PhamNgocHai_21002139_Week10_Gaussian_Mix_EM_1

Phạm Ngọc Hải

April 29, 2024

1 K-measn model

1.1 Ví dụ 1.

(Data: Khởi tạo 3 tập ngẫu nhiên, mỗi tập N điểm theo phân phối chuẩn Gaussian có 3 kỳ vọng - tâm cụm, và ma trận hiệp phương sai cov)

1.1.1 Khởi tạo data

[ ]:
```
#  Gọi các thư viện cần thiết
# Ta tự xây dựng phần k-means nên sẽ không gọi sklearn

from __future__ import print_function
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial.distance import cdist
np.random.seed(11)

# Kỳ vọng và hiệp phương sai của 3 cụm dữ liệu
means = [[2, 2], [8, 3], [3, 6]]
cov = [[1, 0], [0, 1]]

# Số điểm mỗi cụm dữ liệu
N = 500

# Tạo các cụm dữ liệu qua phân bố chuẩn (Gaussian)
X0 = np.random.multivariate_normal(means[0], cov, N)
X1 = np.random.multivariate_normal(means[1], cov, N)
X2 = np.random.multivariate_normal(means[2], cov, N)
# Tổng hợp dữ liệu từ các cụm
X = np.concatenate((X0, X1, X2), axis = 0)

# Số cụm = 3
K = 3

# Gán nhãn ban đầu cho các cụm, sau đó ta test model và so sánh
original_label = np.asarray([0]*N + [1]*N + [2]*N).T
```

1.1.2 Xây dựng thuật toán bằng Numpy

Xây dựng các hàm thực hiện thuật toán

[ ]:
```
def kmeans_display(X, label):
    K = np.amax(label) + 1
    X0 = X[label == 0, :]
    X1 = X[label == 1, :]
    X2 = X[label == 2, :]

    plt.plot(X0[:, 0], X0[:, 1], 'b^', markersize = 4, alpha = .8)
    plt.plot(X1[:, 0], X1[:, 1], 'go', markersize = 4, alpha = .8)
    plt.plot(X2[:, 0], X2[:, 1], 'rs', markersize = 4, alpha = .8)

    plt.axis('equal')
    plt.plot()
    plt.show()
```

[ ]:
```
def kmeans_init_centers(X, k):
    # randomly pick k rows of X as initial centers
    return X[np.random.choice(X.shape[0], k, replace=False)]
```

[ ]:
```
from scipy.spatial.distance import cdist
def kmeans_assign_labels(X, centers):
    # calculate pairwise distances btw data and centers
    D = cdist(X, centers)
    # return index of the closest center
    return np.argmin(D, axis = 1)
```

[ ]:
```
def kmeans_update_centers(X, labels, K):
    centers = np.zeros((K, X.shape[1]))
    for k in range(K):
        # collect all points assigned to the k-th cluster
        Xk = X[labels == k, :]
        # take average
        centers[k,:] = np.mean(Xk, axis = 0)
    return centers
```

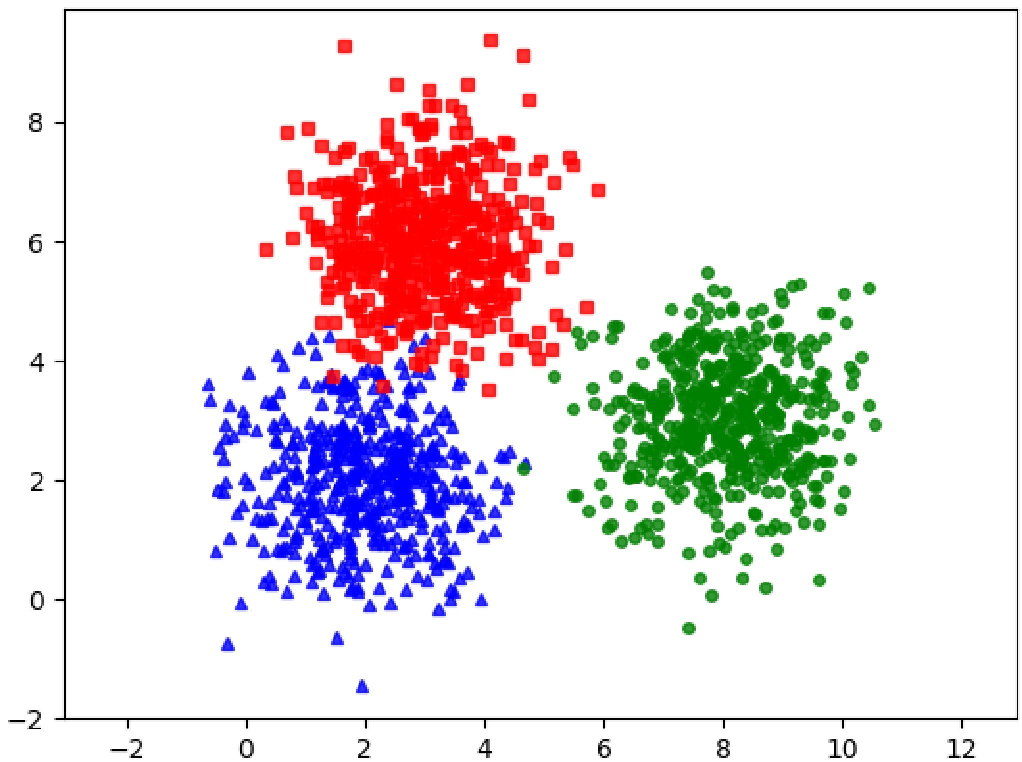
[ ]:
```
def has_converged(centers, new_centers):
    # return True if two sets of centers are the same
    return (set([tuple(a) for a in centers]) == set([tuple(a) for a in␣
 ↪new_centers]))
```

[ ]:
```
def kmeans(X, K):
    centers = [kmeans_init_centers(X, K)]
    labels = []
    it = 0
    while (it < 100000):
```

```
        labels.append(kmeans_assign_labels(X, centers[-1]))
        new_centers = kmeans_update_centers(X, labels[-1], K)
        if has_converged(centers[-1], new_centers):
            break
        centers.append(new_centers)
        it += 1
    return (centers, labels, it)
```

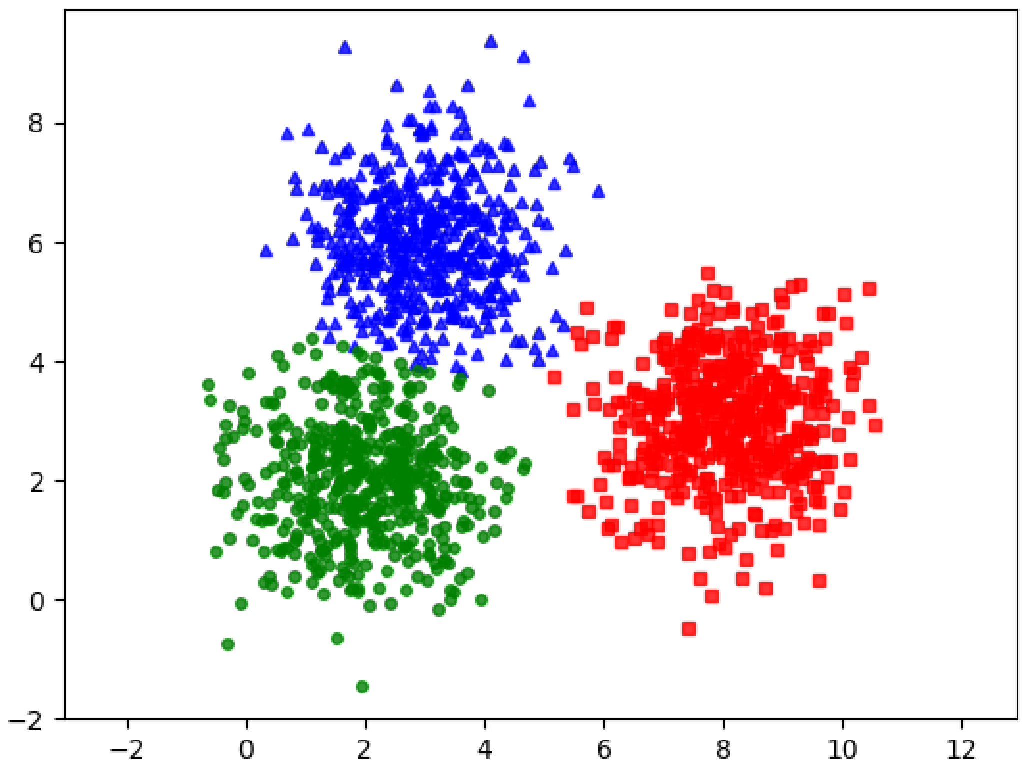
Sử dụng hàm đã xây dựng

[ ]:
```
kmeans_display(X, original_label)
```



[ ]:
```
(centers, labels, it) = kmeans(X, K)
print('Centers found by our algorithm:')
print(centers[-1])

kmeans_display(X, labels[-1])
```

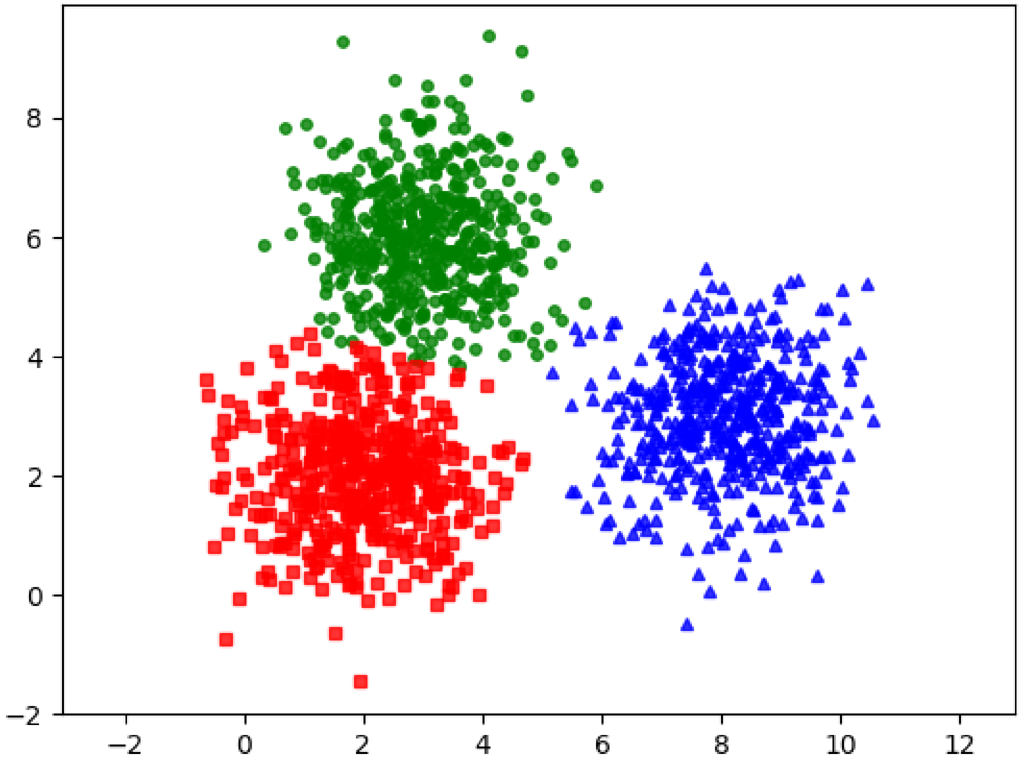
```
Centers found by our algorithm:
[[2.99084705 6.04196062]
 [1.97563391 2.01568065]
 [8.03643517 3.02468432]]
```



1.1.3 Dùng thư viện sklearn

[ ]:
```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3, random_state=0).fit(X)
print('Centers found by scikit-learn:')
print(kmeans.cluster_centers_)
pred_label = kmeans.predict(X)
kmeans_display(X, pred_label)
```

```
Centers found by scikit-learn:
[[8.0410628  3.02094748]
 [2.99357611 6.03605255]
 [1.97634981 2.01123694]]
```



1.2 Ví dụ 2.

(Thực hiện phân cụm cho bộ dữ liệu chữ số viết tay)
- Đọc 500 mẫu từ phần training
- Thực hiện phân cụm k-means
- Kiểm tra trong mỗi cụm, tỷ lệ có nhãn nào là cao nhất
- Đếm và in ra tỷ lệ mẫu không thuộc nhãn đó nhưng được phân vào cùng 1 nhãn

Nội dung code mẫu sẽ sử dụng thư viện, tý sẽ dùng hàm numpy tự xây dựng để thực hành ở bài 2

1.2.1 Xây dựng hàm trực quan hóa dữ liệu

[ ]:
```
# !pip install python-mnist
```

```
Collecting python-mnist
  Downloading python_mnist-0.7-py2.py3-none-any.whl.metadata (3.5 kB)
Downloading python_mnist-0.7-py2.py3-none-any.whl (9.6 kB)
Installing collected packages: python-mnist
Successfully installed python-mnist-0.7
```

[ ]:
```
import numpy as np
from mnist import MNIST
import matplotlib
```

```python
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.neighbors import NearestNeighbors
from sklearn.preprocessing import normalize
```

[ ]:
```python
# This function visualizes filters in matrix A. Each column of A is a
# filter. We will reshape each column into a square image and visualizes
# on each cell of the visualization panel.
# All other parameters are optional, usually you do not need to worry
# about it.
# opt_normalize: whether we need to normalize the filter so that all of
# them can have similar contrast. Default value is true.
# opt_graycolor: whether we use gray as the heat map. Default is true.
# opt_colmajor: you can switch convention to row major for A. In that
# case, each row of A is a filter. Default value is false.
# source: https://github.com/tsaith/ufldl_tutorial


def display_network(A, m=-1, n=-1):
    opt_normalize = True
    opt_graycolor = True

    # Rescale
    A = A - np.average(A)

    # Compute rows & cols
    (row, col) = A.shape
    sz = int(np.ceil(np.sqrt(row)))
    buf = 1
    if m < 0 or n < 0:
        n = np.ceil(np.sqrt(col))
        m = np.ceil(col / n)

    image = np.ones(shape=(buf + m * (sz + buf), buf + n * (sz + buf)))

    if not opt_graycolor:
        image *= 0.1

    k = 0

    for i in range(int(m)):
        for j in range(int(n)):
            if k >= col:
                continue

            clim = np.max(np.abs(A[:, k]))
```

```python
            if opt_normalize:
                image[
                    buf + i * (sz + buf) : buf + i * (sz + buf) + sz,
                    buf + j * (sz + buf) : buf + j * (sz + buf) + sz,
                ] = (
                    A[:, k].reshape(sz, sz) / clim
                )
            else:
                image[
                    buf + i * (sz + buf) : buf + i * (sz + buf) + sz,
                    buf + j * (sz + buf) : buf + j * (sz + buf) + sz,
                ] = A[:, k].reshape(sz, sz) / np.max(np.abs(A))
            k += 1

    return image


def display_color_network(A):
    """
    # display receptive field(s) or basis vector(s) for image patches
    #
    # A the basis, with patches as column vectors
    # In case the midpoint is not set at 0, we shift it dynamically
    :param A:
    :param file:
    :return:
    """
    if np.min(A) >= 0:
        A = A - np.mean(A)

    cols = np.round(np.sqrt(A.shape[1]))

    channel_size = A.shape[0] / 3
    dim = np.sqrt(channel_size)
    dimp = dim + 1
    rows = np.ceil(A.shape[1] / cols)

    B = A[0:channel_size, :]
    C = A[channel_size : 2 * channel_size, :]
    D = A[2 * channel_size : 3 * channel_size, :]

    B = B / np.max(np.abs(B))
    C = C / np.max(np.abs(C))
    D = D / np.max(np.abs(D))

    # Initialization of the image
    image = np.ones(shape=(dim * rows + rows - 1, dim * cols + cols - 1, 3))
```

```
    for i in range(int(rows)):
        for j in range(int(cols)):
            # This sets the patch
            image[i * dimp : i * dimp + dim, j * dimp : j * dimp + dim, 0] = B[
                :, i * cols + j
            ].reshape(dim, dim)
            image[i * dimp : i * dimp + dim, j * dimp : j * dimp + dim, 1] = C[
                :, i * cols + j
            ].reshape(dim, dim)
            image[i * dimp : i * dimp + dim, j * dimp : j * dimp + dim, 2] = D[
                :, i * cols + j
            ].reshape(dim, dim)

    image = (image + 1) / 2

    return image
```

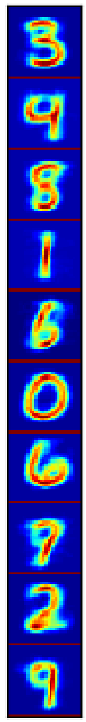
1.2.2 Chạy cho 1000 mẫu đầu tiên

[ ]:
```
mndata = MNIST('/mnt/DataK/Univer/UniSubject/_3th_year/_2nd_term/3ii_ML/Ass/
 ↪Week4/data/number_writing/')
mndata.gz = True
mndata.load_testing()
X = mndata.test_images
X0 = np.asarray(X)[:1000,:]/256.0
X = X0

K = 10
kmeans = KMeans(n_clusters=K).fit(X)

pred_label = kmeans.predict(X)
```

[ ]:
```
print(type(kmeans.cluster_centers_.T))
print(kmeans.cluster_centers_.T.shape)
A = display_network(kmeans.cluster_centers_.T, K, 1)

f1 = plt.imshow(A, interpolation='nearest', cmap = "jet")
f1.axes.get_xaxis().set_visible(False)
f1.axes.get_yaxis().set_visible(False)
plt.show()
# plt.savefig('a1.png', bbox_inches='tight')


# a colormap and a normalization instance
cmap = plt.cm.jet
norm = plt.Normalize(vmin=A.min(), vmax=A.max())
```

```
# map the normalized data to colors
# image is now RGBA (512x512x4)
image = cmap(norm(A))
# import imageio
# imageio.imwrite('number_writing.jpg', image)
```

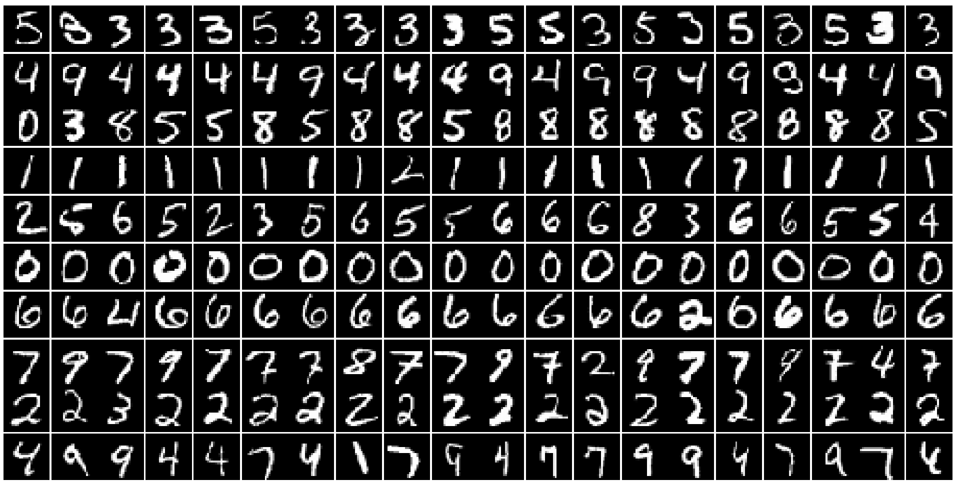
```
<class 'numpy.ndarray'>
(784, 10)
```



[ ]:
```
N0 = 20
X1 = np.zeros((N0 * K, 784))
X2 = np.zeros((N0 * K, 784))

for k in range(K):
    Xk = X0[pred_label == k, :]

    center_k = [kmeans.cluster_centers_[k]]
    # neigh = NearestNeighbors(N0).fit(Xk)
    # dist, nearest_id = neigh.kneighbors(center_k, N0)

    neigh = NearestNeighbors(n_neighbors=N0).fit(Xk)
    dist, nearest_id = neigh.kneighbors(center_k, return_distance=True)
```

```
    X1[N0 * k : N0 * k + N0, :] = Xk[nearest_id[0], :]
    # X1[N0 * k : N0 * k + N0, :] = Xk[nearest_id, :]
    X2[N0 * k : N0 * k + N0, :] = Xk[:N0, :]
```

[ ]:
```
plt.axis('off')
A = display_network(X2.T, K, N0)
f2 = plt.imshow(A, interpolation='nearest' )
plt.gray()
plt.show()
```



1.3 Bài tập tự thực hành

1.3.1 Bài 1

Hãy áp dụng đoạn chương trình chúng ta tự xây dựng trong ví dụ 1, với dữ liệu là hình ảnh chữ số viết tay như trong ví dụ 2, chạy để xem xét kết quả.
(Làm lại ví dụ 2 nhưng bằng hàm tự xây dựng với Numpy)

[ ]:
```
from mnist import MNIST
import numpy as np

mndata = MNIST('/mnt/DataK/Univer/UniSubject/_3th_year/_2nd_term/3ii_ML/Ass/
 ↪Week4/data/number_writing/')
mndata.gz = True
mndata.load_testing()
X = mndata.test_images
X0 = np.asarray(X)[:1000,:]/256.0
X = X0
```

```
K = 10  # 10 chữ số viết tay
(centers, labels, it) = kmeans(X, K)
```

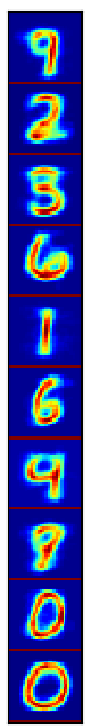
[ ]:
```
print(centers[-1].shape)
```

```
(10, 784)
```

[ ]:
```
print(labels[-1].shape)
```

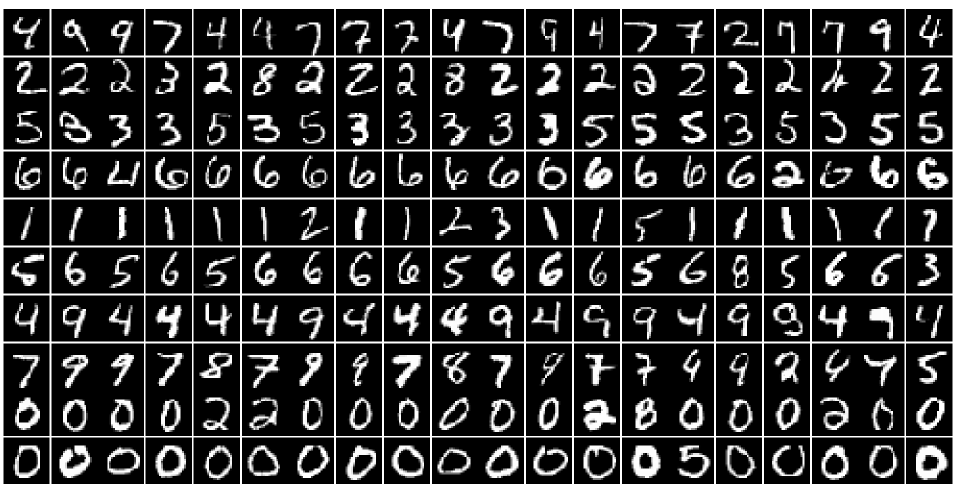
```
(1000,)
```

[ ]:
```
A = display_network(centers[-1].T, K, 1)

f1 = plt.imshow(A, interpolation='nearest', cmap = "jet")
f1.axes.get_xaxis().set_visible(False)
f1.axes.get_yaxis().set_visible(False)
plt.show()

# a colormap and a normalization instance
cmap = plt.cm.jet
norm = plt.Normalize(vmin=A.min(), vmax=A.max())

# map the normalized data to colors
# image is now RGBA (512x512x4)
image = cmap(norm(A))
```



```
NO = 20
X1 = np.zeros((NO * K, 784))
X2 = np.zeros((NO * K, 784))

for k in range(K):
    Xk = X0[labels[-1] == k, :]

    center_k = [centers[-1][k]]
    # neigh = NearestNeighbors(NO).fit(Xk)
    # dist, nearest_id = neigh.kneighbors(center_k, NO)

    neigh = NearestNeighbors(n_neighbors=NO).fit(Xk)
    dist, nearest_id = neigh.kneighbors(center_k, return_distance=True)

    X1[NO * k : NO * k + NO, :] = Xk[nearest_id[0], :]
    # X1[NO * k : NO * k + NO, :] = Xk[nearest_id, :]
    X2[NO * k : NO * k + NO, :] = Xk[:NO, :]
```

```
plt.axis("off")
A = display_network(X2.T, K, NO)
f2 = plt.imshow(A, interpolation="nearest")
plt.gray()
plt.show()
```



1.3.2 Bài 2

Áp dụng mô hình trên cho bài tập phân loại ảnh chó-mèo (xem lại phần CNN), thử thực hiện phân cụm thành 02 cụm và kiểm tra kết quả.

Load ảnh chó mèo

[ ]:
```
import os

base_dir = "/mnt/DataK/Univer/UniSubject/_3th_year/_2nd_term/3ii_ML/Ass/Week8/
↪data/cat_dog_panda"
# Change the base_dir to where you put dataset
print("Contents of base directory:")
print(os.listdir(base_dir))

print("\nContents of train directory:")
print(os.listdir(f"{base_dir}/train"))

print("\nContents of validation directory:")
print(os.listdir(f"{base_dir}/validation"))

train_dir = os.path.join(base_dir, "train")
validation_dir = os.path.join(base_dir, "validation")

# Directory with training cat/dog pictures
train_cats_dir = os.path.join(train_dir, "cats")
train_dogs_dir = os.path.join(train_dir, "dogs")

# Directory with validation cat/dog pictures
validation_cats_dir = os.path.join(validation_dir, "cats")
validation_dogs_dir = os.path.join(validation_dir, "dogs")

print("\nContents of train directory:")
print(os.listdir(f"{base_dir}/train"))

print("\nContents of validation directory:")
print(os.listdir(f"{base_dir}/validation"))

train_cat_fnames = os.listdir(train_cats_dir)
train_dog_fnames = os.listdir(train_dogs_dir)

print(train_cat_fnames[:10])
print(train_dog_fnames[:10])

print("total training cat images :", len(os.listdir(train_cats_dir)))
print("total training dog images :", len(os.listdir(train_dogs_dir)))

print("total validation cat images :", len(os.listdir(validation_cats_dir)))
print("total validation dog images :", len(os.listdir(validation_dogs_dir)))
```

```
Contents of base directory:
['train', 'validation']

Contents of train directory:
```

```
['cats', 'dogs', 'panda']

Contents of validation directory:
['cats', 'dogs', 'panda']

Contents of train directory:
['cats', 'dogs', 'panda']

Contents of validation directory:
['cats', 'dogs', 'panda']
['cats_00306.jpg', 'cats_00612.jpg', 'cats_00001.jpg', 'cats_00002.jpg',
'cats_00003.jpg', 'cats_00004.jpg', 'cats_00005.jpg', 'cats_00006.jpg',
'cats_00007.jpg', 'cats_00008.jpg']
['dogs_00306.jpg', 'dogs_00612.jpg', 'dogs_00001.jpg', 'dogs_00002.jpg',
'dogs_00003.jpg', 'dogs_00004.jpg', 'dogs_00005.jpg', 'dogs_00006.jpg',
'dogs_00007.jpg', 'dogs_00008.jpg']
total training cat images : 1000
total training dog images : 1000
total validation cat images : 1000
total validation dog images : 1000
```

[ ]:
```
import os
import numpy as np
import matplotlib.pyplot as plt
import cv2
from sklearn.cluster import KMeans
from sklearn import metrics

# Function to load and preprocess images
def load_and_preprocess_images(directory):
    images = []
    for filename in os.listdir(directory):
        img = cv2.imread(os.path.join(directory, filename))
        img = cv2.resize(img, (100, 100))  # Resize image to 100x100
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # Convert BGR to RGB
        images.append(img)
    return np.array(images)

# Load and preprocess training images
train_cats_dir = '/mnt/DataK/Univer/UniSubject/_3th_year/_2nd_term/3ii_ML/Ass/
↪Week8/data/cat_dog_panda/train/cats'
train_dogs_dir = '/mnt/DataK/Univer/UniSubject/_3th_year/_2nd_term/3ii_ML/Ass/
↪Week8/data/cat_dog_panda/train/dogs'
train_cat_images = load_and_preprocess_images(train_cats_dir)
train_cat_images = load_and_preprocess_images(train_cats_dir)
train_dog_images = load_and_preprocess_images(train_dogs_dir)
```

```
# Combine cat and dog images
X_train = np.concatenate([train_cat_images, train_dog_images])
y_train = np.array([0]*len(train_cat_images) + [1]*len(train_dog_images))

# Reshape images into a 1D array
X_train_flat = X_train.reshape(X_train.shape[0], -1)
```

Thực hiện phân cụm K-means

[ ]:
```
# Use K-means to cluster images
kmeans = KMeans(n_clusters=2, random_state=42)
kmeans.fit(X_train_flat)
```

[ ]:
```
KMeans(n_clusters=2, random_state=42)
```

Đánh giá kết quả

Theo Accuracy Mục đích: đánh giá dự đoán của phân cụm so với nhãn thực tế đúng đến bao nhiêu

[ ]:
```
# Predicted labels
predicted_labels = kmeans.labels_

# Convert cluster labels to match with true labels
predicted_labels[predicted_labels == 0] = np.argmax(np.
 ↪bincount(y_train[predicted_labels == 0]))
predicted_labels[predicted_labels == 1] = np.argmax(np.
 ↪bincount(y_train[predicted_labels == 1]))

# Calculate accuracy
accuracy = np.mean(predicted_labels == y_train)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.542
```

Theo Adjusted Rand Index Mục đích: đánh giá độ tương đồng giữa hai phân cụm, trong trường hợp này là giữa các nhãn cụm dự đoán và nhãn thực tế. Chỉ số này có giá trị trong khoảng [-1, 1], với 1 cho biết hai phân cụm hoàn toàn giống nhau, 0 cho biết hai phân cụm không tốt hơn so với phân phối ngẫu nhiên của các nhãn, và -1 cho biết sự không tương quan giữa hai phân cụm

[ ]:
```
# Predicted labels
predicted_labels = kmeans.labels_

# Cluster centers
centers = kmeans.cluster_centers_

# Evaluation
```

```
print("Adjusted Rand Index:", metrics.adjusted_rand_score(y_train,␣
 ↪predicted_labels))
```

```
Adjusted Rand Index: 0.0065634244272180986
```

2 Gaussian Mixture model

2.1 Triển khai thuật toán EM dùng Numpy

2.1.1 Code mẫu

Dữ liệu

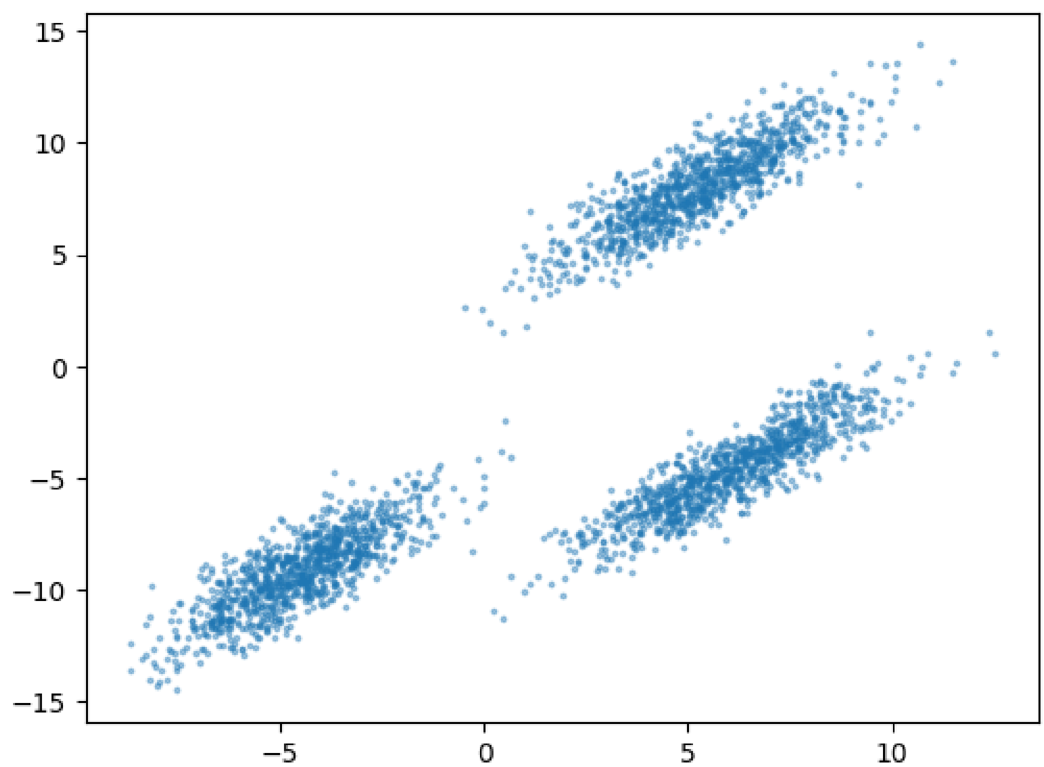
Hàm đọc và trực quan hóa dữ liệu

```
[ ]: import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal


def gen_data(k=3, dim=2, points_per_cluster=200, lim=[-10, 10]):
    """
    Generates data from a random mixture of Gaussians in a given range.
    Will also plot the points in case of 2D.
    input:
        - k: Number of Gaussian clusters
        - dim: Dimension of generated points
        - points_per_cluster: Number of points to be generated for each cluster
        - lim: Range of mean values
    output:
        - X: Generated points (points_per_cluster*k, dim)
    """
    x = []
    mean = random.rand(k, dim) * (lim[1] - lim[0]) + lim[0]
    for i in range(k):
        cov = random.rand(dim, dim + 10)
        cov = np.matmul(cov, cov.T)
        _x = np.random.multivariate_normal(mean[i], cov, points_per_cluster)
        x += list(_x)
    x = np.array(x)
    if dim == 2:
        fig = plt.figure()
        ax = fig.gca()
        ax.scatter(x[:, 0], x[:, 1], s=3, alpha=0.4)
        ax.autoscale(enable=True)
    return x


def plot(title):
```

```
    '''
    Draw the data points and the fitted mixture model.
    input:
        - title: title of plot and name with which it will be saved.
    '''
    fig = plt.figure(figsize=(8, 8))
    ax = fig.gca()
    ax.scatter(X[:, 0], X[:, 1], s=3, alpha=0.4)
    ax.scatter(gmm.mu[:, 0], gmm.mu[:, 1], c=gmm.colors)
    gmm.draw(ax, lw=3)
    ax.set_xlim((-22, 22))
    ax.set_ylim((-22, 22))

    plt.title(title)
    # plt.savefig(title.replace(':', '_'))
    plt.show()
    plt.clf()
```

Thực hiện việc đọc và trực quan hóa dữ liệu

[ ]:
```
# Generate random 2D data with 3 clusters
X = gen_data(k=3, dim=2, points_per_cluster=1000)
```



Model

Định nghĩa class dùng build GMM model

[ ]:
```
import numpy as np
from numpy import random
from matplotlib.patches import Ellipse
import matplotlib.transforms as transforms
from scipy.stats import multivariate_normal

class GMM():
    def __init__(self, k, dim, init_mu=None, init_sigma=None, init_pi=None,␣
↪colors=None):
        '''
        Define a model with known number of clusters and dimensions.
        input:
            - k: Number of Gaussian clusters
            - dim: Dimension
            - init_mu: initial value of mean of clusters (k, dim)
                       (default) random from uniform[-10, 10]
            - init_sigma: initial value of covariance matrix of clusters (k,␣
↪dim, dim)
                          (default) Identity matrix for each cluster
            - init_pi: initial value of cluster weights (k,)
                       (default) equal value to all cluster i.e. 1/k
            - colors: Color valu for plotting each cluster (k, 3)
                      (default) random from uniform[0, 1]
        '''
        self.k = k
        self.dim = dim
        if(init_mu is None):
            init_mu = random.rand(k, dim)*20 - 10
        self.mu = init_mu
        if(init_sigma is None):
            init_sigma = np.zeros((k, dim, dim))
            for i in range(k):
                init_sigma[i] = np.eye(dim)
        self.sigma = init_sigma
        if(init_pi is None):
            init_pi = np.ones(self.k)/self.k
        self.pi = init_pi
        if(colors is None):
            colors = random.rand(k, 3)
            for i in range(k):
                colors[i, 2] = i/k
        self.colors = colors

    def init_em(self, X):
```

```
        '''
        Initialization for EM algorithm.
        input:
            - X: data (batch_size, dim)
        '''
        self.data = X
        self.num_points = X.shape[0]
        self.z = np.zeros((self.num_points, self.k))

    def e_step(self):
        '''
        E-step of EM algorithm.
        '''
        for i in range(self.k):
            self.z[:, i] = self.pi[i] * multivariate_normal.pdf(self.data,␣
↪mean=self.mu[i], cov=self.sigma[i])
        self.z /= self.z.sum(axis=1, keepdims=True)

    def m_step(self):
        '''
        M-step of EM algorithm.
        '''
        sum_z = self.z.sum(axis=0)
        self.pi = sum_z / self.num_points
        self.mu = np.matmul(self.z.T, self.data)
        self.mu /= sum_z[:, None]
        for i in range(self.k):
            j = np.expand_dims(self.data, axis=1) - self.mu[i]
            s = np.matmul(j.transpose([0, 2, 1]), j)
            self.sigma[i] = np.matmul(s.transpose(1, 2, 0), self.z[:, i] )
            self.sigma[i] /= sum_z[i]

    def log_likelihood(self, X):
        '''
        Compute the log-likelihood of X under current parameters
        input:
            - X: Data (batch_size, dim)
        output:
            - log-likelihood of X: Sum_n Sum_k log(pi_k * N( X_n | mu_k,␣
↪sigma_k ))
        '''
        ll = []
        for d in X:
            tot = 0
            for i in range(self.k):
                tot += self.pi[i] * multivariate_normal.pdf(d, mean=self.mu[i],␣
↪cov=self.sigma[i])
```

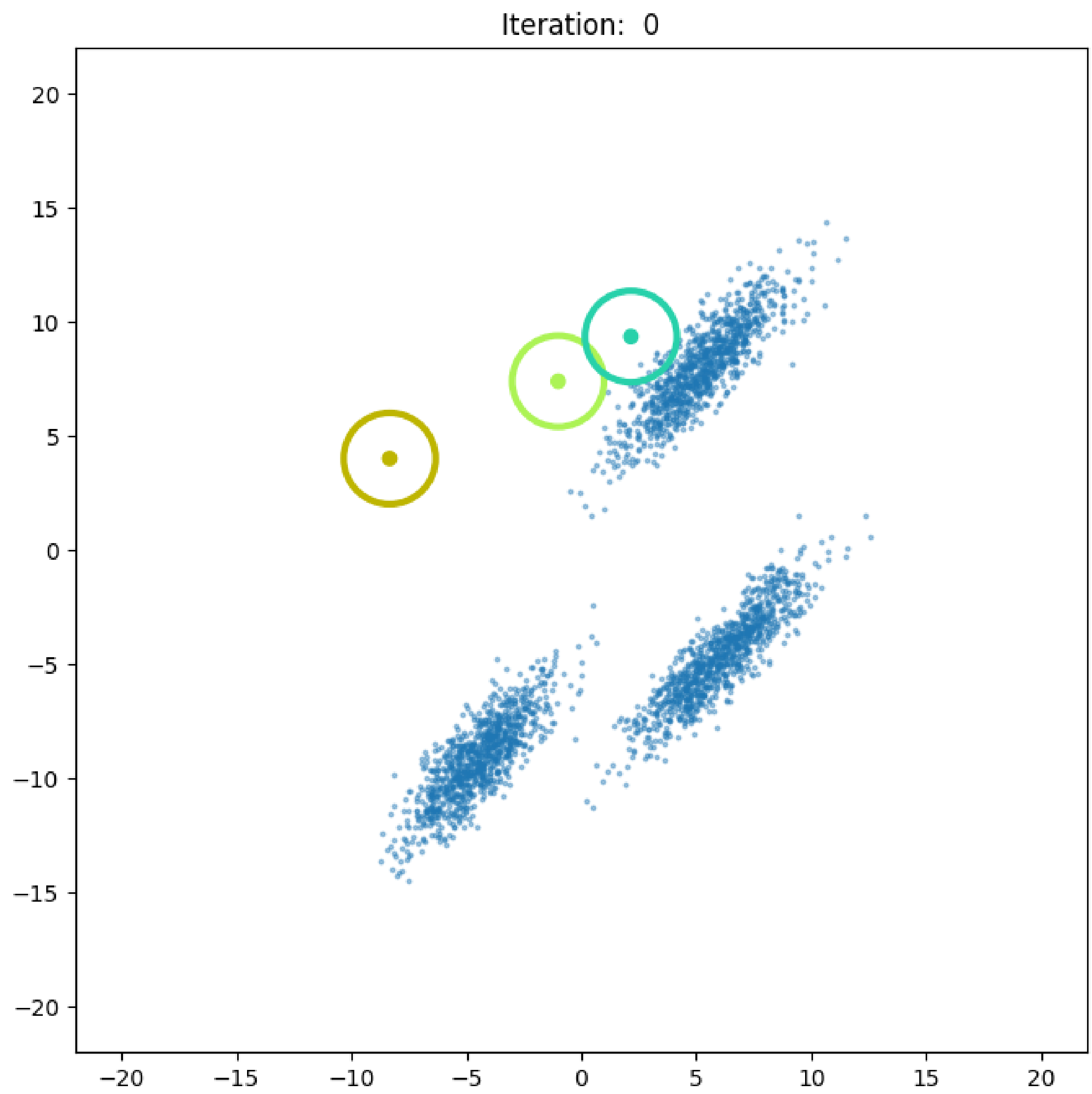
```
            ll.append(np.log(tot))
        return np.sum(ll)

    def plot_gaussian(self, mean, cov, ax, n_std=3.0, facecolor='none',␣
↪**kwargs):
        '''
        Utility function to plot one Gaussian from mean and covariance.
        '''
        pearson = cov[0, 1]/np.sqrt(cov[0, 0] * cov[1, 1])
        ell_radius_x = np.sqrt(1 + pearson)
        ell_radius_y = np.sqrt(1 - pearson)
        ellipse = Ellipse((0, 0),
            width=ell_radius_x * 2,
            height=ell_radius_y * 2,
            facecolor=facecolor,
            **kwargs)
        scale_x = np.sqrt(cov[0, 0]) * n_std
        mean_x = mean[0]
        scale_y = np.sqrt(cov[1, 1]) * n_std
        mean_y = mean[1]
        transf = transforms.Affine2D() \
            .rotate_deg(45) \
            .scale(scale_x, scale_y) \
            .translate(mean_x, mean_y)
        ellipse.set_transform(transf + ax.transData)
        return ax.add_patch(ellipse)

    def draw(self, ax, n_std=2.0, facecolor='none', **kwargs):
        '''
        Function to draw the Gaussians.
        Note: Only for two-dimensionl dataset
        '''
        if(self.dim != 2):
            print("Drawing available only for 2D case.")
            return
        for i in range(self.k):
            self.plot_gaussian(self.mu[i], self.sigma[i], ax, n_std=n_std,␣
↪edgecolor=self.colors[i], **kwargs)
```

Xây dựng GMM model

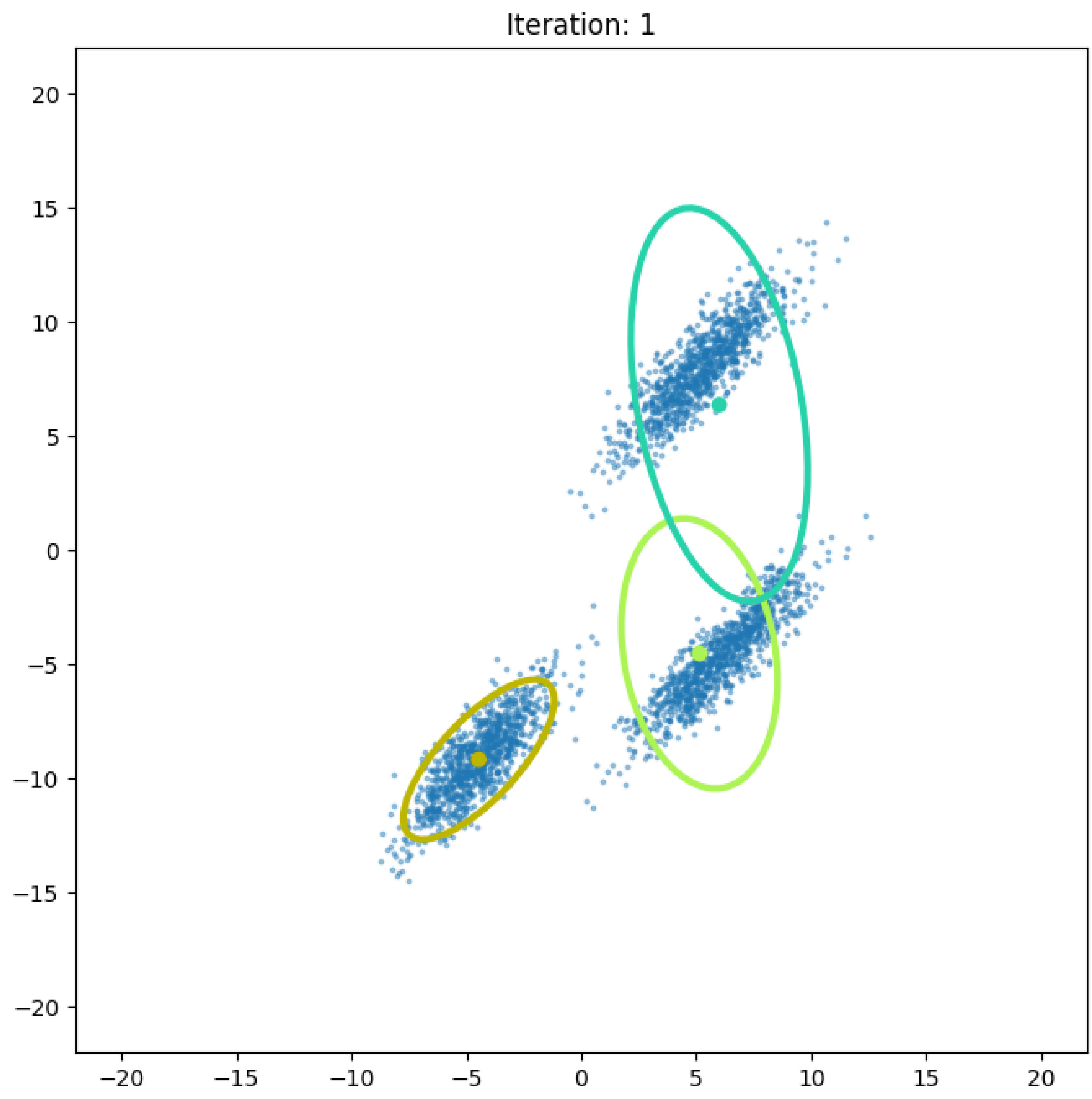
[ ]:
```
# Create a Gaussian Mixture Model
gmm = GMM(3, 2)
```

[ ]:
```
# Training the GMM using EM

# Initialize EM algo with data
gmm.init_em(X)
```

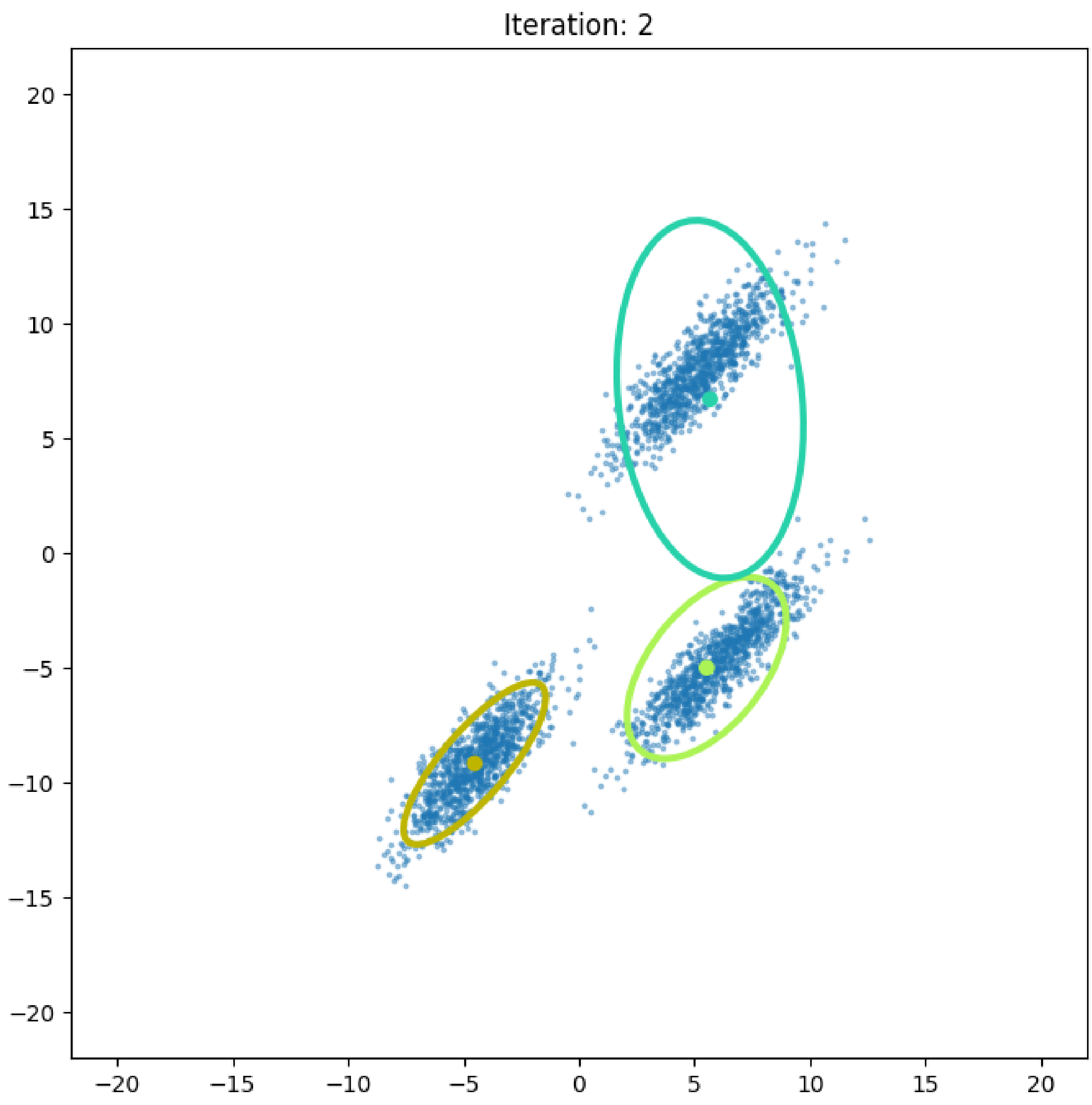
```
num_iters = 20
# Saving log-likelihood
log_likelihood = [gmm.log_likelihood(X)]
# plotting
plot("Iteration:  0")
for e in range(num_iters):
    # E-step
    gmm.e_step()
    # M-step
    gmm.m_step()
    # Computing log-likelihood
    log_likelihood.append(gmm.log_likelihood(X))
    print("Iteration: {}, log-likelihood: {:.4f}".format(e+1,␣
 ↪log_likelihood[-1]))
    # plotting
    plot(title="Iteration: " + str(e+1))
```



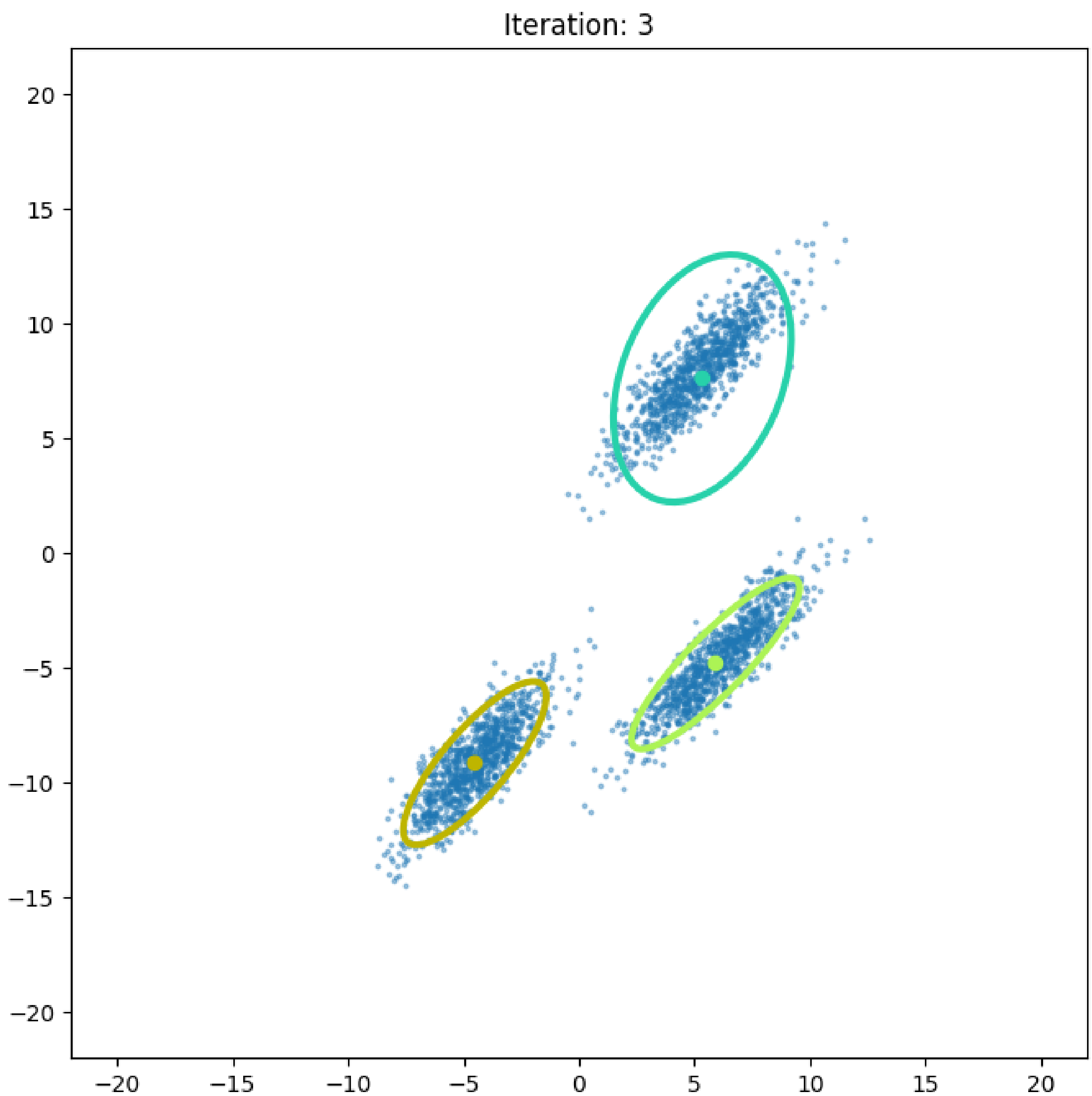

```
Iteration: 1, log-likelihood: -15702.0323
<Figure size 640x480 with 0 Axes>
```



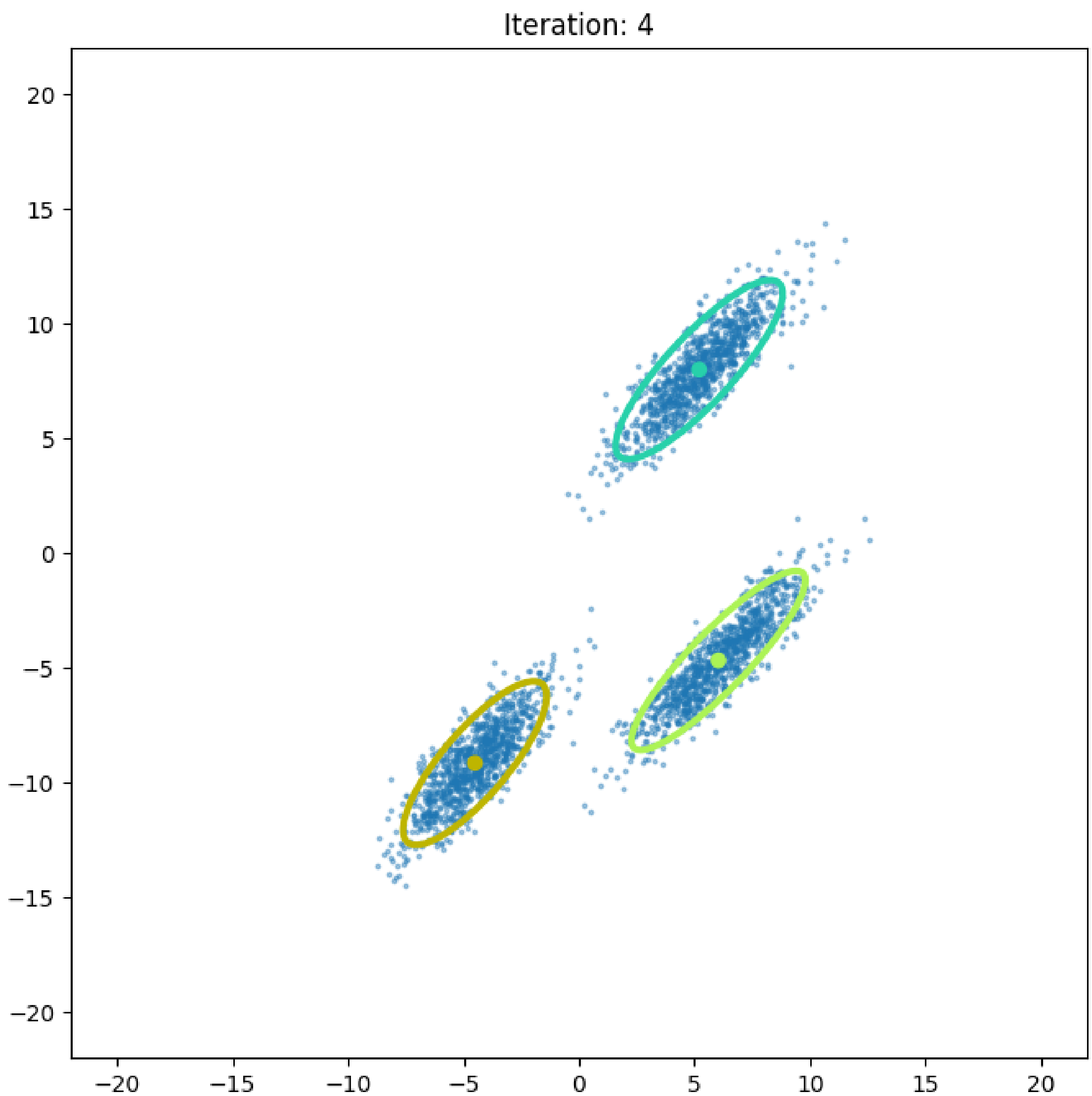

```
Iteration: 2, log-likelihood: -14872.1328
<Figure size 640x480 with 0 Axes>
```



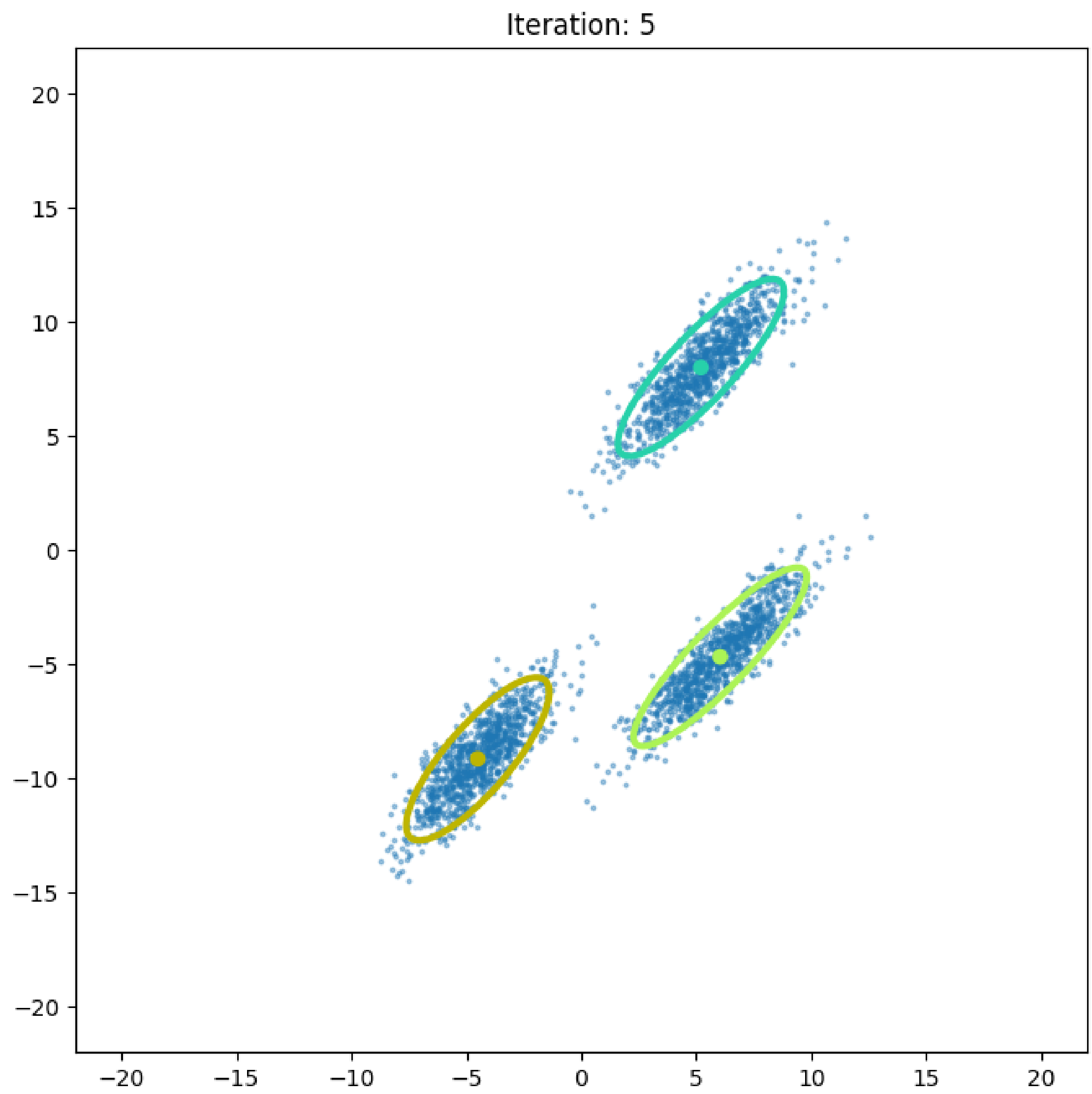

```
Iteration: 3, log-likelihood: -13973.2406
<Figure size 640x480 with 0 Axes>
```



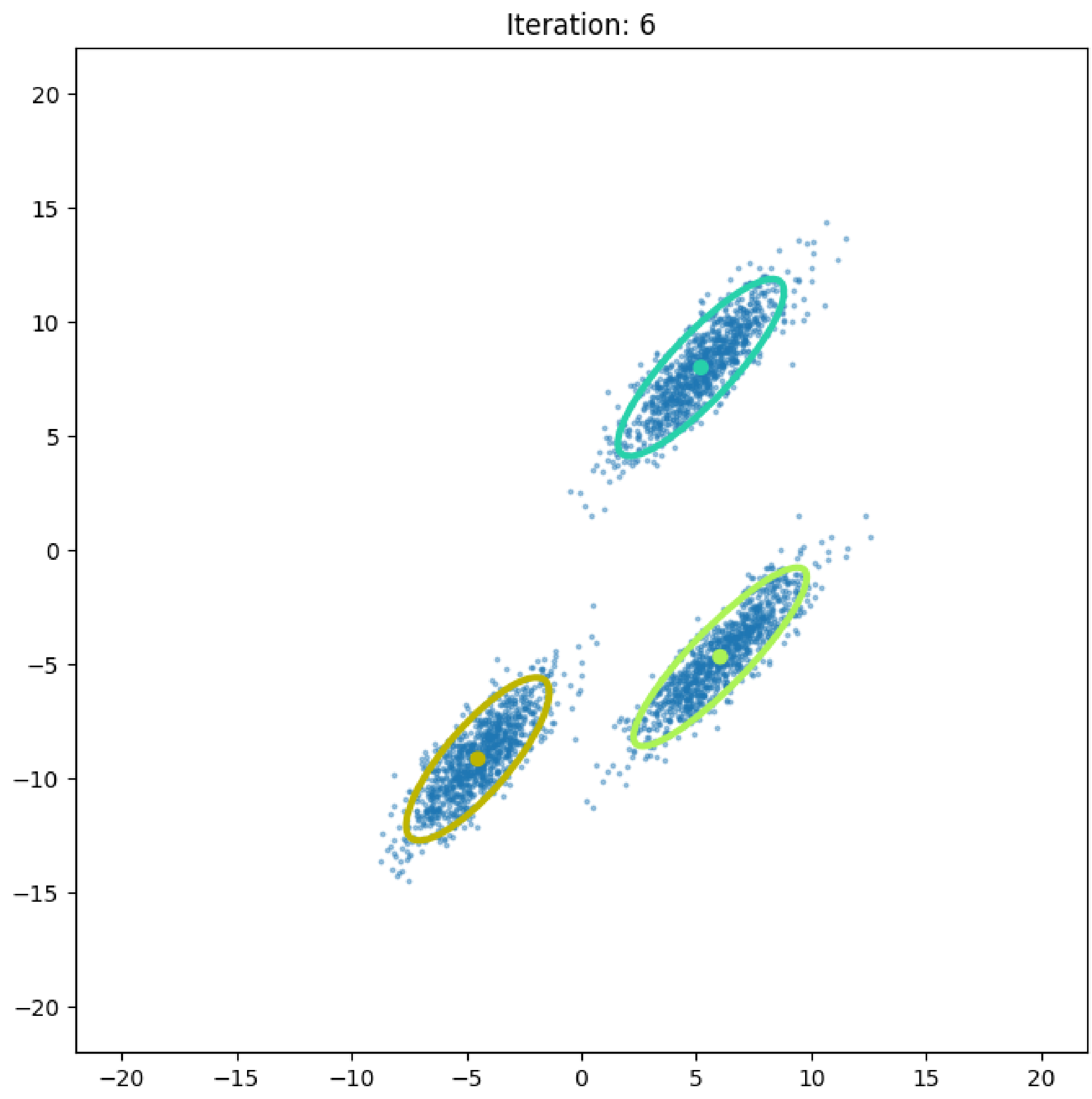

```
Iteration: 4, log-likelihood: -13373.0102
<Figure size 640x480 with 0 Axes>
```



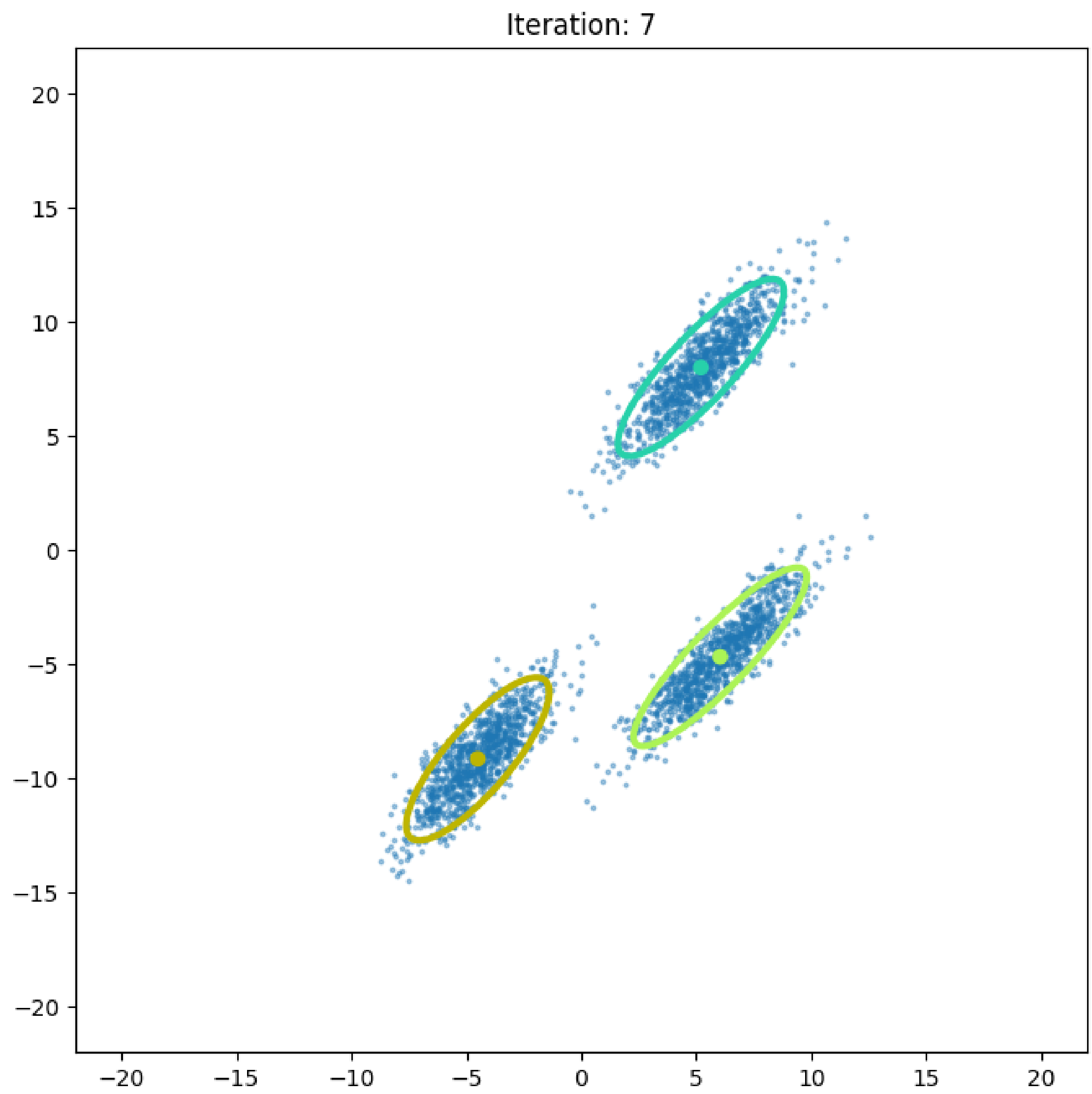

```
Iteration: 5, log-likelihood: -13371.6950
<Figure size 640x480 with 0 Axes>
```



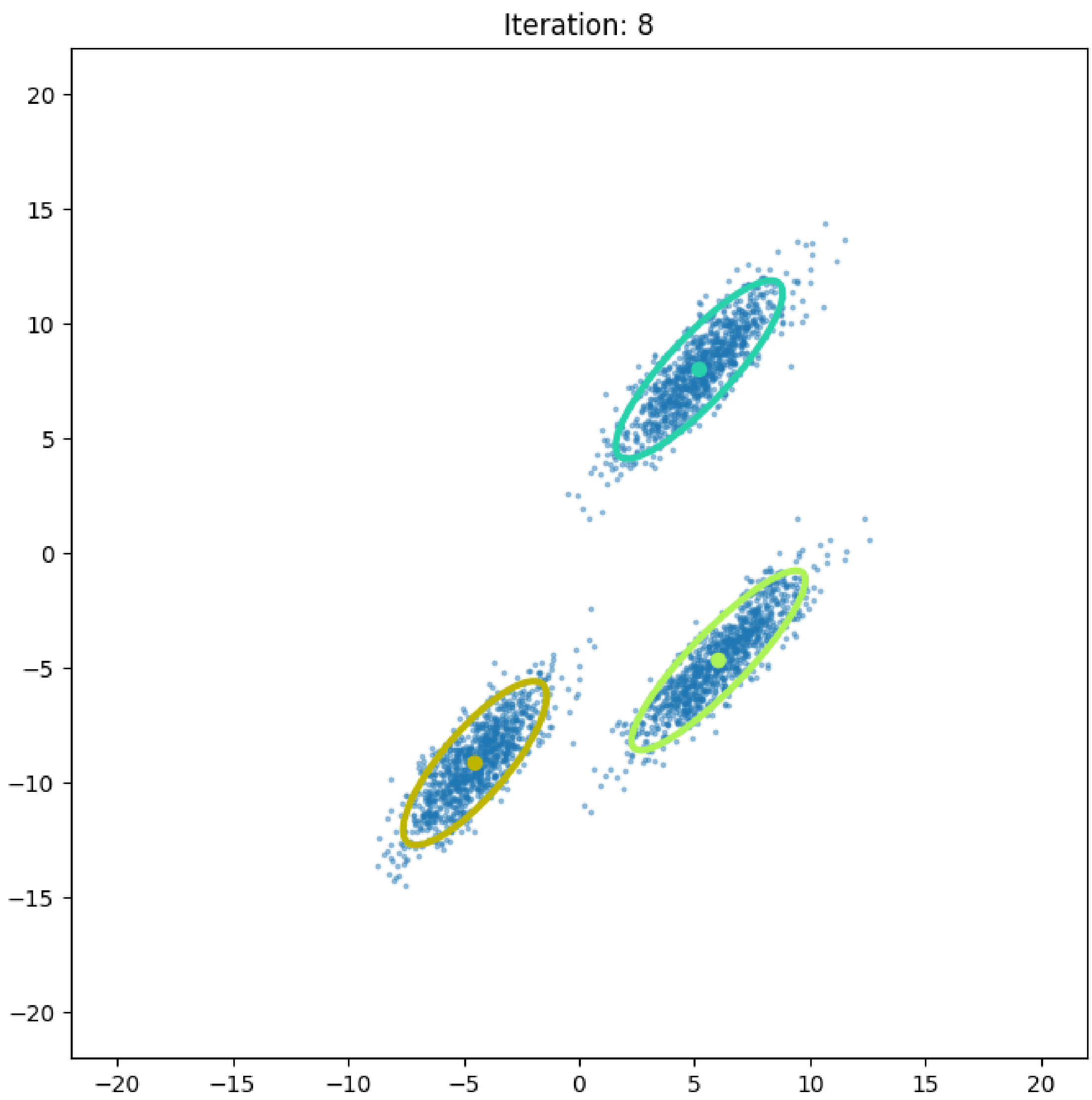

```
Iteration: 6, log-likelihood: -13371.6950
<Figure size 640x480 with 0 Axes>
```



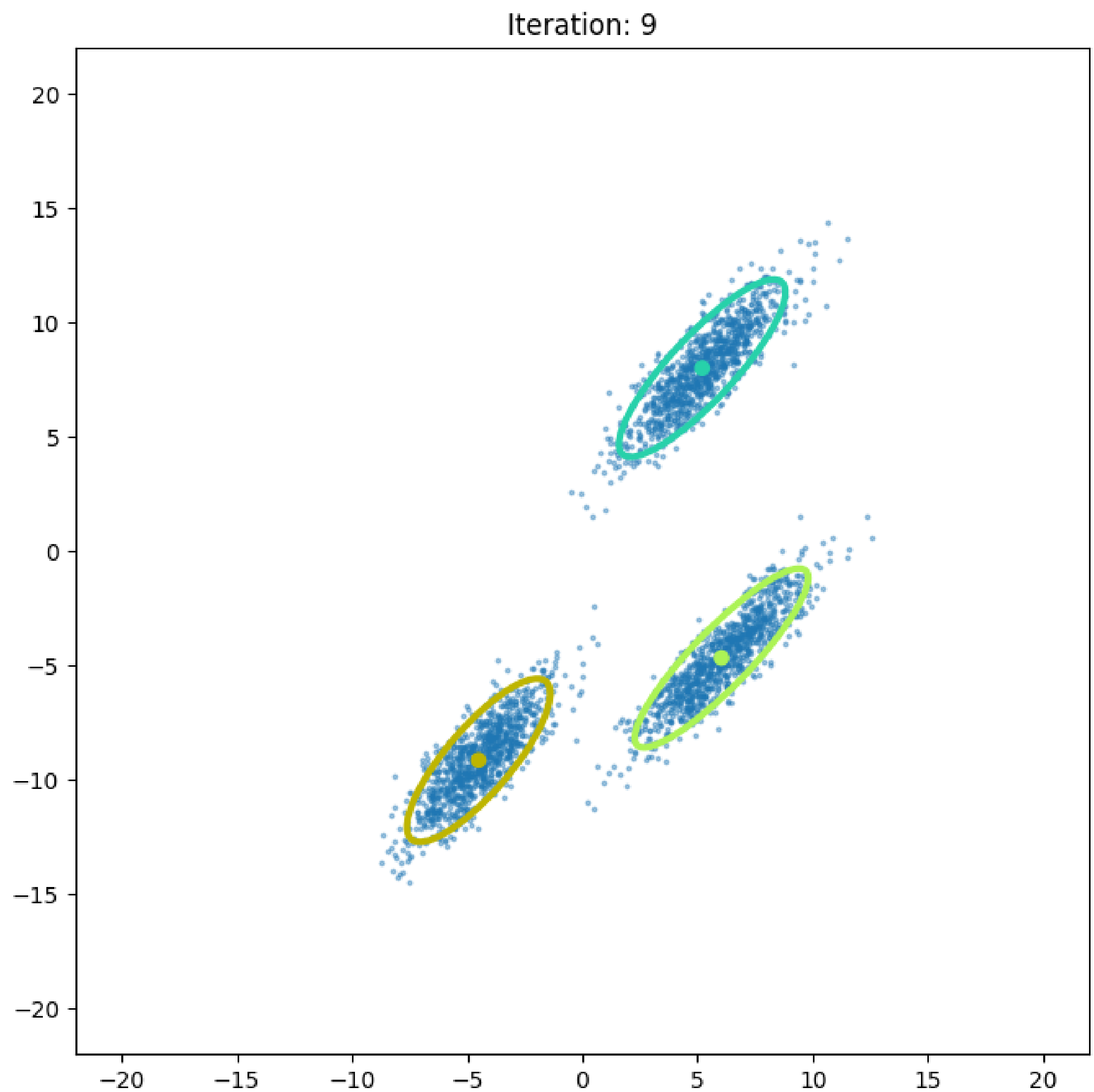

```
Iteration: 7, log-likelihood: -13371.6950
<Figure size 640x480 with 0 Axes>
```



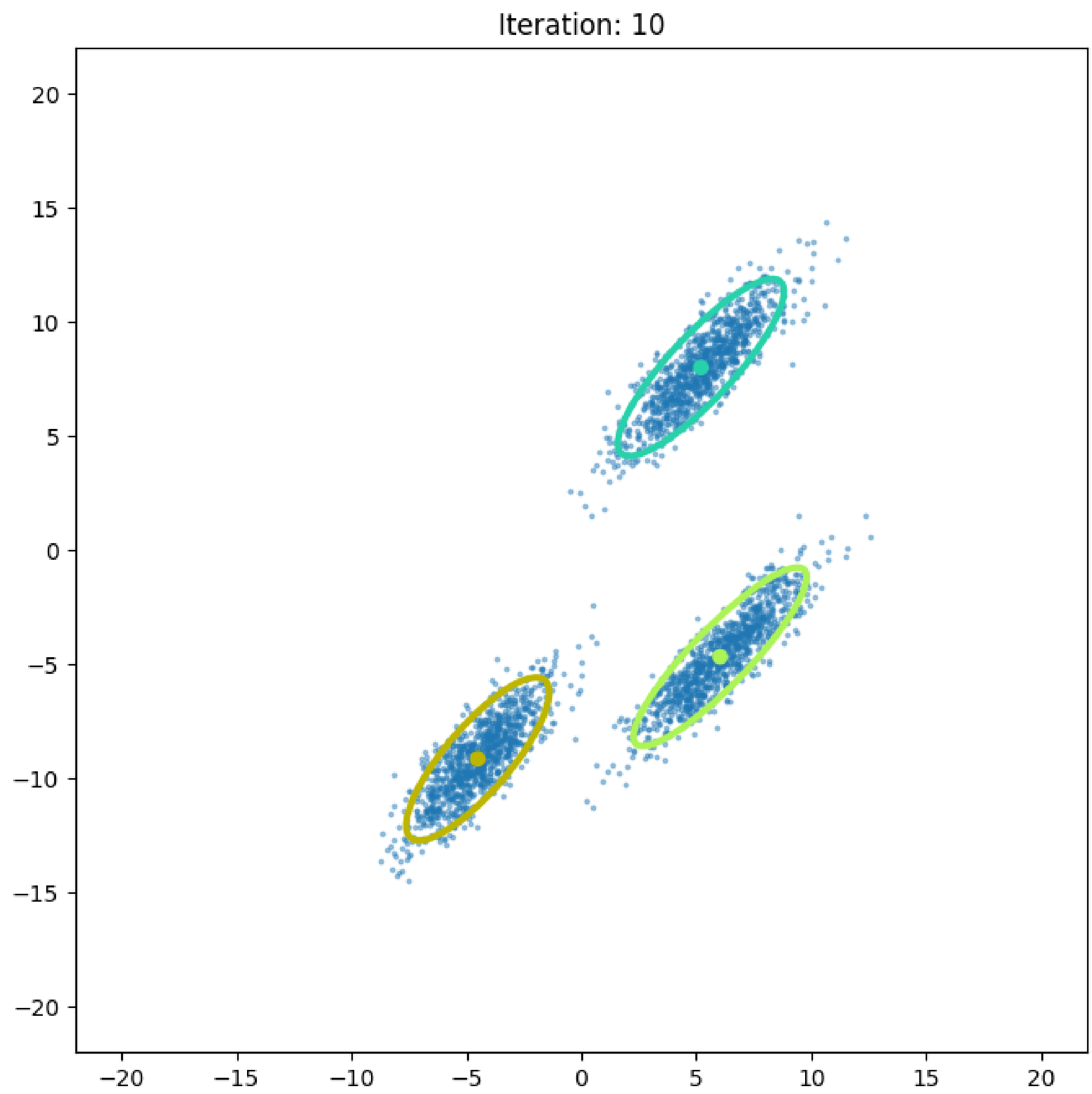

```
Iteration: 8, log-likelihood: -13371.6950
<Figure size 640x480 with 0 Axes>
```



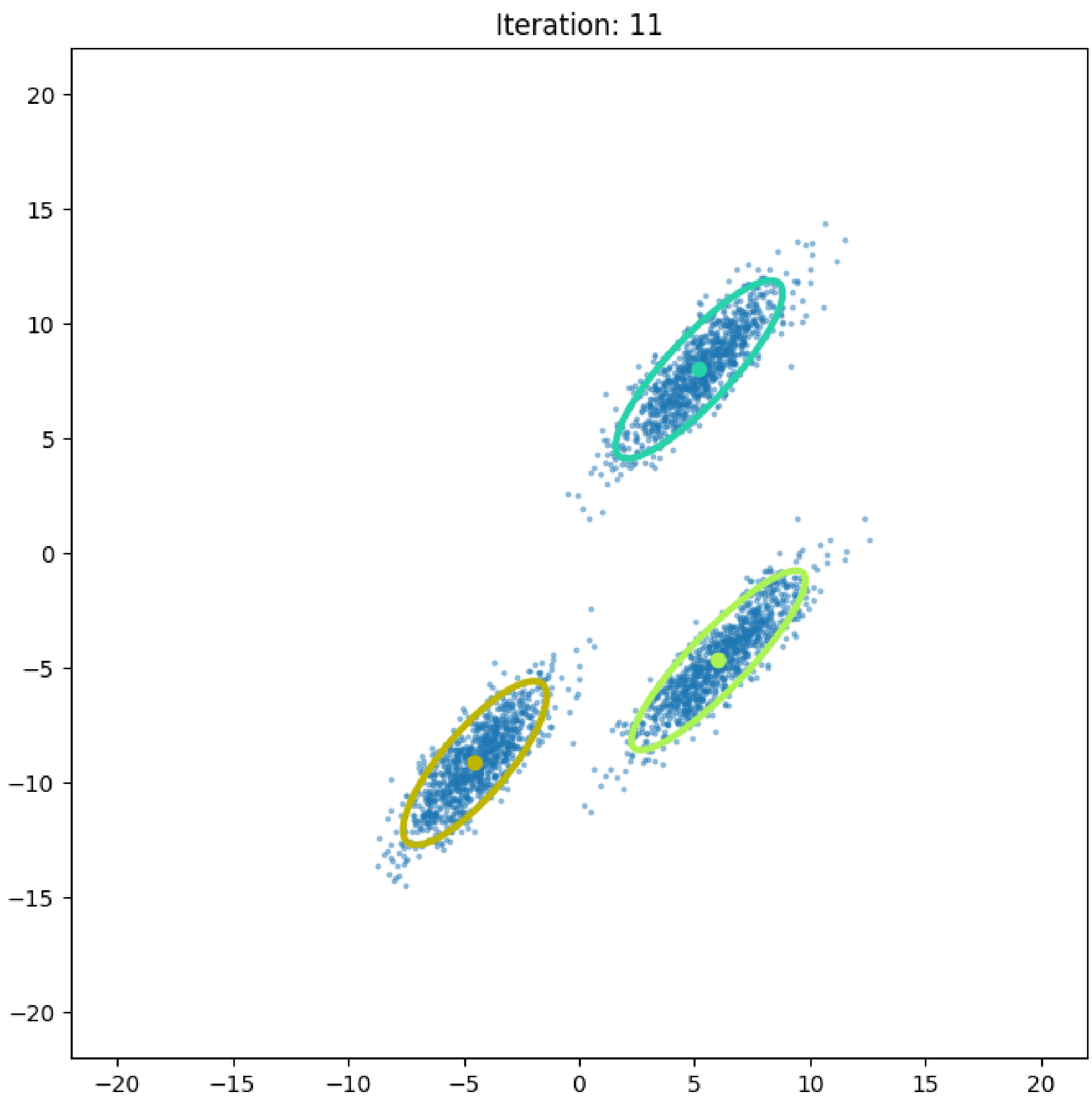

```
Iteration: 9, log-likelihood: -13371.6950
<Figure size 640x480 with 0 Axes>
```




```
Iteration: 10, log-likelihood: -13371.6950
<Figure size 640x480 with 0 Axes>
```

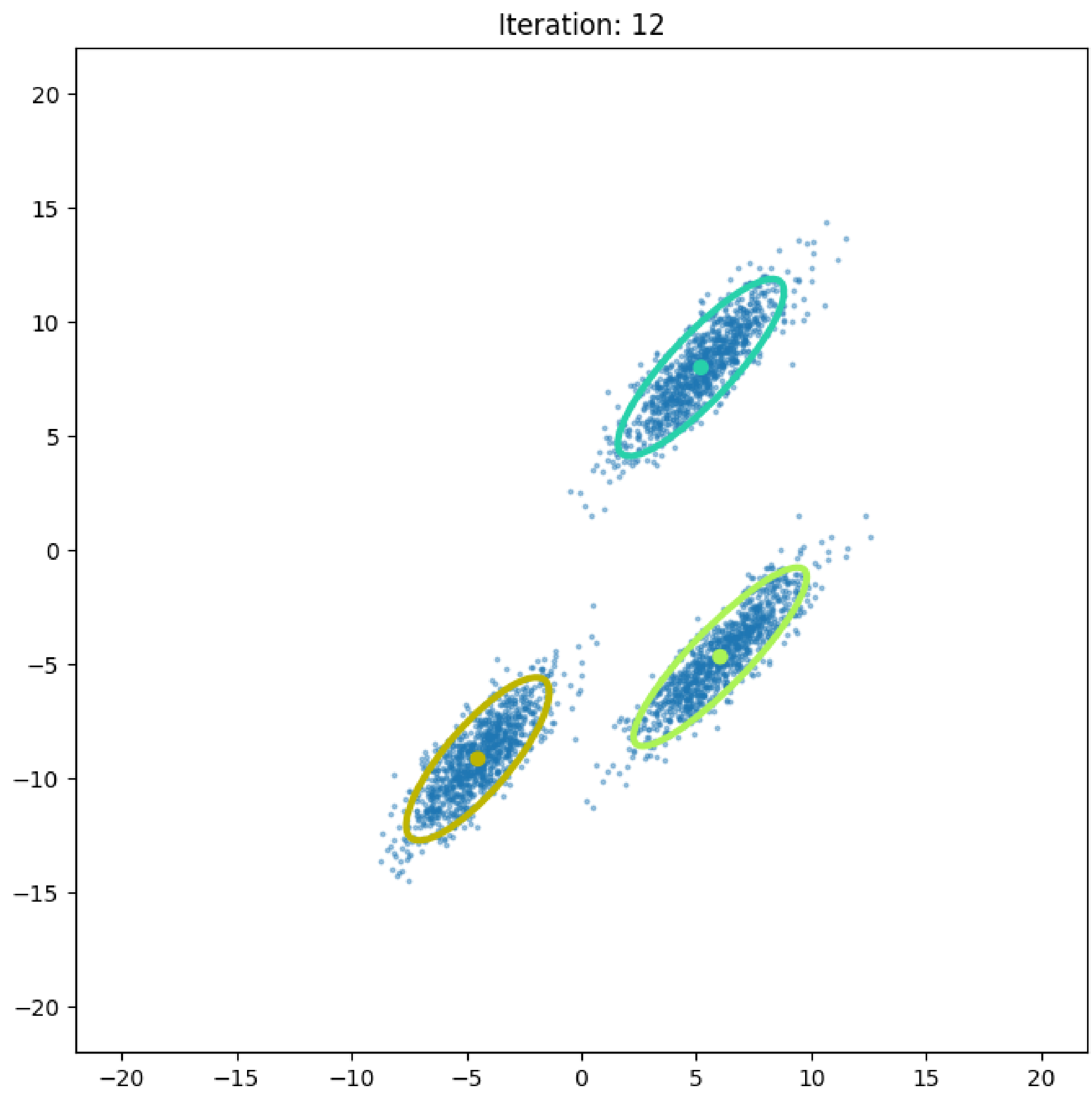



```
Iteration: 11, log-likelihood: -13371.6950
<Figure size 640x480 with 0 Axes>
```

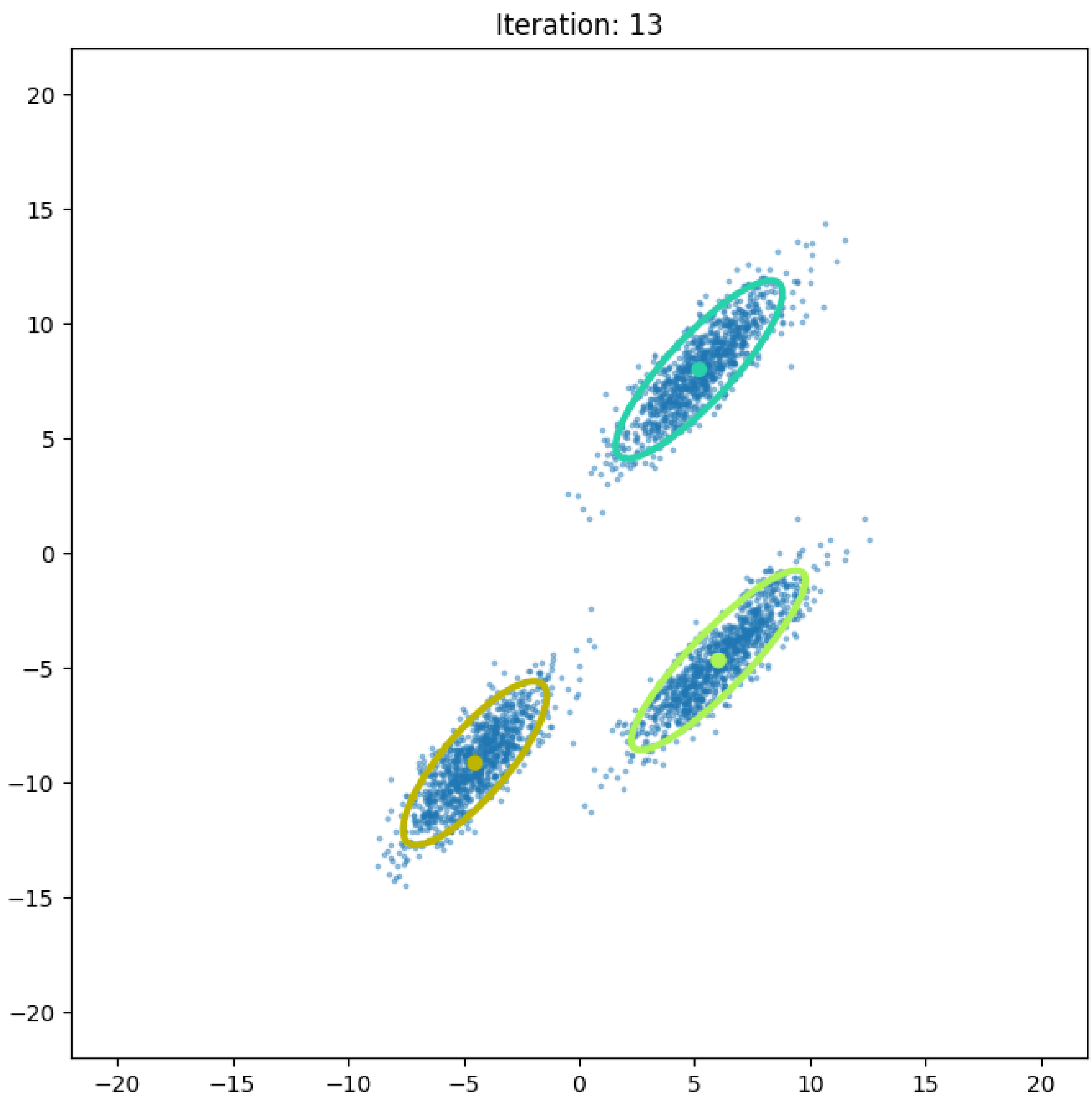


Iteration: 12, log-likelihood: -13371.6950

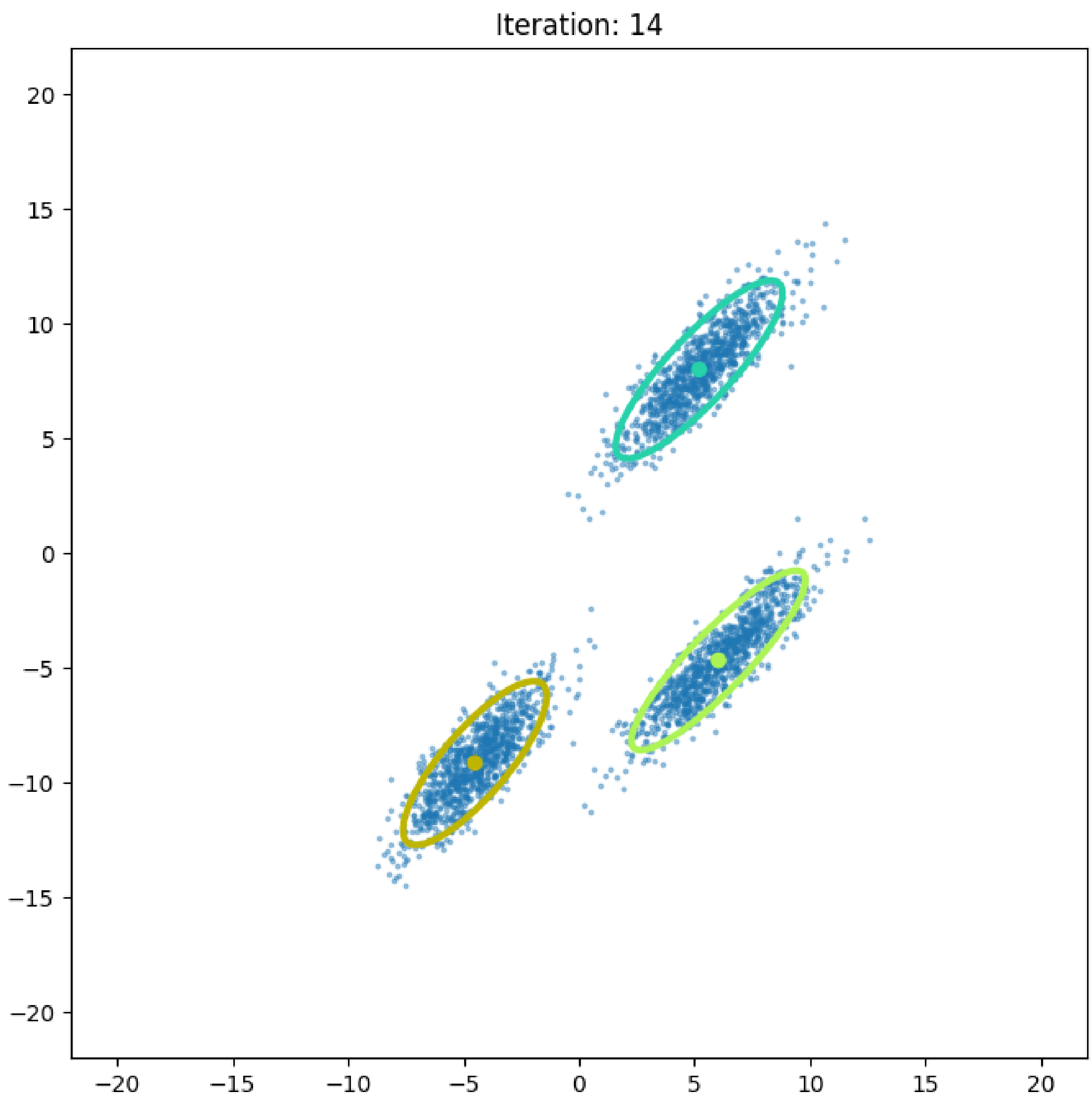
<Figure size 640x480 with 0 Axes>




```
Iteration: 13, log-likelihood: -13371.6950
<Figure size 640x480 with 0 Axes>
```

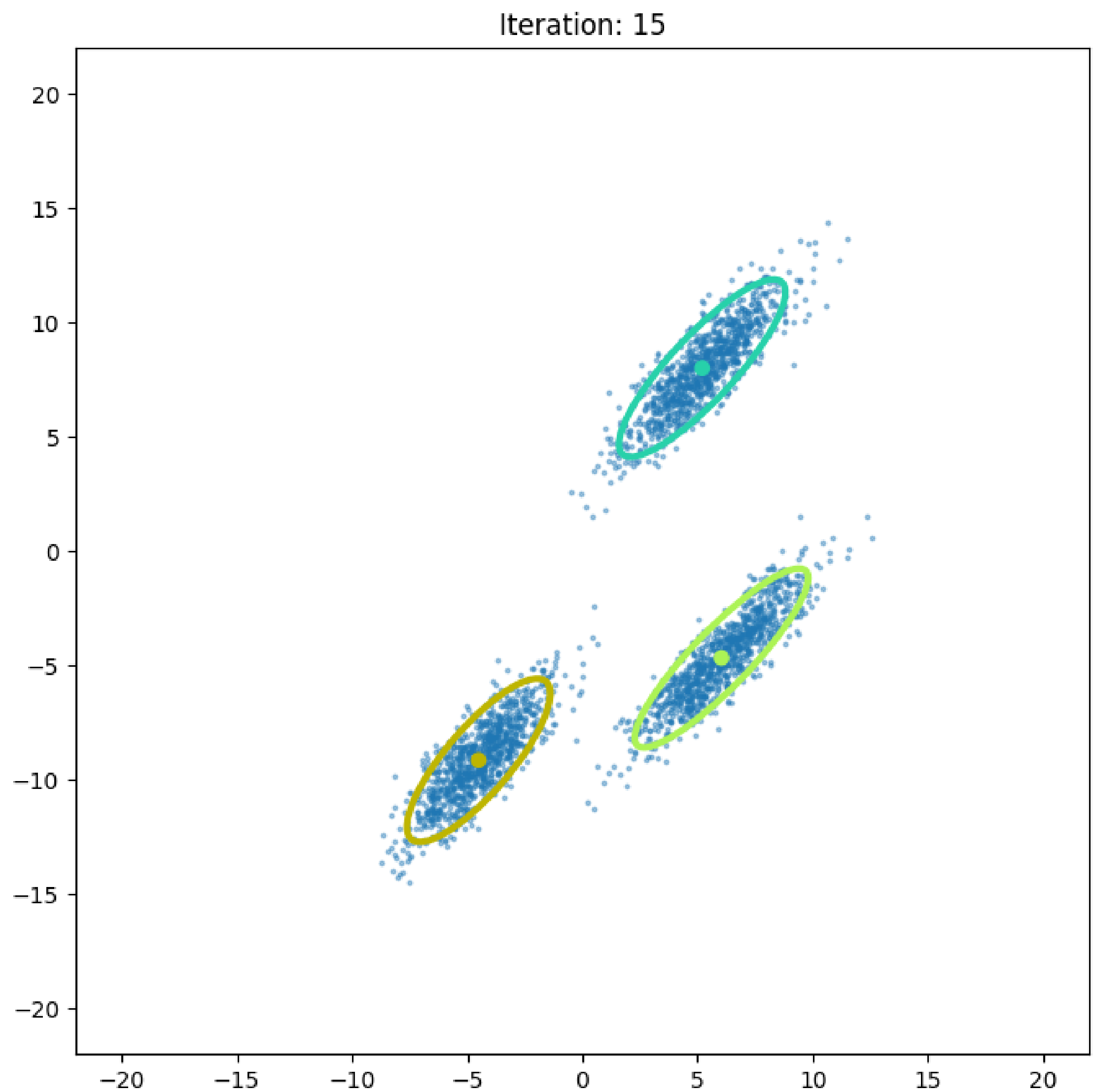



```
Iteration: 14, log-likelihood: -13371.6950
<Figure size 640x480 with 0 Axes>
```

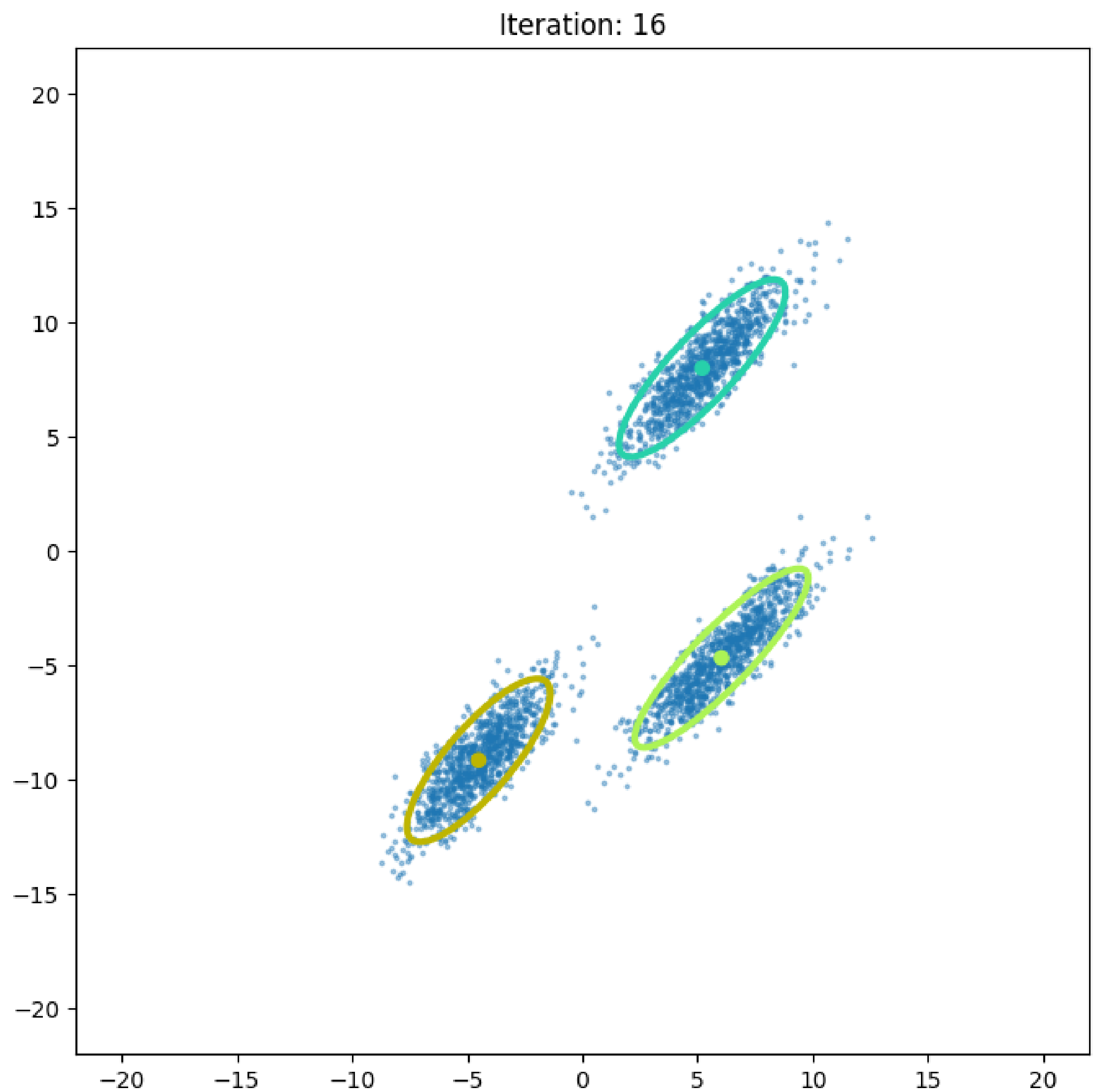



```
Iteration: 15, log-likelihood: -13371.6950
```

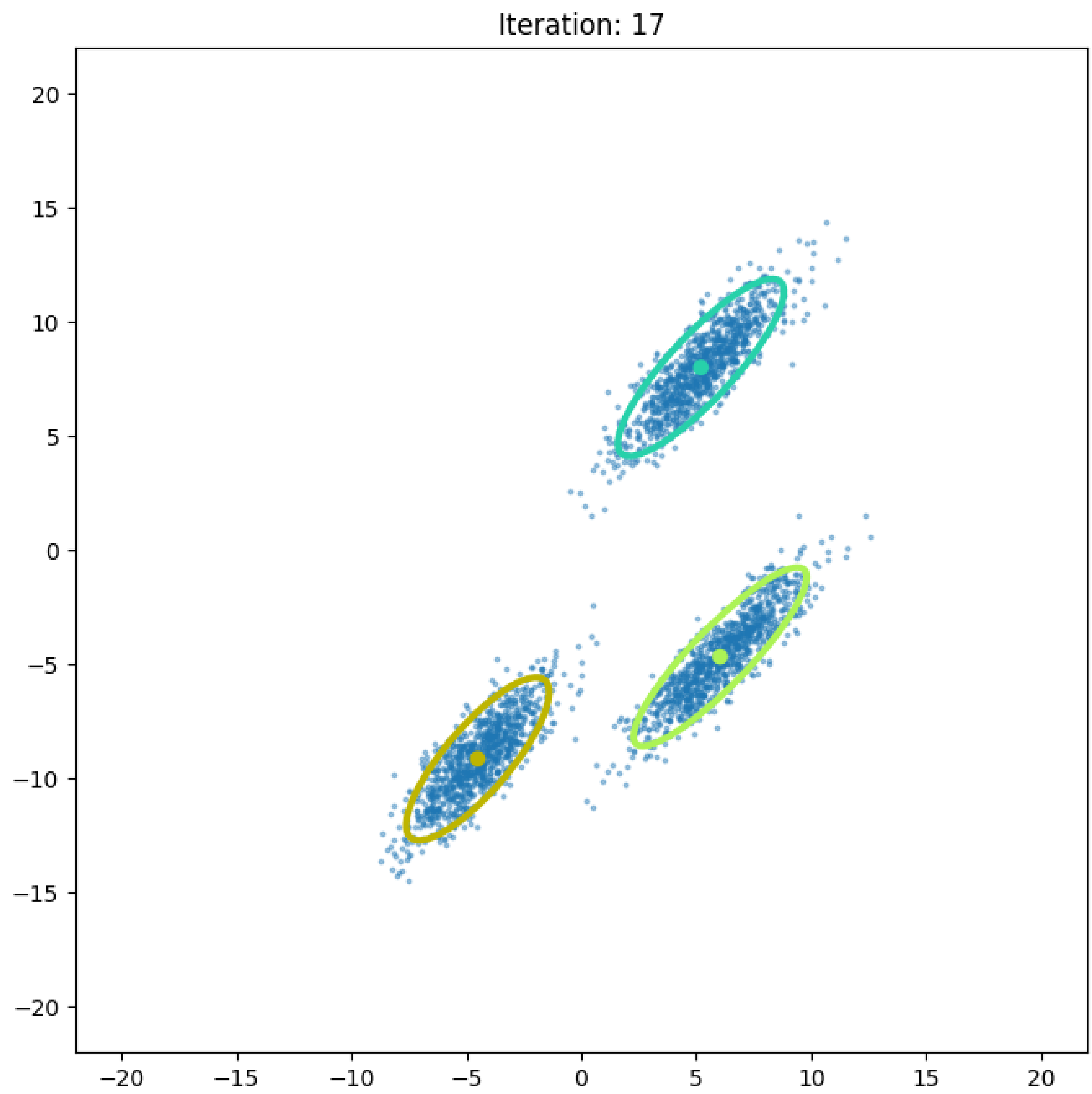
```
<Figure size 640x480 with 0 Axes>
```



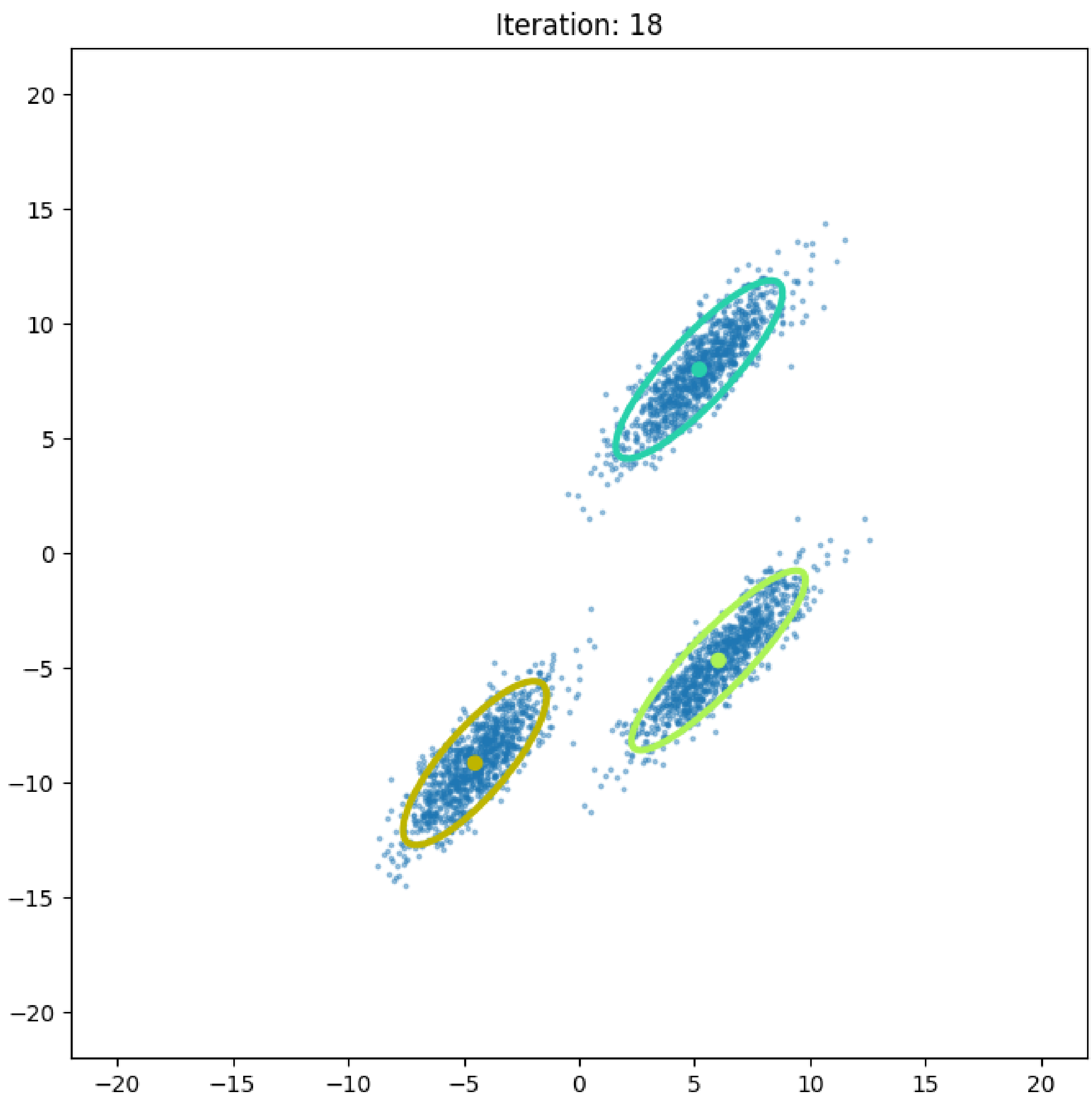

```
Iteration: 16, log-likelihood: -13371.6950
<Figure size 640x480 with 0 Axes>
```



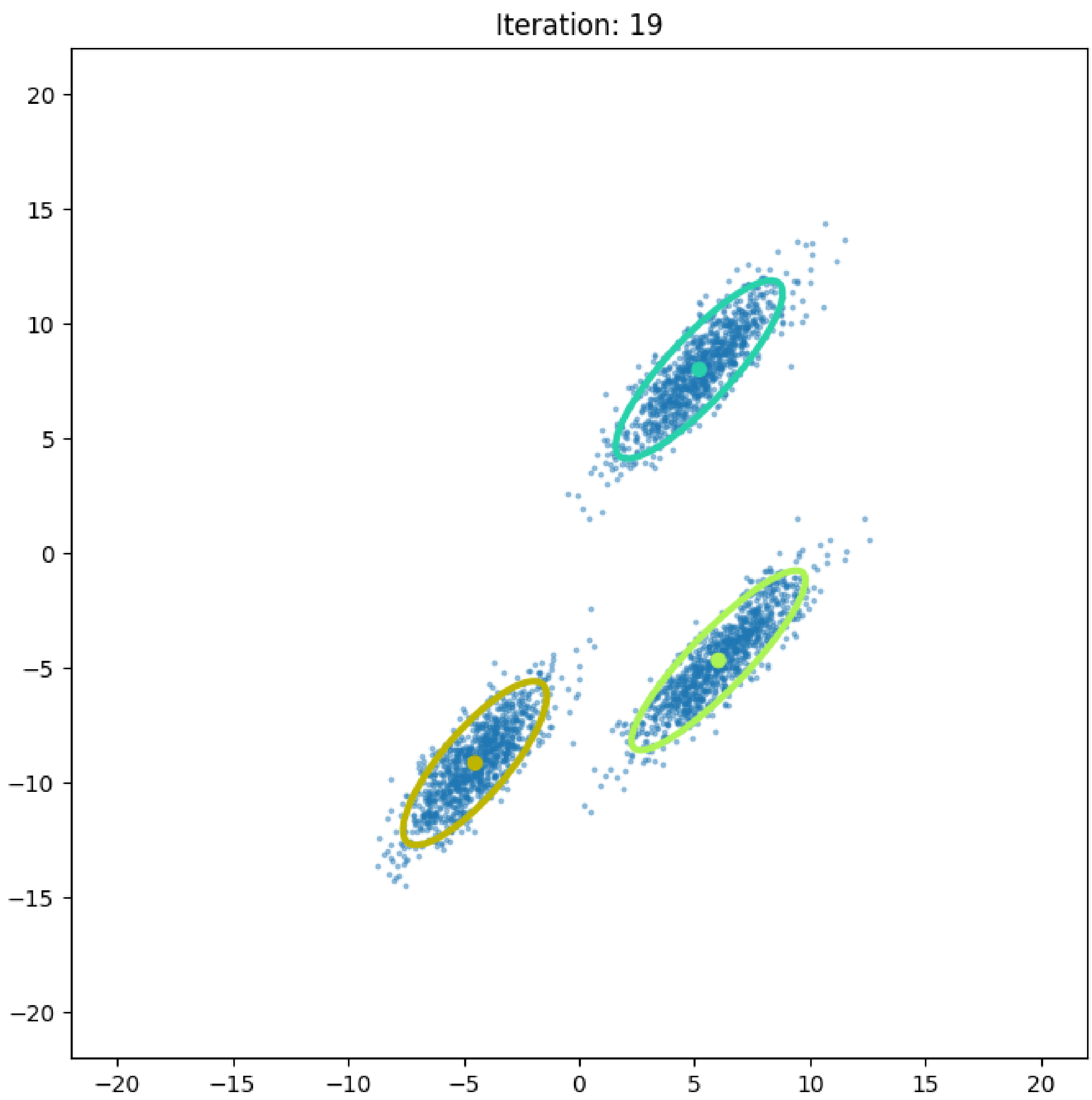

```
Iteration: 17, log-likelihood: -13371.6950
<Figure size 640x480 with 0 Axes>
```



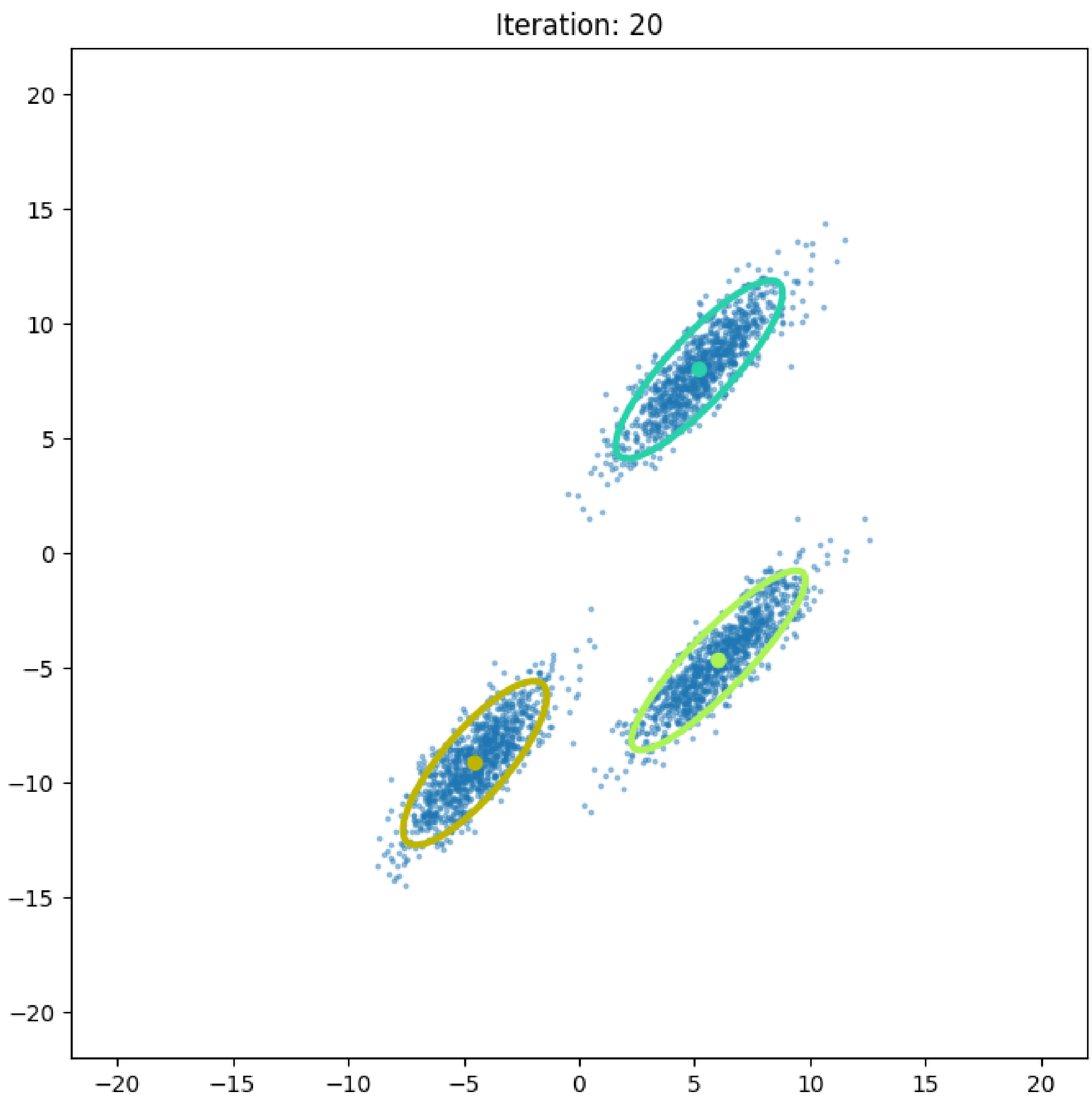

```
Iteration: 18, log-likelihood: -13371.6950
<Figure size 640x480 with 0 Axes>
```




```
Iteration: 19, log-likelihood: -13371.6950
<Figure size 640x480 with 0 Axes>
```




```
Iteration: 20, log-likelihood: -13371.6950
<Figure size 640x480 with 0 Axes>
```




```
<Figure size 640x480 with 0 Axes>
```

2.1.2 Bài tập tự thực hành

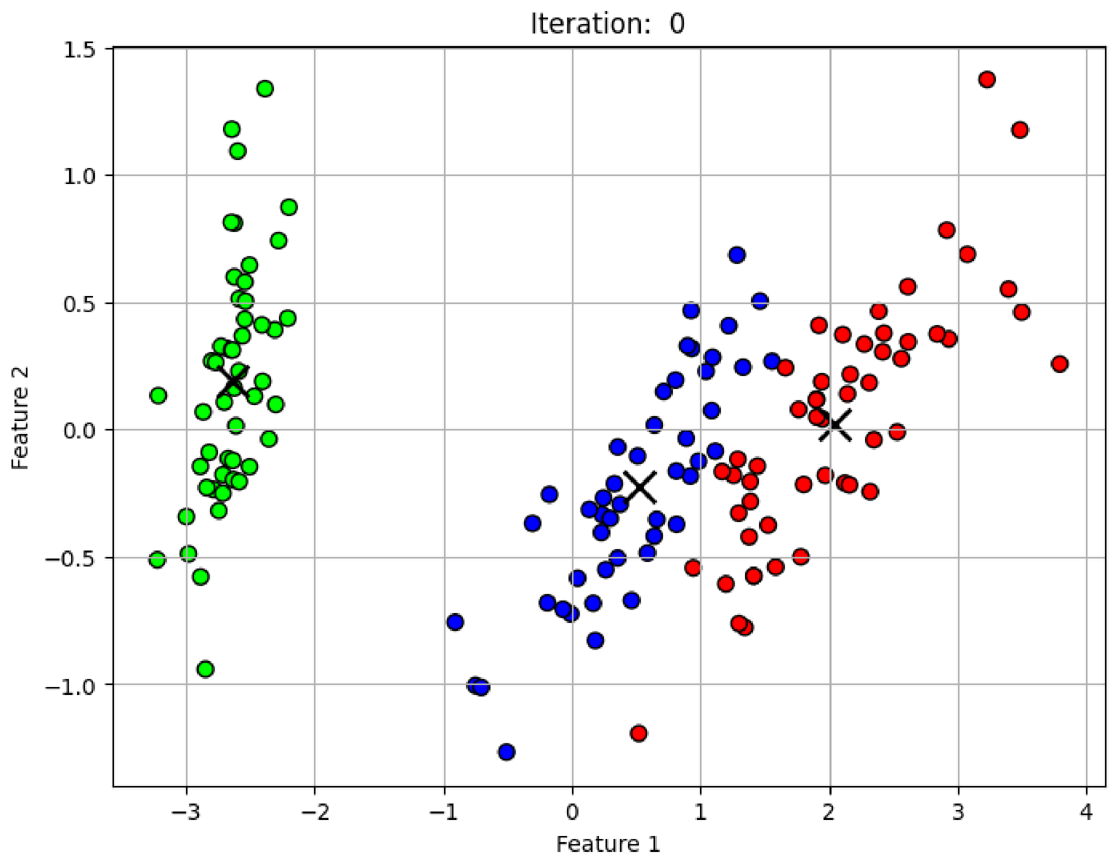
Bài 1 Sử dụng đoạn code trên để áp dựng cho dữ liệu là phần đầu vào của tập dữ liệu hoa Iris (bỏ trường tên loại hoa). Sau khi phân cụm xong hãy đối sánh kết quả với các phân loại đúng.

[ ]:
```
from sklearn.datasets import load_iris

# Load labels for comparison
iris = load_iris()
X = iris.data
iris_labels = iris.target
```

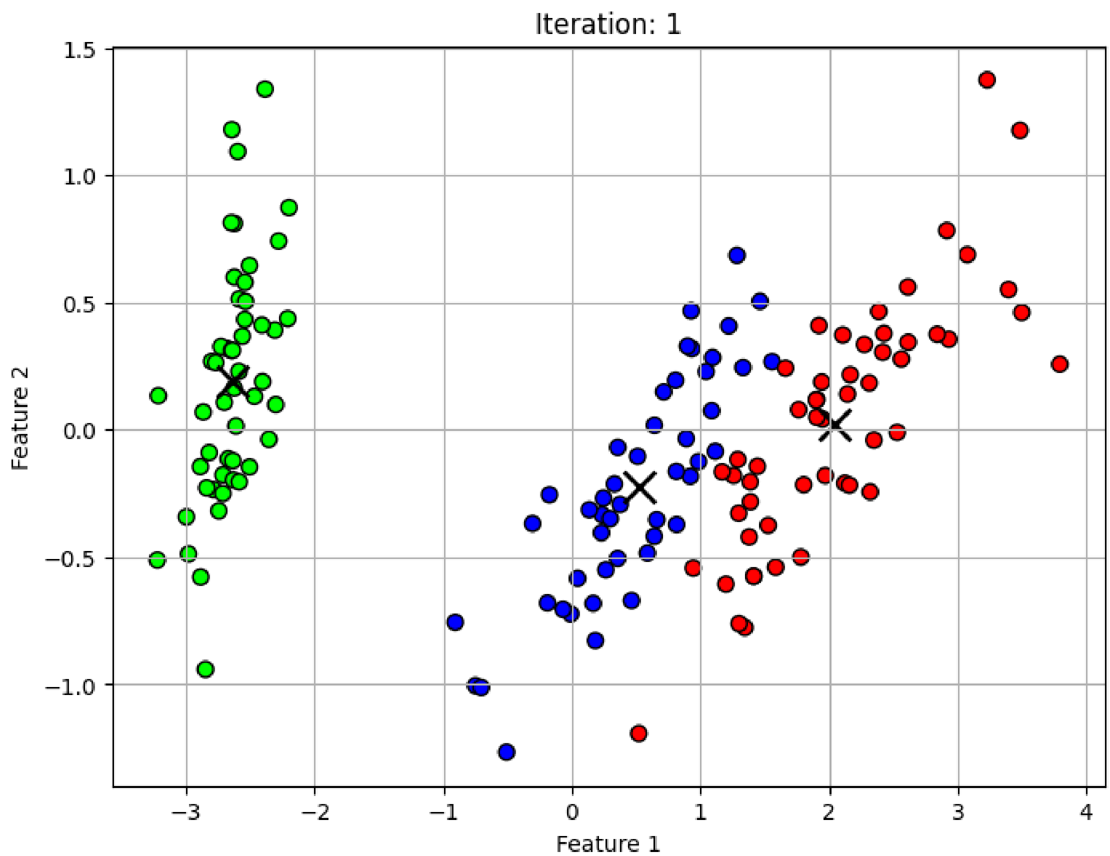
```
[ ]: from sklearn.decomposition import PCA
     # Reduce the dimensionality of X to 2 dimensions using PCA
     pca = PCA(n_components=2)
     X_reduced = pca.fit_transform(X)
```

```
[ ]: import matplotlib.pyplot as plt
     from sklearn.metrics import adjusted_rand_score
     from sklearn.mixture import GaussianMixture
     from matplotlib.colors import ListedColormap

     # Define a colormap for the plot
     cmap = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])

     # Function to plot data and GMM clusters
     def plot_iteration(X, gmm, title):
         plt.figure(figsize=(8, 6))
         plt.scatter(X[:, 0], X[:, 1], c=gmm.predict(X), marker='o', s=50,␣
      ↪cmap=cmap, edgecolor='k')
         plt.scatter(gmm.means_[:, 0], gmm.means_[:, 1], marker='x', s=200,␣
      ↪color='k', linewidths=2)
         plt.title(title)
         plt.xlabel('Feature 1')
         plt.ylabel('Feature 2')
         plt.grid(True)
         plt.show()

     # Training the GMM using EM
     gmm = GaussianMixture(n_components=3, random_state=42)

     # Fit the model
     gmm.fit(X_reduced)

     num_iters = 3   # vì ngay sau lần lặp đầu tiên gần như không có sự thay đổi␣
      ↪tiếp
     # Saving log-likelihood
     log_likelihood = [gmm.score(X_reduced)]
     # plotting
     plot_iteration(X_reduced, gmm, title="Iteration:  0")
     for e in range(num_iters):
         # E-step
         gmm.fit(X_reduced)
         # M-step
         # Computing log-likelihood
         log_likelihood.append(gmm.score(X_reduced))
         print("Iteration: {}, log-likelihood: {:.4f}".format(e+1,␣
      ↪log_likelihood[-1]))
         # plotting
```

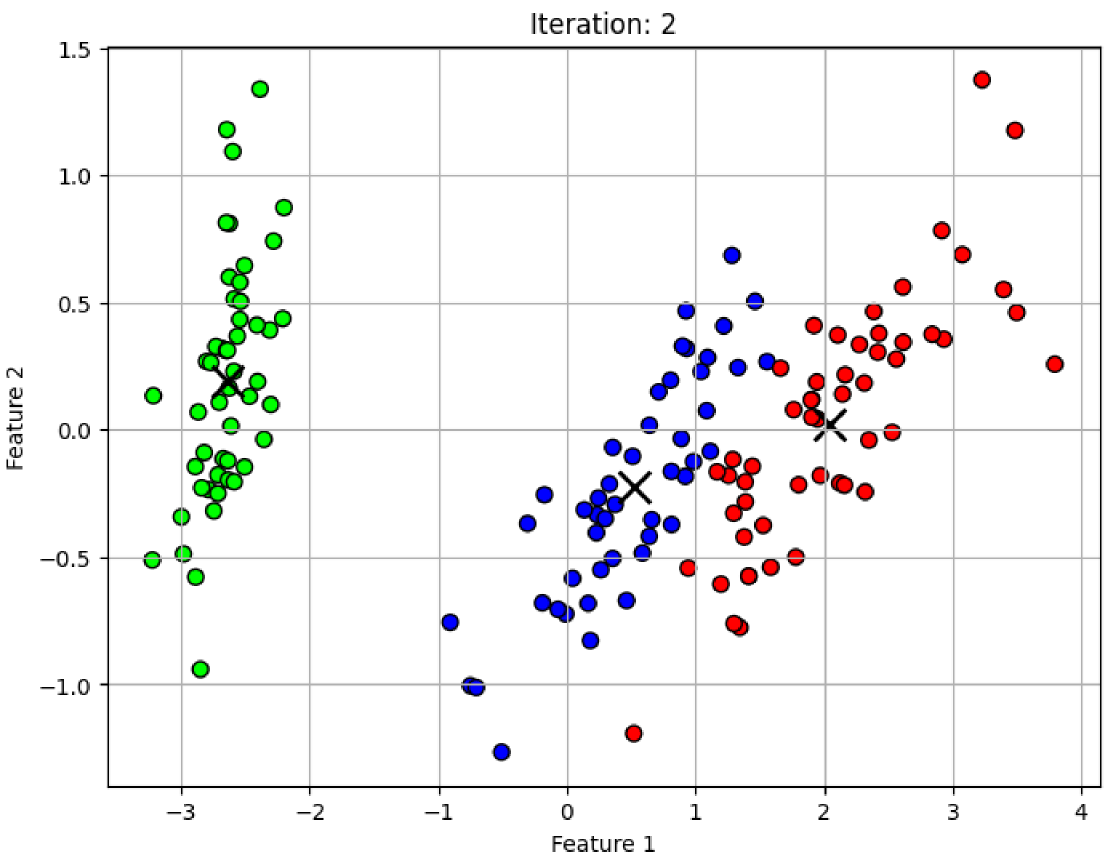
```python
    plot_iteration(X_reduced, gmm, title="Iteration: " + str(e+1))

# Convert cluster labels to match with true labels
predicted_labels_gmm = gmm.predict(X_reduced)
ari_gmm = adjusted_rand_score(iris_labels, predicted_labels_gmm)

print("Adjusted Rand Index (GMM):", ari_gmm)
```



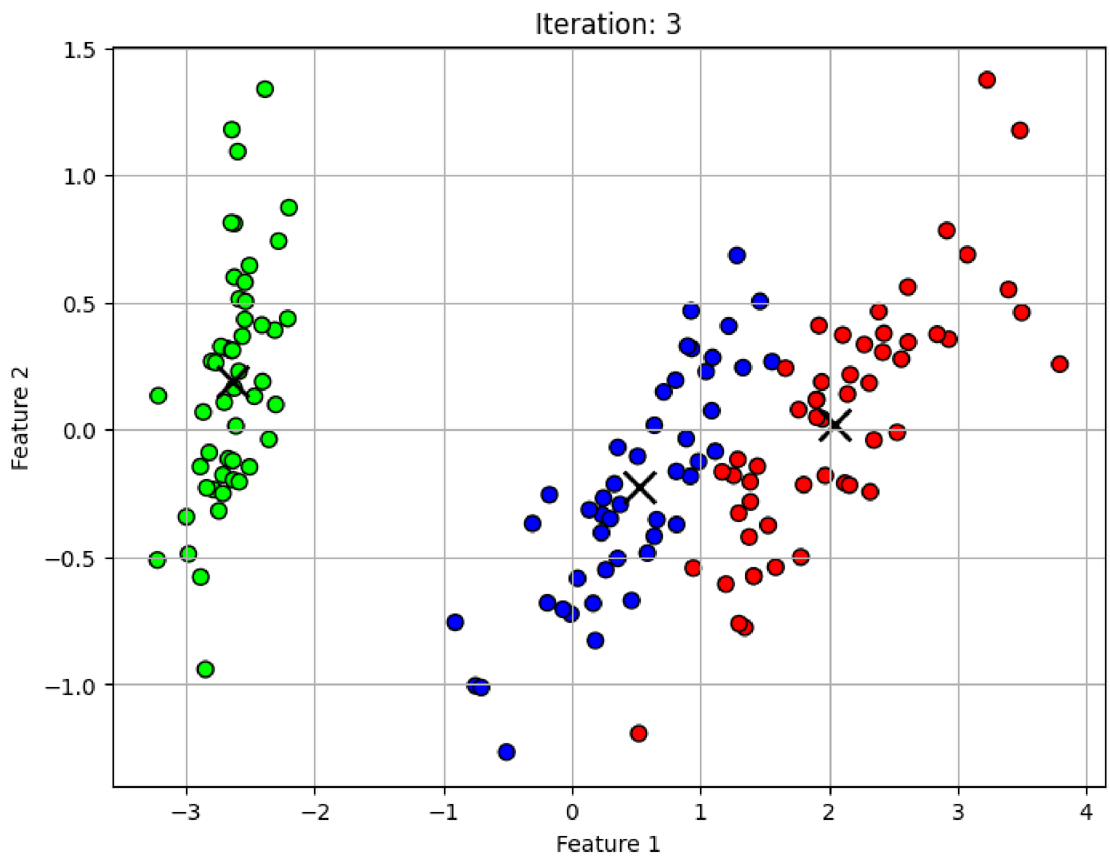

```
Iteration: 1, log-likelihood: -1.8747
```



Iteration: 2, log-likelihood: -1.8747



Iteration: 3, log-likelihood: -1.8747




```
Adjusted Rand Index (GMM): 0.9410449800736683
```

Bài 2 Hãy sử dụng Gaussian Mixture model để phân cụm dữ liệu tự tạo đã có trong ví dụ 1 của phần này. So sánh và giải thích kết quả so với cách dùng K-means ở ví dụ 1.

[ ]:
```
import numpy as np
from numpy import random
from matplotlib.patches import Ellipse
import matplotlib.transforms as transforms
from scipy.stats import multivariate_normal
from sklearn.cluster import KMeans

class GMM_Fix:
    """
    Rewrite to fix:
    LinAlgError: When `allow_singular is False`, the input matrix must be␣
↪symmetric positive definite.
    """

    def __init__(
```

```
        self, k, dim, init_mu=None, init_sigma=None, init_pi=None, colors=None
    ):
        """
        Define a model with known number of clusters and dimensions.
        input:
            - k: Number of Gaussian clusters
            - dim: Dimension
            - init_mu: initial value of mean of clusters (k, dim)
                       (default) random from uniform[-10, 10]
            - init_sigma: initial value of covariance matrix of clusters (k,␣
↪dim, dim)
                          (default) Identity matrix for each cluster
            - init_pi: initial value of cluster weights (k,)
                       (default) equal value to all cluster i.e. 1/k
            - colors: Color valu for plotting each cluster (k, 3)
                      (default) random from uniform[0, 1]
        """
        self.k = k
        self.dim = dim
        if init_mu is None:
            init_mu = random.rand(k, dim) * 20 - 10
        self.mu = init_mu
        if init_sigma is None:
            init_sigma = np.zeros((k, dim, dim))
            for i in range(k):
                init_sigma[i] = np.eye(dim)
        self.sigma = init_sigma
        if init_pi is None:
            init_pi = np.ones(self.k) / self.k
        self.pi = init_pi
        if colors is None:
            colors = random.rand(k, 3)
            for i in range(k):
                colors[i, 2] = i / k
        self.colors = colors

    # def init_em(self, X):
    #     """
    #     Initialization for EM algorithm.
    #     input:
    #         - X: data (batch_size, dim)
    #     """
    #     self.data = X
    #     self.num_points = X.shape[0]
    #     self.z = np.zeros((self.num_points, self.k))
    def init_em(self, X):
        '''
```

```
        Initialization for EM algorithm.
        input:
            - X: data (batch_size, dim)
        '''
        self.data = X
        self.num_points = X.shape[0]
        self.z = np.zeros((self.num_points, self.k))

        # Initialize mu using KMeans
        kmeans = KMeans(n_clusters=self.k, random_state=0).fit(X)
        self.mu = kmeans.cluster_centers_

        # Initialize sigma using covariance of clusters
        for i in range(self.k):
            cluster_points = X[kmeans.labels_ == i]
            self.sigma[i] = np.cov(cluster_points.T)
            if not np.all(np.linalg.eigvals(self.sigma[i]) > 0):
                self.sigma[i] += 1e-6 * np.abs(self.sigma[i]).max() * np.
↪eye(self.dim)

    def e_step(self):
        """
        E-step of EM algorithm.
        """
        for i in range(self.k):
            self.z[:, i] = self.pi[i] * multivariate_normal.pdf(
                self.data, mean=self.mu[i], cov=self.sigma[i]
            )
        self.z /= self.z.sum(axis=1, keepdims=True)

    # def m_step(self):
    #     '''
    #     M-step of EM algorithm.
    #     '''
    #     sum_z = self.z.sum(axis=0)
    #     self.pi = sum_z / self.num_points
    #     self.mu = np.matmul(self.z.T, self.data)
    #     self.mu /= sum_z[:, None]
    #     for i in range(self.k):
    #         j = np.expand_dims(self.data, axis=1) - self.mu[i]
    #         s = np.matmul(j.transpose([0, 2, 1]), j)
    #         self.sigma[i] = np.matmul(s.transpose(1, 2, 0), self.z[:, i] )
    #         self.sigma[i] /= sum_z[i]
    # def m_step(self):
    #     """
    #     M-step of EM algorithm.
    #     """
```

```
    #     sum_z = self.z.sum(axis=0)
    #     self.pi = sum_z / self.num_points
    #     self.mu = np.matmul(self.z.T, self.data)
    #     self.mu /= sum_z[:, None]
    #     for i in range(self.k):
    #         j = np.expand_dims(self.data, axis=1) - self.mu[i]
    #         s = np.matmul(j.transpose([0, 2, 1]), j)
    #         self.sigma[i] = np.matmul(s.transpose(1, 2, 0), self.z[:, i])
    #         self.sigma[i] /= sum_z[i]
    #         # Add a small value to the diagonal to ensure positive
↪definiteness
    #         self.sigma[i] += 1e-6 * np.eye(self.dim)
    def m_step(self):
        '''
        M-step of EM algorithm.
        '''
        sum_z = self.z.sum(axis=0)
        self.pi = sum_z / self.num_points
        self.mu = np.matmul(self.z.T, self.data)
        self.mu /= sum_z[:, None]
        for i in range(self.k):
            self.sigma[i] = np.cov(self.data.T, aweights=self.z[:, i],
↪bias=True)
            if not np.all(np.linalg.eigvals(self.sigma[i]) > 0):
                self.sigma[i] += 1e-6 * np.abs(self.sigma[i]).max() * np.
↪eye(self.dim)


    def log_likelihood(self, X):
        """
        Compute the log-likelihood of X under current parameters
        input:
            - X: Data (batch_size, dim)
        output:
            - log-likelihood of X: Sum_n Sum_k log(pi_k * N( X_n | mu_k,
↪sigma_k ))
        """
        ll = []
        for d in X:
            tot = 0
            for i in range(self.k):
                tot += self.pi[i] * multivariate_normal.pdf(
                    d, mean=self.mu[i], cov=self.sigma[i]
                )
            ll.append(np.log(tot))
        return np.sum(ll)
```

```python
    def plot_gaussian(self, mean, cov, ax, n_std=3.0, facecolor="none",␣
↪**kwargs):
        """
        Utility function to plot one Gaussian from mean and covariance.
        """
        pearson = cov[0, 1] / np.sqrt(cov[0, 0] * cov[1, 1])
        ell_radius_x = np.sqrt(1 + pearson)
        ell_radius_y = np.sqrt(1 - pearson)
        ellipse = Ellipse(
            (0, 0),
            width=ell_radius_x * 2,
            height=ell_radius_y * 2,
            facecolor=facecolor,
            **kwargs
        )
        scale_x = np.sqrt(cov[0, 0]) * n_std
        mean_x = mean[0]
        scale_y = np.sqrt(cov[1, 1]) * n_std
        mean_y = mean[1]
        transf = (
            transforms.Affine2D()
            .rotate_deg(45)
            .scale(scale_x, scale_y)
            .translate(mean_x, mean_y)
        )
        ellipse.set_transform(transf + ax.transData)
        return ax.add_patch(ellipse)

    def draw(self, ax, n_std=2.0, facecolor="none", **kwargs):
        """
        Function to draw the Gaussians.
        Note: Only for two-dimensionl dataset
        """
        if self.dim != 2:
            print("Drawing available only for 2D case.")
            return
        for i in range(self.k):
            self.plot_gaussian(
                self.mu[i],
                self.sigma[i],
                ax,
                n_std=n_std,
                edgecolor=self.colors[i],
                **kwargs
            )
```

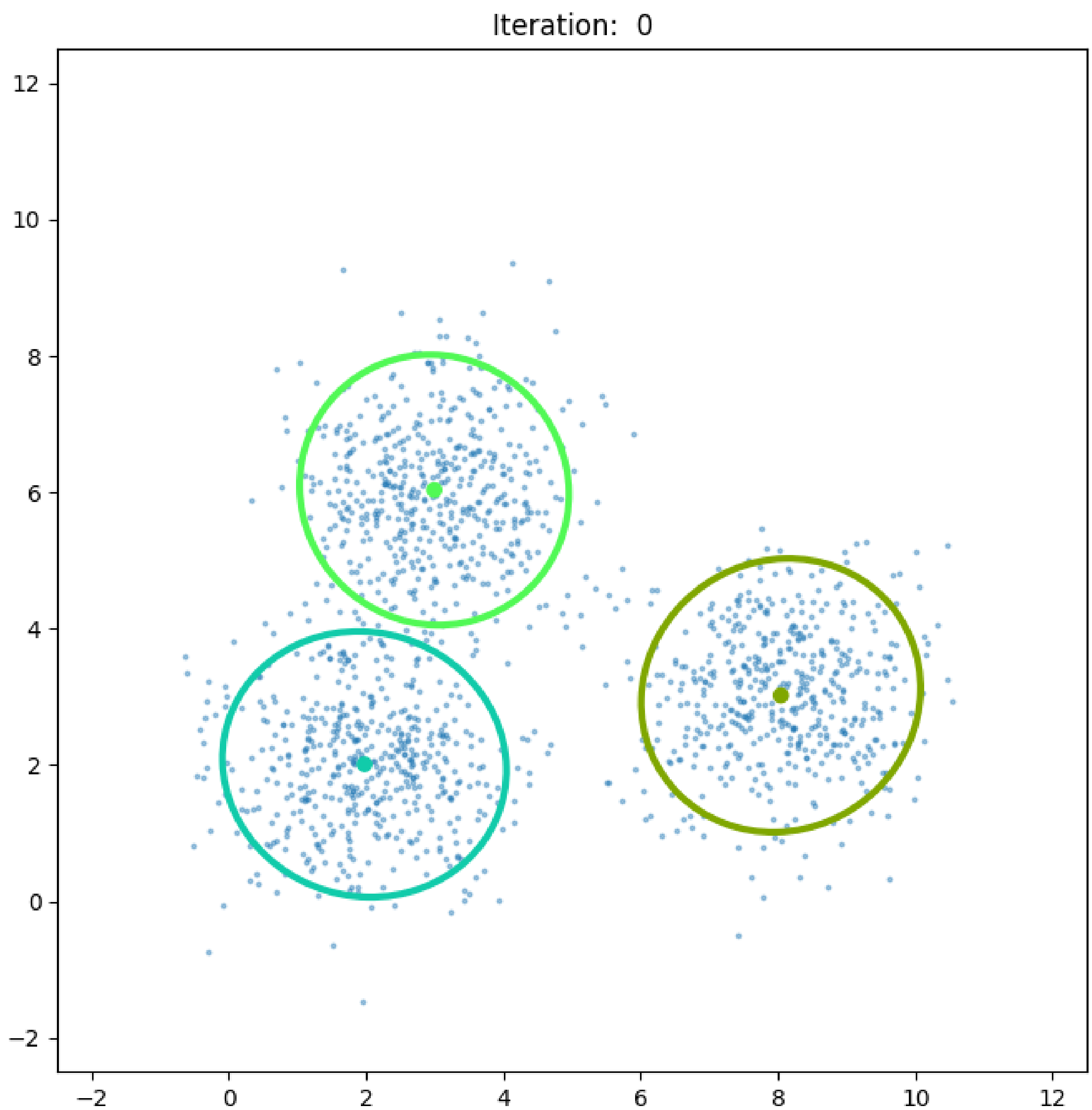
[ ]:
```python
# Gọi các thư viện cần thiết
# Ta tự xây dựng phần k-means nên sẽ không gọi sklearn

from __future__ import print_function
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial.distance import cdist
np.random.seed(11)

# Kỳ vọng và hiệp phương sai của 3 cụm dữ liệu
means = [[2, 2], [8, 3], [3, 6]]
cov = [[1, 0], [0, 1]]

# Số điểm mỗi cụm dữ liệu
N = 500

# Tạo các cụm dữ liệu qua phân bố chuẩn (Gaussian)
X0 = np.random.multivariate_normal(means[0], cov, N)
X1 = np.random.multivariate_normal(means[1], cov, N)
X2 = np.random.multivariate_normal(means[2], cov, N)
# Tổng hợp dữ liệu từ các cụm
X = np.concatenate((X0, X1, X2), axis = 0)

# Số cụm = 3
K = 3

# Gán nhãn ban đầu cho các cụm, sau đó ta test model và so sánh
original_label = np.asarray([0]*N + [1]*N + [2]*N).T
```

[ ]:
```python
# Create a Gaussian Mixture Model
gmm = GMM_Fix(3, 2) # 3 cụm, 2 chiều
```

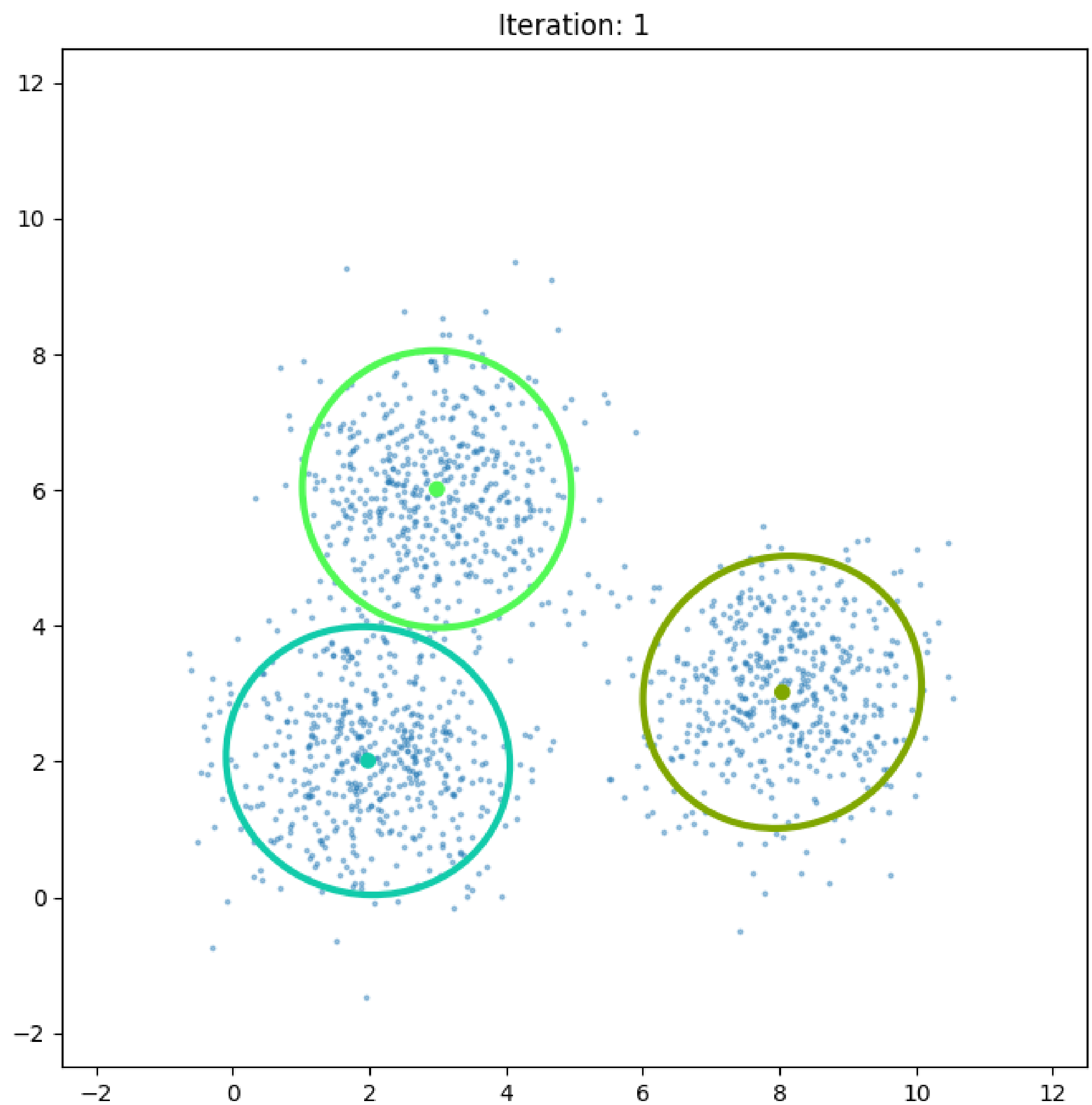
[ ]:
```python
def plot_ex2_gaussian(title):
    '''
    Draw the data points and the fitted mixture model.
    input:
        - title: title of plot and name with which it will be saved.
    '''
    fig = plt.figure(figsize=(8, 8))
    ax = fig.gca()
    ax.scatter(X[:, 0], X[:, 1], s=3, alpha=0.4)
    ax.scatter(gmm.mu[:, 0], gmm.mu[:, 1], c=gmm.colors)
    gmm.draw(ax, lw=3)
    ax.set_xlim((-2.5, 12.5))
    ax.set_ylim((-2.5, 12.5))

    plt.title(title)
```

```
    # plt.savefig(title.replace(':', '_'))
    plt.show()
    plt.clf()
```

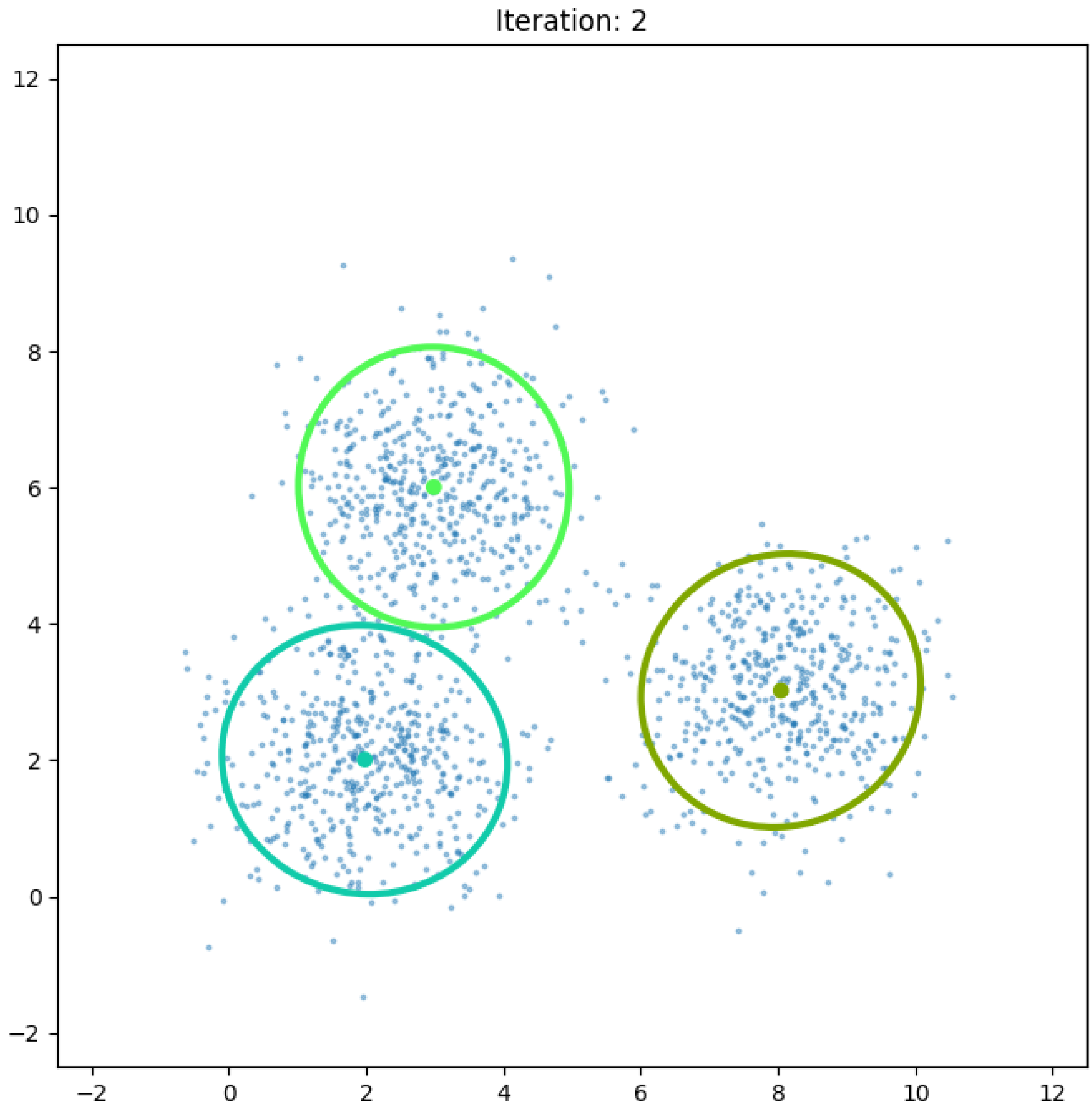
[ ]:
```
# Training the GMM using EM

# Initialize EM algo with data
gmm.init_em(X)

num_iters = 20
# Saving log-likelihood
log_likelihood = [gmm.log_likelihood(X)]
# plotting
plot_ex2_gaussian("Iteration:  0")
for e in range(num_iters):
    # E-step
    gmm.e_step()
    # M-step
    gmm.m_step()
    # Computing log-likelihood
    log_likelihood.append(gmm.log_likelihood(X))
    print("Iteration: {}, log-likelihood: {:.4f}".format(e+1,␣
 ↪log_likelihood[-1]))
    # plotting
    plot_ex2_gaussian(title="Iteration: " + str(e+1))
```



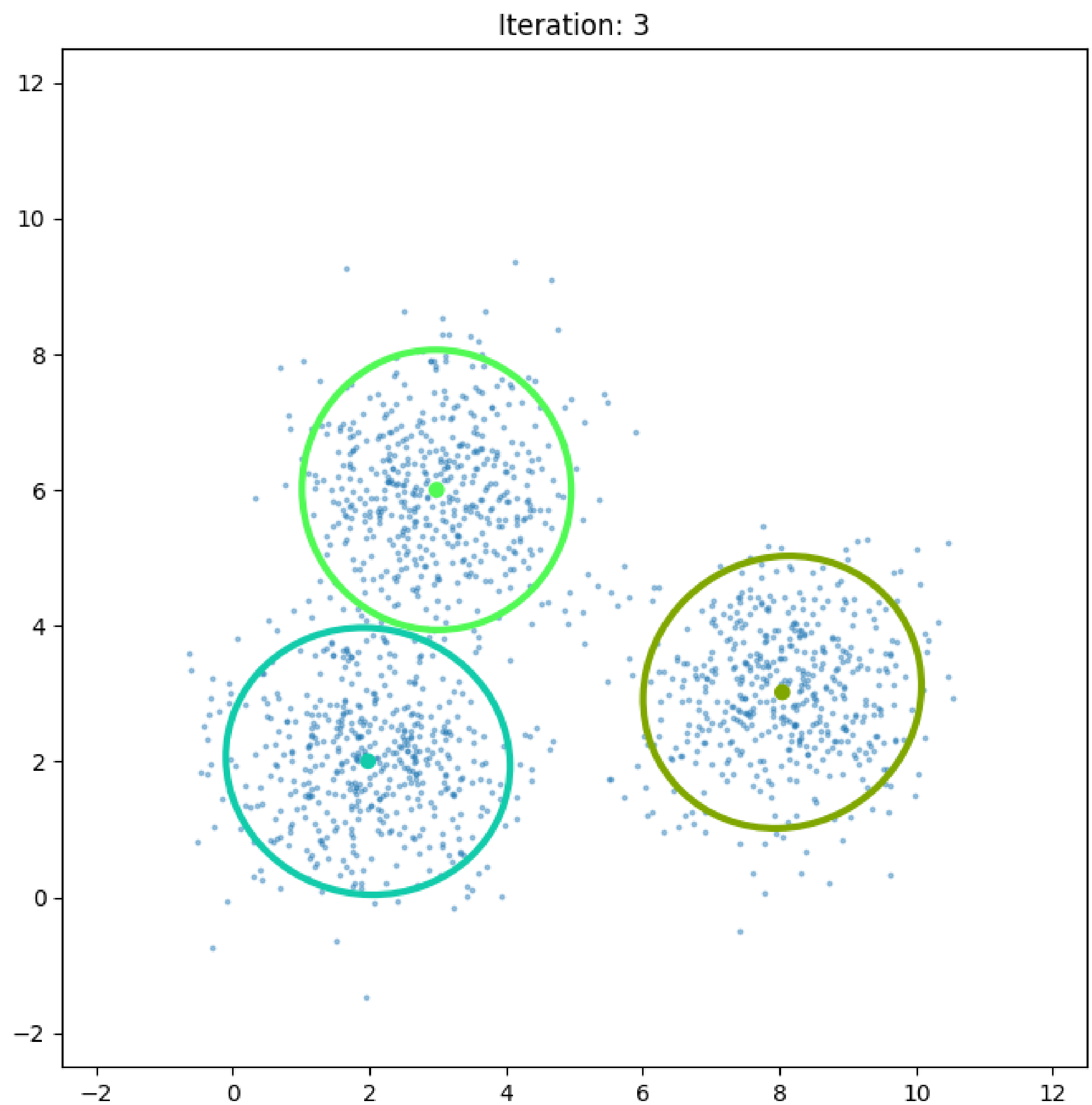

```
Iteration: 1, log-likelihood: -5874.6403
<Figure size 640x480 with 0 Axes>
```



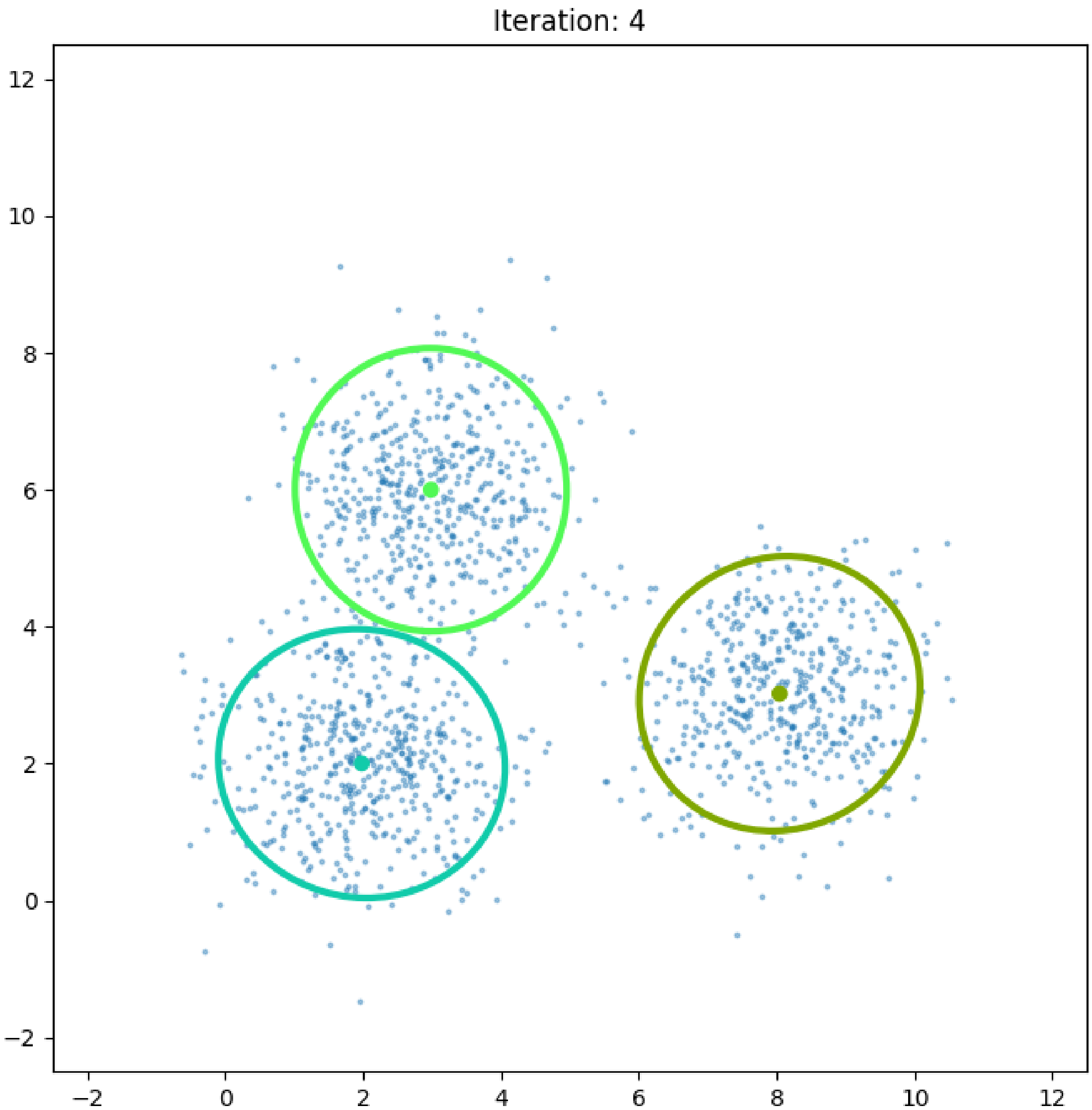

```
Iteration: 2, log-likelihood: -5874.5441
<Figure size 640x480 with 0 Axes>
```



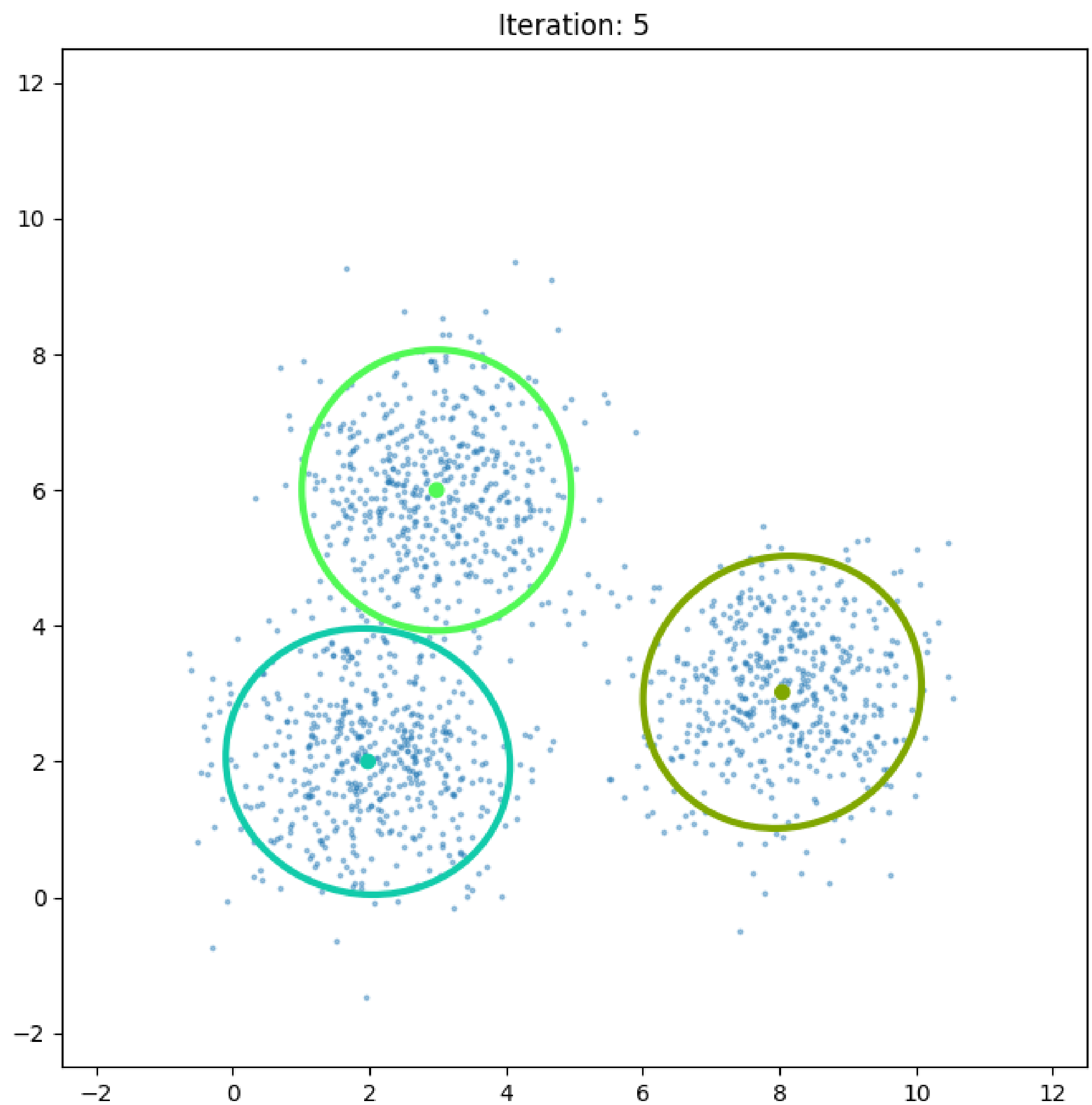

```
Iteration: 3, log-likelihood: -5874.5096
<Figure size 640x480 with 0 Axes>
```




```
Iteration: 4, log-likelihood: -5874.4947

<Figure size 640x480 with 0 Axes>
```

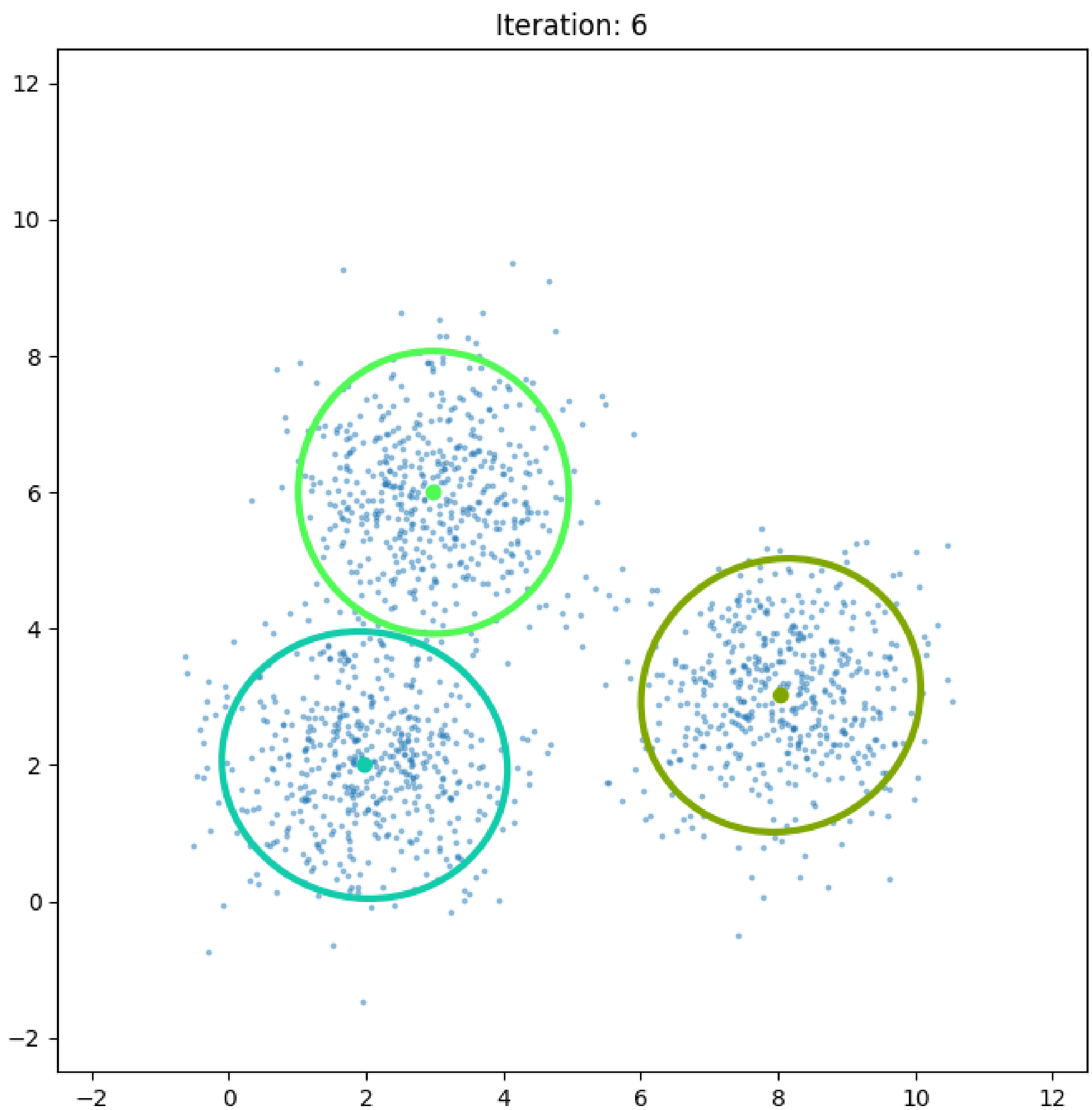



```
Iteration: 5, log-likelihood: -5874.4881
<Figure size 640x480 with 0 Axes>
```

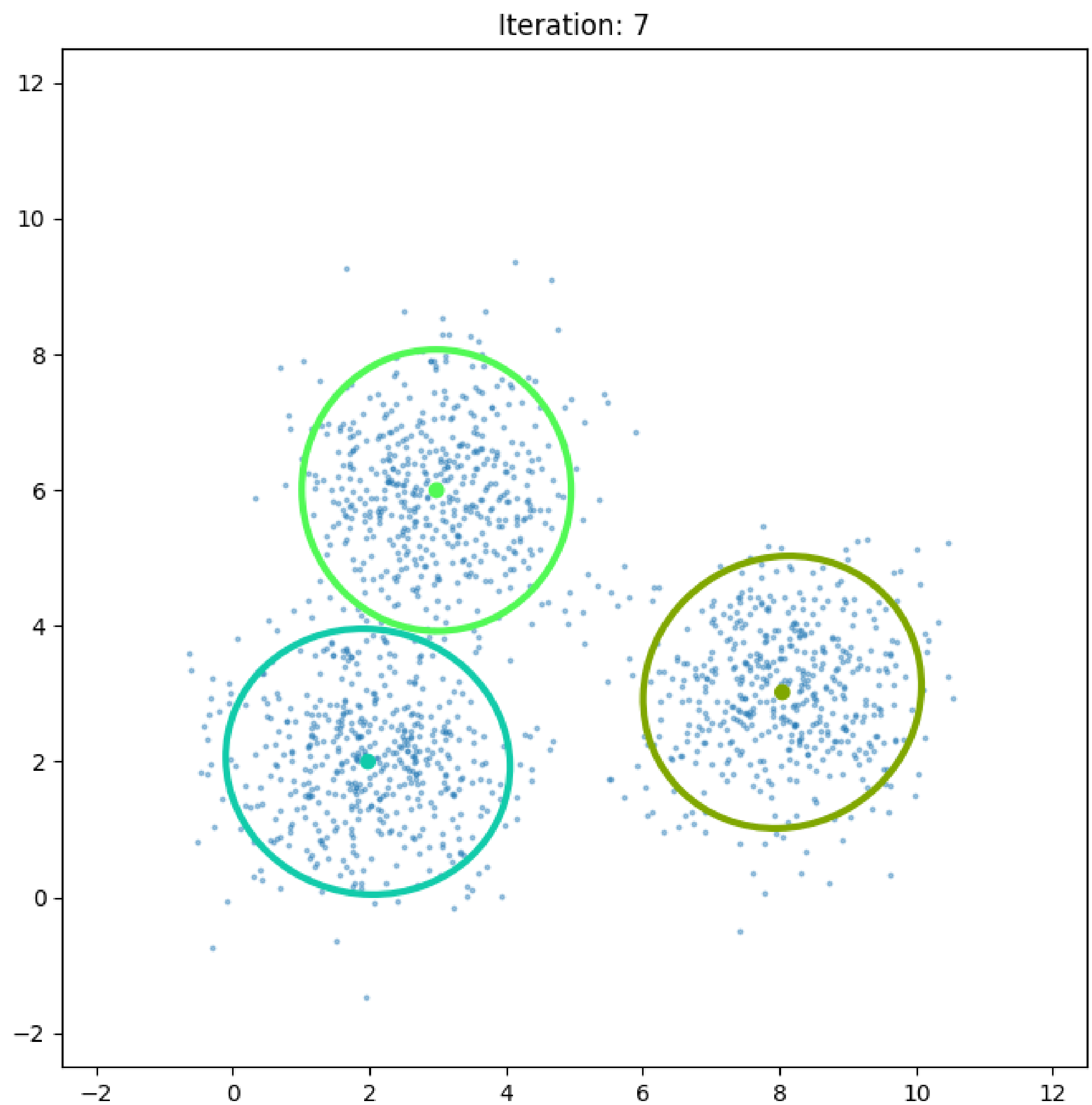


Iteration: 6, log-likelihood: -5874.4852

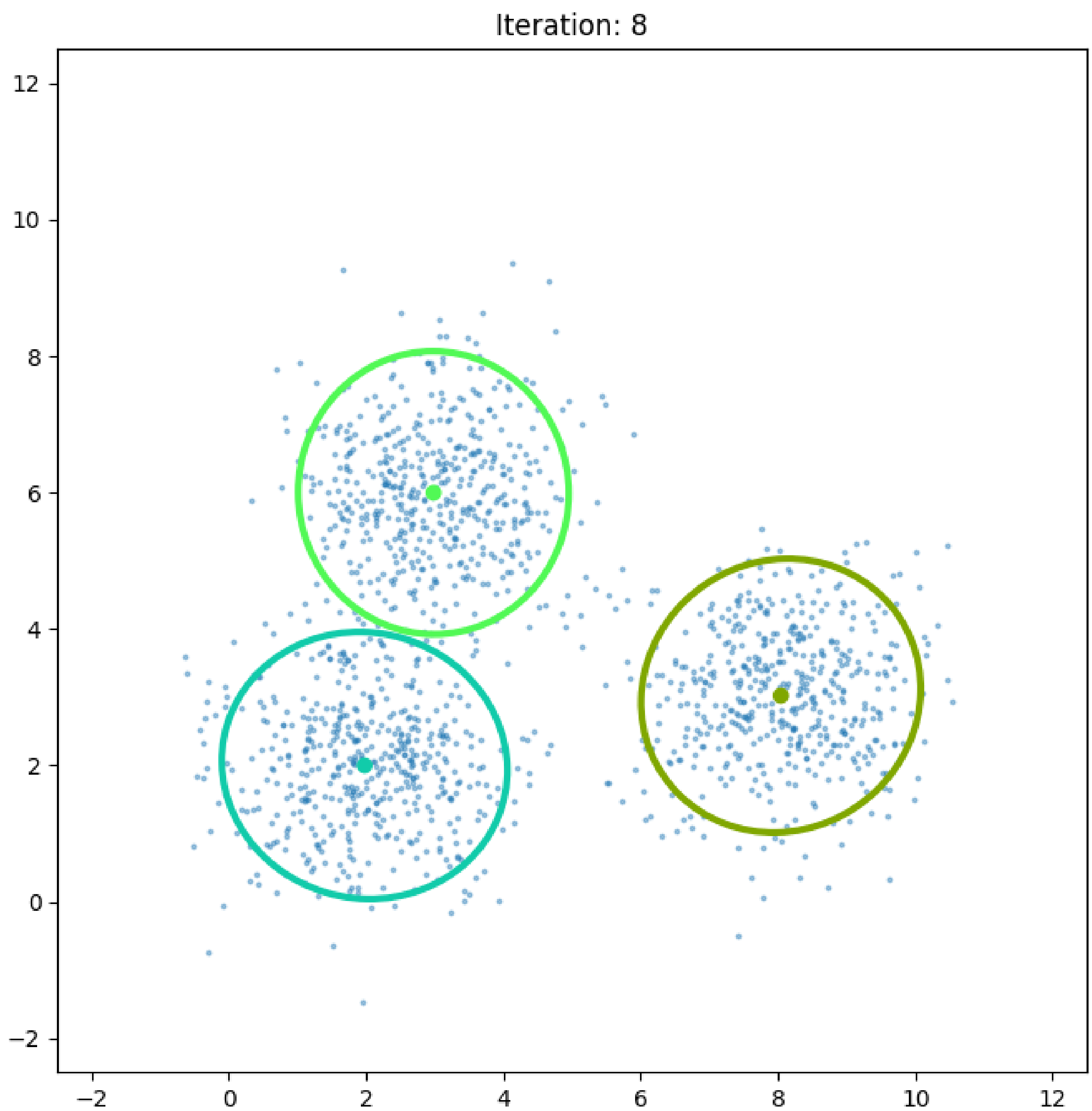
<Figure size 640x480 with 0 Axes>



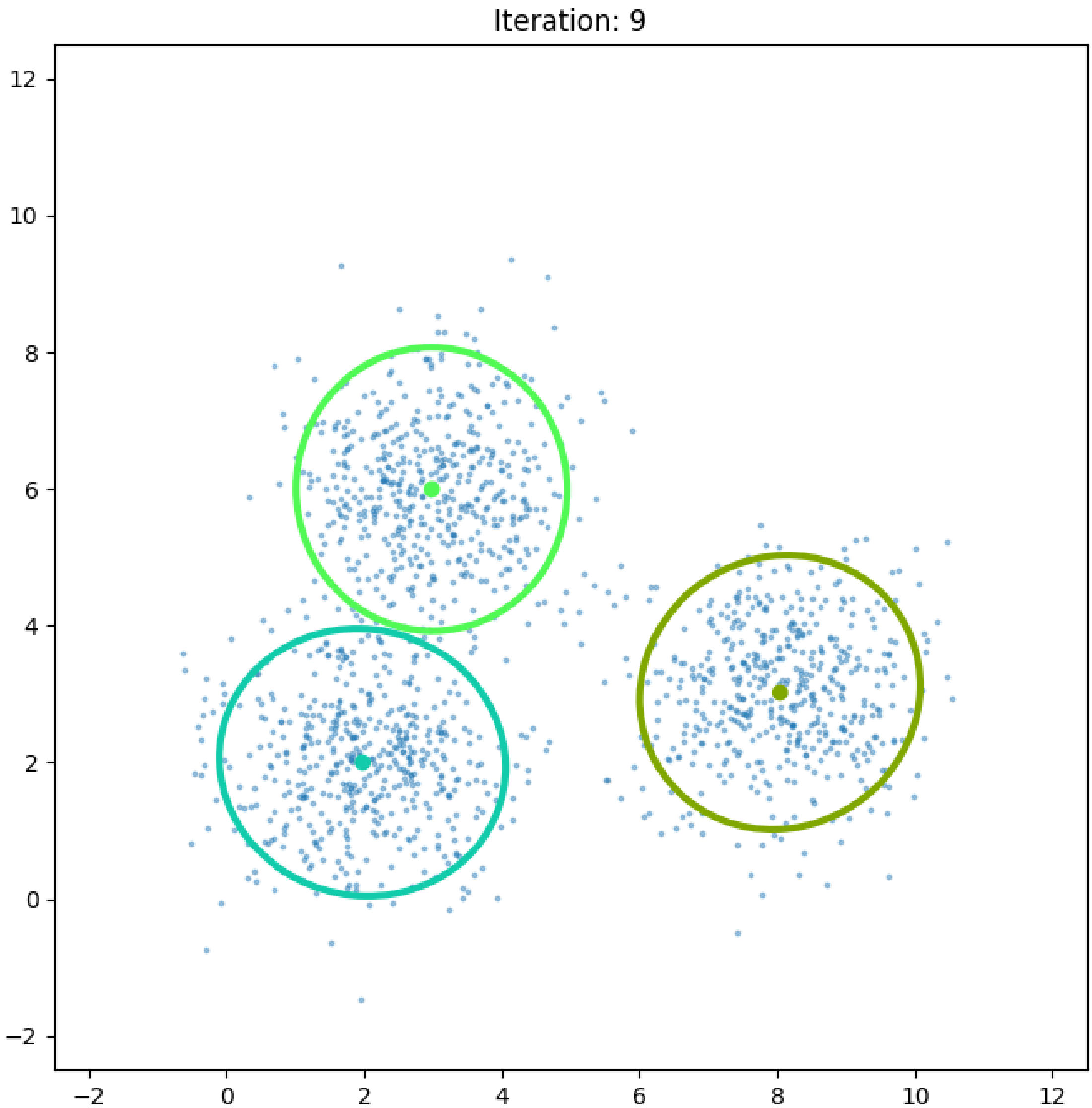

```
Iteration: 7, log-likelihood: -5874.4840
<Figure size 640x480 with 0 Axes>
```




```
Iteration: 8, log-likelihood: -5874.4834
<Figure size 640x480 with 0 Axes>
```

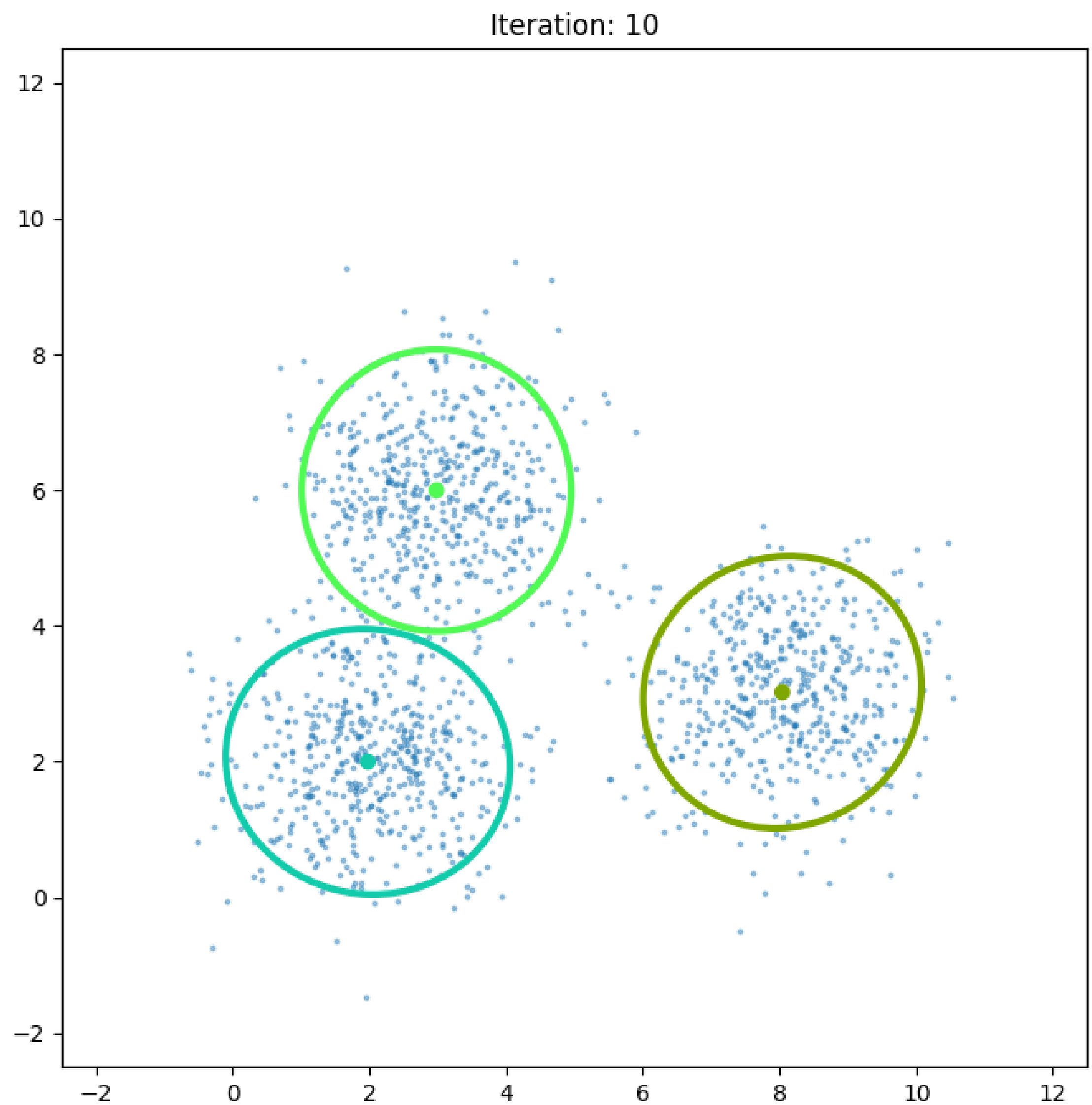



```
Iteration: 9, log-likelihood: -5874.4832
<Figure size 640x480 with 0 Axes>
```

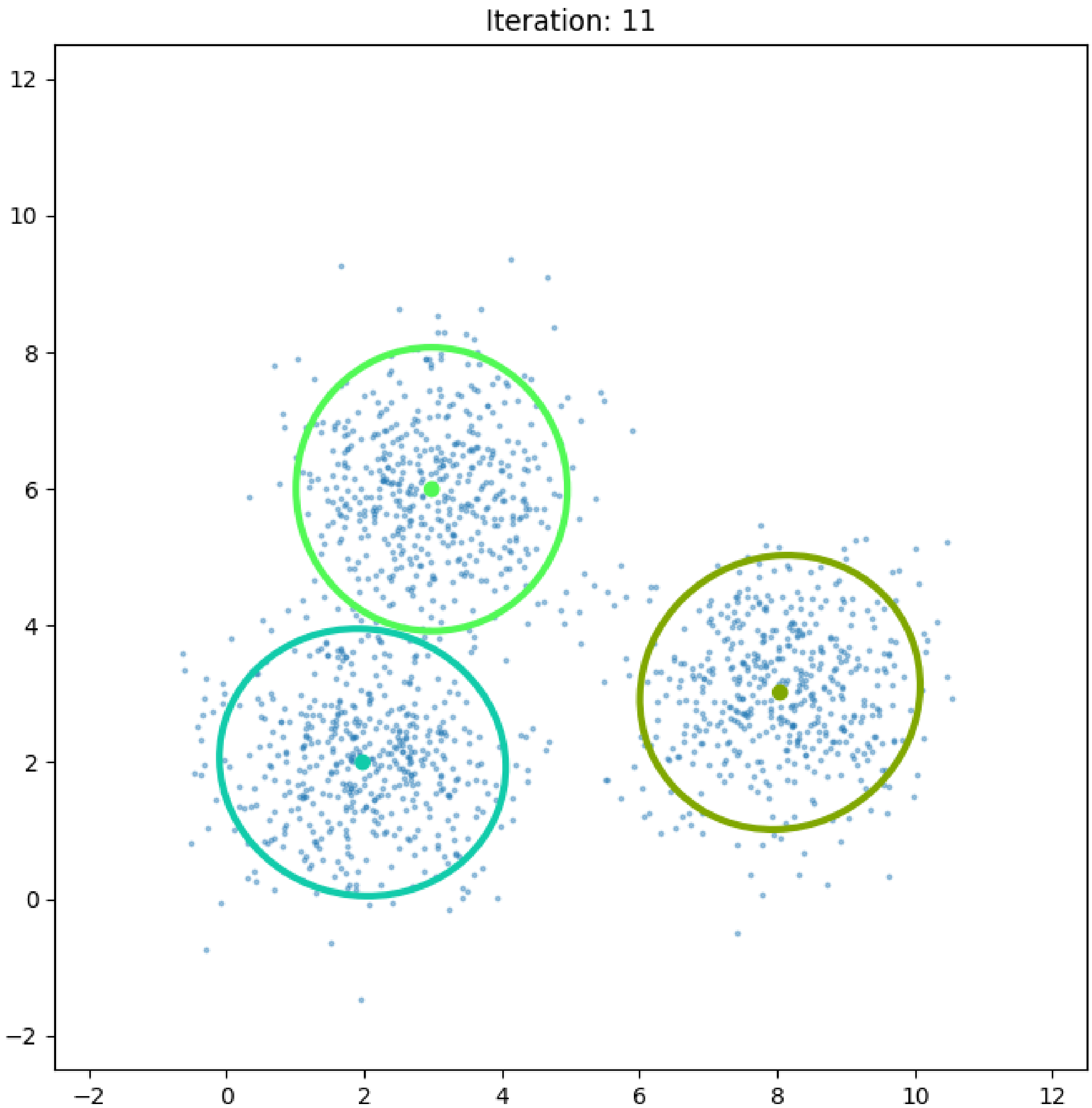



```
Iteration: 10, log-likelihood: -5874.4830
```

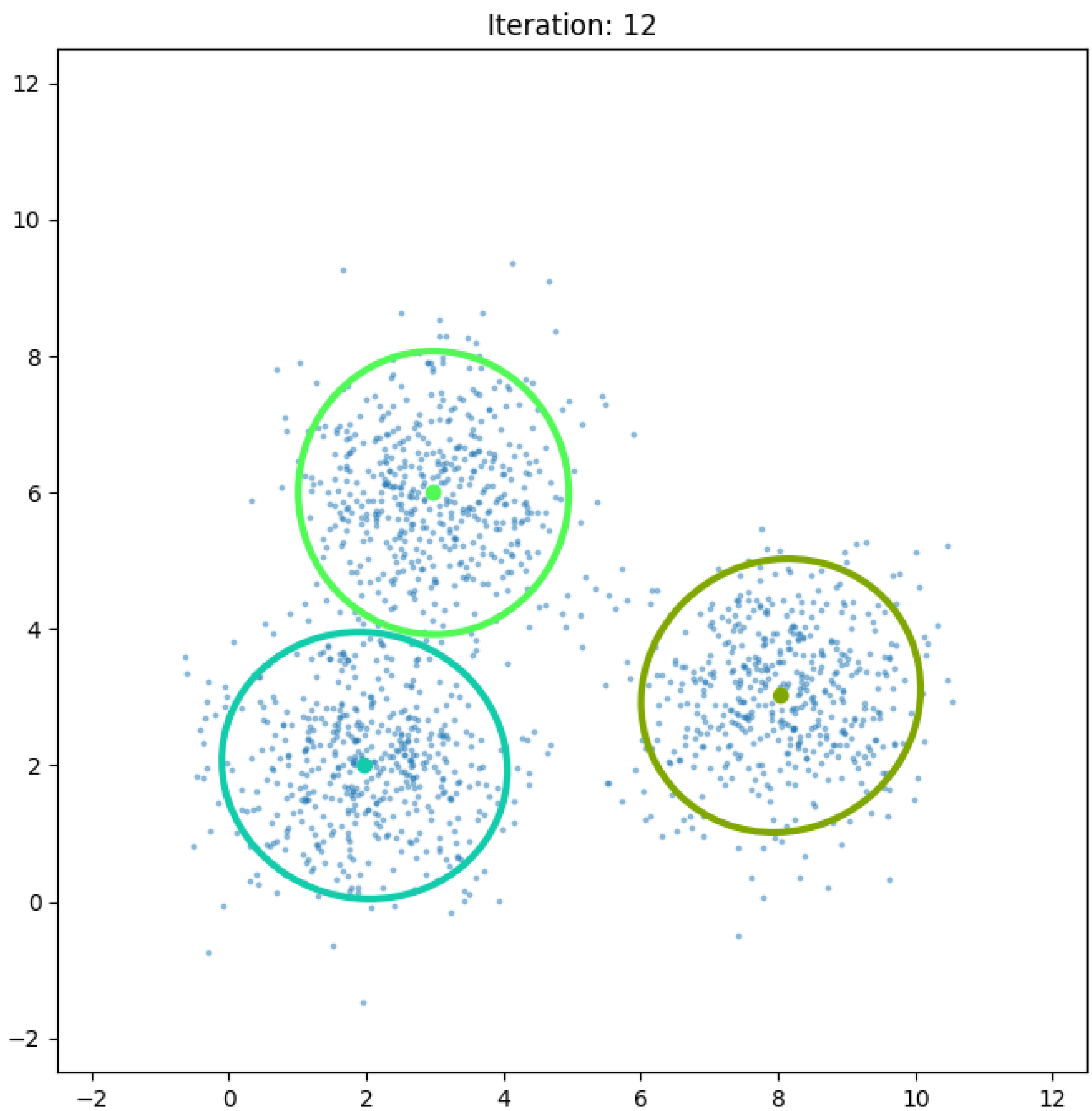
```
<Figure size 640x480 with 0 Axes>
```



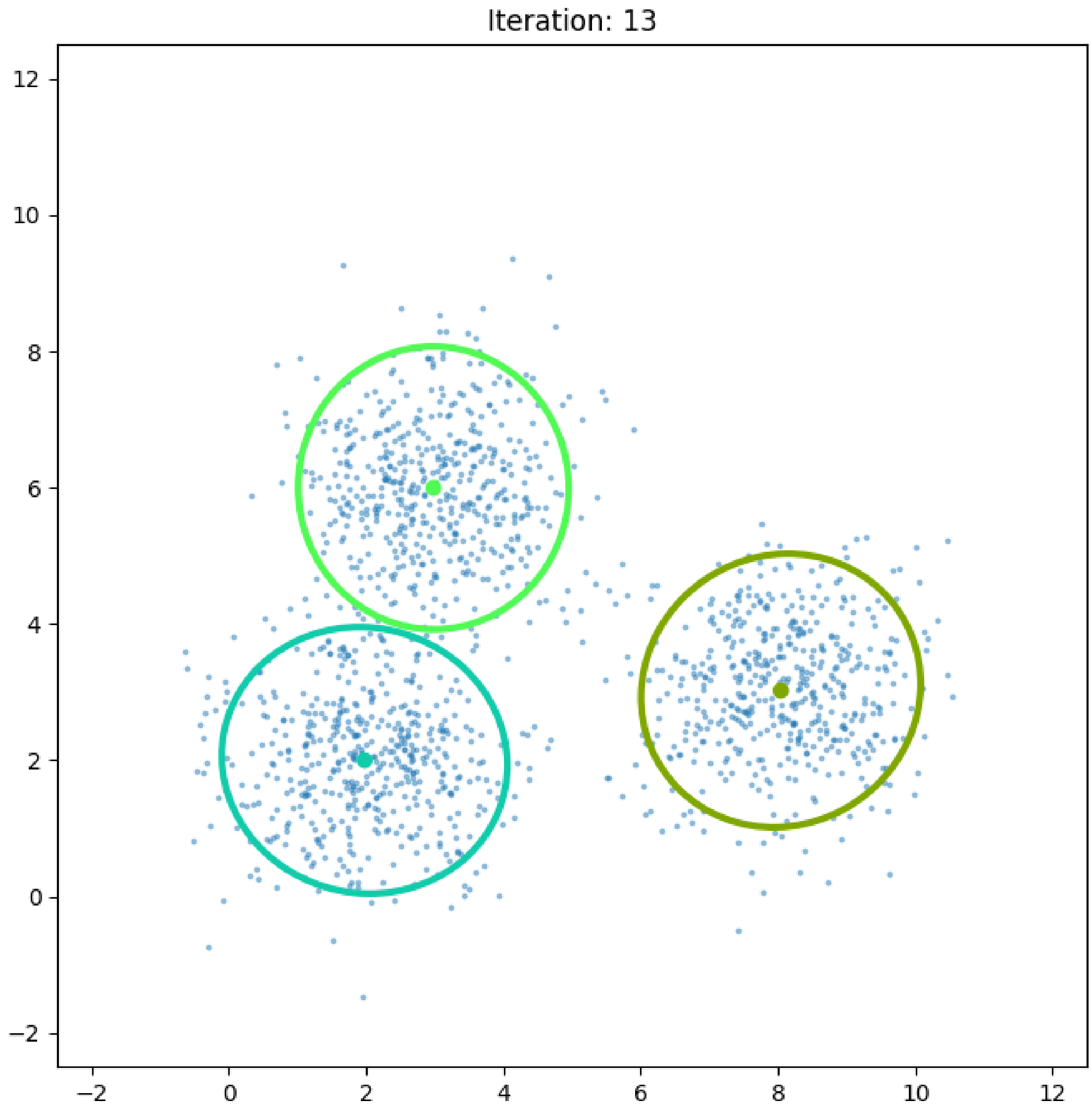

```
Iteration: 11, log-likelihood: -5874.4830
<Figure size 640x480 with 0 Axes>
```




```
Iteration: 12, log-likelihood: -5874.4830
<Figure size 640x480 with 0 Axes>
```

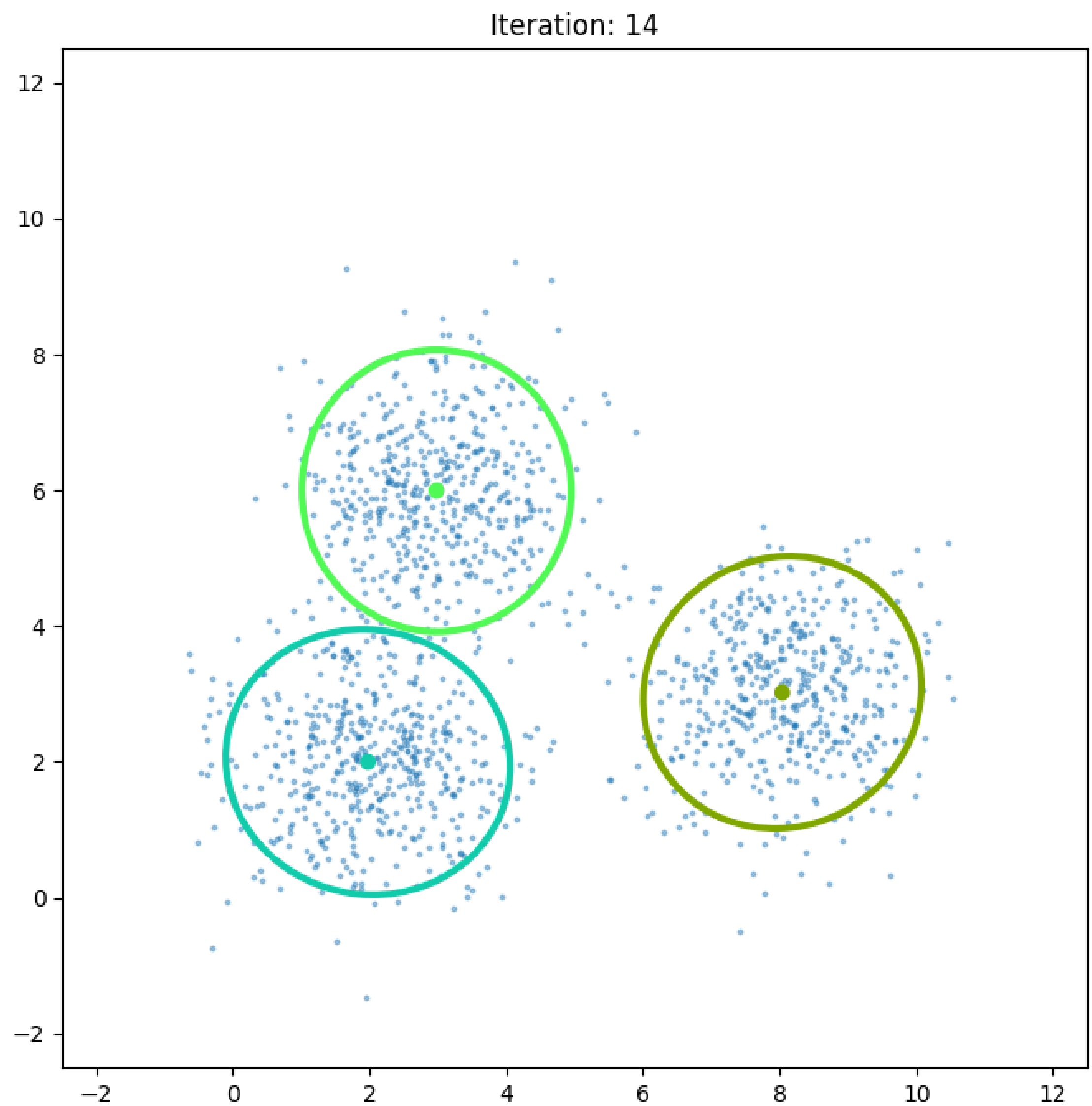



```
Iteration: 13, log-likelihood: -5874.4830
<Figure size 640x480 with 0 Axes>
```

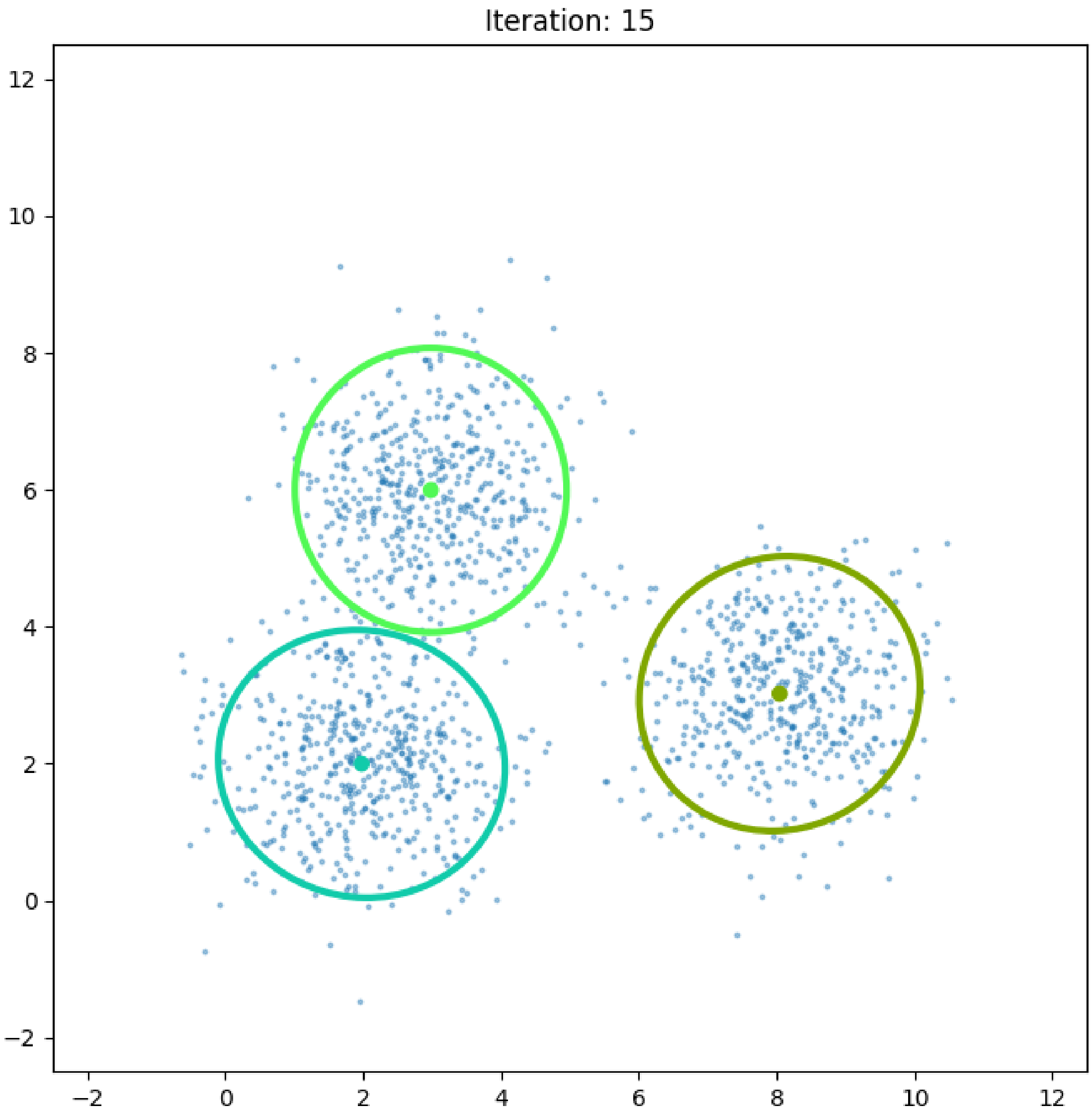


Iteration: 14, log-likelihood: -5874.4830

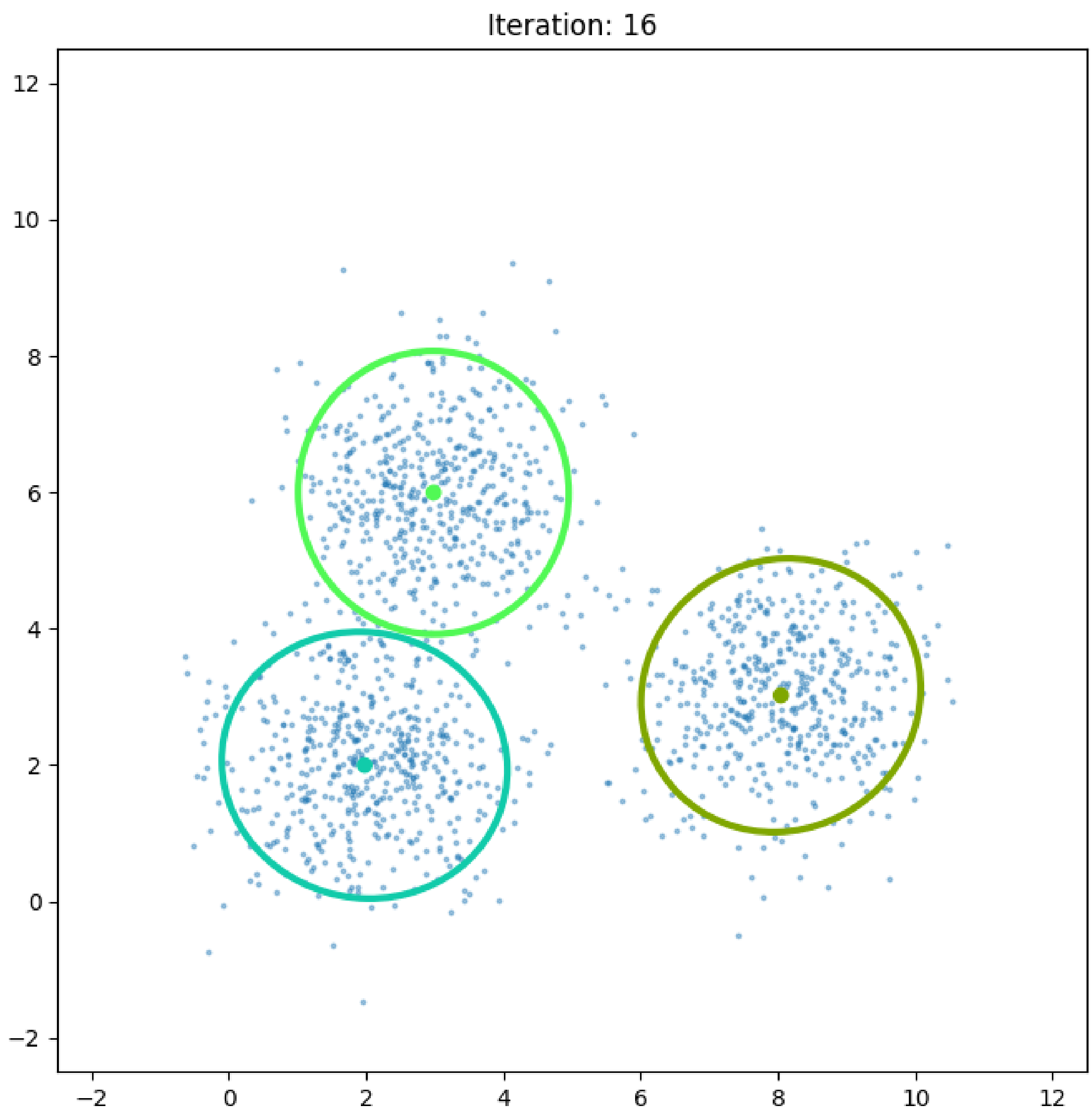
<Figure size 640x480 with 0 Axes>



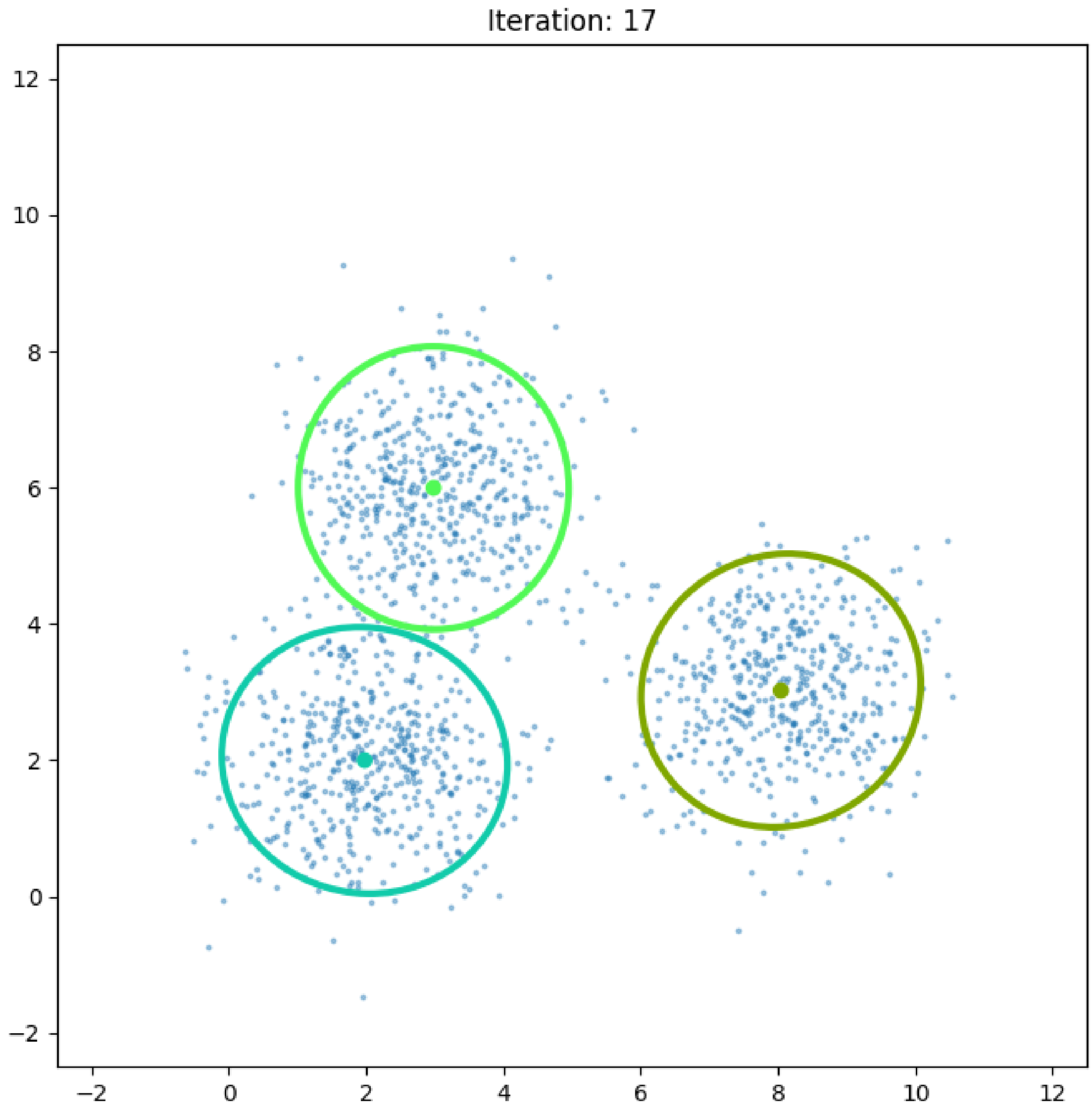

```
Iteration: 15, log-likelihood: -5874.4830
<Figure size 640x480 with 0 Axes>
```



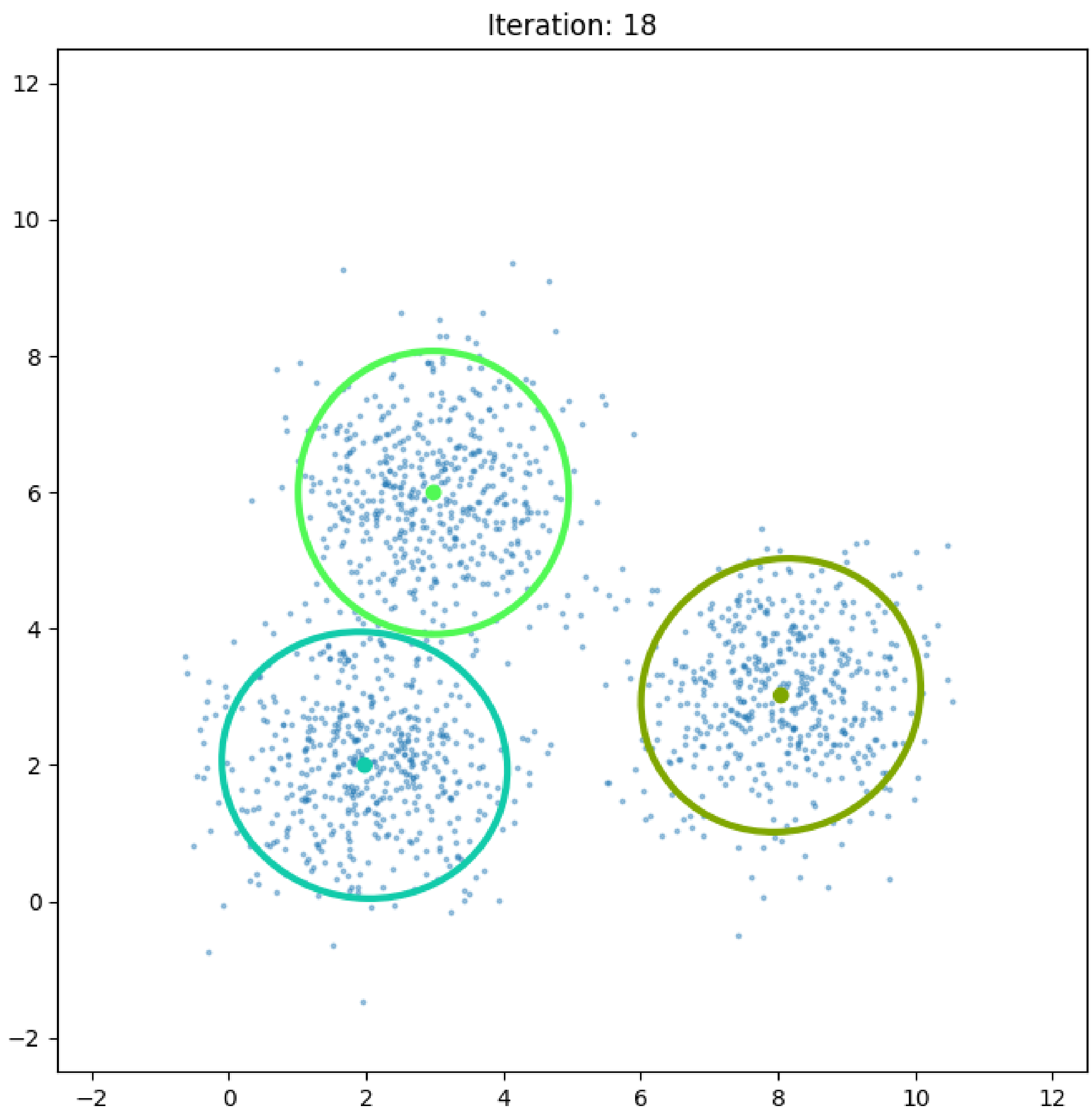

```
Iteration: 16, log-likelihood: -5874.4830
<Figure size 640x480 with 0 Axes>
```



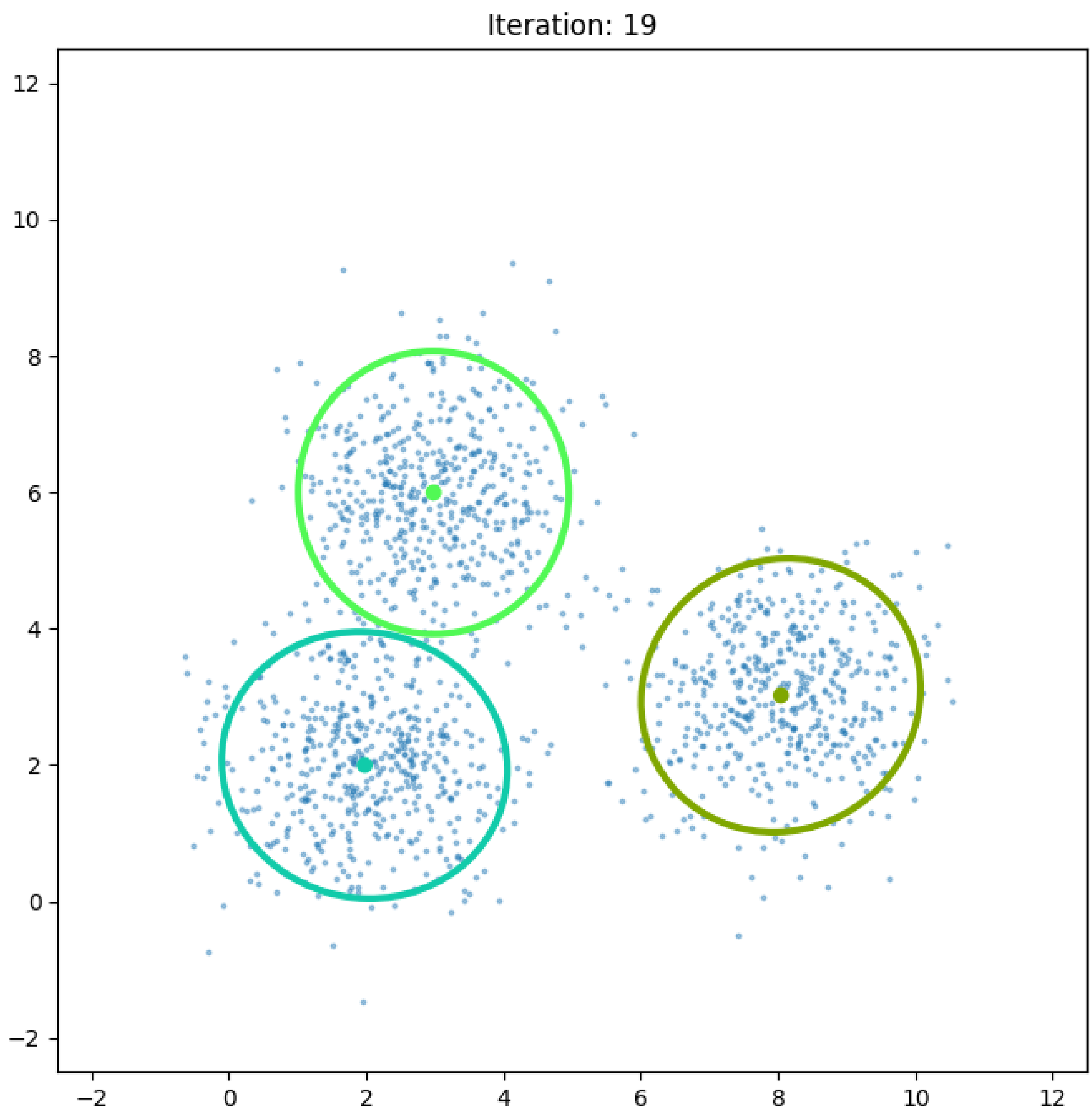

```
Iteration: 17, log-likelihood: -5874.4830
<Figure size 640x480 with 0 Axes>
```




```
Iteration: 18, log-likelihood: -5874.4830
<Figure size 640x480 with 0 Axes>
```

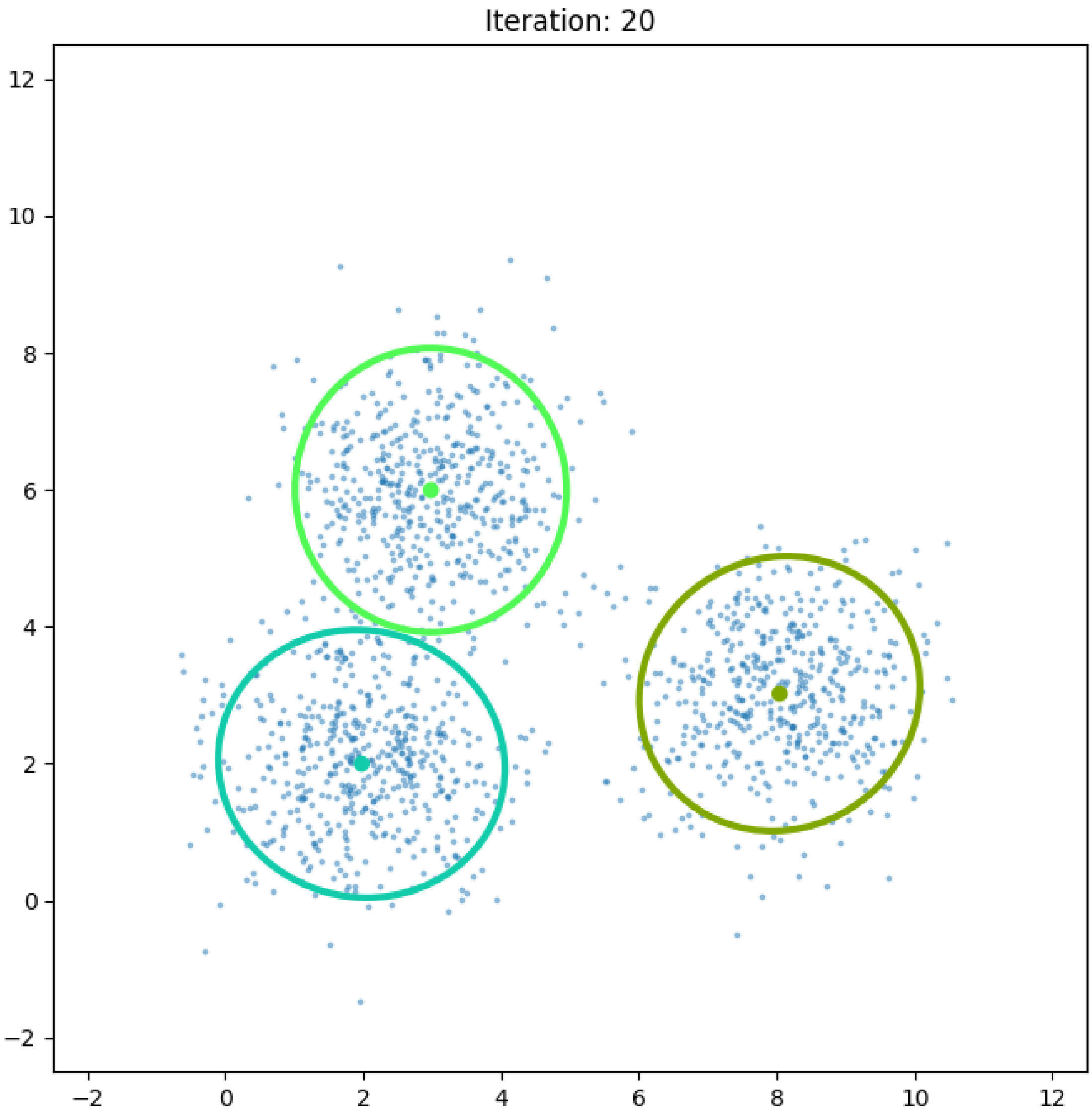



```
Iteration: 19, log-likelihood: -5874.4830
<Figure size 640x480 with 0 Axes>
```



Iteration: 20, log-likelihood: -5874.4830

<Figure size 640x480 with 0 Axes>




```
<Figure size 640x480 with 0 Axes>
```

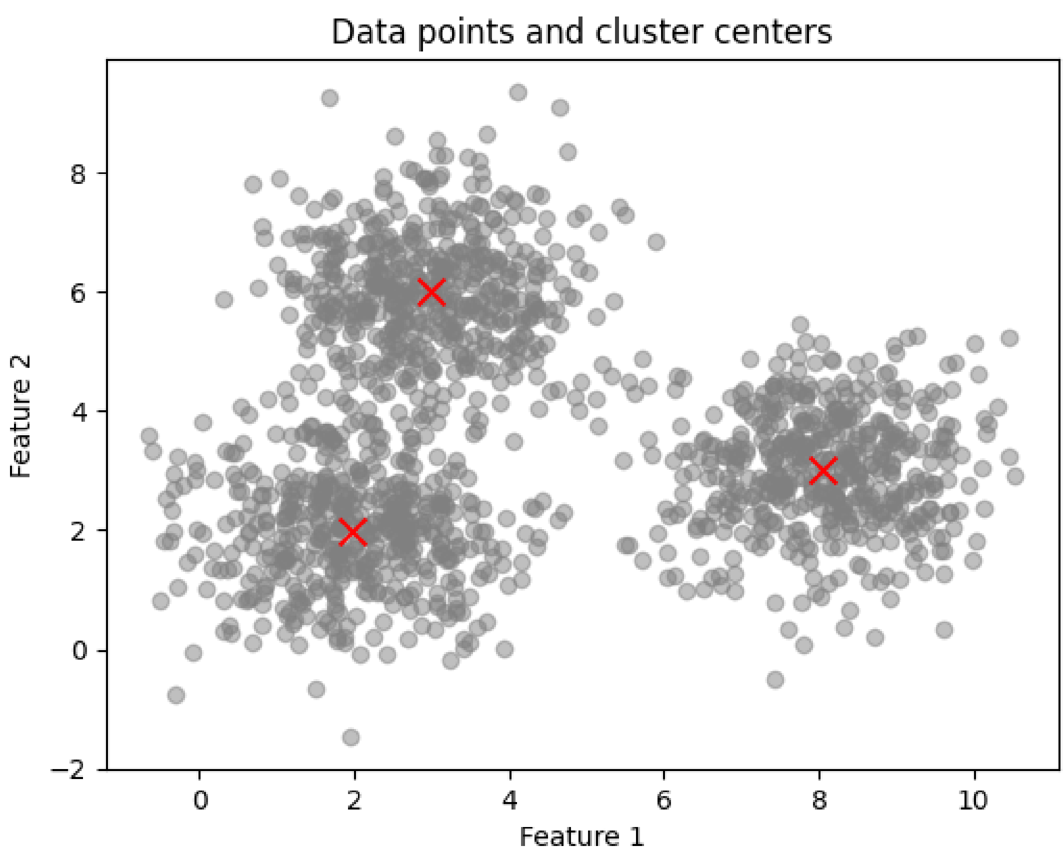
Nhận xét: Từ vòng lặp thứ 10 thì log-likelihood ngừng thay đổi.

[ ]:
```python
# Get cluster centers
cluster_centers = gmm.mu
print("Cluster centers:")
print(cluster_centers)
```

```
Cluster centers:
[[8.04122305 3.02189219]
 [2.98191289 5.99579256]
 [1.97775702 1.99782045]]
```

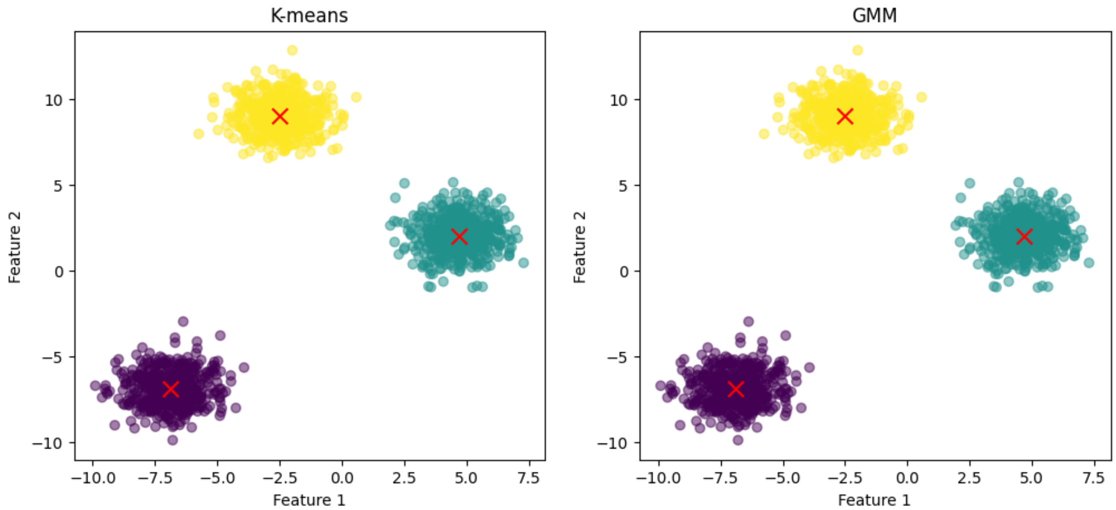
So sánh với kết quả tâm cụm khi dùng K-means ta thấy:
Centers found by our algorithm:
[[2.99084705 6.04196062]
[1.97563391 2.01568065]
[8.03643517 3.02468432]]
Có sự khác biệt không quá đáng kể.
Lý do là vì data có phân phối dạng cầu, hai phương pháp cho kết quả tương tự nhau. Tuy nhiên, GMM có thể cung cấp thông tin thêm về mật độ xác suất của dữ liệu trong mỗi cụm.

[ ]:
```
# Plot data and cluster centers
plt.scatter(X[:, 0], X[:, 1], c='gray', alpha=0.5)
plt.scatter(cluster_centers[:, 0], cluster_centers[:, 1], c='red', marker='x',␣
↪s=100)
plt.title('Data points and cluster centers')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```



Thử chứng minh khẳng định thông qua kết quả dùng thư viện

[ ]:
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture

# Tạo dữ liệu có phân phối cầu
X, _ = make_blobs(n_samples=1500, centers=3, cluster_std=1.0, random_state=42)

# Sử dụng K-means
kmeans = KMeans(n_clusters=3, random_state=0)
kmeans.fit(X)
kmeans_centers = kmeans.cluster_centers_

# Sử dụng GMM
gmm = GaussianMixture(n_components=3, random_state=0)
gmm.fit(X)
gmm_centers = gmm.means_

# Vẽ biểu đồ
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.scatter(X[:, 0], X[:, 1], c=kmeans.labels_, cmap='viridis', alpha=0.5)
plt.scatter(kmeans_centers[:, 0], kmeans_centers[:, 1], c='red', marker='x',␣
↪s=100)
plt.title('K-means')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')

plt.subplot(1, 2, 2)
plt.scatter(X[:, 0], X[:, 1], c=gmm.predict(X), cmap='viridis', alpha=0.5)
plt.scatter(gmm_centers[:, 0], gmm_centers[:, 1], c='red', marker='x', s=100)
plt.title('GMM')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')

plt.show()
```



2.2 Sử dụng thư viện sk-learn

2.2.1 Ví dụ triển khai lý thuyết

(Thông tin về chỉ số mua sắm và mức thu nhập)

Đọc dữ liệu

[ ]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import itertools
from scipy import linalg
import matplotlib.pyplot as plt
import matplotlib.patheffects as PathEffects
from matplotlib.patches import Ellipse
from sklearn.preprocessing import MinMaxScaler
from sklearn.mixture import GaussianMixture
# Thư viện chứa model Gaussian Mixture
```

[ ]:
```
data = pd.read_csv("data/shopping-data.csv", header=0, index_col=0)
print(data.shape)
data.head()

# Lấy ra thu nhập va điểm shopping
X = data.iloc[:, 2:4].values

# Chuẩn hoá dữ liệu
std = MinMaxScaler()
X_std = std.fit_transform(X)
print(X_std.shape)
```

```
(200, 4)
(200, 2)
```

Khởi tạo Gaussian Mix model & fit data

[ ]:
```
# Khởi tạo đối tượng mô hình GaussianMixture
gm = GaussianMixture(n_components=5, covariance_type="full", random_state=0)
gm.fit(X_std)
print("means: \n", gm.means_)
print("covariances: \n ", gm.covariances_)
```

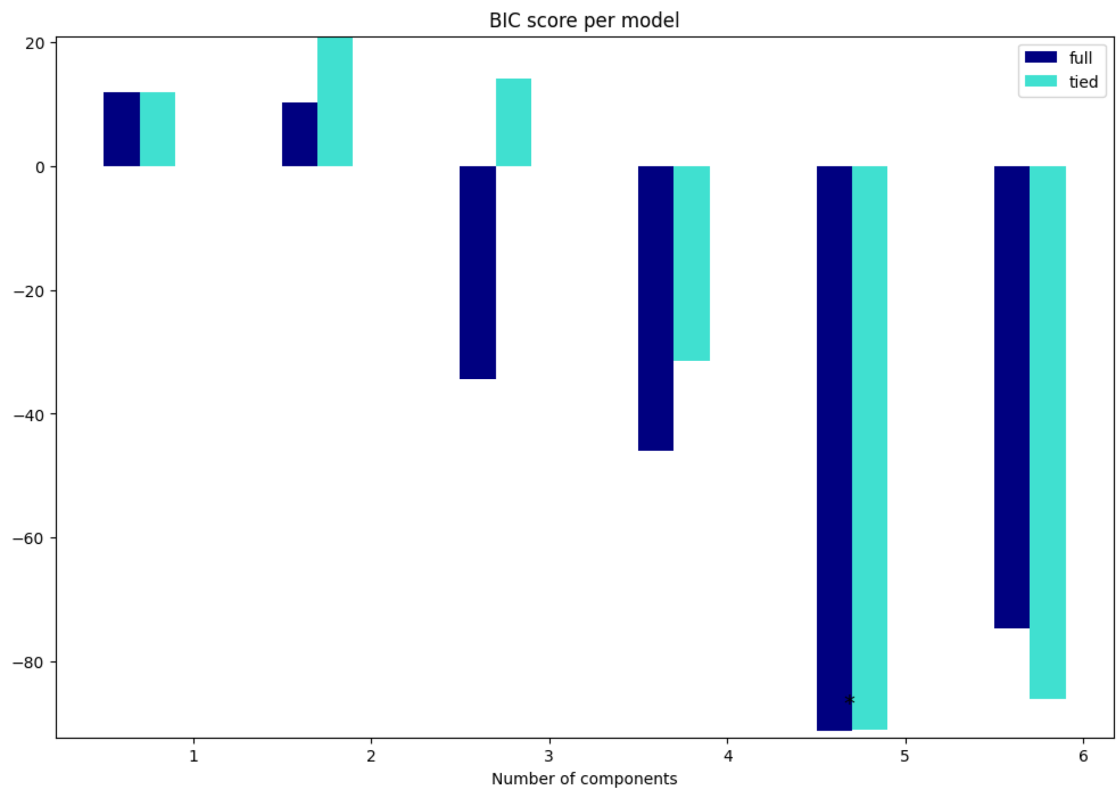
```
means:
 [[0.60502531 0.15433196]
 [0.33368985 0.49394756]
 [0.58393969 0.82673863]
 [0.0829305  0.80743088]
 [0.09861098 0.21597752]]
covariances:
  [[[ 0.01818446  0.00433814]
  [ 0.00433814  0.00873064]]

 [[ 0.00613567 -0.00231927]
  [-0.00231927  0.0051635 ]]

 [[ 0.01808598 -0.00031096]
  [-0.00031096  0.0091568 ]]

 [[ 0.00337483 -0.0001437 ]
  [-0.0001437   0.01026088]]

 [[ 0.00453005  0.00255303]
  [ 0.00255303  0.01918353]]]
```

Tìm số cụm phân chia hợp lý nhất Bản chất trong code này là thử với 1 tập các giá trị và chọn ra giá trị cho kết quả phân cụm tốt nhất

[ ]:
```
lowest_bic = np.infty
bic = []
n_components_range = range(1, 7)
# cv_types = ['spherical', 'tied', 'diag', 'full']
cv_types = ["full", "tied"]
for cv_type in cv_types:
    for n_components in n_components_range:
        # Fit Gaussian mixture theo phương pháp huấn luyện EM
        gmm = GaussianMixture(n_components=n_components,␣
 ↪covariance_type=cv_type)
        gmm.fit(X_std)
        bic.append(gmm.bic(X_std))
```

```
        # Gán model có BIC scores thấp nhất là model tốt nhất
        if bic[-1] < lowest_bic:
            lowest_bic = bic[-1]
            best_gmm = gmm

bic = np.array(bic)
color_iter = itertools.cycle(["navy", "turquoise"])
clf = best_gmm
bars = []

# Vẽ biểu đồ BIC scores
plt.figure(figsize=(12, 8))
for i, (cv_type, color) in enumerate(zip(cv_types, color_iter)):
    xpos = np.array(n_components_range) + 0.2 * (i - 2)
    bars.append(
        plt.bar(
            xpos,
            bic[i * len(n_components_range) : (i + 1) *␣
 ↪len(n_components_range)],
            width=0.2,
            color=color,
        )
    )
plt.xticks(n_components_range)
plt.ylim([bic.min() * 1.01 - 0.01 * bic.max(), bic.max()])
plt.title("BIC score per model")
xpos = (
    np.mod(bic.argmin(), len(n_components_range))
    + 0.65
    + 0.2 * np.floor(bic.argmin() / len(n_components_range))
)
plt.text(xpos, bic.min() * 0.97 + 0.03 * bic.max(), "*", fontsize=14)
plt.xlabel("Number of components")
plt.legend([b[0] for b in bars], cv_types)
```

```
[ ]: <matplotlib.legend.Legend at 0x7c6421c5c110>
```

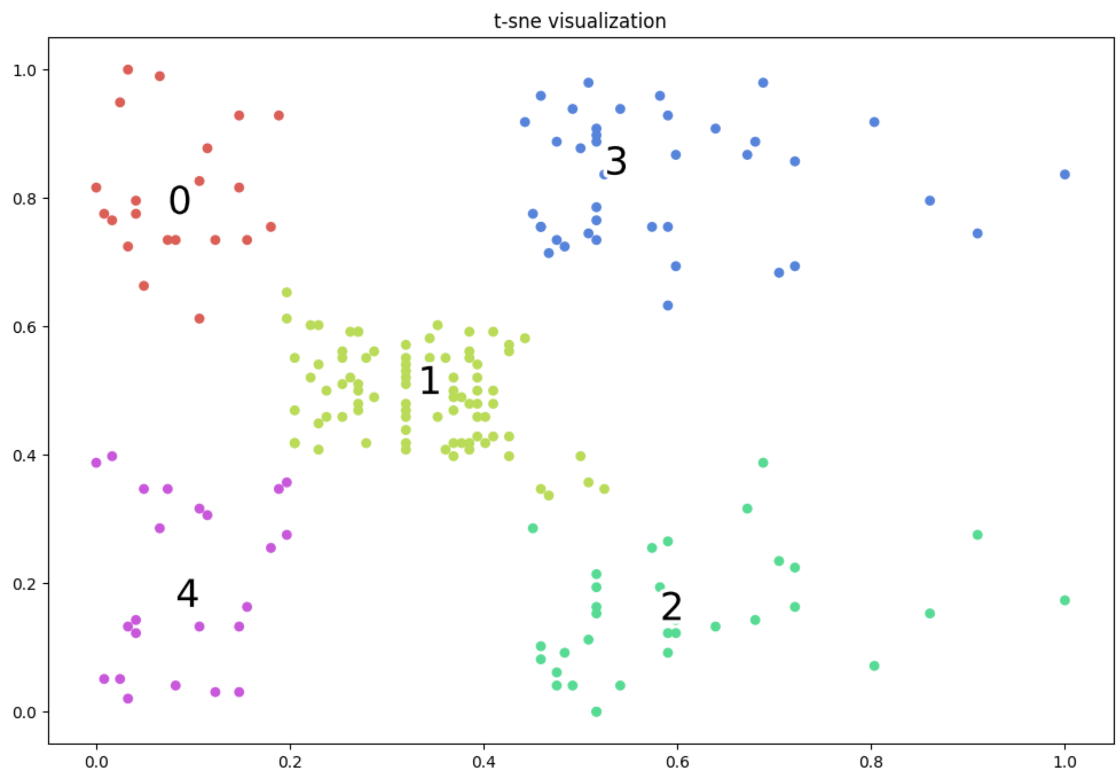


Trực quan hóa kết quả

```
[ ]: def _plot_kmean_scatter(X, labels):
         '''
         X: dữ liệu đầu vào
         labels: nhãn dự báo
         '''
         # lựa chọn màu sắc
         num_classes = len(np.unique(labels))
         palette = np.array(sns.color_palette("hls", num_classes))

         # vẽ biểu đồ scatter
         fig = plt.figure(figsize=(12, 8))
         ax = plt.subplot()
         sc = ax.scatter(X[:,0], X[:,1], lw=0, s=40,
         c=palette[labels.astype(int)])

         # thêm nhãn cho mỗi cluster
         txts = []

         for i in range(num_classes):
             # Vẽ text tên cụm tại trung vị của mỗi cụm
```

```
        xtext, ytext = np.median(X[labels == i, :], axis=0)
        txt = ax.text(xtext, ytext, str(i), fontsize=24)
        txt.set_path_effects([
            PathEffects.Stroke(linewidth=5, foreground="w"),
            PathEffects.Normal()])
        txts.append(txt)
    plt.title('t-sne visualization')
```

[ ]:
```
labels = best_gmm.predict(X_std)
_plot_kmean_scatter(X_std, labels)
```



2.2.2 Ví dụ mở rộng

(Thực hiện tìm số cụm tối ưu cho bài toán X nhiều chiều)

[ ]:
```
data = pd.read_csv("data/Sales_Transactions_Dataset_Weekly.csv", header=0,␣
 ↪index_col=0)
print(data.shape)
data.head()

# Lấy ra từ week0 - cột 1 đến week51 - cột 52
# (có thể dùng từ cột 53 cũng được nhưng sẽ chọn cách này)
```

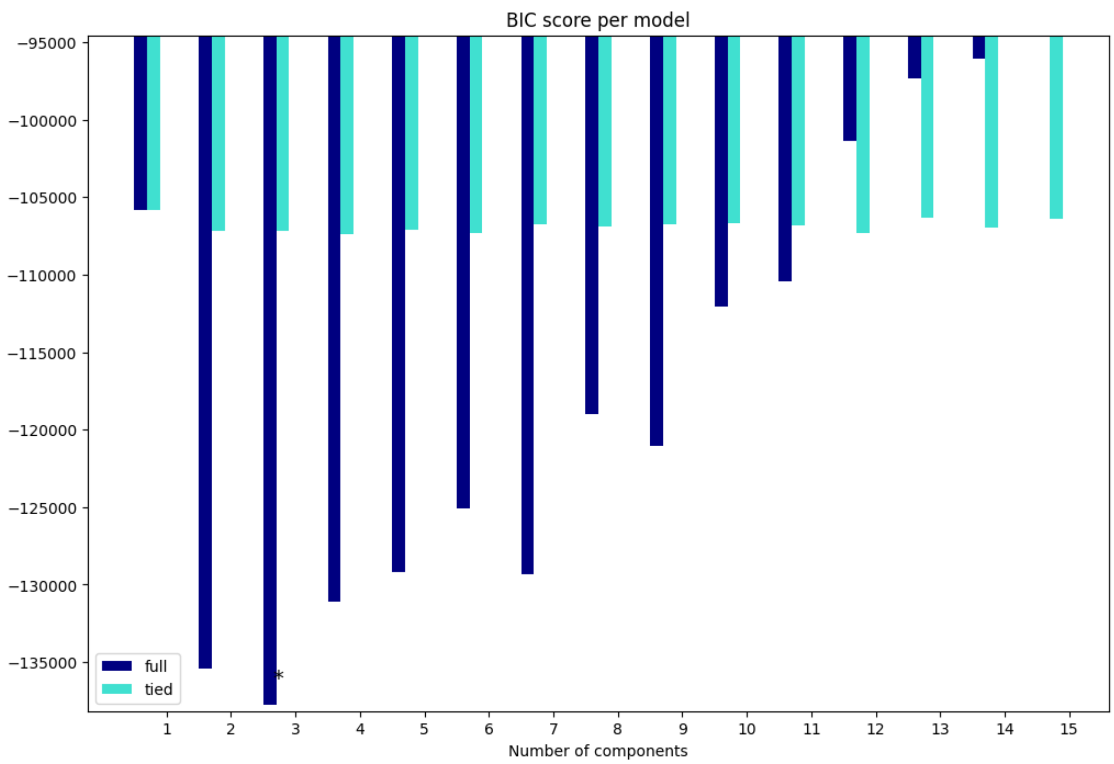
```python
X = data.iloc[:, 1:53].values

# Chuẩn hoá dữ liệu
std = MinMaxScaler()
X_std = std.fit_transform(X)
print(X_std.shape)
```

```
(811, 106)
(811, 52)
```

[ ]:
```python
lowest_bic = np.infty
bic = []
n_components_range = range(1, 16)
# cv_types = ['spherical', 'tied', 'diag', 'full']
cv_types = ["full", "tied"]
for cv_type in cv_types:
    for n_components in n_components_range:
        # Fit Gaussian mixture theo phương pháp huấn luyện EM
        gmm = GaussianMixture(n_components=n_components,␣
 ↪covariance_type=cv_type)
        gmm.fit(X_std)
        bic.append(gmm.bic(X_std))
        # Gán model có BIC scores thấp nhất là model tốt nhất
        if bic[-1] < lowest_bic:
            lowest_bic = bic[-1]
            best_gmm = gmm

bic = np.array(bic)
color_iter = itertools.cycle(["navy", "turquoise"])
clf = best_gmm
bars = []

# Vẽ biểu đồ BIC scores
plt.figure(figsize=(12, 8))
for i, (cv_type, color) in enumerate(zip(cv_types, color_iter)):
    xpos = np.array(n_components_range) + 0.2 * (i - 2)
    bars.append(
        plt.bar(
            xpos,
            bic[i * len(n_components_range) : (i + 1) *␣
 ↪len(n_components_range)],
            width=0.2,
            color=color,
        )
    )
plt.xticks(n_components_range)
plt.ylim([bic.min() * 1.01 - 0.01 * bic.max(), bic.max()])
```

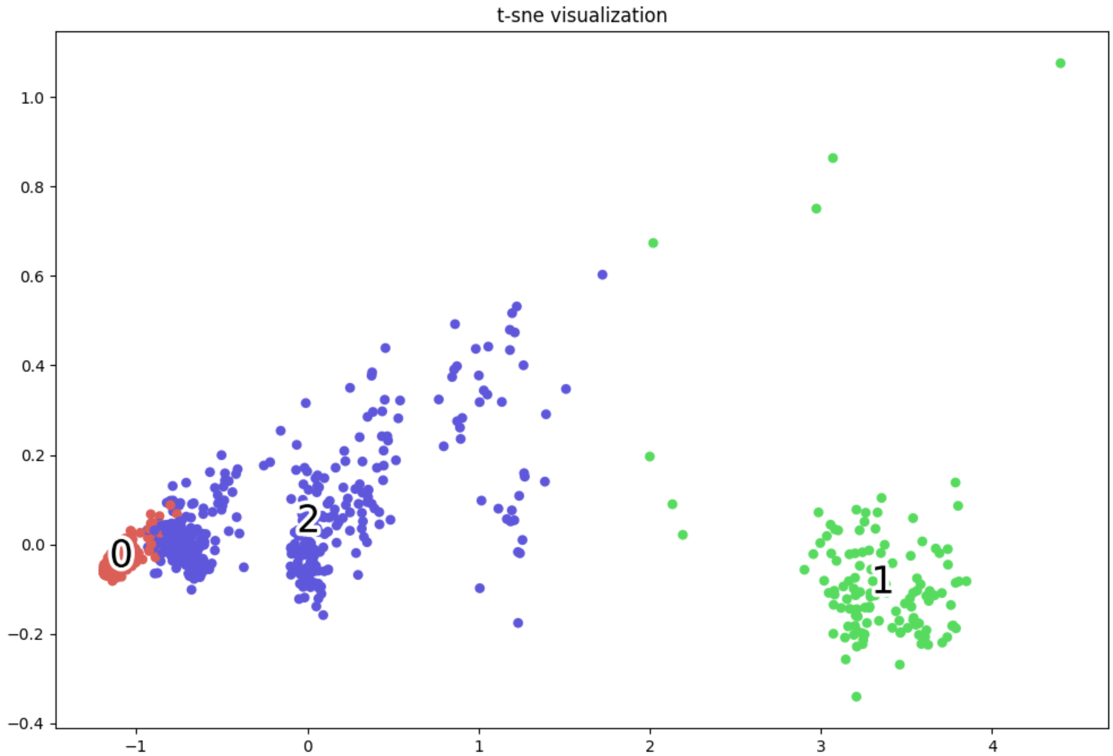
```
plt.title("BIC score per model")
xpos = (
    np.mod(bic.argmin(), len(n_components_range))
    + 0.65
    + 0.2 * np.floor(bic.argmin() / len(n_components_range))
)
plt.text(xpos, bic.min() * 0.97 + 0.03 * bic.max(), "*", fontsize=14)
plt.xlabel("Number of components")
plt.legend([b[0] for b in bars], cv_types)
```

[ ]: <matplotlib.legend.Legend at 0x7b1e60625510>



[ ]:
```
from sklearn.decomposition import PCA

# Áp dụng PCA để giảm chiều dữ liệu xuống còn 2D
pca = PCA(n_components=2)
X_std_pca2D = pca.fit_transform(X_std)
```

[ ]:
```
labels = best_gmm.predict(X_std)
_plot_kmean_scatter(X_std_pca2D, labels)
```



Như vậy các mặt hàng nhìn chung có thể được phân vào 3 cụm (mặc dù đề bài cho số cụm tối ưu nói trong [5, 15] tuy nhiên test theo cách lấy X theo tuần (week 0 đến week 51 - cột 1 đến cột 52) thì 3 mới là số cụm tối ưu) dựa trên tiêu chí lượng giao dịch đã cho trong dữ liệu.