Assignment8

December 4, 2017

1 Unsupervised learning: Mixture of Gaussians

In this assignment, your task will be to first generate some data from a mixture of Gaussians model and subsequently to fit a mixtures of Gaussians model to this data, and recover the original parameters.

- Question 1 This part has been done for you. You only have to read the code in the functions `mixGaussGen` and `sample_discrete` to understand how we generate data from our Mixture of Gaussians.
- Question 2 Fill in the code for function `mixGaussPDF`
- Question 3 Fill in the missing code for function `getMixGaussLogLike`
- Question 4 Fill in the missing code in the EM algorithm.
- Question 5 Fit mixture of Gaussians to data for classification.
  - Question 5a Fit MoG to positive class.
  - Question 5b Fit MoG to negative class.
- Question 6 Calculate the posterior for the positive class using Bayes' rule and compare it to the actual posterior.

1.1 Imports

In [1]:
```
%load_ext autoreload
%autoreload 2

import numpy as np
import matplotlib.pyplot as plt
from construct_data_mod import construct_data, drawGaussianOutline, getGaussian2SD
from scipy.stats import multivariate_normal
from IPython import display

import time
import sys
flt_min = sys.float_info.min


%matplotlib inline
plt.rcParams['figure.figsize'] = (5.0, 4.0) # set default size of plots
```

```
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'
```

1.1.1 Define parameters for a mixture of k Gaussians (MoG)

Here we are going to define the true parameters for a mixture of 3 Gaussians. We are representing the mixture of Gaussians as a numpy dictionary. In d dimenisions, the mean field will be a $d \times k$ matrix and the cov field will be a $d \times d \times k$ matrix. The weight field corresponds to the weight of the mixture component i, π_i

```
In [2]: mixGaussTrue = dict()
        mixGaussTrue['k'] = 3
        mixGaussTrue['d'] = 2
        mixGaussTrue['weight'] = np.array([0.1309, 0.3966, 0.4725])
        mixGaussTrue['mean'] = np.array([[ 4.0491 , 4.8597],[ 7.7578 , 1.6335],[ 11.9945, 8.9206
        mixGaussTrue['cov'] = np.reshape([0.5, 0.25], newshape=(1,1,2))
        mixGaussTrue['cov'] = np.zeros(shape=(mixGaussTrue['d'],mixGaussTrue['d'],mixGaussTrue['
        mixGaussTrue['cov'][:,:,0] = np.array([[  4.2534, 0.4791], [0.4791, 0.3522]])
        mixGaussTrue['cov'][:,:,1] = np.array([[  0.9729, 0.8723],[  0.8723,  2.6317]])
        mixGaussTrue['cov'][:,:,2] = np.array([[  0.9886, -1.2244],[ -1.2244, 3.0187]])
```

Generate data from MoG

Function `mixGaussGen` generates data by randomly sampling a mixture of gaussians. In order to sample the data: 1. We need to pick one of k components by sampling the discrete distribution formed by the MoG's weights. 2. Using the mean and covariance corresponding to the k-th component we sample a new daatapoint from a multivariate Gaussian distribution.

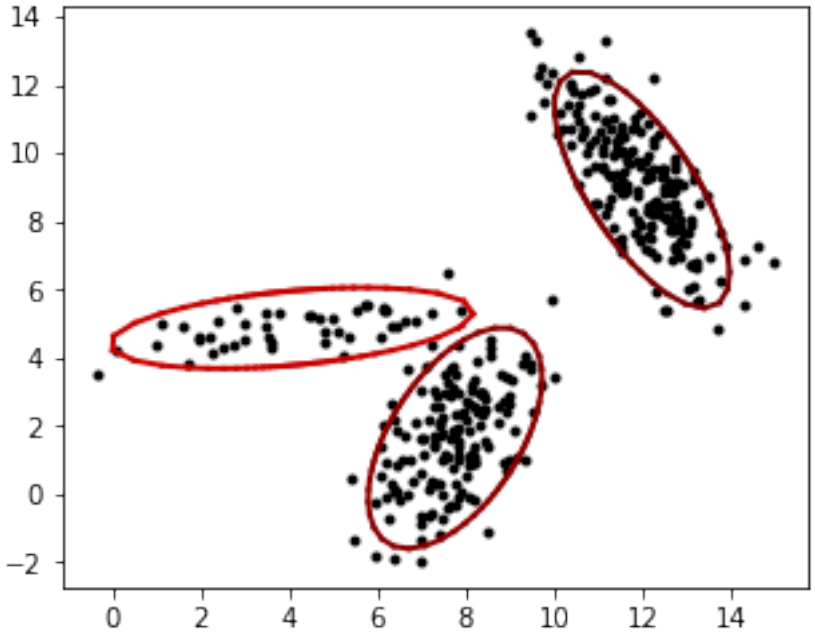
Question 1: Fill in the missing function from function `mixGaussGen`. The `sample_discrete` which randomly selects a component is given to you. Hint: use function `np.random.multivariate_normal` to randomly sample a multivariate Gaussian distribution.

```
In [3]: # define number of samples to generate
        nData = 400;

In [4]: # draws a random sample from a discrete probability distribution
        def sample_discrete(discrete_distribution):
            bins = np.cumsum(discrete_distribution)
            rand_value = np.random.rand()
            for i in range(len(bins)):
                if rand_value<=bins[i]:
                    return i

In [5]: # this function generates data from a k-dimensional
        # mixture of Gaussians structure.
        def mixGaussGen(mixGauss, nData):
            # create space for output data

            ############ TO DO QUESTION 1 #################3

            data = np.zeros(shape=(mixGauss['d'], nData))
```

```
        # for each data point
        for cData in range(nData):
            # randomly choose Gaussian according to probability distributions
            h = sample_discrete(mixGauss['weight'])
            # draw a sample from the appropriate Gaussian distribution
            # using function np.random.multivariate_normal with the correct mean and covaria
            curMean = mixGauss['mean'][:,h]
            curCov = mixGauss['cov'][:, :, h]
            data[:, cData] = np.random.multivariate_normal(curMean, curCov)

        ############ END - TO DO QUESTION 1 ##################3


        return data
```

In [6]:
```
# this routine draws the generated data and plots the MoG model on top of it
def drawEMData2d(data,mixGauss, title_text=""):
    x_min = np.min(data[0,:])
    x_max = np.max(data[0,:])
    y_min = np.min(data[1,:])
    y_max = np.max(data[1,:])
    axes = plt.gca()
    axes.set_xlim([x_min-0.5,x_max+0.5])
    axes.set_ylim([y_min-0.5,y_max+0.5])
    plt.cla()
    plt.plot(data[0,:],data[1,:],'k.')
    plt.title(title_text)
    for cGauss in range(mixGauss['k']):
        drawGaussianOutline(mixGauss['mean'][:,cGauss],mixGauss['cov'][:,:,cGauss],mixGa
    return
```

In [7]:
```
#generate data from the mixture of Gaussians
#TO DO - fill in this routine (above)
data = mixGaussGen(mixGaussTrue,nData)

#draw data, true Gaussians
drawEMData2d(data,mixGaussTrue)
```



2 Calculate probability density of Mixture of gaussians

Question 2: Fill in the missing code for function `mixGaussPDF`. This function should give the result of the function:

$$p(\vec{x}) = \sum_{k=1}^{K} \pi_k * N(\vec{x}|\mu_k, \Sigma_k)$$

It should be able to handle multiple datapoints at once. The input should have dimensions $2 \times N$ and the output $1 \times N$. Hint: use the function `multivariate_normal.pdf` from scipy.stats to get the probability density of a multivariate normal distribution.

```
In [8]: def mixGaussPDF(data, mixGaussEst):
            data = np.atleast_2d(data)
            # find total number of data items
            nDims, nData = data.shape
            if nDims!=2:
                print('Error! Wrong number of dimensions for data!')

            probDensity = 0

            ############ TO DO QUESTION 2 ################3


            for curC in range(mixGaussEst['k']):
                curWeight = """??????"""
```

```
            curMean = """??????"""
            curCov = """??????"""
            probDensity = probDensity + curWeight*multivariate_normal.pdf(data.T, curMean,cu
        ############ END - TO DO QUESTION 2 ##################3
        return probDensity
```

Calculate log likelihood of mixture of gaussians

Question 3: Fill in the missing code for function `getMixGaussLogLikelihood`. This function should return the log-likelihood for a set of points:

$$l(\theta; x) = \sum_{i=1}^{N} log(\sum_{k=1}^{K} \pi_k \mathcal{N}(\vec{x}^i | \mu_k, \Sigma_k))$$

The input should have dimensions 2ŒN and the output is a single number. Hint: use the function multivariate_normal.pdf from scipy.stats to get the probability density of a multivariate normal distribution.

```
In [9]: def getMixGaussLogLikelihood(data, mixGaussEst):
            data = np.atleast_2d(data)
            # find total number of data items
            nDims, nData = data.shape
            # initialize log likelihoods
            logLike = 0;

            # run through each data item
            for cData in range(nData):
                thisData = data[:, cData]
                ############ TO DO QUESTION 3 #################

                # TO DO - calculate likelihood of this data point under mixture of
                # Gaussians model. Replace this
                like = 0;
                for curC in range(mixGaussEst['k']):
                    curWeight = """??????"""
                    curMean = """??????"""
                    curCov = """??????"""
                    like = """??????"""

                # add to total log like
                logLike = """??????"""
                ############ END - TO DO QUESTION 3 #################

            return  np.asscalar(logLike)
```

Fit Mixture of Gaussians model to data

This is the main part of our EM algorithm. Within this algorithm we iterate between the following two steps: * Expectation Step: in this step, we calculate a complete posterior distribution over each of the hidden variables (for each datapoint, we have a hidden variable assigning it to

one of the mixtures) * Maximization Step: in this step we update the parameters of the Gaussians (mean, cov, w) using the posterior distributions calculated during the expectation step.

The pdf `MoGCribSheet.pdf` is given to you to help you with this part of the assignment. Question 4: Fill in the missing code in the EM algorithm. Follow the instructions given in the comments as well as the forementioned pdf.

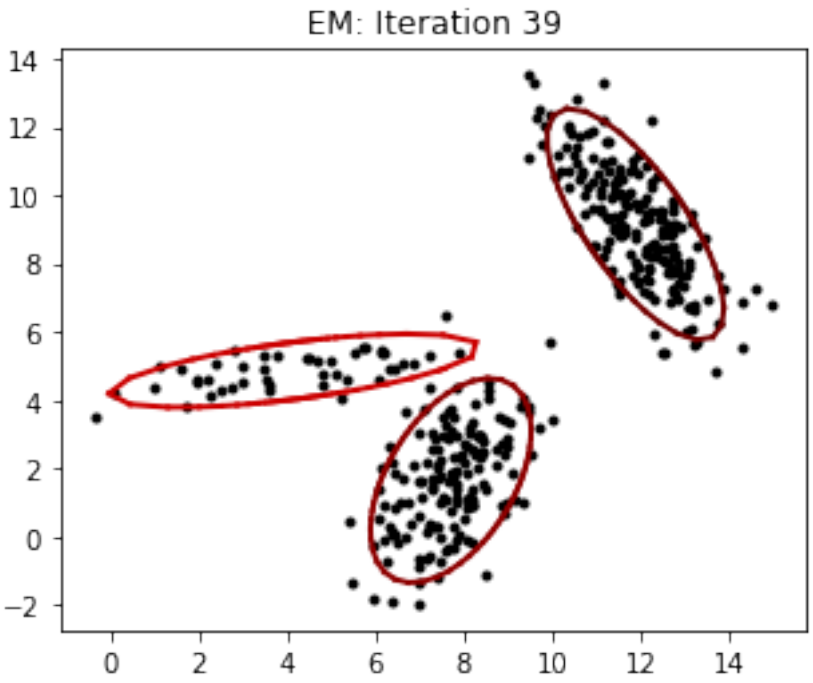
```
In [10]: def fitMixGauss(data, k, nIter = 20):
             nDims, nData = data.shape

             #     MAIN E-M ROUTINE
             #     there are nData data points, and there is a hidden variable associated
             #     with each.  If the hidden variable is 0 this indicates that the data was
             #     generated by the first Gaussian.  If the hidden variable is 1 then this
             #     indicates that the hidden variable was generated by the second Gaussian
             #     etc.

             postHidden = np.zeros(shape=(k, nData))

             #     in the E-M algorithm, we calculate a complete posterior distribution over eac
             #     the (nData) hidden variables in the E-Step.  In the M-Step, we
             #     update the parameters of the Gaussians (mean, cov, w).

             # we will initialize the values to random values
             mixGaussEst = dict()
             mixGaussEst['d'] = nDims
             mixGaussEst['k'] = k
             mixGaussEst['weight'] = (1 / k) * np.ones(shape=(k))

             # randomly initialize means and covariances
             # mixGaussEst['mean'] = 2 * np.random.randn(nDims, k)
             # mixGaussEst['cov'] = np.zeros(shape=(nDims, nDims, k))
             # for cGauss in range(k):
             #     mixGaussEst['cov'][:, :, cGauss] = 2.5 + 1.5 * np.random.uniform() * np.eye(n

             # initialize means and covariances using data statistics
             mean_data = np.mean(data,axis=1)
             mixGaussEst['mean'] = (1+0.1*np.random.normal(size=(2,3)))*np.expand_dims(mean_data
             mixGaussEst['cov'] = np.zeros(shape=(nDims, nDims, k))
             for cGauss in range(k):
                 cov_data = np.cov(data)
                 mixGaussEst['cov'][:, :, cGauss] = cov_data*(1 +  0.1*np.random.normal())

             ############ TO DO QUESTION 4 #################
             # calculate current likelihood
             # TO DO - fill in this routine
             logLikelihoods_list = []
             logLikehood = getMixGaussLogLikelihood(data, mixGaussEst)
             #print('Log Likelihood Iter 0 : {:4.3f}\n'.format(logLike))
```

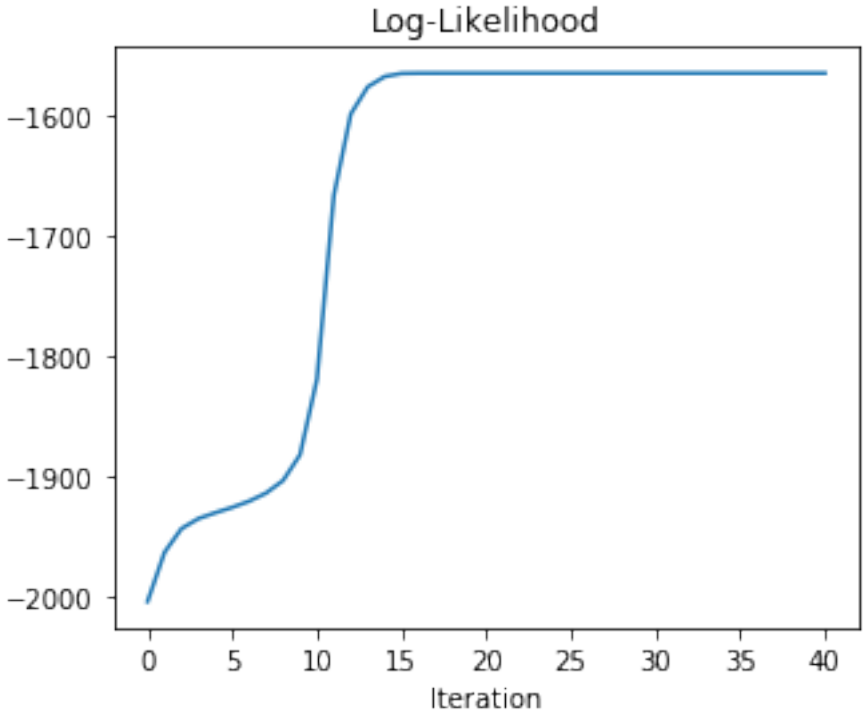
```
logLikelihoods_list.append(logLikehood)

fig, ax = plt.subplots(1, 1)

for cIter in range(nIter):

    # ====================== ======================
    # Expectation step (4a)
    # ====================== ======================

    for cData in range(nData):
        ############ TO DO QUESTION 4a #################
        # fill in column of 'hidden' - calculate posterior probability that
        # this data point came from each of the Gaussians
        # replace this:

        thisData = data[:, cData]
        for curC in range(  """??????"""       ):
            curWeight = """??????"""
            curMean = """??????"""
            curCov = """??????"""
            postHidden[curC, cData] = """??????"""

        ## normalize the posterior probabilities
        postHidden[:, cData] = postHidden[:, cData] / (np.sum(postHidden[:, cData])



    # ====================== ======================
    # Maximization Step (4b)
    # ====================== ======================
    ############ TO DO QUESTION 4b #################

    # for each constituent Gaussian
    for cGauss in range(k):
        # TO DO (4c) Update weighting parameters mixGauss.weight based on the total
        # posterior probability associated with each Gaussian. Replace this:

        thisResp = postHidden[cGauss, :]

        mixGaussEst['weight'][cGauss] = """??????"""

        # TO DO (4d):  Update mean parameters mixGauss.mean by weighted average
        # where weights are given by posterior probability associated with
        # Gaussian.  Replace this:

        mixGaussEst['mean'][:, cGauss] = """??????"""
```

```
            # TO DO (4e):  Update covariance parameter based on weighted average of
            # square distance from update mean, where weights are given by
            # posterior probability associated with Gaussian

            mixGaussEst['cov'][:, :, cGauss] = """??????"""


        ############ END - TO DO QUESTION 4 #################
        # draw the new solution
        title_text = "EM: Iteration %d"%(cIter)
        drawEMData2d(data, mixGaussEst, title_text )
        display.display(plt.gcf())
        display.clear_output(wait=True)
        time.sleep(0.3)

        # calculate the log likelihood
        logLikehood = getMixGaussLogLikelihood(data, mixGaussEst)
        logLikelihoods_list.append(logLikehood)
        #print('Log Likelihood After Iter {} : {:4.3f}\n'.format(cIter, logLike))

    # Plot the likelihoods after each iteration
    fig = plt.figure()
    plt.plot(logLikelihoods_list)
    plt.xlabel('Iteration')
    plt.title('Log-Likelihood')

    return mixGaussEst
```

Now we use the completed function `fitMixGauss` to fit our data.

In [11]:
```
#define number of components to estimate
nGaussEst = 3

#fit mixture of Gaussians (Pretend someone handed you some data. Now what?)
#TO DO fill in this routine (below)
mixGaussEst = fitMixGauss(data,nGaussEst,nIter=40);
```



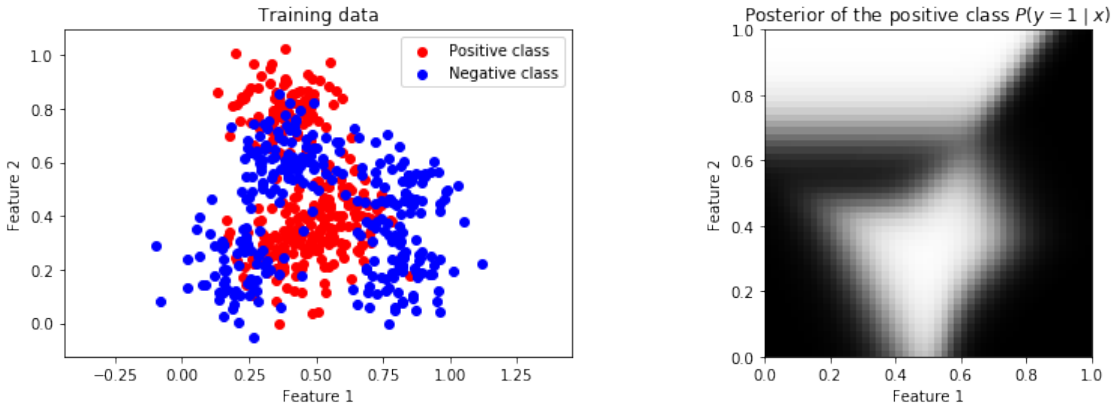


Use mixture of Gaussians for classification

We will now use the dataset we used in previous assignments for classification. This dataset is actually generated using 2 mixtures of Gaussians with 3 and 4 components for the positive and negative classes respectively.

Question 5: Use function `fitMixGauss` to get an estimate on the parameters of these two mixtures of gaussians.

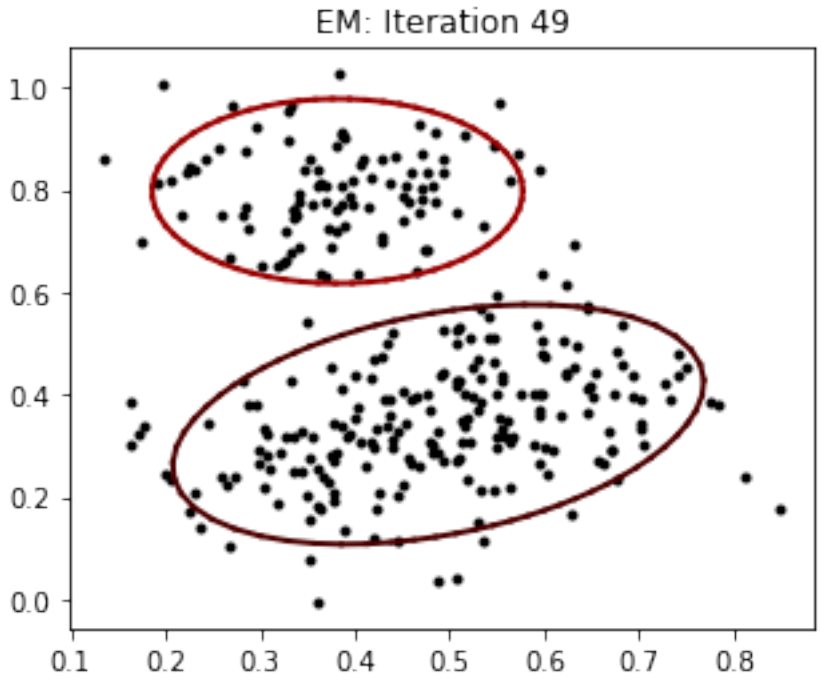
In [12]:
```
training_features, training_labels, posterior = construct_data(600, 'train', 'nonlinear

# Extract features for both classes
features_pos = training_features[training_labels == 1].T
features_neg = training_features[training_labels != 1].T

# Display data
fig = plt.figure(figsize=plt.figaspect(0.3))
ax = fig.add_subplot(1, 2, 1)
ax.scatter(features_pos[0,:], features_pos[1,:], c="red", label="Positive class")
ax.scatter(features_neg[0,:], features_neg[1,:], c="blue", label="Negative class")
ax.axis('equal')
ax.set_title("Training data")
ax.set_xlabel("Feature 1")
ax.set_ylabel("Feature 2")
ax.legend()

ax = fig.add_subplot(1, 2, 2)
ax.imshow(posterior, extent=[0, 1, 0, 1], origin='lower')
ax.set_title("Posterior of the positive class $P(y=1 \mid x)$")
ax.set_xlabel("Feature 1")
ax.set_ylabel("Feature 2")
plt.show()
```

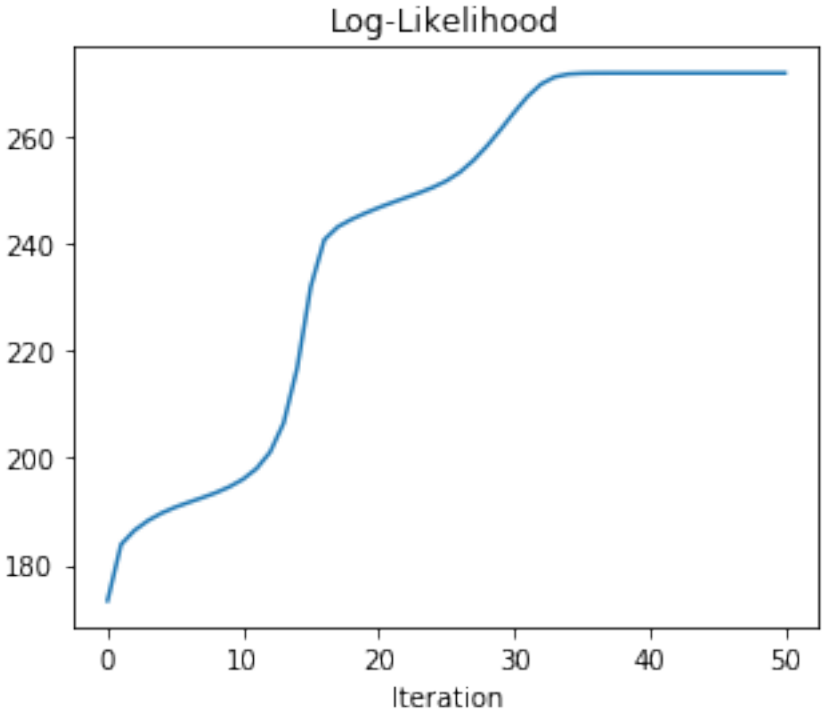


Fit mixture of Gaussians with 2 components to the positive class.

In [13]:
```
#define number of components to estimate
numGaussPositive_Est = 2
```

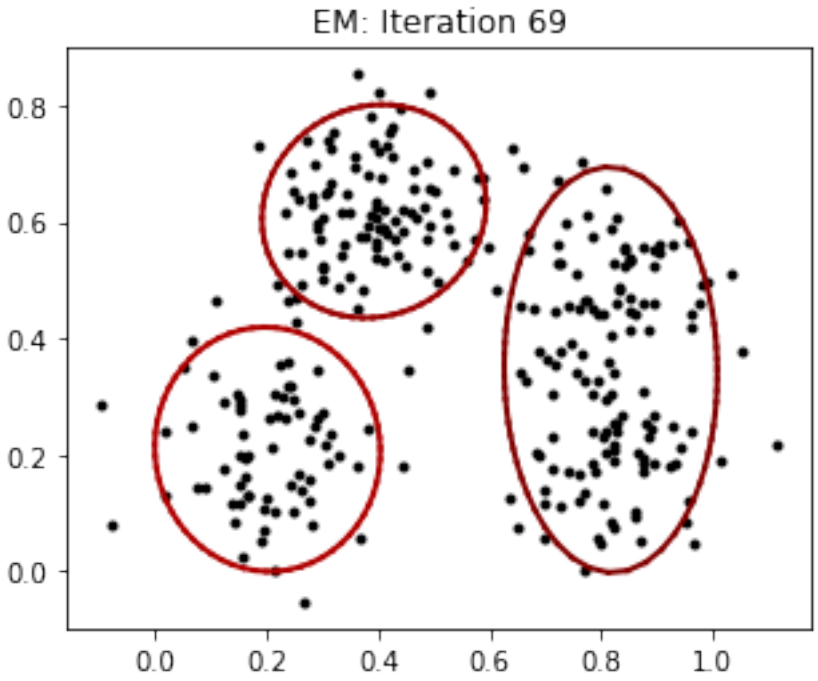
```
############ TO DO QUESTION 5a #################
# fill in the correct arguments
mixGaussPositive_Est = fitMixGauss("""??????""","""??????""", nIter = 50)
```

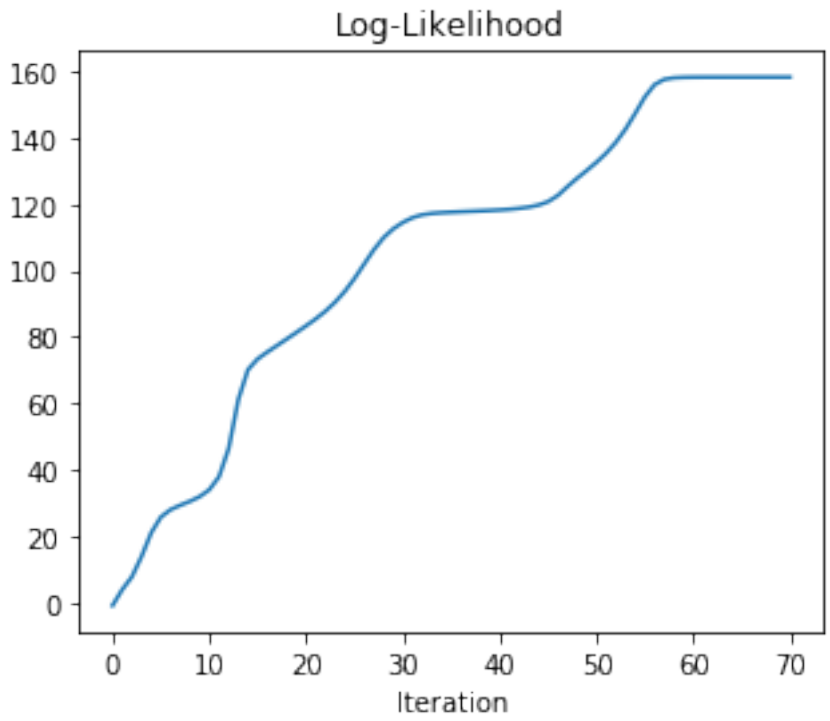




Fit mixture of Gaussians with 2 components to the negative class.

In [14]:
```
#define number of components to estimate
numGaussNegative_Est = 3

############ TO DO QUESTION 5b #################
# fill in the correct arguments
mixGaussNegative_Est = fitMixGauss("""??????""","""??????""", nIter = 70)
```





Calculate the posterior for the positive class. For this part of the assignment you need to use the two class conditional distributions for the positive and the negative class (the mixture of gaussians you've just estimated), the priors for each class and Bayes' rule to calculate the posterior distribution for the positive class. You're also going to use function `mixGaussPDF` here.

Question 6: Calculate the posterior for the positive class using Bayes' rule and compare it to the actual posterior.

```
In [15]: x_range = np.linspace(0, 1, 50)
         y_range = np.linspace(0, 1, 50)
         grid_x, grid_y = np.meshgrid(x_range, y_range)
         xy_array = np.row_stack([grid_x.flat, grid_y.flat])

         ############ TO DO QUESTION 6 #################
         # calculate class conditional probabilities for positive and negative class

         pos_class_on_grid = """??????"""
         neg_class_on_grid = """??????"""

         # calculate prior probabilities for positive and negative class
         prior_pos = """??????"""
         prior_neg = """??????"""

         # calculate posterior probabilities for positive class using Bayes' rule
         posterior_positive = """??????"""
```

```
############ END - TO DO QUESTION 6 #################


# reshape posterior probability to plot it as an image
posterior_positive = posterior_positive.reshape(grid_x.shape)
```

In [16]:
```
fig = plt.figure(figsize=plt.figaspect(0.3))
ax = fig.add_subplot(1, 2, 1)
ax.imshow(posterior_positive, extent=[0, 1, 0, 1], origin='lower')
ax.set_title("Estimated posterior of the class $P(y=1 \mid x)$")
ax.set_xlabel("Feature 1")
ax.set_ylabel("Feature 2")

ax = fig.add_subplot(1, 2, 2)
ax.imshow(posterior, extent=[0, 1, 0, 1], origin='lower')
ax.set_title("Posterior of the positive class $P(y=1 \mid x)$")
ax.set_xlabel("Feature 1")
ax.set_ylabel("Feature 2")
plt.show()
```