Graphical models - HWK 3

Yifan WANG
Xiao CHU

Notation

- u_t: observation at time step t
- q_t: latent state at time step t
- $u_{t_1:t_2}$: sequence of observations $u_{t_1}...u_{t_2}$
- $q_{t_1:t_2}$: sequence of states $q_{t_1}...q_{t_2}$
- A: transition matrix, $A_{ij} = p(q_t = i|q_{t-1} = j)$
- π: marginal probability of initial latent node, $\pi_k = p(q_1 = k)$
- μ_k: mean of k-th Gaussian distribution in GMM
- Σ_k: covariance matrix of k-th Gaussian distribution in GMM
- T: sequence length
- K: number of different latent states

1 HMM-Implementation

1. Implement the recursions α and β to compute $p(q_t|u_1, ..., u_T)$ and $p(q_t, q_{t+1}|u_1, ..., u_T)$.

We can compute $p(q_t|u_1, ..., u_T)$ and $p(q_t, q_{t+1}|u_1, ..., u_T)$ with α and β obtained with forward and backward algorithm.

$$p(q_t|u_{1:T}) = \frac{\alpha(q_t)\beta(q_t)}{\sum_{q=1}^{K} \alpha(q)\beta(q)}$$

$$p(q_t, q_{t+1}|u_{1:T}) = \frac{\alpha(q_t)\beta(q_{t+1})p(q_{t+1}|q_t)p(u_{t+1}|q_{t+1})}{\sum_{q=1}^{K} \alpha(q)\beta(q)}$$

where in our setting, $p(u_{t+1}|q_{t+1}) = \mathcal{N}(u_{t+1}; \mu_{q_{t+1}}, \Sigma_{q_{t+1}})$
Please refer to MVA_DM3_WANG Yifan_CHU Xiao.ipython to see the implementation.

2. Finally, represent $p(q_t|u_1, ..., u_T)$ for each of the 4 states as a function of time for the 100 first data points in the file (EMGaussienne.test).

$p(q_t|u_1, ..., u_T)$ for each of the 4 states are shown in Figure 1.

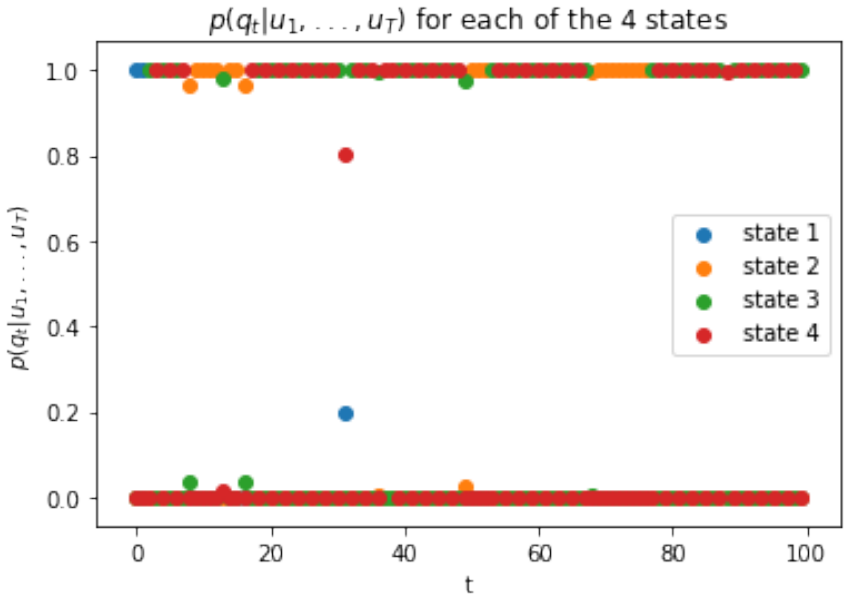


Figure 1: $p(q_t|u_1, ..., u_T)$ for 4 states

3. Derive the estimation equations of the EM algorithm

The lower bound of the complete log-likelihood that we want to maximize is

$$\begin{aligned} l(\theta, u_{1:T}) =& E_{p(q_{1:T}|u_{1:T})}[log(p(q_1)) + \sum_{t=1}^{T} log(p(q_t|u_t)) + \sum_{t=1}^{T-1} log(p(q_{t+1}|q_t))] \\ =& \sum_{k=1}^{K} p(q_1 = k) log(\pi_k) + \sum_{t=1}^{T} \sum_{k=1}^{K} p(q_t = k) log(\mathcal{N}(u_t; \mu_k, \Sigma_k)) + \\ & \sum_{t=1}^{T-1} \sum_{j=1}^{K} \sum_{k=1}^{K} p(q_t = k, q_{t+1} = j) log(A_{jk}) \\ =& \sum_{k=1}^{K} \gamma(q_{1k}) log(\pi_k) + \sum_{t=1}^{T} \sum_{k=1}^{K} \gamma(q_{tk}) log(\mathcal{N}(u_t; \mu_k, \Sigma_k)) + \sum_{t=1}^{T-1} \sum_{j=1}^{K} \sum_{k=1}^{K} \xi(q_{tk}, q_{t+1,j}) log(A_{jk}) \end{aligned}$$

where $\gamma(q_{tk}) = p(q_1 = k)$, $\xi(q_{tk}, q_{t+1,j}) = p(q_t = k, q_{t+1} = j)$ and $\mathcal{N}(u_t; \mu_k, \Sigma_k)$ is the probability density of u_t for distribution k. Besides, we have $\sum_{k=1}^{K} \pi_k = 1$ and $\forall i = 1...K, \sum_{k=1}^{K} A_{ki} = 1$.

We write its Lagrangian

$$L(\theta, \alpha, \beta) = l(\theta, u_{1:T}) + \alpha(\sum_{k=1}^{K} \pi_k - 1) + \sum_{i=1}^{K} \beta_i (\sum_{k=1}^{K} {}_{ki} - 1)$$

By setting its partial derivatives to 0, we have the maximum likelihood estimators.

$$\pi_k = \frac{\gamma(q_{1k})}{\sum_{j=1}^{K} \gamma(q_{1j})}$$

$$A_{jk} = \frac{\sum_{t=1}^{T-1} \xi(q_{tk}, q_{t+1,j})}{\sum_{i=1}^{K} \sum_{t=1}^{T-1} \xi(q_{tk}, q_{t+1,i})}$$

$$\mu_k = \frac{\sum_{t=1}^{T} \gamma(q_{tk}) u_t}{\sum_{t=1}^{T} \gamma(q_{tk})}$$

$$\Sigma_k = \frac{\sum_{t=1}^{T} \gamma(q_{tk})(u_t - \mu_k)(u_t - \mu_k)^T}{\sum_{t=1}^{T} \gamma(q_{tk})}$$

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

Please refer to MVA_DM3_WANG Yifan_CHU Xiao.ipython to see the implementation.

5. Plot the log-likelihood on the train and test data. Comment.

The log-likelihoods are shown in Figure 2. We notice that the log-likelihood is higher on train data than on test data, and they both increase with similar speed.

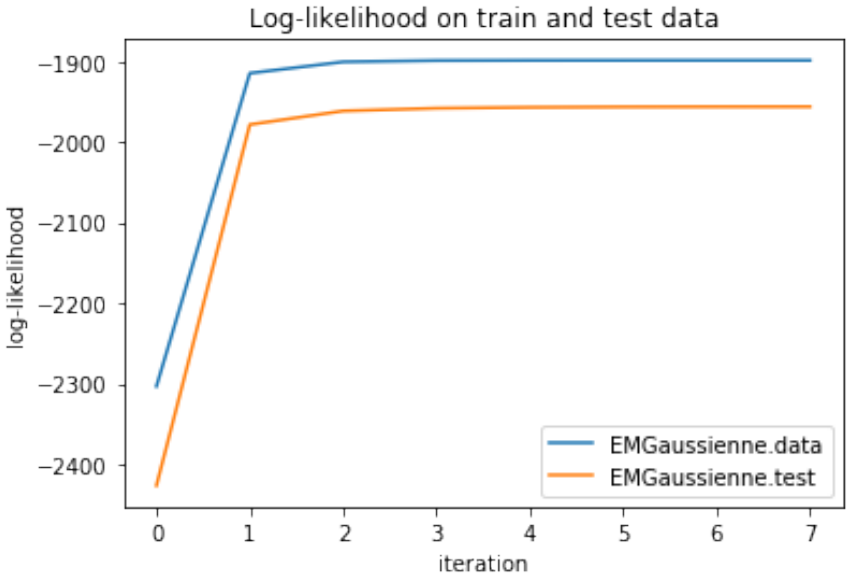


Figure 2: Log-likelihood on the train and test data

6. 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. Compare these log-likelihoods as well with the log-likelihoods obtained for the dierent models in the previous homework.

The log-likelihoods are shown on Table 1. HMM achieves a higher likelihood on both dataset, and for both model, the likelihood is higher on train data. However, it doesn't really make sense to compare these values, since we have assume different distributions with two models, and we can have a higher log-likelihood by making the model more complex, for example, by increasing the number of clusters in GMM.

	Train data	Test data
GMM	-2373.0	-2453.4
HMM	-1898.9	-1956.3

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

7. Provide a description and pseudo-code for the Viterbi decoding algorithm (aka MAP inference algorithm or max-product algorithm) that estimates the most likely sequence of states

The most-likely path is obtained by:

$$q^*_{1:T} = argmax_{q_{1:T}} p(q_{1:T}|u_{1:T}) = argmax_{q_{1:T}} p(u_{1:T}|q_{1:T})p(q_{1:T})$$

As we know that:

$$max_{q_{1:T}} p(u_{1:T}|q_{1:T})p(q_{1:T}) = max_{q_T} p(y_T|q_T) max_{q_{N-1}} p(q_N|q_{N-1}) ... max_{q_2} p(q_3|q_2)p(u_2|q_2) max_{q_1} p(q_2|q_1)p(u_1|q_1)p(q_1)$$

Thus we can obtain the most-likely path by getting the q_1, q_2,...,q_T which can maximize this equation.

The pseudo-code is shown in the Table 2.

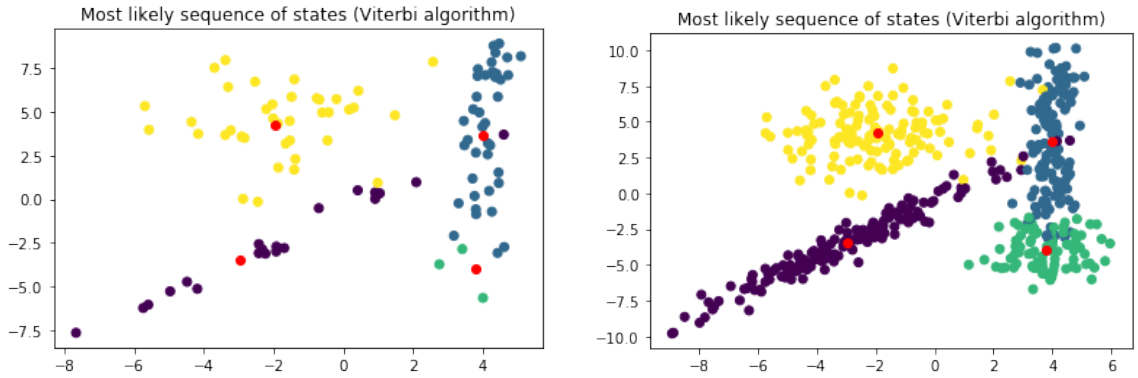
Here, we suppose that S=$\{s_1, s_2, ..., s_K\}$ the state space, O=$\{o_1, o_2, ..., o_N\}$ the observation space, U =$(u_1, ..., u_T)$ a sequence of observations, A the transition matrix such that A_{ij} stores the transition probability of transiting from state s_i to state s_j, π the an array of initial probabilities such that π_i stores the probability that $q_1 = s_i$.

Viterbi Algorithm
Initialisation
log_probs = zeros(K, T) # Initialize path costs going into each state
prev_state = zeros(K, T) # Initialize arrays to store best paths
Viterbi_path = zeros(T, 1); # Initialiaze array to store the most likely path of Viterbi algorithm
log_probs [:,1] = log(pi) # Begin with p(q_1)
Iterate over all observations, updating costs and highest-probability states
for t = 2,...,T
Iterate over all states, find the best previous state
for i = 1,...,K
costs = log_probs (:,t-1) + logA(i,:)^T + $logp(u_{t-1}|q)$;
Choose the state with the maximum value and then update log_probs and prev_state
[maxValue, maxIndex] = max(costs)
log_probs(i,t) = maxValue
prev_state(i,t) = maxIndex
Find the state with highest log-probability at the end
finalvalue, finalstate= max(log_probs(:,T)+logp($u_T|q$));
Backtracking to find the most likely path
Viterbi_path(T) = finalstate
for t = T-1, T-2, ..., 1
Viterbi_path(t) = prev_state(Viterbi_path(t+1), t+1);
Output: Viterbi_path

Table 2: The Viterbi decoding algorithm

8. Implement Viterbi decoding. For the set of parameters learned with the EM algorithm, compute the most likely sequence of states with the Viterbi algorithm

We compute the most likely sequence of states with Viterbi algorithm and represent the data in 2D with the cluster centers (4 red points) and with markers of different colors for the datapoints belonging to different classes, as shown in the Figure 3.



(a) First 100 points (b) All points

Figure 3: Most likely sequence of states obtained by Viterbi algorithm

9. For the datapoints in the test file "EMGaussienne.test", compute the marginal probability $p(qt|u_1, ..., u_T)$ for each point to be in state 1, 2, 3, 4 for the parameters learned on the training set.

We compute the marginal probability $p(q_t|u_{1:T})$ (smoothing distribution) for each point and the probability of being in that state as a function of time for the 100 first points in shown in the Figure 4.

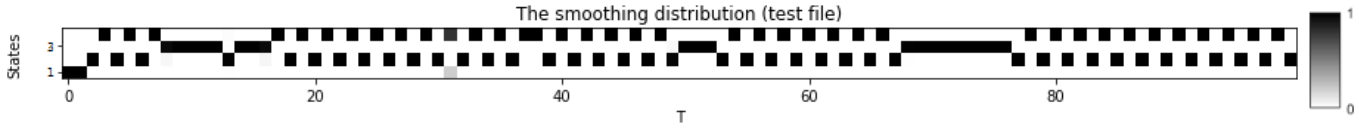


Figure 4: Probability of being in each state for the 100 first points

The darker the color is, the higher the probability is. From this Figure, we can observe that the marginal probability $p(q_t|u_{1:T})$ (smoothing distribution) is almost 1 for one state and is almost 0 for the other states for all points.

10. For each of these same 100 points, compute their most likely state according to the marginal probability computed in the previous question.

We compute their most likely state according to the smoothing marginal probability as we chose the states with the highest probability. The most likely state in 1, 2, 3, 4 as function of time for these 100 points is shown in the Figure 5.

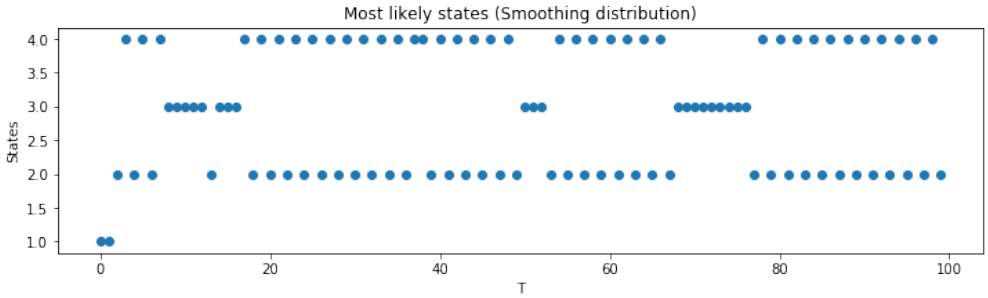


Figure 5: Most likely state according to smoothing distribution

11. 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.

We apply the Viterbi decoding on the test data and the most likely states obtained by Viterbi algorithm are shown in the Figure 6.

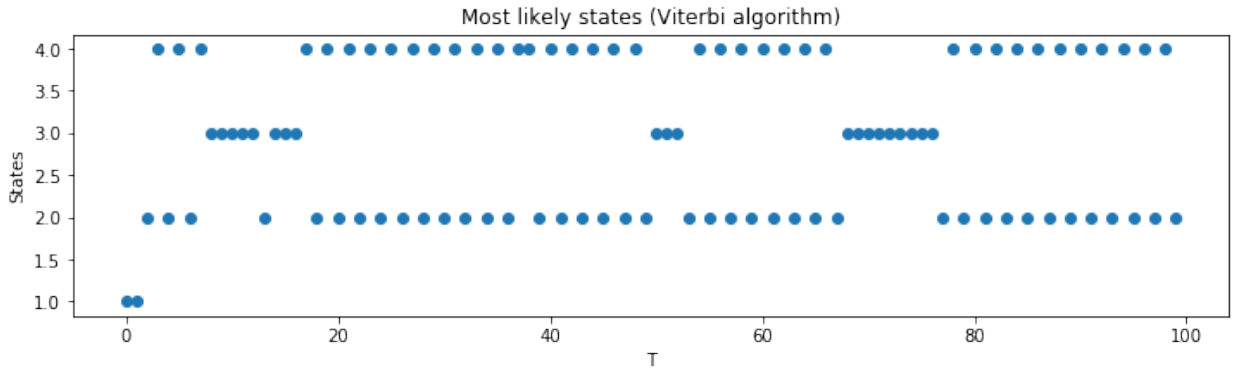


Figure 6: Most likely state according to Viterbi algorithm

Compared with the most likely states according to smoothing distribution obtained in the previous question, we can find that the most likely states obtained by the Viterbi algorithm is exactly the same.

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

We can split the train data into two parts, and use one part as validation data. We train the model on the new train data with different values of K (number of states), and choose the one that has best performance on validation data. It is also possible to use cross-validation to choose this hyper-parameter.

As for the performance metric, as mentioned, log-likelihood might not be a good metric since it will leads to a very complex model. If we're given some specific metrics in some situation, we can just use these metrics to choose K. Otherwise, we can try criteria like Bayesian Information Criteria (BIC) which take into account both the likelihood and the model complexity.