Université de Montréal

IFT6269 - Probabilistic Graphical Models

# Homework 4

*Author*
Patrice Béchard
20019173

November 21, 2017

## 1 Entropy & Mutual Information

1. Let $X$ be a discrete random variable on a finite space $\mathcal{X}$ with $|\mathcal{X}| = k$.

**(a)** We know that the entropy is given by :

$$\begin{aligned} H(X) &\triangleq -\sum_{x\in\mathcal{X}} p(x)\log p(x) \qquad (1)\\ &= -\sum_{i=1}^{k} p(X = x_i)\log p(X = x_i) \end{aligned}$$

where we sum over all possible values that $X$ can take. In the general case, where $X$ can take multiple possible values, knowing that the probability $p(X = x_i)$ of an event happening is $\in [0, 1[$ and knowing that the logarithm of a value in that range is always negative, we have:

$$H(X) = -\sum_{i=1}^{k} \underbrace{p(X = x_i)}_{\in[0,1[}\underbrace{\log p(X = x_i)}_{\text{always } <0}$$

We thus have **minus** the sum of $k$ **negative** terms, which yields a **positive** value. However, if $X$ is a constant, then $p(X = x) = 1$ and $\log p(X = x) = 0$. In that case, the entropy is given by :

$$H(X) = -\underbrace{p(X = x)}_{1}\underbrace{\log p(X = x)}_{0} = 0$$

We thus have proven that $H(x) \geq 0$, with equality only when $X$ is a constant. ■

**(b)** We now denote by $p$ the distribution of $X$ and $q$ the uniform distribution on $\mathcal{X}$. We know that the Kullback-Leibler divergence is given by :

$$D_{KL}(p||q) \triangleq \sum_{x\in\mathcal{X}} p(x)\log\frac{p(x)}{q(x)} \qquad (2)$$

We can develop this expression :

$$\begin{aligned} D_{KL}(p||q) &= \underbrace{\sum_{x\in\mathcal{X}} p(x)\log p(x)}_{-H(X)} - \sum_{x\in\mathcal{X}} p(x)\log\underbrace{q(x)}_{=1/k}\\ &= -H(X) - \left(\log\frac{1}{k}\right)\underbrace{\sum_{x\in\mathcal{X}} p(x)}_{1}\\ &= -H(X) + \log k \qquad (3) \end{aligned}$$

We find that the Kullback-Leibler divergence $D_{KL}(p||q)$ is simply minus the entropy $H(X)$ of the distribution $p$ plus the logarithm of the number of possible values that $X$ can take.

**(c)** Knowing that maximizing entropy is the same as minimizing the Kullback-Leibler divergence, and knowing that one of the properties of the Kullback-Leibler divergence is that $D_{KL}(p||q) \geq 0$, we can use the result obtained in the previous problem (equation 3) to find an upper bound on the entropy that depends on $k$:

$$D_{KL}(p||q) = -H(X) + \log k$$
$$H_{max}(X) = \log k - \underbrace{D_{KL,min}(p||q)}_{0}$$
$$\boxed{H_{max}(X) = \log k}$$

2. Considering a pair of random variables $(X_1, X_2)$ defined over the finite set $\mathcal{X}_1 \times \mathcal{X}_2$. Let $p_{1,2}$, $p_1$ and $p_2$ denote respectively the joint distribution, the marginal distribution of $X_1$ and the marginal distribution of $X_2$. The mutual information $I(X_1, X_2)$ is defined as :

$$I(X_1, X_2) := \sum_{(x_1,x_2)\in\mathcal{X}_1\times\mathcal{X}_2} p_{1,2}(x_1, x_2) \log \frac{p_{1,2}(x_1, x_2)}{p_1(x_1)p_2(x_2)} \tag{4}$$

**(a)** We can prove that $I(X_1, X_2) \geq 0$. First, we notice that $\sum_{(x_1,x_2)\in\mathcal{X}_1,\mathcal{X}_2} p_{1,2}(x_1, x_2) = 1$. Second, we see that by inverting the logarithm, we can obtain a convex function :

$$I(X_1, X_2) = -\sum_{(x_1,x_2)\in\mathcal{X}_1,\mathcal{X}_2} p_{1,2}(x_1, x_2) \log \frac{p_1(x_1)p_2(x_2)}{p_{1,2}(x_1, x_2)}$$

We can apply Jensen's inequality on the convex function to obtain :

$$\begin{aligned} I(X_1, X_2) &\geq -\log \left[ \sum_{(x_1,x_2)\in\mathcal{X}_1,\mathcal{X}_2} \cancel{p_{1,2}(x_1, x_2)} \frac{p_1(x_1)p_2(x_2)}{\cancel{p_{1,2}(x_1, x_2)}} \right] \\ &= -\log \left[ \sum_{(x_1,x_2)\in\mathcal{X}_1,\mathcal{X}_2} p_1(x_1)p_2(x_2) \right] \\ &= 0 \quad \blacksquare \end{aligned}$$

**(b)** By starting from the definition of the mutual information (equation 4), writing the sum over all possible states $\sum_{(x_1,x_2)\in\mathcal{X}_1\times\mathcal{X}_2}$ as $\sum_{x_1,x_2}$, we have :

$$\begin{aligned} I(X_1, X_2) &= \sum_{x_1,x_2} p_{1,2}(x_1, x_2) \log \frac{p_{1,2}(x_1, x_2)}{p_1(x_1)p_2(x_2)} \\ &= \underbrace{\sum_{x_1,x_2} p_{1,2}(x_1, x_2) \log p_{1,2}(x_1, x_2)}_{-H(X_1,X_2)} - \sum_{x_1,x_2} p_{1,2}(x_1, x_2) \log p(x_1) - \sum_{x_1,x_2} p_{1,2}(x_1, x_2) \log p(x_2) \\ &= -H(X_1, X_2) - \sum_{x_1,x_2} p(x_2 \mid x_1)p(x_1) \log p(x_1) - \sum_{x_1,x_2} p(x_1 \mid x_2)p(x_2) \log p(x_2) \end{aligned}$$

$$= -H(X_1, X_2) - \sum_{x_1} \left( p(x_1) \log p(x_1) \underbrace{\sum_{x_2} p(x_2 \mid x_1)}_{=1} \right) - \sum_{x_2} \left( p(x_2) \log p(x_2) \underbrace{\sum_{x_1} p(x_1 \mid x_2)}_{=1} \right)$$

$$= -H(X_1, X_2) - \underbrace{\sum_{x_1} p(x_1) \log p(x_1)}_{-H(X_1)} - \underbrace{\sum_{x_2} p(x_2) \log p(x_2)}_{-H(X_2)}$$

$$= -H(X_1, X_2) + H(X_1) + H(X_2)$$

**(c)** Since the mutual information $I(X_1, X_2) \geq 0$, we have that the maximum entropy that $p_{1,2}(x_1, x_2)$ can have is $H_{\max}(X_1, X_2) = H(X_1) + H(X_2)$. This happens when $p_{1,2}(x_1, x_2) = p_1(x_1)p_2(x_2)$, that is when random variables $X_1$ and $X_2$ are mutually independent.

## 2 HMM - Implementation

1. The implementation of the $\alpha$ and $\beta$ recursions can be found in the file `hwk4.ipynb`. They are implemented as the methods `_forward` and `_backward` of the `HMM` class. This is a great example of dynamic programming, where we initiate an empty array of $\alpha$ or $\beta$ values, we first compute the base case for both cases, and we continue to compute the following values using the values already in the array. We will be able to compute the *smoothing* distribution $\gamma_t^{(i)} = p\left(z_t^{(i)} | \mathbf{x}, \theta\right)$ and the pair-marginals $\xi_{t-1,t}^{(i,j)} = p\left(z_{t-1}^{(i)}, z_t^{(j)} | \mathbf{x}, \theta\right)$. We can develop to find these distributions as functions of $\alpha$ and $\beta$ :

$$\begin{aligned}
\gamma_t^{(i)} &= p\left(z_t^{(i)} | \mathbf{x}\right) \\
&= \frac{p\left(\mathbf{x} | z_t^{(i)}\right) p\left(z_t^{(i)}\right)}{p(\mathbf{x})} \\
&= \frac{p\left(x_1, \ldots, x_t | z_t^{(i)}\right) p\left(x_{t+1}, \ldots, x_T | z_t^{(i)}\right) p\left(z_t^{(i)}\right)}{p(\mathbf{x})} \\
&= \frac{\alpha_t^{(i)} \beta_t^{(i)}}{\sum_{k=1}^K \alpha_t^{(k)} \beta_t^{(k)}}
\end{aligned}$$

$$\begin{aligned}
\xi_{t-1,t}^{(i,j)} &= p\left(z_{t-1}^{(i)}, z_t^{(j)} | \mathbf{x}\right) \\
&= \frac{p\left(\mathbf{x} | z_{t-1}^{(i)}, z_t^{(j)}\right) p\left(z_{t-1}^{(i)}, z_t^{(j)}\right)}{p(\mathbf{x})} \\
&= \frac{p\left(x_1, \ldots, x_{t-1} | z_{t-1}^{(i)}\right) p\left(x_t | z_t^{(j)}\right) p\left(x_{t+1}, \ldots, x_T | z_t^{(j)}\right) p\left(z_t^{(j)} | z_{t-1}^{(i)}\right) p\left(z_{t-1}^{(i)}\right)}{p(\mathbf{x})} \\
&= \frac{\alpha_{t-1}^{(i)} \, \mathcal{N}(x_t | \mu_j, \Sigma_j) \, A_{ji} \, \beta_t^{(i)}}{p(\mathbf{x})}
\end{aligned}$$

2. Using the same parameters for the means and covariance matrix of the 4 Gaussians learned in homework 3 (these are given here), and initializing the initial probability distribution $\pi_k = \frac{1}{4}$ and

setting $A$ to be the matrix with diagonal coefficients $A_{ii} = \frac{1}{2}$ and off-diagonal coefficients $A_{ij} = \frac{1}{6}$ for all $(i, j) \in \{1 \dots 4\}^2$, we can compute the $\alpha_t$'s and $\beta_t$'s using the implementation of the $\alpha$ and $\beta$ recursions. We can compute the smoothing distribution $\gamma_t^{(i)}$. Figure 1 presents the probability for each of the 4 states as a function of $t$ for the first 100 datapoints in the file `EMGaussian.test`.

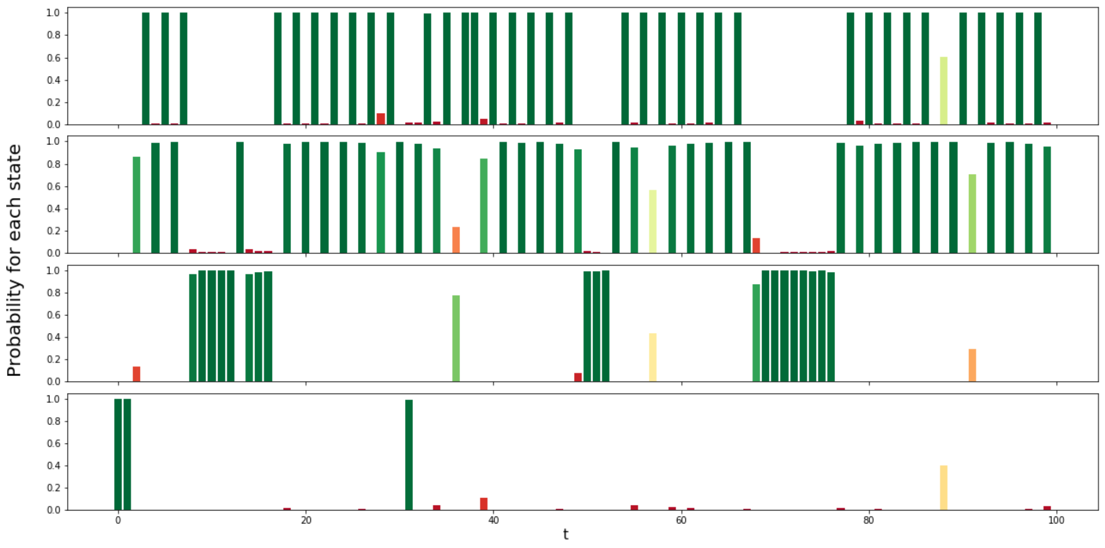

Figure 1: Probability $p(z_t \mid x_1, \dots, x_T)$ for each of the 4 states (top panel is state 1, bottom panel is state 4) as a function of $t$ for the first 100 datapoints in the file `EMGaussian.test`. The color of each bar represents the probability of the datapoint for each state, going from red to green

3. Here we derive the M-step update for $\hat{\pi}$, $\hat{A}$, $\hat{\mu}_k$ and $\hat{\Sigma}_k$ for $k = 1, \dots, 4$ during the EM algorithm, as a function of the quantities computed during the E-step. We first start by taking the log-likelihood of the probability distribution given by the HMM :

$$\begin{aligned}
\log p(x, z|\theta) &= \log \left( p(z_1|\pi) \prod_{t=2}^{T} p(z_t|z_{t-1}, A) \prod_{t=1}^{T} p(x_t|z_t, \mu, \Sigma) \right) \\
&= \log \left( \prod_{k=1}^{K} \pi_k^{z_1^{(k)}} \prod_{t=2}^{T} \prod_{i=1}^{K} \prod_{j=1}^{K} A_{ji}^{z_{t-1}^{(i)} z_t^{(j)}} \prod_{t=1}^{T} \prod_{k=1}^{K} \mathcal{N}(x_t|\mu_k, \Sigma_k)^{z_t^{(k)}} \right) \\
&= \sum_{k=1}^{K} z_1^{(k)} \log \pi_k + \sum_{t=2}^{T} \sum_{i=1}^{K} \sum_{j=1}^{K} \left[ z_{t-1}^{(i)} z_t^{(j)} \right] \log A_{ji} + \sum_{t=1}^{T} \sum_{k=1}^{K} z_t^{(k)} \log \mathcal{N}(x_t|\mu_k, \Sigma_k) \\
&= \sum_{k=1}^{K} \gamma_1^{(k)} \log \pi^k + \sum_{t=2}^{T} \sum_{i=1}^{K} \sum_{j=1}^{K} \xi_{t-1,t}^{(i,j)} \log A_{ji} + \sum_{t=1}^{T} \sum_{k=1}^{K} \gamma_t^{(k)} \log \mathcal{N}(x_t|\mu_k, \Sigma_k)
\end{aligned}$$

where we have substituted for the appropriated sufficient statistics. Now, all that we have to do is to derivate with respect to the wanted parameter to find the wanted update. Starting with $\pi^{(k)}$, we have, using the constraint that $\sum_{k=1}^{K} \pi_k = 1$ and using the Lagrange multipliers method:

$$
\begin{aligned}
\frac{\partial}{\partial \pi_k} \log p(x, z|\theta) &= \frac{\partial}{\partial \pi_k} \left[ \sum_{k=1}^{K} \gamma_1^{(k)} \log \pi_k + \cancel{\sum_{t=2}^{T} \sum_{i=1}^{K} \sum_{j=1}^{K} \xi_{t-1,t}^{(i,j)}} \log A_{ji} + \cancel{\sum_{t=1}^{T} \sum_{k=1}^{K} \gamma_t^{(k)} \log \mathcal{N}(x_t|\mu_k, \Sigma_k)} - \lambda \left( \sum_{k=1}^{K} \pi_k - 1 \right) \right] \\
&= \frac{\partial}{\partial \pi_k} \left[ \sum_{k=1}^{K} \gamma_1^{(k)} \log \pi_k - \lambda \left( \sum_{k=1}^{K} \pi_k - 1 \right) \right] \\
&= \frac{\gamma_1^{(k)}}{\pi_k} - \lambda \\
&= 0
\end{aligned}
$$

We then have :

$$
\begin{aligned}
\frac{\gamma_1^{(k)}}{\pi_k} &= \lambda \\
\pi_k &= \frac{\gamma_1^{(k)}}{\lambda}
\end{aligned}
$$

Using the constraint, we find that $\lambda = \sum_{i=1}^{K} \gamma_1^{(k)}$, which gives us :

$$
\boxed{\hat{\pi}_k = \frac{\gamma_1^{(k)}}{\sum_{i=1}^{K} \gamma_1^{(k)}}}
$$

We now do the same with the transition matrix to find the update of $A_{ij}$, using the Lagrange multipliers method and the constraint that $\sum_{i=1}^{K} A_{ij} = 1$ :

$$
\begin{aligned}
\frac{\partial}{\partial A_{ij}} \log p(x, z|\theta) &= \frac{\partial}{\partial A_{ij}} \left[ \cancel{\sum_{k=1}^{K} \gamma_1^{(k)} \log \pi_k} + \sum_{t=2}^{T} \sum_{i=1}^{K} \sum_{j=1}^{K} \xi_{t-1,t}^{(i,j)} \log A_{ji} + \cancel{\sum_{t=1}^{T} \sum_{k=1}^{K} \gamma_t^{(k)} \log \mathcal{N}(x_t|\mu_k, \Sigma_k)} - \lambda \left( \sum_{i=1}^{K} A_{ij} - 1 \right) \right] \\
&= \frac{\partial}{\partial A_{ij}} \left[ \sum_{t=2}^{T} \sum_{i=1}^{K} \sum_{j=1}^{K} \xi_{t-1,t}^{(i,j)} \log A_{ji} - \lambda \left( \sum_{i=1}^{K} A_{ij} - 1 \right) \right] \\
&= \frac{\sum_{t=2}^{T} \xi_{t-1,t}^{(i,j)}}{A_{ij}} - \lambda \\
&= 0
\end{aligned}
$$

We then have :

$$
\begin{aligned}
\frac{\sum_{t=2}^{T} \xi_{t-1,t}^{(i,j)}}{A_{ij}} &= \lambda \\
A_{i,j} &= \frac{\sum_{t=2}^{T} \xi_{t-1,t}^{(i,j)}}{\lambda}
\end{aligned}
$$

Once again, using the constraint that each column of the transition matrix must sum to 1, we find :

$$
\boxed{\hat{A}_{ij} = \frac{\sum_{t=2}^{T} \xi_{t-1,t}^{(i,j)}}{\sum_{l=1}^{K} \sum_{t=2}^{T} \xi_{t-1,t}^{(i,l)}}}
$$

We now do the same thing for both parameters $\mu$ and $\Sigma$. In this case, no need to use the Lagrange multipliers method. We will also reuse some results already found in other homework to go faster :

$$\begin{aligned}
\frac{\partial}{\partial \mu_k} \log p(x, z|\theta) &= \frac{\partial}{\partial \mu_k} \left[ \cancel{\sum_{k=1}^{K} \gamma_1^{(k)} \log \pi_k} + \cancel{\sum_{t=2}^{T} \sum_{i=1}^{K} \sum_{j=1}^{K} \xi_{t-1,t}^{(i,j)}} \log A_{ji} + \sum_{t=1}^{T} \sum_{k=1}^{K} \gamma_t^{(k)} \log \mathcal{N}(x_t|\mu_k, \Sigma_k) \right] \\
&= \frac{\partial}{\partial \mu_k} \left[ \sum_{t=1}^{T} \sum_{k=1}^{K} \gamma_t^{(k)} \log \mathcal{N}(x_t|\mu_k, \Sigma_k) \right] \\
&= \sum_{t=1}^{T} \gamma_t^{(k)} \underbrace{\frac{\partial}{\partial \mu_k} \log \mathcal{N}(x_t|\mu_k, \Sigma_k)}_{\Sigma^{-1}(x_t-\mu_k)} \\
&= 0
\end{aligned}$$

We then have :

$$\begin{aligned}
\cancel{\Sigma^{-1}} \left( \sum_{t=1}^{T} \gamma_t^{(k)} x_t - \mu_k \sum_{t=1}^{T} \gamma_t^{(k)} \right) &= 0 \\
\mu_k \sum_{t=1}^{T} \gamma_t^{(k)} &= \sum_{t=1}^{T} \gamma_t^{(k)} x_t \\
\boxed{\hat{\mu}_k = \frac{\sum_{t=1}^{T} \gamma_t^{(k)} x_t}{\sum_{t=1}^{T} \gamma_t^{(k)}}}
\end{aligned}$$

Finally, for the covariance matrix $\Sigma$, we have :

$$\begin{aligned}
\frac{\partial}{\partial \Sigma_k} \log p(x, z|\theta) &= \frac{\partial}{\partial \Sigma_k} \left[ \cancel{\sum_{k=1}^{K} \gamma_1^{(k)} \log \pi_k} + \cancel{\sum_{t=2}^{T} \sum_{i=1}^{K} \sum_{j=1}^{K} \xi_{t-1,t}^{(i,j)}} \log A_{ji} + \sum_{t=1}^{T} \sum_{k=1}^{K} \gamma_t^{(k)} \log \mathcal{N}(x_t|\mu_k, \Sigma_k) \right] \\
&= \frac{\partial}{\partial \Sigma_k} \left[ \sum_{t=1}^{T} \sum_{k=1}^{K} \gamma_t^{(k)} \log \mathcal{N}(x_t|\mu_k, \Sigma_k) \right] \\
&= \sum_{t=1}^{T} \gamma_t^{(k)} \underbrace{\frac{\partial}{\partial \Sigma_k} \log \mathcal{N}(x_t|\mu_k, \Sigma_k)}_{(x_t-\mu_k)(x_t-\mu_k)^T-\Sigma_k} \\
&= 0
\end{aligned}$$

We finally have :

$$\begin{aligned}
\sum_{t=1}^{T} \gamma_t^{(k)} (x_t - \mu_k)(x_t - \mu_k)^T &= \Sigma_k \sum_{t=1}^{T} \gamma_t^{(k)} \\
\boxed{\hat{\Sigma}_k = \frac{\sum_{t=1}^{T} \gamma_t^{(k)} (x_t - \mu_k)(x_t - \mu_k)^T}{\sum_{t=1}^{T} \gamma_t^{(k)}}}
\end{aligned}$$

4. The implementation of the EM algorithm to learn the parameters of the model can be found in the file `hwk4.ipynb`. It is implemented in the methods `_e_step` and `_m_step` of the `HMM` class. We use the method `train` to call the EM algorithm a certain number of iterations.

5. Figure 2 presents the normalized log-likelihood as a function of the iterations of the algorithm for both the training set `EMGaussian.train` and the test set `EMGaussian.test`.

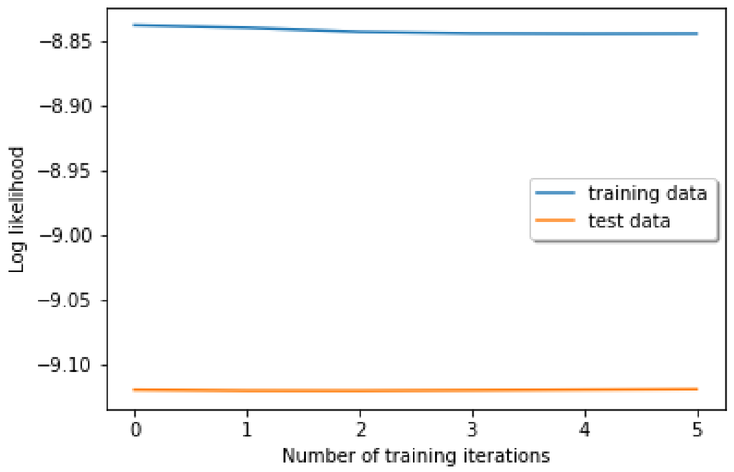

Figure 2: Log-likelihood as a function of the iterations of the EM algorithm for both the training set and the test set during the training of the Hidden Markov Model.

   We see that the log-likelihood for both the training set and the test set remain pretty constant during the training of the HMM, which only takes 5 iterations before it converges (criteria defined by a difference of less than 0.00001 for consecutive log-likelihood computations). We also see that the values obtained for both the training set and test set are pretty similar, even though the figure makes it look like they are really different.

6. Table 1 presents the values of the log-likelihoods of the full covariance Gaussian mixture model, the Gaussian mixture model with isotropic covariance matrix, the HMM and the k-means algorithm on the train and test data.

Table 1: Values of the log-likelihoods for every studied model after training on the training data as well as the test data.

| | Training data | Test data |
|---|---|---|
| HMM | $-8.84$ | $-9.12$ |
| GMM (Full) | $8.65 \times 10^{-2}$ | $-7.88 \times 10^{-2}$ |
| GMM (Isotropic) | $-7.95 \times 10^{-1}$ | $-9.05 \times 10^{-1}$ |
| K-Means | $6.70$ | $6.51$ |

   We see that the values of the normalized log-likelihoods for every model is pretty similar when comparing the value obtained with the training set and the test set. However, the values obtained for the full GMM and the HMM are pretty different from one another. This is due to the fact that the log-likelihood for both models are not computed using the same formula, which makes the comparison between these values useless. However, it can be useful to compare both GMM for different covariance matrices, since the log-likelihood is computed in the same way for both. I have included the value of the normalized objective function for the K-Means algorithm even though this is not a log-likelihood. Finally, it is not really useful to compare log-likelihood values for different models that use a different formula for log-likelihood.

7. The Viterbi algorithm estimates the most likely sequence of states for a Hidden Markov Model. After having trained the model, we first select a starting state depending on the initial probability $\pi$ of each state and the emission term associated with this state. After that, for each state, we compute the most probable transition knowing the transition matrix we learned during the training process as well as the probability of each state for the last iteration. The new probabilities are the product of this probability with the emission term for this data point. After propagating this algorithm over all the data points, we propagate back to search for each most probable state starting from the end to find the most likely sequence. Algorithm 1 presents a pseudocode for the viterbi algorithm.

**Algorithm 1:** Viterbi Algorithm

```
Function Viterbi($x, \pi, A, T, K$)
    path = zeros($T, K$);
    scores = zeros($T, K$);
    /* Forward propagation */
    for $t = 0$ to $T$ do ;                              /* for each data point */

        for $k = 0$ to $K$ do ;                          /* for each state */

            if $t = 0$ then ;                            /* Base case */

                scores[$t, k$] = $\pi[k] * $emission[$x[t], k$];
            else
                max_prob_transit = max(scores[$t-1, k$] $* A[k, :]$);
                path[$t, k$] = argmax(scores[$t-1, k$] $* A[k, :]$);
                scores[$t, k$] = max_prob_transit $*$ emission[$x[t], k$];
            end
        end
    end
    /* Backward propagation */
    sequence = zeros[$t$];
    for $t = T$ to $0$ do
        if $t = T$ then ;                                /* Base case */

            sequence[$t$] = argmax(scores[$t$]);
        else
            sequence[$t$] = path[$t+1$, sequence[$t+1$]];
        end
    end
    return sequence, scores ;
```

8. The Viterbi algorithm is implemented in the file `hwk4.ipynb` and it can be found as the method `viterbi` in the class `HMM`. Figure 3 presents the data from the training set `EMGaussian.train` where each data point has been assigned to a cluster following the Viterbi algorithm, representing the

most likely sequence of states.

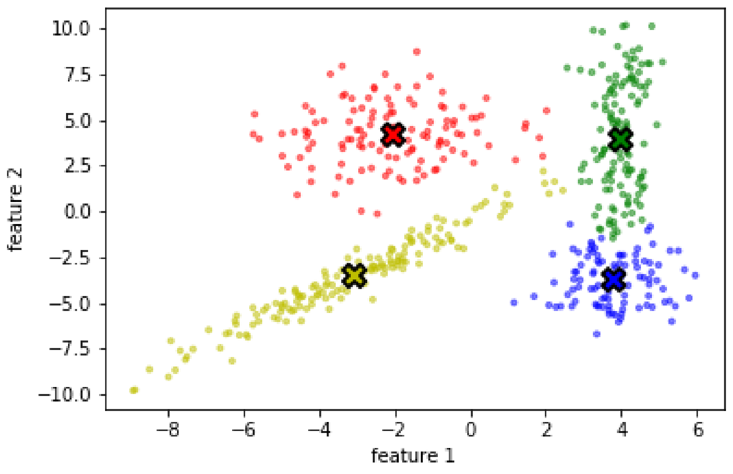

Figure 3: Representation of the training data and the cluster means with state assigned by the Viterbi algorithm.

9. For the datapoints in the test set, we can compute the marginal probability $p(z_t|x_1, \ldots, x_T)$ for each point to be in state $\{1, 2, 3, 4\}$ for the parameters learned on the training set, similarly to question 2, where the parameters were given. Figure 4 presents the probability of being in one of the four states as a function of time ($t$) for the first 100 points of the test set.

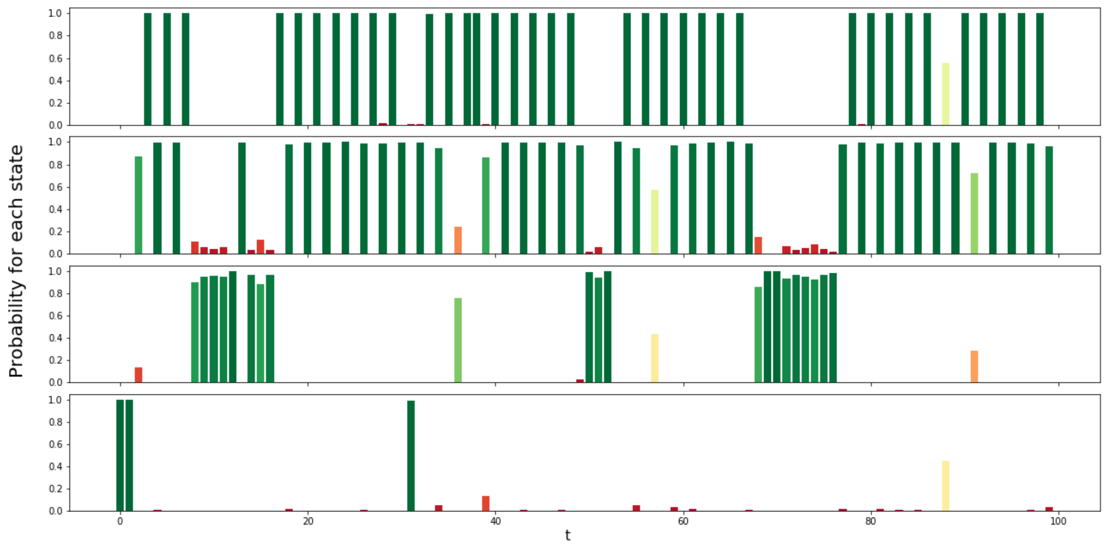

Figure 4: Probability of the first 100 datapoints from the test set `EMGaussian.test` to be in state $\{1, 2, 3, 4\}$ (top panel is state 1, bottom panel is state 4) as a function of time.

10. For each of these same 100 points, figure 5 presents the state where the marginal probability was the highest on figure 4 as a function of time.

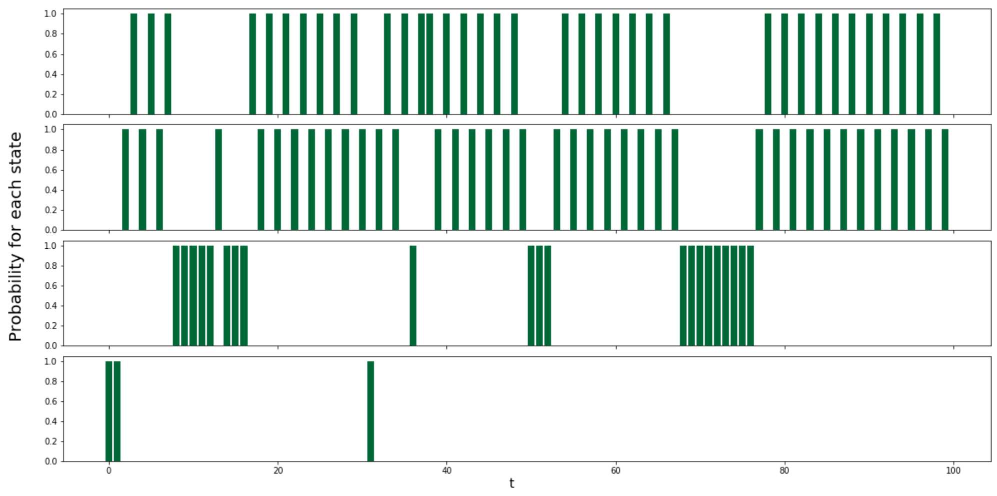

Figure 5: Most likely state of the first 100 datapoints from the test set `EMGaussian.test` (top panel is state 1, bottom panel is state 4) as a function of time.

11. We can run the Viterbi algorithm on the test data to compare the most likely sequence of states obtained by the model with the sequence of states obtained at the previous question. Figure 6 presents the most likely sequence of states for the first 100 points of the test set.

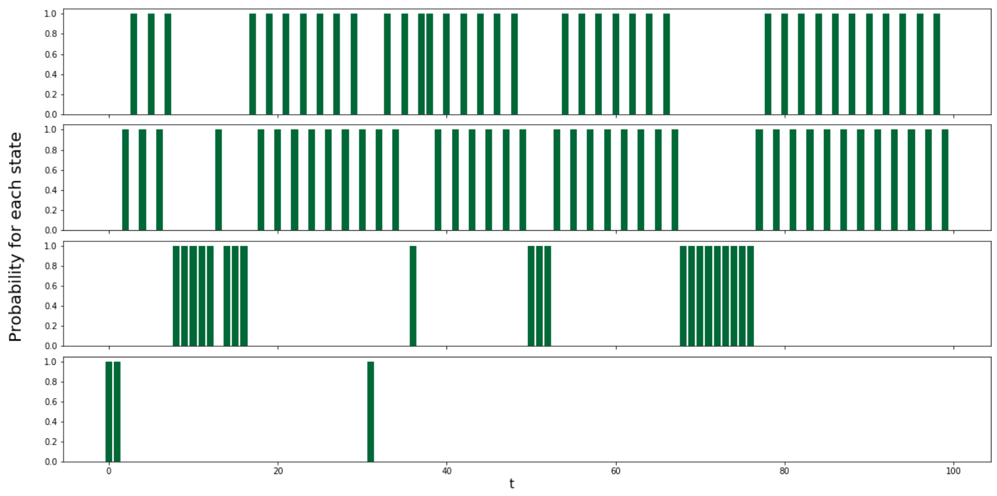

Figure 6: Most likely sequence of states of the first 100 datapoints from the test set `EMGaussian.test` (top panel is state 1, bottom panel is state 4) using the Viterbi algorithm as a function of time.

We notice that the most likely sequence of states obtained with the Viterbi algorithm is exactly the same as the sequence of states obtained in question 10. This means that the data from the training set and the test set really depend on time and is not IID, as we postulated in the last homework. We thus have successfully trained our HMM!

12. When the number of states $K$ is not known, we can set $K$ as an hyperparameter. Using three different datasets (a training set, a validation set and a test set), we can try different values of K and choose the value that yields the best results on sets that were not used for the training. We do so to ensure that our model can generalize well to new data. We can select the number of classes by trying them all, starting with 1, then 2, etc. or we can use a random search, trying different values randomly in a given range.