Exercises
Computational Intelligence Lab
SS 2020

Machine Learning Institute
Dept. of Computer Science, ETH Zürich
Prof. Dr. Thomas Hofmann
Web http://da.inf.ethz.ch/cil

Series 7, Solutions (K-means and Mixture Models)

1 The K-means Algorithm

Problem 1 (Theory):

1. (Convergence of the K-Means Algorithm) The K-means algorithm converges since at each iteration it either reduces or keeps the same the value of the objective function J, where

$$J = \sum_{n=1}^{N}\sum_{k=1}^{K} z_{k,n}\|\mathbf{x}_n - \mathbf{u}_k\|_2^2 \quad \left(\|\mathbf{x}_n - \mathbf{u}_k\|_2^2 = (x_{1,n} - u_{1,k})^2 + \cdots + (x_{d,n} - u_{d,k})^2\right)$$

with the constraint

$$\sum_{k=1}^{K} z_{k,n} = 1 \quad \text{and} \quad z_{k,n} \in \{0,1\}.$$

When initializing the algorithm, at step 2 of the K-means algorithm we set

$$z_{k^*(\mathbf{x}_n),n} = 1 \quad \text{and} \quad z_{k',n} = 0,$$

where

$$k^*(\mathbf{x}_n) = \operatorname*{argmin}_k \left\{\|\mathbf{x}_n - \mathbf{u}_1\|_2^2, \dots, \|\mathbf{x}_n - \mathbf{u}_k\|_2^2, \dots, \|\mathbf{x}_n - \mathbf{u}_K\|_2^2\right\}.$$

This makes the value of J minimal considering that we have to assign the value 1 to one and only one $z_{k,n}$, and 0 to all others.

At step 3, the centroid update term you are familiar with:

$$\mathbf{u}_k = \frac{\sum_{n=1}^{N} z_{k,n}\mathbf{x}_n}{\sum_{n=1}^{N} z_{k,n}} \;\forall k,\; k = 1, \dots, K$$

means that

$$0 = \sum_{n=1}^{N} z_{k,n}(\mathbf{x}_n - \mathbf{u}_k) \;\forall k,\; k = 1, \dots, K$$

Note that this equals setting the derivative of J with respect to $\mathbf{u}_k$ to zero for all k, $k = 1, \dots, K$, as a partiuclar derivative is given by:

$$\frac{\partial J}{\partial \mathbf{u}_k} = \frac{\partial \sum_{n=1}^{N} z_{k,n}\|\mathbf{x}_n - \mathbf{u}_k\|_2^2}{\partial \mathbf{u}_k} = \sum_{n=1}^{N} z_{k,n}\begin{bmatrix} \frac{\partial(x_{1,n}-u_{1,k})^2}{\partial u_{1,k}} \\ \vdots \\ \frac{\partial(x_{d,n}-u_{d,k})^2}{\partial u_{d,k}} \end{bmatrix} = -2\sum_{n=1}^{N} z_{k,n}(\mathbf{x}_n - \mathbf{u}_k)$$

Note that $\frac{\partial^2 J}{\partial \mathbf{u}_k^2} \geq 0$, or in other words, J is convex with respect to $\mathbf{u}_k$, which indicates that this is a minimizer. Thus, the value of J does not increase after the centroid update. Considering all the above, it follows that repeating steps 2 and 3 in iterations means that the value of J will converge.

2. (The K-Means Algorithm and Matrix Factorization) Notice that

$$\begin{aligned}
\|X - UZ\|_F^2 &= \sum_{i=1}^{D}\sum_{j=1}^{N}(x_{i,j} - \sum_{k=1}^{K} u_{i,k} z_{k,j})^2 && \text{(expand matrix norm)} && (1)\\
&= \sum_{i=1}^{D}\sum_{j=1}^{N}\left(\sum_{k=1}^{K} z_{k,j}(x_{i,j} - u_{i,k})\right)^2 && \text{(push } x_{i,j} \text{ inside summation)} && (2)\\
&= \sum_{i=1}^{D}\sum_{j=1}^{N}\sum_{k=1}^{K} z_{k,j}^2 (x_{i,j} - u_{i,k})^2 && \text{(evaluate squared term)} && (3)\\
&= \sum_{k=1}^{K}\sum_{j=1}^{N} z_{k,j}^2 \sum_{i=1}^{D}(x_{i,j} - u_{i,k})^2 && \text{(reorder summations)} && (4)
\end{aligned}$$

hence

$$\|X - UZ\|_F^2 = \sum_{k=1}^{K}\sum_{j=1}^{N} z_{k,j}\|\mathbf{x}_j - \mathbf{u}_k\|_2^2.$$

For this derivation, we exploited two key properties of the problem. (i) For any data point j, $z_{k,j}$ is 1 only for one k and 0 elsewhere, which allows us to push $x_{i,j}$ inside the summation in line 2 without affecting the value of the objective function. Similarly, in line 3, the evaluation of the squared term does not result in any interaction terms (the summation can simply be rewritten as-is). (ii) $z_{k,j}^2 = z_{k,j}$ since $z_{k,j}$ is either 0 or 1. In the final step we simply rewrite the summation over D as a norm.

3. (Termination) Since K-Means uses hard cluster assignments, there is a finite number of possible cluster assignments (even though this is exponential in the number of data points). This entails that the algorithm must enter a cycle at some point. This cycle has length 1 because the value of the objective function J cannot increase between subsequent iterations (recall from exercise 1). In practice, full convergence may require a large number of steps. The typical workaround consists in setting a maximum number of iterations or stopping when the difference of J between subsequent iterations is lower than a threshold ϵ.

Problem 2 (Practical exercise):

The assignments of the first iteration are already provided. The corresponding cluster centers are $c_1 = 5$ and $c_2 = 2$. The next assignments are $C_1 = \{-3, -2, 1\}$ and $C_2 = \{4, 5, 6, 8, 9\}$, which correspond to the centers $c_1 = -4/3$ and $c_2 = 6.4$. The following iteration results in the same assignments, which means that the algorithm has converged.

Problem 3 (Implementation):

The code is provided in the notebook `kmeans.ipynb`.

1. (Image size) The image can be represented as a $W \times H \times 3$ tensor. Each channel is 8 bits long, i.e. 24 bits per pixel. The uncompressed size (excluding metadata) is $465 \times 606 \times 24 =$ 6,762,960 bits.

2. (Size reduction) Figure 1 shows the compression result for $k = 4$. For a general k, the number of bits per pixels to encode an assignment is $\log_2 k$, that is, 2 bits per pixel for $k = 4$, 4 bits for $k = 16$, and 6 bits for $k = 64$, corresponding to a size reduction of a factor (respectively) 12, 6, and 4.

3. (Empty clusters) This can happen due to poor initialization (as the algorithm can get stuck in bad local minima), or if the number of clusters is too large (consider the extreme case of having more clusters than data points). In practice, most implementations tackle this issue by assigning empty clusters to a random point, or by removing them entirely (ending up with a lower k when the algorithm terminates).

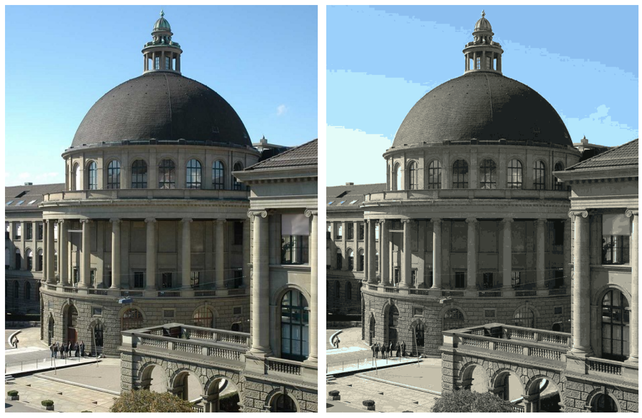
4. (Coding) The Shannon entropy of a probability distribution is given by

$$H = -\sum_{k=1}^{K} p_k \cdot \log_2 p_k$$

Here we use the base 2 since we are encoding bits (binary data). The actual result depends on the image and may vary across different runs of the algorithm. The results that we observed using our sample implementation are summarized in Table 1. The resulting distribution of colors is relatively uniform and optimal coding cannot compress the data further, which shows why K-means is useful for data compression. Coding schemes that exploit neighborhood (e.g. run-length encoding) can compress the data further.

k	Uncompressed	Binary coded	Optimally coded
4	24 bits/pixel	2 bits/pixel	1.966 bits/pixel
16	24 bits/pixel	4 bits/pixel	3.871 bits/pixel
64	24 bits/pixel	6 bits/pixel	5.757 bits/pixel

Table 1: Bits per pixels for different values of k.



(a) Original (b) Compressed with $k = 16$

Figure 1: `eth.jpg`

2 Mixture Models

Problem 1 (EM Algorithm):

In this exercise, we derive the two steps of the Expectation Maximization algorithm.

1. The log-likelihood of the data is given by

$$\ln p(\mathbf{X} \mid \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^{N} \ln \left\{ \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\}.$$

2. Any distribution, γ, can be used to obtain a lower bound of the log-likelihood. For sake of simplicity let's define $p_{\theta_k}(\mathbf{x_n}) = p(\mathbf{x}; \theta_k) = \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$, then:

$$\begin{aligned} \log p(\mathbf{X}; \theta_k) = \sum_{n=1}^{N} \log \left[\sum_{k=1}^{K} \pi_k p_{\theta_k}(\mathbf{x}_n) \right] &= \sum_{n=1}^{N} \log \left[\sum_{k=1}^{K} \gamma_{nk} \frac{\pi_k \, p_{\theta_k}(\mathbf{x}_n)}{q_k} \right] \\ &\geq \sum_{n=1}^{N} \sum_{k=1}^{K} \gamma_{nk} \left[\log p_{\theta_k}(\mathbf{x}_i) + \log \pi_k - \log \gamma_{nk} \right] \end{aligned}$$

where we used Jensen's inequality in the last step.
Indeed, for $\alpha_i \geq 0$, $\sum \alpha_i = 1$ and any $\{x_i > 0\}$ it holds:

$$\log \left(\sum_i \alpha_i x_i \right) \geq \sum_i \alpha_i \log(x_i)$$

3. See lecture slides (similar procedure of 5).
4. The optimized distribution, γ, is equal to the probability of the latent variables z.

$$\gamma_{nk} = p(z_{nk}) = p(z_k \mid \mathbf{x}_n) = \text{probability that } x_n \text{ is assigned to cluster } k.$$

5. Include the constraint $\sum_{k=1}^K \pi_k = 1$ with a Lagrange multiplier:

$$\sum_{n=1}^N \sum_{k=1}^K \gamma_{nk} \Big(\log \pi_k + \log \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \Sigma_k) \Big) + \lambda \left(\sum_{k=1}^K \pi_k - 1 \right)$$

Take the derivative with respect to π_k and set it to zero:

$$\sum_{n=1}^N \gamma_{nk} \Big(\frac{1}{\pi_k} \Big) + \lambda = 0 \iff \pi_k = \frac{\sum_{n=1}^N \gamma_{nk}}{-\lambda}$$

To determine λ, use the normalization condition:

$$\sum_{k=1}^K \pi_k = \sum_{k=1}^K \frac{\sum_{n=1}^N \gamma_{nk}}{-\lambda} = \frac{\sum_{n=1}^N \sum_{k=1}^K \gamma_{nk}}{-\lambda} = \frac{N}{-\lambda} = 1 \iff -\lambda = N$$

Here we used the fact that $\sum_{k=1}^K \gamma_{nk} = \sum_{k=1}^K p(z_k \mid x_n) = 1$. Hence the mixing coefficients are:

$$\pi_k = \frac{\sum_{n=1}^N \gamma_{nk}}{N}$$

6. Take the derivative with respect to $\boldsymbol{\mu}_k$ and set it to zero:

$$\begin{aligned} &\frac{\partial}{\partial \boldsymbol{\mu}_k} \left\{ \sum_{n=1}^N \sum_{k=1}^K \gamma_{nk} \Big(\log \pi_k + \log \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \Sigma_k) \Big) \right\} \\ &= \frac{\partial}{\partial \boldsymbol{\mu}_k} \left\{ \sum_{n=1}^N \sum_{k=1}^K \gamma_{nk} \left[-\frac{1}{2} (\mathbf{x}_n - \boldsymbol{\mu}_k)^T \Sigma_k^{-1} (\mathbf{x}_n - \boldsymbol{\mu}_k) \right] \right\} \\ &= \sum_{n=1}^N \gamma_{nk} \Sigma_k^{-1} (\mathbf{x}_n - \boldsymbol{\mu}_k) = \Sigma_k^{-1} \sum_{n=1}^N \gamma_{nk} (\mathbf{x}_n - \boldsymbol{\mu}_k) = 0 \end{aligned}$$

Here we used the property that for a symmetric matrix $A \in \mathrm{R}_D^D$ and a vector $\mathbf{x} \in \mathrm{R}^D$, then $\frac{\partial}{\partial \mathbf{x}} \mathbf{x}^T A \mathbf{x} = 2A\mathbf{x}$. One possible solution is then:

$$\boldsymbol{\mu}_k = \frac{\sum_{n=1}^N \gamma_{nk} \mathbf{x}_n}{\sum_{n=1}^N \gamma_{nk}}$$

Problem 2 (Singularities in Gaussian Mixture Models):

In this section, we study the problem of singularities in the mixture of Gaussian models. Consider the data set $\mathbf{X}$ consisting of N i.i.d observations $\{\mathbf{x}_1, \dots, \mathbf{x}_N\}$. The goal is to cluster this data set using mixture of K Gaussians.

1. For the data point $\mathbf{x}_n$ we have log-likelihood

$$\ln p(\mathbf{x}_n \mid \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \ln \left\{ \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\}.$$

2. Computing the likelihood assuming $\boldsymbol{\mu}_j = \mathbf{x}_n$ leads to

$$\begin{aligned} p(\mathbf{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) &= \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) \\ &= \mathcal{N}(\mathbf{x}_n|\mathbf{x}_n, \sigma_j^2\mathbf{I}) \\ &= \frac{1}{(2\pi)^{D/2}}\frac{1}{\sigma_j^D} \end{aligned} \tag{5}$$

3. We see that as $\sigma_j \to 0$, (5) goes to infinity and so the *likelihood function* will also go to infinity. Thus the maximization of the log likelihood function is not a well posed problem and causes the convergence to be very slow. This can lead to a very poor clustering. Such singularities will always be present and will occur whenever one of the Gaussian components 'collapses' onto a specific data point.

4. This problem does not arise in the case of a single Gaussian distribution. If a single Gaussian collapses onto a data point it will contribute multiplicative factors to the likelihood function arising from the other data points and these factors will go to zero exponentially fast, giving an overall likelihood that goes to zero rather than infinity.

   However, once we have (at least) two components in the mixture, one of the components can have a finite variance and therefore assign finite probability to all of the data points while the other component can shrink onto one specific data point and thereby contribute an ever increasing additive value to the log likelihood.

5. We can hope to avoid the singularities by using suitable heuristics, for instance by detecting when a Gaussian component is collapsing and resetting its mean to a randomly chosen value while also resetting its covariance to some large value, and then continuing with the optimization.

Problem 3 (Identifiability):

A further issue in finding maximum likelihood solutions arises from *identifiability*. In this section we study *identifiability* in mixture models.

1. For any given maximum likelihood solution, a K-component mixture will have a total of $K!$ equivalent solutions corresponding to the $K!$ ways of assigning K sets of parameters to K components.

2. Because any of the equivalent solutions is as good as any other. Using any permutation of these parameters leads to the same clustering with permuted cluster indices.