Advanced machine learning

Joakim Olsen

May 2019

Introduction

I have answered more questions than suggested, with different difficulties. This is because I learn a lot from doing exercises, and I therefore wanted to do more than suggested. I have therefore included easier tasks in the report, even though they will not matter for the evaluation since I also did more difficult tasks. At the bottom there is one task of (*****), and 2 tasks of (****), which should be sufficient for the evaluation. I therefore don't expect the first tasks (*)-(***) to be corrected/evaluated, but they are there since I already did them. Feel free to only correct the tasks that are relevant for the evaluation.

1 Exercises (*)

1.1 Write an example similar to the example in the previous slide but changing the joint distributions, where variables have a real meaning (real sense) (e.g. alcohol in blood versus car accident, weather versus car accident, ...). Then compute the $H(X), H(X,Y), H(X|Y)$ and $I(X;Y)$.

The following table shows a (completely fictive) example concerning weather versus car accidents:

Table 1: Joint distribution where $X = \{\text{sun, fog, rain, snow}\}$ is weather and $Y = \{\text{few, medium, many}\}$ are the occurence of car accidents.

$Y \setminus X$	sun	fog	rain	snow	$\sum$
few	7/32	1/32	2/32	1/32	11/32
medium	3/32	2/32	4/32	2/32	11/32
many	2/32	2/32	3/32	3/32	10/32
$\sum$	12/32	5/32	9/32	6/32	1

We then have

$$\begin{aligned} H(X) &= -\sum_{x\in X} p(x)\ \log p(x) \\ &= -\frac{3}{8}\log\frac{3}{8} - \frac{5}{32}\log\frac{5}{32} - \frac{9}{32}\log\frac{9}{32} - \frac{3}{16}\log\frac{3}{16} = 1.92, \end{aligned} \tag{1}$$

$$\begin{aligned} H(X,Y) &= -\sum_{x\in X}\sum_{y\in Y} p(x,y)\ \log p(x,y) \\ &= -\frac{7}{32}\log\frac{7}{32} - \frac{3}{32}\log\frac{3}{32} - \frac{2}{32}\log\frac{2}{32} - \frac{1}{32}\log\frac{1}{32} - \frac{2}{32}\log\frac{2}{32} - \frac{2}{32}\log\frac{2}{32} \\ &- \frac{2}{32}\log\frac{2}{32} - \frac{4}{32}\log\frac{4}{32} - \frac{3}{32}\log\frac{3}{32} - \frac{1}{32}\log\frac{1}{32} - \frac{2}{32}\log\frac{2}{32} - \frac{3}{32}\log\frac{3}{32} \\ &= 3.38, \end{aligned} \tag{2}$$

$$\begin{aligned} H(X|Y) &= \sum_{y\in Y} p(y)\, H(X|Y=y) \\ &= \frac{11}{32}H(\frac{7}{11},\frac{1}{11},\frac{2}{11},\frac{1}{11}) + \frac{11}{32}H(\frac{3}{11},\frac{2}{11},\frac{4}{11},\frac{2}{11}) + \frac{10}{32}H(\frac{2}{10},\frac{2}{10},\frac{3}{10},\frac{3}{10}) \\ &= \frac{11}{32}\cdot 1.49 + \frac{11}{32}\cdot 1.94 + \frac{10}{32}\cdot 1.97 = 1.79 \end{aligned} \tag{3}$$

and

$$I(X;Y) = H(X) - H(X|Y) = 1.92 - 1.79 = 0.13. \tag{4}$$

1.2 Compute $H(C), H(S), H(C|S), H(S|C), H(C,S)$ and $I(C;S)$ (class slides, p.13) [1].

$$\begin{aligned} H(C) &= -\sum_{c\in C} p(c)\, \log p(c) \\ &= -0.46\log 0.46 - 0.33\log 0.33 - 0.21\log 0.21 = 1.52 \end{aligned} \tag{5}$$

$$\begin{aligned} H(S) &= -\sum_{s\in S} p(s)\, \log p(s) \\ &= -0.86\log 0.86 - 0.14\log 0.14 = 0.58 \end{aligned} \tag{6}$$

$$\begin{aligned} H(C|S) &= \sum_{s\in S} p(s)\, H(C|S=s) \\ &= 0.86\cdot H(\frac{1}{2},\frac{15}{43},\frac{13}{86}) + 0.14\cdot H(\frac{3}{14},\frac{3}{14},\frac{8}{14}) \\ &= 0.86\cdot 1.44 + 0.14\cdot 1.41 = 1.44 \end{aligned} \tag{7}$$

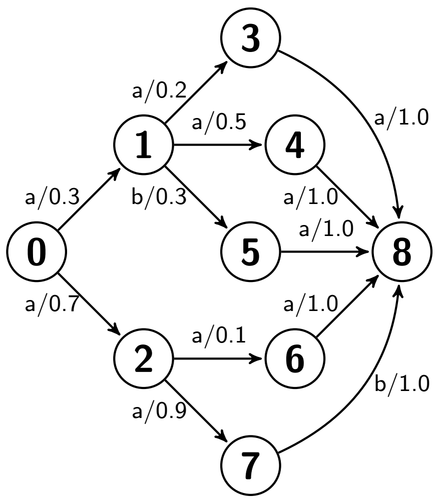
$$\begin{aligned} H(S|C) &= \sum_{c\in C} p(c)\, H(S|C=c) \\ &= 0.46\cdot H(\frac{43}{46},\frac{3}{46}) + 0.33\cdot H(\frac{30}{33},\frac{3}{33}) + 0.21\cdot H(\frac{13}{21},\frac{8}{21}) \\ &= 0.46\cdot 0.35 + 0.33\cdot 0.44 + 0.21\cdot 0.96 = 0.51 \end{aligned} \tag{8}$$

$$\begin{aligned} H(C,S) &= H(C) + H(S|C) = H(S) + H(C|S) \\ &= 1.52 + 0.51 = 0.58 + 1.44 = 2.02/2.03 \\ &\text{(Round off error)} \end{aligned} \tag{9}$$

$$\begin{aligned} I(C;S) &= H(C) - H(C|S) = H(S) - H(S|C) \\ &= 1.52 - 1.44 = 0.58 - 0.51 = 0.07/0.08 \\ &\text{(Round off error)} \end{aligned} \tag{10}$$

1.3 Define a PFA similar to the one that is in class slide p.15 [1] and compute expressions (17)-(19). Choose a string with more than two derivations for (17).

We define a PFA $\mathcal{A}$ as follows:



Sequence entropy:

$$\begin{aligned}
H(\mathcal{A}) &= -\sum_{w \in L(\mathcal{A})} p_{\mathcal{A}}(w) \log p_{\mathcal{A}} \\
&= -p_{\mathcal{A}}(aaa) \log p_{\mathcal{A}}(aaa) - p_{\mathcal{A}}(aba) \log p_{\mathcal{A}}(aba) - p_{\mathcal{A}}(aab) \log p_{\mathcal{A}}(aab) \\
&= -0.28 \log(0.28) - 0.09 \log(0.09) - 0.63 \log(0.63) = 1.25
\end{aligned} \tag{11}$$

Entropy on the paths given observation sequence aaa:

$$\begin{aligned}
H_{\mathcal{A}}(\theta|aaa) &= -\sum_{\theta \in \Theta_{\mathcal{A}}(aaa)} p_{\mathcal{A}}(\theta|aaa) \log p_{\mathcal{A}}(\theta|aaa) \\
&= -p_{\mathcal{A}}(0138|aaa) \log p_{\mathcal{A}}(0138|aaa) - p_{\mathcal{A}}(0148|aaa) \log p_{\mathcal{A}}(0148|aaa) \\
&\quad - p_{\mathcal{A}}(0268|aaa) \log p_{\mathcal{A}}(0268|aaa) \\
&= -\frac{6}{28} \log \frac{6}{28} - \frac{15}{28} \log \frac{15}{28} - \frac{7}{28} \log \frac{7}{28} = 1.46
\end{aligned} \tag{12}$$

Derivational entropy:

$$\begin{aligned}
H_d(\mathcal{A}) &= -\sum_{\theta \in \Theta(\mathcal{A})} p_{\mathcal{A}}(\theta) \log p_{\mathcal{A}}(\theta) \\
&= -p_{\mathcal{A}}(0138) \log p_{\mathcal{A}}(0138) - p_{\mathcal{A}}(0148) \log p_{\mathcal{A}}(0148) - p_{\mathcal{A}}(0158) \log p_{\mathcal{A}}(0158) \\
&\quad - p_{\mathcal{A}}(0268) \log p_{\mathcal{A}}(0268) - p_{\mathcal{A}}(0278) \log p_{\mathcal{A}}(0278) \\
&= -0.06 \log(0.06) - 0.15 \log(0.15) - 0.09 \log(0.09) - 0.07 \log(0.07) - 0.63 \log(0.63) \\
&= 1.66
\end{aligned} \tag{13}$$

2 Exercises (**)

2.1 Compute $H(C|L,S), H(L|C,S), H(S|C,L)$ (class slides, p.13) [1].

$$\begin{aligned} H(C|L,S) &= \sum_{l\in L}\sum_{s\in S} p(l,s)\, H(C|L=l,S=s) \\ &= 0.57\cdot H(\frac{30}{57},\frac{20}{57},\frac{7}{57}) + 0.29\cdot H(\frac{13}{29},\frac{10}{29},\frac{6}{29}) + 0.05\cdot H(\frac{1}{5},\frac{1}{5},\frac{3}{5}) + 0.09\cdot H(\frac{2}{9},\frac{2}{9},\frac{5}{9}) \\ &= 0.57\cdot 1.39 + 0.29\cdot 1.52 + 0.05\cdot 1.37 + 0.09\cdot 1.44 = 1.43 \end{aligned} \tag{14}$$

$$\begin{aligned} H(L|C,S) &= \sum_{c\in C}\sum_{s\in S} p(c,s)\, H(L|C=c,S=s) \\ &= 0.43\cdot H(\frac{30}{43},\frac{13}{43}) + 0.30\cdot H(\frac{20}{30},\frac{10}{30}) + 0.13\cdot H(\frac{7}{13},\frac{6}{13}) \\ &+ 0.03\cdot H(\frac{1}{3},\frac{2}{3}) + 0.03\cdot H(\frac{1}{3},\frac{2}{3}) + 0.08\cdot H(\frac{3}{8},\frac{5}{8}) \\ &= 0.43\cdot 0.88 + 0.36\cdot 0.92 + 0.13\cdot 1.00 + 0.08\cdot 0.95 = 0.92 \end{aligned} \tag{15}$$

$$\begin{aligned} H(S|C,L) &= \sum_{c\in C}\sum_{l\in L} p(c,l)\, H(S|C=c,L=l) \\ &= 0.31\cdot H(\frac{30}{31},\frac{1}{31}) + 0.21\cdot H(\frac{20}{21},\frac{1}{21}) + 0.10\cdot H(\frac{7}{10},\frac{3}{10}) \\ &+ 0.15\cdot H(\frac{13}{15},\frac{2}{15}) + 0.12\cdot H(\frac{10}{12},\frac{2}{12}) + 0.11\cdot H(\frac{6}{11},\frac{5}{11}) \\ &= 0.31\cdot 0.21 + 0.21\cdot 0.28 + 0.10\cdot 0.88 + 0.15\cdot 0.57 + 0.12\cdot 0.65 + 0.11\cdot 0.99 = 0.48 \end{aligned} \tag{16}$$

2.2 Reproduce an example (class slides, p.54-55 [1]) similar to the previous example with three unidimensional distributions, with known mean and equal and known variance where only π_1, π_2, π_3 are unknown. The data have to be generated with three gaussians. Obtain the corresponding plots.

In this task, I have made a gaussian mixture model where the true probabilities are given by $\pi_1 = 0.5$, $\pi_2 = 0.3$ and $\pi_3 = 0.2$. The means are known and given by $\mu_1 = 0$, $\mu_2 = 2$ and $\mu_3 = 5$. The variance is known and equal for all, $\sigma_1^2 = \sigma_2^2 = \sigma_3^2 = 1$. The training data for the EM-algorithm is made by 10000 samples drawn from the gaussian mixture model with the parameters as given above. Figure 1 shows the true distribution, together with a histogram for the training data.

The EM-algorithm is implemented in python, and can be read in Listing 1 in appendix. The membership weights, $\hat{z}_{km}$, are first calculated by equation (67) on page 53 in the class slides [1]. Then $\pi_k^{(t+1)}$ is calculated by equation (74) on page 59 in the class slides [1]. Note that with $\lambda = 0$, this expression will become identical to equation (68) on page 53 in the class slides [1]. The algorithm is run using initial probabilities $\pi_1 = 0.8$ and $\pi_2 = \pi_3 = 0.1$. λ is set to zero since this is not the regularized EM-algorithm. The convergence the first 5 iterations is shown in figure 2.

After 5 iterations, we have $\pi_1 = 0.514$, $\pi_2 = 0.287$ and $\pi_3 = 0.199$, which is already very close to the true values. Thus, it seems to converge quickly towards the true values.

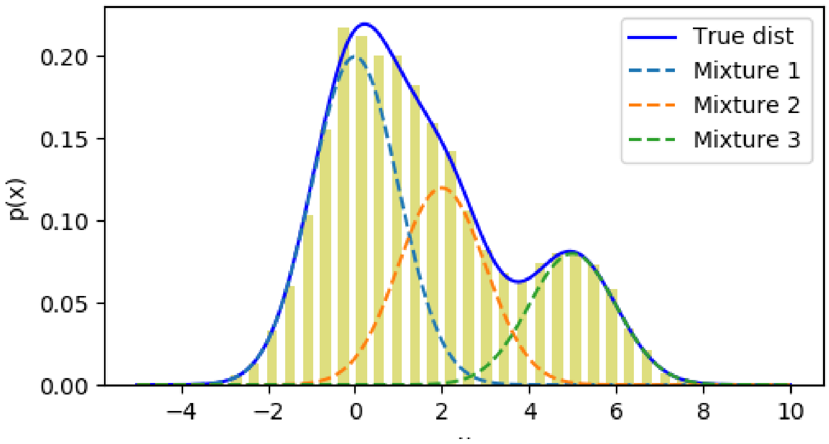


Figure 1: True distribution of gaussian mixture model with $\pi_1 = 0.5$, $\pi_2 = 0.3$, $\pi_3 = 0.2$, $\mu_1 = 0$, $\mu_2 = 2$, $\mu_3 = 5$ and $\sigma_1^2 = \sigma_2^2 = \sigma_3^2 = 1$, together with histogram of 10000 samples drawn from the distribution.

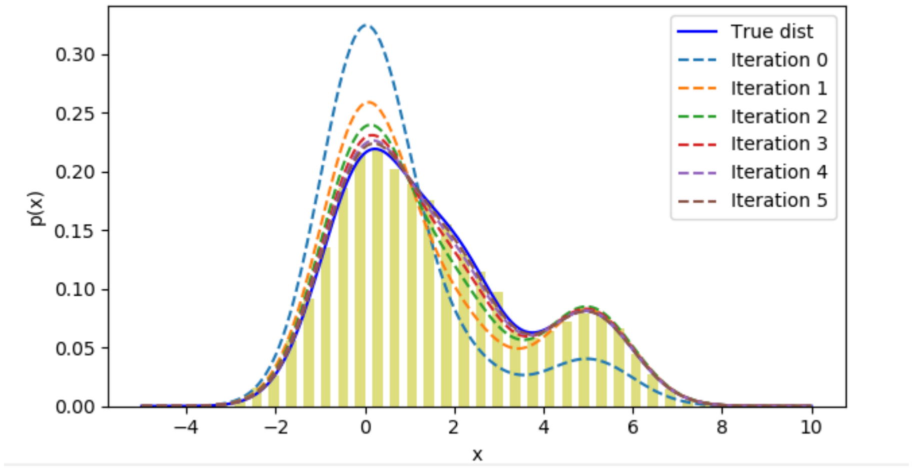


Figure 2: Plot showing the model after the first 5 iterations of the EM-algorithm, where the true distribution and the training set is as in figure 1, and initial parameters are $\pi_1 = 0.8$, $\pi_2 = \pi_3 = 0.1$, with μ and σ^2 equal to the true distribution.

3 Exercises (***)

3.1 Reproduce an example similar to the previous example (class slides, p.54-55 [1]) with two unidimensional distributions, with equal and known variance where the mean and π_1, π_2 are unknown. Obtain the corresponding plots.

In this task, I have made a gaussian mixture model where the true probabilities are given by $\pi_1 = 0.7$ and $\pi_2 = 0.3$, and the means are given by $\mu_1 = 2$ and $\mu_2 = 5$. The variance is known and equal for all, $\sigma_1^2 = \sigma_2^2 = 1$. The training data for the EM-algorithm is made by 10000 samples drawn from the gaussian mixture model with the parameters as given above. Figure 3 shows the true distribution, together with a histogram for the training data.

The EM-algorithm is implemented in python, and can be read in Listing 1 in appendix. The membership weights, $\hat{z}_{km}$, are first calculated by equation (67) on page 53 in the class slides [1]. Then $\pi_k^{(t+1)}$

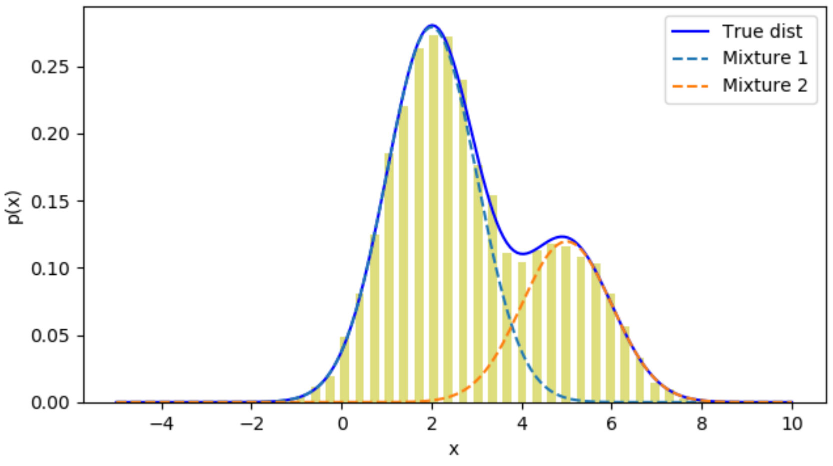


Figure 3: True distribution of gaussian mixture model with $\pi_1 = 0.7$, $\pi_2 = 0.3$, $\mu_1 = 2$, $\mu_2 = 5$ and $\sigma_1^2 = \sigma_2^2 = 1$, together with histogram of 10000 samples drawn from the distribution.

and $\mu_k^{(t+1)}$ is calculated by equation (74) and (76) on page 59 in the class slides [1]. Note that with $\lambda = 0$, equation (74) will become identical to equation (68) on page 53 in the class slides [1], and equation (76) will simply become the empirical mean weighted by the membership values $\hat{z}_{km}$. The algorithm is run using initial parameters $\pi_1 = 0.1$, $\pi_2 = 0.9$, $\mu_1 = 1$ and $\mu_2 = 2$. λ is set to zero since this is not the regularized EM-algorithm. Also, `unknown_mys` is set to `True`, since they are also to be modified by the algorithm. The convergence the first 15 iterations is shown in figure 4.

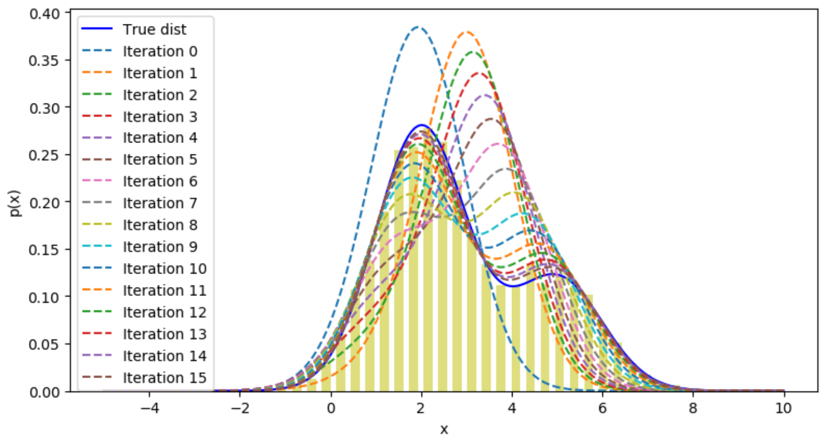


Figure 4: Plot showing the model after the first 15 iterations of the EM-algorithm, where the true distribution and the training set is as in figure 3, and initial parameters are $\pi_1 = 0.1$, $\pi_2 = 0.9$, $\mu_1 = 1$, $\mu_2 = 2$, with σ^2 equal to the true distribution.

After 15 iterations, we have $\pi_1 = 0.676$, $\pi_2 = 0.324$, $\mu_1 = 1.962$ and $\mu_2 = 4.861$ which is very close to the true values. It converges a little slower than the example where only π is unknown, but this is natural since there now is one more parameter to adjust. All in all, it seems to converge quickly towards the true values.

3.2 Reproduce an example similar to the previous example (class slides, p.54-55 [1]) with two unidimensional distributions, with equal and known mean where the variance of each distribtuion and π_1, π_2 are unknown. Obtain the corresponding plots.

In this task, I have made a gaussian mixture model where the true probabilities are given by $\pi_1 = 0.7$ and $\pi_2 = 0.3$, and the variances are given by $\sigma_1^2 = 3$ and $\sigma_2^2 = 0.4$. The mean is known and given by, $\mu_1 = 2$ and $\mu_2 = 5$. The training data for the EM-algorithm is made by 10000 samples drawn from the gaussian mixture model with the parameters as given above. Figure 5 shows the true distribution, together with a histogram for the training data.

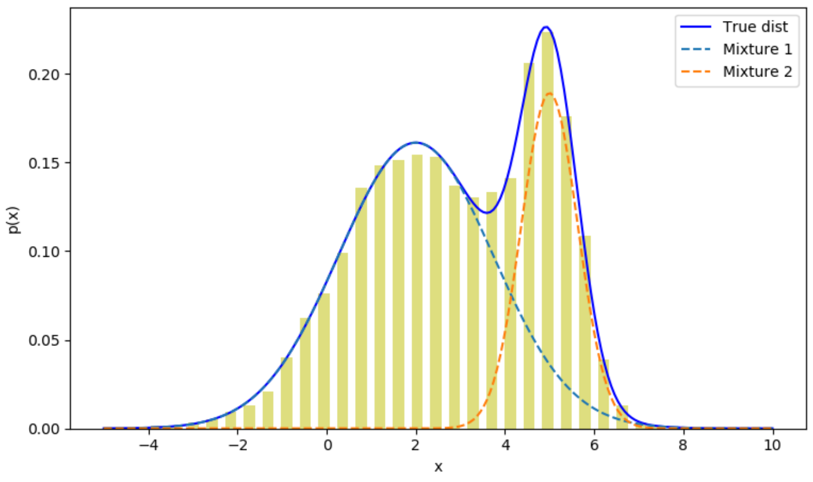


Figure 5: True distribution of gaussian mixture model with $\pi_1 = 0.7$, $\pi_2 = 0.3$, $\mu_1 = 2$, $\mu_2 = 5$, $\sigma_1^2 = 3$ and $\sigma_2^2 = 0.4$, together with histogram of 10000 samples drawn from the distribution.

The EM-algorithm is implemented in python, and can be read in Listing 1 in appendix. The membership weights, $\hat{z}_{km}$, are first calculated by equation (67) on page 53 in the class slides [1]. Then $\pi_k^{(t+1)}$ and $\sigma_k^{2(t+1)}$ is calculated by equation (74) and (77) on page 59 in the class slides [1]. Note that with $\lambda = 0$, equation (74) will become identical to equation (68) on page 53 in the class slides [1], and equation (77) will simply become the empirical variance weighted by the membership values $\hat{z}_{km}$. The algorithm is run using initial parameters $\pi_1 = 0.1$, $\pi_2 = 0.9$ and $\sigma_1^2 = \sigma_2^2 = 1$. λ is set to zero since this is not the regularized EM-algorithm. Also, `unknown_sigmas` is set to `True`, since they are also to be modified by the algorithm. The convergence of the first 10 iterations is shown in figure 6.

After 10 iterations, we have $\pi_1 = 0.680$, $\pi_2 = 0.320$, $\sigma_1^2 = 2.891$ and $\sigma_2^2 = 0.416$ which is very close to the true values. It converges a little slower than the example where only π is unknown, but this is natural since there now is one more parameter to adjust. It converges a little faster than the previous task with unknown μ. Thus it seems easier for the algorithm to find σ^2 than μ. All in all, it seems to converge quickly towards the true values.

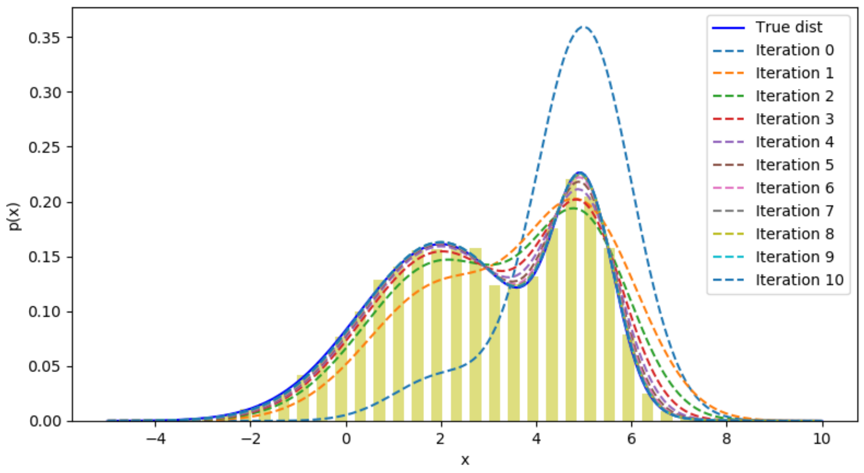


Figure 6: Plot showing the model after the first 10 iterations of the EM-algorithm, where the true distribution and the training set is as in figure 5, and initial parameters are $\pi_1 = 0.1$, $\pi_2 = 0.9$ and $\sigma_1^2 = \sigma_2^2 = 1$, with μ equal to the true distribution.

4 Exercises (****)

4.1 Reproduce an example similar to the unidimensional example (class slides, p.61-63 [1]) with two unidimensional distributions, with equal and known variance where the mean of each distribtuion and π_1, π_2 are unknown. Obtain the corresponding plots.

In this task, I have made two models to work with. The first one is identical to the task from section 3.1 (see figure 3). The second is a gaussian mixture model where the true probabilities are given by $\pi_1 = 0.2$, $\pi_2 = 0.5$ and $\pi_3 = 0.3$, and the mean is given by $\mu_1 = 1$, $\mu_2 = 4$ and $\mu_3 = 8$. The variance is known and equal, $\sigma_1^2 = \sigma_2^2 = \sigma_3^2 = 1$. The training data for the EM-algorithm is made by 10000 samples drawn from the gaussian mixture model with the parameters as given above. Figure 7 shows the true distribution, together with a histogram for the training data.

The regularized EM-algorithm is implemented in python, and can be read in Listing 1 in appendix. The membership weights, $\hat{z}_{km}$, are first calculated by equation (67) on page 53 in the class slides [1]. Then $\pi_k^{(t+1)}$ and $\mu_k^{(t+1)}$ is calculated by equation (74) and (76) on page 59 in the class slides [1]. λ is set to 0.2. Also, `unknown_mys` is set to `True`, since they are also to be modified by the algorithm. The algorithm is run two times for the first example: First using initial parameters $\pi_1 = 0.1$, $\pi_2 = 0.9$, $\mu_1 = 1$ and $\mu_2 = 2$. The convergence of the first 15 iterations is shown in figure 8.

After 15 iterations, we have $\pi_1 = 0.690$, $\pi_2 = 0.310$, $\mu_1 = 1.952$ and $\mu_2 = 5.101$ which is very close to the true values. It also seems to converge a little faster than for the non-regularized version (see figure 4), but the difference does not seem to be that big. Thus, for this task there is not really that much motivation for using the regularized algorithm.

Let us now look at the next run for the same example. In this case, we have initial parameters given by $\pi_1 = 0.3$, $\pi_2 = 0.3$, $\pi_3 = 0.4$, $\mu_1 = 1$, $\mu_2 = 1.5$ and $\mu_3 = 2$. The plot obtained from running the regularized algorithm is very similar to figure 8, and we therefore do not include it. The interesting thing here to note is that by using $\lambda = 0.2$, after 20 iterations, one of the gaussians is removed, and we obtain the model with parameters $\pi_1 = 0.712$, $\pi_2 = 0.288$, $\mu_1 = 2.004$ and $\mu_2 = 5.181$, which is very close to the true values. However, by setting $\lambda = 0$, which corresponds to the non-regularized EM-algorithm, we have after 20 iterations parameters $\pi_1 = 0.313$, $\pi_2 = 0.392$, $\pi_3 = 0.296$, $\mu_1 = 1.920$, $\mu_2 = 2.088$ and $\mu_3 = 4.996$. As we can see, there are two gaussians that are practically on top of each other, and the probabilities $\pi_1 + \pi_2 = 0.705$ shows that this model has two gaussians that should in fact be merged together. The regularized version penalizes mutual information, so that it is able to remove gaussians if there are initially too many. This is a very interesting feature of the regularized algorithm, that doesn't

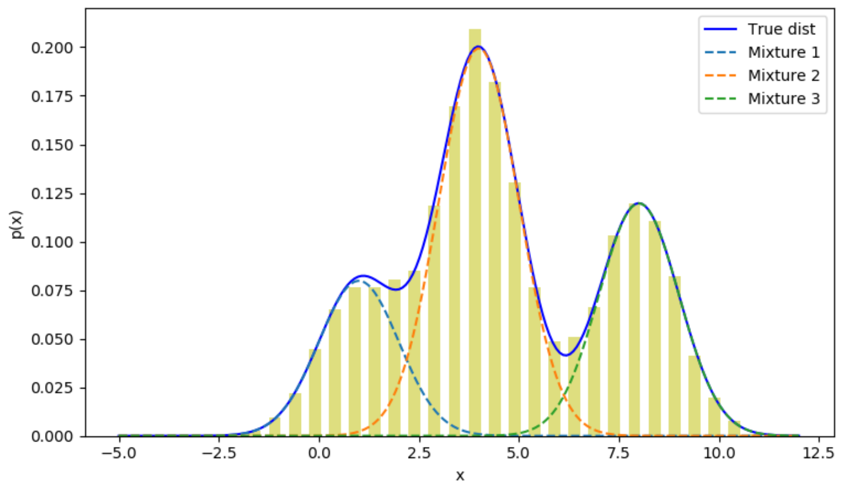


Figure 7: True distribution of gaussian mixture model with $\pi_1 = 0.2$, $\pi_2 = 0.5$, $\pi_3 = 0.3$, $\mu_1 = 1$, $\mu_2 = 4$, $\mu_3 = 8$ and $\sigma_1^2 = \sigma_2^2 = \sigma_3^2 = 1$, together with histogram of 10000 samples drawn from the distribution.

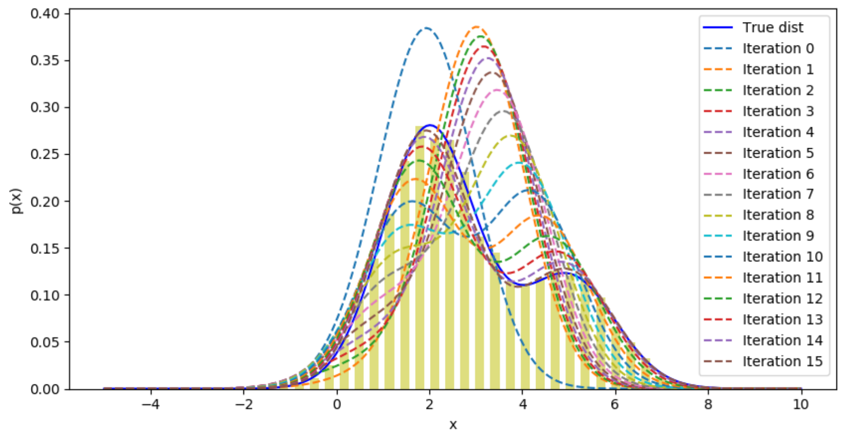


Figure 8: Plot showing the model after the first 15 iterations of the regularized EM-algorithm with $\lambda = 0.2$, where the true distribution and the training set is as in figure 3, and initial parameters are $\pi_1 = 0.1$, $\pi_2 = 0.9$, $\mu_1 = 1$ and $\mu_2 = 2$, with σ^2 equal to the true distribution.

work for the normal one. If we don't know how many gaussian models the mixed model is composed of, the regularized EM-algorithm can remove reduntant ones, to give a model that is closer to the real one.

Let us look at potential problems with the algorithm, using the other example described in figure 7. We run the algorithm with parameters $\lambda = 0.2$, $\pi_1 = 0.3$, $\pi_2 = 0.3$, $\pi_3 = 0.4$, $\mu_1 = 1$, $\mu_2 = 3$ and $\mu_3 = -5$. Now, there is a mechanism that removes gaussians, and it is based on removing gaussians that are too unlikely the be true. In the first iteration, the third gaussian is removed, since it is extremely unlikely to have a gaussian with $\mu = -5$ given the training data. Thus, the mechanism that can remove gaussians that are redundant, can also remove gaussians that are necessary, though badly initialized. It seems the algorithm needs proper initialization in order to work. We also see this when changing the initial means to $\mu_1 = 1$, $\mu_2 = 3$ and $\mu_3 = 5$. In this case, the algorithm converges nicely towards the true values. Thus, it seems necessary for the regularized EM algorithm to have proper initialization.

4.2 Reproduce an example similar to the bi-dimensional example, but the inicialization has to be performed with the c-means algorithm. Obtain the corresponding plots.

In this task we will look into the problem discussed at the end of 4.1: When there is a mechanism for removing gaussians that are too unlikely, the algorithm might do this wrongly if the initialization is bad. We should therefore initialize the means μ. We will implement the C-means algorithm to do this. Since the C-means algorithm allows points to belong to several clusters, it is also possible to estimate π from this algorithm. Finally, the empirical covariance can be calculated by adding each point to its most likely cluster.

In this task I have made an example where the parameters are as described in figure 9. A training set of 1000 samples is drawn from this gaussian mixture model, and the training set is shown as a scatter plot in figure 9, together with 95 % ellipses for each gaussian in the mixture model.

The C-means algorithm is made based on [2], by first initializing random means. Then for each iteration, the matrix $U^{(k)}$ is calculated using

$$u_{ij} = \frac{1}{\sum_{k=1}^{c}\left(\frac{||x_i - c_j||}{||x_i - c_k||}\right)^2}, \tag{17}$$

and then the means are calculated by the formula

$$c_j^{(k+1)} = \frac{\sum_{i=1}^{N} u_{ij}^2 \cdot x_i}{\sum_{i=1}^{N} u_{ij}^2}. \tag{18}$$

The C-means algorithm is implemented in Python and can be read in Listing 2 in Appendix.

The means obtained after running the C-means algorithm with 12 clusters, together with the 95 % confidence ellipses obtained by estimating the covariance matrices in the C-means algorithm, is also shown in figure 9. We see from the plot that the initial parameters from the C-means algorithm are quite close to the true value, and we therefore expect the regularized EM-algorithm to perform well with these initial data. Here, we have used 12 initial clusters. We see that some of the mixtures have too many clusters (like the dark green and pink one). We trust the regularized EM-algorithm to remove the redundant gaussians, since the algorithm penalizes models with mutual information. Also, we see one case of too few clusters in an area (there is only one cluster covering the blue and light pink data). Ideally, we need each gaussian to have at least one cluster, since we don't expect the algorithm to understand it has to move one of the gaussians quite far (as discussed in the last section, if the initialization is bad, the gaussian is likely to be removed by the algorithm). We therefore see that using 12 clusters is not a guarantee that each gaussian gets a cluster, and we should therefore initialize the C-means algorithm with more clusters.

Figure 10 shows the result of running the regularized EM-algorithm after using the C-means algorithm as initialization.

The log-likelihood of the models, as well as the number of gaussians in the models, as a function of iterations is shown in figure 11.

We see from both figures that when using 15 clusters and $\lambda \geq 0.15$, the final model has correctly found the 10 gaussians. We see from figure 10 that not all of the ellipses are quite right, especially the one for the dark green data points is a bit wrong. 1000 training samples is after all not that many, and with more data points this could be improved. There are however two clusters on top of each other, and correctly distinguishing them is difficult. When $\lambda < 0.15$, the algorithm does not remove reduntant gaussians. The log-likelihood is naturally higher for lower values of λ, but for $\lambda < 0.15$, we are witnessing a kind of over-fitting, which is why it is better to use a higher λ even though the log-likelihood is lower. All in all, it seems the algorithm works very well on this data, and the fitted model is very close to the true model.

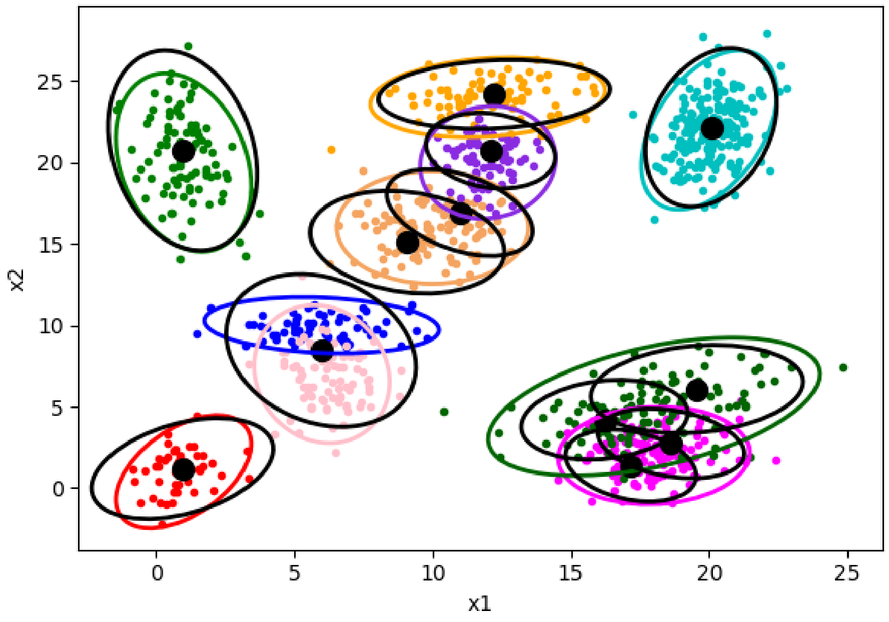


Figure 9: Plot showing 95 % confidence ellipses for each gaussian in a gaussian mixture model as colored ellipses, together with 1000 samples drawn from the model, given by probabilities $\pi_1 = 0.05$, $\pi_2 = 0.09$, $\pi_3 = 0.07$, $\pi_4 = 0.11$, $\pi_5 = 0.08$, $\pi_6 = 0.19$, $\pi_7 = 0.14$, $\pi_8 = 0.12$, $\pi_9 = 0.07$ and $\pi_{10} = 0.08$, means $\mu_1 = (1, 1)$, $\mu_2 = (1, 20)$, $\mu_3 = (6, 10)$, $\mu_4 = (10, 16)$, $\mu_5 = (12, 24)$, $\mu_6 = (20, 22)$, $\mu_7 = (18, 2)$, $\mu_8 = (18, 5)$, $\mu_9 = (12, 20)$ and $\mu_{10} = (6, 7)$, and covariance matrices $\Sigma_1 = [[1, 0.7], [0.7, 2]]$, $\Sigma_2 = [[1, -0.6], [-0.6, 5]]$, $\Sigma_3 = [[3, -0.2], [-0.2, 0.5]]$, $\Sigma_4 = [[2, 0], [0, 2]]$, $\Sigma_5 = [[3, 0.3], [0.3, 1]]$, $\Sigma_6 = [[1, 0.9], [0.9, 4]]$, $\Sigma_7 = [[2, 0.1], [0.1, 1.5]]$, $\Sigma_8 = [[6, 2], [2, 3]]$, $\Sigma_9 = [[1, 0.1], [0.1, 2]]$ and $\Sigma_{10} = [[1, -0.2], [-0.2, 3]]$. Each gaussian in the mixture model has its own color. The plot also shows the centers obtained by running the C-means algorithm with 12 clusters, as big black dots. The empirical covariance matrices are also calculated from the C-means algorithm, and then used to draw the black 95 % confidence ellipses.

5 Exercises (*****)

5.1 Reproduce an example similar to the bi-dimensional example, but the inicialization has to be performed with the c-means algorithm. Apply the algorithm to a real classification task.

I will now intend to apply the algorithm from the previous task to a real classification task, namely the MNIST data. This data contains 60000 images of the digits 0-9. Since one image is 28x28 pixels, the dimensions of this data is 784, which is very high. I will therefore apply PCA, to reduce the number of dimensions the EM-algorithm will work on. The idea behind using a gaussian mixture model on the MNIST data is as follows: Within each class, there could be many ways to write a number. We can expect these different ways to write a number to be somewhat clustered together, thus saying that there are for example 5 different ways to write the number 6. This way, using a gaussian mixture model can be better than for example using a single gaussian. For this to make sense, we must perform PCA within the different classes, since we want to pick up the within-class variance when we fit the gaussian mixture model. After fitting the model to the train data, there will be one reduced dimension space and one gaussian mixture model for each digit. Then the test data can be predicted by performing the dimension reduction for all the digits on the test sample, and then see how probable the test sample is according to the different gaussian mixture models. Then, the sample will be predicted to the most probable model. The code to perform PCA + EM-algorithm to analyze the digit clusters, as well as code to fit a dimension reduction space + gaussian mixture model to each digit and then predict the test data, is done in python

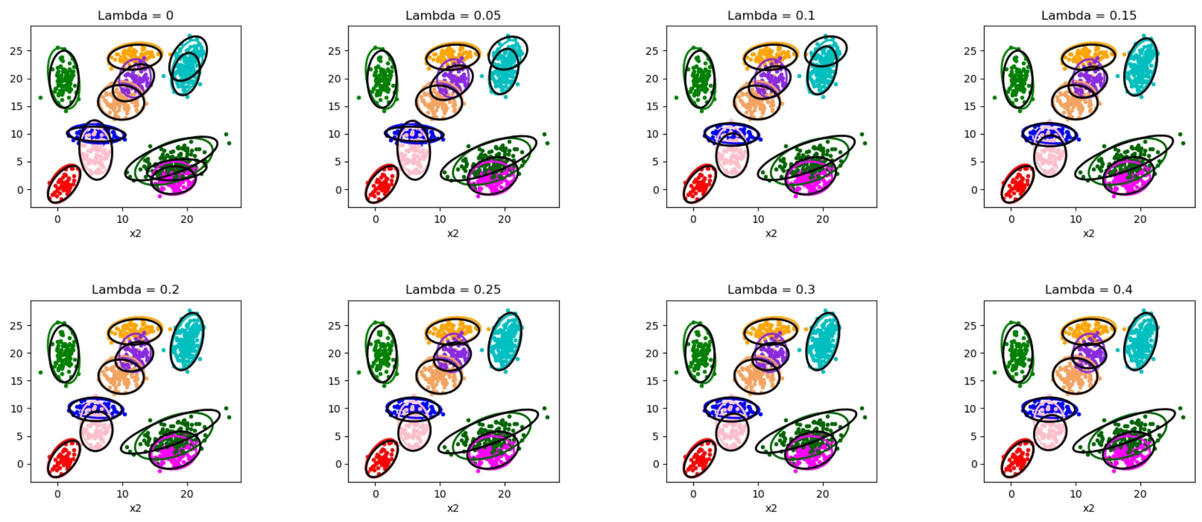


Figure 10: The training set from figure 9 with the true 95 % confidence ellipses as the colored ellipses, while the black ellipses are 95 % confidence ellipses of the models after running 200 iterations of the regularized EM-algorithm, initialized with the C-means algorithm using 15 clusters, for different values of λ.

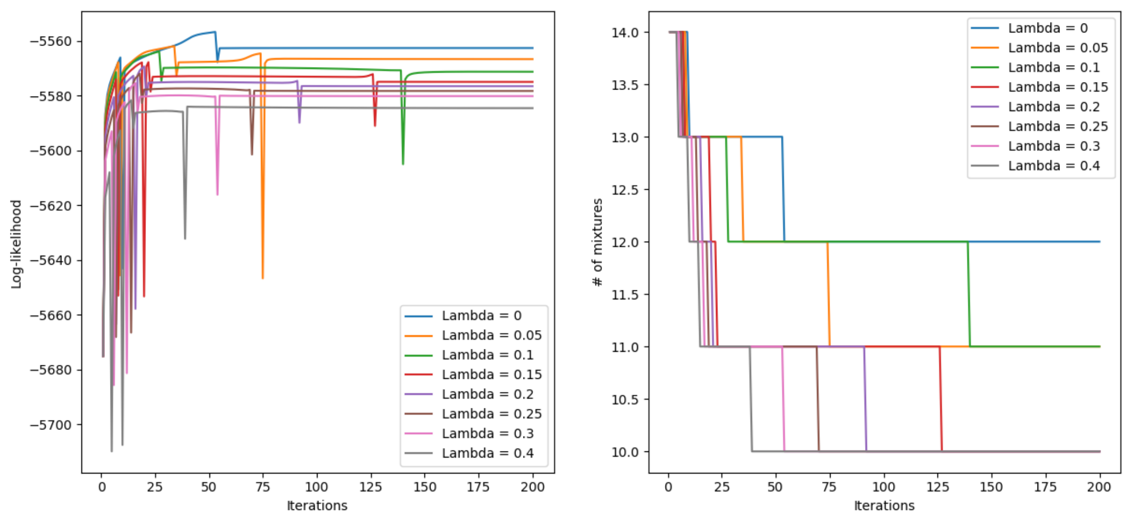


Figure 11: Log-likelihood and number of gaussians in the mixture models as a function of iterations, when running the EM-algorithm from figure 10.

and attached with this report.

Figure 12 and 13 shows the number of gaussians and the log-likelihood of the model as functions of iterations after running the C-means and regularized EM-algorithm on the digits 2 and 5, after performing PCA with 20 dimensions.

It looks like it makes sense to say that each digit has within-class clusters. More specifically, it seems like the number 2 has five different ways to write it, while the number 5 has two different ways of writing it. Note that we now use PCA with 20 dimensions. For the digit 2, this gives 67 % explained variance, while for the digit 5, it gives 69 % explained variance. 67 and 69 % is a little low, and we would like to include some more dimensions, but we then encounter some problems. Sometimes the algorithm stops working, and sometimes it just removes all gaussians but one, resulting in a singe gaussian model every time. We know that when the number of dimensions grow, the distance between points also grows, and

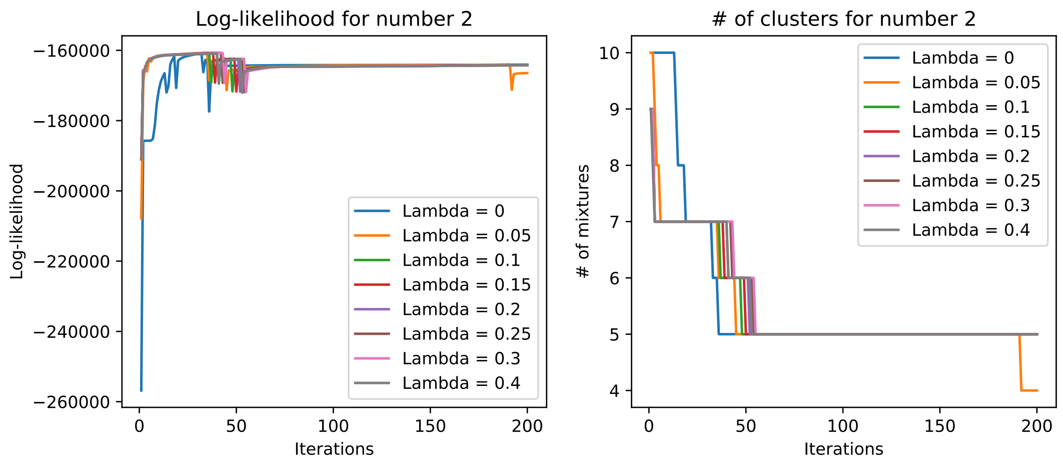


Figure 12: Log-likelihood and number of gaussians as functions of iterations from running the regularized EM-algorithm, using the C-means algorithm as initialization, on all 2-digits. PCA with 20 dimentions is performed before fitting the gaussian mixture model.

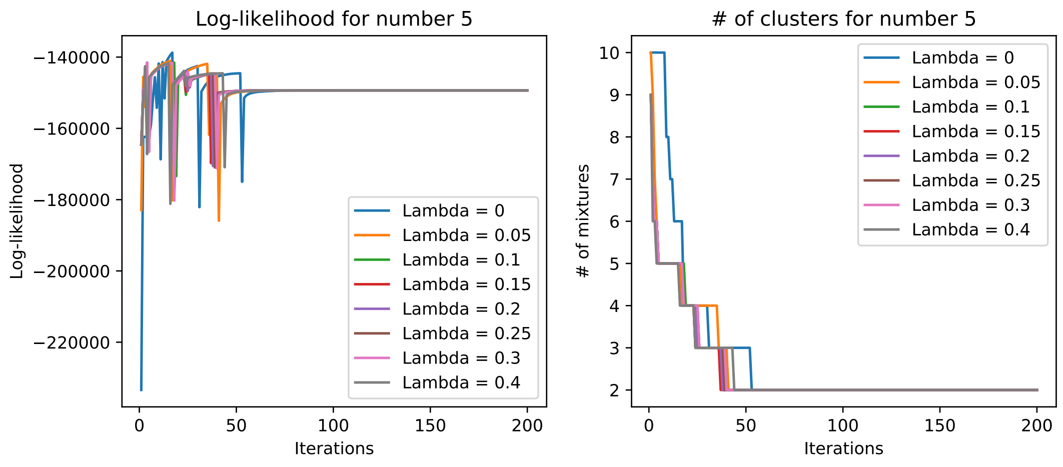


Figure 13: Log-likelihood and number of gaussians as functions of iterations from running the regularized EM-algorithm, using the C-means algorithm as initialization, on all 5-digits. PCA with 20 dimentions is performed before fitting the gaussian mixture model.

that probably results in some points being very unlikely to come from some gaussians. When this occurs, they will be removed from the algorithm. The current threshold for removing gaussians is when they become so unlikely that python would do a round-off error, resulting in log(0). Since this results error, it is not possible to lower the threshold any more. Thus, for higher dimensions, the algorithm doesn't really work, and we always end up with a single gaussian (or errors). To fix this problem, one would have to solve the problem of log (0), possibly by having a minimum value that python can handle, and finding a different criterion of removing gaussians. This has not been explored in this report.

We will now try to use this algorithm to classify the test data. A value of $\lambda = 0.2$ seems to work well with all digits, and this is therefore used when fitting the model. This gives a test error equal to 38.1 % when using 20 dimensions, which is not too bad. To compare this result, by initializing with only one center for each digit (effectively making a gaussian model for each digit, instead of a gaussian mixture model), the test error becomes 57.1 %, which is a lot worse. However, if we wanted to fit a gaussian model, it would probably be better to perform PCA on all the training data, to pick up the between-class variation instead of the within-class variation. Thus, this comparison is not really that relevant. All in all, the result is not very good, since 38.1 % is a very high error. However, we have included quite few dimensions, and then we can't really expect to get very good results.

To improve the test accuracy, we can perform PCA with more dimensions. As previously discussed, this will, because of challenges with dimensionality in the EM-algorithm, almost always give only a single gaussian for each digit when dimensions is higher than 20, thus effectively making it a gaussian model. In theory, adding more dimensions should not make the clusters go away, since we know that the dimensions we include have quite low variance (the first 20 dimensions have 67 or 69 % of the variance, and the last $\tilde{3}0\%$ are therefore spread out on the last 764 dimensions). The fact that we end up with only one gaussian therefore seems to be faults in the algorithm because of the challenge of dimensions, rather than the fact that more dimensions makes the clusters disappear. In the end, with higher dimensions, we don't really get a gaussian mixture model anymore, but it still uses our EM-algorithm to fit a gaussian model to each digit. Figure 14 shows the test error as a function of dimensions.

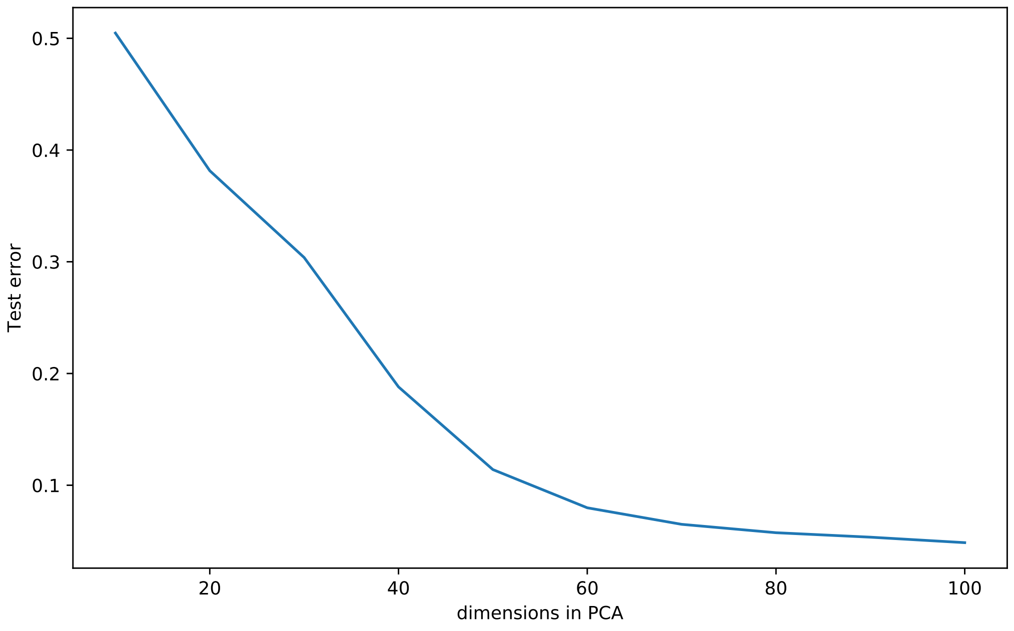


Figure 14: Test error as a function of dimensions in the PCA/dimension reduction, when using our model to predict.

We see from the plot that the test accuracy can be improved a lot by increasing the number of dimensions, thus increasing the explained variance used in the dimension reductions. With 100 dimensions, our algorithm can give an error of 4.86 %, which is very good. It is not not technically a gaussian mixture model anymore, since our algorithm has some trouble with this many dimensions, and it ends up returning a model that always have a single gaussian in the mixture model. The analysis done for fewer dimensions does however show that it makes sense to talk about clusters within each digit class, and in theory, the model could be improved by solving the dimension problems in the algorithm. To do this, I would introduce a minimum value that python could handle to remove possible errors, and then explore different criterias for removing gaussians in the regularized EM-algorithm. We do however see that even with the flaws in the algorithm I have made, it can classify the MNIST data quite well. Since the MNIST data is a very big data set with many dimensions and 10 different classes, it is not the easiest data to find a underlying structure, and it is interesting to see that a model as simple as the gaussian mixture model, trained by the reguralized EM-algorithm, can handle this kind of data.

References

[1] Sánchez, J.A. *Advanced Machine Learning: Entropy concepts used for machine learning* V1.2.

[2] *A tutorial on Clustering Algorithms: Fuzzy C-Means Clustering*
URL: home.deib.polimi.it/matteucc/Clustering/tutorial_html/cmeans.html

Appendix

Listing 1: EM-algorithm

```python
def EMAlgorithm(train, pis, mys, sigmas, iterations, x=None, unknown_mys=False,
                unknown_sigmas=False, plot=False, lambd=0):

    # Lists to store log-likelihood and number of gaussians as function of iterations
    logLikList = np.zeros(iterations)
    numberGaussiansList=np.zeros(iterations)

    #If plot: Plot model initial model
    if plot:
        y = trueDist(x, pis, mys, sigmas)
        plt.plot(x, y, '--', label='Iteration 0')

    z=np.zeros((len(pis),len(train)))

    # For each iteration
    for i in range(iterations):

        print("--- Iteration "+str(i+1)+" ---")

        #Calculates z as in formula on page 53 in class slides
        for j in range(len(pis)):
            z[j] = pis[j] * scipy.stats.multivariate_normal(mys[j], sigmas[j]).pdf(train)
        newz = z/np.sum(z, axis=0)

        # This loop controls an error that may arise. If z contains values that are too small,
        # python will have presicion error, resulting in negative infinite values. This will
        # result in NaN, and error. This mechanism removes the gaussians that have too small z,
        #  thus removing the problem. This while loop thus removes gaussians that are not
        # probable, thus allowing the algorithm to remove gaussians from the model.
        while np.log(np.min(newz))==-np.inf:

            #index of gaussian to remove
            index=np.argmin(newz)//len(newz[0])

            pis = np.delete(pis, index)
            mys = np.delete(mys, index, axis=0)
            sigmas = np.delete(sigmas, index, axis=0)
            z = np.delete(z, index, axis=0)
            newz = z/np.sum(z, axis=0)

        # Calculates log-likelihood as in page 52 in class slides, and adds numbers of
            remaining gaussians.
        logLikList[i] = np.sum(np.log(np.sum(z, axis=0)))
        #print("Log-lik: ", logLikList[i])
        numberGaussiansList[i] = len(pis)

        # Final z after removing gaussians that give error
        z=newz

        # Numerator from equation (74) in page 59 in slides
        Nk = np.sum(z * (1 + lambd * np.log(z)), axis=1)

        # Calculates new pis as in page 59 in slides. lambd=0 gives pis as in page 53 in slides
        pis = Nk / np.sum(Nk)
        #print("Pis: ", pis)

        # Calculates mys as in page 59 in slides
        if unknown_mys:
            mys = np.dot(z * (1 + lambd * np.log(z)), train) / Nk[:, np.newaxis]
            #print("Mys: ", mys)

        # Calculates sigmas as in page 59 in slides
        if unknown_sigmas:
            for j in range(len(pis)):
                sigmas[j] = np.dot(np.transpose(train-mys[j])*z[j] * (1 + lambd *
                    np.log(z[j])),train-mys[j])/Nk[j]
            #print("Sigmas: ", sigmas)

        # If plot: Plots model after each iteration
        if plot:
            y=trueDist(x, pis, mys, sigmas)
            plt.plot(x,y,'--', label='Iteration '+str(i+1))

    return pis, mys, sigmas, logLikList, numberGaussiansList
```

Listing 2: C-means algorithm

```
def cMeans(train, c, dim, rangeMin, rangeMax, iterations):

    # Random initial centers
    centers=np.random.uniform(rangeMin, rangeMax, size=(c,dim))
    initialU=np.zeros((c, len(train)))

    #Parameter to measure ||newU-oldU||, as stopping criterion
    diff=1

    # For each iteration step, until ||oldU-newU|| is smaller than threshold
    while diff>1e-3:

        oldU = initialU.copy()
        # Calculates U based on current centers (see report for formula)
        for j in range(c):
            initialU[j]=np.sum((train-centers[j])**2, axis=1)
        initialU = 1 / (initialU * np.sum(1 / initialU, axis=0))

        # Set diff to ||oldU-newU||, to be used as stopping criterion
        diff = np.linalg.norm(oldU-initialU, ord=2)

        # Calculates new centers based on U (see report for formula)
        centers = np.matmul(initialU ** 2, train) / (np.sum(initialU**2, axis=1)).reshape(c, 1)

    # Estimates pis based on U
    pis = np.sum(initialU, axis=1)/len(train)

    # Estimates sigmas based on U
    sigmas = np.zeros(((c, dim, dim)))
    numberList = np.zeros(c)
    uTranspose = np.transpose(initialU)

    #For each training sample
    for i in range(len(train)):

        # Choose most probable center, calculate (x-mu)(x-mu)^T
        index = np.argmax(uTranspose[i])
        distance = np.asarray(train[i] - centers[index])
        numberList[index] += 1
        sigmas[index] += np.outer(distance, distance)

```

```python
    # Divide by numbers belonging to each center
    for i in range(c):
        sigmas[i] /= numberList[i]


    return centers, pis, sigmas
```