PGM Homework 3
HMM Implementation

Bastien PONCHON
{bponchon}@ens-paris-saclay.fr

1 α and β recursion

In this section, I implemented the alpha-pass (or forward) and beta-pass (or backward) algorithm in order to compute the distributions $p(q_t|u_1...u_T)$ and $p(q_t, q_{t+1}|u_1, ..., u_T)$ for all $t \in [1, T]$.

1.1 α recursion

The alpha-recursion algorithm enables to compute the values $\alpha_t(q_t) = p(q_t, u_1, ..., u_t), \forall t, q_t \in [1, T] \times [1, K]$, using the following recursion:

$$\begin{aligned}
\forall q_1 \in [1, K], \alpha_1(q_1) &= p(q_1)p(u_1|q_1) = \pi(q_1)p(u_1|q_1) \\
\forall t, q_t \in [2, T] \times [1, K], \alpha_t(q_t) &= p(u_t|q_t) \sum_{q_{t-1}} p(q_t|q_{t-1})\alpha_{t-1}(q_{t-1}) \\
&= p(u_t|q_t) \sum_{q_{t-1}} A_{q_{t-1},q_t}\alpha_{t-1}(q_{t-1})
\end{aligned}$$

However, the implementation of this algorithms usually leads to underflow as the lengths of the sequence (T) increases, the values of α_t getting closer to zero.

In order to keep these values in the dynamic range of the machine, a scaled version of this algorithm has to be used:

$$\begin{aligned}
\forall q_1 \in [1,K], \alpha'_1(q_1) &= p(q_1)p(u_1|q_1) = \pi(q_1)p(u_1|q_1) \\
\hat{\alpha}_1(q_1) &= \frac{\alpha'_1(q_1)}{\sum_{q_1}\alpha'_1(q_1)} = \frac{\alpha'_1(q_1)}{c_1} \\
\forall t, q_t \in [2,T]\times[1,K], \alpha'_t(q_t) &= p(u_t|q_t)\sum_{q_{t-1}} A_{q_{t-1},q_t}\hat{\alpha}_{t-1}(q_{t-1}) \\
\hat{\alpha}_t(q_t) &= \frac{\alpha'_t(q_t)}{\sum_{q_t}\alpha'_t(q_t)} = \frac{\alpha'_t(q_t)}{c_t}
\end{aligned}$$

We can easily show that $\forall t, q_t \in [1,T]\times[1,K], \hat{\alpha}_t(q_t) = \frac{\alpha'_t(q_t)}{\sum_{q_t}\alpha'_t(q_t)} = \frac{\alpha_t(q_t)}{\sum_{q_t}\alpha_t(q_t)} = \frac{\alpha_t(q_t)}{\prod_{s=1}^{t} c_s} = p(q_t|u_1,...,u_t)$ which implies the following property of the scaling coefficients $c_t, t \in [1,T]$: $\forall t \in [1,T], \prod_{s=1}^{t} c_s = p(u_1,...,u_t)$, that is to say: $\forall t \in [2,T], c_t = p(u_t|u_1,...,u_{t-1})$ and $c_1 = p(u_1)$.

1.2 β recursion

The beta-recursion algorithm enables to compute the values $\beta_t(q_t) = p(u_{t+1},...,u_T|q_t), \forall t, q_t \in [1,T-1]\times[1,K]$, using the following recursion:

$$\begin{aligned}
\forall q_T \in [1,K], \beta_T(q_T) &= 1 \\
\forall t, q_t \in [1,T-1]\times[1,K], \beta_t(q_t) &= \sum_{q_{t+1}} p(q_{t+1}|q_t)p(u_{t+1}|q_{t+1})\beta_{t+1}(q_{t+1}) \\
&= \sum_{q_{t+1}} A_{q_t,q_{t+1}}p(u_{t+1}|q_{t+1})\beta_{t+1}(q_{t+1})
\end{aligned}$$

However, just like the forward algorithm, the beta-recursion is subject to underflow, and requires a scaled version to be implemented, using the very same scaling coefficients that have been computed in the scaled forward algorithm:

$$\begin{aligned}
\forall q_T \in [1,K], \hat{\beta}_T(q_T) &= \beta_T(q_T) = 1 \\
\forall t, q_t \in [1,T-1]\times[1,K], \beta'_t(q_t) &= \sum_{q_{t+1}} A_{q_t,q_{t+1}}p(u_{t+1}|q_{t+1})\hat{\beta}_{t+1}(q_{t+1}) \\
\hat{\beta}_t(q_t) &= \frac{\beta'_t(q_t)}{c_{t+1}}
\end{aligned}$$

We can easily show that $\forall t, q_t \in [1, T-1] \times [1, K], \hat{\beta}_t(q_t) = \frac{\beta_t(q_t)}{\prod_{s=t+1}^{T} c_s}$ with $\prod_{s=t+1}^{T} c_s = \frac{p(u_1,...,u_T)}{p(u_1,...,u_t)}$.

1.3 Computing $p(q_t|u_1, ..., u_T)$

In the non scaled version of the algorithm, we can easily compute $\gamma_t(q_t) = p(q_t|u_1, ..., u_T)$, using:

$$\begin{aligned}
p(q_t, u_1, ..., u_T) &= \alpha_t(q_t)\beta_t(q_t) \\
p(u_1, ..., u_T) &= \sum_{q_t} \alpha_t(q_t)\beta_t(q_t) \\
p(q_t|u_1, ..., u_T) &= \frac{p(q_t, u_1, ..., u_T)}{p(u_1, ..., u_T)} \\
&= \frac{\alpha_t(q_t)\beta_t(q_t)}{\sum_{q_t} \alpha_t(q_t)\beta_t(q_t)}
\end{aligned}$$

This is even simpler in the scaled version of the algorithm, as we have:

$$\begin{aligned}
\hat{\alpha}_t(q_t)\hat{\beta}_t(q_t) &= p(q_t|u_1, ..., u_t)p(u_{t+1}, ..., u_T|q_t)\frac{p(u_1, ..., u_t)}{p(u_1, ..., u_T)} \\
&= p(u_1, ..., u_t|q_t)p(u_{t+1}, ..., u_T|q_t)\frac{p(q_t)}{p(u_1, ..., u_T)} \\
&= p(u_1, ..., u_T|q_t)\frac{p(q_t)}{p(u_1, ..., u_T)} \\
&= p(q_t|u_1, ..., u_T)
\end{aligned}$$

1.4 Computing $p(q_t, q_{t+1}|u_1, ..., u_T)$

In the non-scaled version of the algorithm, we can compute the quantity $p(q_t, q_{t+1}|u_1, ..., u_T)$ using:

$$p(q_t, q_{t+1}|u_1, ..., u_T) = \frac{1}{p(u_1, ..., u_T)}\alpha_t(q_t)\beta_{t+1}(q_{t+1})p(q_{t+1}|q_t)p(u_{t+1}|q_{t+1})$$

In the scaled version, this formula becomes:

$$\begin{aligned} p(q_t, q_{t+1}|u_1, ..., u_T) &= \frac{1}{p(u_1, ..., u_T)} \alpha_t(q_t)\beta_{t+1}(q_{t+1})p(q_{t+1}|q_t)p(u_{t+1}|q_{t+1}) \\ &= \frac{\beta_t(q_t)}{\beta_t(q_t)} \frac{\alpha_t(q_t)}{\sum_{q_t} \alpha_t(q_t)\beta_t(q_t)} \beta_{t+1}(q_{t+1})p(q_{t+1}|q_t)p(u_{t+1}|q_{t+1}) \\ &= \hat{\alpha}_t(q_t)\hat{\beta}_t(q_t)\frac{\beta_{t+1}(q_{t+1})}{\beta_t(q_t)} p(q_{t+1}|q_t)p(u_{t+1}|q_{t+1}) \\ &= \hat{\alpha}_t(q_t)\hat{\beta}_t(q_t)\frac{\hat{\beta}_{t+1}(q_{t+1}) \prod_{s=t+2}^{T} c_s}{\hat{\beta}_t(q_t) \prod_{s=t+1}^{T} c_s} p(q_{t+1}|q_t)p(u_{t+1}|q_{t+1}) \\ &= \frac{\hat{\alpha}_t(q_t)\hat{\beta}_{t+1}(q_{t+1})}{c_{t+1}} p(q_{t+1}|q_t)p(u_{t+1}|q_{t+1}) \end{aligned}$$

2 Testing and plotting

After implementing the algorithms both in scaled and non-scaled version (and noticing that we indeed note that the non-scaled forward and backward algorithms lead to underflow), we plotted the probability $p(q_t|u_1, ..., u_T)$ as a function of time for each of the 4 possible states q_t and for the 100 first data points.

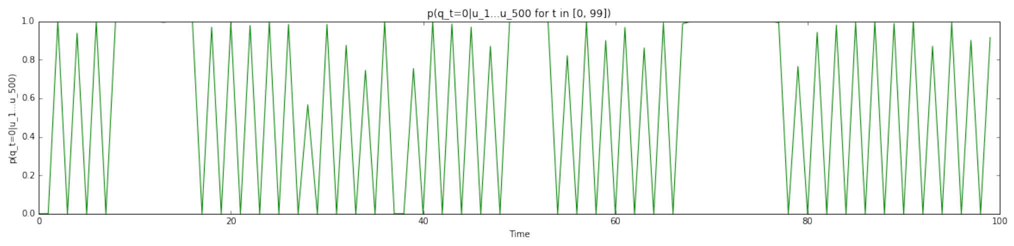

Figure 1: $p(q_t = 0|u_1, ..., u_{500})$ for $t \in [0, 99]$

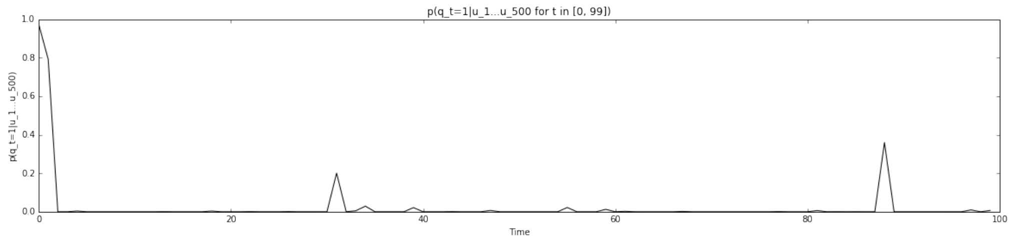

Figure 2: $p(q_t = 1|u_1, ..., u_{500})$ for $t \in [0, 99]$

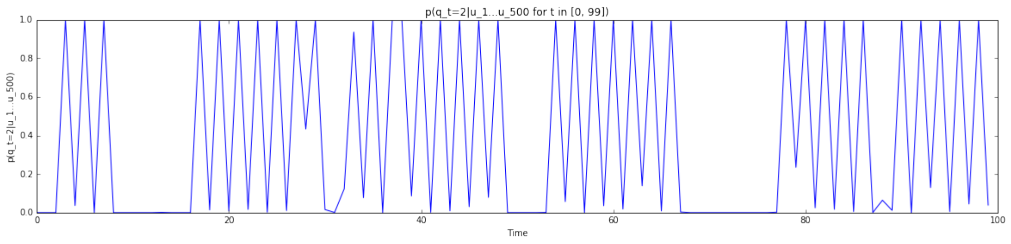

Figure 3: $p(q_t = 3|u_1, ..., u_{500})$ for $t \in [0, 99]$

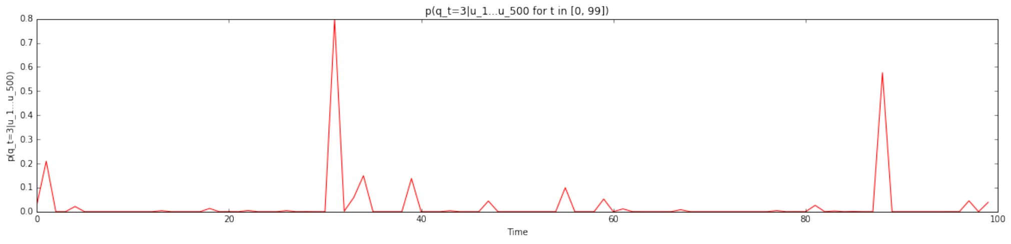

Figure 4: $p(q_t = 3|u_1, ..., u_{500})$ for $t \in [0, 99]$

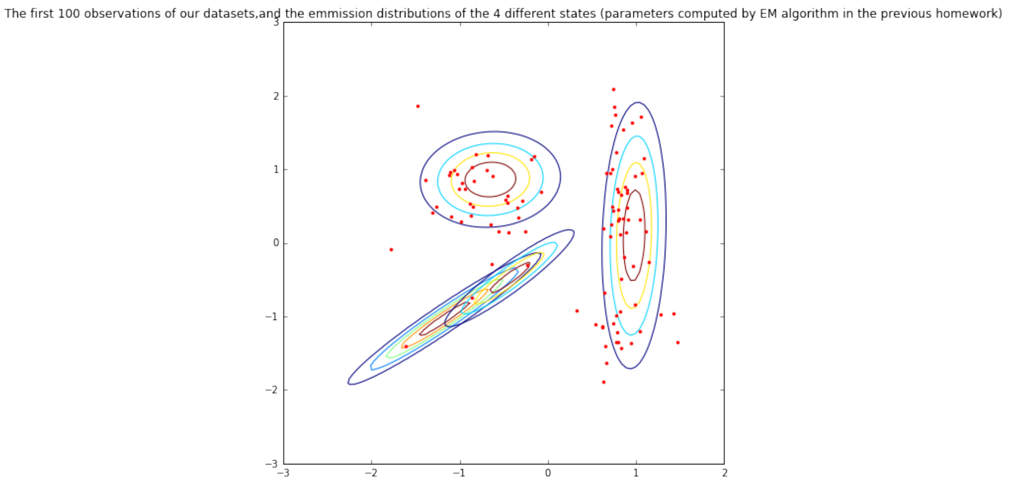

Figure 5: The first 100 points of the data set

3 EM algorithm

Let us derive the estimation equation of the EM algorithm for our HMM model. In our case, the hidden variables are the state $q_1, ..., q_T$ that generated each data point $u_1, ..., u_T$,and the parameters we want to infer are $\theta = (\pi, A, (\mu_k, \Sigma_k)\forall k \in [1, 4])$.

For the sake of simplicity, we will drop the index representing the current iteration of the EM algorithm, in order not to confuse it with the state index k or the data point index t.

Let first derive the complete log likelihood $l_c(\theta) = log\left(p_\theta(u_1, ..., u_T, q_1, ..., q_T)\right)$:

$$
\begin{aligned}
l_c(\theta) =& log\left(p(q_0)\prod_{t=1}^{T-1} p(q_{t+1}|q_t)\prod_{t=0}^{T} p(u_t|q_t)\right) \\
=& log(p(q_0)) + \sum_{t=1}^{T-1} log(p(q_{t+1}|q_t)) + \sum_{t=1}^{T} log(p(u_t|q_t)) \\
=& \sum_{k=1}^{4} \delta(q_0 = k) log(\pi_k) + \sum_{t=1}^{T-1}\sum_{i,j=1}^{4} \delta(q_t = i, q_{t+1} = j) log(A_{i,j}) \\
& + \sum_{t=1}^{T}\sum_{k=1}^{4} \delta(q_t = k) log(\mathcal{N}(u_t|\mu_k, \Sigma_k)) \\
=& \sum_{k=1}^{4} \delta(q_0 = k) log(\pi_k) + \sum_{t=1}^{T-1}\sum_{i,j=1}^{4} \delta(q_t = i, q_{t+1} = j) log(A_{i,j}) \\
& - T log(2\pi) - \frac{1}{2}\sum_{t=1}^{T}\sum_{k=1}^{4} \delta(q_t = k) log(|\Sigma_k|) \\
& - \frac{1}{2}\sum_{t=1}^{T}\sum_{k=1}^{4} \delta(q_t = k)(u_t - \mu_k)^T \Sigma_k^{-1}(u_t - \mu_k)
\end{aligned}
$$

At each iteration, we do:

(i) First compute the probability of our hidden variables given our data. In our case, the hidden variables are the state $q_1, ..., q_T$ that generated each data point $u_1, ..., u_T$. The quantity we want to derive is thus $p(q_1, ..., q_T|u_1, ..., u_T)$, but as we can guess from the expression of the complete log likelihood we just derived, we actually only need to compute $p(q_t|u_1, ..., u_T)$ and $p(q_t, q_{t+1}|u_1, ..., u_T)$. Fortunately, we have just seen in an algorithm enabling us to compute these values for all t, q_t, and q_{t+1} in the first section.

(ii) **E-Step** : calculate the expected value of the complete log likelihood function, with respect to the conditional distribution of $q_1, ..., q_T$ given $u_1, ..., u_T$ under the current estimate of the parameter θ. In our case, this boils down to replacing $\delta(q_t = k)$ and $\delta(q_t = i, q_{t+1} = j)$ by $p(q_t = k|u_1, ..., u_T)$ and $p(q_t = i, q_{t+1} = j|u_1, ..., u_T)$ respectively, in the previous expression of the log likelihood.

(iii) **M-Step** : Maximize the previous expression of the expected value of the complete log likelihood function with regards to the parameter θ:

(i) π : we want to maximize : $\sum_{k=1}^{4} p(q_0 = k|u_1, ..., u_T)log(\pi_k)$ with regards to π, with the constraint $\sum_{k=1}^{4} \pi_k = 1$. We solve this by introducing a Lagrange factor, and get:

$$\forall k \in [1, 4], \pi_k = p(q_0 = k|u_1, ..., u_T)$$

(ii) A : we want to maximize : $\sum_{t=1}^{T-1} \sum_{i,j=1}^{4} p(q_t = i, q_{t+1} = j|u_1, ..., u_T)log(A_{i,j})$ with regards to A, with the 4 constraints $\sum_{j=1}^{4} A_{i,j} = 1, \forall i \in [1, 4]$. We solve this by introducing 4 Lagrange factors, and get:

$$\forall (i, j) \in [1, 4]^2, A_{i,j} = \frac{\sum_{t=1}^{T-1} p(q_t = i, q_{t+1} = j|u_1, ..., u_T)}{\sum_{t=1}^{T-1} p(q_t = i|u_1, ..., u_T)}$$

(iii) μ : we want to maximize :

$$-\frac{1}{2}\sum_{t=1}^{T}\sum_{k=1}^{4} p(q_t = k|u_1, ..., u_T)(u_t - \mu_k)^T\Sigma_k^{-1}(u_t - \mu_k)$$
$$= -\frac{1}{2}\sum_{t=1}^{T}\sum_{k=1}^{4} p(q_t = k|u_1, ..., u_T)(u_t^T\Sigma_k^{-1}u_t + \mu_k^T\Sigma_k^{-1}\mu_k - 2u_t^T\Sigma_k^{-1}\mu_k)$$

with regards to μ. We get:

$$\forall k \in [1, 4], \mu_k = \frac{\sum_{t=1}^{T} p(q_t = k|u_1, ..., u_T)u_t}{\sum_{t=1}^{T} p(q_t = k|u_1, ..., u_T)}$$

(iv) Σ : we want to maximize :

$$-\frac{1}{2}\sum_{t=1}^{T}\sum_{k=1}^{4} p(q_t = k|u_1, ..., u_T)log(|\Sigma_k|)$$
$$-\frac{1}{2}\sum_{t=1}^{T}\sum_{k=1}^{4} p(q_t = k|u_1, ..., u_T)(u_t - \mu_k)^T\Sigma_k^{-1}(u_t - \mu_k)$$

with regards to Σ. We get:

$$\forall k \in [1, 4], \Sigma_k = \frac{\sum_{t=1}^{T} p(q_t = k|u_1, ..., u_T)(u_t - \mu_k)(u_t - \mu_k)^T}{\sum_{t=1}^{T} p(q_t = k|u_1, ..., u_T)}$$

4 Implementation

I implemented the EM algorithm for our HMM model, initializing the means and covariances with the ones obtained in the previous homework.The model is trained from the training data in 'EMGaussienne.dat'.

5 Plotting the log-likelihood on the train and test data as a function of the iteration

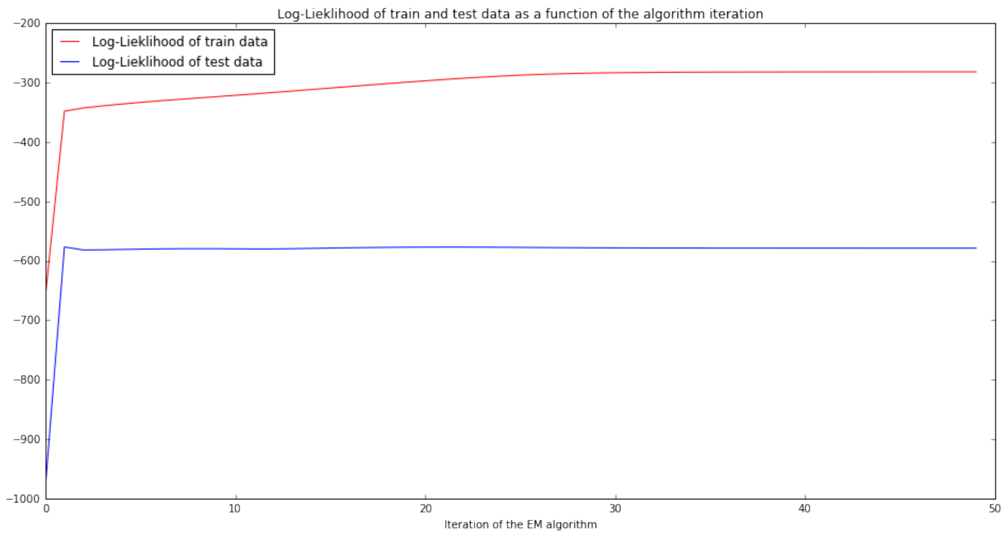

Figure 6: Log-likelihood on the train and test data as a function of the iteration

We can see in Figure 6 that, as expected, the log-likelihood on train and test data increase at each iteration of the EM algorithm to reach an asymptotic limit (as the EM algorithm converges). We can also note that, logically enough, the log-likelihood of the test data remains smaller that the log-likelihood on the train data, as the model has been trained on the the latter.

6 Comparing with the Gaussian Mixture Model

Let us report the log-likelihood of the Test and Train data sets at the 10th iteration of the EM algorithm for our three models:

Model	Train Set	Test Set
Isotropic Mixture of Gaussian	-1615.98	-1707.76
General Mixture of Gaussian	-462.913	-761.764
HMM	-323.565	-579.546
HMM on shuffled version of Test set		-1384.247

We cannot really compare Gaussian Mixture and HMM. Indeed, in the latter the sequence in which the observations were emitted has been taken into account, while this is not the case the GMM. Hence we can assume that the HMM model overfitted the order of the sequence, as we can see in the last line of the table.

7 Viterbi Algorithm

The Viterbi Decoder estimates the most likely sequence of states, that is to say $argmax_q p(q_1, ..., q_T|u_1, ..., u_T)$ by computing the values $\delta_t(q_t) = max_{q_1,...,q_{t-1}} p(q_1, ..., q_t, u_1, ..., u_t)$ and keeping track of the state sequence producing each maximum, using the following recursion:

(i) $\forall q_1 \in [1, K], \delta_1(q_1) = p(q_1, u_1) = p(u_1|q_1)p(q_1) = p(u_1|q_1)\pi_{q_1}$

(ii) $\forall t, q_t \in [2, T] \times [1, K], \delta_t(q_t) = p(u_t|q_t) * max_{q_{t-1}} \delta_{t-1}(q_{t-1})p(q_t|q_{t-1})$ and keep track of the previous state that produced this maximum: $z_t(q_t) = argmax_{q_{t-1}} \delta_{t-1}(q_{t-1})p(q_t|q_{t-1})$

(iii) The probability of the best path is $max_q p(q_1, ..., q_T, u_1, ..., u_T) = max_{q_T} \delta_T(q_T)$. And we have the final state of the most likely state sequence: $\hat{q}_T = argmax_{q_T} \delta_T(q_T)$

(iv) Trace back all the states of the most likely sequence: $\forall t \in [1, T - 1], \hat{q}_t = z_{t+1}(\hat{q}_{t+1})$

Once again, we have to implement a modified version of Viterbi algorithm in order to avoid underflow. This can be done by adding the logarithm of the various probabilities instead of multiplying them, since there are no additions in the original algorithm (unlike in the forward and backward algorithms) and since logarithm is a increasing function (and therefore preserves the maximum).

The logarithmic version of the Viterbi algorithm is thus:

(i) $\forall q_1 \in [1, K], \hat{\delta}_1(q_1) = log(\delta_1(q_1)) = log(p(q_1, u_1)) = log(p(u_1|q_1)) + log(p(q_1)) = log(p(u_1|q_1)) + log(\pi_{q_1})$

(ii)

$$\begin{aligned}\forall t, q_t \in [2,T]\times[1,K], \hat{\delta}_t(q_t) =& log(\delta_t(q_t))\\ =& log(p(u_t|q_t))\\ &+ max_{q_{t-1}}\left(\hat{\delta}_{t-1}(q_{t-1}) + log(p(q_t|q_{t-1}))\right)\end{aligned}$$

and keep track of the previous state that produced this maximum: $z_t(q_t) = argmax_{q_{t-1}}\left(\hat{\delta}_{t-1}(q_{t-1}) + log(p(q_t|q_{t-1}))\right)$

(iii) The probability of the best path is $max_q log(p(q_1,...,q_T,u_1,...,u_T)) = max_{q_T}\hat{\delta}_T(q_T)$. And we have the final state of the most likely state sequence: $\hat{q}_T = argmax_{q_T}\hat{\delta}_T(q_T)$

(iv) Trace back all the states of the most likely sequence: $\forall t \in [1, T-1], \hat{q}_t = z_{t+1}(\hat{q}_{t+1})$

8 Implementation and test

We run our Viterbi algorithm on our test data with the hmm trained on the training data using the EM algorithm. We get the following assignments (Figure 7). Only a few points got assigned to the 4th cluster.

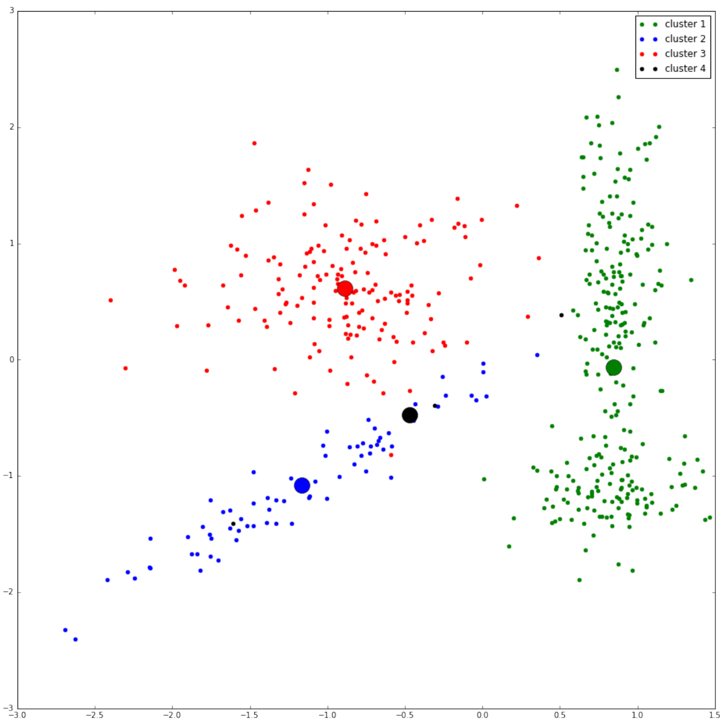

Figure 7: Viterbi decoding of our Test Data with an HMM trained with the EM algorithm on the train data

9 Plotting the marginal probability of each states with the EM-trained model

Plotting again $p(q_t|u_1, ..., u_T) \forall t \in [0, 99], q_t \in [1, 4]$ for the first 100 points of the test data, but, this time, for our hmm trained with the EM algorithm on the train data.

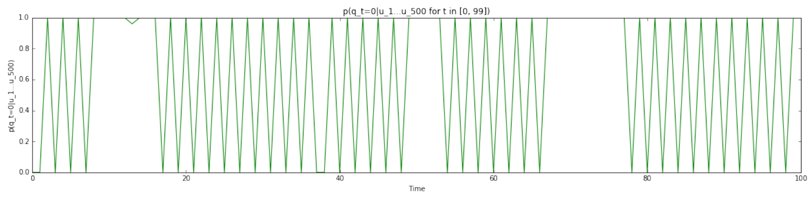

Figure 8: $p(q_t = 0|u_1, ..., u_{500})$ for $t \in [0, 99]$

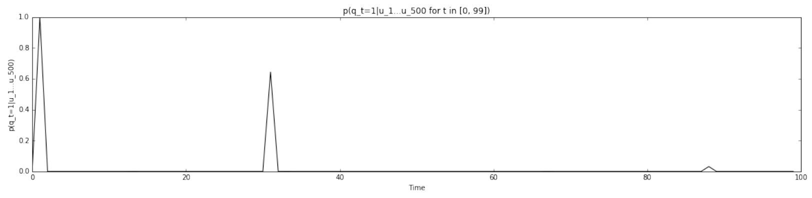

Figure 9: $p(q_t = 1|u_1, ..., u_{500})$ for $t \in [0, 99]$

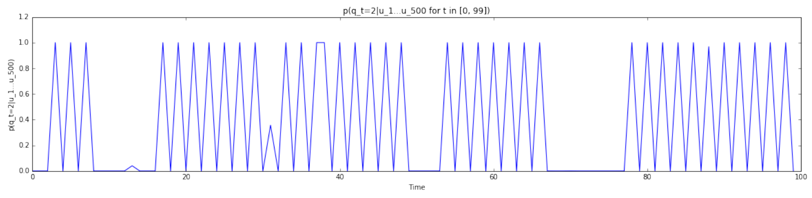

Figure 10: $p(q_t = 3|u_1, ..., u_{500})$ for $t \in [0, 99]$

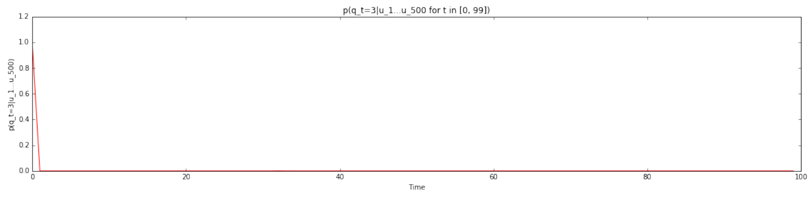

Figure 11: $p(q_t = 3|u_1, ..., u_{500})$ for $t \in [0, 99]$

When comparing with the corresponding plots in question 2, we can note that the trained model leads to marginal probabilities closer to 0 and 1 and hence to less points assigned to the cluster 2 and 4 in the 100 first data points.

10 Most Likely states according to the marginal probability

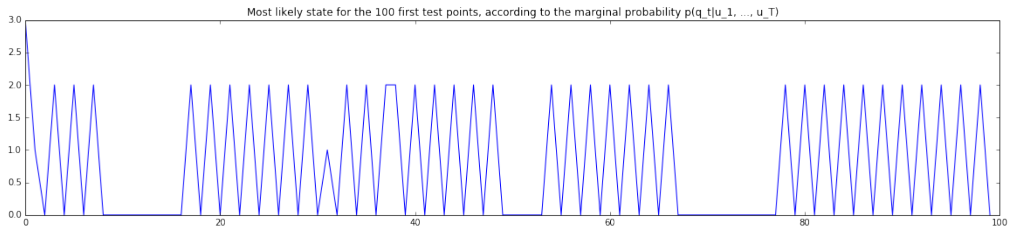

Figure 12: Most likely state for the 100 first test points, according to the marginal probability $p(q_t|u_1, ..., u_T)$

11 Most likely state according to Viterbi decoding

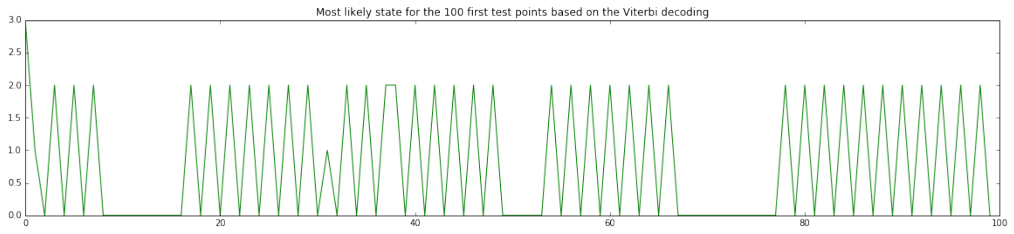

Figure 13: Most likely state for the 100 first test points, according to Viterbi decoding

We can see in Figure 12 and Figure 13 that both the gamma pass algorithm (computing the marginal probability $p(q_t|u_1, ..., u_T)$ and detailled in the first section) and the Viterbi decoding leads to the same assignment of data points to cluster.

12 Number of states

If the number of states is not known, we can try different number, and compare the results doing a cross validation.