Assignment 8 - Unsupervised learning: Mixture of Gaussians

In this assignment, your tasks will be: (i) to generate some data from a mixture of Gaussians (MoG) model, and (ii) subsequently to fit a MoG model to the generated data, in order to recover the original parameters.

- Question 1 This part has been done for you. You only have to read the code in `mixGaussGen` to understand how we generate data from our Mixture of Gaussians model. (0%)
- Question 2 Fill in the missing code for the function `mixGaussPDF` (15%)
- Question 3 Fill in the missing code for the function `getMixGaussLogLike` (10%)
- Question 4 Fill in the missing code in the EM algorithm. (40%)
- Question 5 Fit a mixture of Gaussians to the data for classification.
  - Question 5a Fit a MoG to the positive class. (10%)
  - Question 5b Fit a MoG to the negative class. (10%)
- Question 6 Calculate the posterior for the positive class using Bayes' rule and compare it to the actual posterior. (15%)

Imports

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

import sys
import time

import numpy as np
from IPython import display
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal

from construct_data_mod import construct_data, drawGaussian
Outline

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'
```

Define parameters for a mixture of k Gaussians (MoG)

Here we define the true parameters for a mixture of $K = 3$ Gaussians. We are representing the mixture of Gaussians as a dictionary. In d dimensions, the 'mean' field is a $d \times K$ matrix and the 'cov' field is a $d \times d \times K$ matrix. The 'weight' field is a list of weights π_i for each mixture component 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]]).T
mixGaussTrue['cov'] = np.zeros(shape=(mixGaussTrue['d'], mi
xGaussTrue['d'], mixGaussTrue['K']))
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 the MoG

The function `mixGaussGen` generates data points by randomly sampling a mixture of Gaussians. In order to sample the data:

1. We need to pick one of the K components by sampling the discrete distribution formed by the MoG's weights.
2. Using the mean and covariance corresponding to the selected component, we sample a new point from a multivariate Gaussian distribution.

Question 1: This part has been done for you. You only have to read the code in mixGaussGen to understand how we generate data points from our Mixture of Gaussians model. (0%)

Hint: numpy provides `np.random.choice` to randomly select a value from a distribution given the weights (the 'p' parameter), and `np.random.multivariate_normal` to randomly sample a multivariate Gaussian distribution.

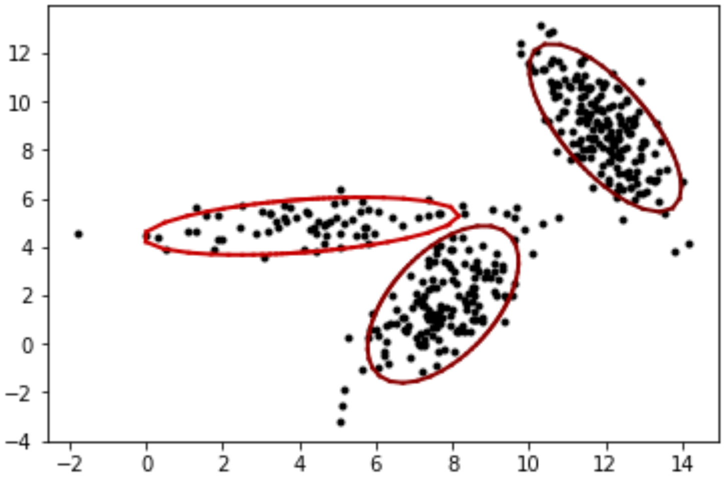
In [3]:

```
# define the number of samples to generate
nData = 400

# this function generates data from a k-dimensional
# mixture of Gaussians structure.
def mixGaussGen(mixGauss, nData):
    np.random.seed(123)

    ############ TO DO QUESTION 1 #################
    # allocate space for output data
    data = np.zeros(shape=(mixGauss['d'], nData))
    # for each data point
    for cData in range(nData):
        # randomly choose a Gaussian component according to
the probability distribution
        h = np.random.choice(mixGauss['K'], p=mixGauss['wei
ght'])
        # draw a sample from the sampled Gaussian distribut
ion
        # using the function np.random.multivariate_normal
with the correct mean and covariance
        curMean = mixGauss['mean'][:,h]
        curCov = mixGauss['cov'][:,:,h]
        data[:, cData] = np.random.multivariate_normal(curM
ean, curCov)

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

    return data
```

In [4]:

```
# this routine draws the generated data and plots the MoG m
odel on top of it
def drawEMData2d(ax, data, mixGauss, title_text=""):
    ax.plot(data[0,:], data[1,:], 'k.')
    ax.set_title(title_text)
    for cGauss in range(mixGauss['K']):
        drawGaussianOutline(ax,
                            mixGauss['mean'][:,cGauss],
                            mixGauss['cov'][:,:,cGauss],
                            mixGauss['weight'][cGauss])
    return
```

In [5]:
```
# generate data points from the mixture of Gaussians
data = mixGaussGen(mixGaussTrue,nData)

# draw data, MOG distributions
fig, ax = plt.subplots()
drawEMData2d(ax, data, mixGaussTrue)
```



Calculate probability density of Mixture of gaussians

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

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

It should be able to handle multiple data points 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` (imported in the first cell) to get the probability density function of a multivariate normal distribution.

In [6]:
```
def mixGaussPDF(data, mixGaussEst):
    data = np.atleast_2d(data)
    # find the total number of data items
    nDims, nData = data.shape
    if nDims != mixGaussEst['d']:
        print('Error! Wrong number of dimensions for dat
a!')

    K = mixGaussEst['K']
    weight = mixGaussEst['weight']
    mean = mixGaussEst['mean']
    cov = mixGaussEst['cov']

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

    ############ END - TO DO QUESTION 2 #################
    return probDensity
```

Calculate the log likelihood of the Mixture of Gaussians

Question 3: Fill in the missing code for the function `getMixGaussLogLikelihood` . This function should return the log-likelihood of $\theta = (\{\mu_k\}, \{\Sigma_k\}, \{\pi_k\})$ given a set of points $\mathbf{x}$:

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

The input should have dimensions $2 \times N$ and the output is a single number. Hint: call the function `mixGaussPDF` defined above.

In [7]:
```
def getMixGaussLogLikelihood(data, mixGaussEst):
    data = np.atleast_2d(data)
    ############ TO DO QUESTION 3 #################

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

    return np.asscalar(logLike)
```

Fit the Mixture of Gaussians model to the 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 on 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, weight) by maximising the posterior distributions calculated during the expectation step.

The file "MoGCribSheet.pdf" is given to you to help you with this part of the assignment.

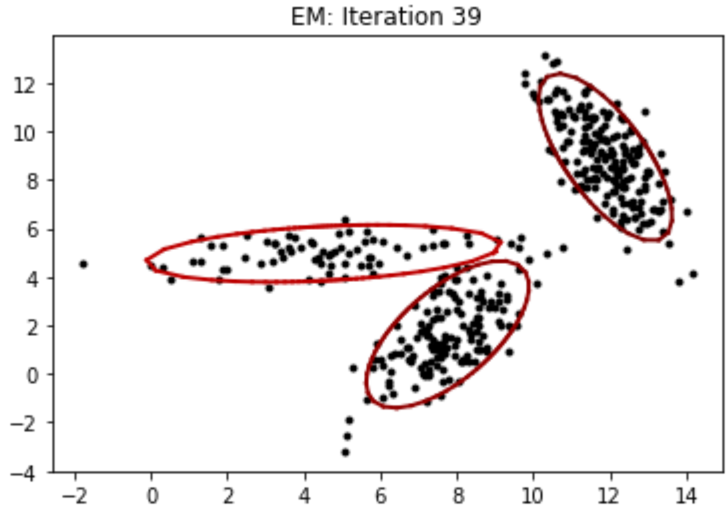
Question 4: Fill in the missing code in the EM algorithm. Follow the instructions given in the comments as well as the forementioned pdf. Then, write a short comment to explain how the MoG has been initialized for the EM algorithm.

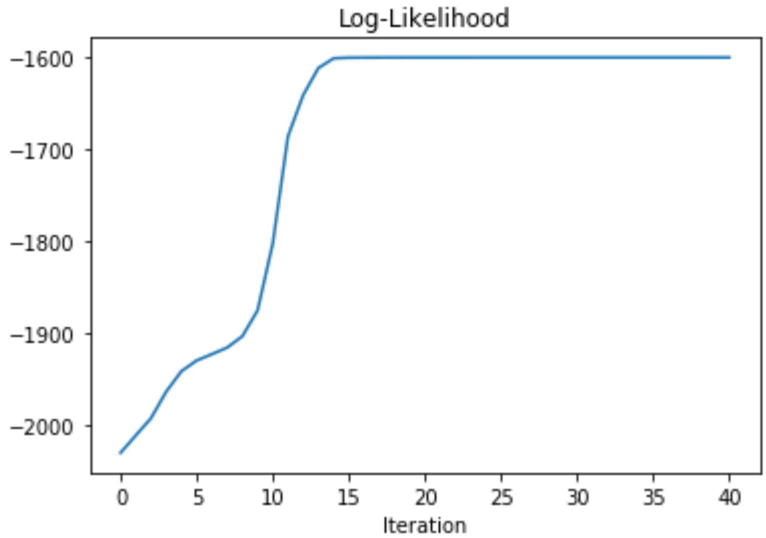
In [8]:
```
def fitMixGauss(data, K, nIter=20):
    nDims, nData = data.shape

    #     MAIN E-M ROUTINE
    #     there are nData data points, and there is a hidde
n variable associated
    #     with each.  If the hidden variable is 0 this indi
cates that the data was
    #     generated by the first Gaussian.  If the hidden v
ariable 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 pos
terior distribution over each of
    #     the (nData) hidden variables in the E-Step.  In t
he M-Step, we
    #     update the parameters of the Gaussians (mean, co
v, w).

    ############ TO DO QUESTION 4 #################
    # initialize parameters
    mixGaussEst = dict()
    mixGaussEst['d'] = nDims
    mixGaussEst['K'] = K
    mixGaussEst['weight'] = np.full(shape=K, fill_value=1/
K)
    # initialize means and covariances using data statistic
s
    mean_data = np.mean(data, axis=1).reshape(nDims, 1)
    mixGaussEst['mean'] = (1 + 0.1*np.random.normal(size=(n
Dims, K))) * mean_data
    mixGaussEst['cov'] = np.zeros(shape=(nDims, nDims, K))
    cov_data = np.cov(data)
    for k in range(K):
        mixGaussEst['cov'][:, :, k] = cov_data * (1 +  0.1*
np.random.normal())
    ############ END - TO DO QUESTION 4 #################

    ############ TO DO QUESTION 4 #################
    # calculate current likelihood
    # TO DO - fill in this routine
    logLikelihoodsList = []
    logLikehood = getMixGaussLogLikelihood(data, mixGaussEs
t)
    #print('Log Likelihood Iter 0 : {:4.3f}\n'.format(logLi
ke))
    logLikelihoodsList.append(logLikehood)

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

    for cIter in range(nIter):
```

In [ ]:
```
############ TO DO QUESTION 4 #################
# Insert your short comment on the MoG initialization here

############ END - TO DO QUESTION 4 #################
```

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

In [9]:
```
# define the number of components to estimate
nGaussEst = 3

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



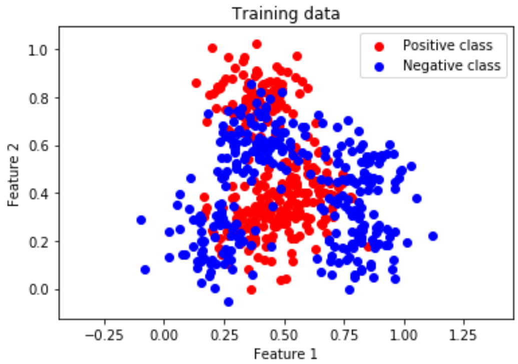


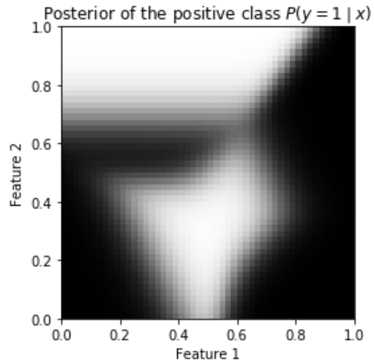
Use the Mixture of Gaussians for classification

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.

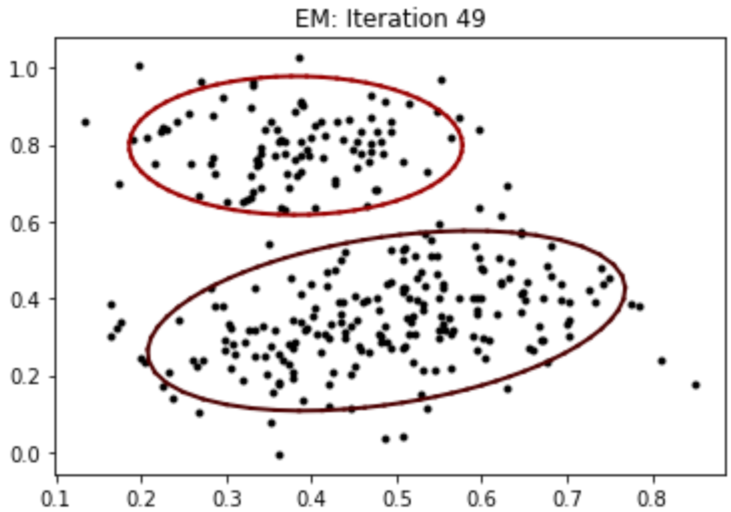
In [10]:
```
training_features, training_labels, posterior = construct_d
ata(600, 'train', 'nonlinear' , plusminus=False)

# 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", l
abel="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()
```

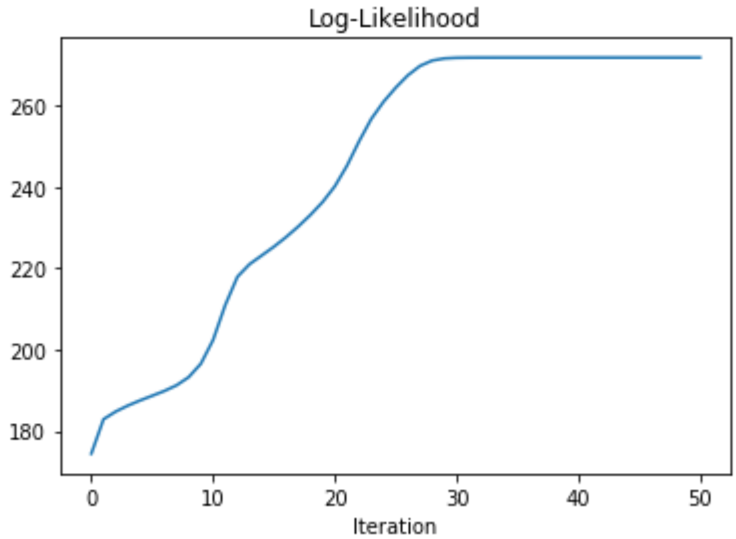




Fit a Mixture of Gaussians with 2 components to the positive class.

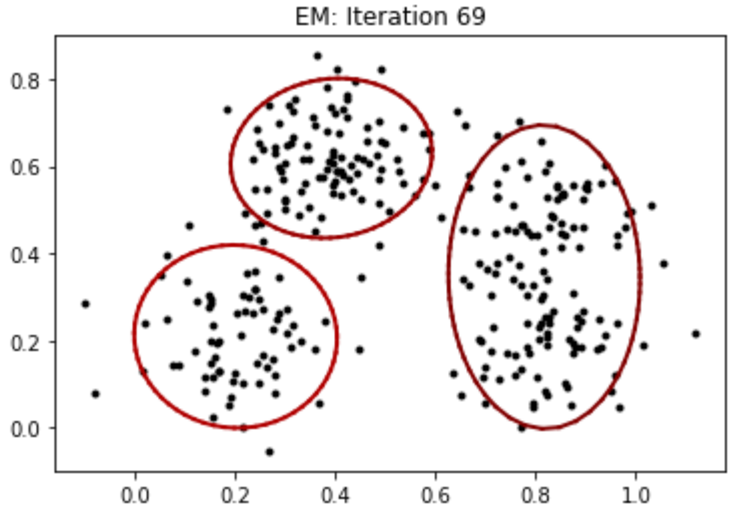
In [11]:
```
# define the number of components to estimate
numGaussPositiveEst = 2

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

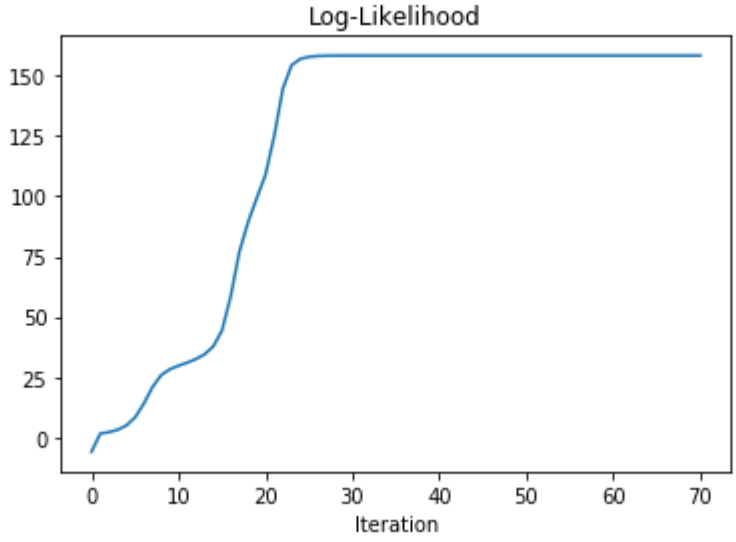




Fit a Mixture of Gaussians with 3 components to the negative class.

In [12]:
```
# define the number of components to estimate
numGaussNegativeEst = 3

############ TO DO QUESTION 5b #################
# fill in the correct arguments
mixGaussNegativeEst = fitMixGauss("""???""", """???""", nIt
er=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 are expected to use the function `mixGaussPDF` here.

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

In [13]:

```
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])

# Prior probabilities for positive and negative class
prior_pos = 0.5
prior_neg = 0.5

############ TO DO QUESTION 6 #################
# calculate class conditional probabilities for positive an
d negative class
pos_class_on_grid = """???"""
neg_class_on_grid = """???"""

# calculate posterior probabilities for positive class usin
g 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.shap
e)
```

In [14]:

```
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()
```