Homework 3

Romain Vial

romain.vial@ens-paris-saclay.fr

HMM - Implementation

We have sequential data of the form $u_t = (x_t, y_t)$ for $t = 1 \dots T$. We consider the following HMM model: the chain $(q_t)_{t=1\dots T}$ has 4 possible states with an initial probability distribution π and a probability transition matrix A. Conditionally on the current state, the observations are obtained from a Gaussian distribution $u_t | q_t = k \sim \mathcal{N}(\mu_k, \Sigma_k)$.

Q1. Implement the recursions α et β seen in class to compute $p(q_t|u_1, \dots, u_T)$ and $p(q_t, q_{t+1}|u_1, \dots, u_T)$.

Cf. code. We use the same notation as in the polycopié, i.e. $\gamma(q_t) = p(q_t|u_1, \dots, u_T)$ and $\xi(q_t, q_{t+1}) = p(q_t, q_{t+1}|u_1, \dots, u_T)$.

Q2. Represent $p(q_t|u_1, \dots, u_T)$ for each of the 4 states as a function of time for the 100 first datapoints in the file.

For this question, we set π to be the uniform probability distribution over the 4 states and $A = \frac{1}{6} \times \begin{pmatrix} 3 & 1 & 1 & 1 \\ 1 & 3 & 1 & 1 \\ 1 & 1 & 3 & 1 \\ 1 & 1 & 1 & 3 \end{pmatrix}$. The μ_k and Σ_k are initialized with the ones obtained with a GMM model.

Figure 1 shows $\gamma(q_t = k)$ for $k \in \{1, 2, 3, 4\}$ as a function of time for the 100 first datapoints in the test file before fitting the parameters. One can see that the algorithm predicts with a score very close to 1 most of the points. Nevertheless, some points are more ambiguous, e.g. the point with index 35.

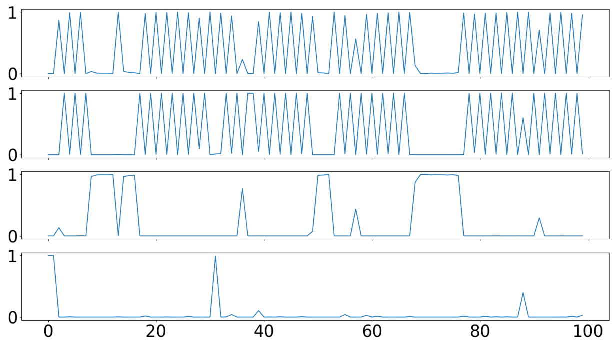


Figure 1: Representation of $\gamma(q_t = k)$ for $k \in \{1,2,3,4\}$ as a function of time for the 100 first datapoints in the test file before fitting the parameters.

Q3. Derive the estimation equations of the EM algorithm.

The complete log-likelihood can be written as follows:

$$\begin{aligned}
\log p(u,q) &= \log\left(p(q_0)\prod_{t=0}^{T}p(u_t|q_t)\prod_{t=1}^{T}p(q_t|q_{t-1})\right)\\
&= \log p(q_0) + \sum_{t=0}^{T}\log p(u_t|q_t) + \sum_{t=1}^{T}\log p(q_t|q_{t-1})\\
&= \sum_{k=1}^{K}\delta_{q_0=k}\log\pi_k + \sum_{k=1}^{K}\sum_{t=0}^{T}\delta_{q_t=k}\log\mathcal{N}(u_t|\mu_k,\Sigma_k)+\\
&\sum_{k=1}^{K}\sum_{k'=1}^{K}\sum_{t=1}^{T}\delta_{q_t=k,q_{t-1}=k'}\log A_{k'k}
\end{aligned}$$

Then, let note that:

$$\begin{aligned}
\mathbb{E}_{q|u}[\delta_{q_0=k}] &= \mathbb{E}[\delta_{q_0=k}|\bar{u}]\\
&= p(q_0=k|\bar{u})\\
&= \gamma(q_0=k)\\
\mathbb{E}_{q|u}[\delta_{q_t=k}] &= \gamma(q_t=k)\\
\mathbb{E}_{q|u}[\delta_{q_t=k,q_{t-1}=k'}] &= \xi(q_{t-1}=k',q_t=k)
\end{aligned}$$

The expectation step follows immediately:

$$\begin{aligned}
\mathbb{E}_{q|u}[\log p(u,q)] = &\sum_{k=1}^{K}\gamma(q_0=k)\log\pi_k + \sum_{k=1}^{K}\sum_{t=0}^{T}\gamma(q_t=k)\log\mathcal{N}(u_t|\mu_k,\Sigma_k)+\\
&\sum_{k=1}^{K}\sum_{k'=1}^{K}\sum_{t=1}^{T}\xi(q_{t-1}=k',q_t=k)\log A_{k'k}
\end{aligned}$$

As everything decouples, the maximization step finally gives us:

$$\begin{aligned}
\pi_k =& \gamma(q_0 = k) \\
A_{k'k} =& \frac{\sum_{t=1}^{T} \xi(q_{t-1} = k', q_t = k)}{\sum_{t=1}^{T} \sum_{k=1}^{K} \xi(q_{t-1} = k', q_t = k)} \\
\mu_k =& \frac{\sum_{t=0}^{T} \gamma(q_t = k) u_t}{\sum_{t=0}^{T} \gamma(q_t = k)} \\
\Sigma_k =& \frac{\sum_{t=0}^{T} \gamma(q_t = k)(u_t - \mu_k)(u_t - \mu_k)^\intercal}{\sum_{t=0}^{T} \gamma(q_t = k)}
\end{aligned}$$

Q4. Implement the EM algorithm to learn the parameters of the model.

Cf. code

Q5. Plot the log-likelihood on the train data and on the test data as a function of the iterations of the algorithm. Comment.

Figure 2 shows the log-likelihood against the number of iterations of EM. One can see that the algorithm converges quite fast in approximately 2 iterations. The test log-likelihood is higher than the train log-likelihood as the parameters hasn't been fitted on this dataset.

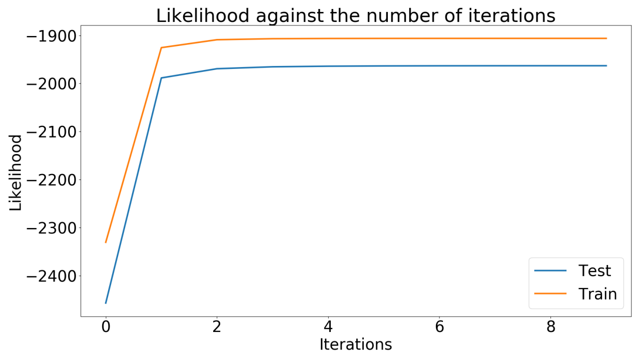


Figure 2: Log-likelihood on the train data and on the test data as a function of the iterations of the algorithm.

Q6. Return in a table the values of the log-likelihoods of the Gaussian mixture models and of the HMM on the train and on the test data. Compare these values. Does it make sense to make this comparison ? Conclude.

Table 1 shows the log-likelihoods of the GMM and of the HMM on the train and on the test data. One can see that the HMM has the highest log-likelihood for both the train and the test data. The situation is logical as the HMM is a more complex and more flexible model than the GMM, hence it is more effective at capturing the underlying distribution of the data. In addition, the test log-likelihood is also increasing which means that we haven't overfit the data yet.

Model	Train	Test
GMM - spherical	-2645.68	-2690.65
GMM - general	-2327.72	-2408.98
HMM	**-1905.97**	**-1962.83**

Table 1: Log-likelihoods of the GMM and of the HMM on the train and on the test data.

The comparison between the log-likelihoods make sense in this case because we are comparing the likelihood of the data, i.e. $p(y)$, and not the complete likelihood, i.e. $p(y, z)$. The only new assumptions that we are making in the HMM lies in the definition of z as a chain, hence y still refers to the same object in both cases.

Q7. Provide a description and pseudo-code for the Viterbi decoding algorithm that estimates the most likely sequence of states.

Algorithm 1 shows a pseudo-code for the Viterbi decoding algorithm. It first stores all possible initializations for the paths. For example, in our case all the possible paths begin with a state in $\{1, 2, 3, 4\}$. Hence we store these four initial paths along with their log-likelihood value. Then, we loop among the data and update the initial paths with the states that maximize their log-likelihood. At the end, we select the path that has the highest log-likelihood value.

Algorithm 1: Pseudo-code for the Viterbi decoding algorithm

```
Data: Learned parameters $\pi, A, \mu_k, \Sigma_k$ and data $u_0, \ldots, u_T$
Result: Best sequence $[q_0, \ldots, q_T]$
/* Store all the possible initializations for the paths */
for $k$=1...$K$ do
    paths[k] = {'ll': $\log(\pi_k \times \mathcal{N}(u_0|\mu_k, \Sigma_k))$, 'states': [k]}
/* Update all the initial paths with the states maximizing their
   log-likelihood */
for $t$=1...$T$ do
    for $k$=1...$K$ do
        k', prev_ll= paths[k]['states'][-1], paths[k]['ll']
        next_state = $\arg\max_{k''} [\log \mathcal{N}(u_t|\mu_k, \Sigma_k) + \log A_{k'k''} + prev\_ll]$
        next_ll = $\max_{k''} [\log \mathcal{N}(u_t|\mu_k, \Sigma_k) + \log A_{k'k''} + prev\_ll]$
        paths[k] = {'ll'+= next_ll, 'states'+= [next_state]}
Return the list of states among paths which has the highest 'log-value'.
```

Q8. Implement Viterbi decoding. Represent the data in 2D with the cluster centers and with markers of different colors for the datapoints belonging to different classes.

Cf. code for the implementation. Figure 3 shows the most likely sequence of states along with the different learned parameters (cluster centers and covariance matrices).

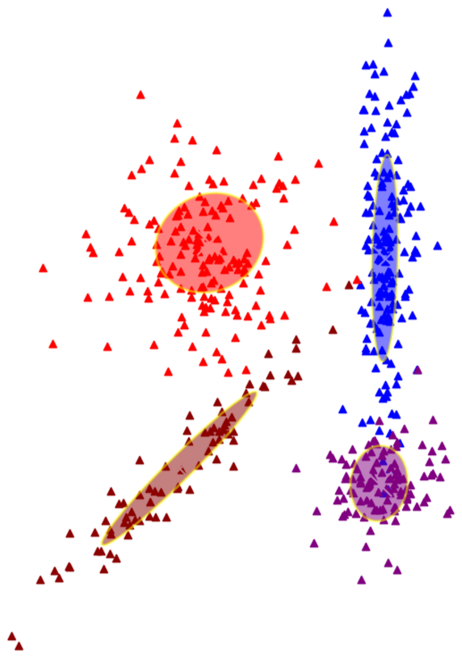
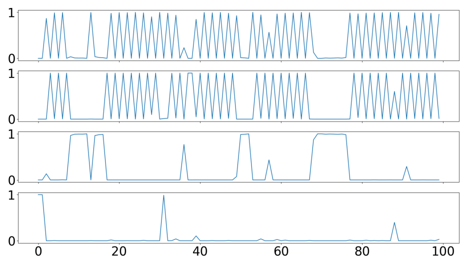


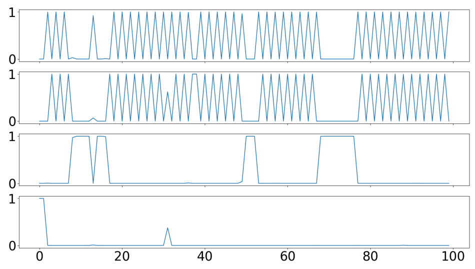
Figure 3: Representation of the most likely sequence of states along with the different learned parameters.

Q9. For each state plot the probability of being in that state as a function of time for the 100 first points of the test dataset.

Figure 4b shows for each state the probability of being in that state as a function of time for the 100 first points of the test dataset. Compared to Fig. 4a, one can see that the output is more smooth and most of the ambiguous points have disappeared.



(a) With initial parameters of Q2.



(b) After learning the parameters with EM

Figure 4: Representation of the most likely state as a function of time for the 100 first datapoints in the test file.

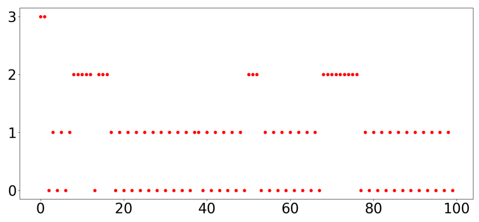
Q10. Make a plot representing the most likely state in $\{1, 2, 3, 4\}$ based on the marginal probability computed in the previous question as function of time for these 100 points.

Figure 5a shows the most likely state in $\{1, 2, 3, 4\}$ based on the marginal probability computed in the previous question.

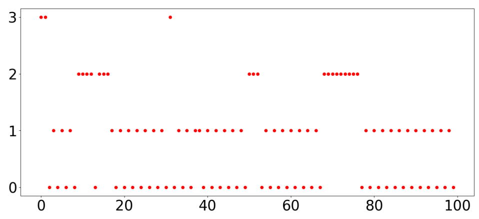
Q11. Run Viterbi on the test data. Compare the most likely sequence of states obtained for the 100 first data points with the sequence of states obtained in the previous question. Make a similar plot. Comment.

Figure 5b shows the most likely state in $\{1, 2, 3, 4\}$ based on the Viterbi decoding algorithm. One can see that for most of the points we predict the same state with the two algorithms. Nevertheless, in some cases, the sequence based on the marginal probabilities has some mistakes.

We can conclude that the method using marginal probabilities is a good proxy to have a quick and reliable answer but it can make mistakes. The Viterbi decoding can find the exact solution but needs more computational resources.



(a) Computed with the marginal probability



(b) Computed with Viterbi decoding

Figure 5: Representation of the most likely state as a function of time for the 100 first datapoints in the test file after learning the parameters with EM.

Q12. In this problem the number of states was known. How would you choose the number of states if you did not know it ?

In the case where the number of states is unknown and not obvious when looking at the data because of e.g. the dimensionality, two methods could be explored:

(i) cross-validation: the idea is to compute the log-likelihood over a dev set with models trained on the train set with different number of states. Once the best number of states according to the dev set has been found, one can finally evaluate on the test set.

(ii) Bayesian Information Criterion (BIC): the idea is to compute the BIC for a set of models and to choose the model with the lowest score. The BIC is defined as:

$$\text{BIC}(\mathcal{M}) = k \log n - 2 \log L$$

where k is the number of parameters, n is the number of datapoints and L is the maximum likelihood of the model $\mathcal{M}$. Hence, it penalizes the minimum negative log-likelihood by the number of free parameters.