SLML Project_4 PCA & GMM

Shun Zhang, student ID:15300180012

December 29, 2017

1 Dimensionality Reduction

1.1 Freyface

Here our database is X that contains 1965 images of Brendan Frey's face and first let's take a look at some of Frey's 'faces'.

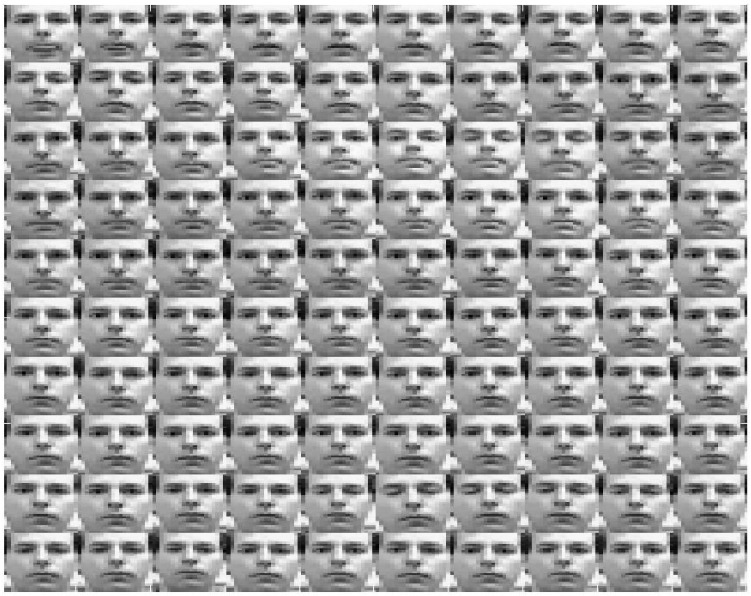


Figure 1: 100 different face images of Frey

1.2 Principal Components Analysis

In this part, we would like to introduce the most commonly used method for dimensionality reduction in practice.

Suppose we have a data matrix X of size (D, N), where D is the dimension of our original input data and N is the number of data points. That is to say, every column of X is a sample.

Recall the procedure of PCA:

Table 1: main procedure of PCA

Data	$X = (\boldsymbol{x_1}, \boldsymbol{x_2}, \cdots, \boldsymbol{x_N})$	
steps		
(1)	centralize all the data points	$\boldsymbol{x_{i,ctr}} \leftarrow \boldsymbol{x_i} - \frac{1}{N}\sum_{j=1}^{N} \boldsymbol{x_j},\ \forall i$ $X_{ctr} \leftarrow (\boldsymbol{x_{1,ctr}}, \boldsymbol{x_{2,ctr}}, \cdots, \boldsymbol{x_{N,ctr}})$
(2)	derive the covariance matrix of X_{ctr}	$\Sigma_{ctr} \leftarrow X_{ctr} X_{ctr}^T$
(3)	apply eigenvalue decomposition to Σ_{ctr}	$\Sigma_{ctr} * U = U * D_\lambda$ $D_\lambda = diag\{\lambda_1, \lambda_2, \cdots, \lambda_N\}$ $U = (\boldsymbol{v_1}, \boldsymbol{v_2}, \cdots, \boldsymbol{v_N})$
(4)	order the eigenvalues update the eigenvectors' order	$D_\lambda \leftarrow diag\{\lambda_{(1)}, \lambda_{(2)}, \cdots, \lambda_{(N)}\}$ $U \leftarrow (\boldsymbol{v_{(1)}}, \boldsymbol{v_{(2)}}, \cdots, \boldsymbol{v_{(N)}})$
Finally	choose the biggest k eigenvalues also with the eigenvectors	$\{\lambda_{(1)}, \lambda_{(2)}, \cdots, \lambda_{(k)}\}$ $\{\boldsymbol{v_{(1)}}, \boldsymbol{v_{(2)}}, \cdots, \boldsymbol{v_{(k)}}\}$

Then the matrix $W^* = (\boldsymbol{v_{(1)}}, \boldsymbol{v_{(2)}}, \cdots, \boldsymbol{v_{(k)}})$ is called the *projection matrix*.

And an important point is that our data must be centralized and note that the eigenvectors are all normalized.

So among the codes below, only those centralized ones, such as $Vctr, lambda\_ctr$, are required in PCA, where $lambda\_ctr$ is the ordered eigenvalues, increasingly, and $Vctr$ is the corresponding eigenvectors.

```
load freyface.mat
X = double(X);
N = size(X,2);
[Vun, Dun] = eig(X*X'/N);
[lambda_un, order] = sort(diag(Dun));
Vun = Vun(:, order);
Xctr = X - repmat(mean(X, 2), 1, N);
[Vctr, Dctr] = eig(Xctr*Xctr'/N);
[lambda_ctr, order] = sort(diag(Dctr));
```

```
Vctr = Vctr(:, order);
```

1.2.1 Choose k for PCA

Here we introduce the concept of *the percentage of variance retained*, which is that if we retain k principal components then the percentage of variance retained is given by

$$pvr(k) := \frac{\sum_{i=1}^{k} \lambda_{(i)}}{\sum_{j=1}^{N} \lambda_j}$$

In the application of image process, the threshold is recommended to be over 99%, so here we could derive the smallest k such that $pvr(k) \geq 0.99$, which turns out to be $k = 203$. (see Figure 2)

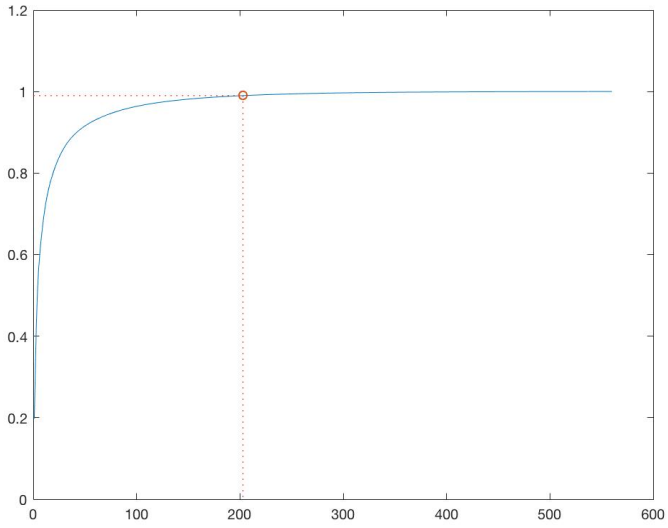


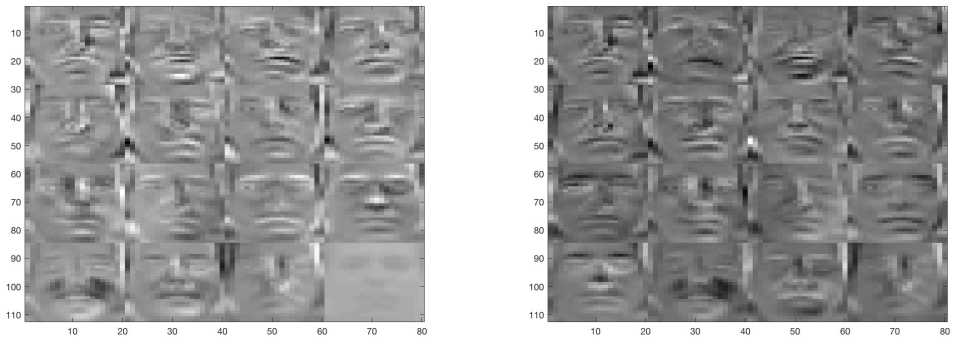
Figure 2: $pvr(k)$ according to different k

Of cause, you can derive different k according to different demand of the threshold of *the percentage of variance retained*.

1.2.2 Top 16 eigenvectors in each case

Visualization see Figure 3

Figure 3: top 16 eigenvectors in each case



un-centralized case centralized case

From Figure 3, the 16 faces in each case can be interpreted as the main facial expressions of Frey.

- One apparent observation is that the eigenvectors of centralized case are more dark than those of un-centralized case. More importantly, the centralized case shows more comparative images. That is to say, the face in the centralized case is more identifiable.

  The reason why centralized version is more 'dark' may be that after the pre-process of centralization, the eigenvectors will be distributed near the axises and thus, the elements within a certain eigenvector may be either negative or positive. While for the un-centralized version, signs of the elements within a certain eigenvector may be more likely the same.

  The second reason may be that without centralization, the missing information after doing such dimension reduction will be larger.

- Another observation is that the orders of the faces are little different from each case. This may also related to the projection process after centralization.

1.2.3 Project to top two dimension

First we consider the case that $V = Vctr$, which is exactly consistent to PCA's process.

Figure 6 is corresponding to the areas in Figure 4, except for the Red box, which is the main samples of Frey's front face. Note that the 'partition' of Figure 4 is not exact, it is just an empirical result, from which we can see the power of PCA.

Then we consider the case where $V = Vun$.

Figure 6 also corresponds to the areas, with the same notation, in Figure 5, except for the Red box, which is the main samples of Frey's front face.

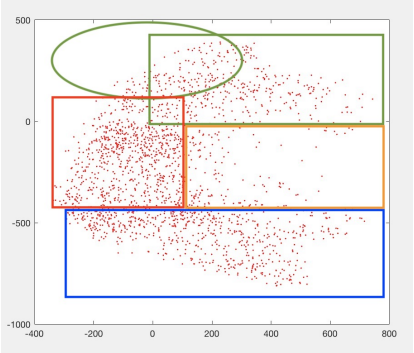


Figure 4: centralized Frey manifold(2D)

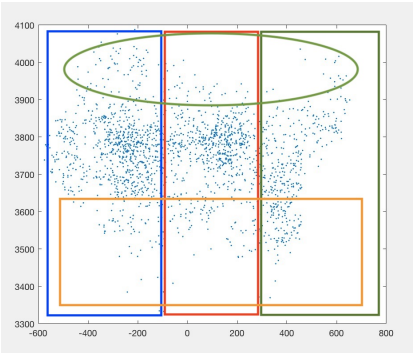


Figure 5: un-centralized Frey manifold(2D)

Green box

Blue box

Orange box

Green circle

Figure 6: different faces in different areas in the manifold

- The first difference is that the axises scale of the two figures are different. Apparently, samples of centralized version all gather near 0.
- The second difference is that the 'partition' (in Figure 4,5) of each case are different. That is to say, the area holds for a certain kind of face is different.

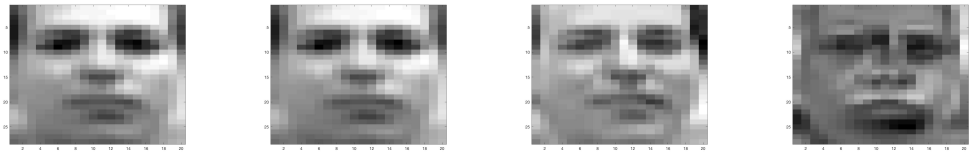
1.2.4 Reconstructing a face from an arbitrary point

Here, we interpret 'this space' (in prof's PDF) as the space of 2-dimension. So, the reconstructing process should be:

the projection matrix:	$W := Vctr(:, end-1:end)$
the mean:	$X_m := repmat(mean(X, 2), 1, 1)$
step 1:	$y \leftarrow \sigma * randn(2, 1)$
step 2:	$x_r^* \leftarrow W * y$
step 3:	$x_r \leftarrow x_r^* + X_m$

Note that here we consider the case of centralized version (the un-centralized case is similar) and the $\boldsymbol{\sigma}$ in step 1 is the standard deviation of the normal distribution. Then we

have several reconstructed faces and we want to know how the face changes as σ increases. (see Figure 7)



$\sigma = 1$ $\sigma = 50$ $\sigma = 500$ $\sigma = 2000$

Figure 7: several reconstructed faces

Recall Figure three, we find that the faces we reconstruct here are similar to the top two eigenvectors to a certain extent, especially the first three faces that are quite reasonable. What's more, we can see that as the σ increases, which means the randomness rises, the face we reconstruct become more vague. The reason may be that with large σ, the point we randomly generate is far from those samples in Figure 4, thus the reconstructed face 'get's' less information. And an extreme example is the case with $\sigma = 2000$ where the face is not identifiable.

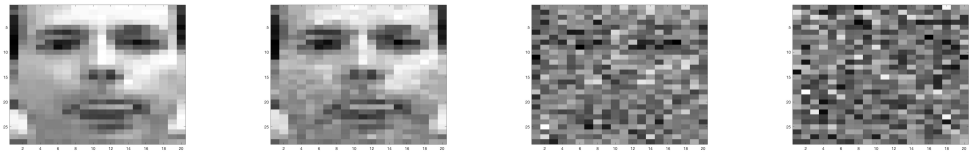
1.2.5 Adding a noise

Here we still consider the case of centralized version (the un-centralized case is similar). And the procedure should be:

the projection matrix:	$W := Vctr(:, end-1:end)$
the mean:	$X_m := repmat(mean(X, 2), 1, 1)$
choose a face:	X(:,1) for example
step 1:	$x^* \leftarrow X(:,1) + \sigma * randn(560, 1)$
step 2:	$y \leftarrow W^T * x^*$
step 3:	$x_r^* \leftarrow W * y$
step 4:	$x_r \leftarrow x_r^* + X_m$

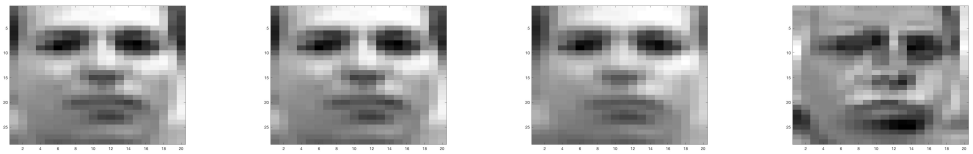
Similarly, we have a σ here. Then we have several reconstructed faces after adding a noise before projection and we want to know how the face changes as σ increases.

From the faces in Figure 8 & 9, we can see that even when the face after adding a noise doesn't make any sense, applying PCA and then reconstruction still do a great job. However, as the σ increases, the reconstructed faces aren't so well.



$\sigma = 1$ $\sigma = 10$ $\sigma = 100$ $\sigma = 1000$

Figure 8: after adding a noise

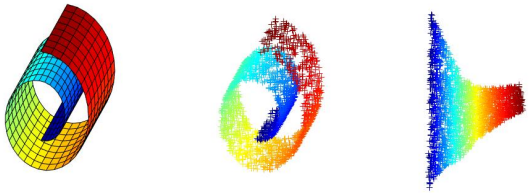


$\sigma = 1$ $\sigma = 10$ $\sigma = 100$ $\sigma = 1000$

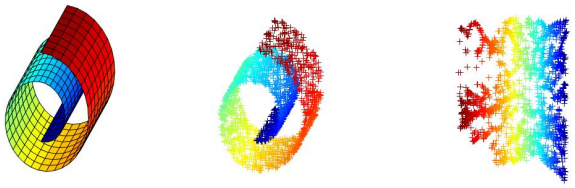
Figure 9: adding a noise $\rightarrow$ projection $\rightarrow$ reconstruction

1.3 Swiss roll

The result of LLE algorithm:



Another popular algorithm for manifold learning is *Isomap*, and the result of it:



2 Gaussian Mixture Model

2.1 EM algorithm for GMM

Here our goal is to deal with the Gaussian mixture model

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

Though it's easy to see 'mixture of Gaussian models', it is really confusing by just calling eq.(1) GMM and let's find out why.

First, we have K different normal distributions with different means $\{\boldsymbol{\mu_k}\}$ and covariance $\{\boldsymbol{\Sigma_k}\}$.

Then we have the likelihood function

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

which is just the same as eq.(1)

Here we encounter a new coefficient: π_k. To explain this, we should 'invite' some latent variables

$$\forall 1 \leq k \leq K,\ z_k \in \{0, 1\} \quad \text{and} \quad \sum_{k=1}^{K} \pi_k = 1$$

so apparently, there are only one of them is non-zero, which is called '1-of-K' encoding. And after including this kind of encoding, we have

$$p(\mathbf{x} \mid z_l = 1) = \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu_l}, \boldsymbol{\Sigma_l}) = \prod_{k=1}^{K} \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu_k}, \boldsymbol{\Sigma_k})^{z_k} = p(\mathbf{x} \mid \mathbf{z}) \tag{2}$$

Since we've got the conditional probability, what about the priors?

Here we denote the priors as $\{\pi_k\}$, which is

$$p(z_k = 1) := \pi_k = \prod_{k=1}^{K} \pi_k^{z_k} = p(\mathbf{z})$$

And then we have the joint distribution

$$p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z}) = \prod_{k=1}^{K} \pi_k^{z_k} \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu_k}, \boldsymbol{\Sigma_k})^{z_k}$$

Everything seems good but what if we fail to know the true value of the priors? Or means? Or covariances? And here lies the essence of EM algorithm.

Recall what we do when dealing with marginal distrubutions

$$p(\mathbf{x}) = \sum_{\mathbf{z}} p(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z}) = \sum_{k=1}^{K} p(\mathbf{x} \mid z_k = 1) p(z_k = 1) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu_k}, \boldsymbol{\Sigma_k})$$

which is just the same as eq.(1).

Then for data matrix $X = (\boldsymbol{x_1}, \boldsymbol{x_2}, \cdots, \boldsymbol{x_N})$, the probability is

$$p(X) = \prod_{n=1}^{N} p(\boldsymbol{x_n})$$

and the log-likelihood of it is

$$\begin{aligned} \ln p(\boldsymbol{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) &= \sum_{n=1}^{N} \ln p(\boldsymbol{x_n}) \\ &= \sum_{n=1}^{N} \ln \Big\{ \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x_n} \mid \boldsymbol{\mu_k}, \boldsymbol{\Sigma_k}) \Big\} \end{aligned} \tag{3}$$

Before we move on, another notation should be included, which is

$$\gamma_{nk} := p(z_k = 1 \mid \boldsymbol{x_n}) = \frac{p(\boldsymbol{x_n} \mid z_k = 1) p(z_k = 1)}{p(\boldsymbol{x_n})} = \frac{\pi_k \mathcal{N}(\boldsymbol{x_n} \mid \boldsymbol{\mu_k}, \boldsymbol{\Sigma_k})}{\sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x_n} \mid \boldsymbol{\mu_k}, \boldsymbol{\Sigma_k})} \tag{4}$$

here γ_{nk} could be interpreted as the responsibility of the k-th component to data $\boldsymbol{x_n}$.

Well, to maximize the log-likelihood function (eq.(3)) is not so easy as the single Gaussian model, which can be achieved by a close form of derivatives.

Then appears the EM algorithm to solve such problem, partly. ('cause EM do not guarantee a global maximum)

Here we also derive the MLE of those parameters assuming that all the variances are the same $\boldsymbol{\Sigma_k} = \Sigma, \forall k$.

$$\begin{aligned} \frac{\partial \ln p(\boldsymbol{x} \mid \boldsymbol{\mu}, \Sigma)}{\partial \boldsymbol{\mu_k}} &= \sum_{n=1}^{N} \left\{ \frac{\pi_k \frac{\partial}{\partial \boldsymbol{\mu_k}} \mathcal{N}(\boldsymbol{x_n} \mid \boldsymbol{\mu_k}, \Sigma)}{\sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x_n} \mid \boldsymbol{\mu_k}, \Sigma)} \right\} \\ &= \sum_{n=1}^{N} \left\{ \frac{\pi_k \mathcal{N}(\boldsymbol{x_n} \mid \boldsymbol{\mu_k}, \Sigma)}{\sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x_n} \mid \boldsymbol{\mu_k}, \Sigma)} \big(-\Sigma^{-1}(\boldsymbol{x_n} - \boldsymbol{\mu_k}) \big) \right\} = 0 \\ \Rightarrow \sum_{n=1}^{N} \Big\{ \gamma_{nk} (\boldsymbol{x_n} - \boldsymbol{\mu_k}) \Big\} &= 0 \\ \Rightarrow \boldsymbol{\mu_k} &= \frac{\sum_{n=1}^{N} \gamma_{nk} \boldsymbol{x_n}}{\sum_{n=1}^{N} \gamma_{nk}} \end{aligned} \tag{5}$$

Similarly, we have the MLE for covariance

$$\begin{aligned}
\frac{\partial \ln\, p(\boldsymbol{x} \mid \boldsymbol{\mu}, \Sigma)}{\partial \Sigma} &= \sum_{n=1}^{N} \frac{\sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x_n} \mid \boldsymbol{\mu_k}, \Sigma) \frac{1}{|\Sigma|^{1/2}} \big[(\boldsymbol{x_n} - \boldsymbol{\mu_k})^T \Sigma^{-1} (\boldsymbol{x_n} - \boldsymbol{\mu_k}) - 1\big]}{\sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x_n} \mid \boldsymbol{\mu_k}, \Sigma)} \\
&= \sum_{n=1}^{N} \sum_{k=1}^{K} \gamma_{nk} \frac{1}{|\Sigma|^{1/2}} \big[(\boldsymbol{x_n} - \boldsymbol{\mu_k})^T \Sigma^{-1} (\boldsymbol{x_n} - \boldsymbol{\mu_k}) - 1\big] = 0 \\
\Rightarrow \sum_{n=1}^{N} \sum_{k=1}^{K} \gamma_{nk} (\boldsymbol{x_n} - \boldsymbol{\mu_k})^T \Sigma^{-1} (\boldsymbol{x_n} - \boldsymbol{\mu_k}) &= \sum_{n=1}^{N} \sum_{k=1}^{K} \gamma_{nk} \\
\Rightarrow \Sigma = \frac{\sum_{n=1}^{N} \sum_{k=1}^{K} \gamma_{nk} (\boldsymbol{x_n} - \boldsymbol{\mu_k})(\boldsymbol{x_n} - \boldsymbol{\mu_k})^T}{\sum_{n=1}^{N} \sum_{k=1}^{K} \gamma_{nk}} &\qquad (6)
\end{aligned}$$

Finally, we maximize eq.(3) using coefficients $\{\pi_k\}$ with constraint of $\sum_{k=1}^{K} \pi_k = 1$ by Lagrange multiplier.

$$\begin{aligned}
\mathcal{L}(\lambda) &:= \ln\, p(\boldsymbol{x} \mid \boldsymbol{\mu}, \Sigma) + \lambda \Big(\sum_{k=1}^{K} \pi_k - 1 \Big) \\
&\Rightarrow \frac{\partial\, \mathcal{L}(\lambda)}{\partial \pi_k} = 0 = \sum_{n=1}^{N} \frac{\pi_k \mathcal{N}(\boldsymbol{x_n} \mid \boldsymbol{\mu_k}, \Sigma)}{\sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x_n} \mid \boldsymbol{\mu_k}, \Sigma)} + \lambda \\
&\Rightarrow \lambda = -N \\
&\Rightarrow \pi_k = \frac{\sum_{n=1}^{N} \gamma_{nk}}{N} \qquad (7)
\end{aligned}$$

So, till here, we have the three main MLEs, eq.(5,6,7).
And here we go with the **EM algorithm**:

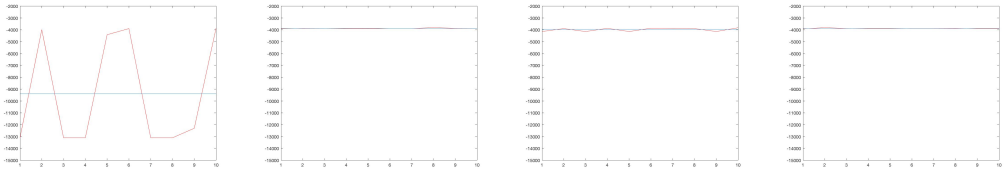
- **Initialize** : apply *k-means* to initialize the parameters $\{\boldsymbol{\mu_k}\}$, $\{\Sigma\}$ & $\{\pi_k\}$
- **E-step** : derive the responsibilities $\{\gamma_{nk}\}$ according to eq.(4)
- **M-step** : re-estimate the parameters according to eq.(5,6,7)
- **Repeat** : repeat the E-step & M-step until the change in parameters goes below a certain threshold

2.2 Training

So, again we are going to deal with the classification of hand-written digits.

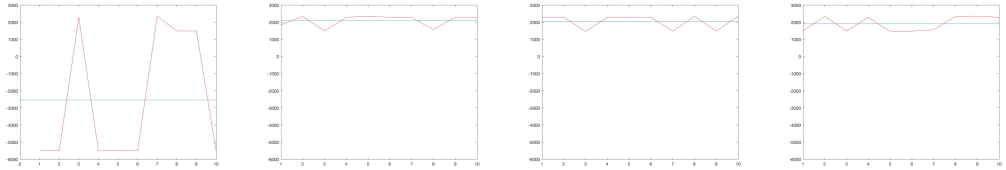
Here we train a mixture-of-Gaussions with 2 clusters and minimum variance 0.01. Since we will initialize the parameters with a *randConst*, we then execute *mogEM_rconst* a few

Figure 10: local optimas of different *randConst* for *digit 2*



randConst = 0.1 *randConst* = 1 *randConst* = 5 *randConst* = 10

Figure 11: local optimas of different *randConst* for *digit 3*

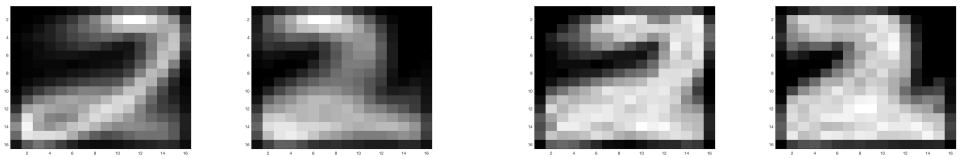


randConst = 0.1 *randConst* = 1 *randConst* = 5 *randConst* = 10

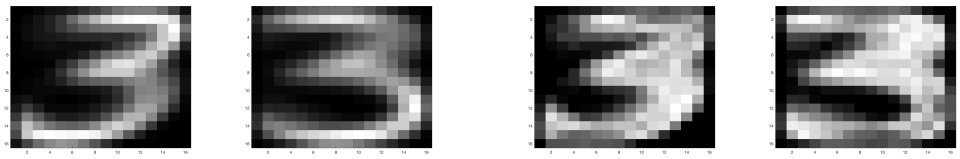
times for each digit and see the local optima the EM algorithm ends.
From Figure 10 & 11, we can see that $randConst = 1, 5, 10$ almost perform the same for either *digit 2* or *digit 3*. (**Note that** the blue line shows the mean of each plot and the scales of y-axis are the same for each digit.)
So, we simply make the choice that $randConst = 1$ both for the two digits.

Then we move on to show the results based on $randConst = 1$.

Figure 12: *means* and *variances* for each model



means of digit 2 *variances* of digit 2



means of digit 3 *variances* of digit 3

And the mixing proportions for the clusters are:

Mixing proportions	cluster 1	cluster 2
train2	0.5333	0.4667
train3	0.5005	0.4995

The *final log-probability* of training data for each model:

Model	train2	train3
log-probability	-3891	2289

2.3 Initializing a Mixture of Gaussian with K-means

In this part, we change the initialization of the means in *mogEM_kmeans.m* by k-means algorithm with 5 iterations, which enables our model to speed up.

Then we move on to compare a MoG model with 20 components on all 600 training vectors (both 2's and 3's) using the original initialization and the one based on k-means.

Here we compare the speed of different initializations under the same iterations. (30, for example)

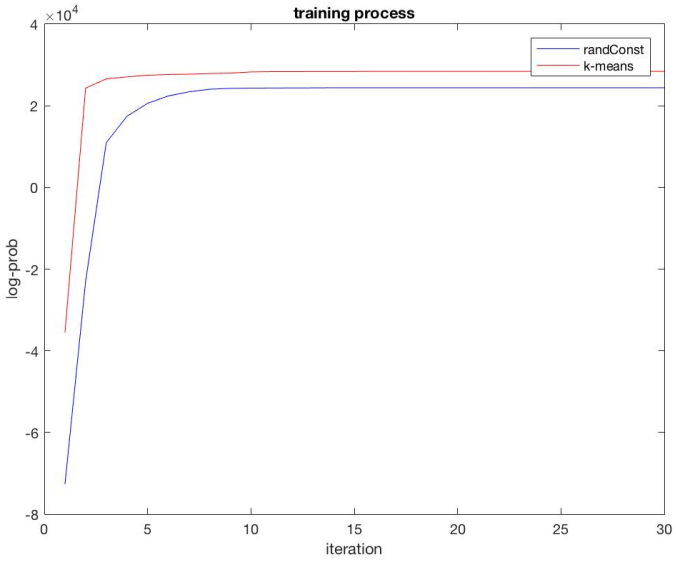


Figure 13: training process for two different initializations

From Figure 13, we can see that the final *log-prob* of *k-means* initialization is better than that of *randConst*. But for the speed of convergence, it may be hard to draw a definite conclusion from Figure 13 alone.
Let's take a look at Figure 14, in which we train digit 2 and 3 separately with 2 clusters, 30 iterations.

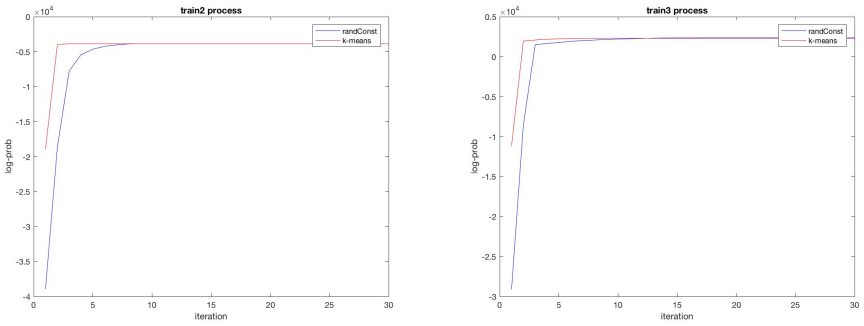


Figure 14: training process of two different initializations

And now we can say that the speed of convergence of *k-means* initialization is faster than that of *randConst* initialization.

2.4 Classification with MoGs

Now we will investigate using the trained mixture models for classification. The goal is to decide which digit class d a new input image x belongs to. We'll assign $d = 1$ to the 2's and $d = 2$ to the 3's.

For each mixture, after training, the likelihoods $P(x \mid d)$ for each class can be computed for an image x by consulting the model trained on examples from that class; probabilistic inference can be used to compute $P(d \mid x)$, and the most probable digit class can be chosen to classify the image.

And we are going to compare models trained with the same number of mixture components for $\{2, 5, 15, 25\}$.

From Figure 15, we can derive the following statements:

- The error rates on the training sets generally decrease as the number of clusters increases because as the number of clusters increases, the complexity of the model grows, thus leading to a better perform on training set. Consider the extreme case of the number of clusters **equals** the number of training samples, then it is easy to gain a zero error rate on training set by matching every sample a cluster. So, this kind of error decreasing shows a sign of *overfitting*.

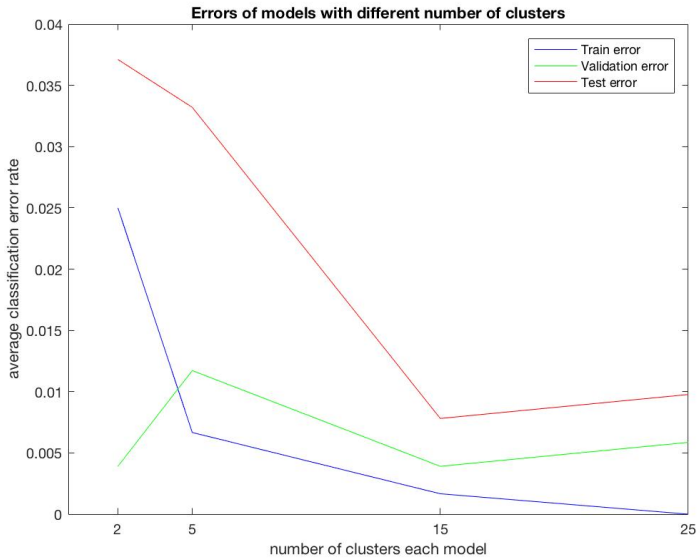


Figure 15: classification error rate on each model

- And from the error rate curve for the test set we can see that with the increasing of number of clusters, there do exist the problem of *overfitting*, which causes the increase on test error after some decrease.

- If we want to choose a particular model from the experiments, we would choose the model whose classification error rate on validation set is the lowest. From Figure 9, we can see that the model with 15 clusters performs best on validation set.

  If our aim is to achieve the lowest error rate possible on the new images, we will still choose the model whose validation error is the lowest. Recall the principle of Machine Learning, 'one should never choose a model according to the test performance'. So we keep the choice of 15 clusters.

Remark: Here, we set the random seed as $rng(10)$.