Master MVA
Probabilistic Graphical Models

Homework 3: HMM

Victor Busa
victor.busa@ens-paris-saclay.fr

January 5, 2018

1 HMM - Implementation

1 Recursions α and β To implement α, β recursions we will use the $log(\sum_{a \in \mathcal{A}} exp(p(a))$ trick. I won't detail the implementation of α and β recursion and I will instead focus my attention on the joint probability calculation. The joint probability $\mathbb{P}[q_{t+1} = i, q_t = i | u_0, \dots, u_T]$ is given by:

$$\mathbb{P}[q_{t+1} = i, q_t = i | u_0, \dots, u_T] = \frac{1}{\mathbb{P}[u_0, \dots, u_T]} . \alpha_t(q_t) . \beta_{t+1}(q_{t+1}) . \mathbb{P}[q_{t+1} = i, q_t = i] . \mathbb{P}[u_{t+1} | q_{t+1} = i]$$

So, using the logarithm function it comes:

$$\begin{aligned} log\ \mathbb{P}[q_{t+1} = i, q_t = i | u_0, \dots, u_T] = & \overbrace{-log\ \mathbb{P}[u_0, \dots, u_T]}^{(1)} + \overbrace{log\ \alpha_t(q_t = j)}^{(2)} + \overbrace{log\ \beta_{t+1}(q_{t+1} = i)}^{(3)} \\ & + \underbrace{log\ \mathbb{P}[q_{t+1} = i, q_t = j]}_{(4)} + \underbrace{log\ \mathbb{P}[u_{t+1} | q_{t+1} = i]}_{(5)} \end{aligned}$$

We can rewrite (1) as:

$$\begin{aligned} (1) &= -\ log \left[\sum_{q_t} \alpha_t(q_t) \beta_t(q_t) \right] \\ &= -\max_{q_t} [log\ \alpha_t(q_t) + log\ \beta_t(q_t)] - log \left[\sum_{q_t} exp \left(log\ \alpha_t(q_t) + log\ \beta_t(q_t) - \max_{q_t} [log\ \alpha_t(q_t) + log\ \beta_t(q_t)] \right) \right] \end{aligned}$$

In python we can compute such a quantity (1) as follow:

```
# compute - log(p[uO, ... uT])
pa = (alpha + beta)[0, :]
logsum = np.log(np.sum(np.exp(pa - np.max(pa))))
term1 = np.max(pa) + logsum

result -= term1
```

terms (2, 3, 4, 5) are straightforward to compute as we have:

$$
\begin{aligned}
(2) &= log\ \alpha_t(q_t = j) = alpha[t, j] \\
(3) &= log\ \beta_{t+1}(q_{t+1} = i) = beta[t+1, i] \\
(4) &= log\ \mathbb{P}[q_{t+1} = i, q_t = j] = log\ A_{i,j} \\
(5) &= log\ \mathbb{P}[u_{t+1}|q_{t+1} = i] = log\ \mathcal{N}_{t+1}(\mu_i, \Sigma_i)
\end{aligned}
$$

As we want the joint probability for all values of i, j, and T we have to iterates over all these variables, such that the naive implementation of the joint probability calculation in python can be written as:

```
def joint_proba(alpha, beta, data, mu, sigma, A):
    (T, K) = alpha.shape

    # need to store p[qt+1 = i, qt = j | uo,... uT]
        # for all i, j in {1,...,K} and all t in 0 T-1
    result = np.zeros((T-1, K,K))

    # compute - log(p[uO, ... uT])
    pa = (alpha + beta)[0, :]
    logsum = np.log(np.sum(np.exp(pa - np.max(pa))))
    term1 = np.max(pa) + logsum

    result -= term1

    for t in range(T-1):
        proba_t = np.zeros((K, K))

        for i in range(K):
            # proba[i, j] = log(p[qt+1 = i, qt = j | uo,... uT])
            proba_t[i,:] += np.log(multivariate_normal.pdf(data[t+1],
                        mean=mu[i], cov=sigma[i])) + beta[t+1, i]

            for j in range(K):
                proba_t[i, j] += np.log(A[i,j]) + alpha[t, j]

        result[t, :] += proba_t

    return np.exp(result)
```

Note: The above code is a naive implementation and it is here for explanation reason only. For efficiency sackness the code has been improved by using matrix operation to avoid looping over i and j.

2. Smoothing computation To compute the smoothing quantity: $p(q_t|u_1, \ldots, u_T)$ we will use the $log(\sum_{a \in \mathcal{A}} exp(p(a))$ trick. Let's recall that the smoothing quantity is defined as:

$$p(q_t|u_1,\ldots,u_T) = \frac{\alpha_t(q_t)\beta_t(q_t)}{\sum\limits_{q_t}\alpha_t(q_t)\beta_t(q_t)}$$

Thus we have:

$$\begin{aligned}
log\ p(q_t|u_1,\ldots,u_T) &= log\ \alpha_t(q_t) + log\ \beta_t(q_t) - log\left(\sum_{q_t}\alpha_t(q_t)\beta_t(q_t)\right)\\
&= log\ \alpha_t(q_t) + log\ \beta_t(q_t) - log\left(\sum_{q_t} exp[log(\alpha_t(q_t)\beta_t(q_t))]\right)\\
&= log\ \alpha_t(q_t) + log\ \beta_t(q_t)\\
&\quad - log\left(exp\left[\max_{q_t}(log\ \alpha_t(q_t)\beta_t(q_t))\right]\sum_{q_t} exp\left[log\left(\alpha_t(q_t)\beta_t(q_t) - \max_{q_t}\left(\alpha_t(q_t)\beta_t(q_t)\right)\right)\right]\right)\\
&= log\ \alpha_t(q_t) + log\ \beta_t(q_t)\\
&\quad - \left(\max_{q_t}(log\ \alpha_t(q_t) + log\ \beta_t(q_t))\right.\\
&\quad \left. +\ log\left(\sum_{q_t} exp\left[log\left(\alpha_t(q_t)\beta_t(q_t) - \max_{q_t}\left(\alpha_t(q_t)\beta_t(q_t)\right)\right)\right]\right)\right)
\end{aligned}$$

The function `smoothing(alpha, beta)` exactly use the above identity.

3. Derivation of the estimation equations of the EM algorithm As we are dealing with an HMM the complete log-likelihood is (where $\theta = (\pi, A, \{\mu_k, \Sigma_k : k = 1,\ldots,4\})$)

$$\begin{aligned}
\ell_c(\theta) &= log\left(p(q_0)\prod_{t=0}^{T-1} p(q_{t+1}|q_t)\prod_{t=0}^{T} p(u_t|q_t)\right)\\
&= log(p(q_0)) + \sum_{t=0}^{T-1} log\ p(q_{t+1}|q_t) + \sum_{t=0}^{T} log\ p(u_t|q_t)\\
&= \sum_{i=1}^{K}\delta(q_0 = i)log\ \pi_i + \sum_{t=0}^{T-1}\sum_{i,j}^{K}\delta(q_{t+1} = i, q_t = j)log(A_{i,j})\\
&\quad + \sum_{t=0}^{T}\sum_{i=1}^{K}\delta(q_t = i)log\left(\mathcal{N}(u_i,\Sigma_i)\right)\\
&= \sum_{i=1}^{K}\delta(q_0 = i)log\ \pi_i + \sum_{t=0}^{T-1}\sum_{i,j}^{K}\delta(q_{t+1} = i, q_t = j)log(A_{i,j})\\
&\quad - \frac{1}{2}\sum_{t=0}^{T}\sum_{i=1}^{K}\delta(q_t = i)\left(log|\Sigma_i| + (u_t - \mu_i)^T\Sigma_i^{-1}(u_t - \mu_i)\right) - \frac{T.K.d}{2}log\ 2\pi
\end{aligned}$$

E-Step: We want to compute the quantity $\mathbb{E}_{q|u}[\ell_c(\theta)]$ but as we have:

$$\mathbb{E}_{q|u}[\delta(q_o = i)] = \mathbb{P}[q_0 = i|u; \theta^{k-1}] \tag{1}$$
$$\mathbb{E}_{q|u}[\delta(q_{t+1} = i, q_t = j)] = \mathbb{P}[q_{t+1} = i, q_t = i|u; \theta^{k-1}] \tag{2}$$
$$\mathbb{E}_{q|u}[\delta(q_t = i)] = \mathbb{P}[q_t = i|u; \theta^{k-1}] \tag{3}$$

it comes:

$$\begin{aligned}\mathbb{E}_{q|u}[\ell_c(\theta)] = &\sum_{i=1}^{K} \mathbb{P}[q_0 = i|u; \theta^{k-1}] log\ \pi_i + \sum_{t=0}^{T-1}\sum_{i,j}^{K} \mathbb{P}[q_{i+1} = i, q_t = i|u; \theta^{k-1}] log\ A_{i,j} \\ &- \frac{1}{2}\sum_{t=0}^{T}\sum_{i=1}^{K} \mathbb{P}[q_t = i|u; \theta^{k-1}] \left(log|\Sigma_i| + (u_t - \mu_i)^T \Sigma_i^{-1}(u_t - \mu_i) \right) + cst\end{aligned}$$

M-Step: We want to maximize w.r.t each parameter. Recall that here $\theta = (\pi, A, \{\mu_k, \Sigma_k : k = 1, \ldots, 4\})$

π_k: $f : \pi_i \to \sum_{i=1}^{K} \mathbb{P}[q_0 = i|u; \theta^{k-1}] log\ \pi_i$ is strictly concave as a positive weighted sum of logarithm functions which are strictly concave on their domain. So we can recover the minimum by annulling the gradient of f w.r.t π_i. Using the condition $\sum_{i=1}^{K} \pi_i = 1$ we can write the Lagrangian as:

$$\mathcal{L}(\pi, \lambda) = \sum_{i=1}^{K} \mathbb{P}[q_0 = i|u; \theta^{k-1}] log\ \pi_i + \lambda \left(1 - \sum_{i=1}^{K} \pi_i \right)$$

from what follows:

$$\begin{aligned} \frac{\partial \mathcal{L}(\pi_i, \lambda)}{\partial \pi_i} &= 0 \\ \Leftrightarrow \frac{\mathbb{P}[q_0 = i|u; \theta^{k-1}]}{\pi_i} - \lambda &= 0 \\ \Leftrightarrow \lambda \pi_i &= \mathbb{P}[q_0 = i|u; \theta^{k-1}] \end{aligned}$$

Then using the condition $\sum_{i=1}^{K} \pi_i = 1$, it comes:

$$\lambda \underbrace{\sum_{i=1}^{K} \pi_i}_{=1} = \sum_{i=1}^{K} \mathbb{P}[q_0 = i|u; \theta^{k-1}] \triangleq 1$$

So finally:

$$\pi_i^k = \mathbb{P}[q_0 = i|u;\theta^{k-1}]$$

where π_i^k means the i^{th} component of π at time k.

A: We want to maximize $g : A_{i,j} \rightarrow \sum_{t=0}^{T-1}\sum_{i,j}^{K}\mathbb{P}[q_{t+1} = j, q_t = i|u;\theta^{k-1}]log\ A_{i,j}$ w.r.t $A_{i,j}$. g is a strictly concave function as a positive weighted sum of logarithm functions strictly concave on their domain. Using the condition: $\sum_{j=1}^{K} A_{i,j} = 1$ (Warning: $\sum_{i} A_{i,j} \neq 1$!!) we can write the Lagrangian as:

$$\mathcal{L}(A_{i,j},\lambda) = \sum_{t=0}^{T-1}\sum_{i,j=1}^{K}\mathbb{P}[q_{t+1} = j, q_t = i|u;\theta^{k-1}]log\ A_{i,j} + \lambda(1 - \sum_{j=1}^{K} A_{i,j})$$

We can recover the minimum by annulling the gradient of g as g is a strictly concave function on its domain:

$$\frac{\partial\mathcal{L}(A_{i,j},\lambda)}{\partial A_{i,j}} = \frac{\sum_{t=0}^{T-1}\mathbb{P}[q_{t+1} = j, q_t = i|u;\theta^{k-1}]}{A_{i,j}} - \lambda = 0$$

Then, using the condition $\sum_{j=1}^{K} A_{i,j} = 1$ it comes:

$$\lambda\underbrace{\sum_{j=1}^{K} A_{i,j}}_{=1} = \sum_{t=0}^{T-1}\sum_{j=1}^{K}\mathbb{P}[q_{t+1} = j, q_t = i|u;\theta^{k-1}]$$

So finally:

$$A_{i,j}^k = \frac{\sum_{t=0}^{T-1}\mathbb{P}[q_{t+1} = j, q_t = i|u;\theta^{k-1}]}{\lambda} = \frac{\sum_{t=0}^{T-1}\mathbb{P}[q_{t+1} = j, q_t = i|u;\theta^{k-1}]}{\sum_{t=0}^{T-1}\sum_{j'=1}^{K}\mathbb{P}[q_{t+1} = j', q_t = i|u;\theta^{k-1}]}$$

Where $A_{i,j}^k$ refers to the value of $A_{i,k}$ at time k.

mu_k: We want to maximize $h : \mu_k \to -\frac{1}{2}\sum_{t=0}^{T}\sum_{i=1}^{K}\mathbb{P}[q_t = i|u;\theta^{k-1}](u_t-\mu_i)^T\Sigma_i^{-1}(u_t-\mu_i)$ w.r.t μ_k, $\forall k \in \{1,2,3,4\}$. h is a concave function as it is the sum of the opposite of quadratic forms (Σ_i positive semi-definite and so do Σ^{-1}), so we can recover the maximum of h by annulling the gradient of h:

$$
\begin{aligned}
-\frac{1}{2}\nabla_{\mu_i}\left(\sum_{t=0}^{T}\sum_{i=1}^{K}\mathbb{P}[q_t = i|u;\theta^{k-1}](u_t-\mu_i)^T\Sigma_i^{-1}(u_t-\mu_i)\right) &= 0 \\
\Leftrightarrow \Sigma_i^{-1}\sum_{t=0}^{T}\mathbb{P}[q_t = i|u;\theta^{k-1}](u_t-\mu_i) &= 0 \\
\Leftrightarrow \mu_i &= \frac{\sum_{t=0}^{T}\mathbb{P}[q_t = i|u;\theta^{k-1}]u_t}{\sum_{t=0}^{T}\mathbb{P}[q_t = i|u;\theta^{k-1}]}
\end{aligned}
$$

where we have used the fact that Σ_i^{-1} is invertible $\forall i \in \{1,2,3,4\}$ in the last equivalence.

Hence, finally we have, $\forall i \in \{1,2,3,4\}$:

$$\mu_i^k = \frac{\sum_{t=0}^{T}\mathbb{P}[q_t = i|u;\theta^{k-1}]u_t}{\sum_{t=0}^{T}\mathbb{P}[q_t = i|u;\theta^{k-1}]}$$

Where μ_i^k stands for the i^{th} means at time k.

Σ_k: We have already seen this calculation several times in class and in the 2 previous homework, I won't detail the calculation anymore here as it is a bit cubersome. At the end we have:

$$\Sigma_i^k = \frac{\sum_{t=0}^{T}\mathbb{P}[q_t = i|u;\theta^{k-1}](u_t-\mu_i^k)(u_t-\mu_i^k)^T}{\sum_{t=0}^{T}\mathbb{P}[q_t = i|u;\theta^{k-1}]}$$

4. Expectation Maximization Implementation See python code

5. log-likelihood plots Using the parameter values computed by the EM algorithm of homework 2, we get the following plots for the log-likelihood:

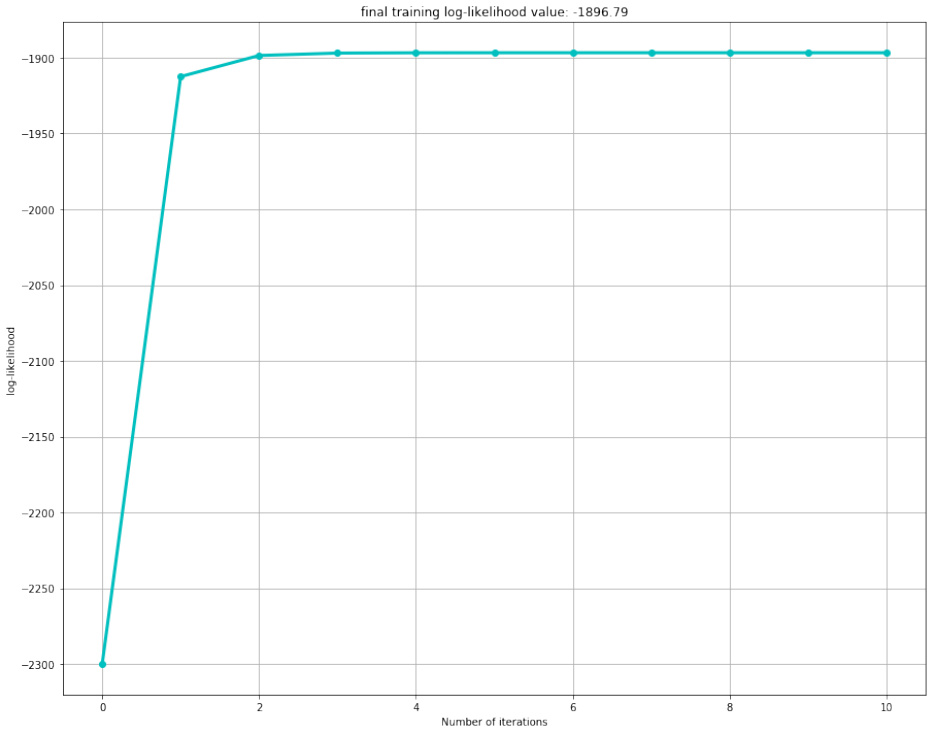

Figure 1.1: Log-likelihood plot on the training dataset using the Expectation Maximization algorithm on the HMM

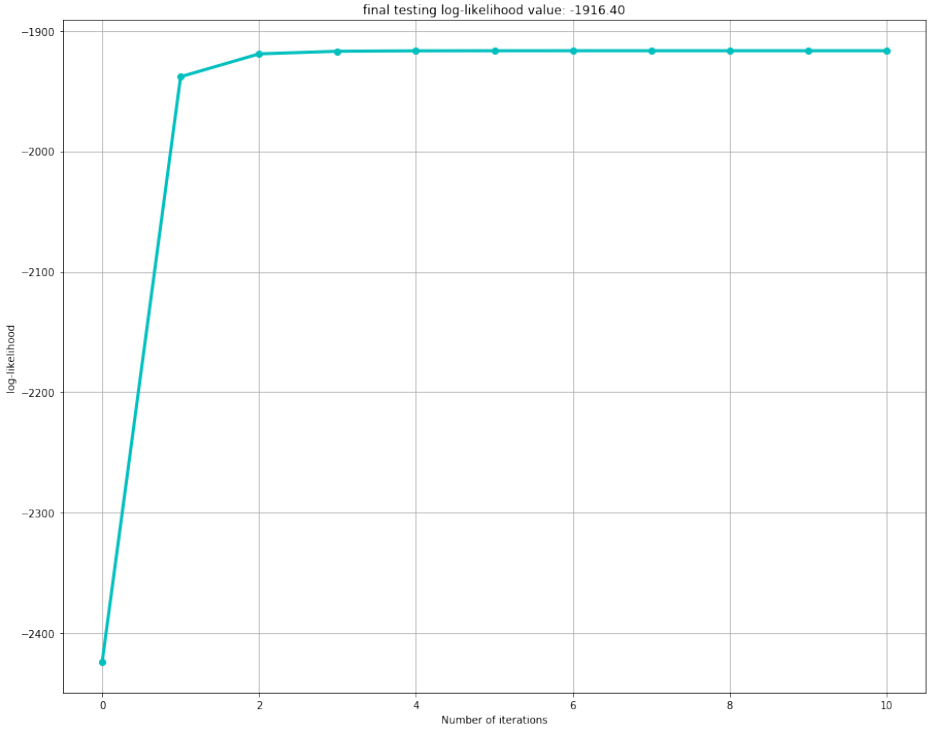

Figure 1.2: Log-likelihood plot on the testing dataset using the Expectation Maximization algorithm on the HMM

6. Comparison of the log-likelihoods The Table 1 show the different log-likelihood values we obtained using either a Gaussian Mixture Model or an HMM model.

	Gaussian mixture model	**Hidden Markov model**
Training set	-2327.72	-1896.79
Testing set	-2408.97	-1916.40

Table 1: Log-likehood obtained using either a Gaussian Mixture Model or an HMM

HMM outperforms the Gaussian Mixture Model because it works on a larger set. It is not a surprise that the log-likelihood is better with the HMM as the GMM can be seen as an HMM without the edges $y_t \to y_{t+1}$. In other words, HMM should perform better than GMM as it accounts for the same distribution + another set of transitions coming from $y_t \to y_{t+1}$. It makes sense to compare these models as both of them are Graphical models.

7. Viterbi algorithm The Viterbi algorithm estimates the most likely sequence of states, i.e $arg\max_{q_t} \mathbb{P}[q_1, \dots, q_T | y_1, \dots, u_T]$. It is the analog of filtering. It browses the sequence forward, calculating the message m at each step using the same equation as the one for the α recursion were we have replaced the summation by a maximization. Hence the Viterbi algorithm can be directly derive from the identity:

$$\max_{q_1,\dots,q_t} \mathbb{P}[q_1, \dots, q_t, q_{t+1} | u_1, \dots, u_{t+1}]$$
$$= \mathbb{P}[u_{t+1}|q_{t+1}] \max_{q_t} \left(\mathbb{P}[q_{t+1}|q_t] \max_{q_1,\dots,q_{t-1}} \mathbb{P}[q_1, \dots, q_{t-1}, q_t | u_1, \dots, u_t] \right)$$

8. Viterbi decoding See code for the implementation of Viterbi algorithm. The result of applying the Viterbi algorithm to the training dataset in shown on Figure 1.3

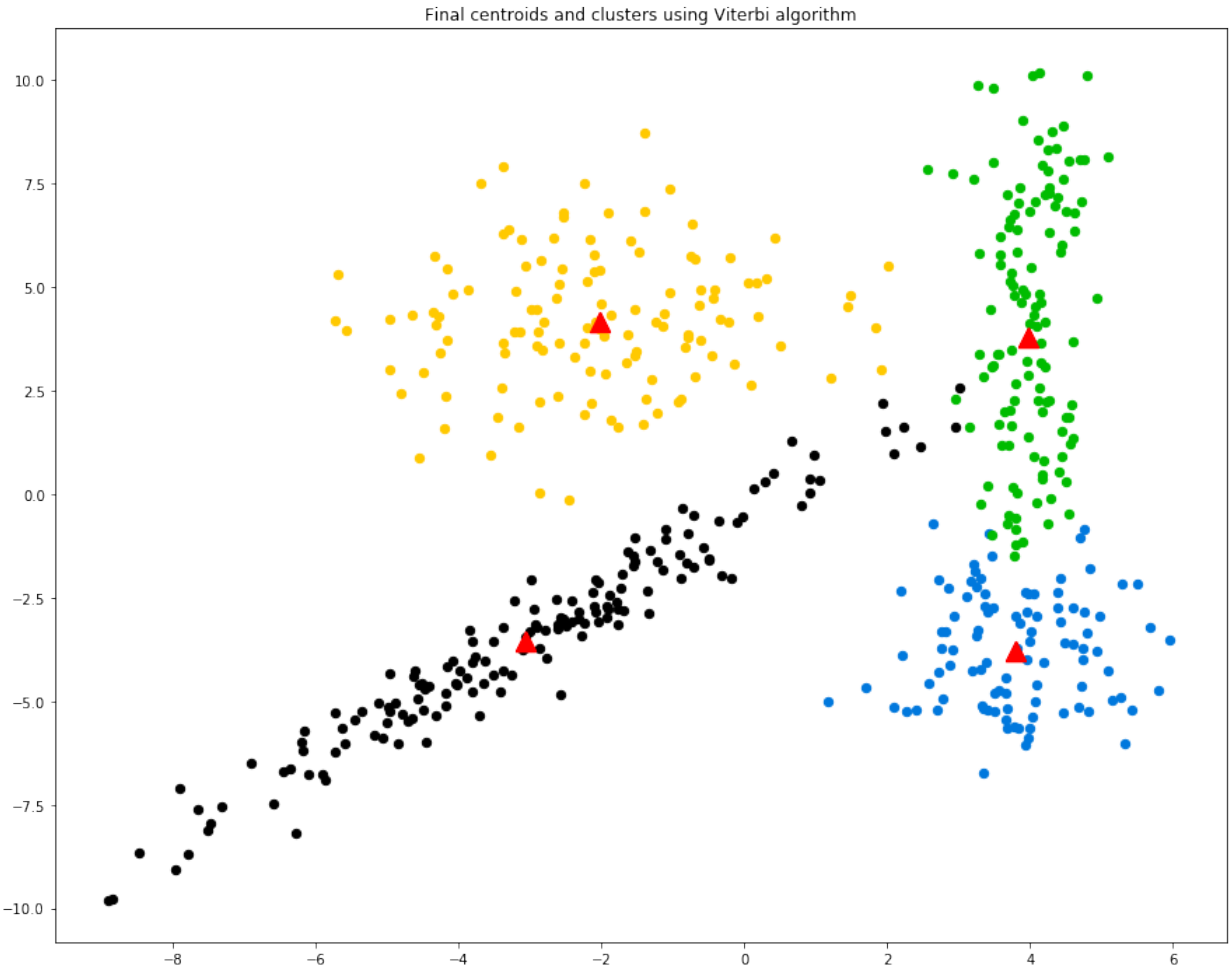

Figure 1.3: Final centroids and clusters using Viterbi algorithm on the training dataset

9. Probability plots of $\mathbb{P}[q_t|u_1, \ldots, u_T]$ for $T = 100$ The probability plots of $\mathbb{P}[q_t|u_1, \ldots, u_T]$ for $T = 100$ is shown on Figure 1.4

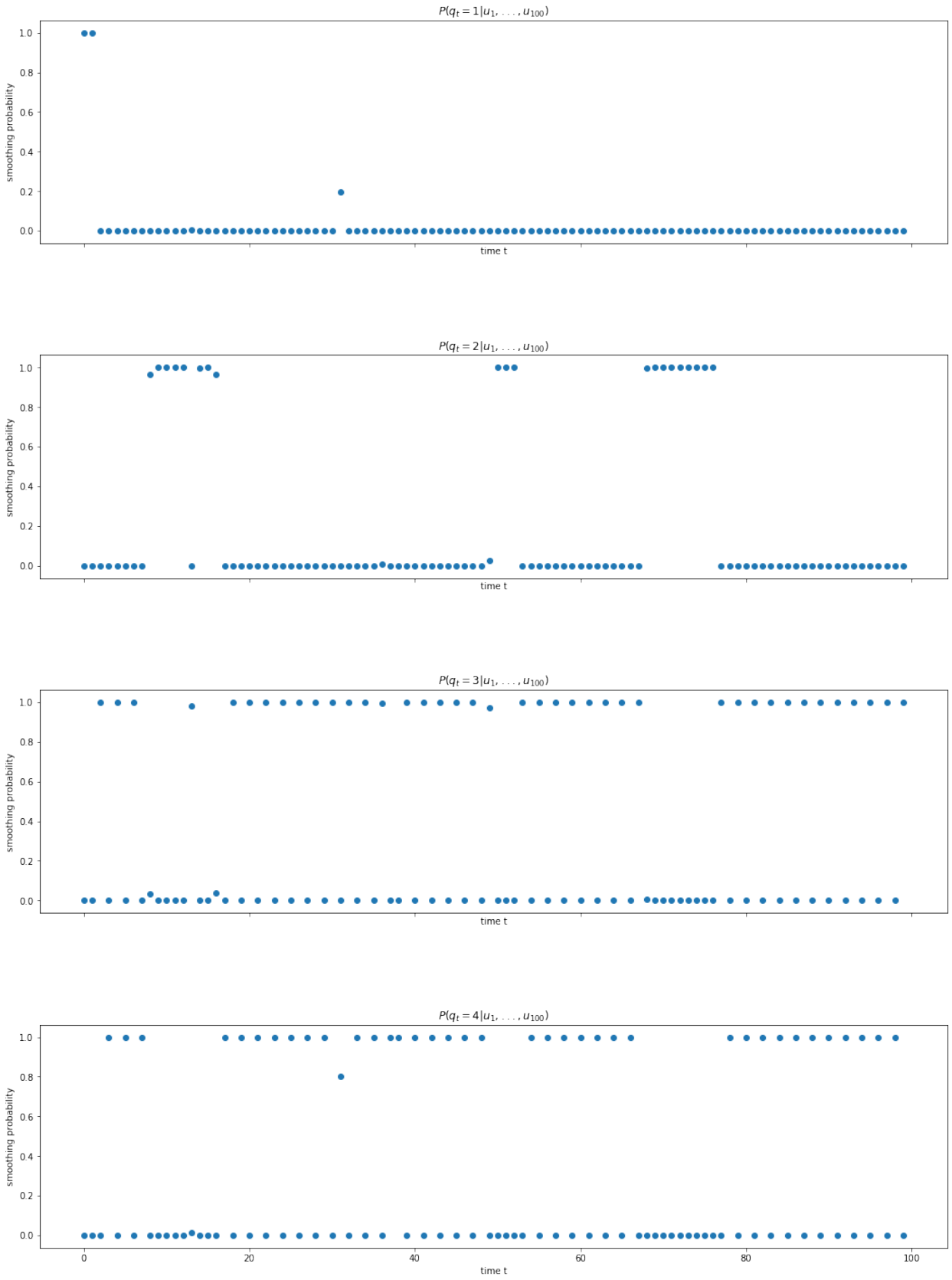

Figure 1.4: $\mathbb{P}[q_t|u_1, \ldots, u_T]$ for $T = 100$ for $q_t \in \{1, 2, 3, 4\}$

10. Most likely states of the previous points according to theirs marginal probabilities The most likely states according to theirs marginal probabilities are shown below

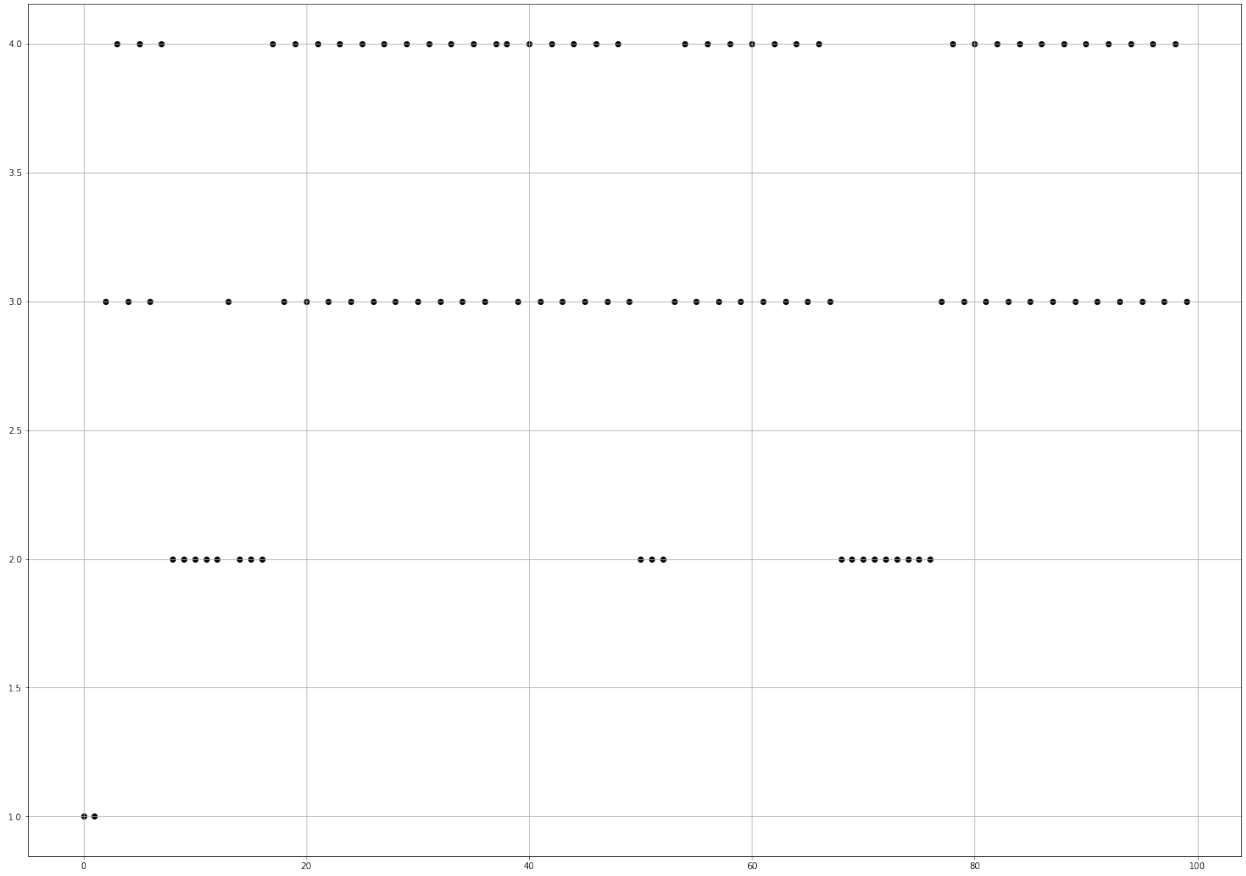

Figure 1.5: most probable state as a function of time for $\mathbb{P}[q_t|u_1, \ldots, u_T]$

11. Most likely state of the previous points using Viterbi algorithm The most likely state using Viterbi algorithm are shown below:

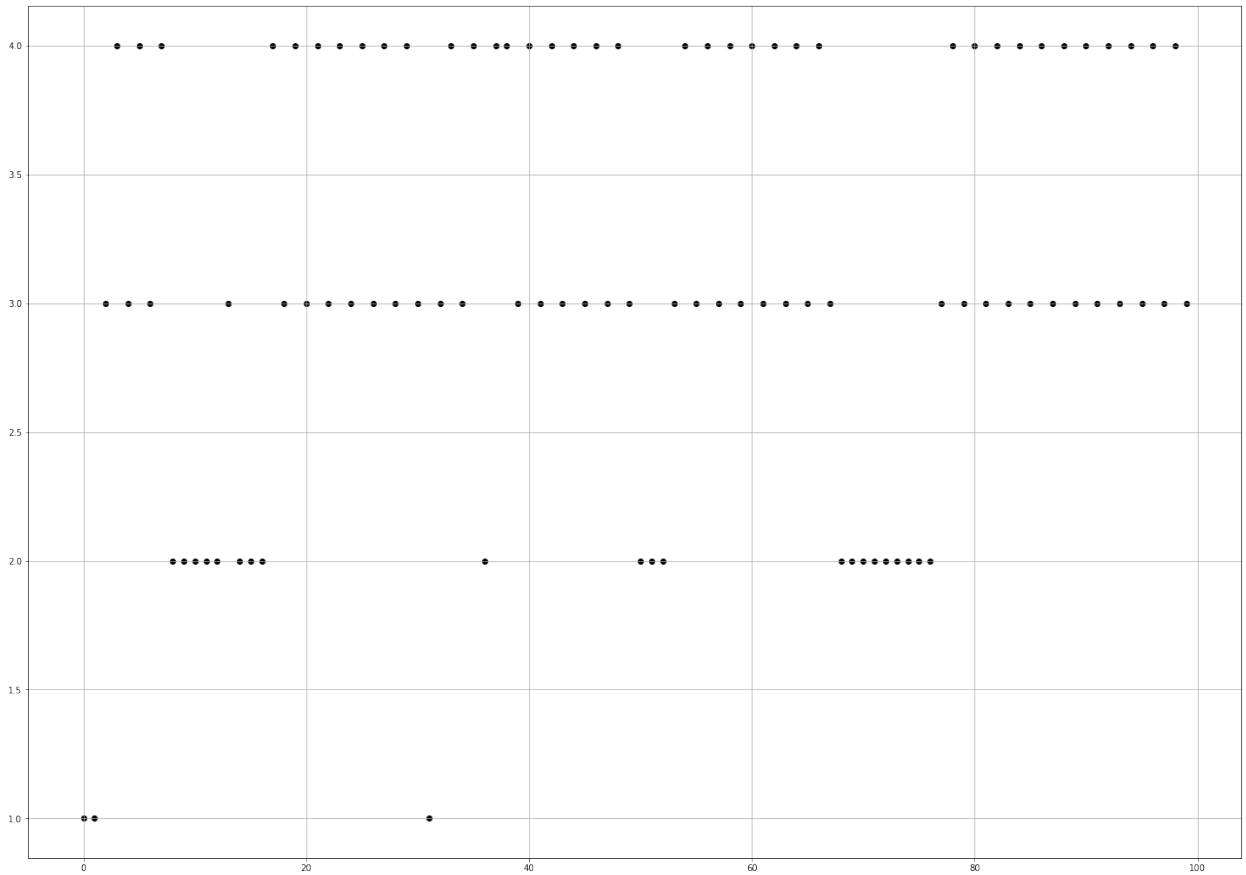

Figure 1.6: most probable state as a function of time for $\mathbb{P}[q_t|u_1, \ldots, u_T]$ using viterbi algorithm

As we can see on Figure 1.6, Viterbi algorithm and the previous question associate almost always the same classes to the data points. Using all the points (500), we can see that the Viterbi algorithm and the previous marginal computation associate the same classes to the data points 97% of the time (485/500) (See code).

12. How would you choose the number of states if you did not know it? If the number of states wasn't known I would have chosen K (the number of states) such that it maximizes the log-likelihood obtained by the Expectation Maximization algorithm.