 정확도 비교

```
found larger:  -848.1615133461664
found larger:  -848.1396566579525
둘 차이가 매우 적음
final log likelihood :  -848.1396566579525
accuracy :  1.0
가장 높은 정확도를 가지는 permutation :  (0, 2, 1)
```

실제 class

```
y_true[:,2]
```

```
array([2., 2., 3., 1., 3., 1., 2., 2., 1., 1., 1., 1., 2., 3., 2., 2., 3.,
       3., 2., 2., 2., 1., 3., 1., 3., 2., 3., 1., 2., 1., 1., 2., 1., 2.,
       3., 1., 1., 3., 2., 2., 2., 2., 3., 2., 2., 1., 3., 1., 1., 1., 1.,
       2., 1., 3., 1., 1., 1., 2., 1., 3., 2., 1., 2., 3., 3., 3., 1., 1.,
       3., 3., 1., 3., 2., 2., 1., 2., 3., 2., 1., 3., 1., 2., 1., 2., 3.,
       1., 2., 1., 3., 2., 1., 3., 2., 2., 3., 2., 3., 2., 2., 1., 1., 1.,
       3., 1., 1., 1., 3., 3., 1., 1., 3., 2., 3., 3., 3., 2., 2., 3., 2.,
       3., 2., 3., 3., 1., 3., 2., 3., 3., 2., 3., 3., 3., 1., 2., 1., 1.,
       1., 1., 2., 1., 2., 3., 3., 2., 1., 3., 1., 2., 1., 1.])
```

예측 class

```
r_list[index]
```

```
[array([2, 2, 3, 1, 3, 1, 2, 2, 1, 1, 1, 1, 2, 3, 2, 2, 3, 3, 2, 2, 2, 1,
        3, 1, 3, 2, 3, 1, 2, 1, 1, 2, 1, 2, 3, 1, 1, 3, 2, 2, 2, 2, 3, 2,
        2, 1, 3, 1, 1, 1, 1, 2, 1, 3, 1, 1, 1, 2, 1, 3, 2, 1, 2, 3, 3, 3,
        1, 1, 3, 3, 1, 3, 2, 2, 1, 2, 3, 2, 1, 3, 1, 2, 1, 2, 3, 1, 2, 1,
        3, 2, 1, 3, 2, 2, 3, 2, 3, 2, 2, 1, 1, 1, 3, 1, 1, 1, 3, 3, 1, 1,
        3, 2, 3, 3, 3, 2, 2, 3, 2, 3, 2, 3, 3, 1, 3, 2, 3, 3, 2, 3, 3, 3,
        1, 2, 1, 1, 1, 1, 2, 1, 2, 3, 3, 2, 1, 3, 1, 2, 1, 1], dtype=int64)]
```

EM 알고리즘은 이전과 현재의 log likelihood값의 차이가 10^{-6} 보다 작을 때 종료된다. 본 실험에서 2번의 iteration 후 최종 log likelihood값은 -848.1396이고 100%의 정확도를 가지며 그때 사용된 순열은 (0, 2, 1)이다. 실제 class번호와 예측된 class 번호를 비교해 보면 모든 데이터가 동일한 것을 확인할 수 있다.

③ 각 component 가 선택될 확률 α_i, 평균, 분산 비교

- 각 component 가 선택될 확률 α_i

```
예측 : class 별 선택될 확률:  [0.35873994 0.32792673 0.31333333]
실제 : class 별 선택될 확률  [0.33333333 0.33333333 0.33333333]
```

- 평균

```
['예측 : class 별 means'] [4.87752708 7.23812228]
['예측 : class 별 means'] [17.65719568 26.9451474 ]
['예측 : class 별 means'] [51.11369717 28.53257961]
------------------
['실제 : class 별 means1'] [4.91565362 6.77174859]
['실제 : class 별 means2'] [18.12571353 27.133631  ]
['실제 : class 별 means3'] [51.54425165 28.29984737]
```

- 분산

```
['예측 : class 별 variances']
[[3.98743974 0.19604601]
 [0.19604601 4.24230946]]
['예측 : class 별 variances']
[[4.15768783 0.440436  ]
 [0.440436   5.62455894]]
['예측 : class 별 variances']
[[5.1721744  0.90252717]
 [0.90252717 4.80119372]]
['실제 : class 별 variances1']
[[4.54405362 0.7385667 ]
 [0.7385667  4.52184433]]
['실제 : class 별 variances2']
[[5.12520209 0.2357074 ]
 [0.2357074  4.58307392]]
['실제 : class 별 variances3']
[[4.77386667 0.41422585]
 [0.41422585 5.06487465]]
```

마지막으로 원래 data의 분산, 평균, 각 component가 선택될 확률 α_i을 EM알고리즘이 종료된 후 우리가 구한 값들과 비교해 보면 모두 비슷한 값을 가지는 것을 확인할 수 있다.

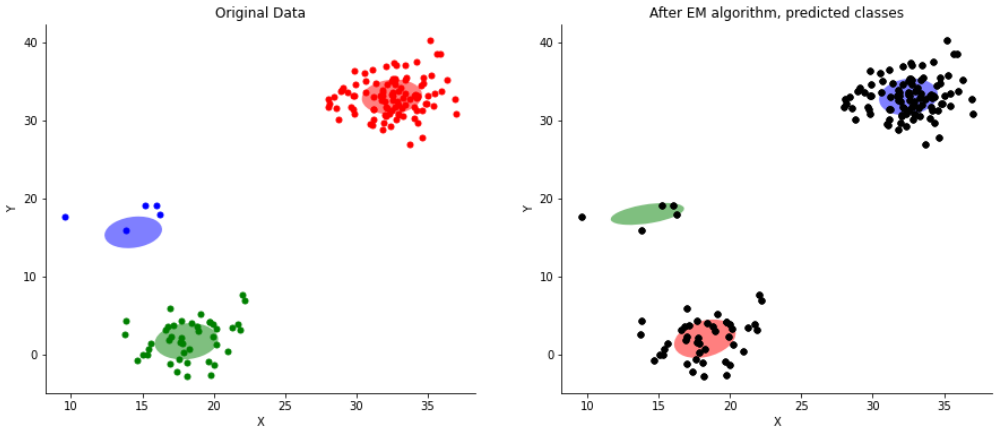
3.1.2 각 component 별 선택될 확률 α_i가 다를 때

```
The probabilities of each classes from 1 to 3
[0.04209889 0.30362989 0.65427122]
```

```
The number of objects in each classes from 1 to 3
[  5  43 102]
```

각 component별로 선택될 확률 α_i가 모두 다르게 설정된다. 본 실험에서는 각각 왼쪽 위의 사진과 같은 확률로 선택되었고 위 확률을 바탕으로 데이터는 각각 class에 5, 43, 102 개로 나누어 졌다.

① 원래 데이터 분포와 clustering 이후의 분포 비교



본 실험 데이터를 확인해 보면 왼쪽 위의 그림과 같이 분포하는 것을 확인할 수 있다. 다음으로 데이터를 random한 순서로 섞은 후 EM 알고리즘을 반복적으로 실행하여 clustering 한 결과 오른쪽 위의 그림과 같이 데이터가 잘 clustering 된 것을 확인할 수 있다.

② log likelihood값과 정확도 비교

```
found larger:  -775.5100896717679
found larger:  -775.509642378552
둘 차이가 매우 적음
final log likelihood :  -775.509642378552
```

```
accuracy :  1.0
가장 높은 정확도를 가지는 permutation :  (1, 2, 0)
```

실제 class

```
y_true[:,2]
```

```
array([2., 3., 2., 2., 3., 3., 2., 1., 1., 2., 3., 3., 3., 3., 3., 3.,
       3., 3., 1., 3., 2., 2., 3., 3., 3., 3., 3., 3., 2., 3., 2., 2.,
       3., 3., 3., 3., 2., 2., 3., 3., 3., 3., 3., 3., 3., 2., 2., 3.,
       3., 2., 3., 3., 3., 2., 2., 3., 3., 3., 3., 3., 3., 2., 3., 2.,
       2., 3., 3., 2., 2., 3., 3., 3., 3., 3., 3., 3., 2., 3., 3.,
       2., 3., 3., 3., 3., 3., 2., 2., 2., 2., 2., 3., 3., 2., 3., 3., 3.,
       3., 3., 3., 3., 2., 1., 3., 2., 3., 2., 3., 3., 3., 2., 3., 2., 2.,
       3., 3., 2., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 1., 2.,
       2., 2., 3., 3., 3., 2., 3., 3., 2., 3., 3., 2., 3., 3.])
```

예측 class

```
r_list[index]
```

```
[array([2, 3, 2, 2, 3, 3, 2, 1, 1, 2, 3, 3, 3, 3, 3, 3, 3, 3, 1, 3, 2,
        2, 3, 3, 3, 3, 3, 3, 3, 2, 3, 2, 2, 3, 3, 3, 2, 2, 3, 3, 3, 3,
        3, 3, 3, 3, 2, 2, 3, 3, 2, 3, 3, 3, 2, 2, 3, 3, 3, 3, 3, 3, 2,
        3, 2, 2, 3, 3, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 3, 3, 2, 3, 3,
        3, 3, 3, 2, 2, 2, 2, 2, 3, 3, 2, 3, 3, 3, 3, 3, 3, 3, 2, 1, 3, 2,
        3, 2, 3, 3, 3, 2, 3, 2, 2, 2, 3, 3, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3,
        3, 3, 1, 2, 2, 2, 3, 3, 3, 2, 3, 3, 2, 3, 3, 2, 3, 3], dtype=int64)]
```

본 실험에서 2번의 iteration 후 최종 log likelihood값은 -775.5096으로 매우 높은 log likelihood값을 가지고 100%의 정확도를 가지며 그때 사용된 순열은 (1, 2, 0)이다. 실제 class 번호와 예측된 class 번호를 비교해 보면 모든 데이터가 동일한 것을 확인할 수 있다.

③ 각 component 가 선택될 확률 α_i, 평균, 분산 비교

- 각 component 가 선택될 확률 α_i

```
예측 : class 별 선택될 확률:  [0.03343882 0.28666667 0.67989451]
실제 : class 별 선택될 확률  [0.04209889 0.30362989 0.65427122]
```

- 평균

```
['예측 : class 별 means'] [14.21776444 18.0006327 ]
['예측 : class 별 means'] [18.24530985  2.00989778]
['예측 : class 별 means'] [32.49941559 33.00272934]
------------------
['실제 : class 별 means1'] [14.36519237 15.64321937]
['실제 : class 별 means2'] [18.08490044  1.67440489]
['실제 : class 별 means3'] [32.48698592 32.9993909 ]
```

- 분산

```
['예측 : class 별 variances']
[[6.61399054 1.72288471]
 [1.72288471 1.97545714]]
['예측 : class 별 variances']
[[4.64601837 1.48999735]
 [1.48999735 6.06933057]]
['예측 : class 별 variances']
[[4.22418411 0.66452188]
 [0.66452188 5.53622835]]
```

```
['실제 : class 별 variances1']
[[4.05134675 0.75473282]
 [0.75473282 4.11333748]]
['실제 : class 별 variances2']
[[4.94344278 0.62156561]
 [0.62156561 5.41594375]]
['실제 : class 별 variances3']
[[4.45787018 0.47465811]
 [0.47465811 4.89681898]]
```

마지막으로 원래 data 의 분산, 평균, 각 component 가 선택될 확률 α_i을 EM 알고리즘이 종료된 후 우리가 구한 값들과 비교해 보면 평균과 α_i는 비슷한 값을 가지는 것을 확인할 수 있다. 분산은 첫번째 class 에서 분산의 차이가 조금 있지만 분류는 잘 되었음을 확인할 수 있다.

3.2 class 5 개

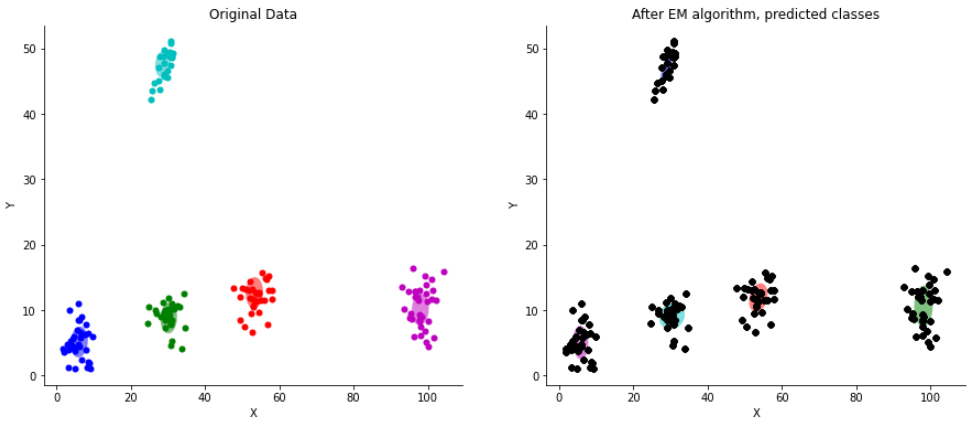
3.2.1 각 component 별 선택될 확률 α_i가 동일할 때 (α_i=0.2)

```
The probabilities of each classes from 1 to 5
[0.2 0.2 0.2 0.2 0.2]
```

```
The number of objects in each classes from 1 to 5
[33 28 32 23 34]
```

각 component별로 선택될 확률 α_i가 동일하므로 각각의 component들은 모두 20%의 확률로 데이터가 선택될 것이다. 그 확률을 바탕으로 각 component의 데이터는 33, 28, 32, 23, 34개로 나누어 졌다.

① 원래 데이터 분포와 clustering 이후의 분포 비교



실험 데이터를 확인해 보면 왼쪽 위의 그림과 같이 분포하는 것을 확인할 수 있다. 다음으로 EM 알고리즘을 반복적으로 실행하여 clustering 한 결과 오른쪽 위의 그림과 같이 데이터가 잘 clustering 된 것을 확인할 수 있다.

② log likelihood값과 정확도 비교

```
found larger:  -927.6670469066942
found larger:  -927.6110706557229
found larger:  -927.6087787517704
둘 차이가 매우 적음
final log likelihood :  -927.6087787517704
```

```
accuracy :  1.0
가장 높은 정확도를 가지는 permutation :  (4, 3, 2, 0, 1)
```

실제 **class**

```
y_true[:,2]
```

```
array([4., 5., 4., 2., 2., 1., 3., 2., 1., 1., 1., 4., 3., 4., 3., 5., 3.,
       1., 3., 5., 5., 2., 3., 3., 2., 2., 5., 5., 3., 2., 5., 2., 3., 2.,
       1., 2., 4., 3., 5., 3., 4., 3., 5., 1., 3., 4., 3., 1., 3., 1., 5.,
       3., 4., 3., 5., 2., 5., 1., 5., 1., 3., 5., 2., 5., 1., 5., 4., 3.,
       3., 3., 5., 4., 5., 2., 3., 4., 2., 4., 3., 1., 1., 5., 4., 2., 5.,
       1., 2., 1., 1., 1., 2., 3., 4., 1., 4., 2., 1., 5., 3., 1., 4., 1.,
       5., 1., 5., 5., 3., 5., 2., 5., 2., 1., 1., 2., 1., 2., 4., 2., 3.,
       1., 1., 2., 3., 5., 5., 2., 5., 5., 5., 1., 2., 3., 1., 1., 4., 4.,
       4., 5., 3., 4., 1., 1., 4., 2., 5., 4., 2., 3., 3., 5.])
```

예측 **class**

```
r_list[index]
```

```
[array([4, 5, 4, 2, 2, 1, 3, 2, 1, 1, 1, 4, 3, 4, 3, 5, 3, 1, 3, 5, 5, 2,
        3, 3, 2, 2, 5, 5, 3, 2, 5, 2, 3, 2, 1, 2, 4, 3, 5, 3, 4, 3, 5, 1,
        3, 4, 3, 1, 3, 1, 5, 3, 4, 3, 5, 2, 5, 1, 5, 1, 3, 5, 2, 5, 1, 5,
        4, 3, 3, 3, 5, 4, 5, 2, 3, 4, 2, 4, 3, 1, 1, 5, 4, 2, 5, 1, 2, 1,
        1, 1, 2, 3, 4, 1, 4, 2, 1, 5, 3, 1, 4, 1, 5, 1, 5, 5, 3, 5, 2, 5,
        2, 1, 1, 2, 1, 2, 4, 2, 3, 1, 1, 2, 3, 5, 5, 2, 5, 5, 5, 1, 2, 3,
        1, 1, 4, 4, 4, 5, 3, 4, 1, 1, 4, 2, 5, 4, 2, 3, 3, 5], dtype=int64)]
```

본 실험에서 3번의 iteration 후의 최종 log likelihood값은 -927.6088이고 100%의 정확도를 가지며 그때 사용된 순열은 (4, 3, 2, 0, 1)이다. 실제 class번호와 예측된 class 번호를 비교해 보면 모든 데이터가 동일한 것을 확인할 수 있다.

③ 각 component 가 선택될 확률 α_i, 평균, 분산 비교

- 각 component 가 선택될 확률 α_i

```
예측 : class 별 선택될 확률:  [0.21989447 0.19030463 0.2098009  0.15333333 0.22666667]
실제 : class 별 선택될 확률  [0.2 0.2 0.2 0.2 0.2]
```

- 평균

```
['예측 : class 별 means'] [5.85606065 5.0821054 ]
['예측 : class 별 means'] [30.44952703  9.11212887]
['예측 : class 별 means'] [53.68696626 11.90295717]
['예측 : class 별 means'] [29.11120401 47.29169928]
['예측 : class 별 means'] [98.09998653 10.58898757]
------------------
['실제 : class 별 means1'] [6.165113   5.14744597]
['실제 : class 별 means2'] [30.24979958  8.80619773]
['실제 : class 별 means3'] [53.29619377 12.67193248]
['실제 : class 별 means4'] [28.69002909 47.60431647]
['실제 : class 별 means5'] [97.93578657 10.10615396]
```

- 분산

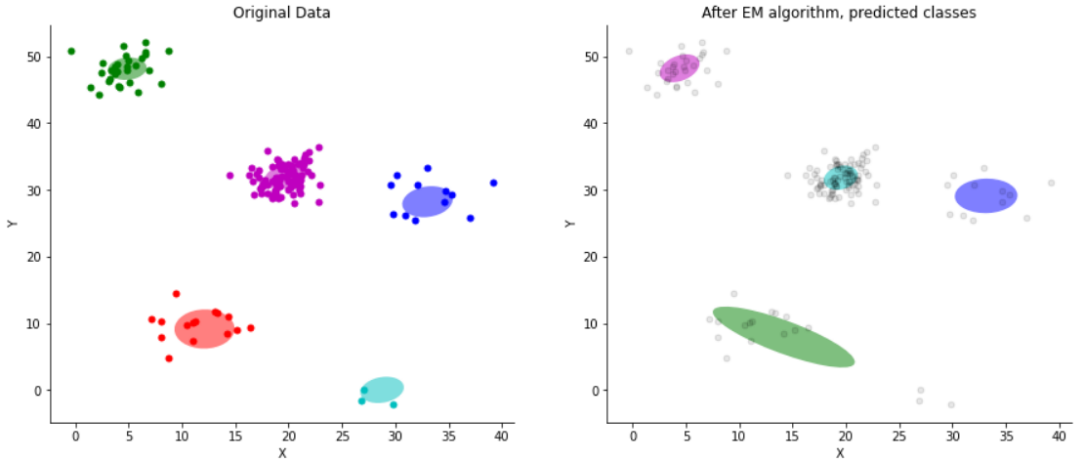
```
['예측 : class 별 variances']  ['실제 : class 별 variances1']
[[ 4.34030407 -0.1976514 ]     [[5.43925437 0.57361047]
 [-0.1976514   6.46683841]]     [0.57361047 5.54905405]]
['예측 : class 별 variances']  ['실제 : class 별 variances2']
[[11.80471322  0.87231485]     [[5.05203438 0.25234161]
 [ 0.87231485  4.18467511]]     [0.25234161 5.5386874 ]]
['예측 : class 별 variances']  ['실제 : class 별 variances3']
[[6.26930894 1.36810443]       [[5.4493659  0.28936344]
 [1.36810443 4.8643263 ]]       [0.28936344 5.72058197]]
['예측 : class 별 variances']  ['실제 : class 별 variances4']
[[2.87515156 3.04509317]       [[4.40060942 0.59382011]
 [3.04509317 5.5075485 ]]       [0.59382011 4.19451614]]
['예측 : class 별 variances']  ['실제 : class 별 variances5']
[[ 6.1729327   0.23272448]     [[5.30478743 0.36983522]
 [ 0.23272448 10.02765927]]     [0.36983522 5.88325945]]
```

마지막으로 원래 data의 분산, 평균, 각 component가 선택될 확률 α_i을 EM알고리즘이 종료된 후 우리가 구한 값들과 비교해 보면 2번 class의 분산을 제외한 나머지에서 모두 비슷한 값을 가지는 것을 확인할 수 있다.

3.2.2 각 component 별 선택될 확률 α_i가 다를 때

```
The probabilities of each classes from 1 to 5
[0.08290288 0.19018933 0.10994608 0.02240081 0.5945609 ]
The number of objects in each classes from 1 to 5
[12 29 15  3 91]
```

각 component 별로 선택될 확률 α_i가 각각 다르게 설정되고 위 확률을 바탕으로 데이터 150 개는 각각 class 에 12, 29, 15, 3, 91 개로 나누어 졌다.



① 원래 데이터 분포와 clustering 이후의 분포 비교

실험 데이터를 확인해 보면 왼쪽 위의 그림과 같이 분포하는 것을 확인할 수 있다. 다음으로 EM 알고리즘을 반복적으로 실행하여 clustering 한 결과 오른쪽 위의 그림과 같이 데이터가 잘 clustering 된 것을 확인할 수 있다.

② log likelihood값과 정확도 비교

```
found larger:  -796.6647635006454
found larger:  -795.6085005131034
found larger:  -795.5750015145373
found larger:  -795.5749228887598
둘 차이가 매우 적음                          accuracy :  0.9733333333333334
final log likelihood :  -795.5749228887598 가장 높은 정확도를 가지는 permutation :  (0, 4, 1, 2, 3)
```

실제 class

```
y_true[:,2]
```

```
array([5., 5., 2., 5., 5., 3., 3., 5., 5., 4., 2., 5., 5., 5., 5., 5., 5.,
       5., 3., 5., 5., 1., 4., 5., 2., 5., 5., 5., 5., 2., 5., 5., 1., 5., 3.,
       3., 5., 4., 5., 5., 5., 5., 3., 5., 5., 5., 5., 5., 2., 2., 2., 5.,
       5., 1., 5., 5., 5., 5., 5., 3., 5., 5., 3., 5., 5., 3., 5., 2., 5.,
       2., 1., 2., 5., 5., 1., 5., 5., 5., 2., 1., 5., 5., 2., 5., 1., 2.,
       2., 5., 2., 5., 5., 5., 2., 5., 1., 5., 5., 2., 5., 5., 2., 5., 5.,
       2., 5., 2., 5., 5., 2., 1., 1., 5., 2., 5., 3., 5., 5., 5., 1., 5.,
       5., 2., 5., 5., 5., 5., 3., 3., 5., 1., 3., 5., 5., 5., 2., 2., 5.,
       5., 5., 5., 5., 2., 3., 2., 5., 3., 2., 2., 5., 5., 5.])
```

예측 class

```
r_list[index]
```

```
[array([5, 5, 2, 5, 5, 3, 3, 5, 5, 3, 2, 5, 5, 5, 5, 5, 5, 5, 3, 5, 5, 1,
        3, 5, 2, 5, 5, 5, 2, 5, 5, 1, 5, 3, 3, 5, 3, 5, 5, 5, 5, 3, 5, 5,
        5, 5, 5, 2, 2, 2, 5, 5, 1, 5, 5, 5, 5, 5, 3, 5, 5, 3, 5, 5, 3, 5,
        2, 5, 2, 1, 2, 5, 5, 1, 5, 5, 5, 2, 1, 5, 5, 2, 5, 1, 2, 2, 5, 2,
        5, 5, 5, 2, 5, 1, 5, 5, 4, 5, 5, 2, 5, 5, 2, 5, 2, 5, 5, 2, 1, 1,
        5, 2, 5, 3, 5, 5, 5, 1, 5, 5, 2, 5, 5, 5, 5, 3, 3, 5, 1, 3, 5, 5,
        5, 2, 2, 5, 5, 5, 5, 5, 2, 3, 2, 5, 3, 2, 2, 5, 5, 5], dtype=int64)]
```

본 실험에서 총 4번의 iteration후의 최종 log likelihood값은 -795.5749이고 97.33%의 정확도를 가지며 그때 사용된 순열은 (0, 4, 1, 2, 3)이다. 실제 class번호와 예측된 class 번호를 비교해 보면 대부분의 데이터가 동일한 것을 확인할 수 있다.

③ 각 component 가 선택될 확률 α_i, 평균, 분산 비교

- 각 component 가 선택될 확률 α_i

```
예측 : class 별 선택될 확률:  [0.08011294 0.18688494 0.12       0.00644839 0.60655373]
실제 : class 별 선택될 확률  [0.08290288 0.19018933 0.10994608 0.02240081 0.5945609 ]
```

- 평균

```
['예측 : class1별 means'] [33.1493688  29.08212688]
['예측 : class2별 means'] [ 4.418582   48.20286269]
['예측 : class3별 means'] [14.18958845  7.9260322 ]
['예측 : class4별 means'] [ 8.00153802 45.87551934]
['예측 : class5별 means'] [19.52714574 31.80285831]
------------------
['실제 : class1별 means'] [33.01571138 28.20515797]
['실제 : class2별 means'] [ 4.87729764 48.13533386]
['실제 : class3별 means'] [12.11662915  9.12871791]
['실제 : class4별 means'] [2.87634913e+01 2.90526292e-03]
['실제 : class5별 means'] [19.6767409  31.94363983]
```

- 분산

```
['예측 : class1의 variances']
[[8.61791938 0.18047025]
 [0.18047025 6.69305913]]
['예측 : class2의 variances']
[[3.4869849  1.34871453]
 [1.34871453 4.37512383]]
['예측 : class3의 variances']
[[ 44.31384944 -24.91417572]
 [-24.91417572  20.79164767]]
['예측 : class4의 variances']
[[2.96133669e-08 1.66930806e-08]
 [1.66930806e-08 9.40990422e-09]]
['예측 : class5의 variances']
[[2.56043161 0.64038294]
 [0.64038294 3.30991046]]
```

```
['실제 : class1의 variances']
[[5.58918699 0.86629497]
 [0.86629497 5.41514729]]
['실제 : class2의 variances']
[[3.28360517 0.2940401 ]
 [0.2940401  2.83162044]]
['실제 : class3의 variances']
[[7.94168142 0.1476452 ]
 [0.1476452  8.61759053]]
['실제 : class4의 variances']
[[4.1433753  0.77780047]
 [0.77780047 3.87313109]]
['실제 : class5의 variances']
[[3.74359065 0.86646088]
 [0.86646088 3.46214135]]
```

마지막으로 원래 data의 분산, 평균, 각 component가 선택될 확률 α_i을 EM알고리즘이 종료된 후 우리가 구한 값들과 비교해 보면 4번 class의 α_i와 평균, 3, 4번 class의 분산을 제외한 나머지에서는 모두 비슷한 값을 가지는 것을 확인할 수 있다.

3.3 class 7 개

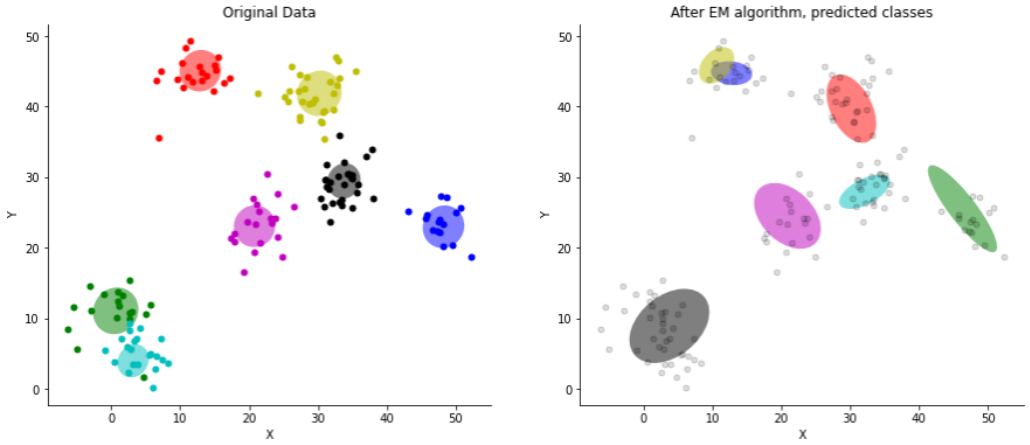
3.3.1 각 component 별 선택될 확률 α_i가 동일할 때 (α_i=0.14285714)

```
The probabilities of each classes from 1 to 7
[0.14285714 0.14285714 0.14285714 0.14285714 0.14285714 0.14285714
 0.14285714]
```

```
The number of objects in each classes from 1 to 7
[16 18 20 21 19 28 28]
```

각 component별로 선택될 확률 α_i가 동일하므로 각각의 component들은 모두 14.29%의 확률로 데이터가 선택될 것이다. 위 확률을 바탕으로 데이터 150개는 각각 class에 16, 18, 20, 21, 19, 28, 28 개로 나누어 졌다.

① 원래 데이터 분포와 clustering 이후의 분포 비교



실험 데이터를 확인해 보면 왼쪽 위의 그림과 같이 분포하는 것을 확인할 수 있다. 다음으로 EM 알고리즘을 반복적으로 실행하여 clustering 한 결과 오른쪽 위의 그림과 같이 모여 있는 몇 개의 class를 제외한 4개의 class는 잘 clustering 된 것을 확인할 수 있다.

② log likelihood값과 정확도 비교

```
found larger:  -1067.1030642675635
found larger:  -1067.1030629283466
found larger:  -1067.1030617118654
found larger:  -1067.1030606068175
found larger:  -1067.1030596029495
둘 차이가 매우 적음
final log likelihood :  -1067.1030596029495
```

```
accuracy :  0.82
가장 높은 정확도를 가지는 permutation :  (1, 5, 0, 6, 4, 2, 3)
```

실제 class

```
y_true[:,2]
```

```
array([2., 7., 2., 2., 6., 3., 6., 4., 6., 4., 1., 5., 6., 3., 3., 3., 6.,
       4., 6., 4., 7., 2., 6., 7., 6., 6., 7., 1., 3., 5., 7., 4., 2., 3.,
       7., 3., 1., 7., 7., 2., 4., 2., 2., 1., 7., 1., 7., 7., 4., 1., 2.,
       3., 3., 3., 7., 7., 6., 6., 2., 4., 4., 6., 6., 1., 4., 1., 7., 7.,
       7., 5., 3., 3., 2., 1., 7., 5., 6., 6., 7., 7., 5., 5., 4., 7., 1.,
       5., 5., 2., 5., 3., 1., 3., 6., 1., 4., 6., 4., 5., 6., 7., 6., 2.,
       1., 4., 3., 5., 6., 2., 5., 3., 7., 6., 2., 1., 7., 2., 5., 6., 4.,
       5., 7., 3., 4., 5., 6., 5., 4., 3., 4., 7., 6., 3., 1., 7., 4., 7.,
       3., 6., 2., 5., 6., 6., 1., 7., 5., 6., 4., 2., 5., 4.])
```

예측 class

```
r_list[index]
```

```
[array([4, 7, 4, 4, 6, 3, 6, 4, 6, 4, 1, 5, 6, 3, 5, 3, 6, 4, 6, 4, 7, 4,
        6, 6, 6, 6, 6, 1, 3, 5, 7, 4, 4, 3, 6, 3, 1, 7, 7, 4, 4, 4, 4, 1,
        7, 1, 7, 7, 4, 1, 4, 3, 3, 3, 7, 6, 6, 6, 4, 4, 4, 6, 6, 1, 4, 1,
        7, 7, 6, 5, 3, 3, 4, 1, 7, 5, 6, 6, 7, 7, 4, 5, 4, 7, 1, 5, 5, 4,
        5, 3, 1, 3, 6, 1, 4, 6, 4, 5, 6, 7, 6, 4, 1, 4, 3, 5, 6, 4, 5, 3,
        7, 6, 4, 1, 7, 4, 5, 6, 4, 5, 7, 3, 4, 7, 6, 5, 4, 3, 4, 7, 6, 3,
        1, 7, 4, 7, 3, 6, 4, 7, 6, 6, 1, 7, 5, 6, 4, 4, 5, 4], dtype=int64)]
```

본 실험에서 많은 iteration 후의 최종 log likelihood값은 -1067.1031이고 82%의 정확도를 가지며 그때 사용된 순열은 (1, 5, 0, 6, 4, 2, 3)이다.

③ 각 component 가 선택될 확률 α_i, 평균, 분산 비교

- 각 component 가 선택될 확률 α_i

```
예측 : class 별 선택될 확률:  [0.11978686 0.01250212 0.11550419 0.28160728 0.09439388 0.2122995
 0.16390616]
실제 : class 별 선택될 확률  [0.14285714 0.14285714 0.14285714 0.14285714 0.14285714 0.14285714
 0.14285714]
```

- 평균

```
['예측 : class1별 means'] [46.29553432 25.52160307]
['예측 : class2별 means'] [10.58663706 45.85311346]
['예측 : class3별 means'] [12.75148727 44.73757579]
['예측 : class4별 means'] [3.73035615 8.90537918]
['예측 : class5별 means'] [20.88097771 24.50590627]
['예측 : class6별 means'] [30.16243868 39.71018253]
['예측 : class7별 means'] [32.08341082 27.92440799]
------------------
['실제 : class1별 means'] [48.24925801 22.96849834]
['실제 : class2별 means'] [ 0.63765012 11.04245077]
['실제 : class3별 means'] [12.93098128 45.08792003]
['실제 : class4별 means'] [3.1748854  3.94998873]
['실제 : class5별 means'] [20.80725061 23.0491444 ]
['실제 : class6별 means'] [30.19900495 41.88722887]
['실제 : class7별 means'] [33.78292659 29.4533018 ]
```

- 분산

```
['예측 : class1의 variances']
[[ 25.17936608 -27.08072384]
 [-27.08072384  37.98751161]]
['예측 : class2의 variances']
[[6.55367002 2.48100629]
 [2.48100629 6.75951431]]
['예측 : class3의 variances']
[[ 8.95137166 -0.61955617]
 [-0.61955617  2.86324473]]
['예측 : class4의 variances']
[[33.64881384 11.48066586]
 [11.48066586 27.63512944]]
['예측 : class5의 variances']
[[23.49028484 -8.88375686]
 [-8.88375686 21.78059965]]
['예측 : class6의 variances']
[[13.01763841 -7.70376467]
 [-7.70376467 23.51420973]]
['예측 : class7의 variances']
[[13.66334753  5.24762774]
 [ 5.24762774  5.91430583]]
['실제 : class1의 variances']
[[9.08757281 0.53196707]
 [0.53196707 9.14580207]]
['실제 : class2의 variances']
[[10.88973586  0.73122705]
 [ 0.73122705 10.97262563]]
['실제 : class3의 variances']
[[9.07102498 0.37415289]
 [0.37415289 8.69118091]]
['실제 : class4의 variances']
[[5.18127894 0.71124884]
 [0.71124884 5.52641392]]
['실제 : class5의 variances']
[[9.07689367 0.1896441 ]
 [0.1896441  8.79866336]]
['실제 : class6의 variances']
[[10.57718596  0.60939905]
 [ 0.60939905 10.38638184]]
['실제 : class7의 variances']
[[5.86020037 0.4553058 ]
 [0.4553058  6.33279555]]
```

마지막으로 원래 data의 분산, 평균, 각 component가 선택될 확률 α_i을 EM알고리즘이 종료된 후 우리가 구한 값들과 비교해 보면 2번 class의 α_i와 평균을 잘 예측하지 못하고, 대부분의 분산을 잘 예측하지 못하는 것을 확인할 수 있다.

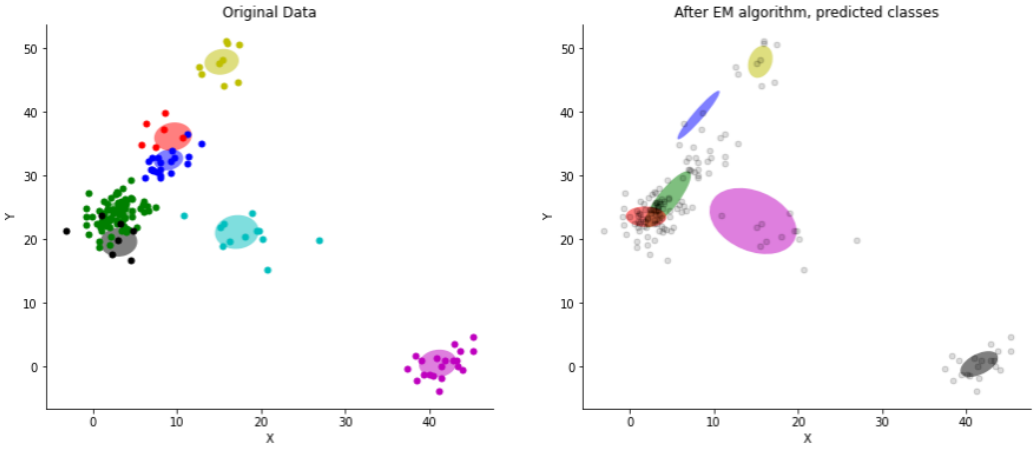
3.3.2 각 component 별 선택될 확률 α_i가 다를 때

```
The probabilities of each classes from 1 to 7
[0.13772253 0.53480718 0.02453108 0.08198857 0.1357185  0.06144018
 0.02379195]
```

```
The number of objects in each classes from 1 to 7
[20 76  6 12 20  9  7]
```

각 component별로 선택될 확률 α_i 가 왼쪽 위의 사진과 같은 각각 다른 확률로 선택되었고 위 확률을 바탕으로 데이터는 각각 class에 20, 76, 6, 12, 20, 9, 7개로 나누어 졌다.

① 원래 데이터 분포와 clustering 이후의 분포 비교



실험 데이터를 확인해 보면 왼쪽 위의 그림과 같이 분포하는 것을 확인할 수 있다. 다음으로 EM 알고리즘을 반복적으로 실행하여 clustering 한 결과 오른쪽 위의 그림과 같이 모여 있지 않은 class에 대한 데이터만 잘 clustering 된 것을 확인할 수 있다.

② log likelihood값과 정확도 비교

```
found larger:  -859.0429950507983
found larger:  -859.0429918655959
found larger:  -859.042989519728
found larger:  -859.0429877918873
found larger:  -859.0429865191616
둘 차이가 매우 적음                    accuracy :  0.7333333333333333
final log likelihood :  -859.0429865191616 가장 높은 정확도를 가지는 permutation :  (3, 1, 0, 4, 6, 5, 2)
```

실제 class

```
y_true[:,2]
```

```
array([2., 2., 2., 2., 2., 1., 5., 2., 2., 2., 1., 3., 2., 2., 4., 5., 2.,
       4., 2., 1., 2., 2., 2., 3., 2., 2., 2., 2., 2., 1., 5., 1., 5., 2.,
       3., 1., 2., 5., 6., 1., 2., 4., 4., 4., 2., 2., 5., 2., 1., 1., 4.,
       4., 2., 2., 2., 2., 2., 5., 1., 3., 2., 2., 4., 4., 6., 1., 2., 2.,
       2., 2., 3., 2., 5., 2., 7., 6., 2., 6., 2., 5., 3., 5., 2., 7., 5.,
       2., 7., 2., 7., 2., 2., 2., 2., 7., 2., 2., 5., 6., 6., 4., 5., 6.,
       2., 2., 2., 1., 6., 2., 5., 2., 4., 5., 2., 2., 5., 2., 2., 2., 5.,
       4., 2., 2., 1., 2., 6., 1., 1., 5., 2., 1., 1., 2., 2., 5., 2., 2.,
       7., 2., 2., 1., 1., 2., 2., 2., 2., 1., 2., 2., 7., 5.])
```

예측 class

```
r_list[index]
```

```
[array([2, 2, 2, 2, 2, 2, 5, 2, 2, 2, 2, 2, 2, 2, 4, 5, 2, 4, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 5, 4, 5, 2, 3, 2, 2, 5, 6, 2, 2, 4, 4, 4,
        2, 7, 5, 2, 2, 2, 1, 4, 7, 2, 2, 2, 7, 5, 2, 2, 7, 2, 4, 4, 6, 4,
        2, 2, 2, 7, 3, 7, 5, 2, 7, 6, 7, 6, 7, 5, 3, 5, 2, 2, 5, 7, 2, 2,
        7, 2, 2, 2, 2, 2, 2, 2, 2, 5, 6, 6, 4, 5, 6, 2, 2, 2, 2, 6, 2, 5, 2,
        4, 5, 2, 7, 5, 2, 2, 2, 5, 4, 2, 2, 2, 2, 6, 2, 2, 5, 2, 2, 2, 2,
        2, 5, 2, 2, 7, 2, 2, 2, 4, 7, 2, 2, 2, 2, 7, 2, 4, 5], dtype=int64)]
```

본 실험에서 많은 iteration 후의 최종 log likelihood값은 -859.0430이고 73.3%의 정확도를 가지며 그때 사용된 순열은 (3, 1, 0, 4, 6, 5, 2)이다.

③ 각 component 가 선택될 확률 α_i, 평균, 분산 비교

- 각 component 가 선택될 확률 α_i

```
예측 : class 별 선택될 확률:  [0.00666667 0.50930085 0.02299707 0.10333754 0.13333333 0.05581583
 0.16854871]
실제 : class 별 선택될 확률  [0.13772253 0.53480718 0.02453108 0.08198857 0.1357185  0.06144018
 0.02379195]
```

- 평균

```
['예측 : class1별 means'] [26.95518639 19.82915564]
['예측 : class2별 means'] [ 4.51428103 26.25049594]
['예측 : class3별 means'] [ 8.17752666 39.49873628]
['예측 : class4별 means'] [14.65611153 22.84773007]
['예측 : class5별 means'] [41.51975465  0.38463965]
['예측 : class6별 means'] [15.5201148  47.86662702]
['예측 : class7별 means'] [ 1.93893823 23.52421805]
------------------
['실제 : class1별 means'] [ 9.00713095 32.46948453]
['실제 : class2별 means'] [ 2.86583966 24.1307389 ]
['실제 : class3별 means'] [ 9.53627318 36.10652766]
['실제 : class4별 means'] [17.12937262 21.12766994]
['실제 : class5별 means'] [41.01227104  0.47782053]
['실제 : class6별 means'] [15.32646781 47.86067735]
['실제 : class7별 means'] [ 3.08900997 19.53911334]
```

- 분산

```
['예측 : class1의 variances']
[[0. 0.]
 [0. 0.]]
['예측 : class2의 variances']
[[ 7.57368042 10.25669521]
 [10.25669521 19.57359707]]
['예측 : class3의 variances']
[[ 6.30273897  9.12952453]
 [ 9.12952453 14.58948911]]
['예측 : class4의 variances']
[[26.70525407 -8.23085845]
 [-8.23085845 26.85898192]]
['예측 : class5의 variances']
[[4.96875114 2.28531426]
 [2.28531426 4.09561557]]
['예측 : class6의 variances']
[[2.13552408 0.85969095]
 [0.85969095 6.3906668 ]]
['예측 : class7의 variances']
[[ 5.7816734  -0.21079182]
 [-0.21079182  2.59355651]]
```

```
['실제 : class1의 variances']
[[3.03873572 0.46609827]
 [0.46609827 2.77780405]]
['실제 : class2의 variances']
[[3.07267146 0.38923667]
 [0.38923667 2.9642182 ]]
['실제 : class3의 variances']
[[4.99280009 0.47609278]
 [0.47609278 4.98085794]]
['실제 : class4의 variances']
[[6.79029298 0.37225363]
 [0.37225363 7.09079959]]
['실제 : class5의 variances']
[[5.06955762 0.42190482]
 [0.42190482 4.73524559]]
['실제 : class6의 variances']
[[4.18876422 0.44568275]
 [0.44568275 4.1255659 ]]
['실제 : class7의 variances']
[[4.89572599 0.16536169]
 [0.16536169 5.18100945]]
```

마지막으로 원래 data의 분산, 평균, 각 component가 선택될 확률 α_i을 EM알고리즘이 종료된 후 우리가 구한 값들과 비교해 보면 1, 7번 class의 α_i와 1번 class의 평균을 잘 예측하지 못하고, 3번 class의 분산을 잘 예측하지 못하는 것을 확인할 수 있다.

4. Conclusion

EM 알고리즘을 Gaussian Mixture Model에 적용하여 Clustering을 진행해 보았다. 본 실험은 초기 확률과 평균, 분산을 random한 값으로 설정한 후 log likelihood값의 차이가 10^{-6}보다 작을 때까지, $p(y_i|x_i,\theta^{(g)})$를 구하는 E-step과, $\theta_j = \{\alpha_j, \Sigma_j\}$와 각 component가 선택될 확률 α_j를 업데이트하는 M-step을 반복하여 진행한다. 즉 EM 알고리즘을 통해 log likelihood값을 최대로 하는 평균과 분산을 찾아서 clustering을 진행한다.

본 실험에서는 비교를 위하여 같은 수의 데이터, 총 150개의 데이터를 class의 개수를 바꾸어 가며 실행하였다. Class 3, 5, 7개로 실행을 해보며 비교를 해 보았는데 class 3개로 진행하였을 때는 보통 100%의 정확도로, 5개로 진행되었을 때는 평균적으로 95% 이상의 정확도로, 마지막으로 7개로 진행하였을 때는 75% 정도의 정확도를 보였다.

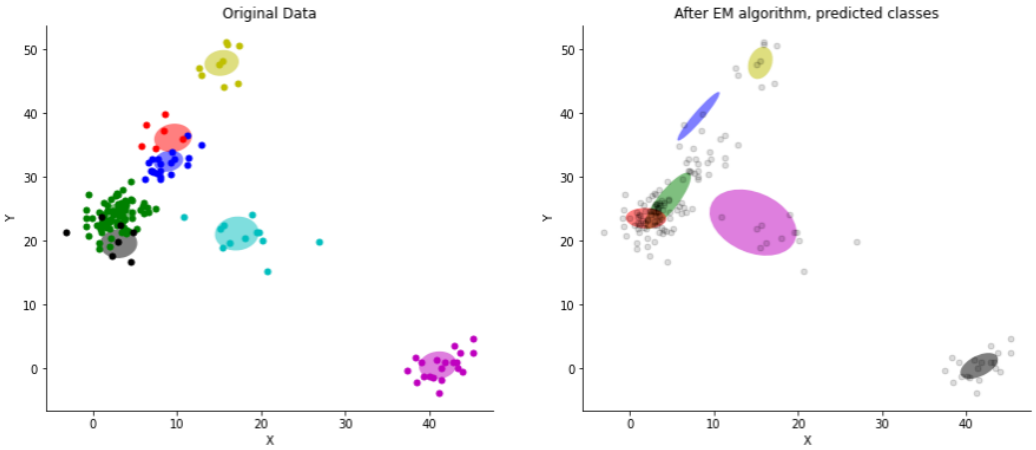


Figure 7 class 7개중 몇 개의 class가 겹쳐진 경우

같은 양의 데이터를 사용해 class를 나누다 보니 class가 적을수록 clustering이 진행이 잘 되었지만 class수가 7개일때는 위의 그림처럼 class들이 모여 있는 현상이 많이 나타났다. 따라서 EM 알고리즘을 반복적으로 실행하여도 직관적으로 모여 있는 데이터들의 clustering이 어려웠고 결과적으로 예측된 7개의 class들의 분산과 평균, 각 component가 선택될 확률 α_i값에서 차이가 많이 발견되었다.

즉 결과적으로 데이터의 수가 동일할 때, class의 수가 많아질수록 모여 있는 class가 많이 발견될 확률이 높아지기 때문에 clustering에 어려움이 따른다.

하지만 class수가 적고 (보통 5이하) 각 class들의 평균 거리가 멀고 잘 퍼져 있다면 clustering 결과는 매우 높고 그때의 log likelihood값도 높은 것을 실험을 통해 확인할 수 있었다.