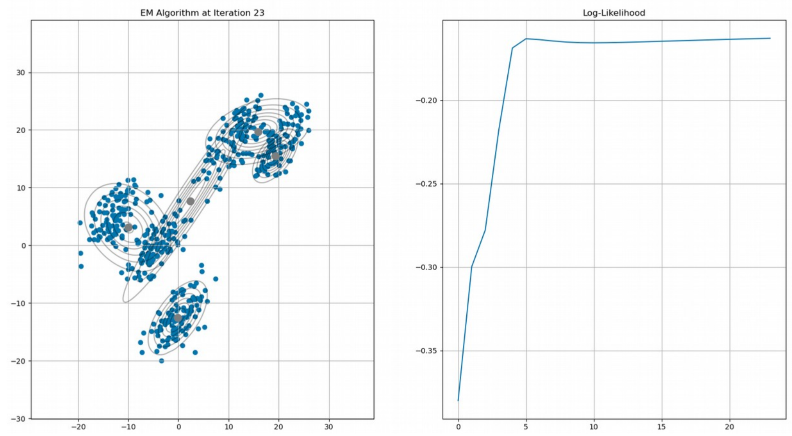
EM Algorithm 을 이용한 GMM Clustering

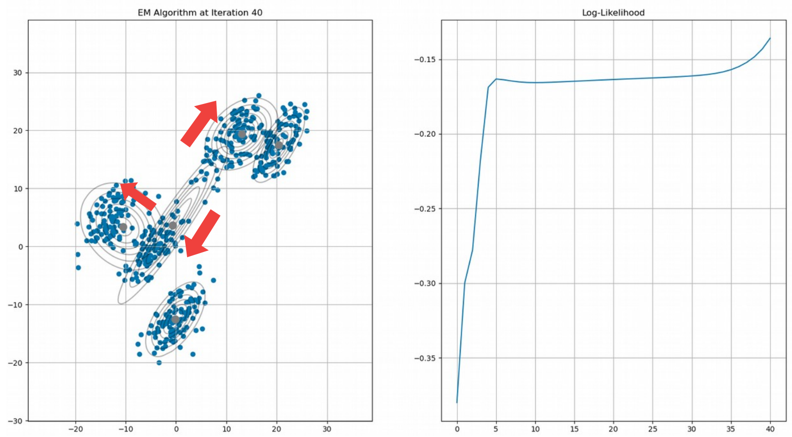
```python
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.datasets import make_moons
import numpy as np
from scipy.stats import multivariate_normal

class GMM:

    # GMM  생성자
    def __init__(self, iterations, dataset_type='blob', n_samples=2000):

        self.iterations = iterations  # GMM 을 최적화기 위한 EM Algorithm  반복 횟수
        self.mu = None    # 각 Gaussian Cluster 별 Mean 값 리스트
        self.cov = []   # 각 Gaussian Cluster 별 Covariance Matrix  리스트
        self.pi = None    # 데이터셋의 각 Gaussian Cluster 에 대한 소속 비율

        # 데이터셋 종류에 따라 준비함
        if dataset_type is 'blob':  # Blob  데이터셋 준비
            X, Y = make_blobs(cluster_std=1.5, random_state=20, n_samples=n_samples, centers=5)    # Blob  데이터셋 생성
            X = np.dot(X, np.random.RandomState(0).randn(2, 2))    # Blob  데이터셋이 타원형이 될 수 있게 만듬
            self.X = X  # 데이터셋 멤버 변수화
            self.number_of_sources = len(np.unique(Y))  # 데이터셋의 클러스터 종류 개수

        elif dataset_type is 'moon':  # Moon  데이터셋 준비
            X, Y = make_moons(n_samples=n_samples, noise=0.05, random_state=0)  # Moon  데이터셋 생성
            self.X = X  # 데이터셋 멤버 변수화
            self.number_of_sources = len(np.unique(Y))  # 데이터셋의 클러스터 종류 개수

    # EM Algorithm 을 이용한 GMM Clustering  수행
    def run(self, random_init=True):

        self.reg_cov = 1e-6 * np.identity(self.X.shape[1])  # Singularity Issue (Gaussian Cluster 가 한 점에 대해 Overfitting  되는 현상 ) 를 방지하기 위해
                                                             # Covariance Matrix 가 0 Matrix 가 되지 않도록 매우 작은 값을 가지는 Identity Matrix 를 더해줌

        # GMM Singularity Issue  해결방법 1 :  매우 작은 값을 추가한 Identity Matrix 를 Covariance Matrix 에 더함 (sklearn  구현방식 )
        # GMM Singularity Issue  해결방법 2 :  특정 Threshold  기준으로 Singularity  발생 여부를 확인되면 Gaussian Cluster 의 Mean  위치를 다른 랜덤한 위치로 재설정함
        # 출처 : https://stats.stackexchange.com/questions/219302/singularity-issues-in-gaussian-mixture-model

        # 각 Gaussian Cluster 의 평균점 (Cluster  중심 좌표 ) 를 랜덤 좌표로 초기화하지 않는다면 ...
        if random_init is False:
            # 모든 Gaussian Cluster 의 평균점 (Cluster  중심 좌표 ) 를 (0, 0) 으로 초기화함
            self.mu = np.zeros((self.number_of_sources, 2))

        # 각 Gaussian Cluster 의 평균점 (Cluster  중심 좌표 ) 를 랜덤 좌표로 초기화하면
        else:
            # 각 Gaussain Cluster 의 평균점 (Cluster  중심 좌표 ) 를 데이터셋 범위내의 랜덤 좌표로 할당함
            self.mu = np.random.randint(min(self.X[:,0]),max(self.X[:,0]),size=(self.number_of_sources,self.X.shape[1]))

        # 각 Gaussian Cluster 의 Covariance Matrix 를 Identity Matrix 로 초기화함
        for i in range(self.number_of_sources):
            self.cov.append(np.identity(self.X.shape[1]))

        # 각 Gaussian Cluster 가 데이터셋에 대해 동일한 비율만큼 차지하는 것으로 초기화함
        self.pi = np.ones(self.number_of_sources) / self.number_of_sources

        log_likelihoods = []  # 데이터셋에 대한 Log-Likelihood 를 저장하는 리스트

        # 정해진 반복횟수만큼 EM  알고리즘을 통해 각 Gaussian Cluster 의 중심 (Mean) 에 최대한 많은 데이터가 모일 수 있도록 중심점 (Mean) 과 포함 범위 (Covariance Matrix) 를 변경해나감
        for i in range(self.iterations):

            ####################################################################
            ### E-Step :  데이터셋의 각 데이터가 특정 Cluster 에 해당되는 확률을 구함 ###
            ####################################################################
            r_ic = np.zeros((self.X.shape[0], self.number_of_sources))
            # 데이터셋 (self.X.shape[0] :  데이터셋 개수 ) 이 각 Gaussian Cluster(self.number_of_sources : Cluster  개수 ) 에 속할 확률을 리스트로 저장함

            # 각 Gaussain Cluster 의 평균 , Covariance Matrix, Pi  ( 데이터별 소속될 비율 ) 에 대해 전체 데이터셋의 소속 정도를 평가함
            for m, co, p, r in zip(self.mu, self.cov, self.pi, range(len(r_ic[0]))):

                co += self.reg_cov  # Singularity Issue 를 방지하기 위해 Covariance Matrix 가 0 이 되지 않게 매우 작은 값을 더함

                mn = multivariate_normal(mean=m, cov=co)  # 현재 사용하는 평균 , Covariance 를 반영한 Gaussian Cluster  준비

                # 현재 Gaussian Cluster 에 대해 모든 데이터셋이 소속될 확률을 구함
                # 해당 확률을 전체 Cluster 에 대한 소속 확률의 합으로 0 ~ 1 사이로 Noramlize 함
                r_ic[:, r] = p * mn.pdf(self.X) / np.sum([pi_c * multivariate_normal(mean=mu_c, cov=cov_c).pdf(self.X) for pi_c, mu_c, cov_c in zip(self.pi, self.mu, self.cov+self.reg_cov)], axis=0)

            #################################################################################################################################################################################################
            ### M-Step :  각 데이터의 Cluster 별 소속 비율을 반영해서 Gaussian Cluster 의 평균 ( 중심점 좌표 ) 를 더 많은 소속 비율을 가진 데이터를 향해서 변동이 되도록 평균을 Weighted Mean, Covriance 를 Weighted Covariance 로 변경함 ###
            #################################################################################################################################################################################################
            self.mu = []    # 각 Gaussian Cluster 별 평균을 새롭게 업데이트하기 위해 준비함
            self.cov = []   # 각 Gaussian Cluster 별 Covariance Matrix 를 새롭게 업데이트하기 위해 준비함
            self.pi = []    # 각 Gaussian Cluster 별 데이터셋에 대한 점유율 업데이트하기 위해 준비함
            log_likelihood = []   # 매 Iteration 마다 Log-Likelihood 를 저장하기 위한 리스트

            # 각 Gaussian Cluster 별 평균 ( 중심점 좌표 ), Covariance Matrix,  점유율을 새롭게 업데이트함
            for c in range(len(r_ic[0])):
                m_c = np.sum(r_ic[:, c], axis=0)  # 각 Gaussian Cluster 에 대해 전체 데이터셋의 소속 비율합 업데이트
                # 만약 소속 비율합이 0 이 되는 경우 Gaussian Cluster  평균 연산시 발생하는 문제를 방지하기 위해  0 이 되는 경우  1 로 만듬
                if (m_c == 0.0): m_c = 1
                mu_c = (1/m_c) * np.sum(self.X * r_ic[:, c].reshape(len(self.X), 1), axis=0)  # 각 Gaussian Cluster 의 평균 ( 중심점 좌표 ) 를 연산함
                self.mu.append(mu_c)  # 각 Gaussian Cluster 의 평균 ( 중심점 좌표 ) 를 업데이트함 (Gaussian Cluster  중심에 더 많은 데이터가 소속되는 방향으로 중심점 좌표가 업데이트됨 )

                # 각 Gaussian Cluster 의 Covariance 를 연산하고 업데이트함
                self.cov.append(((1/m_c)*np.dot((np.array(r_ic[:,c]).reshape(len(self.X),1)*(self.X-mu_c)).T,(self.X-mu_c)))+self.reg_cov)

                self.pi.append(m_c/np.sum(r_ic))  # 각 Gaussian Cluster 의 데이터셋에 대한 점유율 업데이트

            # 전체 데이터셋의 각 Gaussian Cluster 에 해당되는 값들의 합을 Log-Likelihood 로 변환하여 저장함
            log_likelihoods.append(np.log(np.sum([k*multivariate_normal(self.mu[i],self.cov[j]).pdf(self.X) for k,i,j in zip(self.pi,range(len(self.mu)),range(len(self.cov)))])))

            ####################################################
            ### Singularity Check & Random Re-Initialization ###
            ####################################################

            ### Singularity :  데이터 1 개가 Gaussian Cluster  평균 / 중심점에 완전 일치하게 되면서 해당 지점을 중심으로 Gaussian Cluster 가 Overfitting 되는 현상
            ###             :  데이터 1 개 중심으로 Gaussian Cluster 가 Overfitting 되면 해당 지점을 중심으로 Gaussian 이 매우 크게 나타나기 때문에 Covariance 가 매우 낮게 나타남
            ###             :  이러한 현상은 Covariance Matrix 가 Singular Matrix (Determinant = 0) 인 경우 발생하며 ,  이 때 Gaussian Cluster 의 중심점을 랜덤하게 다른 위치로 옮겨서 다르게 Cluster 를 구성하게 만듬

            singularity = np.zeros(self.number_of_sources)  # 각 Gaussian Cluster 별 Singularity  존재여부를 저장하기 위한 리스트

            # 각 Gaussian Cluster 의 업데이트된 Covariance Matrix 의 Determinant 가 특정값 이하인 경우 해당 Cluster 에 Singularity 가 발생했다고 간주함
            # 컴퓨터 연산을 이용한 Determinant  연산은 0 을 0 이 아닌 매우 낮은 값으로 산출할 수 있기 때문에 ,  기준을 0 이 아닌 특정 매우 낮은값 이하로 결정함
            for co, cluster_idx in zip(self.cov, range(self.number_of_sources)):
                if abs(np.linalg.det(co)) < 1e-2: # Determinant 의 크기가 0.01  이하인 경우 Singularity 라 간주함
                    singularity[cluster_idx] = 1
            print('--- Singluarity Check : {} ---'.format(singularity))

            # 각 Gaussian Cluster  중 Singularity 가 발생하면 평균 / 중심점으로 다시 랜덤하게 배정하고 , Covariance Matrix 를 초기화함
            # 그리고 해당 Gaussian Cluster 의 점유 비율을 상대적을 높게 배정해서 기존의 Gaussian Cluster  사이에 침투할 수 있도록 허용해줌
            for singularity_idx in range(len(singularity)):
                if singularity[singularity_idx] == 1:
                    self.mu[singularity_idx] = np.random.randint(min(self.X[:,0]), max(self.X[:,0]), size=(self.X.shape[1]))  # Gaussian Cluster  중심점을 랜덤하게 재할당
                    self.cov[singularity_idx] = np.identity(self.X.shape[1]) + self.reg_cov     # Gaussian Cluster 의 Covariance Matrix  재초기화
                    self.pi[singularity_idx] = 1  # Gaussian Cluster 의 점유 비율을 상대적으로 높게 배정 /  기존 Cluster  영역을 침투할 수 있게 허용함

            ###############################
            ### Clustering  결과 Plotting ###
            ###############################
            # 매 Iteration 마다 Gaussian Cluster 의 평균 / 중심점을 기준으로 등고선을 그림
            plt.subplot(1, 2, 1)
            plt.title('EM Algorithm at Iteration {}'.format(i))
            plt.xlim(1.5 * min(self.X[:,0]), 1.5 * max(self.X[:,0]))
            plt.ylim(1.5 * min(self.X[:,1]), 1.5 * max(self.X[:,1]))
            plt.grid()
            x, y = np.meshgrid(np.sort(self.X[:, 0]), np.sort(self.X[:, 1]))
            XY = np.array([x.flatten(), y.flatten()]).T
            plt.scatter(self.X[:,0],self.X[:,1])
            for m,c in zip(self.mu,self.cov):
                c += self.reg_cov
                multi_normal = multivariate_normal(mean=m,cov=c)
                plt.contour(np.sort(self.X[:,0]),np.sort(self.X[:,1]),multi_normal.pdf(XY).reshape(len(self.X),len(self.X)),colors='black',alpha=0.3)
                plt.scatter(m[0],m[1],c='grey',zorder=10,s=100)
            # 현재까지 수행한 Iteration 에서 산출된 Log-Likelihood 를 그래프로 그림
            plt.subplot(1, 2, 2)
            plt.title('Log-Likelihood')
            plt.grid()
            plt.plot(range(len(log_likelihoods)),log_likelihoods)
            # 지속적으로 GMM Clustering  결과를 그림
            plt.pause(0.005)
            plt.show(block=False)
            plt.clf()

# 메인 함수 실행
if __name__ == '__main__':

    # Blob  또는 Moon  데이터 500 개 데이터셋에 대해 200 회 EM Algorithm 을 통해 GMM Clustering  수행함
    GMM = GMM(iterations=200, dataset_type='blob', n_samples=500)
    GMM.run(random_init=True)
```

EM Algorithm 을 이용한 GMM Clustering 결과 및 분석

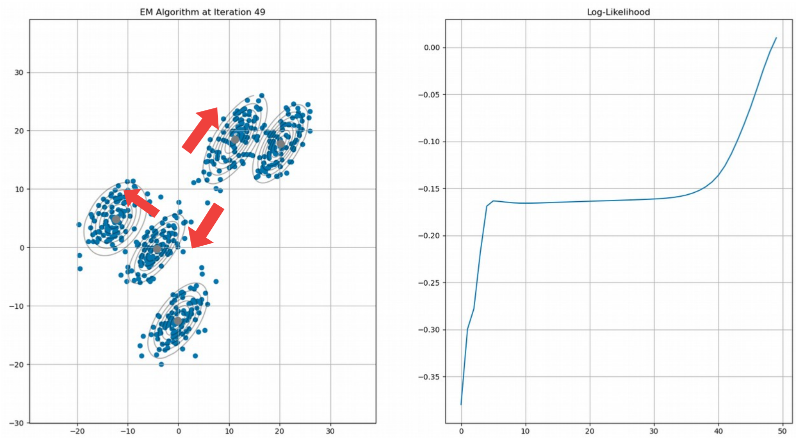
1. Blob Dataset – Ideal 한 GMM Clustering 결과



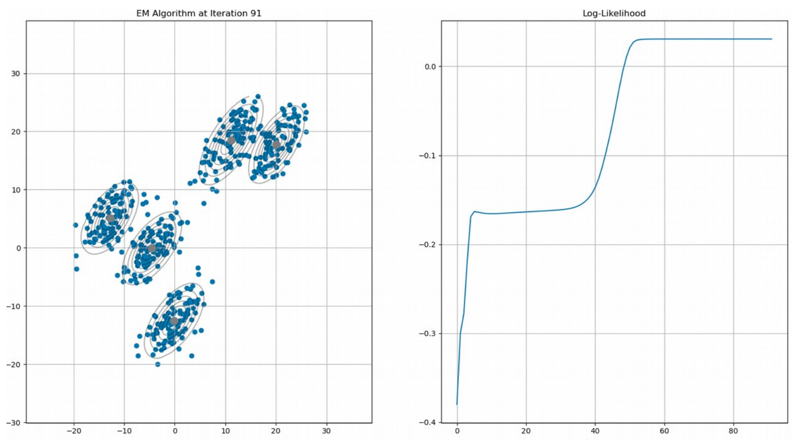
초반에 각 Gaussian Cluster 의 중심점이 적절한 Random 위치에 배치되면 각 Gaussian Cluster 는 제일 가까운 데이터 클러스터를 최대한 많이 포함시기키 위해서 해당 클러스터를 향해 경합적으로 Gaussian Cluster 의 중심점을 업데이트하면서 움직이게됨



Gaussian Cluster 들 사이에 주변 데이터를 최대한 많이 포함시키기 위해 움직이게 되면서 중간에 겹치는 Gaussian Cluster 와 경쟁을 하게됨 .
이는 각 데이터의 각 Gaussian Cluster 에 대한 소속 비율인 r_{ic} 로 표현됨 . 각 Gaussian Cluster 의 평균점 (Cluster 중심) 이 모든 데이터셋의 r_{ic} 를 반영한 Weighted Sum 으로 업데이트 되기 때문에 가까이 많이 있는 데이터 클러스터를 향해서 Gaussian Cluster 가 이동하게됨 .



Gaussian Cluster 가 이동하게되면서 평균 (Mean) 을 중심으로 데이터가 많이 모여서 Log Likelihood 가 증가하게됨 .

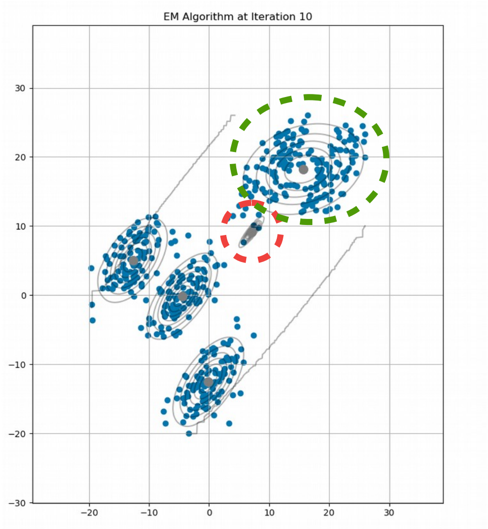


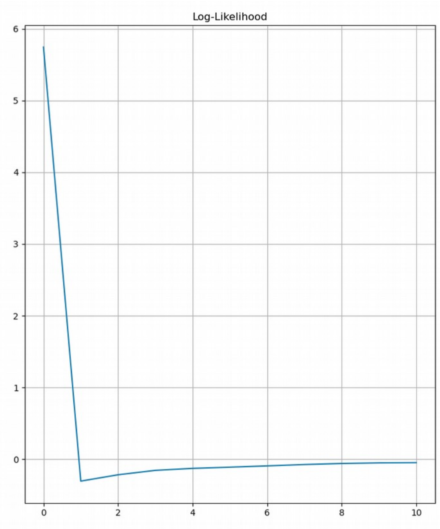
Gaussian Cluster 가 적절한 영역만큼 점유하게 되면 Log Likelihood 가 더이상 증가하지 않고 수렴하게됨 .

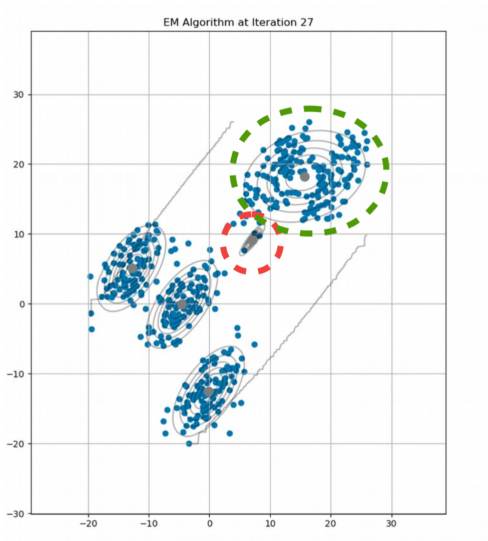
Ideal 한 경우에는 Log Likelihood 가 제대로 수렴되면서 Gaussian Cluster 의 각각의 영역이 적절히 배분되는 것을 볼 수 있음

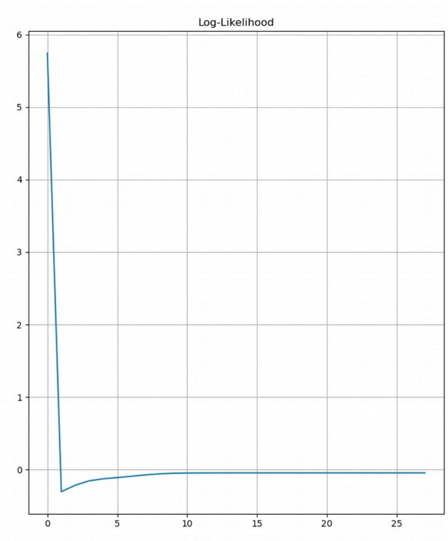
EM Algorithm 을 이용한 GMM Clustering 결과 및 분석

2. Blob Dataset – Gaussian Cluster 의 평균점 / 중심점 Random 초기화에 의한 Clustering 실패 상황









Gaussian Cluster 초기 평균 / 중심 좌표가 잘못 배치되면 일부 Cluster 가 과도한 데이터 클러스터 영역을 점유하고, 다른 Cluster 가 소외되는 상황이 발생할 수 있음 .

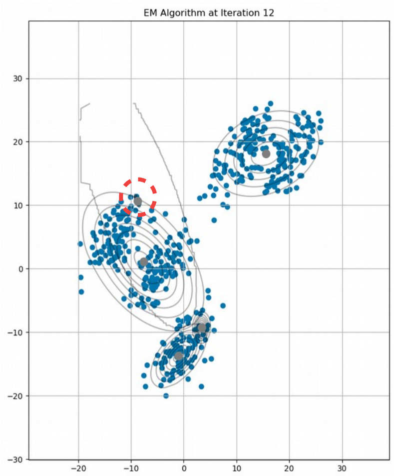
초록색 영역으로 표시된 영역은 초기 Gaussian Cluster 의 평균 / 중심 좌표가 2 개의 매우 인접한 데이터 클러스터의 정중앙에 랜덤으로 배치되면서 2 개의 데이터 클러스터를 점유하게됨 .

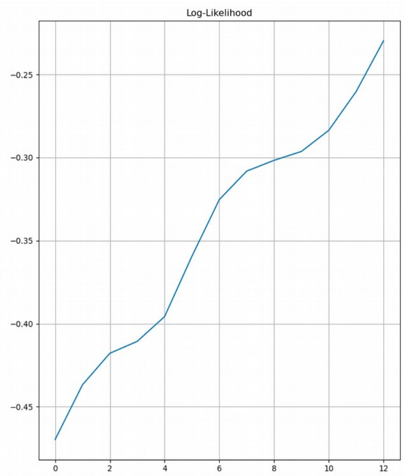
이로 인해 빨간색의 다른 Gaussian Cluster 는 초록색 영역의 Gaussian Cluster 보다 점유율을 높이기 힘들기 때문에 매우 작은 영역만 점유하게됨 .
그러나 이 협소한 Gaussian Cluster 가 Overfitting 이 발생하는 Singularity 경우가 아니기 때문에 다시 재배치할 수 없음 .
단순히 Cluster 의 크기를 기준으로 재배치 여부를 결정할 수 없기에 (매우 작은 데이터 클러스터가 존재할 수 있기 때문임) 현재 상태가 계속 유지되며 Log Likelihood 도 더이상 개선되지 않음 .

이를 통해서 EM 알고리즘을 통한 GMM Clustering 은 초기 Gaussian Cluster 의 배치 상태 / Initialization 에 성능이 결정된다는 한계점을 볼 수 있음 .

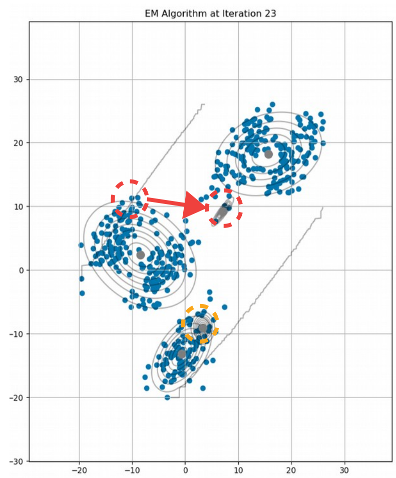
EM Algorithm 을 이용한 GMM Clustering 결과 및 분석

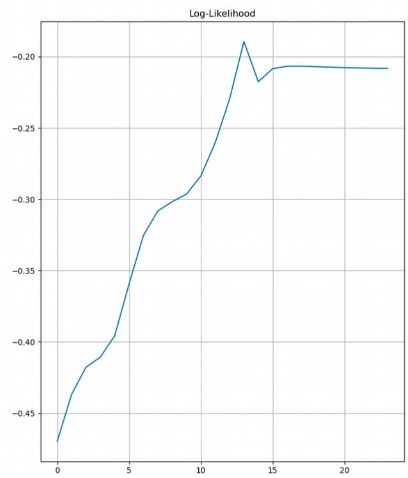
3. Blob Dataset – Covariance Matrix 에 대한 Singularity Check 를 통한 재초기화



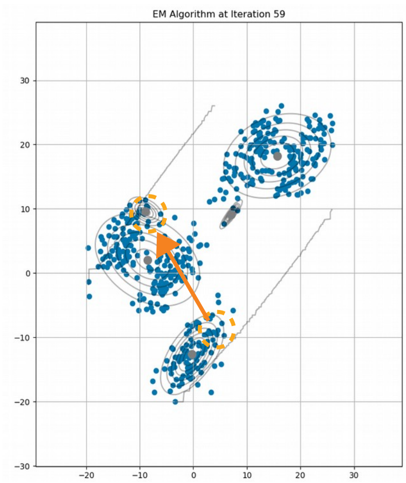


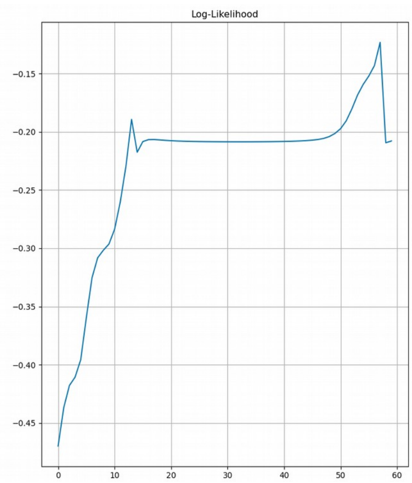
빨간색에 해당되는 Gaussian Cluster 의 평균 / 중심점이 데이터셋의 한 점에 일치하게 되면서 해당 데이터를 중심으로 Gaussian Cluster 가 Fitting 되려는 Singularity 현상이 발생함 . 이로 인해 해당 데이터센의 한 점에 모이려는 경향을 가지게 되면서 Covariance Matrix 가 매우 작아지면서 Gaussian Cluster 가 점점 좁아지게됨 .



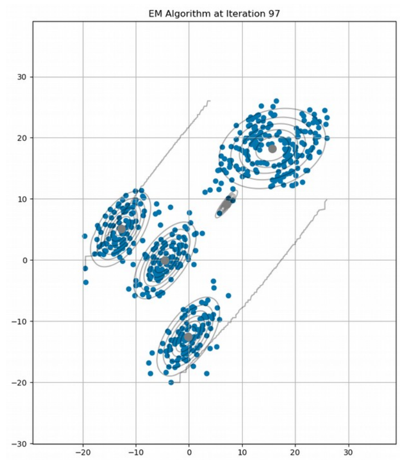


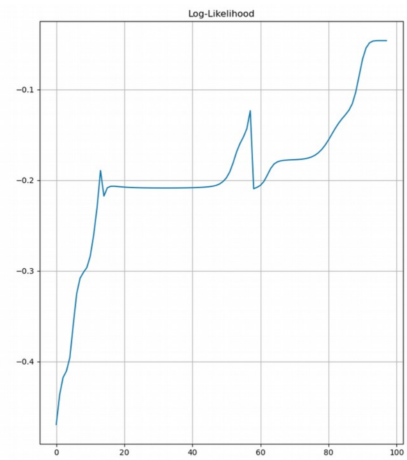
이러한 Singularity 를 검사하기 위해서 Gaussian Cluster 의 Covariance Matrix 의 Determinant 가 0 이 되는지 확임함 . 그러나 컴퓨터 연산에서 Determinant 결과 0 을 0 이라고 산출하지 못하고 1e-6 과 같이 매우 작은 수로 출력함 . Determinant 를 매우 작은 특정 Threshold 값보다 낮은 경우 해당 Gaussian Cluster 를 다른 랜덤한 지점으로 재배치하고 점유 가능율로 높게 줘서 다른 Cluster 를 침투할 수 있게함 .





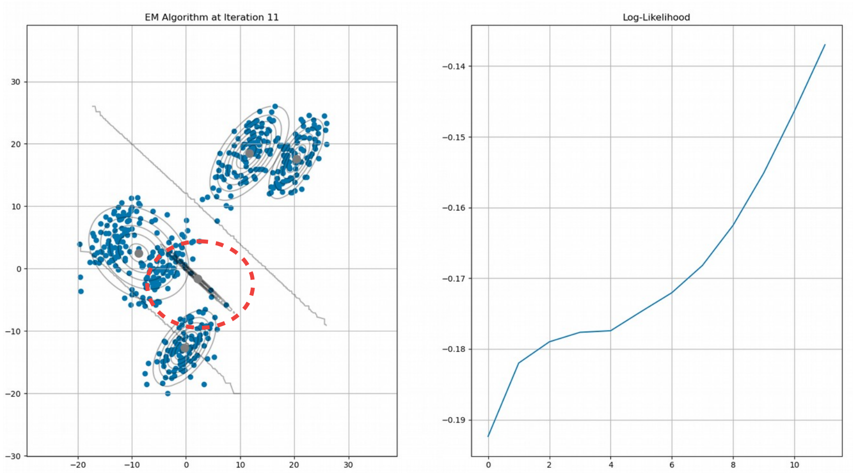
Singularity Check 를 통해 Overfitting 된 Gaussian Cluster 인 빨간색 영역 Gaussian Cluster 와 주황색 영역 Gaussian Cluster 가 다른 위치로 재배치되어 이전보다 개선된 Clustering 결과를 보여줌 . 그러나 이 또한 랜덤한 재배치이기 때문에 이전보다 개선될 뿐 Ideal 한 경우와 같은 완전한 Clustering 결과를 가져오기 힘듬 .



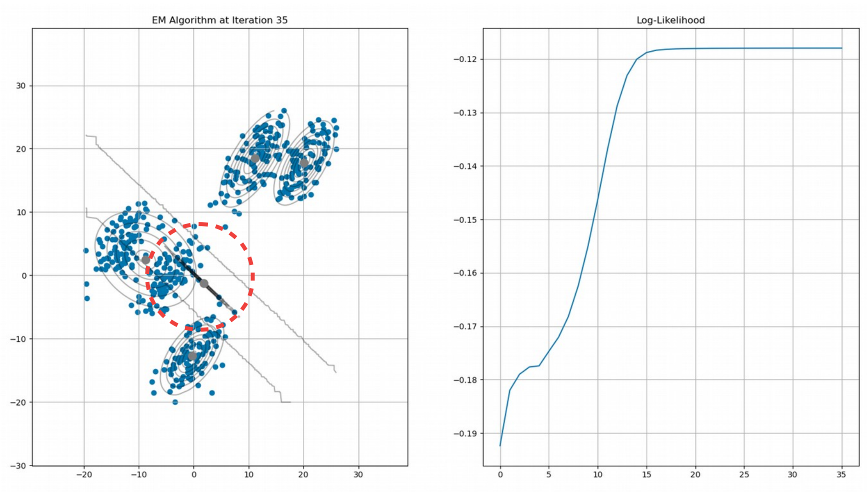


EM Algorithm 을 이용한 GMM Clustering 결과 및 분석

4. Blob Dataset – Covariance Matrix 에 대한 Singularity Check 를 통한 재초기화 실패



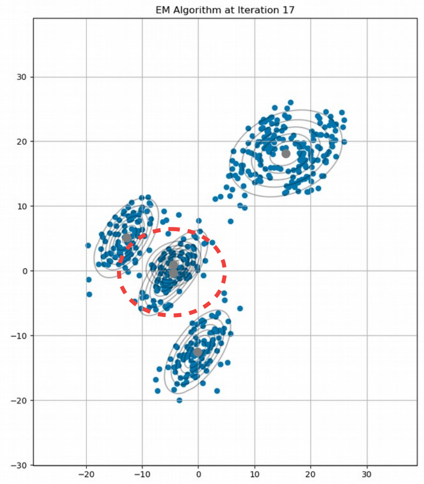
빨간색에 해당되는 Gaussian Cluster 는 데이터셋의 한 점을 중심으로 Overfitting 되는 Singularity 처럼 보이지만 실제로는 Covariance Matrix 의 Determinant 가 0 에 근접하지 않음 . 이러한 상황은 데이터셋의 한 점에 Fitting 되는 것이 아니라 매우 Sparse 한 데이터 클러스터를 포함하게되면서 극단적으로 타원형으로 분포가 생성되는 경우임 . 그로 인해 Covariance 가 한 점으로 수렴되지 않음 .

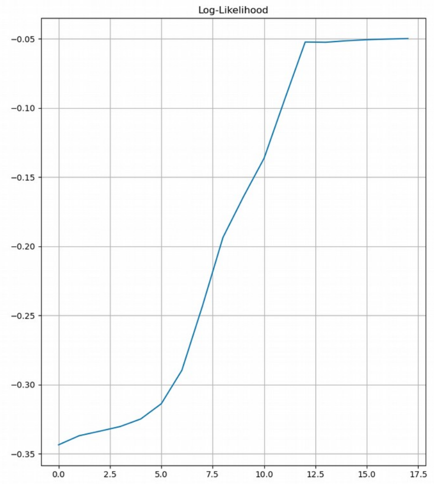


이러한 경우는 Singularity 로 취급할 수 없기 때문에 Singularity Check 를 통한 재배치를 구현하기 매우 까다로움 . 또한 Sparse 한 데이터 클러스터를 영역 내에 포함하기 때문에 Log Likelihood 도 낮게 나오지 않음 .

EM Algorithm 을 이용한 GMM Clustering 결과 및 분석

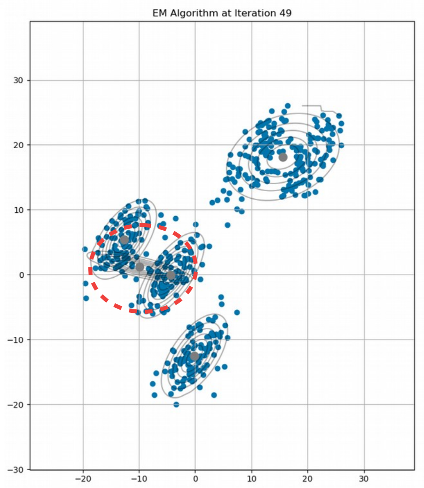
5. Blob Dataset – Gaussian Cluster 의 혼합 공존 상황

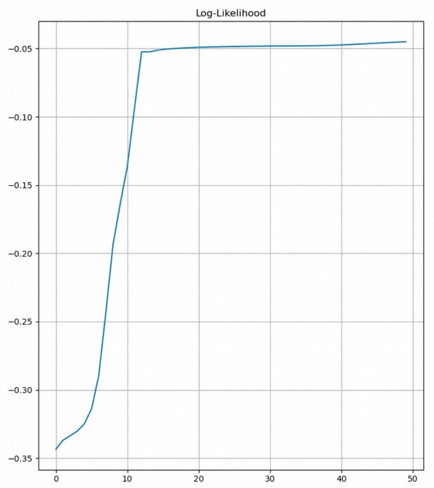




GMM 은 Soft Clustering 기법이기 때문에 하나의 데이터에 대해서 여러 Gaussin Cluster 가 걸쳐있을 수 있음 .

빨간색 영역에 2 ~ 3 개의 Gaussian Clutser 가 공동의 데이터 클러스터를 포함하는 것을 볼 수 있음 . Hard Clustering 기법의 경우에는 데이터가 1 개의 Cluster 만 속하기 때문에 하나의 데이터가 여러 Cluster 에 배정될 수 없음 .

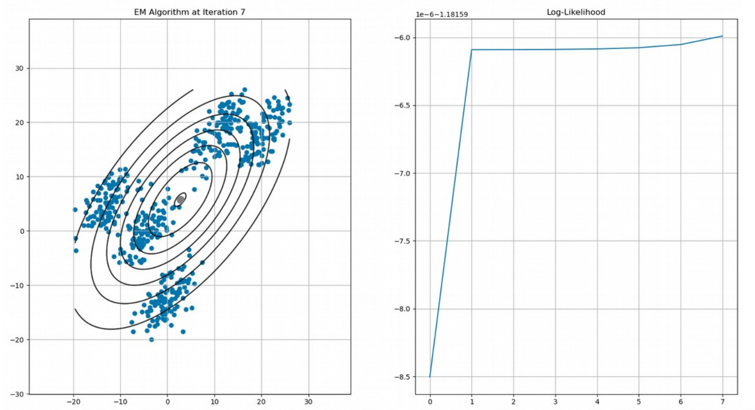


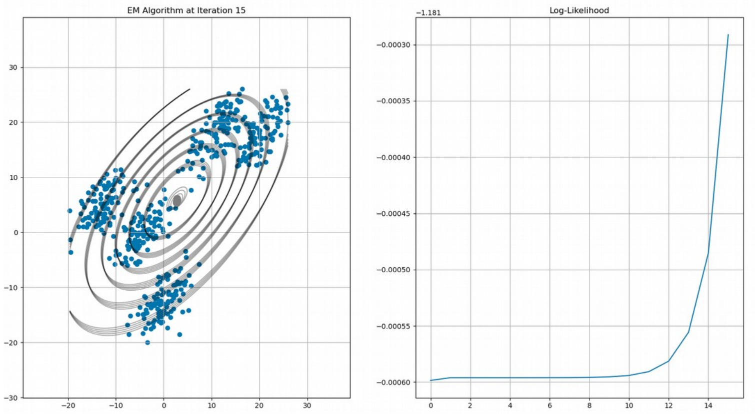


Soft Clustering 기법에서는 하나의 데이터가 일정 비율에 맞춰서 여러 Cluster 에 속할 수 있기에 Cluster 가 서로 중첩될 수 있음 .

EM Algorithm 을 이용한 GMM Clustering 결과 및 분석

6. Blob Dataset – Gaussian Cluster 의 평균점 / 중심점 동일 위치 초기화에 의한 Clustering 결과

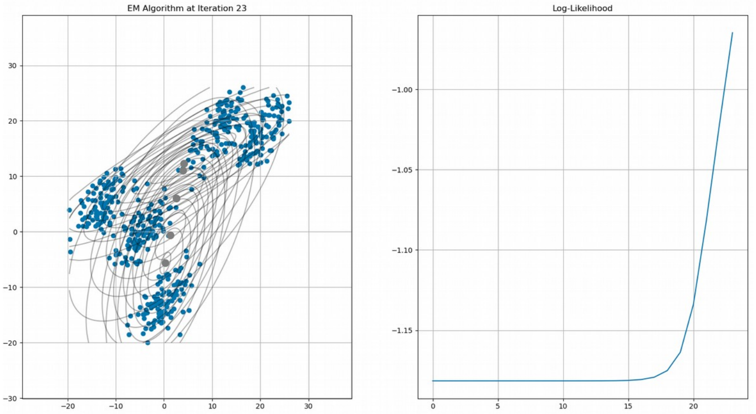


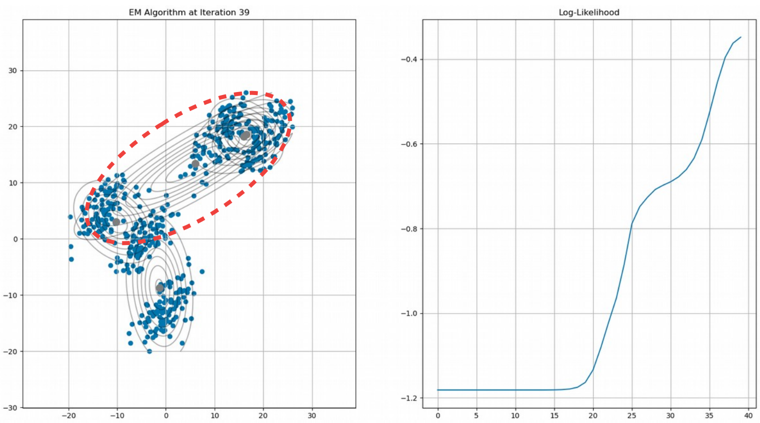


이전 GMM 시나리오 다르게 모든 Gaussian Cluster 를 동일 지점에서 초기화하여 시작하게하였음 .

초반에는 동일 지점에서 시작하게 되어 각 Cluster 의 움직임이 없으나 시간이 지날수록 서로가 주변 데이터 클러스터를 점유하기 위해 멀어지는 것을 볼 수 있음 .

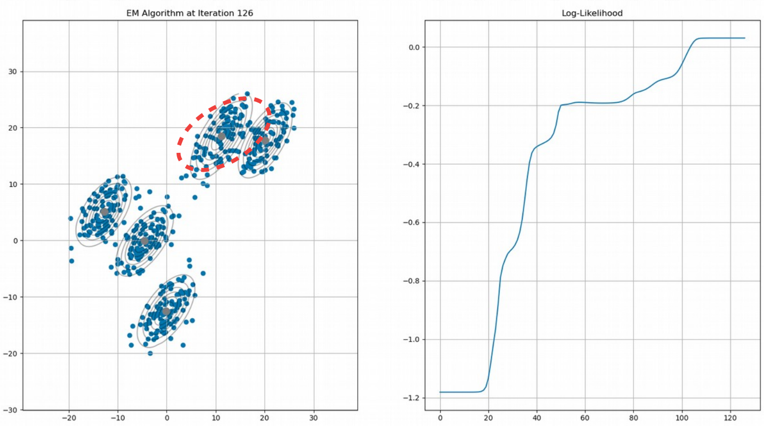
이 때 , 각각의 Gaussian Cluster 는 최대한 주변의 많은 데이터 클러스터를 점유하기 위해 경합을 하게 되고 , 경합 과정에서 주변을 지속 탐색해 나가는 것을 볼 수 있음 .





Iteration 39 에서는 중앙에 데이터 클러스터를 차지하지 못한 빨간색 Gaussian Cluster 가 자신만의 클러스터를 찾기 위해 주변을 탐색함 .

이 때 , 자신만의 클러스터를 보유한 Gaussian Cluster 는 주변 데이터에 대한 점유율이 높기 때문에 빨간색 Gaussian Cluster 가 해당 클러스터 주변을 가지 않는 것을 볼 수 있음 .

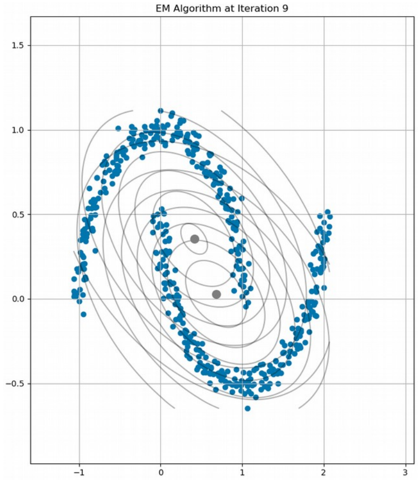


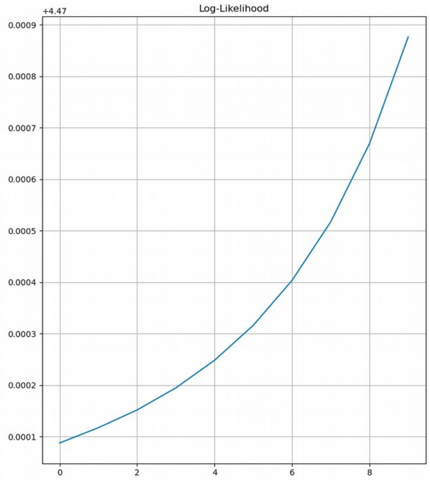
결국 빨간색 Gaussian Cluster 는 비교적 Sparse 하게 펴져있던 상단 Gaussian Cluster 근처로 가서 해당 Cluster 를 밀어내고 그 옆에 있는 데이터 클러스터를 점유하게됨 .

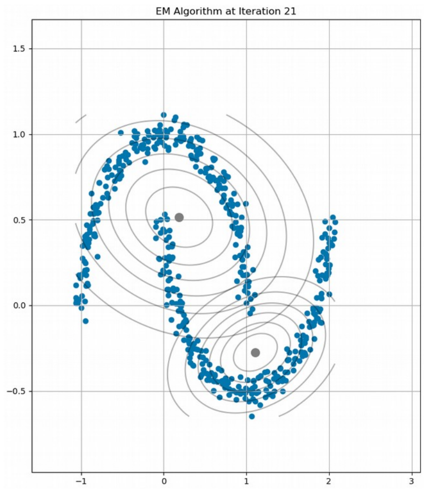
이를 통해 GMM Clustering 이 초기 시작 좌표에 따라 Clustering 성질과 결과가 크게 달라진다는 것을 볼 수 있음 .

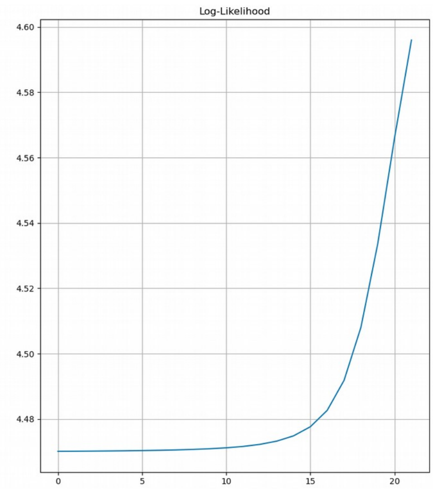
EM Algorithm 을 이용한 GMM Clustering 결과 및 분석

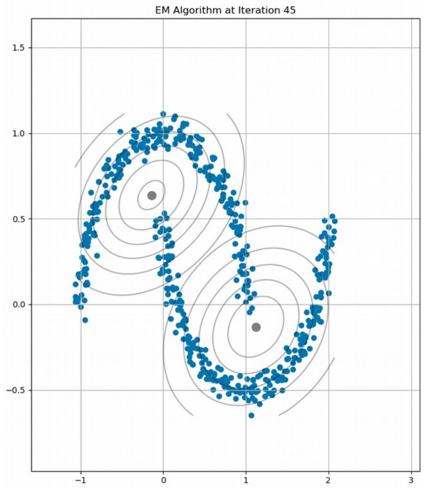
7. Moon Dataset – GMM Clustering 결과 및 GMM 의 한계점

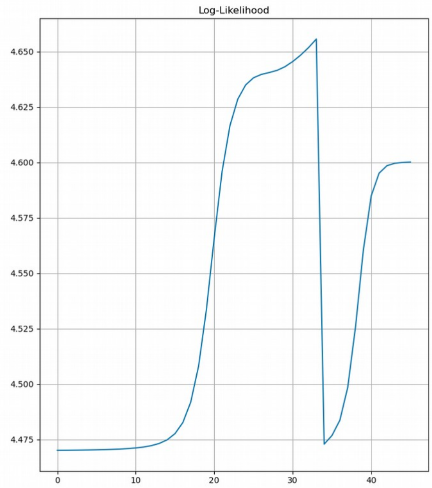


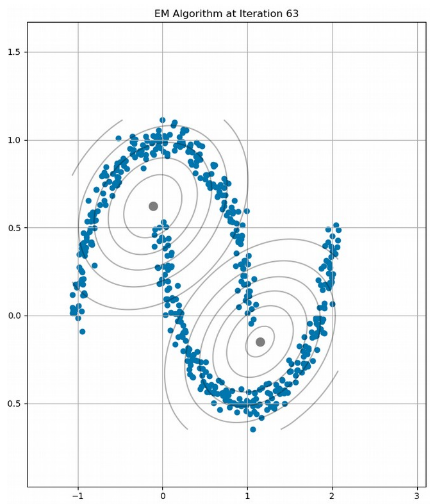


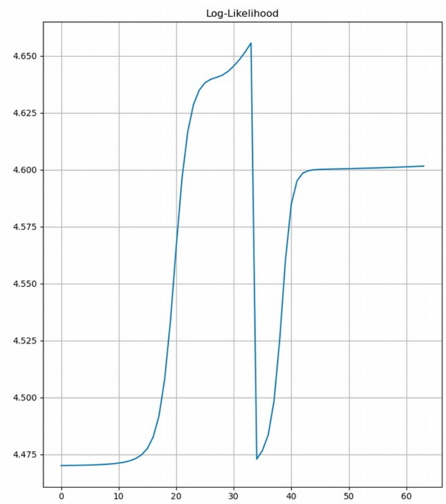












Moon 데이터셋은 Blob 데이터셋과 다르게 원형 구조의 데이터 클러스터가 아닌 달 모양을 하고 있음.

비 원형 / 비 타원형 구조의 데이터셋에서 GMM 은 기본적으로 원형 또는 타원형 구조로 Clustering 을 시도하려고함. 왜냐하면 Gaussian Distribution 이 원형 / 타원형 구조로 생겼기 때문임.

이로 인해 비 원형 / 비 타원형 구조의 데이터셋 환경인 Moon 데이터셋에서 EM 알고리즘을 이용한 GMM 은 Iteration 을 계속 지속하여도 적절한 형태의 Clustering 결과를 산출하지 못하며, 이로 인해 Log Likelihood 가 쉽게 수렴하지 않는 것을 볼 수 있음.

그러므로 GMM 은 비 원형 / 비 타원형 분포의 데이터셋에 대해 Clustering 효율이 급격히 떨어진다는 한계점을 내포하고 있음.