DM2 : Probabilistic Graphical models 2019/2020

AL HOUCINE KILANI
SALMANE NAOUMI

December 30, 2019

EMAILS: al_houceine.kilani@ens-paris-saclay.fr salmane.naoumi@ens-paris-saclay.fr

1 Classification: K-means, and the EM algorithm

1.1 Derive the expressions of the parameters (p$_k$,μ_k, D_k) at each iteration of the corresponding EM algorithm. (Explain your derivations.)

Assumption : (X, Z) are random variables where X is observed (our data) and Z is non observed (unknown cluster center).

Let us consider $\theta = (p,\, \mu,\, D)$

$p_\theta(x, z)$: joint density depending on θ (the model)

The goal is to maximize the following (log) probability :

$log(p_\theta(x)) = log(\sum_z p_\theta(x, z))$

$$\begin{aligned} log(p_\theta(x, z)) &= log(\prod_{i=1}^{n} p_\theta(x_i, z_i)) \\ &= \sum_{i=1}^{n} log(p_\theta(x_i, z_i)) \\ &= \sum_{i=1}^{n} log(p_\theta(x_i|z_i)p_\theta(z_i)) \\ &= \sum_{i=1}^{n} log(p_\theta(x_i|z_i)) + log(p_\theta(z_i)) \end{aligned} \tag{1}$$

Introducing : $z_i^j = 1$ if $z_i = j$ and 0 if else, we can rewrite the log likelihood as :

$$\begin{aligned} log(p_\theta(x, z)) &= \sum_{i=1}^{n} log(\prod_{j=1}^{K} \mathcal{N}(x_i; \mu_j, D_j)^{z_i^j}) + log(\prod_{j=1}^{K} p_j^{z_i^j}) \\ &= \sum_{i=1}^{n}\sum_{j=1}^{K} z_i^j log(\mathcal{N}(x_i; \mu_j, D_j)) + \sum_{i=1}^{n}\sum_{j=1}^{K} z_i^j log(p_j) \end{aligned} \tag{2}$$

Knowing that

$$\mathcal{N}(x_i; \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{d}{2}} |\Sigma|^{\frac{d}{2}}} \exp\left(-\frac{1}{2}(x_i - \mu)^T \Sigma^{-1} (x_i - \mu)\right)$$

where d is the dimensionnality of X

E- Step:

We calculate the expected value of the complete log likelihood function, with respect to the conditional distribution of Z given X under the current estimate of the parameter

$$E_{Z|X}[log(p_\theta(x,z))] = \sum_{i=1}^{n}\sum_{j=1}^{K} E_{Z|X}[z_i^j] log(\mathcal{N}(x_i; \mu_j, D_j)) + \sum_{i=1}^{n}\sum_{j=1}^{K} E_{Z|X}[z_i^j] log(p_j) \tag{3}$$

We have that

$$\begin{aligned} E_{Z|X}[z_i^j] &= p(z_i = j | x_i) \\ &= \frac{p(x_i | z_i = j) p(z_i = j)}{p(x_i)} \\ &= \frac{p(x_i | z_i = j) p(z_i = j)}{\sum_{k=1}^{K} p(x_i | z_i = k) p(z_i = k)} \\ &= \frac{p_j \mathcal{N}(x_i; \mu_j, D_j)}{\sum_{k=1}^{K} p_k \mathcal{N}(x_i; \mu_k, D_k)} \end{aligned} \tag{4}$$

Where we use the current estimate of θ for all the computations.

M- Step:

We maximize w.r.t θ (considering only the terms that are multiplied by (4) and not the term (4) itself). Let us write $E_{Z|X}[z_i^j = C_{ij}$.

- Differentiation w.r.t p_j under the constraint of $\sum_j p_j = 1$ (introduce langrangian factor λ) :

$$\frac{\partial E_{Z|X}[z_i^j]}{\partial p_j} = 0 \implies \frac{\sum_i C_{ij}}{p_j} - \lambda = 0 \tag{5}$$

  since $\sum_j p_j = 1 \implies \lambda = \sum_{ij} C_{ij}$ thus $p_j = \frac{\sum_i C_{ij}}{\sum_{ij} C_{ij}}$

  but we also have $\sum_j C_{ij} = 1$ leading to $\sum_{ij} C_{ij} = n$

  thus

$$p_j = \frac{\sum_{i=1}^{n} C_{ij}}{n}$$

- Differentiation w.r.t μ_j :

  we get that $\frac{\partial E_{Z|X}[z_i^j]}{\partial \mu_j} = 0 \implies \sum_i C_{ij}(x_i - \mu_j)\mu_j D_j^{-1} = 0 \implies \sum_i C_{ij} x_i = \sum_i C_{ij} \mu_j$

  Leading to the final expression :

$$\mu_j = \frac{\sum_i C_{ij} x_i}{\sum_i C_{ij}}$$

Differentiation w.r.t D_j :

$$\frac{\partial E_{Z|X}[z_i^j]}{\partial D_j} = 0$$

$$\sum_{i=1}^{n} C_{ij} \frac{-d}{2} \frac{1}{|D_j|} |D_j| \left(D_j^{-1}\right)^T + \frac{1}{2} \left(D_j^{-1}\right)^T \sum_{i=1}^{n} C_{ij} \left(\mathbf{x}_i - \mu_j\right) \left(\mathbf{x}_i - \mu_j\right)^T \left(D_j^{-1}\right)^T = 0$$

$$-d * \sum_{i=1}^{n} C_{ij} + \left(D_j^{-1}\right)^T \sum_{i=1}^{n} C_{ij} \left(\mathbf{x}_i - \mu_j\right) \left(\mathbf{x}_i - \mu_j\right)^T = 0$$

$$\sum_{i=1}^{n} C_{ij} \left(\mathbf{x}_i - \mu_j\right) \left(\mathbf{x}_i - \mu_j\right)^T = D_j * d * \sum_{i=1}^{n} C_{ij}$$

Finally we get :

$$D_j = \frac{\sum_{i=1}^{n} C_{ij} \left(\mathbf{x}_i - \mu_j\right) \left(\mathbf{x}_i - \mu_j\right)^T}{d * \sum_{i=1}^{n} C_{ij}}$$

This is the general formula without any assumption on D_j, but since we want it to be diagonal, we replace only keep the diagonal terms of the $(x_i - \mu_j)(x_i - \mu_j)^T$ matrix.

N.B : in the calculations there is the d in the denominator, but when computing it we get very small ellipses... and when setting it to 1 we get normal results.

1.2 What may be the advantage of such a model, compared to the more standard Gaussian mixture model, where covariance matrices are full?

I do not know how to prove it but it seems that a linear combination of diagonal matrices can manage to approximate a full covariance matrix as it is mentionned in [1]

"It is also important to note that because the component Gaussian are acting together to model the overall feature density, full covariance matrices are not necessary even if the features are not statistically independent. The linear combination of diagonal covariance basis Gaussians is capable of modeling the correlations between feature vector elements. The effect of using a set of M full covariance matrix Gaussians can be equally obtained by using a larger set of diagonal covariance Gaussians." Also, the number of parameters to optimize for diagonal covariance matrices is more favourable from an algorithmical point of view.

1.3 Implementation

In this section, we are using the IRIS dataset that consists of 3 different types of irises (Setosa, Versicolour, and Virginica) represented by 4 features.
First, we run the k-Means algorithm (sickit-learn version) on the IRIS data.In K-means algorithm, there is no notion of centers and clusters covariance matrices.
Therefore, we only plot the scatter plots for every pair of dimensions, with clusters represented with different colors. We got the following results :

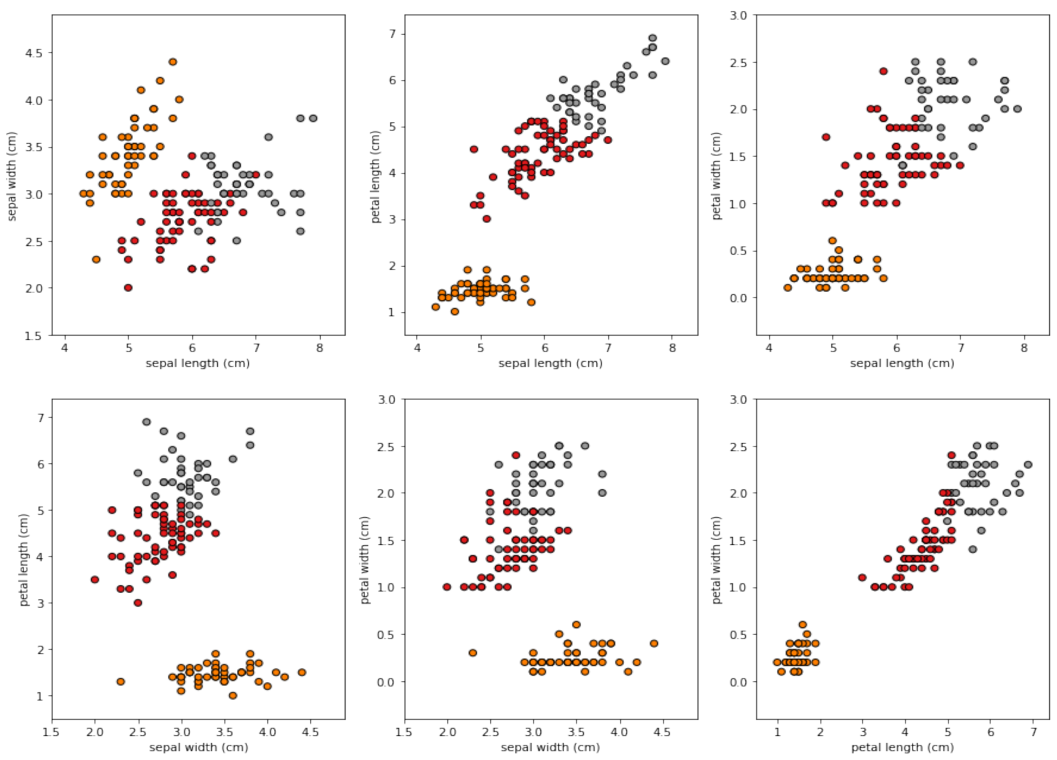

Figure 1: Results of K-means Clustering on IRIS data

Also, we run the GMM model with full covariance matrices (using scikit learn version). We also plot sur-imposed ellipsis that are linked to covariance matrices (using its eigenvectors) and represents confidences ellipsoids that contains for example 95% of all samples that can be drawn from the underlying 2D Gaussian distribution. We got the following results :

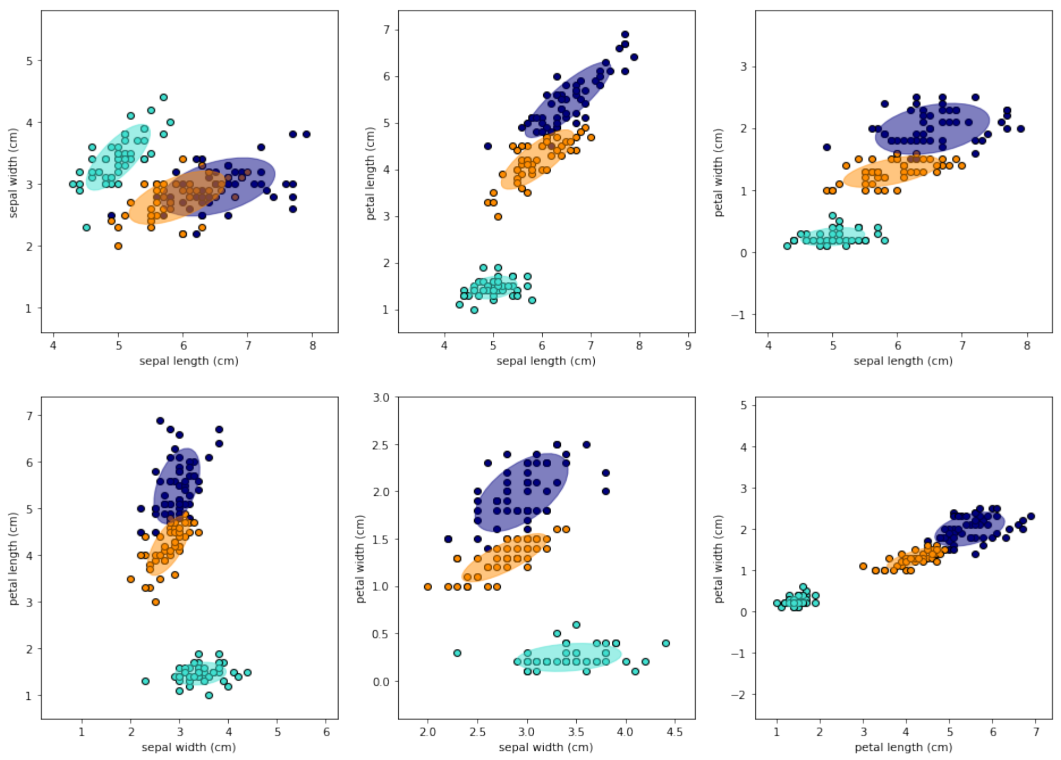

Figure 2: Results of GMM (full covariance matrix) Clustering on IRIS data

Finally, implementing the GMM version with diagonal covariance matrices(formulas as obtained before) we get the following results. As we can see, the vectors defining the ellipses of each cluster are parallel to basis vectors since we consider covariance matrices with only diagonal terms. In fact, this is constraining the search in the hypothesis space and can be bad sometimes for modelling.

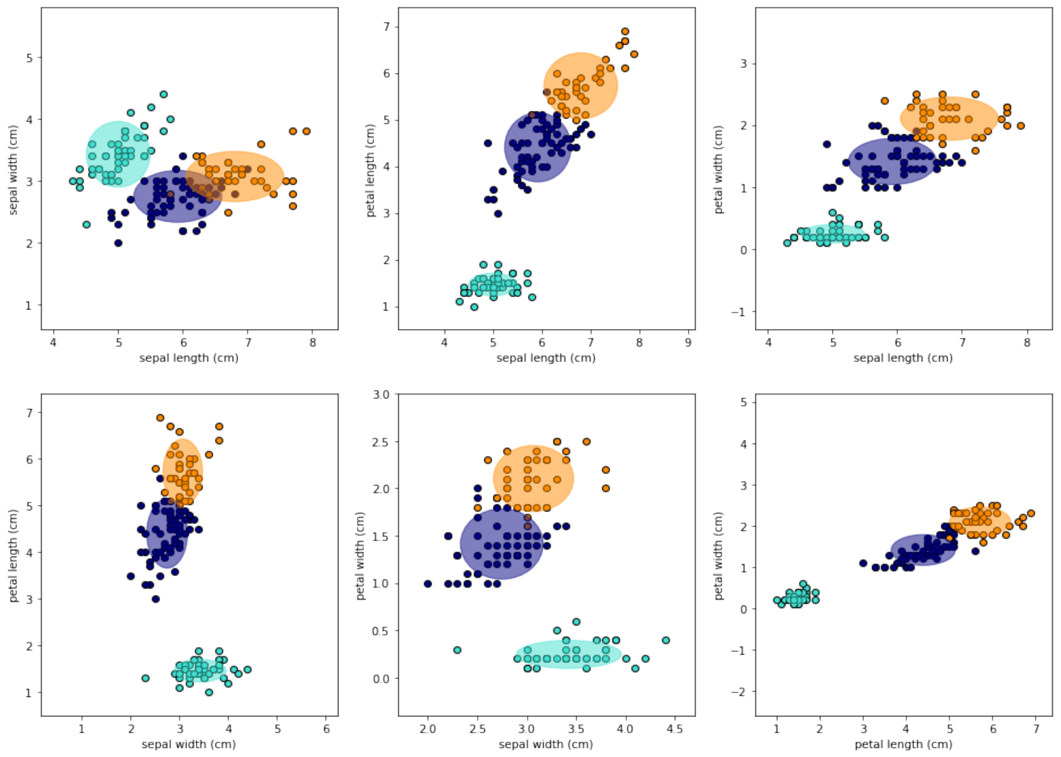

Figure 3: Results of GMM (diagonal covariance matrix) Clustering on IRIS data

Finally, comparing the clustering assignment of each method to the true labels we can see that the GMM algorithm with full covariance matrix performs well than all other algorithms by achieving an accuracy of 97%.

Algorithm	Accuracy
K-means	89%
GMM diagonal matrix version	91%
GMM full matrix version	**97%**

1.4 In which situations K-means is going to be significantly outperformed by the two EM algorithms discussed above? (Think about the shape of the clusters for instance.) Construct a synthetic dataset to illustrate this point (show that K-means fails to capture some of the clusters found by EM).

For instance, if the conditional distribution of X given Z is indeed a normal distribution, then the EM algorithm will be the one that performs the best.

To prove it, lets generate some 2D blobs according to gaussians and see what happens.

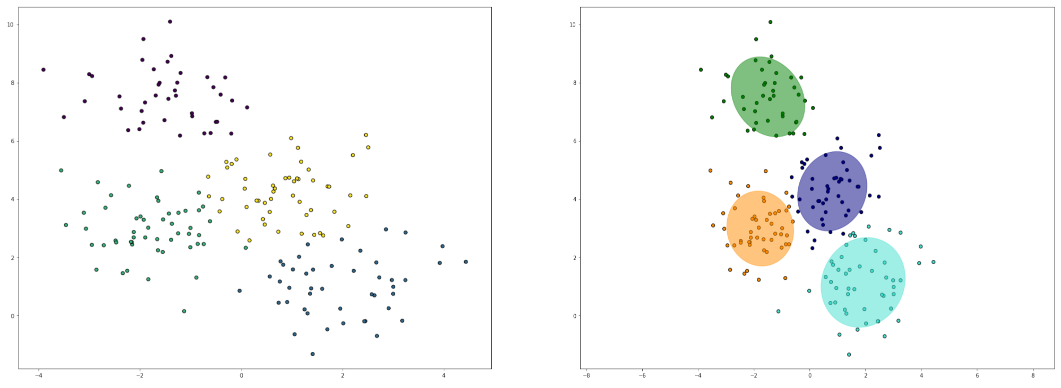

Figure 4: Blob dataset clustering results

2 Graphs, algorithms and Ising

2.1 Implement the sum-product algorithm for an undirected chain, using this trick, in order to compute all the forward and backward messages. (Explain how you represent the input of the algorithm, i.e. the functions ψ_i and $\psi_{i,i+1}$).

In our code, we are supposing that the random variables of the chain can only take K values (all of them for simplification), the factors $\psi_{i,i+1}$ are matrices of shape (K,K). And the ψ_i have a shape (K,1). And we do to bother about the normalization of the messages.

2.2 For w = 10, h = 100, $\alpha = 0$, use your implementation from point 1 to compute exactly Z(α, β) as a function of β, and plot it. (Hint: recall the idea behind the junction tree algorithm).

An easy junction tree to extract from the graph induced by this model (2D grid of points) is a simple chain where :

- Each row of the grid is merged into one vertex (and thus the number of value it takes is 2^w corresponding to all the binary combinations that could take the initial odes on the 2D grid)
- The edges between these new verteces summarize the relationship of all the verteces merged into the node with their neighbours.

Therefore what we did was :

- Generate the grid of all the 2^w binary combinations
- Compute the potential of each "merged" node by calculating the number of neighbors that are equal (number of zeros side by side or ones side by side) in each row $\implies$ This gives the potentials ψ_i

- Compute the potential of each edge between those ”meged” nodes by calculation the number of neighbors that are equal columns wise (vertically) for each pair of states $\implies$ This gives the potentials $\psi_{i,i+1}$

By construction, all these ψ_i are equal and all the $\psi_{i,i+1}$ are equal.
To speed up the calculations of those potentials, we use some tricks involving numpy arrays.

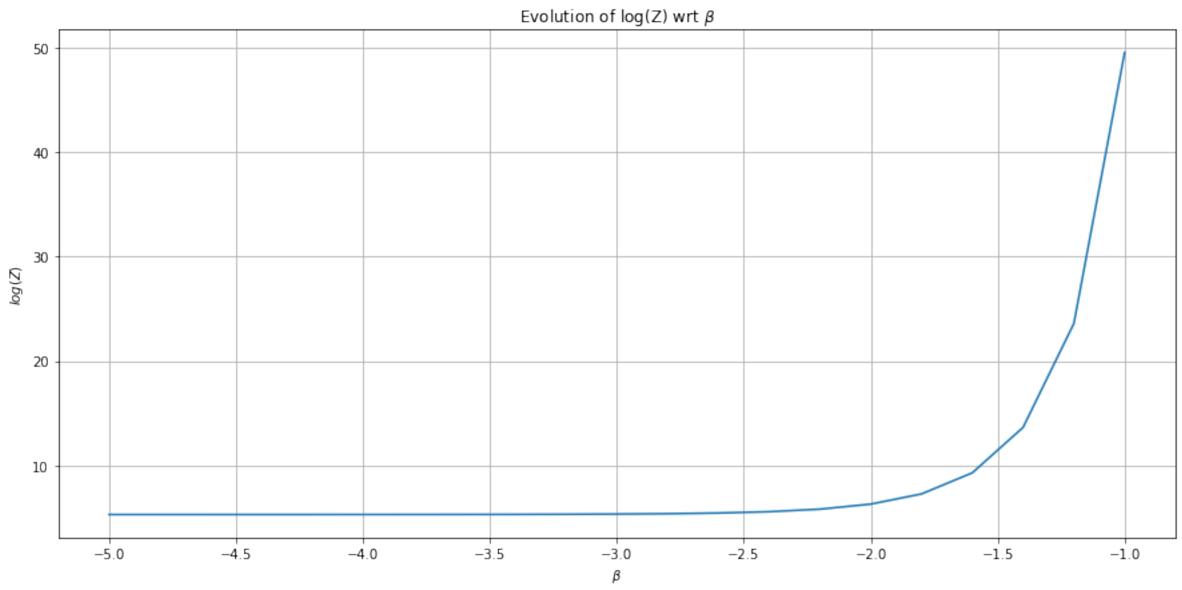

Figure 5: Z(0,β) w.r.t β

2.3 Implement loopy belief propagation to obtain a faster approximation of $Z(\alpha, \beta)$. (Explain.) For which values of β the approximation error gets larger?

Because the initial graph contains cycles, it is possible to turn to approximate probabilistic inference algorithms to obtain faster approximations. In our case, we are using the loopy belief propagation algorithm by iterating the sum product algorithm to pass messages between nodes and finally obtain an approximation of $Z(\alpha, \beta)$ for different values of β.

References

[1] https://math.stackexchange.com/questions/1805946/gmm-with-full-and-diagonal-covariances