Intruduction to Machine Learning

Dr S.Amini

department: Electrical Engineering

Amirreza Velae	400102222
Mohammadreza Hajibabaie	99101416
github	repository

Project

April 23, 2023

Intruduction to Machine Learning

Project

Amirreza Velae 400102222
Mohammadreza Hajibabaie 99101416
github repository

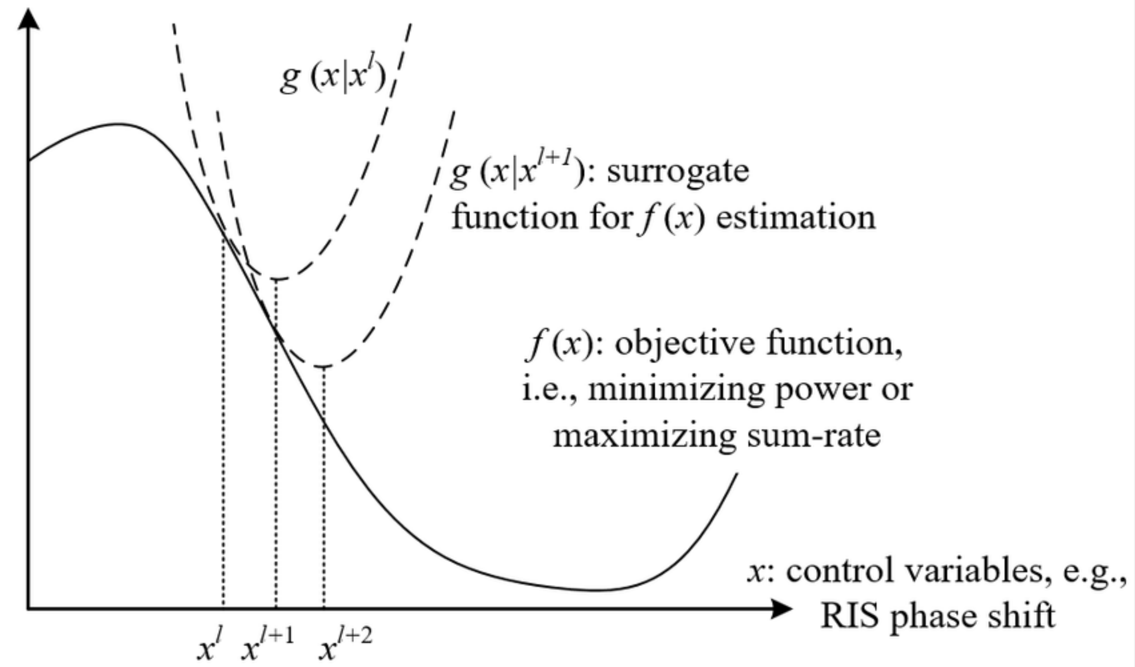
Theory Questions

Theory Question 1.

In your own words, explain how the MM algorithm can deal with non-convex optimization objective functions by considering simpler convex objective functions.

solution

MM algorithm is an iterative algorithm for optimizing a nonconvex or nonconcave $l(\theta)$ that has a complex form. For example if our goal is finding maximum of a function we can consider a simpler concave function $Q(\theta, \theta^{(t)})$ that depends on a certain parameter. Function must be tight lowerbound that means in a certain point $l(\theta^{(t)} = Q(\theta^{(t)}, \theta^{(t)}))$ and also $l(\theta \leq Q(\theta, \theta^{(t)}))$. In each iteration we choose the max of $Q(\theta, \theta^{(t)})$ as next level parameter $\theta^{(t+1)} = argmax_\theta Q(\theta, \theta^{(t)})$ and we move on to reach the maximum of the main function $l(\theta)$ and according to the following inequality we act correctly:
$l(\theta^{(t+1)}) \leq Q(\theta^{(t+1)}, \theta^{(t)}) \leq Q(\theta^{(t)}, \theta^{(t)}) = l(\theta^{(t)})$ also we can do same for finding minimum but with convex function.It means we can choose a function that $l(\theta) \leq Q(\theta, \theta^t)$ and we move on curve to find the minimum.



$g(x|x^l)$
$g(x|x^{l+1})$: surrogate function for $f(x)$ estimation
$f(x)$: objective function, i.e., minimizing power or maximizing sum-rate
x: control variables, e.g., RIS phase shift
x^l
x^{l+1}
x^{l+2}

Theory Question 2.

Briefly explain how the formula for mixture models:

$$p(y_n; \theta) = \sum_{k=1}^{K} p_Z(z_k; \theta) p_{Y|Z}(y|Z = z_k; \theta) \quad (1)$$

is the same as the sum over all possible values of $Z^{(i)}$ in equation

$$ln(p_Y(y_n;\theta)) = \sum_{i=1}^{N} ln(p(y_n^{(i)};\theta)) = \sum_{i=1}^{N} ln(\sum_{k=1}^{K} p_{Y,Z}(y_n^{(i)}, z_n^{(i)} = k;\theta)) \tag{2}$$

Explain why it's easier to optimize $p_{Y,Z}(y_n^i, z_n^i = K;\theta)$ than $p(y_n;\theta)$ in the context of mixture models.

solution

The model is simplified by Bayes' rule:
$p(y,z) = p(y|z)p(z)$
$p(y_n;\theta) = \sum_{z_n} p_{Y,Z}(y_n, z_n;\theta) = \sum_{k=1}^{K} p_Z(z_k;\theta)p_{Y|Z}(y|Z = z_k;\theta)$
It is easier to optimize $p_{Y,Z}(yn, zn;)$ because in this case we know which distribution each data belongs to so optimizing become easier because we can optimize each distribution separately and find the parameters of each one.
But it is hard to optimize if we just know that a data comes from sumation of some distribution and don't know anything about each distribution because we can not fit a single distribution to model.

Theory Question 3.

Read about variational inference (or variational bayesian methods) and compare it with the procedure we used for the EM algorithm (You might want to check Wikipedia for this!)

solution

Variational Bayes (VB) is often compared with expectation maximization (EM). The actual numerical procedure is quite similar, in that both are alternating iterative procedures that successively converge on optimum parameter values. The initial steps to derive the respective procedures are also vaguely similar, both starting out with formulas for probability densities and both involving significant amounts of mathematical manipulations. However, there are a number of differences. Most important is what is being computed. EM computes point estimates of posterior distribution of those random variables that can be categorized as "parameters", but only estimates of the actual posterior distributions of the latent variables (at least in "soft EM", and often only when the latent variables are discrete). The point estimates computed are the modes of these parameters; no other information is available. VB, on the other hand, computes estimates of the actual posterior distribution of all variables, both parameters and latent variables. When point estimates need to be derived, generally the mean is used rather than the mode, as is normal in Bayesian inference. Concomitant with this, the parameters computed in VB do not have the same significance as those in EM. EM computes optimum values of the parameters of the Bayes network itself. VB computes optimum values of the parameters of the distributions used to approximate the parameters and latent variables of the Bayes network. For example, a typical Gaussian mixture model will have parameters for the mean and variance of each of the mixture components. EM would directly estimate optimum values for these parameters.

VB, however, would first fit a distribution to these parameters — typically in the form of a prior distribution, e.g. a normal-scaled inverse gamma distribution — and would then compute values for the parameters of this prior distribution, i.e. essentially hyperparameters. In this case, VB would compute optimum estimates of the four parameters of the normal-scaled inverse gamma distribution that describes the joint distribution of the mean and variance of the component.

Theory Question 4.

Compute estimate of parameters for Gaussian Mixture Models for N observed data $\{x_i\}_{i=1}^{N}$

- Determine model parameters and initialize them.
- Compute complete dataset likelihood.
- Find closed-form solution for parameters using EM algorithm.

solution

We want to find the parameter of latent distributions :

$$\pi = \begin{bmatrix} \pi_1 \\ \pi_2 \\ \vdots \\ \pi_K \end{bmatrix} \mu = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_K \end{bmatrix} \sigma^2 = \begin{bmatrix} \sigma_1^2 \\ \sigma_2^2 \\ \vdots \\ \sigma_K^2 \end{bmatrix}$$

According to project description:

$$P_X(x|\theta) = \prod_{i=1}^{n} \sum_{k=1}^{K} \pi_k p_{X_k}(x_i|\theta_k)$$

$$q_n(z_n) = p(Z = k|\theta_k) = Cat(z|\pi_k) = \pi_k$$

$$l(\theta) \geq \sum_n \sum_{z_n} q_n(z_n) ln(\frac{p_{Y,Z}(y_n, z_n|\theta)}{q_n(z_n)}) = \sum_n E_{q_n}[ln(p_{Y,Z}(y_n, z_n|\theta))] + H(q_n)$$

$H(q_n)$does not depends on θ so we don't consider it for maximizing:

$$Q(\theta, \theta^t) = \sum_n E_{q_n}[ln(p_{Y,Z}(y_n, z_n|\theta))], \theta^{(t+1)} = argmax_\theta Q(\theta, \theta^t)$$

For minimizing KL divergence we must use $q^* = p$:

$$\gamma_n^t(k) = q_n^* = p(Z = k|x_n, \theta) = \frac{p(X = x_n|Z = k, \theta)p(Z = k)}{p(X = x_n)} = \frac{\pi_k^t p(X = x_n|Z = k, \theta)}{\sum_{m=1}^{K} \pi_m^t (X = x_n|Z = m, \theta_m)}$$

$$
\begin{aligned}
Q(\theta, \theta^{(t+1)}) &= \sum_n E_{q_n^t}[ln(p_{X,Z}(x_n, z_n|\theta))] = \sum_n E[ln(p_{X|Z}(x_n|z_n = k, \theta))p(Z_n = k|\theta)] \\
&= E[\sum_n ln(p_{X|Z}(x_n|Z_n = k, \theta)) + (\sum_n ln(p(Z_n = k|\theta)))] \\
&= E[\sum_n ln(\prod_{k=1}^{K} \pi_k^{\mathbb{I}(Z_n=k)}) + \sum_n ln(\prod_k N(x_n|\mu_k, \sum_k)^{\mathbb{I}(Z_n=k)})] \\
&= E[\sum_n \sum_k \mathbb{I}(Z_n = k)ln(\pi_k) + \sum_n \sum_k \mathbb{I}(Z_n = k)ln(N(x_n|\mu_k, \sum_k))] \\
&= \sum_n \sum_k E[\mathbb{I}(Z_n = k)]ln(\pi_k) + \sum_n \sum_k E[\mathbb{I}(Z_n = k)]ln(N(x_n|\mu_k, \sum_k))
\end{aligned}
$$

The expectation just depends on z_n so with placement of $\gamma_n^t(k)$:

$$
\begin{aligned}
Q(\theta, \theta^{t+1}) = &\sum_n \sum_k \gamma_n^t(k)ln(\pi_k) \\
&+ \sum_n \sum_k \gamma_n^t(k)ln(\frac{1}{(2\pi)^{0.5}|\sum_k|^{0.5}}exp(-0.5(x_n - \mu_n)^T \sum_k^{-1}(x_n - \mu_n)))
\end{aligned}
$$

$$
\begin{aligned}
&= \sum_n \sum_k \gamma_n^t(k)ln(\pi_k) + \sum_n \sum_k \gamma_n^t(k)(-0.5ln|\sum_k| - 0.5(x_n - \mu_n)^T \sum_k^{-1}(x_n - \mu_n)) \\
&= Q(\theta, \theta^t) = \sum_n \sum_k \gamma_n^t(k)ln(\pi_k) - 0.5\sum_n \sum_k \gamma_n^t(k)(ln|\sum_k| + (x_n - \mu_n)^T \sum_k^{-1}(x_n - \mu_k))
\end{aligned}
$$

We must maximize Q than μ_k:

$$\frac{\partial Q(\theta, \theta(t))}{\partial \mu_k} = 0,$$

$$y_n = xn - \mu_k$$

$$-\frac{\partial}{\partial y_n} y_n^T \sum_k^{-1} y_n \frac{\partial y_n}{\partial \mu_k} = -(\sum_k^{-1} + \sum_k^{-T})y_n = -(\sum_k^{-1} + \sum_k^{-T})(x_n - \mu_k)$$

so:

$$
\begin{aligned}
\frac{\partial Q(\theta, \theta(t))}{\partial \mu_k} = 0 &\rightarrow -0.5\sum_n \sum_k \gamma_n^t(k)(-(2\sum_k^{-1})(x_n - \mu_k)) = 0 \\
&\rightarrow \sum_n \gamma_n^t(k) \sum_k^{-1}(x_n - \mu_k) = 0 \rightarrow \mu_k^{(t+1)} = \frac{\sum_n \gamma_n^t(k)x_n}{\sum_n \gamma_n^t(k)}
\end{aligned}
$$

$$Q(\theta, \theta^t) = \sum_n \sum_k \gamma_n^t(k)ln(\pi_k) - 0.5\sum_n \sum_k \gamma_n^t(k)(ln|\sum_k| + (x_n - \mu_n)^T \sum_k^{-1}(x_n - \mu_k))$$

$$x^TAx = tr(x^TAx) = tr(xx^TA)$$

$$Q(\theta,\theta^t) = \sum_n\sum_k \gamma_n^t(k)ln(\pi_k) - 0.5\sum_n\sum_k \gamma_n^t(k)(ln|\sum_k| + tr((x_n-\mu_n)^T{\sum_k}^{-1}(x_n-\mu_k)))$$

$$= \sum_n\sum_k \gamma_n^t(k)ln(\pi_k) - 0.5\sum_n\sum_k \gamma_n^t(k)(ln|\sum_k| + tr((x_n-\mu_n)(x_n-\mu_k)^T)\Gamma)$$

$$\frac{Q(\theta,\theta^t)}{\partial\Gamma} = 0 \rightarrow 0.5\sum_n\sum_k \gamma_n^t(k)(\Gamma^{-T} - (x_n-\hat{\mu_k})(x_n-\hat{\mu_k})^T) = 0$$

We do maximization for every latent distribution:

$$\sum_n \gamma_n^t(k)(\sum_k -(x_n-\hat{\mu_k})(x_n-\hat{\mu_k})^T) = 0$$

$${\sum_k}^{t+1} = \frac{\sum_n \gamma_n^t(k)(x_n-\mu_k^t)(x_n-\mu_k^t)^T}{\sum_n \gamma_n^t(k)}$$

Now we maximize Q for finding π_k^{t+1}, just first part of Q depends on π_k so:

$$\pi_k^{t+1} = argmax_{\pi_k}\sum_k\sum_n \gamma_n^t(k)ln(\pi_k)$$

We use The method of Lagrange coefficients to optimize the function:

$$L(\pi_k,\lambda) = \sum_k\sum_n \gamma_n^t(k)ln(\pi_k) + \lambda\sum_k \pi_k - \lambda, \sum_k \pi_k = 1$$

$$\frac{\partial L(\pi_k,\lambda)}{\partial\pi_k} = 0 \rightarrow \sum_n \gamma_n^t(k)\pi_k^{-1} + \lambda = 0 \rightarrow \pi_k = -\frac{\sum_n \gamma_n^t(k)}{\lambda}$$

$$\sum_k \gamma_n^t(k) = \sum_k \frac{p(X=x_n|Z=k,\theta)p(Z=k)}{p(X=x_n)} = 1$$

$$\sum_k \pi_k = -\frac{\sum_n\sum_k \gamma_n^t(k)}{\lambda} = 1 = -\sum_n \frac{1}{\lambda} = -\frac{N}{\lambda} \rightarrow \lambda = -N$$

$$\pi_k^{t+1} = \frac{\sum_n \gamma_n^t(k)}{N}$$

Theory Question 5.

Compute estimate of parameters for Categorical Mixture Models for N observed data $\{x_i\}_{i=1}^{N}$

- Determine model parameters and initialize them.
- Compute complete dataset likelihood.
- Find closed-form solution for parameters using EM algorithm.

Soloution

We want to find the parameter of latent distributions :

$$\pi = \begin{bmatrix} \pi_1 \\ \pi_2 \\ \vdots \\ \pi_K \end{bmatrix} \theta_1 = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_M \end{bmatrix} \dots \theta_k = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_M \end{bmatrix}$$

From previous question we know:

$$\begin{aligned} \gamma_n^t(k) = q_n^* = p(Z=k|x_n,\theta) &= \frac{p(X=x_n|Z=k,\theta)p(Z=k)}{p(X=x_n)} \\ &= \frac{\pi_k^t p(X=x_n|Z=k,\theta)}{\sum_{m=1}^{K} \pi_m^t(X=x_n|Z=m,\theta_m)} \end{aligned}$$

$$Q(\theta,\theta^{t+1}) = E[\sum_n ln(p_{X|Z}(x_n|Z_n=k,\theta)) + (\sum_n ln(p(Z_n=k|\theta)))]$$

$$= E[\sum_n ln(\prod_{k=1}^{K} \pi_k^{\mathbb{I}(Z_n=k)}) + (\sum_n ln(\prod_k Cat(x_n|\theta_k)^{\mathbb{I}(Z_n=k)}))]$$

$$= \sum_n \sum_k E[\mathbb{I}(Z_n=k)]ln(\pi_k) + \sum_n \sum_k E[\mathbb{I}(Z_n=k)]ln(Cat(x_n|\theta_k)) =$$

$$= \sum_n \sum_k \gamma_n^t(k) ln(\pi_k) + \sum_n \sum_k \gamma_n^t(k) ln(\prod_{m=1}^{M} \theta_{km}^{\mathbb{I}(x_n=m)})$$

$$= \sum_n \sum_k \gamma_n^t(k) ln(\pi_k) + \sum_n \sum_k \sum_m \gamma_n^t(k) ln(\theta_{km})\mathbb{I}(x_n=m)$$

We use The method of Lagrange coefficients to optimize the function:

$$\theta^{t+1} = argmax_\theta Q(\theta,\theta^t)$$

$$\sum_m \theta_{km} = 1$$

$$L(\theta,\lambda) = Q(\theta,\theta^t) + \lambda(1 - \sum_m \theta_{km})$$

We take a derivative with respect to one of the parameters of the hidden distributions.(θ_{km} is mth parameter of kth distribution):

$$\frac{\partial L(\theta,\lambda)}{\partial \theta_{km}} = 0 \rightarrow \sum_n \sum_k \gamma_n^t(k) \frac{1}{\theta_{km}} \mathbb{I}(x_n = m) - \lambda = 0$$

$$\lambda = \sum_n \gamma_n^t(k) \frac{1}{\theta_{km}} \mathbb{I}(x_n = m)$$

$$\lambda = \lambda \sum_m \theta_{km} = \sum_m \lambda\theta_{km} = \sum_m (\sum_n \gamma_n^t(k) \frac{\mathbb{I}(x_n = m)}{\theta_{km}})\theta_{km} = \sum_m \sum_n \gamma_n^t(k)\mathbb{I}(x_n = m)$$

$$\lambda = \sum_n \gamma_n^t(k) \rightarrow \theta^{(}_{km}t+1) = \frac{\sum_n \gamma_n^t(k)\mathbb{I}(x_n = m)}{\sum_n \gamma_n^t(k)} = \frac{\sum_n \gamma_n^t(k)x_n}{\sum_n \gamma_n^t(k)}$$

Now we use The method of Lagrange coefficients to find π_k:

$$L(\lambda,\pi_k) = \sum_n \sum_k \gamma_n^t(k) ln(\pi_k) + \lambda(\sum_k \pi_k - 1),$$

$$\frac{\partial L(\theta,\lambda)}{\partial \pi_{km}} = 0 \rightarrow \sum_n \sum_k \gamma_n^t(k) \frac{1}{\pi_k} + \lambda = 0 \rightarrow \pi_k = \frac{-\sum_n \gamma_n^t(k)}{\lambda}$$

Now we find λ:

$$1 = \sum_k \pi_k = -\frac{\sum_n \sum_k \gamma_n^t(k)}{\lambda} \rightarrow 1 = -\frac{\sum_n (1)}{\lambda} \rightarrow \lambda = -N$$

$$\pi_k^{t+1} = \frac{\sum_n \gamma_n^t(k)}{N}$$

Simulation Questions

Libraries used in phase1 are:

- *numpy*
- *pandas*
- *matplotlib*
- from *scipy.stats* import *multivariate_normal*

code is in github

Simulation Question 1.

Each distribution has 200 data points that are concatenated in a two-dimensional array and given to you. Plot the data with three different colors in a graph.

Soloution

```python
#plot data function
def plot_data(name):
    #read data
    data = pd.read_csv(name+'.csv',header=None)

    #split data into 3 arrays
    data1 = data.iloc[0:200,1:3]
    data2 = data.iloc[200:400,1:3]
    data3 = data.iloc[400:600,1:3]

    #convert to numpy array
    data1 = data1.to_numpy()
    data2 = data2.to_numpy()
    data3 = data3.to_numpy()

    #plot the data
    plt.scatter(data1[:,0],data1[:,1],color='red', label='class1')
    plt.scatter(data2[:,0],data2[:,1],color='blue', label='class2')
    plt.scatter(data3[:,0],data3[:,1],color='green', label='class3')
    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.title(name)
    plt.legend()
    plt.show()
#plot data
plot_data('image1')
plot_data('image2')

```

Source Code 1: Simulation Question 1.

results are:

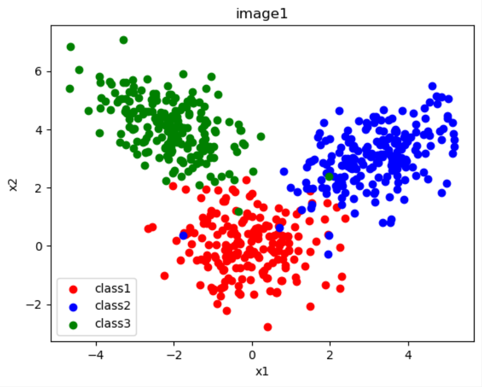


Figure 1: image1 scatter plot

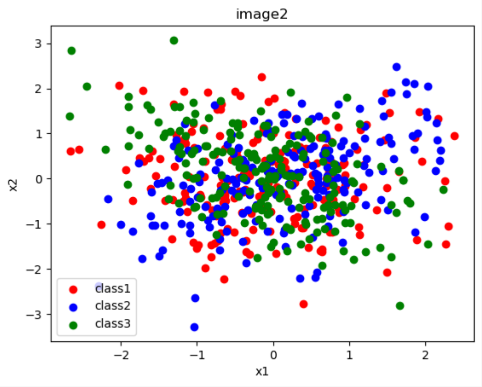


Figure 2: image2 scatter plot

Simulation Question 2.

Write a function that performs the E-step. This means assigning each data to a distribution based on the Euclidean distance. Return as output a 3×600 array specifying which distribution each data belongs to. If the R_{ij} is one, it means that the i–th data is assigned to the j–th distribution. Run this function for one iteration and report the result.

Soloution

Function used to perform E-step is:

```python
#E-Step algorithm
def E_step(data, mu, cov, pi,gamma):
    k = mu.shape[0]
    for j in range(k):
        gamma[:,j] = pi[j]*multivariate_normal(mu[j],cov[j]).pdf(data)
    gamma = gamma/gamma.sum(axis=1).reshape(-1,1)
    return gamma

```

Source Code 2: E-step algorithm

rest of the code is:

```
#initialize parameters
def initialize(data, k):
    n = data.shape[0]
    d = data.shape[1]
    pi = np.ones(k)/k
    mu = data[np.random.randint(data.shape[0], size=k), :]
    cov = np.zeros((k,d,d))
    for i in range(k):
        cov[i] = np.eye(d)
    gamma = np.zeros((n,k))
    return pi, mu, cov, gamma

#E-Step algorithm
def E_step(data, mu, cov, pi,gamma):
    k = mu.shape[0]
    for j in range(k):
        gamma[:,j] = pi[j]*multivariate_normal(mu[j],cov[j]).pdf(data)
    gamma = gamma/gamma.sum(axis=1).reshape(-1,1)
    return gamma

#assign each data point to a class
def assign(data, mu, cov):
    k = mu.shape[0]
    n = data.shape[0]
    gamma = np.zeros((n,k))
    for j in range(k):
        gamma[:,j] = multivariate_normal(mu[j],cov[j]).pdf(data)
    gamma = gamma/gamma.sum(axis=1).reshape(-1,1)
    return gamma.argmax(axis=1), gamma

#plot the results
def plot(mu, cov, data,data_class, name):
    #scatter plot of distribution
    x1 = np.linspace(data[:,0].min(),data[:,0].max(),data.shape[0])
    x2 = np.linspace(data[:,1].min(),data[:,1].max(),data.shape[0])
    X, Y = np.meshgrid(x1,x2)
    pos = np.empty(X.shape + (2,))
    pos[:, :, 0] = X; pos[:, :, 1] = Y
    k = mu.shape[0]
    colors = ['red','blue','green']
    for i in range(k):
        Z = multivariate_normal(mu[i],cov[i])
        plt.contour(X, Y, Z.pdf(pos), colors=colors[i], label='class'+str(i+1))
    plt.scatter(data[:,0],data[:,1],color='black', label='data')
    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.title(name)
    plt.legend()
    plt.show()
    plt.scatter(data[:,0],data[:,1],c=data_class, cmap='viridis')
    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.title(name)
    plt.show()

```

Source Code 3: Simulation Question 2.

```
#read data
data1 = pd.read_csv('image1.csv',header=None)
data2 = pd.read_csv('image2.csv',header=None)

#run E-Step algorithm
pi1, mu1, cov1, gamma1 = initialize(data1.iloc[:,1:3].to_numpy(), 3)
pi2, mu2, cov2, gamma2 = initialize(data2.iloc[:,1:3].to_numpy(), 3)

gamma1 = E_step(data1.iloc[:,1:3].to_numpy(), mu1, cov1, pi1,gamma1)
gamma2 = E_step(data2.iloc[:,1:3].to_numpy(), mu2, cov2, pi2,gamma2)



#assign each data point to a class
data1_class , R1 = assign(data1.iloc[:,1:3].to_numpy(), mu1, cov1)
data2_class , R2 = assign(data2.iloc[:,1:3].to_numpy(), mu2, cov2)

plot(mu1, cov1, data1.iloc[:,1:3].to_numpy(),data1_class, 'image1')
plot(mu2, cov2, data2.iloc[:,1:3].to_numpy(),data2_class, 'image2')

def save_R(R, name):
    #set 1 to max of R's row
    R = R == R.max(axis=1)[:,None]
    #save R as 0 and 1 in csv file in output folder
    R = R.astype(int)
    R = pd.DataFrame(R)
    np.savetxt('../Report/outputs/R'+name.replace('image','')+'.csv', R, delimiter=',', fmt='%d')

save_R(R1, 'image1')
save_R(R2, 'image2')

```

Source Code 4: Simulation Question 2.

results of plotting data are:

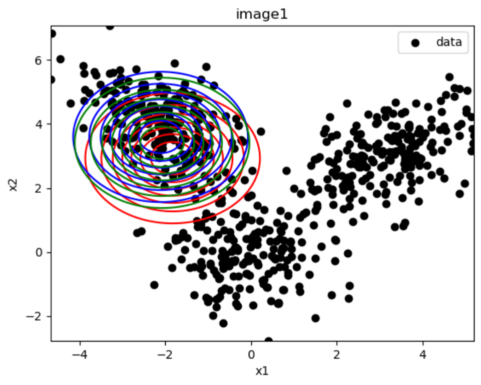


Figure 3: contour plot for estimated distribution after 1 iteration for image1

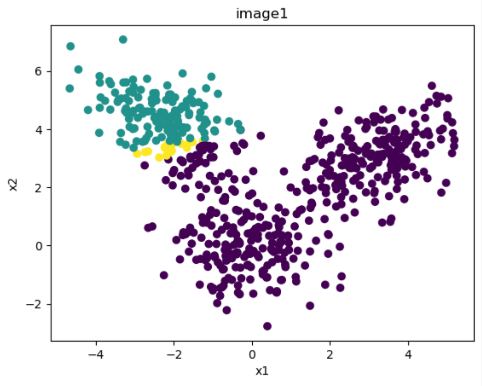


Figure 4: each data point assigned to most possible distribution for image1

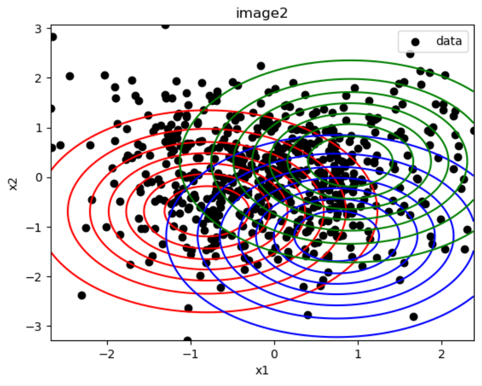


Figure 5: contour plot for estimated distribution after 1 iteration for image2

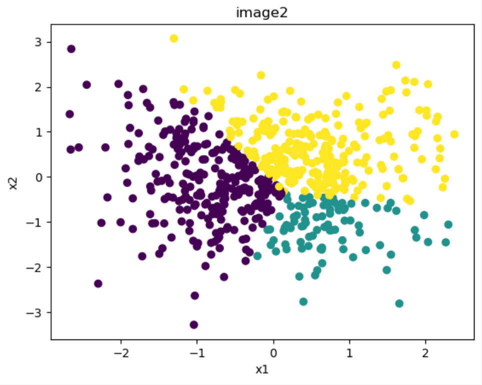


Figure 6: each data point assigned to most possible distribution for image2

Also R matrix is saved in R1 and R2 as csv file with 0 and 1 in output folder.

Simulation Question 3.

Write a function that performs M-step. This means updating the mean and variance of each distribution. Run this function for one iteration and report the new variances and means of each distribution.

Soloution

Function used to perform M-step is as follows:

```python
#M-step algorithm
def M_step(data, gamma,mu,cov,pi):
    k = gamma.shape[1]
    n = data.shape[0]
    for j in range(k):
        mu[j] = gamma[:,j].dot(data)/gamma[:,j].sum()
        cov[j] = (gamma[:,j]*(data-mu[j]).T).dot(data-mu[j])/gamma[:,j].sum()
        pi[j] = gamma[:,j].sum()/n
    return mu, cov, pi

```

Source Code 5: M-step algorithm

rest of the code for simulation question 3 is as follows:

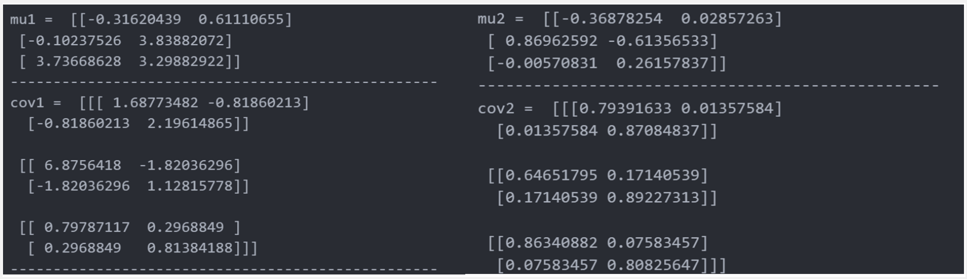
```python
#M-step
mu1, cov1, pi1 = M_step(data1.iloc[:,1:3], gamma1,mu1,cov1,pi1)
mu2, cov2, pi2 = M_step(data2.iloc[:,1:3], gamma2,mu2,cov2,pi2)

#print mean and covariance
print('mu1 = ', mu1)
print('-'*50)
print('cov1 = ', cov1)
print('-'*50)
print('mu2 = ', mu2)
print('-'*50)
print('cov2 = ', cov2)

```

Source Code 6: M-step algorithm

mean and covariance after one iteration for data are:



```
mu1 =  [[-0.31620439  0.61110655]
 [-0.10237526  3.83882072]
 [ 3.73668628  3.29882922]]
--------------------------------------------------
cov1 =  [[[ 1.68773482 -0.81860213]
  [-0.81860213  2.19614865]]

 [[ 6.8756418  -1.82036296]
  [-1.82036296  1.12815778]]

 [[ 0.79787117  0.2968849 ]
  [ 0.2968849   0.81384188]]]
--------------------------------------------------
```

```
mu2 =  [[-0.36878254  0.02857263]
 [ 0.86962592 -0.61356533]
 [-0.00570831  0.26157837]]
--------------------------------------------------
cov2 =  [[[0.79391633 0.01357584]
  [0.01357584 0.87084837]]

 [[0.64651795 0.17140539]
  [0.17140539 0.89227313]]

 [[0.86340882 0.07583457]
  [0.07583457 0.80825647]]]
```

Figure 7: mean and covariance after one iteration for image1 and image2

Simulation Question 4.

Using the functions you have written, run the EM algorithm until a convergence is reached or the maximum number of steps is passed. Replot the three new distributions and compare with the correct labels.

Soloution

Function used to apply EM algorithm is as follows:

```python
        def EM(name,data, k, max_iter):
#initialization
n = data.shape[0]
d = data.shape[1]
pi, mu, cov, gamma = initialize(data, k)
for i in range(k):
    cov[i] = np.eye(d)
gamma = np.zeros((n,k))
log_likelihood = np.zeros(max_iter)
#stop for loop when log_likelihood is not changing or max_iter is reached
for i in range(max_iter):
    #E-step
    gamma = E_step(data, mu, cov, pi,gamma)
    #M-step
    mu, cov, pi = M_step(data, gamma,mu,cov,pi)
    #compute log_likelihood
    log_likelihood[i] = 0
    for j in range(k):
        log_likelihood[i] += np.log(pi[j])*gamma[:,j].sum()
        log_likelihood[i] += gamma[:,j].dot(np.log(multivariate_normal(mu[j],cov[j]).pdf(data)))
    #stop for loop when log_likelihood is not changing or max_iter is reached
    if i > 0:
        if np.abs(log_likelihood[i]-log_likelihood[i-1]) < 1e-7:
            print('log_likelihood for',name,'stopped changing at iteration',i)
            break
return mu, cov, pi, log_likelihood

```

Source Code 7: EM algorithm

reest of the code for simulation question 4 is as follows:

```
#run EM algorithm
mu1, cov1, pi1, log_likelihood1 = EM('image1',data1.iloc[:,1:3].to_numpy
    (), 3, 200)
mu2, cov2, pi2, log_likelihood2 = EM('image2',data2.iloc[:,1:3].to_numpy
    (), 3, 200)

#assign each data point to a class
data1_class , R = assign(data1.iloc[:,1:3].to_numpy(), mu1, cov1)
data2_class , R = assign(data2.iloc[:,1:3].to_numpy(), mu2, cov2)

plot(mu1, cov1, data1.iloc[:,1:3].to_numpy(),data1_class, 'image1')
plot(mu2, cov2, data2.iloc[:,1:3].to_numpy(),data2_class, 'image2')

```

Source Code 8: EM algorithm

we plot distributions as contour plots:

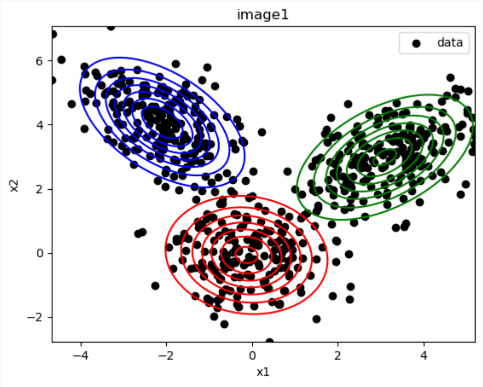


Figure 8: image1 after EM algorithm

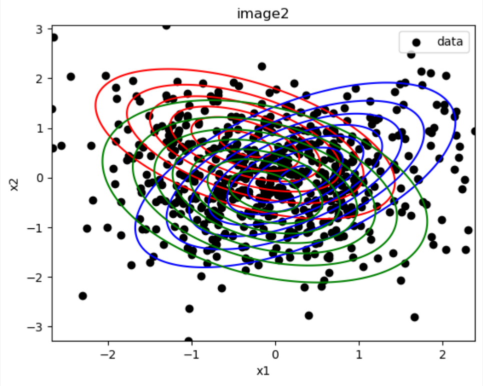


Figure 9: image2 after EM algorithm

also results for plotting classification of image1 and image2 after EM algorithm are as follows:

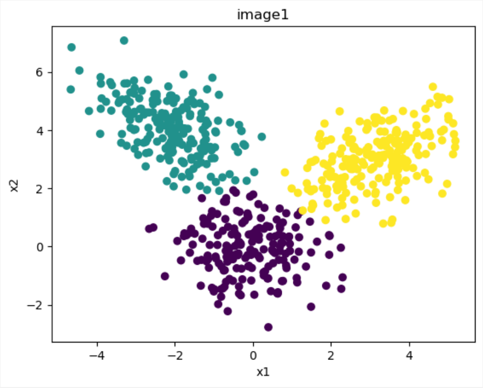


Figure 10: image1 after EM algorithm

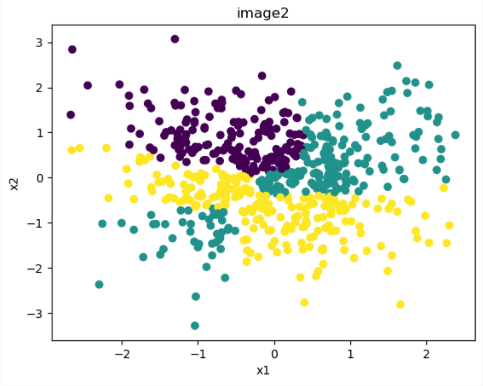


Figure 11: image2 after EM algorithm

to plot NLL and compare with correct labels, we use following code:

```
#plot log_likelihood
plt.plot(log_likelihood1, label='image1')
plt.plot(log_likelihood2, label='image2')
plt.xlabel('iteration')
plt.ylabel('log_likelihood')
plt.title('log_likelihood')
plt.legend()
plt.show()

#plot log_likelihood after 50th iteration
plt.plot(log_likelihood1[50:], label='image1')
plt.plot(log_likelihood2[50:], label='image2')
plt.xlabel('iteration')
plt.ylabel('log_likelihood')
plt.title('log_likelihood after 50th iteration')
plt.legend()
plt.show()
```

Source Code 9: plot NLL

results for plotting NLL are as follows:

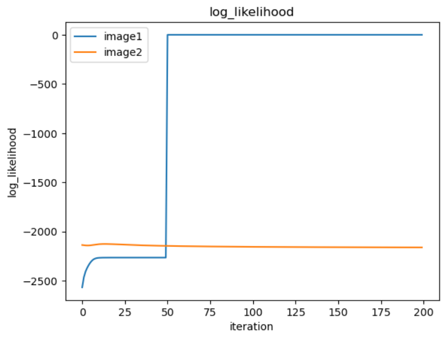


Figure 12: log likelihood

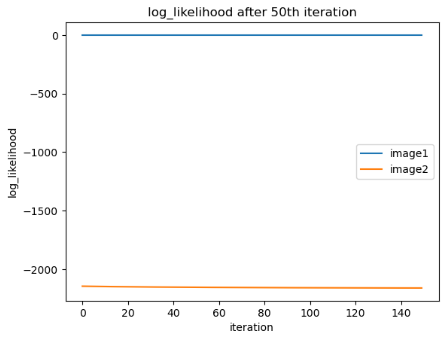


Figure 13: log likelihood after 50th iteration

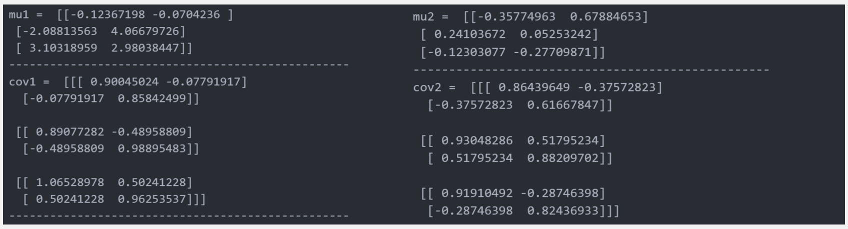
also, to obtain the new mean and covariance, we use following code:

```python
#print mean and covariance
print('mu1 = ', mu1)
print('-'*50)
print('cov1 = ', cov1)
print('-'*50)
print('mu2 = ', mu2)
print('-'*50)
print('cov2 = ', cov2)

```

Source Code 10: obtain new mean and covariance

results for new mean and covariance are as follows:



```
mu1 =  [[-0.12367198 -0.0704236 ]
 [-2.08813563  4.06679726]
 [ 3.10318959  2.98038447]]
--------------------------------------------------
cov1 =  [[[ 0.90045024 -0.07791917]
  [-0.07791917  0.85842499]]

 [[ 0.89077282 -0.48958809]
  [-0.48958809  0.98895483]]

 [[ 1.06528978  0.50241228]
  [ 0.50241228  0.96253537]]]
--------------------------------------------------
```

```
mu2 =  [[-0.35774963  0.67884653]
 [ 0.24103672  0.05253242]
 [-0.12303077 -0.27709871]]
--------------------------------------------------
cov2 =  [[[ 0.86439649 -0.37572823]
  [-0.37572823  0.61667847]]

 [[ 0.93048286  0.51795234]
  [ 0.51795234  0.88209702]]

 [[ 0.91910492 -0.28746398]
  [-0.28746398  0.82436933]]]
```

Figure 14: new mean and covariance

to plot misclassified data, we use following code:

```python
#obtain R
data1_class , R1 = assign(data1.iloc[1:601,1:3].to_numpy(), mu1, cov1)
data2_class , R2 = assign(data2.iloc[1:601,1:3].to_numpy(), mu2, cov2)

#plot missclassified data points
def missclassified(data, R,name):
    #set 1 to max of R's row
    R = R == R.max(axis=1)[:,None]
    #save R as 0 and 1
    R = R.astype(int)
    R = pd.DataFrame(R)
    #find which column has most 1
    a = R.iloc[0:200].sum(axis=0).idxmax()
    b = R.iloc[200:400].sum(axis=0).idxmax()
    c = R.iloc[400:600].sum(axis=0).idxmax()
    #check which column has 1
    for i in range(R.shape[0]):
        if R.iloc[i,a] != 1 and i<200 :
            plt.scatter(data[i,0], data[i,1], color='red')
        if R.iloc[i,b] != 1 and (i>200 and i<400):
            plt.scatter(data[i,0], data[i,1], color='blue')
        if R.iloc[i,c] != 1 and i>400:
            plt.scatter(data[i,0], data[i,1], color='green')
    plt.title('missclassified data points for '+name)
    plt.xlabel('x1')
    plt.ylabel('x2')
    #set legend
    red_patch = mpatches.Patch(color='red', label='distribution 1')
    blue_patch = mpatches.Patch(color='blue', label='distribution 2')
    green_patch = mpatches.Patch(color='green', label='distribution 3')
    plt.legend(handles=[red_patch,blue_patch,green_patch])
    plt.show()

missclassified(data1.iloc[:,1:3].to_numpy(), R1,'image1')
missclassified(data2.iloc[:,1:3].to_numpy(), R2,'image2')

```

Source Code 11: plot misclassified data

results for plotting misclassified data are as follows:

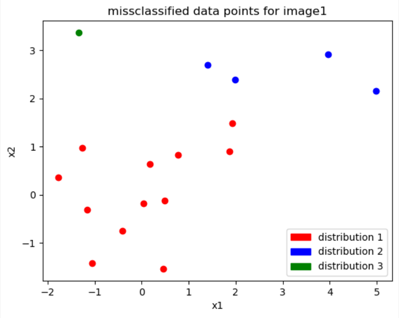


Figure 15: misclassified data points for image1

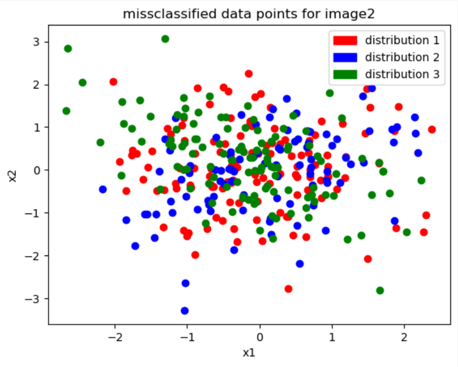


Figure 16: misclassified data points for image2

Simulation Question 5.

Compare the parameters obtained from each of the images and explain the reason for their difference.

Soloution

Data from image1 have big difference between mean and also have little variances. So, we can say that data from image1 are less concentrated.That is why, we can see that data from image1 are more spread out than data from image2 nad are easy to separate via EM algorithm.It's also shows that to run EM algorithm for GMM(Gaussian Mixture Model) is more efficient for distributions with small variances and big difference between means.

End of Project