GaussianMixture_KMeans_unsupervised

December 21, 2023

Before you turn this problem in, make sure everything runs as expected. First, **restart the kernel** (in the menubar, select Kernel→Restart) and then **run all cells** (in the menubar, select Cell→Run All).

Make sure you fill in any place that says `YOUR CODE HERE` or "YOUR ANSWER HERE" and remove every line containing the expression: "raise …" (if you leave such a line your code will not run).

Do not remove any cell from the notebook you downloaded. You can add any number of cells (and remove them if not more necessary).

0.1 IMPORTANT: make sure to rerun all the code from the beginning to obtain the results for the final version of your notebook, since this is the way we will do it before evaluating your notebook!!!

Fill in your name and id number (numero matricola) below:

[1]:
```
NAME = "Tommaso Bergamasco"
ID_number = int("2052409")

import IPython
assert IPython.version_info[0] >= 3, "Your version of IPython is too old,␣
↪please update it."
```

---

1 HOMEWORK #4

1.1 Unsupervised learning

In this notebook we are going to explore the use of unsupervised clustering methods.

[2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from scipy.stats import multivariate_normal
from sklearn.datasets import make_blobs
```

[3]:
```python
# TODO 1: Write a function to compute the probability density function (pdf) of␣
↪a gaussian random vector:
# you just need to apply its definition.

def gv_normalizing_const(sigma : np.ndarray) -> np.float64:
    '''
    Function to compute the normalization coefficient of a vector valued␣
↪Gaussian distribution.
    :param sigma: Covariance of the Gaussian random vector (d x d Positive␣
↪Definite matrix).
    '''
    # YOUR CODE HERE

    normalizing_const = 1/np.sqrt(np.linalg.det(2*np.pi*sigma))

    return normalizing_const

def gaussian_pdf(x : np.ndarray, mu : np.ndarray, sigma : np.ndarray) -> np.
↪ndarray:
    '''
    Function to compute the pdf of a vector valued gaussian distribution on the␣
↪location x given its parameters,
    mu and sigma. We are assuming sigma is invertible (you do not need to check␣
↪its invertibility). For simplicity
    return the pdf with shape (1,1), this should be the shape you get after the␣
↪quadratic form computation.
    '''
    # YOUR CODE HERE

    # I Compute separately some useful factors for the final expression of the␣
↪pdf:
    x_mu = x - mu
    sigma_inv = np.linalg.inv(sigma)

    # Computing the unnormalized pdf:
    unnormalized_pdf = np.exp(-(1/2) * x_mu.T @ sigma_inv @ x_mu)

    return gv_normalizing_const(sigma) * unnormalized_pdf
```

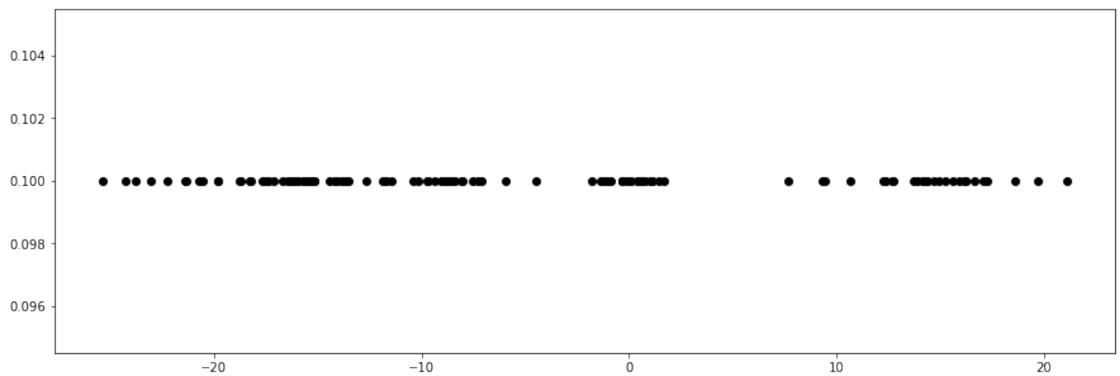
[4]:
```python
assert type(gv_normalizing_const(np.array([[2.]]))) == np.float64
# Test on scalar case
x = np.array([[1.]])
mean, cov = np.array([[2.5]]), np.array([[0.5]])
assert np.isclose(multivariate_normal.pdf(1, mean=mean, cov=cov),␣
↪gaussian_pdf(x, mean, cov), atol=1e-3)
```

```
# Test on vector valued case
mean, temp = np.random.normal(2, 3, 2).reshape(-1,1), np.random.normal(2, 3, 4).
 ↪reshape(2,2)
x = np.random.normal(2, 3, 2).reshape(-1,1)
cov = temp @ temp.T
hand_pdf = gaussian_pdf(x, mean, cov)
scipy_pdf = multivariate_normal.pdf(x.reshape(1,-1)[0], mean.reshape(1,-1)[0],␣
 ↪cov)
assert np.isclose(hand_pdf, scipy_pdf, atol=1e-4)
```

Let's load a 1-D dataset which does not contain any label. It has been generated using K clusters.

Can you tell how many clusters have been used by looking at the scatter plot?

[5]:
```
url = 'https://raw.githubusercontent.com/LucaZancato/ML2020-2021/main/HW_4/
 ↪1_D_dataset.csv'
data = np.array(pd.read_csv(url, sep=';'))

plt.figure(figsize=(15,5))
for i, x in enumerate(data):
    plt.scatter(x, 0.1, color='k')
```



It's not trivial to understand how many clusters are present. Such an issue is present also in the case of Expectation-Maximization (EM) on Gaussian Mixture Models (GMM) (the one we are going to implement) and K-means: for both of these algorithms the number of clusters is fixed a priori, it is a parameter (hyper-parameter) we must decide before processing any data.

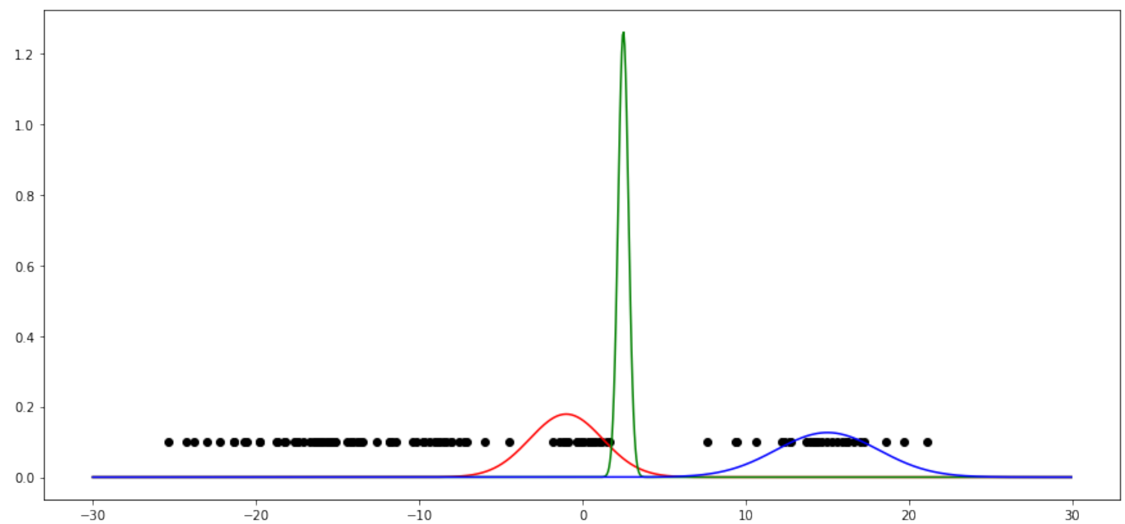
Usually, in order to achieve satisfactory clustering, one needs to try with different number of classes and validate which is the best number.

For now let's make it simple and let's consider the dataset has 3 clusters.

In the following cell we shall parametrize 3 Gaussian random variables specifiying both means, covariances and mixing probabilities. Take a moment to understand the way these parameters are stored, since the EM implementation is built on this notation. For now, means, cov and mixing probabilities are choosen without any specific criteria: we will see how EM applied to GMM will provide us a suboptimal set of means, covariances and mixing probabilities.

[6]:
```
means = np.array([[-1],[2.5],[15]])       # shape (K,d)
covs  = np.array([[[5]],[[0.1]],[[10]]]) # shape (K,d,d)
pi    = np.array([1/2, 1/4, 1/4])        # shape (K,)

X = np.linspace(-30, 30, 1000, endpoint=False).reshape(-1,1)

plt.figure(figsize=(15,7))
for i, x in enumerate(data):
    plt.scatter(x, 0.1, color='k')
plt.plot(X, [gaussian_pdf(x, means[0], covs[0]) for x in X], color='r')
plt.plot(X, [gaussian_pdf(x, means[1], covs[1]) for x in X], color='g')
plt.plot(X, [gaussian_pdf(x, means[2], covs[2]) for x in X], color='b')
```

[6]: [<matplotlib.lines.Line2D at 0x7f3230d07fa0>]



[7]:
```
# TODO 2: Write a function M_step which computes the M-step of the EM algorithm␣
↪under a GMM. Refer
# to the slides you can find on moodle (in which ALL the necessary steps and␣
↪expressions are present).
# Start by looking at the function "M_step" and then build all the helper␣
↪functions.

def update_pi(W : np.ndarray) -> np.ndarray:
    '''
    See M_step docs.
    '''
    # YOUR CODE HERE
```

```python
    # Recall we need a (K=3,) shape for the mixing prob. \pi, therefore for␣
↪every cluster
    # l=1,2,3 we must compute a \sum over i (fixed l):
    new_pi = (1/W.shape[0]) * np.sum(W, axis=0)

    return new_pi

def update_means(data : np.ndarray, W : np.ndarray) -> np.ndarray:
    '''
    See M_step docs.
    '''
    # YOUR CODE HERE

    # Similarly as above but considering that means has shape (K=3,d).
    # We must pay attention to respect the dimensions requirements in the␣
↪multiplications
    # and also to return a vector which is (K=3,d) itself.

    # the numerator of the eqn must be x_1@w_{l1}+x_2@w_{l2}+...+x_m@w_{lm}␣
↪where l is fixed
    numerator = np.zeros((W.shape[1],data.shape[1]))
    # Cycle the Clusters
    for j in range(W.shape[1]):
        # Cycle the data
        for i in range(data.shape[0]):
            numerator[j,:] = numerator[j,:] + data[i,:] * W[i,j]

    # The denominator is practically the same as the last function
    denominator = np.sum(W, axis=0).reshape((W.shape[1]),1)

    # Python recognizes the only possible dimension in which it's possible to␣
↪perform the division:
    new_means = numerator / denominator

    return new_means

def update_covs(data : np.ndarray, W : np.ndarray, new_means : np.ndarray) ->␣
↪np.ndarray:
    '''
    See M_step docs.
    '''
    # YOUR CODE HERE

    numerator = np.zeros((W.shape[1],data.shape[1],data.shape[1]))
    # Cycle the Clusters
    for j in range(W.shape[1]):
```

```
        # Compute (x - \mu)
        x_mu = data - new_means[j,:] # Matrix of shape (N,d)
        # When doing some massaging of the eqn in the notes we can see that the␣
↪numerator we are looking
        # for is: x_mu[1,:]@x_mu[1,:].T*w_{l1} + ... + x_mu[m,:]@x_mu[m,:].
↪T*w_{lm}.
        # Recall we have to obtain a (d,d) shaped matrix for every cluster and␣
↪hence the
        # quantities x_mu[i,:]@x_mu[i,:].T for every i, must be rearranged in␣
↪order to obtain
        # such shape:
        # Cycle the data
        for i in range(data.shape[0]):
            numerator[j,:,:] = numerator[j,:,:] + np.outer(x_mu[i,:].T, x_mu[i,:
↪]) * W[i,j]

    # As above the denominator is:
    denominator = np.sum(W, axis=0).reshape((W.shape[1],1,1))

    # Python recognizes the only possible dimension in which it's possible to␣
↪perform the division:
    new_covs = numerator/denominator

    return new_covs

def M_step(data : np.ndarray, W : np.ndarray) -> tuple:
    '''
    Function to compute the Maximization step on a GMM (use the expressions␣
↪derived on the slides).
    :param data: Dataset N x d (d := number of features)
    :param W: Weight matrix N x K (K :=number of classes). Element in position␣
↪(i,j) represents the probability of
              i-th datum to belong to class j (j-th cluster) given the current␣
↪parameters: pi, means, covs
    :returns: (new_pi, new_mu, new_cov)
        WHERE:
        new_pi: Contains the mixing probabilities. Its shape is (K,)
        new_mu: Contains the new means of the GMM model. Its shape is (K, d)
        new_cov: Contains the new covariances of the GMM model. Its shape is␣
↪(K, d, d)
    '''
    new_pi    = update_pi(W)
    new_means = update_means(data, W)
    new_covs  = update_covs(data, W, new_means)

    return new_pi, new_means, new_covs
```

[8]:
```python
W = np.array([[0.5, 0.5, 0],[0.5, 0, 0.5]])
assert np.isclose(update_pi(W), [0.5 , 0.25, 0.25], atol=1e-4).all()
W = np.random.normal(0,1, 1000).reshape(-1, 25) # Note this W is not normalized␣
↪properly, we do not care not since we are testing only the output shape is␣
↪correct
assert update_pi(W).shape == (25,)
# Test on the means update function
a = np.random.normal(0,1, 80).reshape(-1, 2)
b = update_means(a, W)
assert b.shape  == (25, 2)
# Test on the covs update function
assert update_covs(a, W, b).shape == (25, 2, 2)
```

[9]:
```python
# TODO 3: Write a function E_step which computes the E-step of the EM algorithm␣
↪in case of a GMM. Refer
# to the slides you can find on the moodle (in which ALL the necessary steps␣
↪and expressions are present).
def E_step(data : np.ndarray, pi : np.ndarray, means : np.ndarray, covs : np.
↪ndarray):
    '''
    Function to compute the Expectation step on a GMM model (use the␣
↪expressions derived on the slides) given
    the current values of the GMM parameters: pi, means, covs.
    :param data:  Same as M_step function.
    :param pi:    Same as M_step function.
    :param means: Same as M_step function.
    :param covs:  Same as M_step function.
    :returns: W, which is updated using the parameters of the GMM: pi, means,␣
↪covs. W must be normalized (see
              slides)
    '''
    # YOUR CODE HERE

    # Using the notation used in the M_step function above (slightly different␣
↪from the one
    # of the notes) we are looking for the elements [w_{ij}] of the matrix W␣
↪which are
    # the probabilities for a datum x_i to be classified in the j-th Cluster.
    # In the following we compute separately the factors of interest for the␣
↪final equation:

    # Initialize normalization factor (equal for every row of the matrix W),
    # of shape (N = #of data, 1)
    normalization_factor = np.zeros((data.shape[0],1))
    # Iniziatlize matrix W of shape (N = #of data ,K = #of Clusters):
    W = np.zeros((data.shape[0],pi.shape[0]))
```

```
    # Cycle the clusters
    for j in range(pi.shape[0]):
        # Cycle the data
        for i in range(data.shape[0]):
            # P(x_i | z_i = j) is a sample from a gaussian of which we know the␣
↪parameters.
            # We can use the function we wrote before:
            x_given_z = gaussian_pdf(data[i,:], means[j,:], covs[j,:,:])
            # Prior P(z_i = j)
            prior_pi = pi[j]
            # Assign the (not Normalized) value to the correct element of W:
            W[i,j] = x_given_z * prior_pi
            # Update the normalization factor corresponding to the current row␣
↪of W
            normalization_factor[i] = normalization_factor[i] + W[i,j]

    # Normalize the Matrix W (Python knows in which dimension to perform the␣
↪division)
    W = W / normalization_factor

    return W
```

[10]:
```
assert E_step(data, pi, means, covs).shape == (119, 3)
assert np.isclose(np.sum(E_step(data, pi, means, covs).sum(1) - 1), 0,␣
↪atol=1e-4)
```

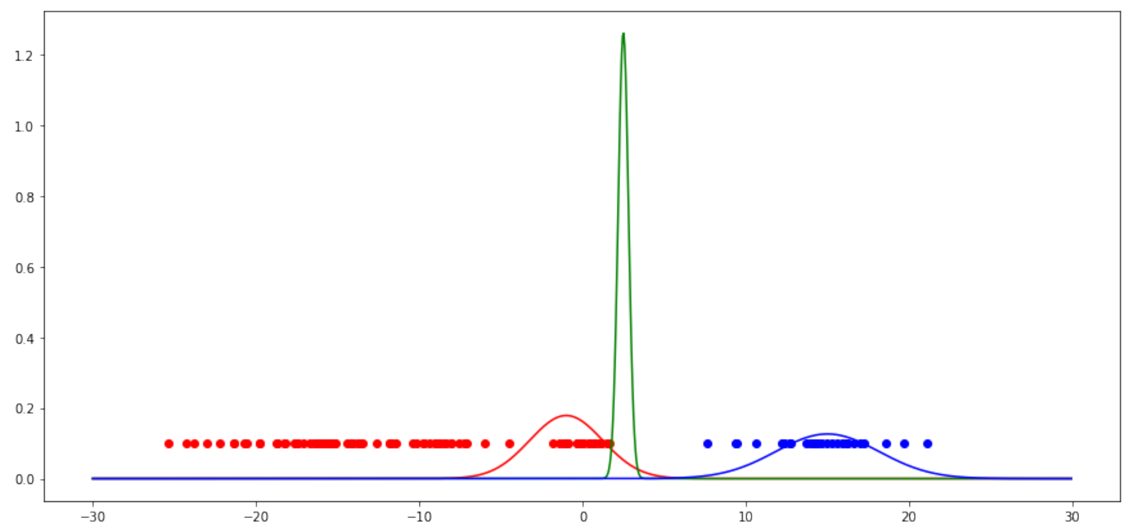
[11]:
```
# TODO 4: Write a function to randomly initialize the parameters of a GMM. This␣
↪is necessary to start with the
# EM iterations.

def randomly_initialize_W(data : np.ndarray, num_classes : int) -> np.ndarray:
    '''
    See random_init function docs.
    '''
    # YOUR CODE HERE

    # Initialize a matrix of zeros of right dimension (N = #of data, K= #of␣
↪Clusters):
    num_of_data = data.shape[0]
    W = np.zeros((num_of_data, num_classes))

    # Compute the number of data that should be in every cluster:
    data_for_every_cluster = num_of_data // num_classes
```

```
    # Cycle through clusters and assign to each one of them the right number of␣
↪data points
    # ("Assign" means to set the probability of such data to be in such cluster␣
↪to 1).
    # To do this we must assign a column vector of ones to the elements of W s.
↪t.
    # W[j*N/3:(j+1)*N/3, j] for j=0,1,2,...,num_classes (and where num_classes␣
↪= 3 for simplicity)
    for j in range(num_classes-1):
        W[j*data_for_every_cluster:(j+1)*data_for_every_cluster,j:j+1] = np.
↪ones((data_for_every_cluster,1))
        # Note that the indexing j:j+1 is useful to keep the shape as (a,1)␣
↪instead of (a,)

    # Assign the remaining (N - (N//K)(K-1)) data to the last cluster:
    data_for_last_cluster = num_of_data -␣
↪(data_for_every_cluster)*(num_classes-1)
    W[(num_classes-1)*data_for_every_cluster:,num_classes-1:num_classes] = np.
↪ones((data_for_last_cluster,1))

    return W

def random_init(data : np.ndarray, num_classes : int) -> np.ndarray:
    '''
    Function to initialize W and GMM parameters. W is generated assigning␣
↪randomly each datapoint only to one
    cluster. We require to assign the same number of points for each cluster␣
↪(even this is not striclty necessary
    to run the EM algorithm on GMM). If the number of data is not exactly␣
↪divisible by the number of clusters
    assign the exceeding data to one single class (it does not matter which␣
↪one).
    See M_step docs for details on W, interpretation and required shape.
    '''
    # Initialize W
    W = randomly_initialize_W(data, num_classes)
    # Use M_step to get GMM parameters
    pi, means, covs = M_step(data, W)
    return W, pi, means, covs
```

[12]:
```
assert (randomly_initialize_W(np.random.normal(0,1, 100).reshape(-1,10), 5).
↪sum(0) == 2).all()
assert (randomly_initialize_W(np.random.normal(0,1, 100).reshape(-1,5), 2).
↪sum(0) == 10).all()
assert (randomly_initialize_W(np.random.normal(0,1, 100).reshape(-1,5), 3).
↪sum(0) -6 == 2).any()
```

[13]:
```
# Let's evaluate the class probability of each datum given current pi, means␣
 ↪and covs.
# Since we have 3 clusters we shall colour-code (rgb) each datum according to␣
 ↪the class probabilities. Note that
# it is not very likely to have a datum coming from the central (very peaked)␣
 ↪distribution.
W = E_step(data, pi, means, covs)

plt.figure(figsize=(15,7))
for i, x in enumerate(data):
    plt.scatter(x, 0.1, color=np.array([W[i][0], W[i][1], W[i][2]]))
plt.plot(X, [gaussian_pdf(x, means[0], covs[0]) for x in X], color='r')
plt.plot(X, [gaussian_pdf(x, means[1], covs[1]) for x in X], color='g')
plt.plot(X, [gaussian_pdf(x, means[2], covs[2]) for x in X], color='b')
```

[13]: [<matplotlib.lines.Line2D at 0x7f323104bca0>]



[14]:
```
# TODO 5: compute the log likelihood of an iid dataset under a GMM model.
def log_likelihood_GMM(data : np.ndarray, pi : np.ndarray, means : np.ndarray,␣
 ↪covs : np.ndarray) -> float:
    '''
    Function to compute the log likelihood for a set of iid observations under␣
 ↪a GMM. Use the function you built
    before "gaussian_pdf" to compute the likelihood.
    :param data: N x d matrix containing a set of N iid data of dimension d
    :param pi:    Same as M_step function.
    :param means: Same as M_step function.
    :param covs:  Same as M_step function.
    '''
```

```
    # YOUR CODE HERE

    # To compute the log_likelihood we can simply use the definition given in␣
↪the EM notes
    # of (standard) likelihood and then apply the log.
    # Note how the parameter theta is implicetely used to define the␣
↪probabilities
    # of the formule.

    # Accumulating variable for the likelyhood
    log_likelihood = 0

    # cycle data
    for i in range(data.shape[0]):
        # Accumulating variable for p(x_i)
        p_xi = 0
        # cycle clusters
        for j in range(pi.shape[0]):
            # Probability P(x_i|j)
            p_xi_given_j = gaussian_pdf(data[i,:], means[j,:], covs[j,:,:])
            # Probability P(z = j)*P(x_i|j) = P(x_i, z=j)
            p_xi_and_j = pi[j] * p_xi_given_j
            # Increase Accumulating variable for p(x_i)
            p_xi = p_xi + p_xi_and_j
        # Increasing the log likelihood
        log_likelihood = log_likelihood + np.log(p_xi)

    return log_likelihood
```

[15]:
```
assert type(log_likelihood_GMM(data, pi, means, covs)) == np.float64
```

[16]:
```
# TODO 6: Write a function to run the EM on GMM using the building blocks we␣
↪created so far. Then test it on the
# data we used so far (choose a meaningful max_iter, you do not need to␣
↪exaggerate).
def run_EM_on_GMM(data : np.ndarray, number_clusters : int, max_iter : int,␣
↪epsilon : float = 1e-3,
                  plot_intermediate : bool = False) -> tuple:
    '''
    Function to run GMM on a given dataset and a given number of clusters. The␣
↪termination conditions of the
    iterative algorithm take into account either a specified max number of␣
↪iterations or the improvement of the
    log likelihood (if the log likelihood does not improve more than epsilon in␣
↪two successive iterations we stop).
    :param data: N x d matrix containing a set of N iid data of dimension d
```

```
    :param number_clusters: # of clusters (information we have a priori, before␣
↪starting the EM)
    :param max_iter: Maximum number of iterations allowed to the EM.
    :param epsilon: Threshold on the improvement of the log likelihood
    :param plot_intermediate: Boolean used to plot intermediate GMM for 2-d␣
↪datasets (you do not need to implement
                              anything).
    :returns: (W, pi, means, covs, log_likelihood_train)
        WHERE:
        W:     Optimal W     after EM reaches termination condition (same shape␣
↪as M_step function).
        pi:    Optimal pi    after EM reaches termination condition (same shape␣
↪as M_step function).
        means: Optimal means after EM reaches termination condition (same shape␣
↪as M_step function).
        covs:  Optimal covs  after EM reaches termination condition (same shape␣
↪as M_step function).
        log_likelihood_train: log likelihoods obtained during training (saved␣
↪using a list).
    '''
    W, pi, means, covs = random_init(data, number_clusters)
    log_likelihood_train = [log_likelihood_GMM(data, pi, means, covs)]
    num_iter = 0
    # Used to plot 2-d data
    if plot_intermediate:
        x_max = np.max(np.abs(X))
        x, y = np.mgrid[-x_max:x_max:.05, -x_max:x_max:.05]
        pos = np.dstack((x, y))

    while (True):
        # Iterate with E-Step and M-step
        # YOUR CODE HERE

        # EXPECTATION STEP
        W = E_step(data, pi, means, covs)
        # MAXIMIZATION STEP
        pi, means, covs = M_step(data, W)

        # Save log likelihood given current GMM parameters
        log_likelihood_train.append(log_likelihood_GMM(data, pi, means, covs))

        if plot_intermediate:
            # Plot scatter plot of training data and corresponding clusters
            fig = plt.figure(figsize=(15,7))
            for k in range(0, number_clusters):
```

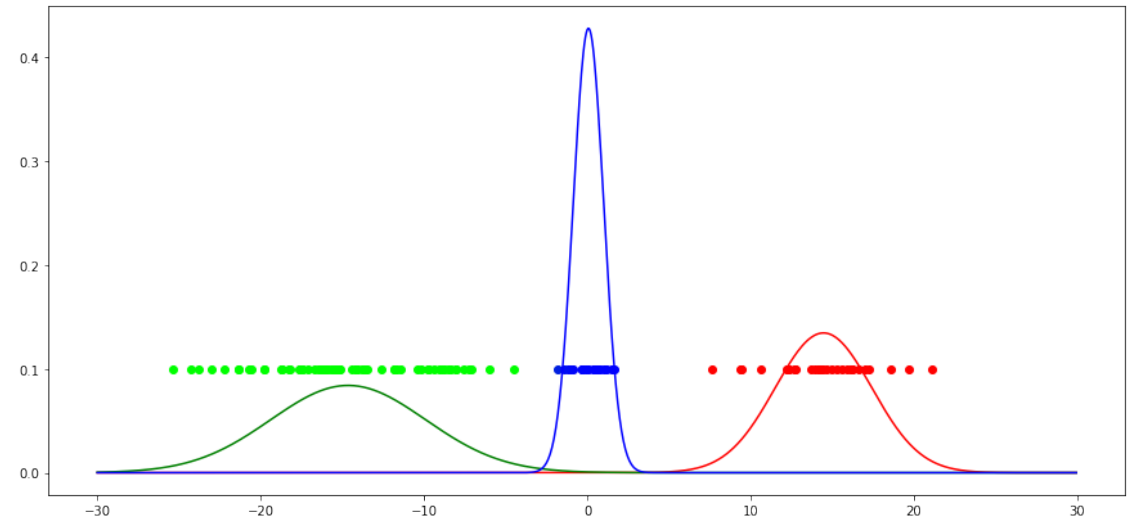
```
                plt.contour(x, y, multivariate_normal(means[k], covs[k]).
↪pdf(pos))
            plt.scatter(data[0:,0], data[0:,1])
            plt.title(f'Iteration {num_iter}')

        print(f'Iteration {num_iter}, log likelihood {log_likelihood_train[-1]:.
↪4f}, '
              f' delta log likelihood {(log_likelihood_train[-1] -␣
↪log_likelihood_train[-2]):.4f}')
        num_iter += 1

        # Use proper termination conditions, on: number of iteration or␣
↪threshold on log likelihood improvement
        # (use the break statement to stop while cycle)
        # YOUR CODE HERE

        # Define the 3 boolean variables of interest
        max_iter_reached = num_iter == max_iter
        no_improvement_1 = False
        no_improvement_2 = False
        # Check if we made at least 2 iterations
        if num_iter >= 2:
            no_improvement_1 = abs(log_likelihood_train[-3] -␣
↪log_likelihood_train[-2]) < epsilon
            no_improvement_2 = abs(log_likelihood_train[-2] -␣
↪log_likelihood_train[-1]) < epsilon

        if (max_iter_reached) or (no_improvement_1 and no_improvement_2):
            break

    return W, pi, means, covs, log_likelihood_train

# Let's try our implementation of the EM algorithm
max_iter = None  # to be overwritten
# YOUR CODE HERE
max_iter = 150
W, pi, means, covs, log_likelihood_train = run_EM_on_GMM(data, 3, max_iter,␣
↪plot_intermediate=False)

plt.figure(figsize=(15,7))
for i, x in enumerate(data):
    plt.scatter(x, 0.1, color=np.array([W[i][0], W[i][1], W[i][2]]))
plt.plot(X, [gaussian_pdf(x, means[0], covs[0]) for x in X], color='r')
plt.plot(X, [gaussian_pdf(x, means[1], covs[1]) for x in X], color='g')
plt.plot(X, [gaussian_pdf(x, means[2], covs[2]) for x in X], color='b')
```

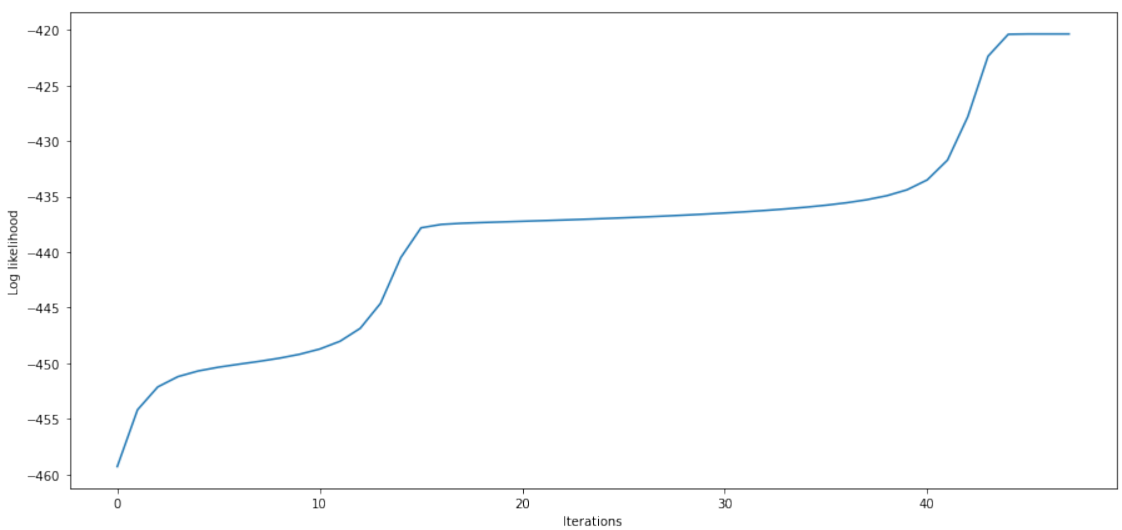
```python
plt.figure(figsize=(15,7))
plt.plot(log_likelihood_train)
plt.xlabel('Iterations')
plt.ylabel('Log likelihood')
```

```
Iteration 0, log likelihood -454.1864,  delta log likelihood 5.1081
Iteration 1, log likelihood -452.1271,  delta log likelihood 2.0592
Iteration 2, log likelihood -451.1994,  delta log likelihood 0.9277
Iteration 3, log likelihood -450.6908,  delta log likelihood 0.5086
Iteration 4, log likelihood -450.3524,  delta log likelihood 0.3384
Iteration 5, log likelihood -450.0848,  delta log likelihood 0.2676
Iteration 6, log likelihood -449.8288,  delta log likelihood 0.2560
Iteration 7, log likelihood -449.5437,  delta log likelihood 0.2851
Iteration 8, log likelihood -449.1915,  delta log likelihood 0.3522
Iteration 9, log likelihood -448.7181,  delta log likelihood 0.4733
Iteration 10, log likelihood -448.0193,  delta log likelihood 0.6988
Iteration 11, log likelihood -446.8530,  delta log likelihood 1.1664
Iteration 12, log likelihood -444.6096,  delta log likelihood 2.2434
Iteration 13, log likelihood -440.4947,  delta log likelihood 4.1148
Iteration 14, log likelihood -437.8132,  delta log likelihood 2.6815
Iteration 15, log likelihood -437.5068,  delta log likelihood 0.3064
Iteration 16, log likelihood -437.4047,  delta log likelihood 0.1021
Iteration 17, log likelihood -437.3392,  delta log likelihood 0.0656
Iteration 18, log likelihood -437.2824,  delta log likelihood 0.0567
Iteration 19, log likelihood -437.2270,  delta log likelihood 0.0554
Iteration 20, log likelihood -437.1703,  delta log likelihood 0.0567
Iteration 21, log likelihood -437.1111,  delta log likelihood 0.0591
Iteration 22, log likelihood -437.0489,  delta log likelihood 0.0622
Iteration 23, log likelihood -436.9830,  delta log likelihood 0.0659
Iteration 24, log likelihood -436.9130,  delta log likelihood 0.0700
Iteration 25, log likelihood -436.8384,  delta log likelihood 0.0746
Iteration 26, log likelihood -436.7586,  delta log likelihood 0.0798
Iteration 27, log likelihood -436.6728,  delta log likelihood 0.0857
Iteration 28, log likelihood -436.5803,  delta log likelihood 0.0925
Iteration 29, log likelihood -436.4800,  delta log likelihood 0.1004
Iteration 30, log likelihood -436.3702,  delta log likelihood 0.1098
Iteration 31, log likelihood -436.2489,  delta log likelihood 0.1213
Iteration 32, log likelihood -436.1132,  delta log likelihood 0.1358
Iteration 33, log likelihood -435.9586,  delta log likelihood 0.1546
Iteration 34, log likelihood -435.7785,  delta log likelihood 0.1801
Iteration 35, log likelihood -435.5619,  delta log likelihood 0.2166
Iteration 36, log likelihood -435.2898,  delta log likelihood 0.2721
Iteration 37, log likelihood -434.9260,  delta log likelihood 0.3639
Iteration 38, log likelihood -434.3927,  delta log likelihood 0.5332
Iteration 39, log likelihood -433.4998,  delta log likelihood 0.8930
Iteration 40, log likelihood -431.7184,  delta log likelihood 1.7814
Iteration 41, log likelihood -427.8222,  delta log likelihood 3.8962
```

```
Iteration 42, log likelihood -422.3769,  delta log likelihood 5.4453
Iteration 43, log likelihood -420.4013,  delta log likelihood 1.9756
Iteration 44, log likelihood -420.3741,  delta log likelihood 0.0272
Iteration 45, log likelihood -420.3736,  delta log likelihood 0.0006
Iteration 46, log likelihood -420.3735,  delta log likelihood 0.0000
```

[16]:
```
Text(0, 0.5, 'Log likelihood')
```





[17]:
```
a, b, c, d, e = run_EM_on_GMM(data, 3, 10, plot_intermediate=False)
assert a.shape == (119, 3)
assert b.shape == (3,)
assert c.shape == (3, 1)
```

```
assert d.shape == (3, 1, 1)
```

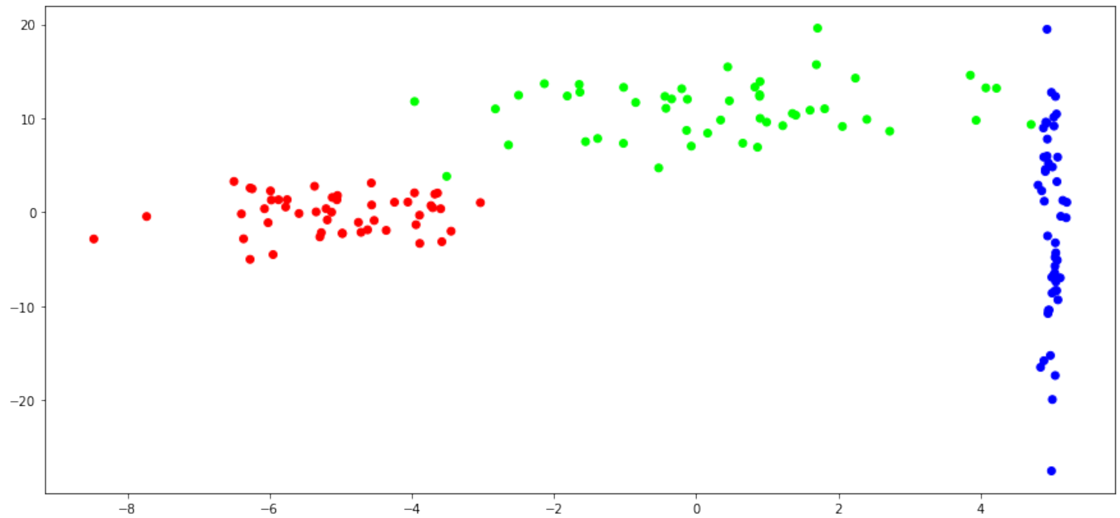
```
Iteration 0, log likelihood -454.1864,  delta log likelihood 5.1081
Iteration 1, log likelihood -452.1271,  delta log likelihood 2.0592
Iteration 2, log likelihood -451.1994,  delta log likelihood 0.9277
Iteration 3, log likelihood -450.6908,  delta log likelihood 0.5086
Iteration 4, log likelihood -450.3524,  delta log likelihood 0.3384
Iteration 5, log likelihood -450.0848,  delta log likelihood 0.2676
Iteration 6, log likelihood -449.8288,  delta log likelihood 0.2560
Iteration 7, log likelihood -449.5437,  delta log likelihood 0.2851
Iteration 8, log likelihood -449.1915,  delta log likelihood 0.3522
Iteration 9, log likelihood -448.7181,  delta log likelihood 0.4733
```

We shall now try the same procedure with a 2-dimensional dataset.

Everything is the same as before but we can apprecaite a better visualization.

[18]:
```
# Build the 2-D dataset.
K = 3
clusters_cov = [[1,2], [2,3], [0.1,10]]
centers = [[-5, 0], [0, 10.5], [5, -1]]
X, Y = make_blobs(cluster_std=clusters_cov, centers=centers,␣
 ↪random_state=ID_number, n_samples=150, shuffle=True)

colormap = np.array(['red', 'lime', 'blue'])

plt.figure(figsize=(15,7))
plt.scatter(X[:,0], X[:,1], c = colormap[Y])
```

[18]: <matplotlib.collections.PathCollection at 0x7f3230959fa0>

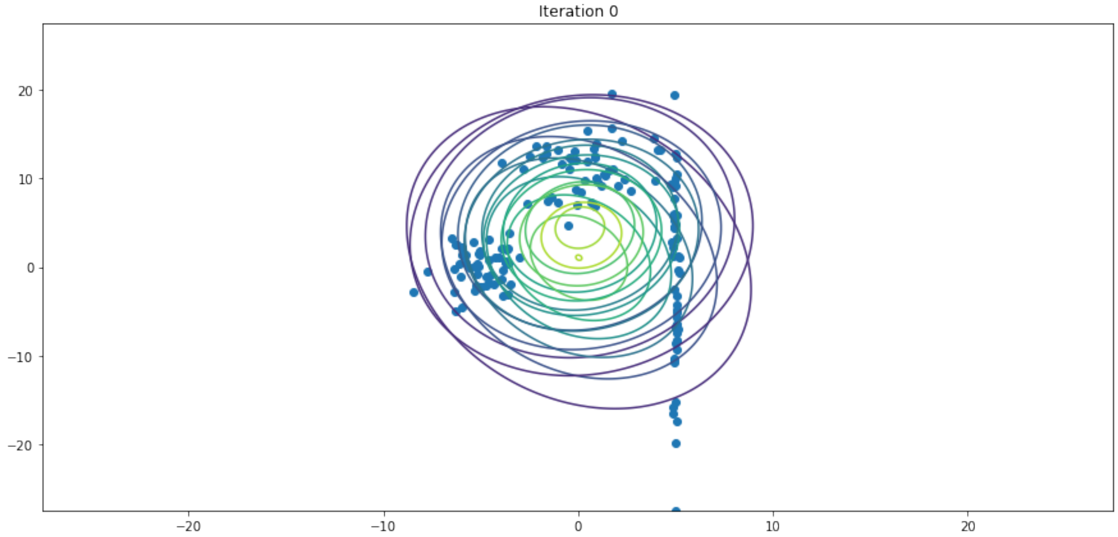


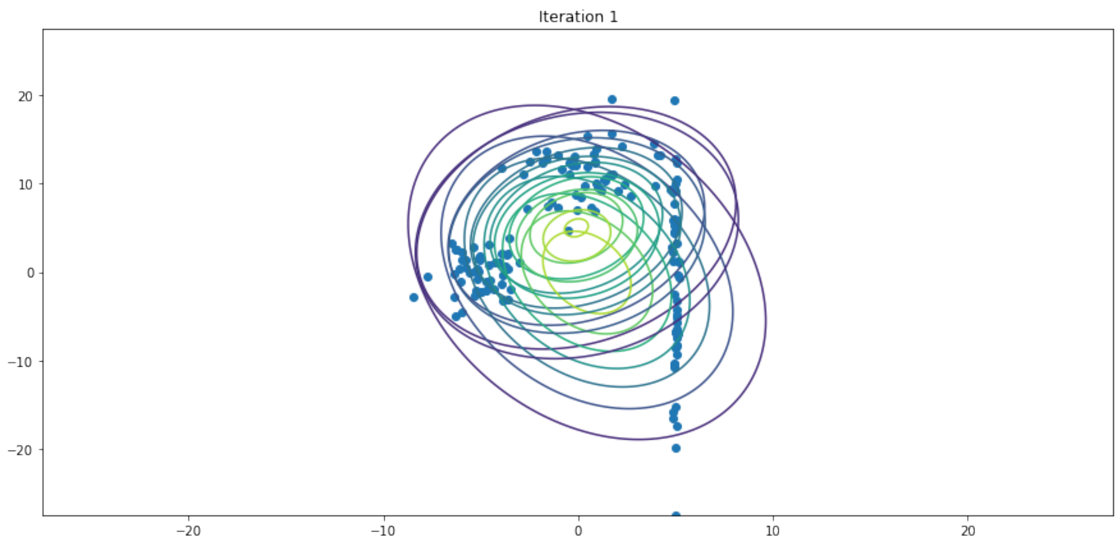
[19]:
```python
# Depending on your implementation this cell might take a while to run... (this␣
↪is mainly due to the plots)
W, pi, means, covs, log_likelihood_train = run_EM_on_GMM(X, K, 150,␣
↪plot_intermediate=True)
```

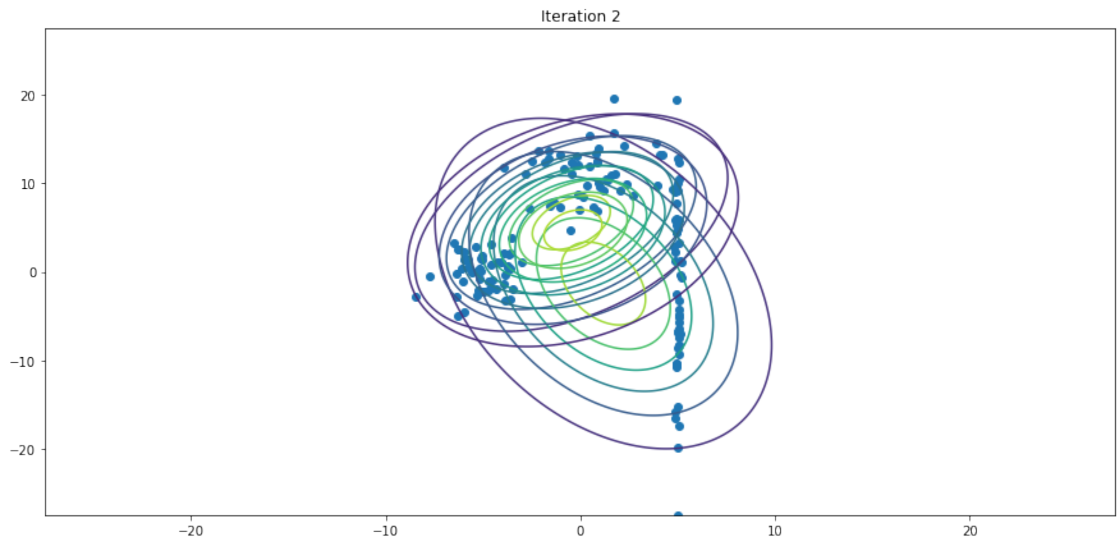
```
Iteration 0, log likelihood -957.2955,  delta log likelihood 3.0573
Iteration 1, log likelihood -950.8833,  delta log likelihood 6.4122
Iteration 2, log likelihood -940.2355,  delta log likelihood 10.6478
Iteration 3, log likelihood -926.5482,  delta log likelihood 13.6874
Iteration 4, log likelihood -911.7034,  delta log likelihood 14.8448
Iteration 5, log likelihood -890.1155,  delta log likelihood 21.5878
Iteration 6, log likelihood -862.2835,  delta log likelihood 27.8320
Iteration 7, log likelihood -821.7484,  delta log likelihood 40.5351
Iteration 8, log likelihood -776.0746,  delta log likelihood 45.6738
Iteration 9, log likelihood -751.1167,  delta log likelihood 24.9579
Iteration 10, log likelihood -749.3223,  delta log likelihood 1.7944
Iteration 11, log likelihood -748.8223,  delta log likelihood 0.5000
Iteration 12, log likelihood -748.2845,  delta log likelihood 0.5379
Iteration 13, log likelihood -747.7246,  delta log likelihood 0.5599
Iteration 14, log likelihood -747.1764,  delta log likelihood 0.5482
Iteration 15, log likelihood -746.6767,  delta log likelihood 0.4997
Iteration 16, log likelihood -746.2503,  delta log likelihood 0.4265
Iteration 17, log likelihood -745.8815,  delta log likelihood 0.3688
Iteration 18, log likelihood -745.4892,  delta log likelihood 0.3923
Iteration 19, log likelihood -744.8898,  delta log likelihood 0.5993
```

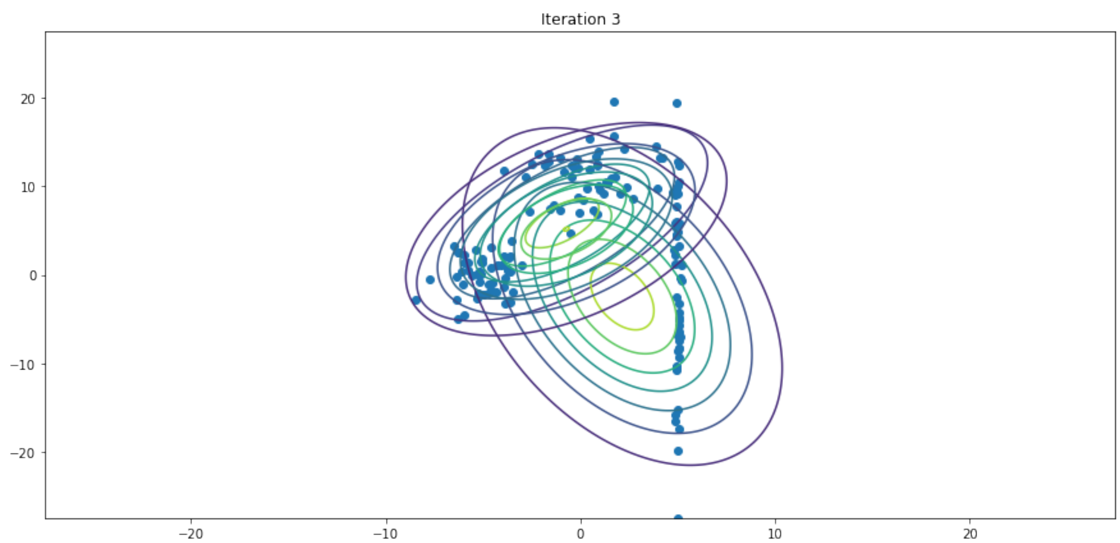
```
/tmp/ipykernel_18810/774263666.py:46: RuntimeWarning: More than 20 figures have
been opened. Figures created through the pyplot interface
(`matplotlib.pyplot.figure`) are retained until explicitly closed and may
consume too much memory. (To control this warning, see the rcParam
`figure.max_open_warning`).
  fig = plt.figure(figsize=(15,7))
```

```
Iteration 20, log likelihood -743.6637,  delta log likelihood 1.2261
Iteration 21, log likelihood -740.9041,  delta log likelihood 2.7597
Iteration 22, log likelihood -736.6646,  delta log likelihood 4.2395
Iteration 23, log likelihood -734.4769,  delta log likelihood 2.1877
Iteration 24, log likelihood -733.1524,  delta log likelihood 1.3245
Iteration 25, log likelihood -731.4800,  delta log likelihood 1.6725
Iteration 26, log likelihood -728.5069,  delta log likelihood 2.9731
Iteration 27, log likelihood -722.7740,  delta log likelihood 5.7329
Iteration 28, log likelihood -714.2061,  delta log likelihood 8.5679
Iteration 29, log likelihood -707.3035,  delta log likelihood 6.9026
Iteration 30, log likelihood -705.2737,  delta log likelihood 2.0298
Iteration 31, log likelihood -705.2211,  delta log likelihood 0.0526
Iteration 32, log likelihood -705.2207,  delta log likelihood 0.0003
Iteration 33, log likelihood -705.2207,  delta log likelihood 0.0000
```

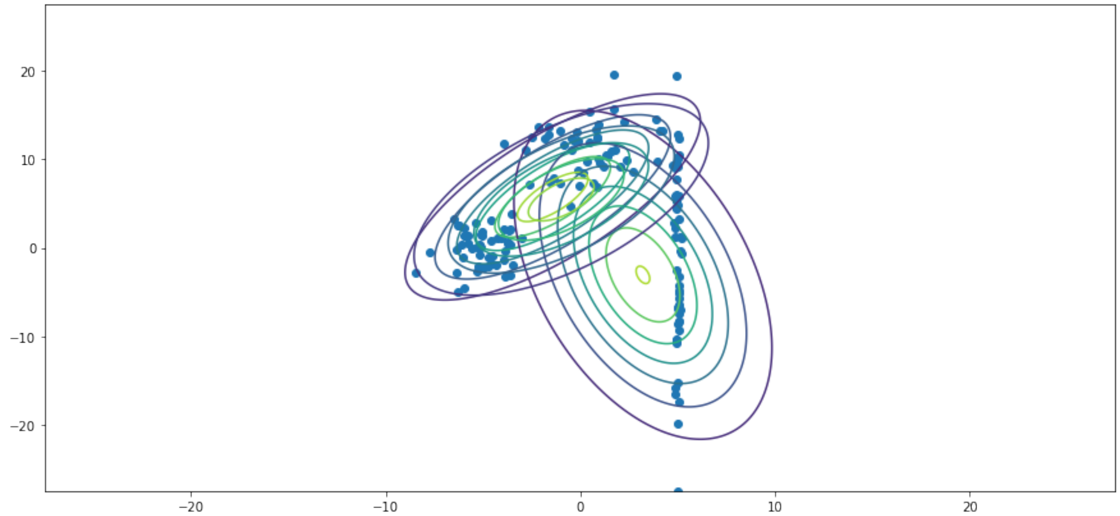

Iteration 0
20
10
0
−10
−20
−20
−10
0
10
20


Iteration 1
20
10
0
−10
−20
−20
−10
0
10
20

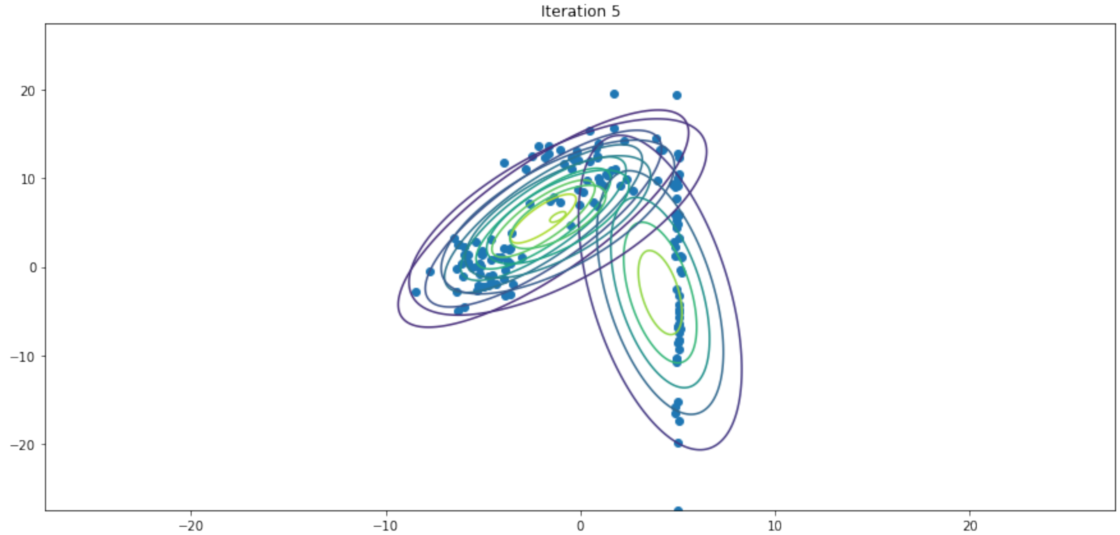

Iteration 2
20
10
0
−10
−20
−20
−10
0
10
20

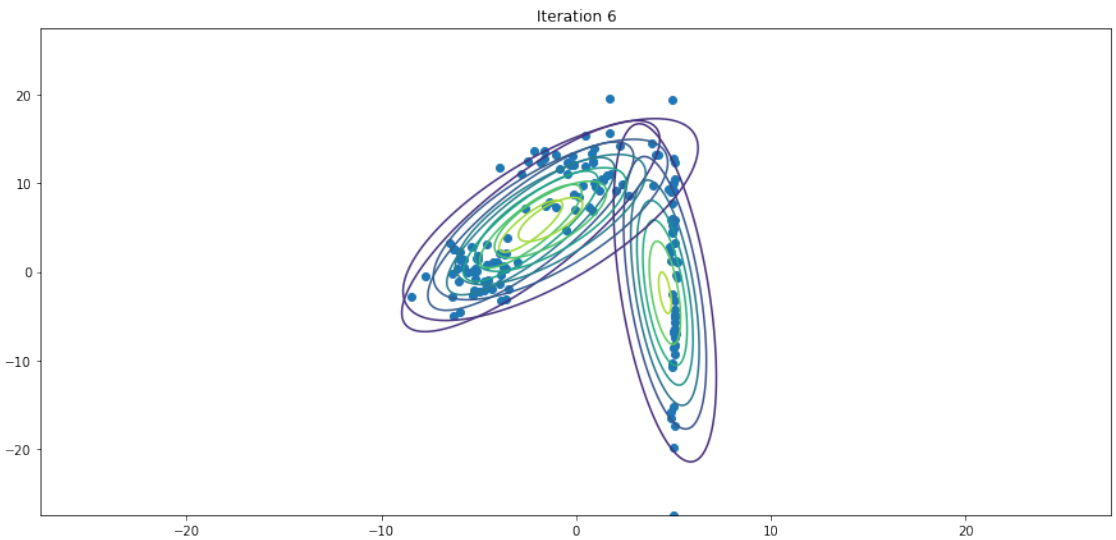

Iteration 3
20
10
0
−10
−20
−20
−10
0
10
20

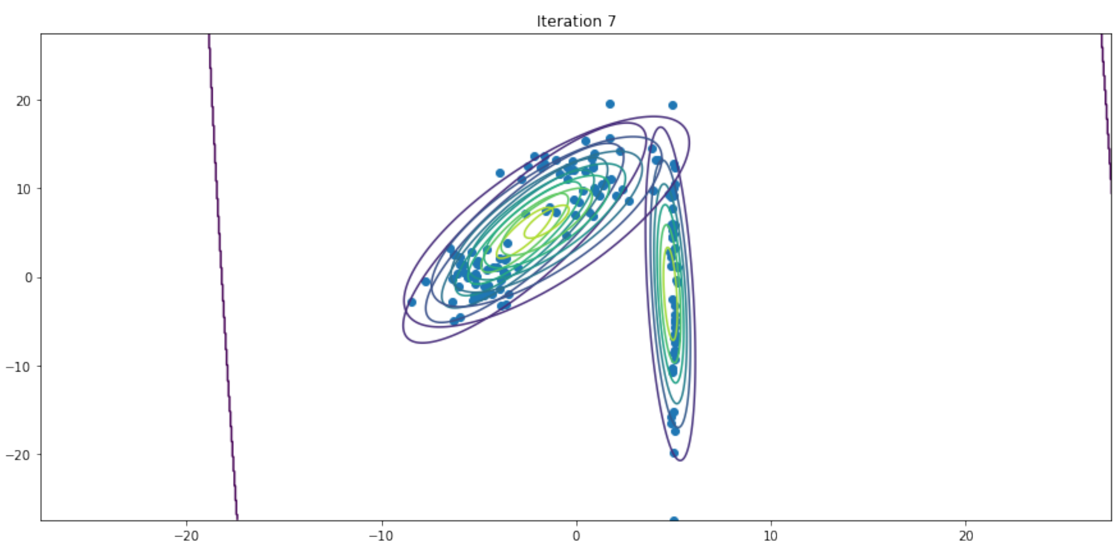
Iteration 4

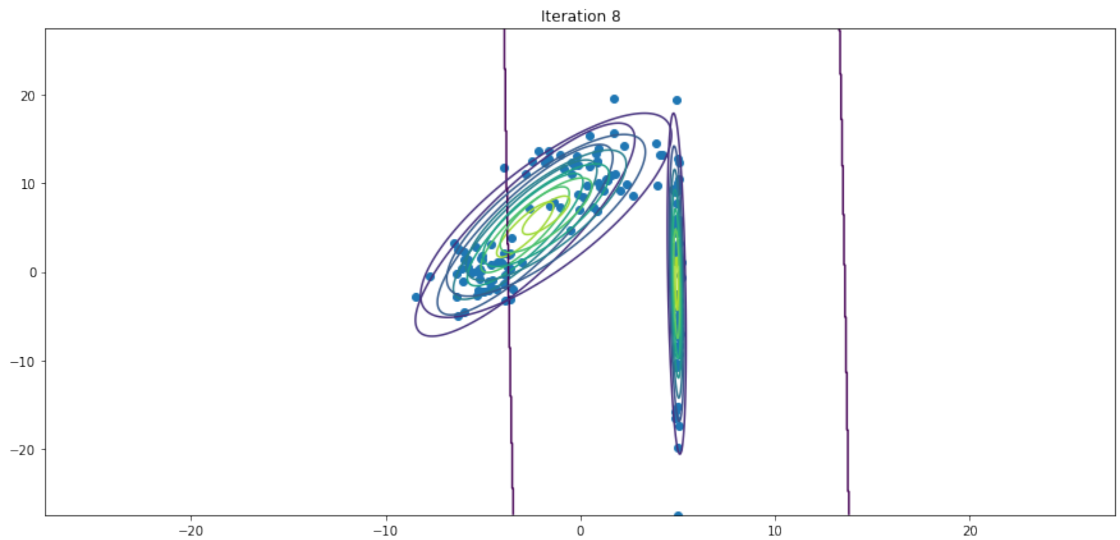


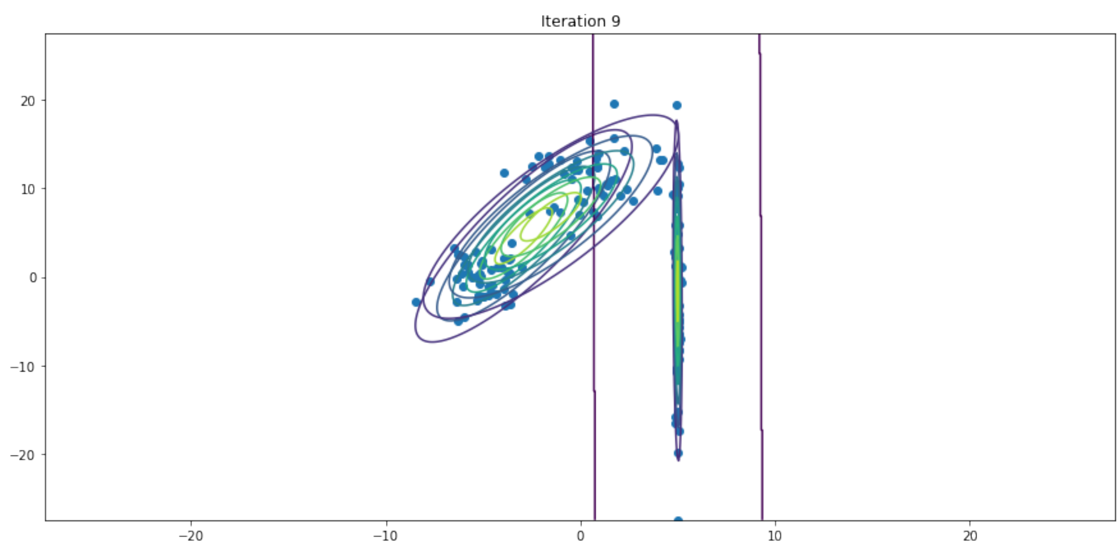
Iteration 5

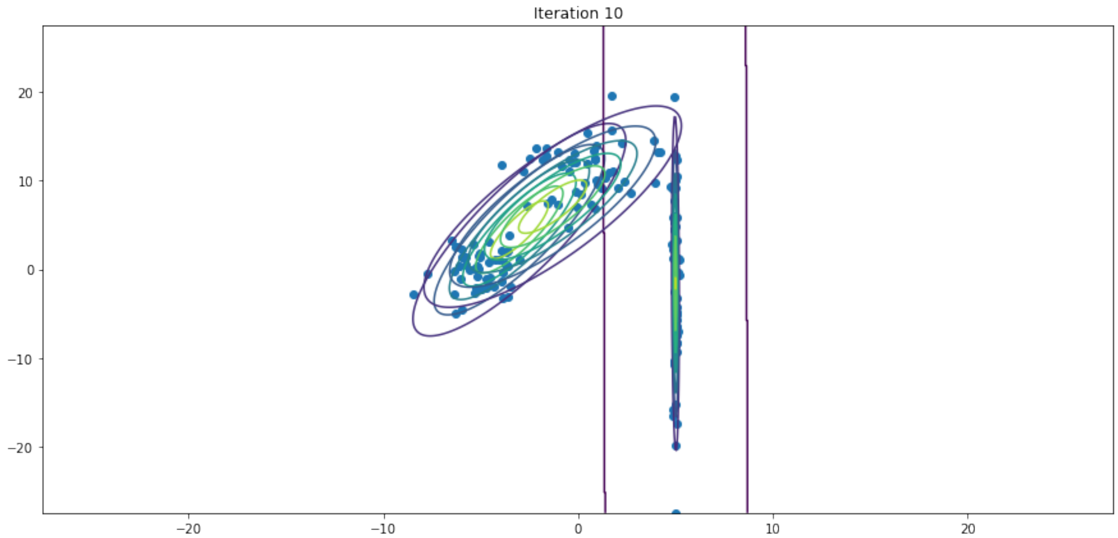


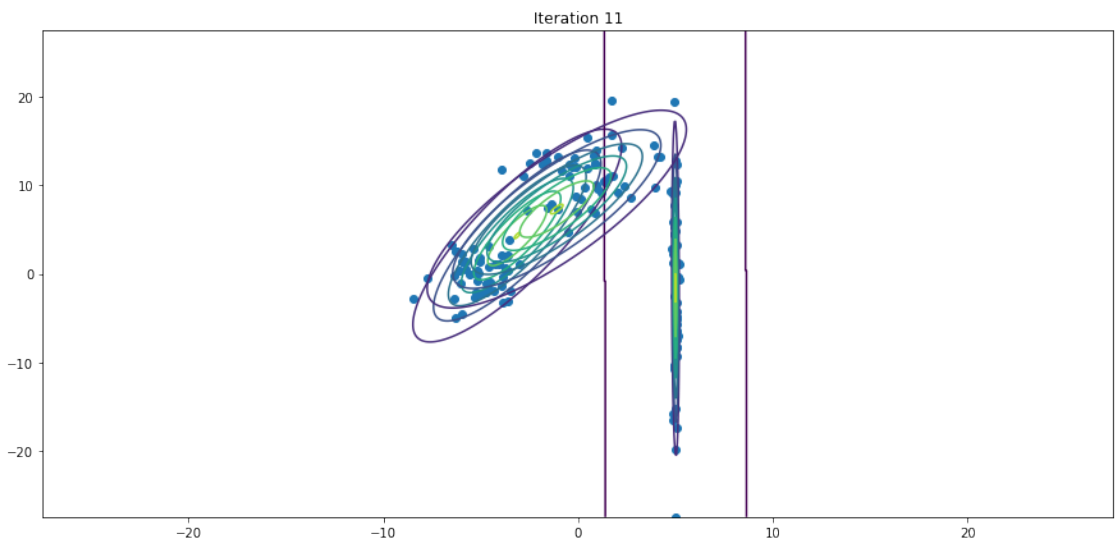

Iteration 6
20
10
0
−10
−20
−20
−10
0
10
20

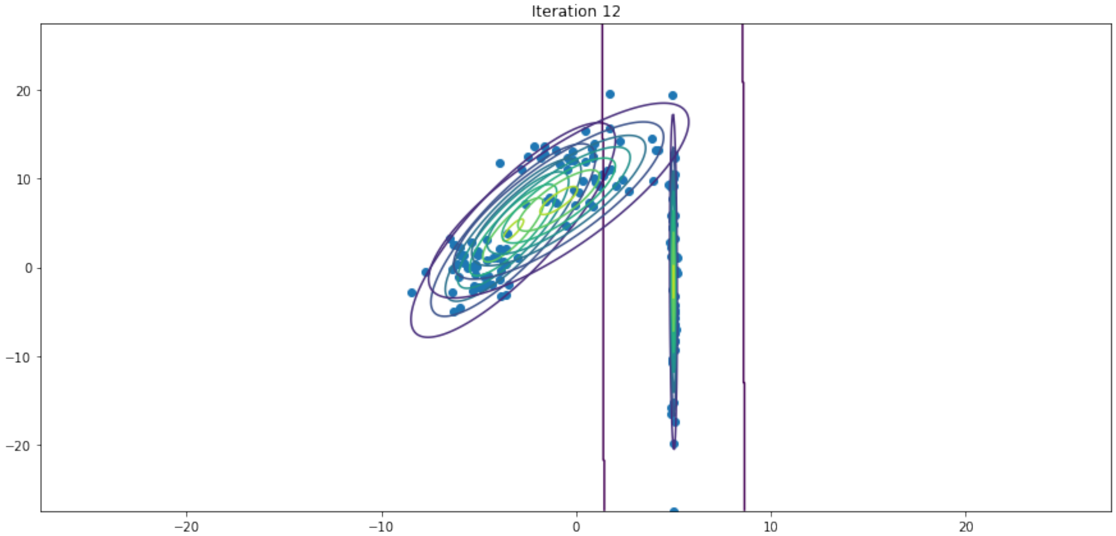

Iteration 7
20
10
0
−10
−20
−20
−10
0
10
20

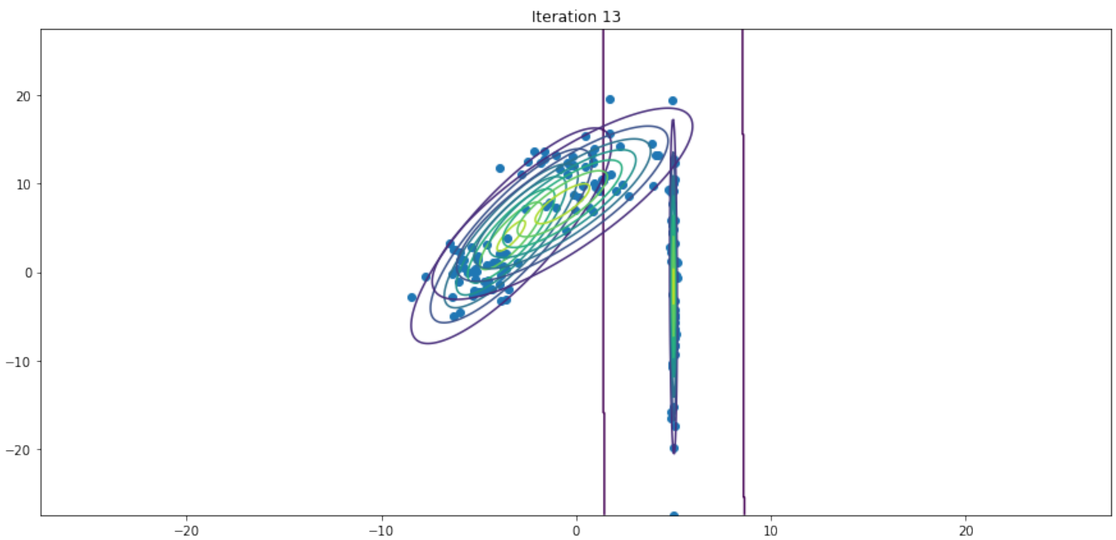

Iteration 8
20
10
0
−10
−20
−20
−10
0
10
20

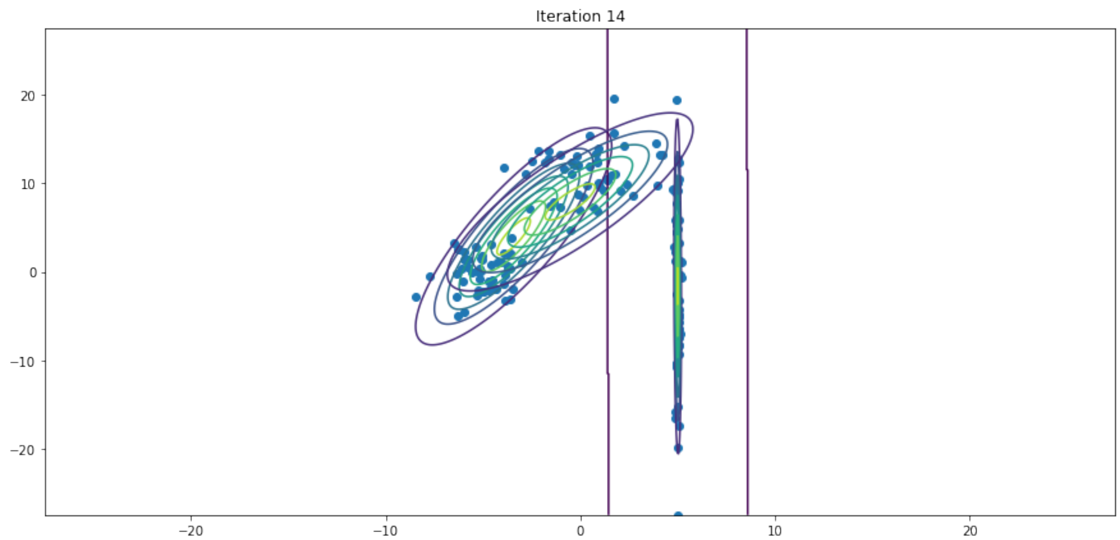

Iteration 9
20
10
0
−10
−20
−20
−10
0
10
20

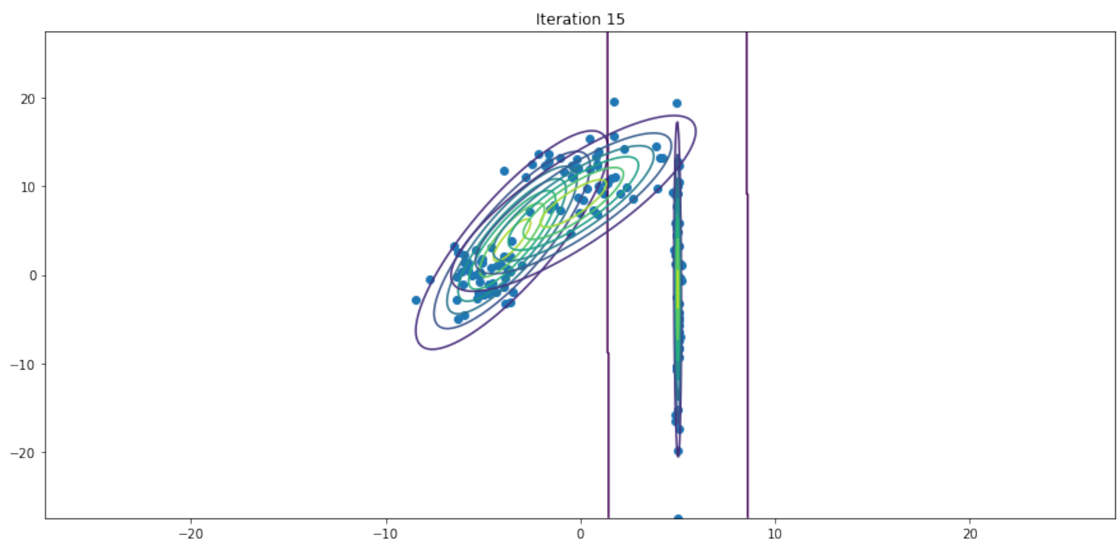

Iteration 10
20
10
0
−10
−20
−20
−10
0
10
20

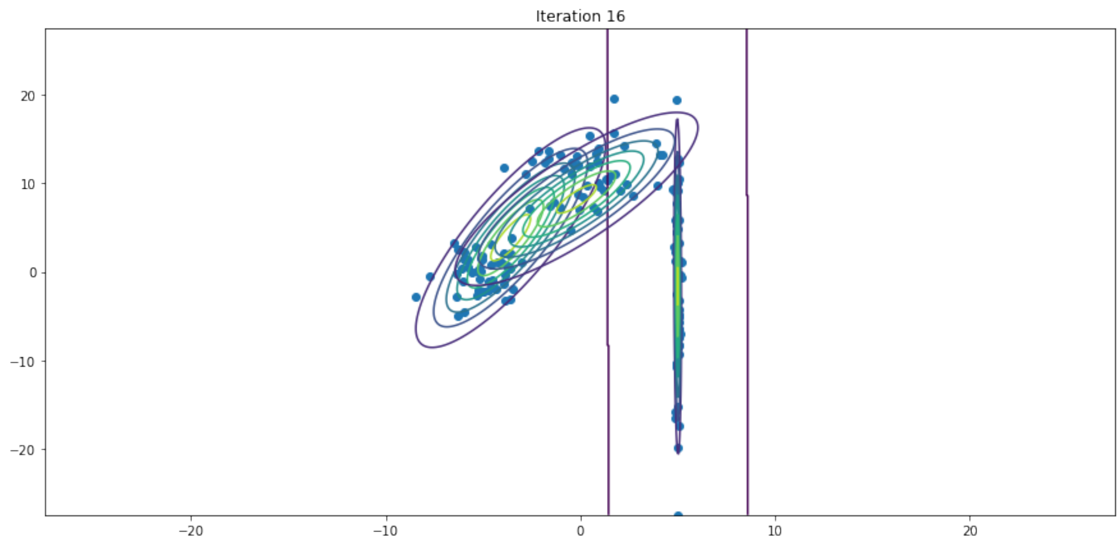

Iteration 11
20
10
0
−10
−20
−20
−10
0
10
20

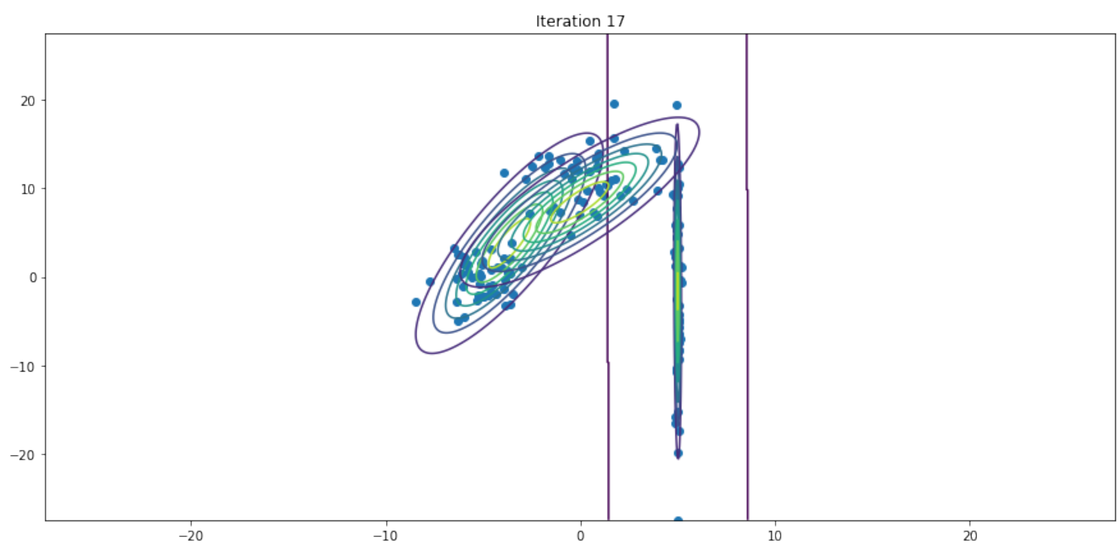

Iteration 12
20
10
0
−10
−20
−20
−10
0
10
20

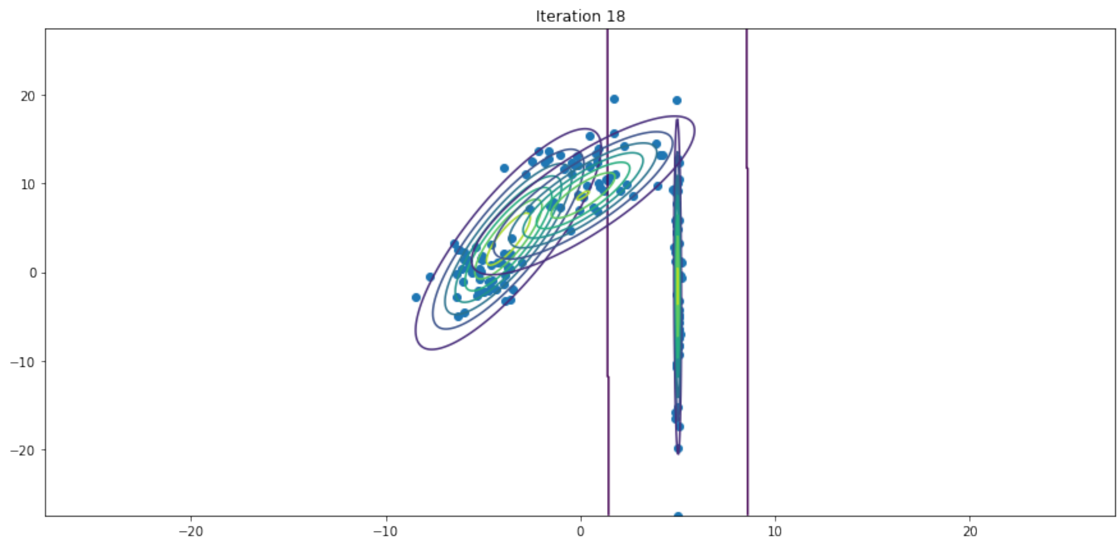

Iteration 13
20
10
0
−10
−20
−20
−10
0
10
20

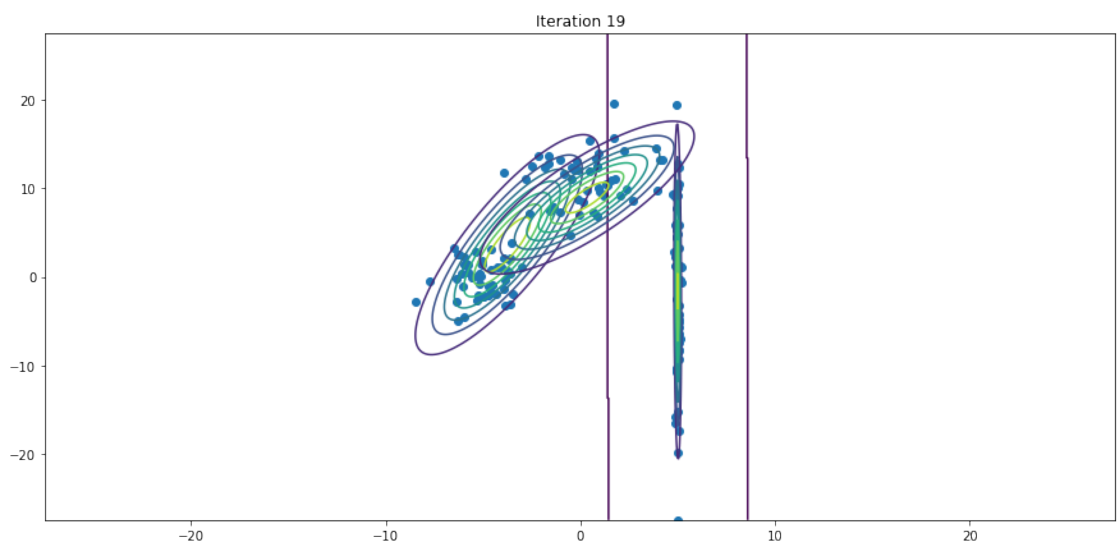

Iteration 14
20
10
0
−10
−20
−20
−10
0
10
20

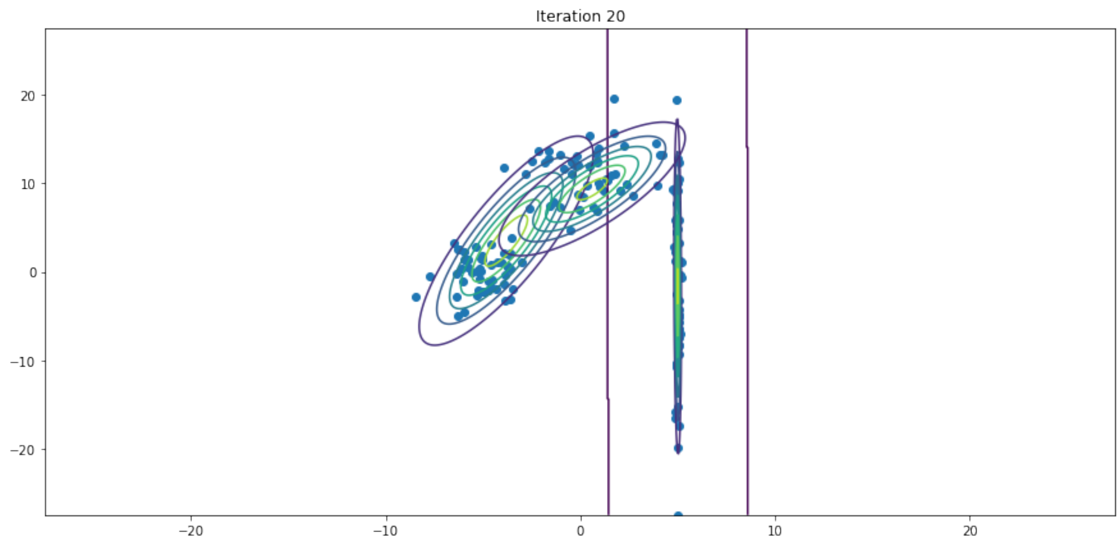

Iteration 15
20
10
0
−10
−20
−20
−10
0
10
20


Iteration 16
20
10
0
−10
−20
−20
−10
0
10
20

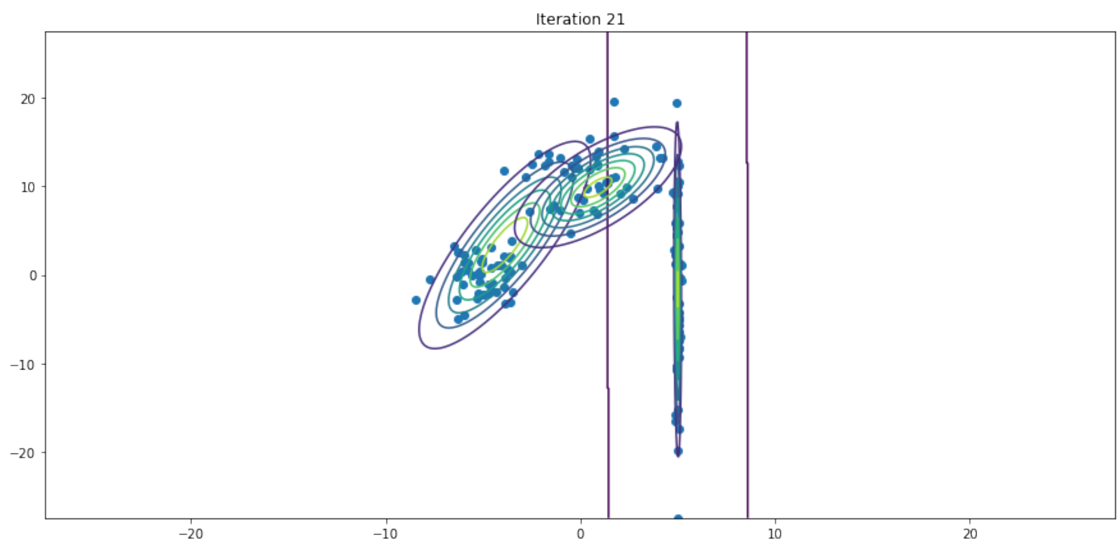

Iteration 17
20
10
0
−10
−20
−20
−10
0
10
20


Iteration 18
20
10
0
−10
−20
−20
−10
0
10
20

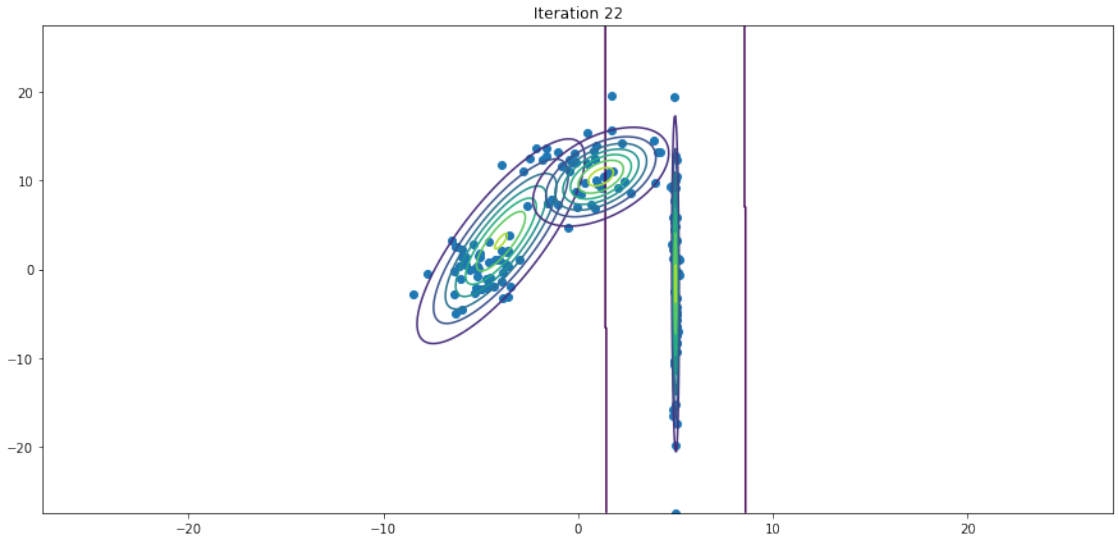

Iteration 19
20
10
0
−10
−20
−20
−10
0
10
20

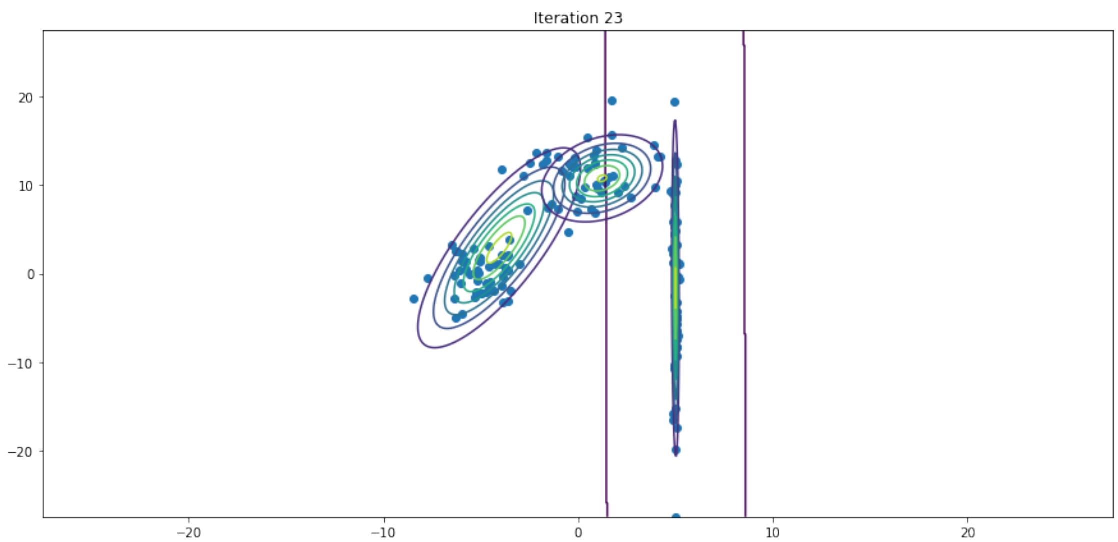


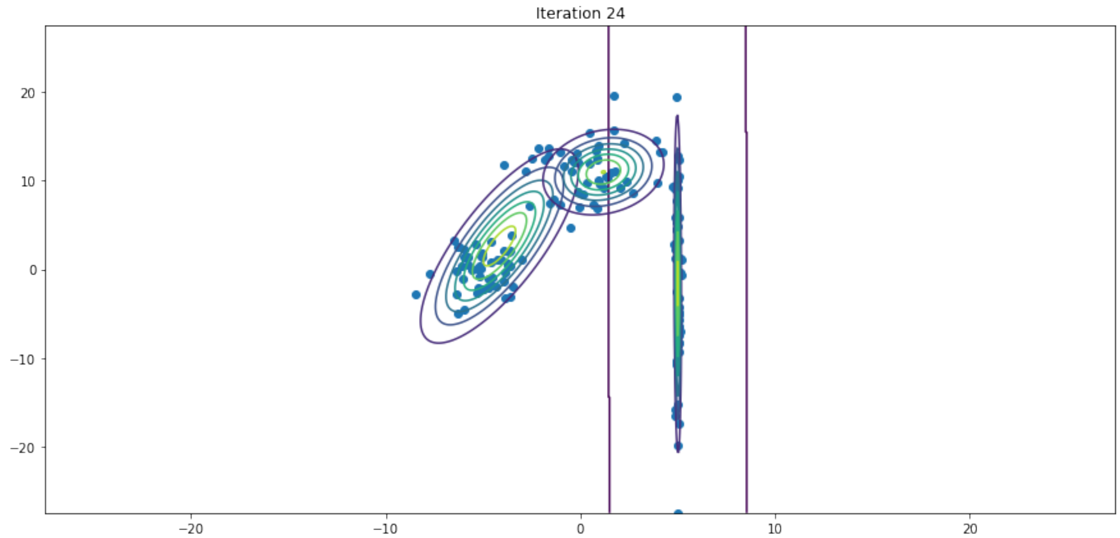
Iteration 20
20
10
0
−10
−20
−20
−10
0
10
20

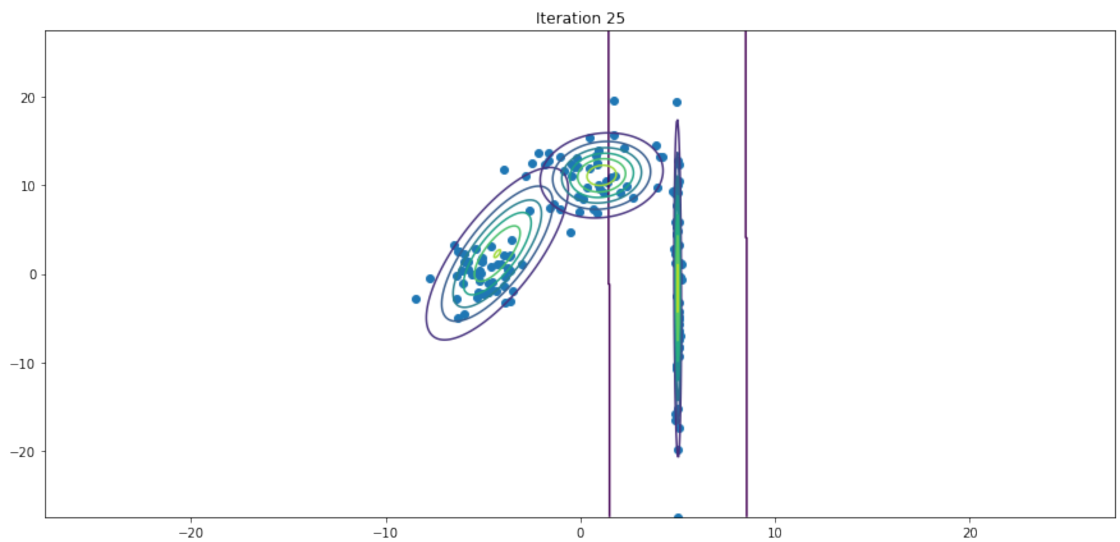


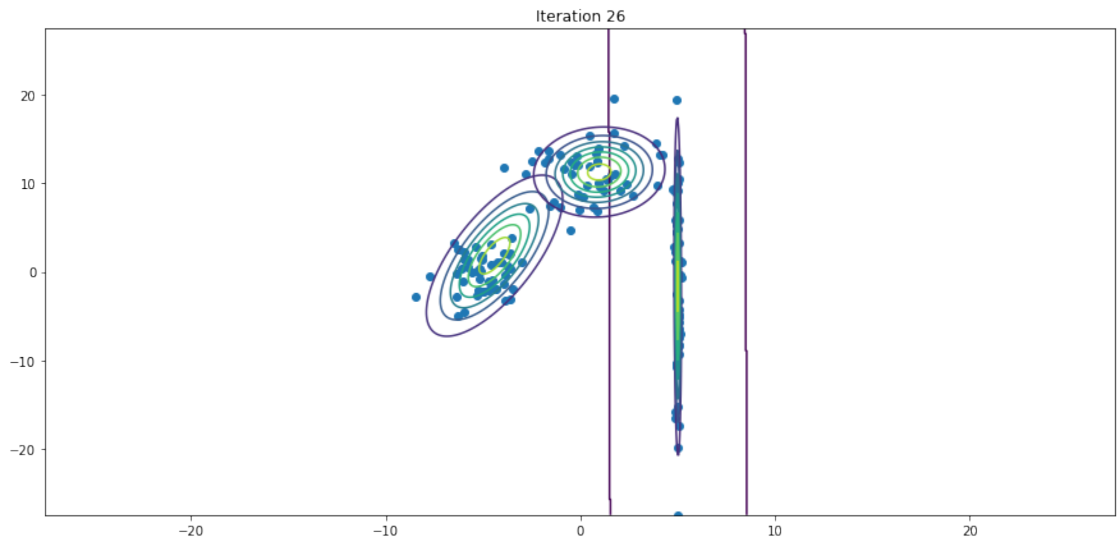
Iteration 21
20
10
0
−10
−20
−20
−10
0
10
20

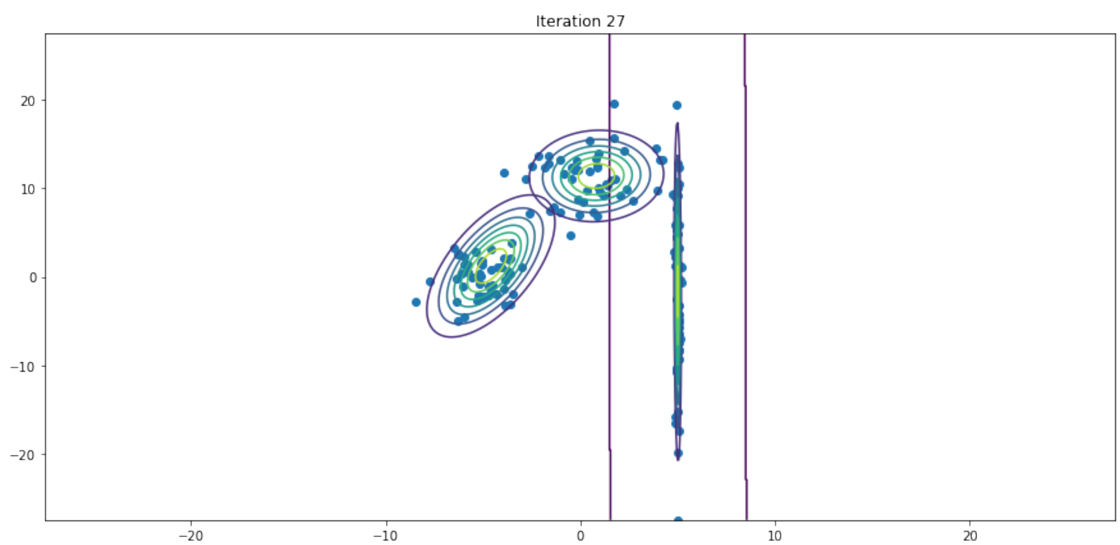

Iteration 22
20
10
0
−10
−20
−20
−10
0
10
20

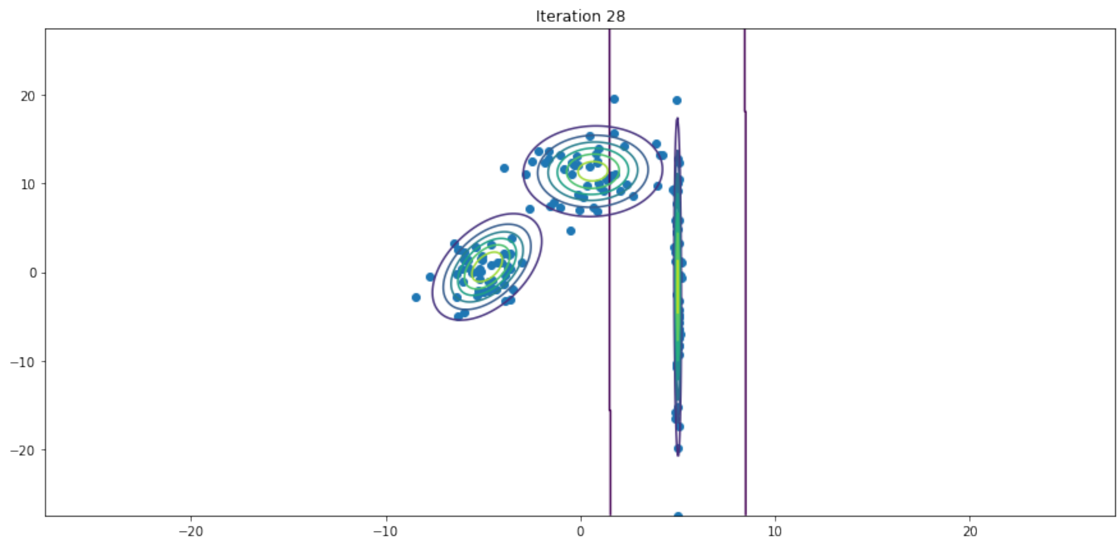

Iteration 23
20
10
0
−10
−20
−20
−10
0
10
20

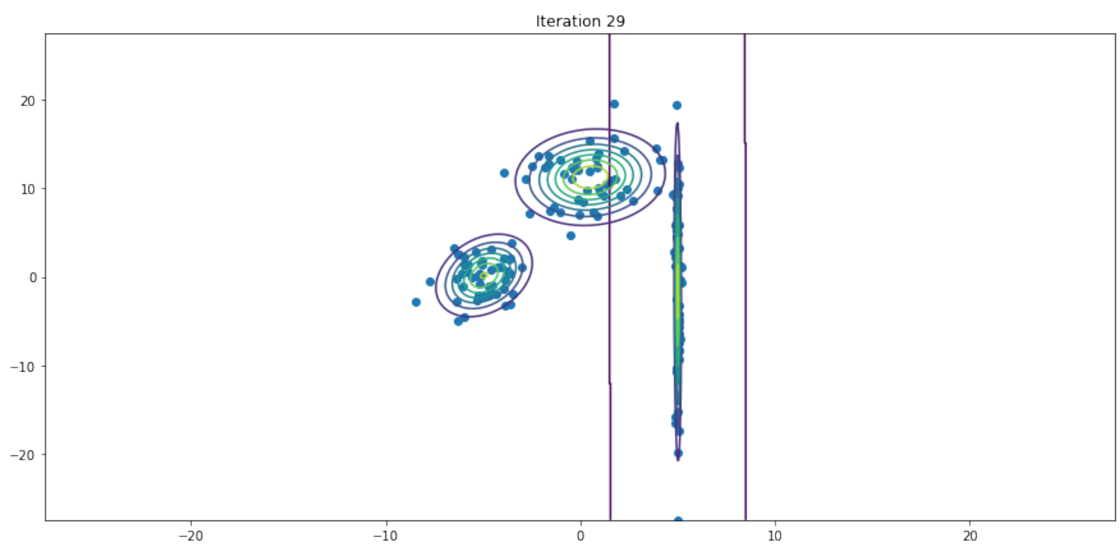

Iteration 24
20
10
0
−10
−20
−20
−10
0
10
20

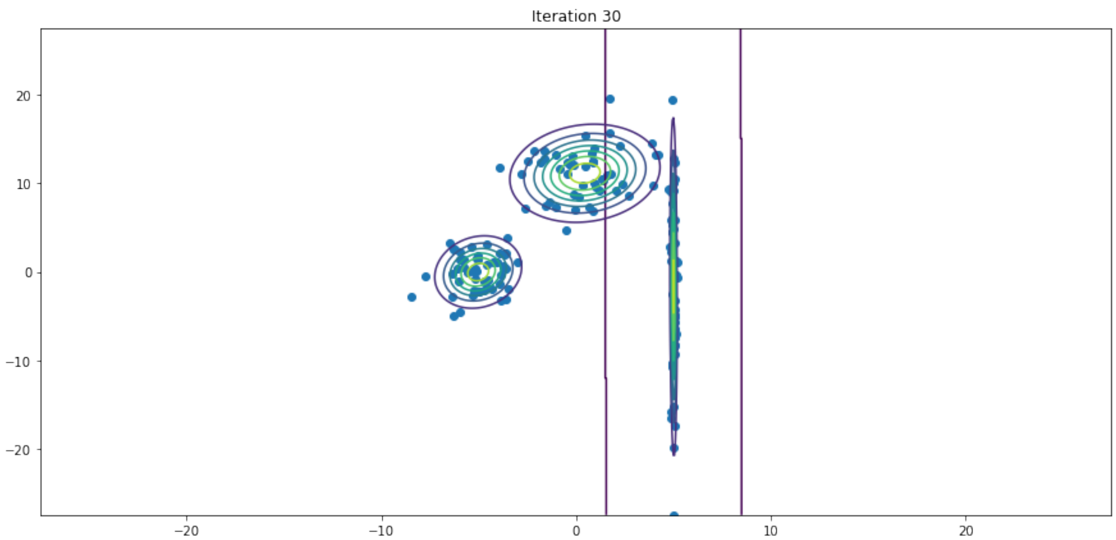

Iteration 25
20
10
0
−10
−20
−20
−10
0
10
20


Iteration 26
20
10
0
−10
−20
−20
−10
0
10
20

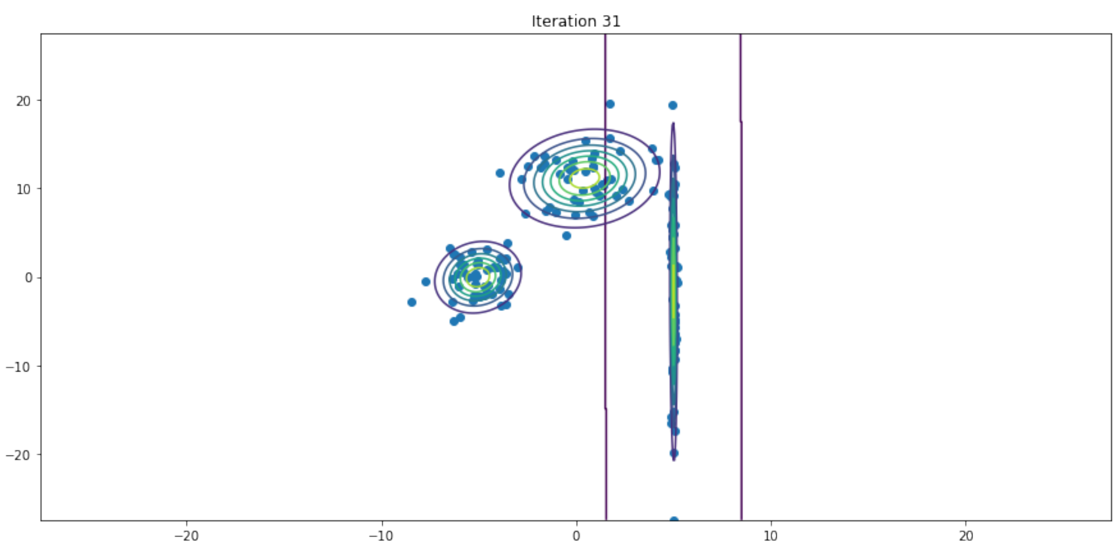

Iteration 27
20
10
0
−10
−20
−20
−10
0
10
20


Iteration 28
20
10
0
−10
−20
−20
−10
0
10
20

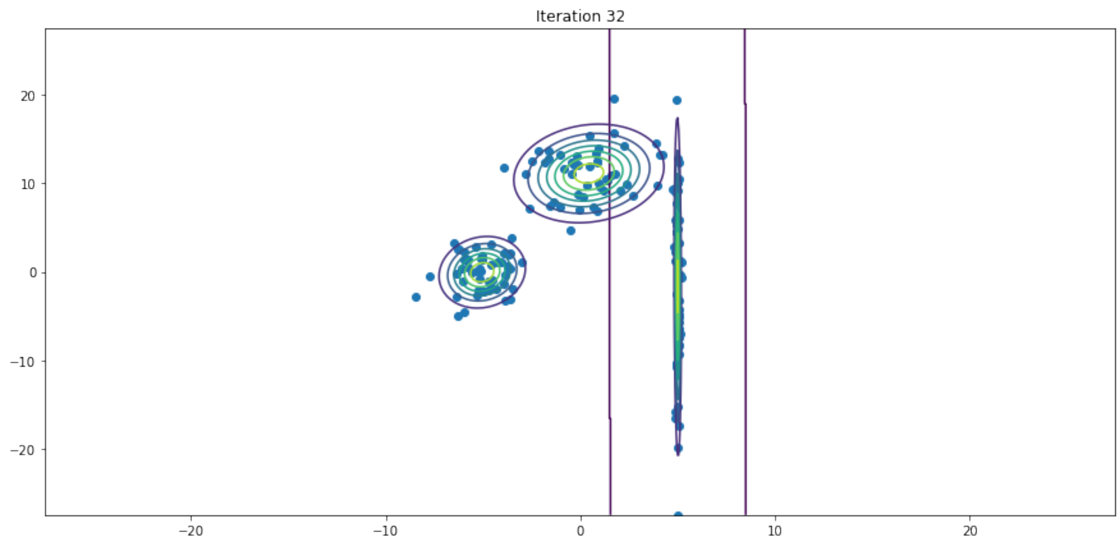

Iteration 29
20
10
0
−10
−20
−20
−10
0
10
20

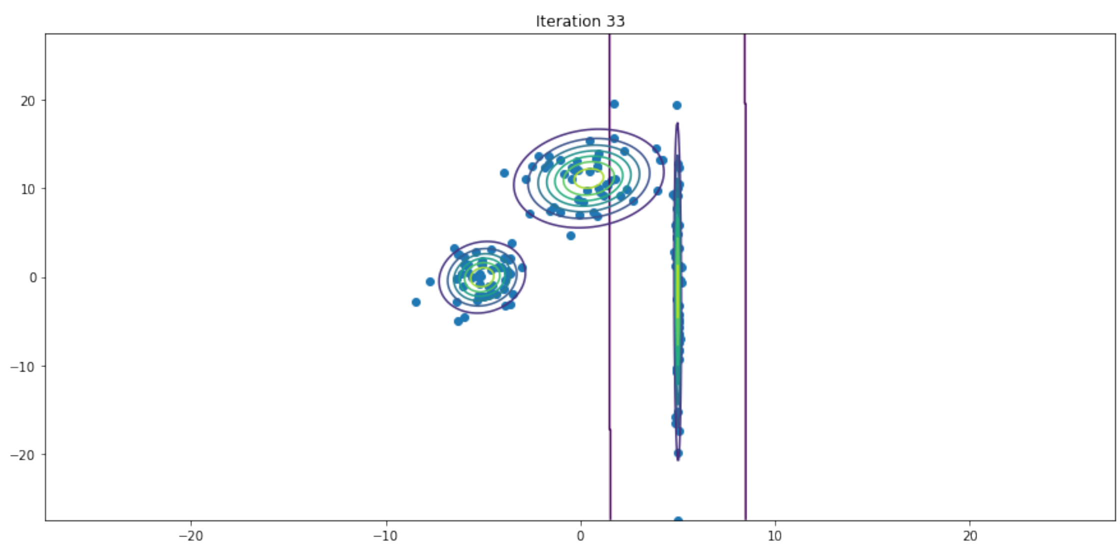


Iteration 30
20
10
0
−10
−20
−20
−10
0
10
20



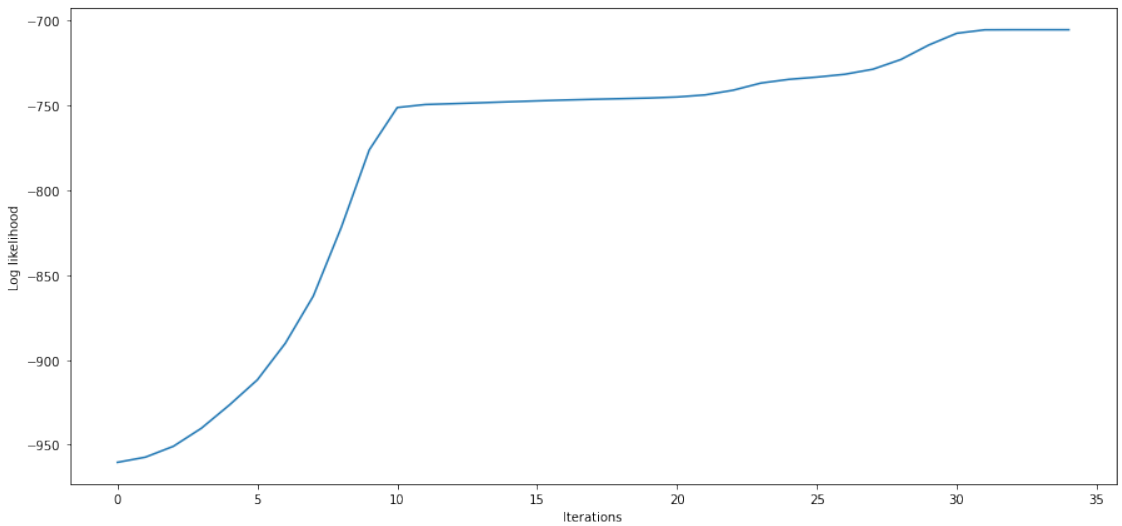
Iteration 31
20
10
0
−10
−20
−20
−10
0
10
20





[20]:
```
plt.figure(figsize=(15,7))
plt.plot(log_likelihood_train)
plt.xlabel('Iterations')
plt.ylabel('Log likelihood')
```

[20]: Text(0, 0.5, 'Log likelihood')



1.2 TODO 7: explain the results you got (max 10 lines)

1-D dataset: - Compare plots in to do 4 with the ones in to do 6, what has changed? Is EM providing us a meaningful clustering?

2-D dataset: - Why is the log likelihood monotonically increasing? Is this what you expect from the theory? Compare both log likelihood trajectory and 2-d plots. - Is delta log likelihood monotonically going to zero?
- Which termination criterion is met first?

(Answer in the next cell, no need to add code)

1.3 # YOUR CODE HERE

1-D Dataset

+ In TODO 4 the normal distributions have been chosen randomly s.t. 1 cluster is never used. In TODO 6 we can see the "ML distributions" which do provide a meaningful 3-class classification.

2-D Dataset

+ We saw in class that the log likelihood must be increasing since:

$$\begin{cases} Q(\vartheta, \hat{\vartheta}^{(k)}) \leq \log P_\vartheta(x_1, \dots, x_m) + c \\ Q(\hat{\vartheta}^{(k)}, \hat{\vartheta}^{(k)}) = \log P_{\hat{\vartheta}^{(k)}}(x_1, \dots, x_m) + c \end{cases} \implies \log P_{\hat{\vartheta}^{(k+1)}}(x_1, \dots, x_m$$

1-D Dataset

+ In TODO 4 the normal distributions have been chosen randomly s.t. 1 cluster is never used. In TODO 6 we can see the "ML distributions" which do provide a meaningful 3-class classification.

+ Note how in the 2d-plots the blue points are "quickly classified" before 10-th iteration and how the log

---

1-D Dataset

+ In TODO 4 the normal distributions have been chosen randomly s.t. 1 cluster is never used. In TODO 6 we can see the “ML distributions” which do provide a meaningful 3-class classification.

+ No, Δ log is not monotonic. See e.g. the Δ log variation from 6-th to 9-th iteration (or also the *log* plot which

1-D Dataset

+ In TODO 4 the normal distributions have been chosen randomly s.t. 1 cluster is never used. In TODO 6 we can see the “ML distributions” which do provide a meaningful 3-class classification.

+ Clearly the “improvement $< \varepsilon$” criterion is met first since it stops after only 33 iterations (“fast”

1-D Dataset + In TODO 4 the normal distributions have been chosen randomly s.t. 1 cluster is never used. In TODO 6 we can see the "ML distributions" which do provide a meaningful 3-class classification.

---

[21]:
```
# What is the effect of choosing a different number of classes?
centers = [[-2, 0], [0, 3], [2, -1]]
X, Y = make_blobs(cluster_std=clusters_cov, centers=centers, random_state=20,
 ↪n_samples=150, shuffle=True)

Ks = [2, 3, 4, 5, 6]
results = []
for k in Ks:
    results.append(run_EM_on_GMM(X, k, 150, plot_intermediate=False))
```

```
Iteration 0, log likelihood -796.0086,  delta log likelihood 3.9850
Iteration 1, log likelihood -790.5347,  delta log likelihood 5.4739
Iteration 2, log likelihood -784.6259,  delta log likelihood 5.9088
Iteration 3, log likelihood -777.4625,  delta log likelihood 7.1634
Iteration 4, log likelihood -768.0228,  delta log likelihood 9.4397
Iteration 5, log likelihood -755.2942,  delta log likelihood 12.7286
Iteration 6, log likelihood -740.6602,  delta log likelihood 14.6340
Iteration 7, log likelihood -731.5797,  delta log likelihood 9.0806
Iteration 8, log likelihood -726.7311,  delta log likelihood 4.8486
Iteration 9, log likelihood -717.1725,  delta log likelihood 9.5585
Iteration 10, log likelihood -699.3376,  delta log likelihood 17.8350
Iteration 11, log likelihood -682.6041,  delta log likelihood 16.7334
Iteration 12, log likelihood -675.7879,  delta log likelihood 6.8162
Iteration 13, log likelihood -674.4362,  delta log likelihood 1.3517
Iteration 14, log likelihood -674.2045,  delta log likelihood 0.2317
Iteration 15, log likelihood -674.1808,  delta log likelihood 0.0237
Iteration 16, log likelihood -674.1782,  delta log likelihood 0.0026
Iteration 17, log likelihood -674.1779,  delta log likelihood 0.0003
Iteration 18, log likelihood -674.1779,  delta log likelihood 0.0000
Iteration 0, log likelihood -796.7393,  delta log likelihood 3.6018
Iteration 1, log likelihood -790.2656,  delta log likelihood 6.4736
Iteration 2, log likelihood -781.7849,  delta log likelihood 8.4807
Iteration 3, log likelihood -772.9982,  delta log likelihood 8.7867
Iteration 4, log likelihood -762.2053,  delta log likelihood 10.7929
Iteration 5, log likelihood -745.6689,  delta log likelihood 16.5364
Iteration 6, log likelihood -723.9224,  delta log likelihood 21.7465
Iteration 7, log likelihood -696.1412,  delta log likelihood 27.7812
Iteration 8, log likelihood -677.2345,  delta log likelihood 18.9067
Iteration 9, log likelihood -670.9121,  delta log likelihood 6.3225
Iteration 10, log likelihood -669.3490,  delta log likelihood 1.5631
Iteration 11, log likelihood -668.7266,  delta log likelihood 0.6224
Iteration 12, log likelihood -668.3147,  delta log likelihood 0.4119
Iteration 13, log likelihood -667.9477,  delta log likelihood 0.3670
Iteration 14, log likelihood -667.5879,  delta log likelihood 0.3598
Iteration 15, log likelihood -667.2150,  delta log likelihood 0.3729
Iteration 16, log likelihood -666.8132,  delta log likelihood 0.4017
Iteration 17, log likelihood -666.3694,  delta log likelihood 0.4438
Iteration 18, log likelihood -665.8697,  delta log likelihood 0.4997
Iteration 19, log likelihood -665.2950,  delta log likelihood 0.5747
Iteration 20, log likelihood -664.6234,  delta log likelihood 0.6716
Iteration 21, log likelihood -663.8491,  delta log likelihood 0.7743
Iteration 22, log likelihood -663.0190,  delta log likelihood 0.8301
Iteration 23, log likelihood -662.2403,  delta log likelihood 0.7787
Iteration 24, log likelihood -661.6115,  delta log likelihood 0.6287
Iteration 25, log likelihood -661.1509,  delta log likelihood 0.4606
Iteration 26, log likelihood -660.8210,  delta log likelihood 0.3299
Iteration 27, log likelihood -660.5875,  delta log likelihood 0.2335
Iteration 28, log likelihood -660.4259,  delta log likelihood 0.1616
```

```
Iteration 29, log likelihood -660.3131,  delta log likelihood 0.1128
Iteration 30, log likelihood -660.2321,  delta log likelihood 0.0810
Iteration 31, log likelihood -660.1732,  delta log likelihood 0.0589
Iteration 32, log likelihood -660.1304,  delta log likelihood 0.0428
Iteration 33, log likelihood -660.0996,  delta log likelihood 0.0308
Iteration 34, log likelihood -660.0777,  delta log likelihood 0.0219
Iteration 35, log likelihood -660.0623,  delta log likelihood 0.0154
Iteration 36, log likelihood -660.0515,  delta log likelihood 0.0108
Iteration 37, log likelihood -660.0440,  delta log likelihood 0.0075
Iteration 38, log likelihood -660.0389,  delta log likelihood 0.0052
Iteration 39, log likelihood -660.0353,  delta log likelihood 0.0035
Iteration 40, log likelihood -660.0329,  delta log likelihood 0.0024
Iteration 41, log likelihood -660.0312,  delta log likelihood 0.0017
Iteration 42, log likelihood -660.0301,  delta log likelihood 0.0011
Iteration 43, log likelihood -660.0293,  delta log likelihood 0.0008
Iteration 44, log likelihood -660.0288,  delta log likelihood 0.0005
Iteration 0, log likelihood -788.6428,  delta log likelihood 8.5240
Iteration 1, log likelihood -776.9729,  delta log likelihood 11.6698
Iteration 2, log likelihood -761.0779,  delta log likelihood 15.8950
Iteration 3, log likelihood -739.8428,  delta log likelihood 21.2351
Iteration 4, log likelihood -714.0605,  delta log likelihood 25.7822
Iteration 5, log likelihood -688.9897,  delta log likelihood 25.0709
Iteration 6, log likelihood -677.0195,  delta log likelihood 11.9702
Iteration 7, log likelihood -674.6182,  delta log likelihood 2.4012
Iteration 8, log likelihood -674.1699,  delta log likelihood 0.4484
Iteration 9, log likelihood -674.1006,  delta log likelihood 0.0693
Iteration 10, log likelihood -674.0646,  delta log likelihood 0.0360
Iteration 11, log likelihood -674.0069,  delta log likelihood 0.0577
Iteration 12, log likelihood -673.8837,  delta log likelihood 0.1232
Iteration 13, log likelihood -673.6063,  delta log likelihood 0.2773
Iteration 14, log likelihood -672.9992,  delta log likelihood 0.6072
Iteration 15, log likelihood -671.8237,  delta log likelihood 1.1754
Iteration 16, log likelihood -670.0655,  delta log likelihood 1.7582
Iteration 17, log likelihood -668.1432,  delta log likelihood 1.9223
Iteration 18, log likelihood -666.3702,  delta log likelihood 1.7730
Iteration 19, log likelihood -664.8346,  delta log likelihood 1.5356
Iteration 20, log likelihood -663.6281,  delta log likelihood 1.2065
Iteration 21, log likelihood -662.7979,  delta log likelihood 0.8302
Iteration 22, log likelihood -662.2714,  delta log likelihood 0.5265
Iteration 23, log likelihood -661.9266,  delta log likelihood 0.3448
Iteration 24, log likelihood -661.6733,  delta log likelihood 0.2534
Iteration 25, log likelihood -661.4631,  delta log likelihood 0.2101
Iteration 26, log likelihood -661.2735,  delta log likelihood 0.1897
Iteration 27, log likelihood -661.0937,  delta log likelihood 0.1798
Iteration 28, log likelihood -660.9187,  delta log likelihood 0.1750
Iteration 29, log likelihood -660.7463,  delta log likelihood 0.1724
Iteration 30, log likelihood -660.5767,  delta log likelihood 0.1696
Iteration 31, log likelihood -660.4123,  delta log likelihood 0.1644
```

```
Iteration 32, log likelihood -660.2577,  delta log likelihood 0.1546
Iteration 33, log likelihood -660.1188,  delta log likelihood 0.1390
Iteration 34, log likelihood -660.0006,  delta log likelihood 0.1181
Iteration 35, log likelihood -659.9061,  delta log likelihood 0.0945
Iteration 36, log likelihood -659.8346,  delta log likelihood 0.0715
Iteration 37, log likelihood -659.7827,  delta log likelihood 0.0519
Iteration 38, log likelihood -659.7458,  delta log likelihood 0.0369
Iteration 39, log likelihood -659.7194,  delta log likelihood 0.0264
Iteration 40, log likelihood -659.6999,  delta log likelihood 0.0196
Iteration 41, log likelihood -659.6844,  delta log likelihood 0.0154
Iteration 42, log likelihood -659.6715,  delta log likelihood 0.0130
Iteration 43, log likelihood -659.6598,  delta log likelihood 0.0116
Iteration 44, log likelihood -659.6488,  delta log likelihood 0.0110
Iteration 45, log likelihood -659.6381,  delta log likelihood 0.0108
Iteration 46, log likelihood -659.6273,  delta log likelihood 0.0108
Iteration 47, log likelihood -659.6164,  delta log likelihood 0.0109
Iteration 48, log likelihood -659.6051,  delta log likelihood 0.0112
Iteration 49, log likelihood -659.5936,  delta log likelihood 0.0116
Iteration 50, log likelihood -659.5816,  delta log likelihood 0.0120
Iteration 51, log likelihood -659.5691,  delta log likelihood 0.0125
Iteration 52, log likelihood -659.5562,  delta log likelihood 0.0130
Iteration 53, log likelihood -659.5427,  delta log likelihood 0.0135
Iteration 54, log likelihood -659.5285,  delta log likelihood 0.0141
Iteration 55, log likelihood -659.5138,  delta log likelihood 0.0148
Iteration 56, log likelihood -659.4983,  delta log likelihood 0.0155
Iteration 57, log likelihood -659.4821,  delta log likelihood 0.0162
Iteration 58, log likelihood -659.4651,  delta log likelihood 0.0170
Iteration 59, log likelihood -659.4473,  delta log likelihood 0.0178
Iteration 60, log likelihood -659.4288,  delta log likelihood 0.0185
Iteration 61, log likelihood -659.4097,  delta log likelihood 0.0191
Iteration 62, log likelihood -659.3901,  delta log likelihood 0.0196
Iteration 63, log likelihood -659.3704,  delta log likelihood 0.0197
Iteration 64, log likelihood -659.3510,  delta log likelihood 0.0194
Iteration 65, log likelihood -659.3325,  delta log likelihood 0.0186
Iteration 66, log likelihood -659.3152,  delta log likelihood 0.0173
Iteration 67, log likelihood -659.2997,  delta log likelihood 0.0155
Iteration 68, log likelihood -659.2862,  delta log likelihood 0.0135
Iteration 69, log likelihood -659.2748,  delta log likelihood 0.0114
Iteration 70, log likelihood -659.2654,  delta log likelihood 0.0094
Iteration 71, log likelihood -659.2578,  delta log likelihood 0.0076
Iteration 72, log likelihood -659.2517,  delta log likelihood 0.0061
Iteration 73, log likelihood -659.2468,  delta log likelihood 0.0049
Iteration 74, log likelihood -659.2429,  delta log likelihood 0.0039
Iteration 75, log likelihood -659.2398,  delta log likelihood 0.0032
Iteration 76, log likelihood -659.2372,  delta log likelihood 0.0026
Iteration 77, log likelihood -659.2350,  delta log likelihood 0.0022
Iteration 78, log likelihood -659.2331,  delta log likelihood 0.0019
Iteration 79, log likelihood -659.2315,  delta log likelihood 0.0016
```

```
Iteration 80, log likelihood -659.2300,  delta log likelihood 0.0015
Iteration 81, log likelihood -659.2287,  delta log likelihood 0.0013
Iteration 82, log likelihood -659.2275,  delta log likelihood 0.0012
Iteration 83, log likelihood -659.2264,  delta log likelihood 0.0011
Iteration 84, log likelihood -659.2253,  delta log likelihood 0.0011
Iteration 85, log likelihood -659.2242,  delta log likelihood 0.0010
Iteration 86, log likelihood -659.2232,  delta log likelihood 0.0010
Iteration 87, log likelihood -659.2222,  delta log likelihood 0.0010
Iteration 88, log likelihood -659.2213,  delta log likelihood 0.0010
Iteration 0, log likelihood -796.7774,  delta log likelihood 3.8161
Iteration 1, log likelihood -788.9660,  delta log likelihood 7.8115
Iteration 2, log likelihood -775.2265,  delta log likelihood 13.7394
Iteration 3, log likelihood -753.8635,  delta log likelihood 21.3630
Iteration 4, log likelihood -726.3080,  delta log likelihood 27.5555
Iteration 5, log likelihood -693.7906,  delta log likelihood 32.5174
Iteration 6, log likelihood -675.1348,  delta log likelihood 18.6558
Iteration 7, log likelihood -669.8566,  delta log likelihood 5.2782
Iteration 8, log likelihood -667.8325,  delta log likelihood 2.0241
Iteration 9, log likelihood -666.6112,  delta log likelihood 1.2213
Iteration 10, log likelihood -665.5425,  delta log likelihood 1.0687
Iteration 11, log likelihood -664.4191,  delta log likelihood 1.1234
Iteration 12, log likelihood -663.1906,  delta log likelihood 1.2284
Iteration 13, log likelihood -661.9796,  delta log likelihood 1.2110
Iteration 14, log likelihood -660.9829,  delta log likelihood 0.9968
Iteration 15, log likelihood -660.2693,  delta log likelihood 0.7136
Iteration 16, log likelihood -659.7810,  delta log likelihood 0.4883
Iteration 17, log likelihood -659.4320,  delta log likelihood 0.3490
Iteration 18, log likelihood -659.1580,  delta log likelihood 0.2741
Iteration 19, log likelihood -658.9231,  delta log likelihood 0.2349
Iteration 20, log likelihood -658.7108,  delta log likelihood 0.2122
Iteration 21, log likelihood -658.5153,  delta log likelihood 0.1956
Iteration 22, log likelihood -658.3357,  delta log likelihood 0.1796
Iteration 23, log likelihood -658.1734,  delta log likelihood 0.1623
Iteration 24, log likelihood -658.0296,  delta log likelihood 0.1438
Iteration 25, log likelihood -657.9041,  delta log likelihood 0.1254
Iteration 26, log likelihood -657.7952,  delta log likelihood 0.1090
Iteration 27, log likelihood -657.6996,  delta log likelihood 0.0955
Iteration 28, log likelihood -657.6141,  delta log likelihood 0.0855
Iteration 29, log likelihood -657.5353,  delta log likelihood 0.0788
Iteration 30, log likelihood -657.4605,  delta log likelihood 0.0749
Iteration 31, log likelihood -657.3872,  delta log likelihood 0.0733
Iteration 32, log likelihood -657.3133,  delta log likelihood 0.0739
Iteration 33, log likelihood -657.2369,  delta log likelihood 0.0764
Iteration 34, log likelihood -657.1558,  delta log likelihood 0.0811
Iteration 35, log likelihood -657.0679,  delta log likelihood 0.0879
Iteration 36, log likelihood -656.9705,  delta log likelihood 0.0974
Iteration 37, log likelihood -656.8607,  delta log likelihood 0.1098
Iteration 38, log likelihood -656.7347,  delta log likelihood 0.1259
```

```
Iteration 39, log likelihood -656.5885,  delta log likelihood 0.1463
Iteration 40, log likelihood -656.4170,  delta log likelihood 0.1715
Iteration 41, log likelihood -656.2146,  delta log likelihood 0.2023
Iteration 42, log likelihood -655.9756,  delta log likelihood 0.2391
Iteration 43, log likelihood -655.6946,  delta log likelihood 0.2810
Iteration 44, log likelihood -655.3695,  delta log likelihood 0.3250
Iteration 45, log likelihood -655.0067,  delta log likelihood 0.3628
Iteration 46, log likelihood -654.6261,  delta log likelihood 0.3805
Iteration 47, log likelihood -654.2605,  delta log likelihood 0.3656
Iteration 48, log likelihood -653.9403,  delta log likelihood 0.3202
Iteration 49, log likelihood -653.6784,  delta log likelihood 0.2619
Iteration 50, log likelihood -653.4721,  delta log likelihood 0.2062
Iteration 51, log likelihood -653.3136,  delta log likelihood 0.1586
Iteration 52, log likelihood -653.1943,  delta log likelihood 0.1193
Iteration 53, log likelihood -653.1062,  delta log likelihood 0.0881
Iteration 54, log likelihood -653.0414,  delta log likelihood 0.0647
Iteration 55, log likelihood -652.9936,  delta log likelihood 0.0479
Iteration 56, log likelihood -652.9575,  delta log likelihood 0.0360
Iteration 57, log likelihood -652.9298,  delta log likelihood 0.0277
Iteration 58, log likelihood -652.9079,  delta log likelihood 0.0219
Iteration 59, log likelihood -652.8902,  delta log likelihood 0.0177
Iteration 60, log likelihood -652.8756,  delta log likelihood 0.0147
Iteration 61, log likelihood -652.8631,  delta log likelihood 0.0124
Iteration 62, log likelihood -652.8524,  delta log likelihood 0.0107
Iteration 63, log likelihood -652.8429,  delta log likelihood 0.0095
Iteration 64, log likelihood -652.8344,  delta log likelihood 0.0085
Iteration 65, log likelihood -652.8266,  delta log likelihood 0.0078
Iteration 66, log likelihood -652.8193,  delta log likelihood 0.0073
Iteration 67, log likelihood -652.8124,  delta log likelihood 0.0069
Iteration 68, log likelihood -652.8058,  delta log likelihood 0.0066
Iteration 69, log likelihood -652.7993,  delta log likelihood 0.0065
Iteration 70, log likelihood -652.7930,  delta log likelihood 0.0064
Iteration 71, log likelihood -652.7866,  delta log likelihood 0.0064
Iteration 72, log likelihood -652.7802,  delta log likelihood 0.0064
Iteration 73, log likelihood -652.7736,  delta log likelihood 0.0065
Iteration 74, log likelihood -652.7669,  delta log likelihood 0.0067
Iteration 75, log likelihood -652.7600,  delta log likelihood 0.0069
Iteration 76, log likelihood -652.7528,  delta log likelihood 0.0072
Iteration 77, log likelihood -652.7452,  delta log likelihood 0.0075
Iteration 78, log likelihood -652.7373,  delta log likelihood 0.0079
Iteration 79, log likelihood -652.7289,  delta log likelihood 0.0084
Iteration 80, log likelihood -652.7200,  delta log likelihood 0.0089
Iteration 81, log likelihood -652.7105,  delta log likelihood 0.0095
Iteration 82, log likelihood -652.7004,  delta log likelihood 0.0102
Iteration 83, log likelihood -652.6894,  delta log likelihood 0.0109
Iteration 84, log likelihood -652.6777,  delta log likelihood 0.0118
Iteration 85, log likelihood -652.6649,  delta log likelihood 0.0128
Iteration 86, log likelihood -652.6511,  delta log likelihood 0.0138
```

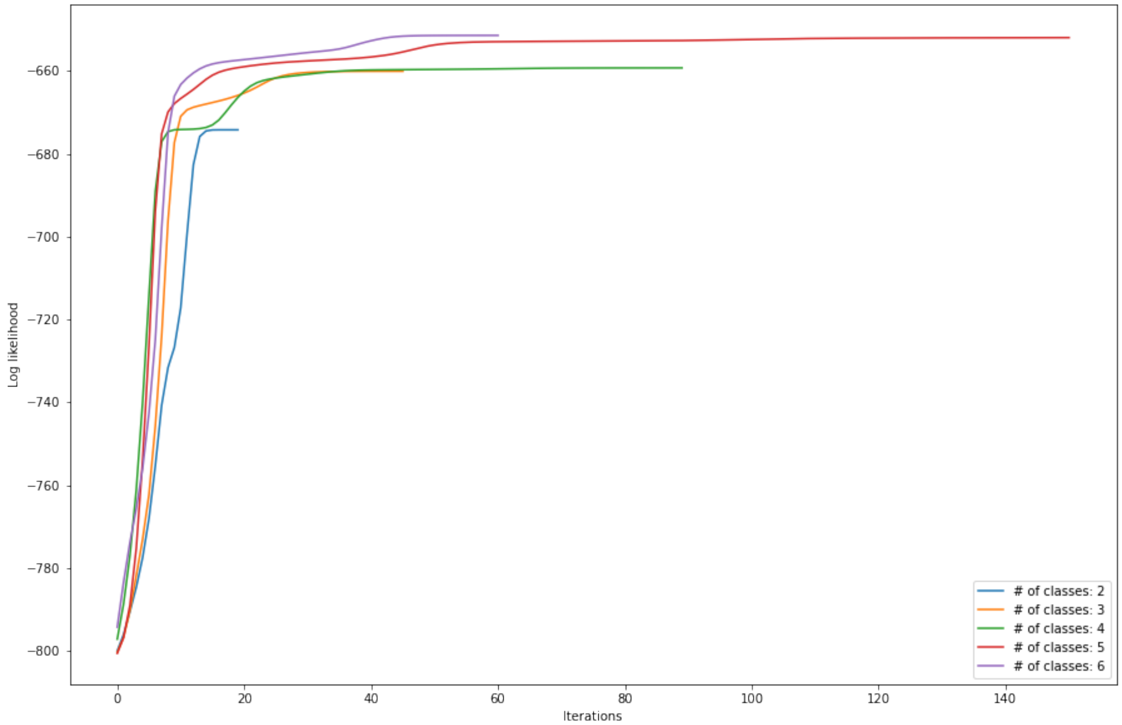
```
Iteration 87, log likelihood -652.6360,  delta log likelihood 0.0150
Iteration 88, log likelihood -652.6197,  delta log likelihood 0.0164
Iteration 89, log likelihood -652.6019,  delta log likelihood 0.0178
Iteration 90, log likelihood -652.5825,  delta log likelihood 0.0194
Iteration 91, log likelihood -652.5615,  delta log likelihood 0.0210
Iteration 92, log likelihood -652.5387,  delta log likelihood 0.0227
Iteration 93, log likelihood -652.5143,  delta log likelihood 0.0244
Iteration 94, log likelihood -652.4882,  delta log likelihood 0.0261
Iteration 95, log likelihood -652.4606,  delta log likelihood 0.0276
Iteration 96, log likelihood -652.4316,  delta log likelihood 0.0289
Iteration 97, log likelihood -652.4018,  delta log likelihood 0.0299
Iteration 98, log likelihood -652.3714,  delta log likelihood 0.0304
Iteration 99, log likelihood -652.3410,  delta log likelihood 0.0304
Iteration 100, log likelihood -652.3112,  delta log likelihood 0.0299
Iteration 101, log likelihood -652.2823,  delta log likelihood 0.0288
Iteration 102, log likelihood -652.2550,  delta log likelihood 0.0273
Iteration 103, log likelihood -652.2294,  delta log likelihood 0.0255
Iteration 104, log likelihood -652.2059,  delta log likelihood 0.0235
Iteration 105, log likelihood -652.1844,  delta log likelihood 0.0215
Iteration 106, log likelihood -652.1649,  delta log likelihood 0.0195
Iteration 107, log likelihood -652.1473,  delta log likelihood 0.0176
Iteration 108, log likelihood -652.1313,  delta log likelihood 0.0160
Iteration 109, log likelihood -652.1168,  delta log likelihood 0.0145
Iteration 110, log likelihood -652.1035,  delta log likelihood 0.0132
Iteration 111, log likelihood -652.0914,  delta log likelihood 0.0121
Iteration 112, log likelihood -652.0802,  delta log likelihood 0.0112
Iteration 113, log likelihood -652.0698,  delta log likelihood 0.0104
Iteration 114, log likelihood -652.0601,  delta log likelihood 0.0097
Iteration 115, log likelihood -652.0511,  delta log likelihood 0.0090
Iteration 116, log likelihood -652.0426,  delta log likelihood 0.0085
Iteration 117, log likelihood -652.0347,  delta log likelihood 0.0080
Iteration 118, log likelihood -652.0272,  delta log likelihood 0.0075
Iteration 119, log likelihood -652.0201,  delta log likelihood 0.0071
Iteration 120, log likelihood -652.0134,  delta log likelihood 0.0067
Iteration 121, log likelihood -652.0070,  delta log likelihood 0.0063
Iteration 122, log likelihood -652.0010,  delta log likelihood 0.0060
Iteration 123, log likelihood -651.9953,  delta log likelihood 0.0057
Iteration 124, log likelihood -651.9899,  delta log likelihood 0.0054
Iteration 125, log likelihood -651.9847,  delta log likelihood 0.0052
Iteration 126, log likelihood -651.9797,  delta log likelihood 0.0049
Iteration 127, log likelihood -651.9750,  delta log likelihood 0.0047
Iteration 128, log likelihood -651.9705,  delta log likelihood 0.0045
Iteration 129, log likelihood -651.9662,  delta log likelihood 0.0043
Iteration 130, log likelihood -651.9621,  delta log likelihood 0.0041
Iteration 131, log likelihood -651.9582,  delta log likelihood 0.0039
Iteration 132, log likelihood -651.9544,  delta log likelihood 0.0038
Iteration 133, log likelihood -651.9508,  delta log likelihood 0.0036
Iteration 134, log likelihood -651.9473,  delta log likelihood 0.0035
```

```
Iteration 135, log likelihood -651.9439,  delta log likelihood 0.0033
Iteration 136, log likelihood -651.9407,  delta log likelihood 0.0032
Iteration 137, log likelihood -651.9377,  delta log likelihood 0.0031
Iteration 138, log likelihood -651.9347,  delta log likelihood 0.0029
Iteration 139, log likelihood -651.9319,  delta log likelihood 0.0028
Iteration 140, log likelihood -651.9292,  delta log likelihood 0.0027
Iteration 141, log likelihood -651.9266,  delta log likelihood 0.0026
Iteration 142, log likelihood -651.9241,  delta log likelihood 0.0025
Iteration 143, log likelihood -651.9217,  delta log likelihood 0.0024
Iteration 144, log likelihood -651.9194,  delta log likelihood 0.0023
Iteration 145, log likelihood -651.9172,  delta log likelihood 0.0022
Iteration 146, log likelihood -651.9151,  delta log likelihood 0.0021
Iteration 147, log likelihood -651.9131,  delta log likelihood 0.0020
Iteration 148, log likelihood -651.9112,  delta log likelihood 0.0019
Iteration 149, log likelihood -651.9093,  delta log likelihood 0.0018
Iteration 0, log likelihood -783.3534,  delta log likelihood 10.9174
Iteration 1, log likelihood -773.8027,  delta log likelihood 9.5507
Iteration 2, log likelihood -765.5965,  delta log likelihood 8.2062
Iteration 3, log likelihood -755.9113,  delta log likelihood 9.6852
Iteration 4, log likelihood -742.5435,  delta log likelihood 13.3679
Iteration 5, log likelihood -724.8649,  delta log likelihood 17.6785
Iteration 6, log likelihood -697.9076,  delta log likelihood 26.9573
Iteration 7, log likelihood -674.8653,  delta log likelihood 23.0423
Iteration 8, log likelihood -666.0494,  delta log likelihood 8.8160
Iteration 9, log likelihood -663.2345,  delta log likelihood 2.8148
Iteration 10, log likelihood -661.6695,  delta log likelihood 1.5650
Iteration 11, log likelihood -660.4343,  delta log likelihood 1.2353
Iteration 12, log likelihood -659.4531,  delta log likelihood 0.9812
Iteration 13, log likelihood -658.7458,  delta log likelihood 0.7073
Iteration 14, log likelihood -658.2761,  delta log likelihood 0.4696
Iteration 15, log likelihood -657.9635,  delta log likelihood 0.3126
Iteration 16, log likelihood -657.7360,  delta log likelihood 0.2275
Iteration 17, log likelihood -657.5493,  delta log likelihood 0.1868
Iteration 18, log likelihood -657.3800,  delta log likelihood 0.1692
Iteration 19, log likelihood -657.2168,  delta log likelihood 0.1632
Iteration 20, log likelihood -657.0540,  delta log likelihood 0.1629
Iteration 21, log likelihood -656.8887,  delta log likelihood 0.1653
Iteration 22, log likelihood -656.7201,  delta log likelihood 0.1686
Iteration 23, log likelihood -656.5482,  delta log likelihood 0.1718
Iteration 24, log likelihood -656.3743,  delta log likelihood 0.1740
Iteration 25, log likelihood -656.1995,  delta log likelihood 0.1747
Iteration 26, log likelihood -656.0256,  delta log likelihood 0.1739
Iteration 27, log likelihood -655.8543,  delta log likelihood 0.1713
Iteration 28, log likelihood -655.6875,  delta log likelihood 0.1668
Iteration 29, log likelihood -655.5272,  delta log likelihood 0.1603
Iteration 30, log likelihood -655.3738,  delta log likelihood 0.1534
Iteration 31, log likelihood -655.2237,  delta log likelihood 0.1500
Iteration 32, log likelihood -655.0669,  delta log likelihood 0.1569
```

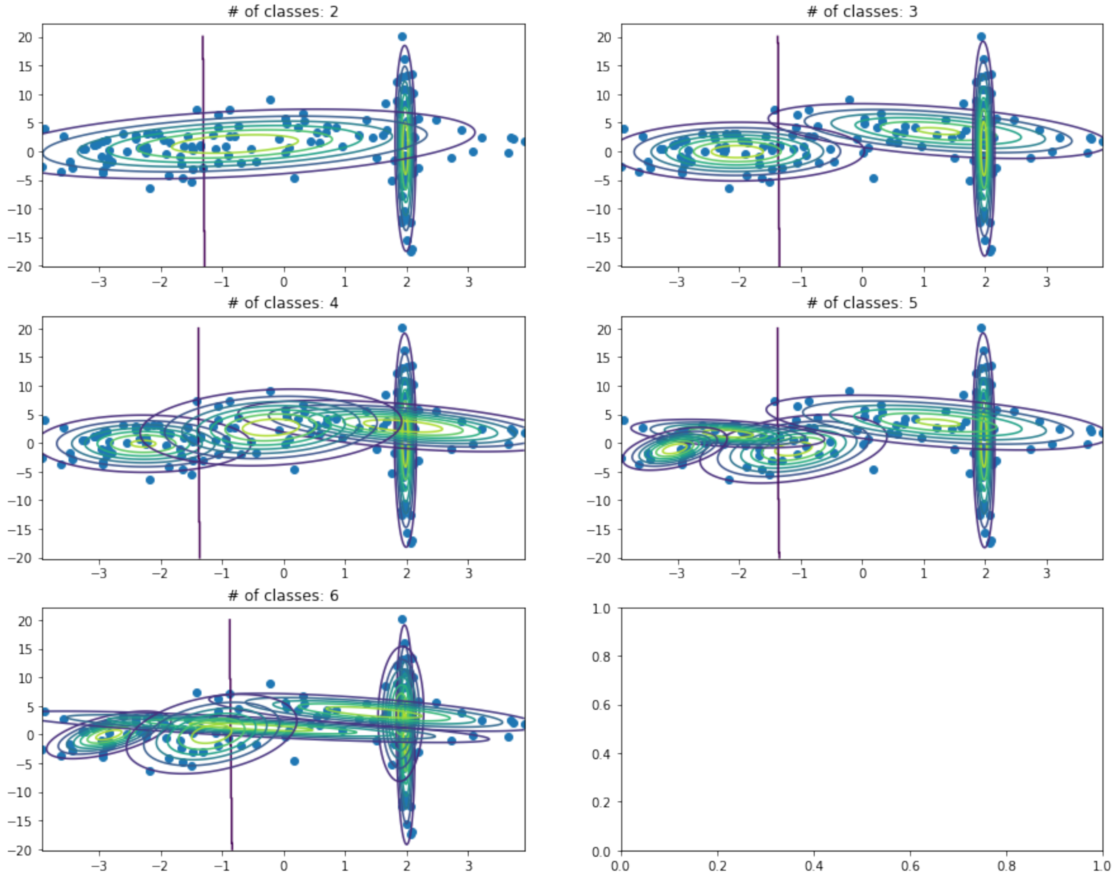
```
Iteration 33, log likelihood -654.8835,  delta log likelihood 0.1833
Iteration 34, log likelihood -654.6437,  delta log likelihood 0.2398
Iteration 35, log likelihood -654.3174,  delta log likelihood 0.3263
Iteration 36, log likelihood -653.9076,  delta log likelihood 0.4098
Iteration 37, log likelihood -653.4683,  delta log likelihood 0.4394
Iteration 38, log likelihood -653.0505,  delta log likelihood 0.4177
Iteration 39, log likelihood -652.6692,  delta log likelihood 0.3813
Iteration 40, log likelihood -652.3290,  delta log likelihood 0.3402
Iteration 41, log likelihood -652.0412,  delta log likelihood 0.2879
Iteration 42, log likelihood -651.8177,  delta log likelihood 0.2235
Iteration 43, log likelihood -651.6599,  delta log likelihood 0.1577
Iteration 44, log likelihood -651.5567,  delta log likelihood 0.1033
Iteration 45, log likelihood -651.4916,  delta log likelihood 0.0650
Iteration 46, log likelihood -651.4510,  delta log likelihood 0.0406
Iteration 47, log likelihood -651.4255,  delta log likelihood 0.0255
Iteration 48, log likelihood -651.4092,  delta log likelihood 0.0163
Iteration 49, log likelihood -651.3985,  delta log likelihood 0.0107
Iteration 50, log likelihood -651.3913,  delta log likelihood 0.0072
Iteration 51, log likelihood -651.3863,  delta log likelihood 0.0050
Iteration 52, log likelihood -651.3827,  delta log likelihood 0.0036
Iteration 53, log likelihood -651.3801,  delta log likelihood 0.0026
Iteration 54, log likelihood -651.3781,  delta log likelihood 0.0020
Iteration 55, log likelihood -651.3764,  delta log likelihood 0.0016
Iteration 56, log likelihood -651.3751,  delta log likelihood 0.0013
Iteration 57, log likelihood -651.3740,  delta log likelihood 0.0011
Iteration 58, log likelihood -651.3730,  delta log likelihood 0.0010
Iteration 59, log likelihood -651.3722,  delta log likelihood 0.0009
```

[22]:

```
# Let's plot the log likelihood trajectories and the final 2-d Clustering
plt.figure(figsize=(15,10))
for res, k in zip(results, Ks):
    plt.plot(np.array(res[-1]), label=f'# of classes: {k}')
plt.legend()
plt.xlabel('Iterations')
plt.ylabel('Log likelihood')

x_max, y_max = np.max(np.abs(X[:,0])), np.max(np.abs(X[:,1]))
x, y = np.mgrid[-x_max:x_max:.01, -y_max:y_max:.01]
pos = np.dstack((x, y))

fig, axes = plt.subplots(3, 2, figsize=(15,12))
# Plot scatter plot of training data and corresponding clusters
for (_, pi, means, covs, _), K, ax in zip(results, Ks, axes.reshape(-1,)):
    for k in range(K):
        ax.contour(x, y, multivariate_normal(means[k], covs[k]).pdf(pos), )
    ax.scatter(X[0:,0], X[0:,1])
    ax.set_title(f'# of classes: {K}')
```



Log likelihood
Iterations
−660
−680
−700
−720
−740
−760
−780
−800
0
20
40
60
80
100
120
140
# of classes: 2
# of classes: 3
# of classes: 4
# of classes: 5
# of classes: 6



2 MNIST clustering

Let's apply GMM to a slightly more complex dataset: MNIST (which we already encountered in the last homework).

In the following we shall use the sklearn implementation of the EM.

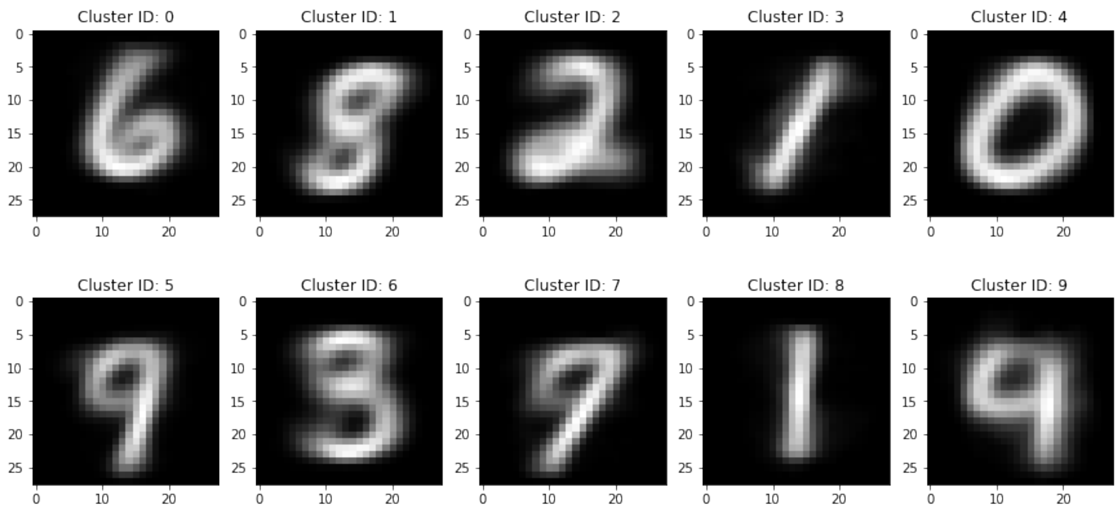
Once we fit the GMM we shall visualize the centers in order to evaluate whether the clustering algorithm came up with a meaningful solution (in an ideal scenario we would expect to have each center representing one single digit).

[23]:

```
from sklearn.datasets import fetch_openml
from sklearn.mixture import GaussianMixture
import sklearn

skver = sklearn.__version__.split('.')
if int(skver[1]) >= 24:
    X, Y = fetch_openml('mnist_784', version=1, return_X_y=True, as_frame=False)
else:
    X, Y = fetch_openml('mnist_784', version=1, return_X_y=True)
```

```
X = X / 255.

from sklearn.model_selection import train_test_split

m_t = 5000
x_train, x_test, y_train, y_test = train_test_split(X, Y, train_size=m_t/
↪len(Y), random_state=ID_number,
                                                    stratify=Y)

# Function to plot a digit and print the corresponding label
def plot_digit(vect_img, ax, cluster_id=None):
    ax.set_title(f'Cluster ID: {cluster_id}')
    ax.imshow(
        vect_img.reshape(28,28),
        cmap          = 'gray',
        interpolation = "nearest"
    )
```

[24]:
```
# TODO 8: use GaussianMixture from skelearn to cluster x_train and then predict␣
↪the labels.
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture

K = 10
# When you initialize the GaussianMixture object use set random_state=ID_number
gmm = None
gmm_pred = None
# YOUR CODE HERE

# Gaussian Mixture Model
gmm = GaussianMixture(n_components=K, random_state=ID_number).fit(x_train)

# GM Predictions
gmm_pred = gmm.predict(x_train)

plt, axes = plt.subplots(2, K // 2, figsize=(15,7))
for k, ax in zip(range(K), axes.reshape(-1,)):
    plot_digit(gmm.means_[k], ax, cluster_id=k)
```



[25]:
```python
assert gmm_pred.shape == (m_t,)
```

3 Comparison with supervised models:

In the next cell we shall compare Unsupervised methods (Kmeans and GMM) with a Supervised method (NNs) on MNIST. Note that supervised methods cannot be applied in the unsupervised setting (i.e. we do not have any label). Nonetheless we can apply an unsupervised method to a supervised problem (we just need to neglect the extra piece of information we have: the labels). In the following we shall train both Kmeans and GMM as if we do not have labels and then we shall compare their predictions to the ground truth labels.

NOTE: in an unsupervised scenario we are not able to compare model predictions with ground truth labels since labels are not part of the problem statement.

[26]:
```python
# TODO 9:
# Remember in ANY unsupervised clustering algorithm the name of the cluster DOES NOT possess any meaning.
# The first class of the GMM model trained on the MNIST dataset does not necessarily represent the digits "0".
# You can see this on the previous plots (plots of the centers of each component).
# Therefore we need to find a way to convert model clusters (predicitons) to ground truth labels. Several choices
# are possible, in the following we shall use a very simple and straightforward rule:
# 1- Find all the indeces of the data belonging to the same cluster predicted by the clustering model
# 2- Use these indeces (in the dataset) to gather the true labels
# 3- Compute the mode on the choosen true labels
```

```
# 4- Assign the mode as the new name of the cluster.
# In this way we are able to compare model predictions with the true labels␣
↪(ground truth labels) and we can
# compute the number of missclassified examples (as we did in previous␣
↪Classification HWs).

from scipy.stats import mode
def convert_prediction_labels(targets : np.ndarray, predictions : np.ndarray,␣
↪num_clusters : int) -> np.ndarray:
    '''
    Function to assign a different label to the predictions of a clustering␣
↪algorithm. Use the 4 steps described
    earlier.
    :param targets: True labels (of shape (N,))
    :param predictions: Labels predicted by the clustering algorithm (of shape␣
↪(N,))
    :param num_clusters: # of clusters in the training dataset
    :returns:
        pred_labels: new labels for each datum (of shape (N,))
    '''
    pred_labels = np.zeros_like(targets)
    for k in range(num_clusters):
        # YOUR CODE HERE

        # find all predicted labels in k-th cluster
        kth_cluster_indeces = np.nonzero(predictions == k)[0]

        # gather the true lables using such indeces
        unique, counts = np.unique(targets.
↪reshape(-1,1)[kth_cluster_indeces,0], return_counts=True)

        # Compute the mode on the choosen true labels
        max_frequency_label = np.argmax(counts)
        right_label = unique[max_frequency_label]

        # Assign the mode as the new name of the cluster
        pred_labels[kth_cluster_indeces] = right_label

    return pred_labels

def compute_score(targets, predictions, num_clusters):
    pred_labels = convert_prediction_labels(targets, predictions, num_clusters)
    errors = sum(pred_labels == targets)
    return (1 - errors / len(targets))
```

[27]:

```
assert convert_prediction_labels(y_train, gmm_pred, 10).shape == (m_t,)
```

[28]:

```python
# TODO 10: Use sklearn GaussianMixture and KMeans to cluster x_train. Then␣
↪evaluate the errors using the groud
# truth labels (y_train) using the functions we built in the previous cell.␣
↪Eventually we compare clustering
# error rates with a supervised classification method: MLP.

# When you initialize the GaussianMixture and KMeans object use set␣
↪random_state=ID_number
gmm, gmm_pred_train, gmm_pred_test = None, None, None
# YOUR CODE HERE

# Gaussian Mixture Model
gmm = GaussianMixture(n_components=K, random_state=ID_number).fit(x_train)

# GM Predictions
gmm_pred_train = gmm.predict(x_train)
gmm_pred_test = gmm.predict(x_test)

gmm_tr_err   = compute_score(y_train, gmm_pred_train, K)
gmm_test_err = compute_score(y_test, gmm_pred_test, K)
print(f'GMM Training err    {gmm_tr_err:.4f}, Test err {gmm_test_err:.4f}')


from sklearn.cluster import KMeans
kmeans, kmeans_pred_train, kmeans_pred_test = None, None, None
# YOUR CODE HERE

# KMeans Model
kmeans = KMeans(n_clusters=K, random_state=ID_number).fit(x_train)

# KMeans Predictions
kmeans_pred_train = kmeans.predict(x_train)
kmeans_pred_test = kmeans.predict(x_test)

kmeans_tr_err   = compute_score(y_train, kmeans_pred_train, K)
kmeans_test_err = compute_score(y_test, kmeans_pred_test, K)
print(f'Kmeans Training err {kmeans_tr_err:.4f}, Test err {kmeans_test_err:.
↪4f}')


from sklearn.neural_network import MLPClassifier
best_mlp_large = MLPClassifier(hidden_layer_sizes=(50, 50), max_iter=1000,␣
↪alpha=1e-4, solver='sgd', tol=1e-4,
                               random_state=ID_number, learning_rate_init=.1)
best_mlp_large.fit(x_train, y_train)
training_error = 1. - best_mlp_large.score(x_train, y_train)
test_error = 1. - best_mlp_large.score(x_test, y_test)
```

```python
print(f'MLP Training err    {training_error:.4f}, Test err {test_error:.4f}')
```

```
GMM Training err    0.4006, Test err 0.5547
Kmeans Training err 0.3894, Test err 0.4060
MLP Training err    0.0000, Test err 0.0651
```

[29]:
```python
assert gmm_pred_train.shape == (m_t,)
assert gmm_pred_test.shape == (70000 - m_t,)
assert kmeans_pred_train.shape == (m_t,)
assert kmeans_pred_test.shape == (70000 - m_t,)
```

3.1 TODO 11 (max 10 lines)

- What is the effect of a wrong choice of the number of clusters? Briefly describe both log-likelihood as a function of iterations and optimal clustering (depicted on the 2-D plot).
- What does the 10 different plots in TODO 8 represent, with respect to the GMM approach?
- The number of errors using GMM on MNIST is quite high, could have you predicted such a behaviour looking only at the plots in TODO 8? Why?
- Compare GMM, Kmeans and NN. Which is the best model? Why? Did you expect the result?

(Answer in the next cell, no need to add code)

3.2 # YOUR CODE HERE

+ A wrong choice of K could lead to Overfit/Underfit. When increasing K the log plot converges to an higher value (Overfit). When $K > 3$ clusters start to overlap in the 2d-plot leading to more uncertain classifications and (except for $K = 6$) to a larger number of iterations in which the log plot is near to its saturation level.

+ They represent the mean value of the Gaussian Distributions of the GMM, namely the element-wise mean of all the matrices belonging to each cluster (according to the fitted model).

+ Yes. In facts ideally we'd like to have a cluster for each digit, but the '5' is clearly missing and the '1' appears twice. Also note how '4' and '7' are heavily "polluted" by '9', while '3' is polluted (probably) by '5' and '9'.

+ The best model is clearly the MLP (expected, since it trains over more info=true labels). Also: clustering methods are sensitive to the "curse of dimensionality" while the non-linear capabilities of the NN perform well even when $d >>$.