Hidden semi-Markov Models (HSMM)

Comparisons with GMM and HMM, and applications to audio processing.

Research Project Report – Probabilistic Graphical Models course

Lilian Besson[0]
Department of Mathematics

Valentin Brunck
Department of Physics

École Normale Supérieure de Cachan (France)
{lbesson,vbrunck}@ens-cachan.fr

Abstract

In this small project report, we study a generalization of Hidden Markov model (HMM) models, called Hidden semi-Markov Models (HSMM). Taking the point of view of Probabilistic Graphical Models, we begin by simplifying the general graph in order to obtain a simpler, tractable factorization. Then, as for HMM, we propose a simple and efficient forward-backward α-β algorithm for filtering, and a less simple E-M algorithm to infer the parameters of our probabilistic model. We then compare HSMM with Gaussians Mixture Models (GMM) and HMM. Finally, we conclude with an overview of the possible applications of the HSMM, and give a partial bibliography in the appendix, along with proofs of complexity for our two algorithms.

Project Advisor: Guillaume Obozinski (ENPC)
Course: *"Probabilistic Graphical Models"*, by F. Bach, S. Lacoste-Julien, G. Obozinski
Master program: Mathématiques, Vision, Apprentissage (MVA) Master 2 at ENS de Cachan.

---

[0] If needed, see on-line at `http://lbo.k.vu/pgm2016` for an e-version of this report, as well as additional resources (poster, code, figures, complete bibliography etc), open-sourced under the MIT License.

1 Presentation and motivation

HMM are a family of *directed probabilistic graphical models* that allows to account for *temporal structure* in the data. They have been thoroughly studied from the last 20 years, and successfully applied to many problems, from gesture recognition to medical diagnosis to phoneme inference ([Bis06]).

One consequence of HMM is that they can only have *geometrically distributed durations*:

$$\mathbb{P}(d|q=j) = (1-A_{j,j})A_{j,j}^{d} \text{ in state } q=j.$$

But this can be inappropriate for some applications, such as music, rainfalls, component's life, etc. This can be a severe limitation because the probability of a state change should depends on the time spent in the current state: some application require a distribution which is not memory-less.

In this project, we study **Hidden Semi-Markovian Models** (HSMM), an extension of the HMM model, introduced in the 60s by Baum and Petrie [BP66]. We follow its presentation in 2002 by [Mur02] and look at applications in audio processing ([CC14, BBC15, NCC+15]).

In a nutshell, HSMM are HMM where each hidden state q_i $(i = 1..N)$ can emit a *sequence* of observations $u_{t:t+d_i}$ ([Mur02]), whereas hidden states in a regular HMM only emit a single observation. H**S**MM also models the time spent on a hidden state, ie. the sequence duration d_i $(i = 1..N)$. Many possible prior belief can be used for the durations of these observations ([BBC15]), leading to various performances when HSMM are applied to a specific problem. A practical application is music sheet matching and alignment, where the sound recorded is following a known pattern but the duration of each state is unknown (or close to a prior given by the music sheet tempo).

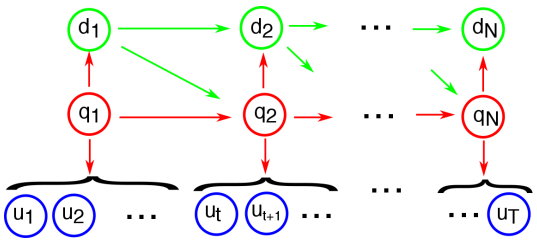
---

2 The HSMM model: equations and illustrations

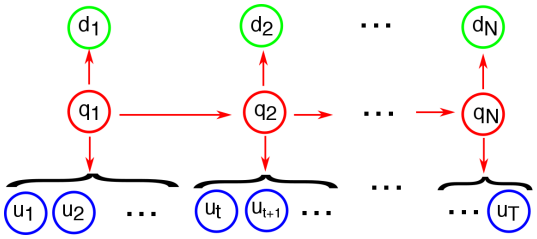
In this section, we start by presenting this family of Probabilistic Graphical Models (**PGM**), with two illustrations, and then derive an equation of its factorization and of the transition and emission matrices. We highlight which hypotheses are relaxed from the HMM models, and their consequences.

2.1 A PGM for HSMM

Let start by showing two illustrations of the HSMM model drawn as a PGM. The first model on the left is not simplified, as proposed in [Mur02, Gué03]. Fig. 1a represents HSMM as a graphical model, by introducing a new random variable d. Fig. 1b is a simplified version, where a transition to the current state is independent to the duration of the previous state and the duration is only conditioned on the current state.



(a) A graphical model for HSMM.



(b) Simplified HSMM

Figure 1: N sequences of observations of a HSMM (hidden state: $q_n \in \{1..K\}$, observed variables: u_t, duration of nth sequence: d_n), for a total length $T = d_1 + \cdots + d_n + \ldots d_N$).

The probabilistic influences showed in this illustration are detailed below in equations.

2.2 Hypotheses and equations

Basically, HSMM are **HMM without the Markov property**. But it is crucial to specify which of the *two* HMM Markov properties we remove. We conserve[1] the hypothesis that states transition $(q_n = i \to q_{n+1} = j)$ are Markovian (with a transition matrix $A_{i,j}$), but relax the hypothesis that each hidden state emits only one observation. The sequence duration is assumed to be independent to the previous state.

We consider N hidden states, written q_n $(n = 1..N)$, in $\{1..K\}$, and each one can emit either:

- For HMM: $q_n \longrightarrow u_n$: only one observation is generated by each hidden state q_n,
- But for HSMM: $q_n \longrightarrow u_{t_n:t_n+d_n} = (u_s)_{t_n \leq s \leq t_n+d_n}$: a *sequence* of observations is now generated by each state q_n, of duration[2] $1 \leq d_n \leq d_{\max}^{(j)}$ if $q_n = j$, with $d_{\max}^{(j)} \leq D_{\max}$ (this is explained with more details later).

Therefore, the semi-Markov model is represented by this factorization, where the random variables for the

[1] Hence the "semi-Markov" part in the HSMM name: we relax only one of the *two* Markov hypotheses.

[2] $d_n > 0$ to not consider useless empty sequences.

durations (d_i) are made implicit at first:

$$\mathbb{P}\Big((q_n, (u_{t_n:t_n+d_n})_n), n = 1..N\Big) = \mathbb{P}(q_1)\mathbb{P}((u_{1:d_1})_1|q_1) \cdots \prod_{n=2..N} \Big(\mathbb{P}(q_n|q_{n-1})\mathbb{P}((u_{t_n:t_n+d_n})_n|q_n)\Big)$$

As for a HMM, this factorization shows that we first need a prior on the hidden states distribution ($\pi(k) = \mathbb{P}(q_1 = k)$), and then we have two types of probabilistic influences:

$$\begin{cases} \mathbb{P}(q_n|q_{n-1}) & : \text{Markovian transition on the states,} \\ \mathbb{P}((u_{1:d_n})_n|q_n) & : \text{Emission of a sequence, for each state.} \end{cases}$$

Below is listed the major hypotheses of the HSMM:

- The transition between hidden states are (still) Markovian and homogeneous. A transition to the current state j is independent to the duration of the previous state j and the duration is only conditioned on the current state, so that we have:

$$A_{i,j} = \mathbb{P}(i \rightarrowtail j) = \mathbb{P}(j|i) = \mathbb{P}(q_{n+1} = j|q_n = i)\ \forall n = 1..N$$

- The observation sequence durations: $D_{j,d} = \mathbb{P}(d|j)$. For HMM, $D_{j,d}$ was implicitly following a geometric law, of parameter $p = A_{j,j}$, ie. $D_{j,d} = (1 - A_{j,j})A_{j,j}^d$. For a HSMM, D_j can now follow any discrete distribution (on $\mathbb{N}$, e.g. geometric, Poisson, truncated geometric etc).

- We only consider Gaussian emission probabilities for the observations u_t [Yu10], parametrized by one mean and covariance (μ_k, Σ_k) for each cluster $k = 1..K$, exactly as for a HMM:

$$B_{t,j,d} = \mathbb{P}(u_{t-d+1:t}|j, d) = \prod_{t'=t-d+1}^{t} \mathcal{N}(y_{t'}|j, d)$$

$$= \prod_{t'=t-d+1}^{t} \mathcal{N}(y_{t'}|\mu_j, \Sigma_j, d) = \prod_{t'=t-d+1}^{t} \mathcal{N}(y_{t'}|\mu_j, \Sigma_j)$$

We can remove the marginal on d because we assume that the j-th Gaussian distribution does not depend on the duration d.

2.3 Quick discussion about the models' parameters and their typical values

Below is given a list of the important numerical parameters when trying to apply a HSMM. Just to make this clearer, for the datasets we used below, these parameters (of $T, N, D_{\max}$) have values typically of the order of:

- r is the dimension of the dataset, each observation u_t belongs to $\mathbb{R}^r$, and in our examples we stayed with $r = 2$. Note that r does not influence that much the complexity of our algorithms, so we could easily scale to bigger r (not high-dimensional like $r = 1000$ but easily with e.g. $r = 3$ or $r = 10$).

- On one hand, T is the size of the dataset, ie. the number of observations u_t we have in the text file. On the first experiment we tried it was $T = 500$, and then $T = 1000$ and $T = 2000$. In order for a HSMM-based sound processing algorithm to scale to real-world data it should be at most linear in T, as T is the basically the length of the song and will usually be quite large.

- On the other hand, N is the size of the hidden states sequence. Obviously, it is unknown when we only provide $(u_t)_{t=1..T}$. We have $1 \leq N \leq T$, and if we try to model sequences of typical length l, then $N \approx T/l$. HSMM will be useful if we do try to model sequences, ie. if $N < T$ (but even with $l = 2$ or 10 HSMM will perform well, see below for a few experiments). It is important to note that N **does not** appear in any algorithm complexity, as it is a hidden parameter (and a random one, depending on our prediction for $(\widehat{q}_n)_{n=1..N}$).

- We also have the parameter K, size of the (discrete and finite) state space. A hidden state q_t will only take his value $q_t = k \in \{1, \dots, K\}$, exactly like a cluster index. Note that this K appears in the complexity of both our algorithms presented below. One classical question is to also learn the number of clusters, but for music related application it might not be that important (e.g. for music sheet alignment, a cluster is a tone so we already know how many are they).

- $D_{\max}$ is the constant prior we impose on the maximum value the random variables d_n can take, ie. it is the longest possible sequence durations. This parameter is very important, and at first we thought that choosing a "brutal" $D_{\max}$ equals to the whole sample (e.g. $D_{\max} = T = 500$) would work, but it turns out the α-β algorithm's complexity (and worse, the EM algorithm's complexity) starts to be huge. EM's complexity is of the order of $KTD_{\max}^2$, so it was of the order of $4 \times 500^3 \simeq 5 \times 10^8$ loops. Note that this rough number is not the number of elementary operations, just the number of time we cycle the various loops! (so its execution will really take a while).
  Long story short, filtering and inference in HSMM become intractable if we try to model too long sequences ($D_{\max} \approx T$ and not $D_{\max} \ll T$, which will anyway have a very small probability for real-world time-dependent data), and that's quite coherent. E.g. for music, HSMM seems to not be appropriate if a 2-minute music track has only two or three notes each being played 40 seconds.

---

3 Filtering with an efficient α-β forward-backward algorithm

The filtering problem is to predict the most probable value for every hidden state, after having observed the *whole*[3] sequence of values u. We need the entire sequence because there is two steps of message passing (it is a MPA), a forward step (from the past u_1 to the last one u_T), and a backward step. The goal is therefore to give an iterative and exact[4] algorithm to compute:

$$\mathbb{P}(q_t = j | u_1, \dots, u_T).$$

Note that here, we introduce the notation q_t (and not q_n), because in practice we do not have access to n but only to t. In that notation, we consider one q_t emitting only one u_t rather than a sequence of observations.

3.1 Forward and backward recurrent equations

We use the notations from [Mur02]. We can define similar quantities as α and β in HMM in the following way:

$$\alpha_{t,j,d} = \mathbb{P}(q_{t-d+1:t} = j, u_1, \dots, u_T)$$

$$\beta_{t,j,d} = \mathbb{P}(u_{t+1,\dots,u_T} | q_{t-d+1:t})$$

The *forward* recursion[5] (α), and the *backward* recursion (β) are given by:

$$\alpha_{t,j} = \sum_{d=1}^{d_{\max}^{(j)}} \alpha_{t,j,d} = \sum_{d=1}^{d_{\max}^{(j)}} B_{t,j,d} D_{j,d} \left(\sum_{i=1}^{K} A_{i,j} \alpha_{t-d,i} \right) \quad (1)$$

$$\beta_{t,i} = \sum_{d=1}^{d_{\max}^{(j)}} \beta_{t,j,d} = \sum_{j=1}^{K} A_{i,j} \left(\sum_{d=1}^{d_{\max}^{(j)}} D_{j,d} B_{t+d,j,d'} \beta_{t+d,j} \right) \quad (2)$$

These two recursions are very similar to the ones for HMM, except for the additional term $D_{j,d}$ (highlighted in blue), and for the sum on d, from 1 to $d_{\max}^{(j)}$ (in green). Below we detail the complexity of the forward-backward algorithm.

This $d_{\max}^{(j)}$ means the longest duration to be considered for the hidden state j. As D is given as a distribution on the domain $\Omega = \{1, \dots, K\} \times \{1, \dots, D_{\max}\}$, for each j its support on d could be smaller than the whole $1, \dots, D_{\max}$. Obviously, as we will have to sum up to these $d_{\max}^{(j)}$, the smaller the better. Having smaller $d_{\max}^{(j)}$ either means a better prior knowledge on our data (e.g. for music, the hidden state $j = 1$ is a high pitch note, never played longer than $250ms$), or a restriction to try to reduce the computational time (e.g. forcing a truncated distribution).

For HSMM, the α-β forward-backward algorithm has a complexity[6] of $O(TD_{\max}K^2)$, with a rather small constant.

3.2 How to work only with logarithms to avoid underflow errors?

[Mur02] explains how to use in HSMM the "log trick"[7] to remove (or at least reduce) *underflow risks*. This is very close to what is done for HMM, we just have more variables and more loops. For the forward messages:

$$\begin{cases} \log \alpha_{t,j} = \text{logsumexp}_d(\log B_{t,j,d} + \log D_{j,d} + \log \alpha^\star_{t-d,j}) \\ \log \alpha^\star_{t',j} = \text{logsumexp}_i(\log A_{i,j} + \log \alpha_{t',i}) \end{cases}$$

And for the backward messages:

$$\begin{cases} \log \beta_{t,i} = \text{logsumexp}_j(\log A_{i,j} + \log \beta^\star_{t,j}) \\ \log \beta^\star_{t,j} = \text{logsumexp}_d(\log B_{t+d,j,d} + \log D_{j,d} + \log \beta_{t+d,j}) \end{cases}$$

The term highlighted in red ($\log D_{j,d}$) does not appear in [Mur02, p.10], but it is obviously a mistake (as it is present in $\beta_{t,i}$ it cannot simply disappear in $\log \beta_{t,i}$!).

3.3 Implementation and application of the α-β algorithm

We implemented the α-β directly using logarithms, and our code is as simple and as clear as possible (please have a look to HSMM.py). It was quite efficient for small values of $D_{\max}$ $(2, 5, 10)$, but at first we forgot to specify a maximum duration, and as explained above in section subsection 2.3, with $D = T$ and $T = 500$, it was running for at least an hour.

Below is plotted in Fig. 2 a filtering obtained for 500 $2D$ points, generated from 4 Gaussian clusters (same data as for HMK3). For this first experiment, we chose to emulate the HMM used in HMK3 with a Geometric-duration HSMM, and, as expected, we got very similar results.

3.4 Using α-β values for the filtering

How can we compute the filtering $\mathbb{P}(q_t = j | u_1, \dots, u_T)$ for HSMM ? Contrary to HMM, we need to compute it at the same time as doing the forward-backward recursions, as we need to sum $\alpha_{t,j,d}$ and $\alpha_{t,j,d}$, because the filtering is given by:

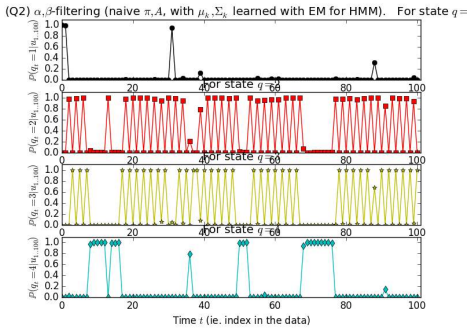
[3] An on-line version of this problem could also be studied, but we did not.

[4] Up to underflows or floating point errors.

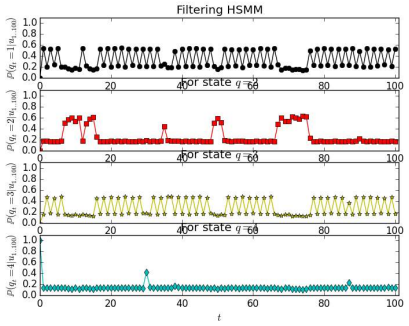
[5] For forward messages, we look back in the past (α_{t-d}), and for backward messages, we look "back to the future" (β_{t+d}).

[6] See section subsection A.1 in the appendix for a proof.

[7] Here we use the notation logsumexp_d for the function $E^D \to \mathbb{R}, (s_d)_{d=1..D} \mapsto \log(\sum_{d=1}^{D} \exp(s_d))$. It can be efficiently computed (on a computer) by first writing $s_0 = \max_d(s_d) = s_{d_0}$, and then $\log(\sum_{d=1}^{D} \exp(s_d)) = \log(\sum_{d=1}^{D} \exp(s_0) \exp(s_d - s_0)) = s_0 + \log(1 + \sum_{d \neq d_0} \exp(s_d - s_0))$ and using an efficient $x \mapsto \log(1+x)$ function (usually called `log1p`).



(a) α-β filtering by a HMM.



(b) α-β filtering by a Geometric-HSMM

Figure 2: Filtering on the first 100 points of HMK3 data, HMM and Geometric-HSMM.

$$\mathbb{P}(q_t = j|u_1, \dots, u_T) = \sum_{\tau \geqslant t} \sum_{d=\tau-t+1}^{d_{max}^{(j)}} \alpha_{\tau,j,d}\beta_{\tau,j,d}$$

4 Inference with an E-M algorithm

We now present an E-M algorithm we studied, in order to infer the hidden variables (expectation step) and to learn the parameters (maximization step).

4.1 Expectation (E step)

We can compute expectation formulas in a similar way as what is done in regular HMM (see [Yu10, Part 4.4.3] and [Gué03] for the details):

$$\eta_{t,j,d} = \alpha_{t,jd}\beta_{t,j,d}$$

$$\xi_t(i, d'; j, d) = \alpha_{t,i,d'} A_{i,j} D_{j,d} B_{t,j,d} \beta_{t+d,j,d}$$

$$\xi_t(i,j) = \mathbb{P}(q_t = i, q_{t+1} = j|u_1..u_T) = \sum_{d'} \sum_{d} \xi_t(i, d'; j, d)$$

4.2 Maximization (M step)

For the M step, we first use the means and covariances of the Gaussian clusters, from the observations $(u_t)_{t=1..T}$, weighted by $\mathbb{P}(q_t = i)$, to update μ_i, Σ_i with this scheme:

$$\widehat{\mu}_i^{\text{new}} \leftarrow \frac{1}{f} \sum_{t=1..T} \mathbb{P}(q_t = i|u_1..u_T) u_t$$

$$\widehat{\Sigma}_i^{\text{new}} \leftarrow \frac{1}{f} \sum_{t=1..T} \left(u_t - \widehat{\mu}_i^{\text{new}}\right)^T \mathbb{P}(q_t = i|u_1..u_T) \left(u_t - \widehat{\mu}_i^{\text{new}}\right)$$

With normalization factor $f = \sum_{t=1..T} \sum_{i=1..K} \mathbb{P}(q_t = i)$.

And then for the HMM or HSMM parameters π, A:

$$\widehat{\pi}_i^{\text{new}} \leftarrow \mathbb{P}(q_0 = i|u_1..u_T) \quad \text{(From } \alpha\text{-}\beta \text{ filtering)}$$

$$\widehat{A}_{i,j}^{\text{new}} \leftarrow \sum_{t=1..T} \xi_t(i,j) / \Big(\sum_{t=1..T, k=1..K} \xi_t(i,k) \Big)$$

As for HMM [Mur02], we used these variables ξ_t to simplify the update formula for $A_{i,j}$:

$$\xi_t(i,j) = \mathbb{P}(q_t = i, q_{t+1} = j|u_1..u_T).$$

Finally, for the HSMM-specific parameter D (duration distribution):

$$\widehat{D}_{j,d}^{\text{new}} \leftarrow \eta(j,d) / \Big(\sum_{d=1..d_{\max}^{(j)}} \eta(j,d) \Big)$$

As expected, the last term $\widehat{D}_{j,d}$ is the most complicated. It uses an additional variable $\eta(j,d)$ [Yu10, Part 2.3.1], which has a very intuitive explanation: $\eta(j,d)$ is the probability of having a sequence of length d generated by the hidden state $q = j$. To compute this, we split the reasoning in two parts:

- First, we need to have been in $q = j$ **at least** d times ($q_{d-v} = j$ for $0 < v < d$) and being in another state after ($q_d \neq j$). This first term is quite simple and easy to compute.
- However we also need to consider the whole history of observations (for $t = 1..T-d-1$), to be able to say that we do not look to sequences longer than d, so we look for patters of the form $q_t \neq j$ first (not in j), then $q_{d-v} = j$ again for $0 < v < d$ (in j for exactly d times), and $q_{t+d+1} \neq j$ at the end (exited from j). Unfortunately, this second term is more complicated to compute, and more time consuming (cf. appendix subsection A.1).

If we combine these two terms, $\eta(j,d)$ can be explicitly written [Yu10, Part 4.4.1]:

$$\eta(j,d) = \mathbb{P}(q_{d-v} = j, v = 1..d, q_d \neq j|u_1..u_T) + \sum_{t=1..T-d-1} \mathbb{P}(q_t \neq j, q_{t+d-v} = j, v = 1..d, q_{t+d+1} \neq j|u_1..u_T)$$

The maximization step has a complexity[8] of $O(TK^2D_{\max}^2)$.

[8] See section subsection A.1 in the appendix for a proof.

5 Implementation and numerical experiments

This section first explains what we implemented ourselves and which Python packages was used for the experiments, and then details a few numerical experiments we did. We only worked with artificial $2D$ data, generated from a GMM, then from a HMM and then from a Poisson-HSMM.

5.1 A quick overview of our implementation

We implemented[9] filtering and inference for HMM, and the α-β recursion and filtering for HSMM with generic distribution ourself, and we also used the open-source `pyhsmm` package [Jc15] for the experiments (for R, `mhsmm` [OH+11] is a good alternative, and for Octave/MATLAB [Yu10] implement a HSMM toolbox). We used HMK 3 data, but also artificial $2D$ data drawn from a toy HMM (and then HSMM): random transitions A, priors π and means μ_i, with $K = 4$ Gaussian, and geometric durations only. In some experiments, $K = 4$ is automatically chosen, based on train/test log-likelihood comparison for HMM for $K = 1..20$ (*cf.* HMK 3, Q.12). For all the experiments, the HMM and HSMM models are fed with the $2D$ data, and then inferred with the EM algorithm (with $m = 200$ steps[10]).

5.2 A first experiment, continuing HMK 2 and 3

A first experiment was to be able to emulate a HMM with a HSMM with a geometric duration distribution, $D_j \sim \mathcal{G}(A_{j,j})$ given by diagonal of the transition matrix A, see [Mur02]. We will apply this "toy" HSMM to artificial $2D$ data, generated from 4 Gaussian cluster (ie. from a GMM, as in HMK3). Below in Fig. 3 is showed two clusterings, predicted by a Geometric-HSMM and a Poisson-HSMM. A Geometric-HSMM gives the same result as a HMM, while a Poisson-HSMM (with big duration prior) fails to separate the north-west nw and north-east ne clusters, as in the HMK3 datafile the points were constantly switching between the two clusters (ie. both clusters almost always have a duration of 1: $D_{nw} = D_{ne} \simeq \text{Dirac}(1)$). This result is quite logical: the Poisson-HSMM will merge the two clusters to try to find long sequences.

[9] Among other things, for more details see HMM.py on-line or in the zip folder.

[10] *Animated* plots of each of the clustering displayed below (obtained at each EM step) are all available on-line as `gif` files, see this one for example.

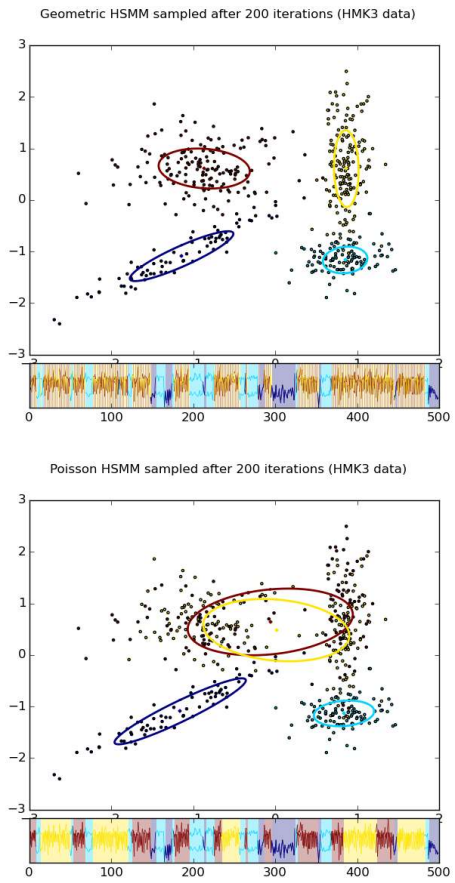


Figure 3: Comparing two HSMM on $2D$ data drawn from a 4-state GMM: Geometric-HSMM succeeds but Poisson-HSMM fails. The small line plot below shows the color of the predicted cluster as a function of time.

5.3 Sampling data from a HMM

Then we wanted to test our HMM and HSMM model on data presenting a true time dependency. We experimented on a few manually chosen HMM, and an interesting one is the following. Still in $2D$, with 4 clusters, we chose a transition matrix where there is **no** transition between state/cluster 1 and 3 (ie. $A_{1,3} = A_{3,1} = 0$), but they have almost the same Gaussian parameters; and no transition neither between 2 and 4, but they are far away from each other. We chose a prior on durations with means about 20. Below in Fig. 4, is displayed the data, with blue lines showing the transition between points, and 4 different clusters. As wanted, cluster 1 and 3 are very close, they are overlapping. Any model which does not take into account the order of appearance (like a GMM) will fail to separate the two overlapping clusters, and we observe this in Fig. 5, where a Geometric-HSMM fails to different the two clusters and predict $K = 3$ as the most probable number of cluster (it merges the overlapping clusters 1 and 3).

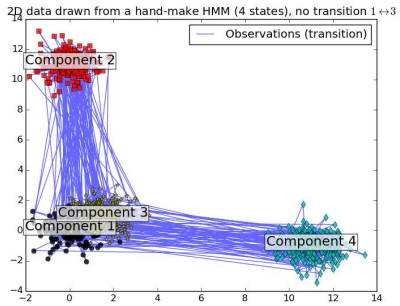


Figure 4: Data drawn from this toy 4-state HMM.

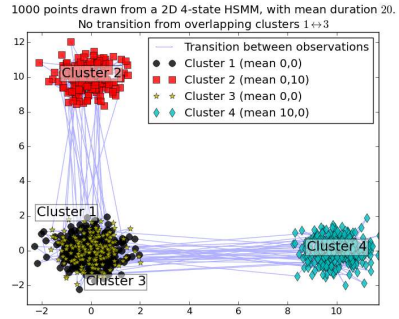


Figure 6: Data drawn from this 4-state Poisson-HSMM.

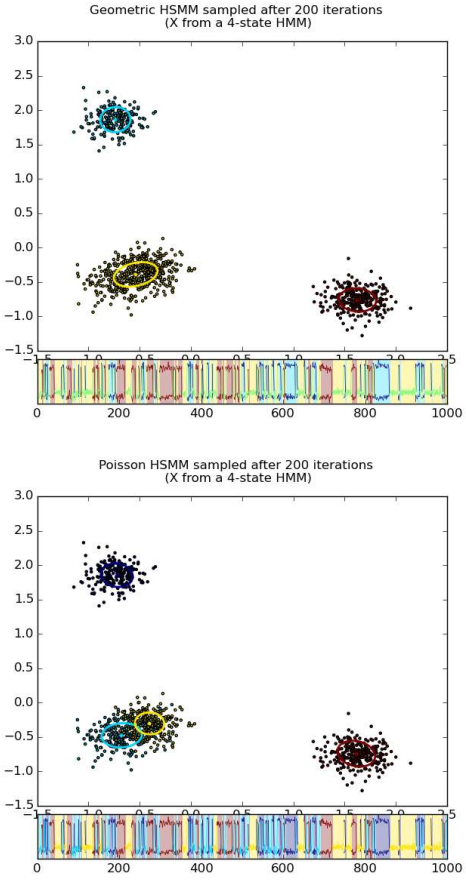
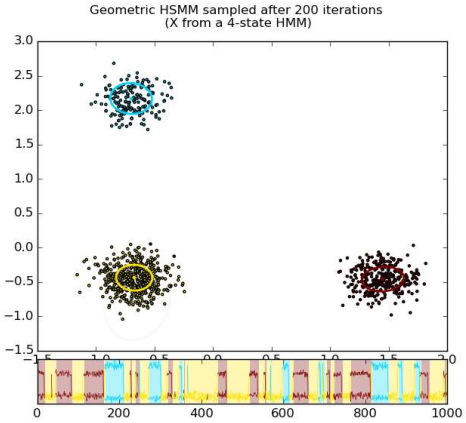
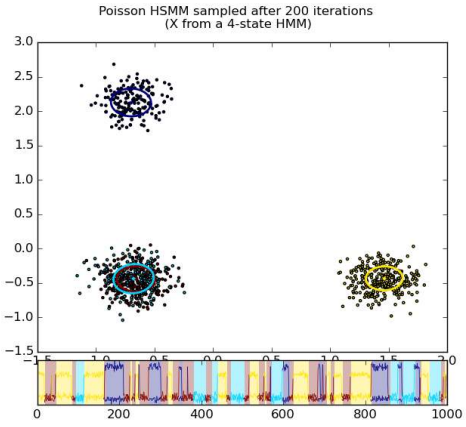


Figure 5: Comparing two HSMM on on Fig. 4 data.



(a) Small durations Geometric-HMM, merges clusters 1 and 3.



(b) Medium durations Poisson-HMM, succeeds to find 4 clusters.

Figure 7: Comparing two HSMM on Fig. 6 data.

5.4 Sampling from a HSMM

The next experiment is very similar, except we generated $T = 1000$ points from a manually chosen $2D$ Poisson-HSMM (with mean duration about 20), with the same kind of transition matrix and clusters (see Fig. 6). We again have overlapping clusters 1 and 3, this time with exactly the same Gaussian emission parameters. We obtained quite satisfactory results (see Fig. 7): a small-duration Geometric-HSMM fails to separate the two overlapping clusters, while a medium-duration Poisson-HSMM succeeds perfectly.

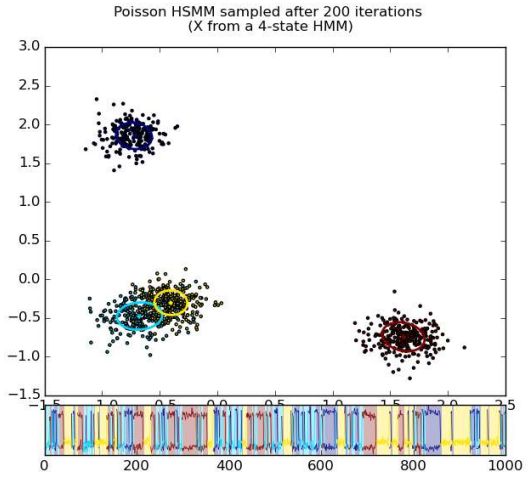
5.5 What's happening if we have no temporal dependency in the data

The last experiment we wanted to expose here is a very simple one. If HMM and HSMM models are great to model time dependency in the data, what happens in we shuffle the data? If we randomly change the order of

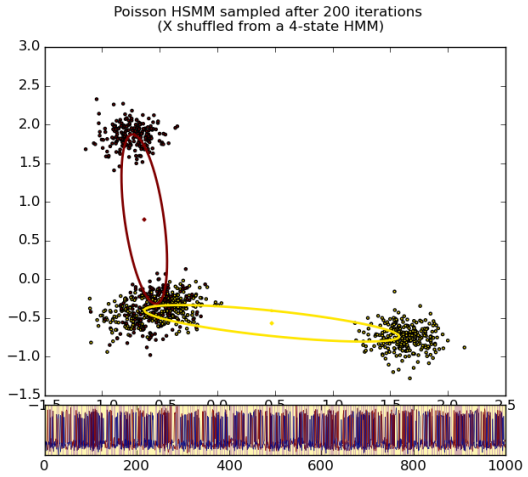
appearance of the observations u_t, then it is reasonable to say that no temporal dependency exists anymore.

The Gaussian Mixture Model (GMM) is of course invariant by shuffling. And both HMM and HSMM will be almost permutation invariant if they try to model sequences of lengths one or two (or other very small values) with very high probability (i.e. if A is almost diagonal for a HMM, and if the distribution $D_{j,d}$ is almost concentrated on $\{1, 2\}$).

For instance, below is shown the clustering predicted by a Poisson-HSMM (with a prior mean duration of 30), on the initial data (correctly time ordered, on the left), and on a randomly shuffled copy of the data. It is clear that trying to find sequence of duration $\simeq 30$ in the shuffled data will fail.



(a) $2D$ data drawn from a 4-state HSMM.



(b) Same data, but shuffled.

Figure 8: A Poisson-HSMM with long d succeeds on ordered data but fails on shuffled data, as expected.

And for the same data, we also tried a Geometric-HSMM with a very small duration mean (linked with how big $A_{j,j}$ is against $A_{j,i}$), and this one was robust against the permutation, because it is very similar to a GMM: trying to predict sequences of length almost all equal to 1 is like forgetting the time component! (These last plots are not included due to space constraints.)

---

6 Conclusion

In our projet report, we showed that:

- HMM are a special case of HSMM, and our HSMM implementation can emulate a HMM,
- For both HMM and HSMM, α-β is tractable, and efficient **for small truncated** $D_{\max}$,
- For Geometric durations, E-M for HSMM is *very* similar to E-M for HMM (cf. HMK3),
- But for other durations distribution, E-M is more complicated, but works in practice (cf. plots),
- The direct (exact) E step is time consuming for big dataset (very big T) or long total sequence length (big $D_{\max}$),
- And the general case of the M step is not more efficient.

We also illustrated the use of the α-β algorithm to predict a clustering from unlabeled but time-ordered data. The general HSMM, with parameters inferred from data with the E-M algorithm proved to be more general than HMM, and usually more efficient if the choice of the durations distribution is appropriate, and so HSMM also generalizes Gaussian Mixture Model. One interesting illustration (Fig. 7) is the ability to separate overlapping Gaussian clusters (same mean and covariance), by taking into account the order of appearance of the data in the HSMM (it is a sequential learning model).

6.1 Perspective and possible future directions of work

We could have explored into more details the Viterbi and E-M algorithm for HSMM. Another direction would be to try a more efficient algorithm for inference, like the variational E-M algorithm, or a more versatile one like on-line E-M (which has been successfully applied to real-world sound processing, see [BBC15]).

6.2 Real-world applications of HSMM, 3 examples

We studied three very recent papers [CC14, BBC15, NCC+15] applying with success HSMM to various audio processing tasks (real-time or not). Due to space constraints, we will not go into more details, please refer to these articles.

- [NCC+15] used a HSMM to match in real-time the music sheet for a piano player, and when applied to two-hands sheets played only with one hand, they succeeded to use this real-time alignement to generate the sound of the other hand. The system learns the duration spent on each note, and use the music-sheet as a prior on durations (using the initial tempo). Their video demonstration is quite cool: youtu.be/YRHgyl8IdNY.
- [CC14] applied HSMM to the classical sound processing task of source separation.
- [BBC15] proposed an on-line E-M algorithm for HSMM, and also present a survey of duration distributions that have been already used for HSMM in other papers.

A Proofs of complexity for α-β and E-M algorithms

In this appendix section we give detailed proofs of the complexity of the two main algorithms for HSMM.

A.1 Proof of α-β complexity

Lemma A.1 *For HSMM, the α-β forward-backward algorithm has a complexity of $O(TD_{\max}K^2)$, with a rather small constant.*

Proof A.2 *We have to compute the whole vectors $\alpha_{t,j}$ and $\beta_{t,i}$, for $t = 1..T$ and $i, j = 1..K$, so we already have a $O(TK)$ complexity.*

- *For each α message, 1 gives an external sum on $d = 1..d^{(j)}_{\max}$, of size bounded by $D_{\max}$, and an internal sum on $i = 1..K$. So for forward messages, we have $O(TD_{\max}K^2)$ as announced.*
- *For each β message, 2 gives symmetrically an external sum on $i = 1..K$, and an internal sum on $d = 1..d^{(j)}_{\max}$, of size bounded by $D_{\max}$, and an internal sum on $i = 1..K$. So for backward messages, we have $O(TD_{\max}K^2)$ as announced.*

As for the constant, the only operations are sums and products (indeed, in its first form it is a sum-product algorithm, SPA), or logsumexp *and sums (in its second form), and all this is quick and efficient.*

A.2 Proof of E-M complexity

Lemma A.3 *For HSMM, performing m steps[11] of the (iterative) E-M algorithm has a complexity of $m \times O(TD^2_{\max}K^2)$, with a* not-so-small *constant.*

Proof A.4 *Each step of the EM algorithm is one E step and one M step, and we do this $m \geq 1$ times:*

- ***E-step***

  We did not cover it in much details here, see [Yu10, Part 2.3.1]. Each expectation step is taking a time about $O(TD^2_{\max}K^2)$. It is basically using the α-β recursion, as explained above, but keeps and performs computations on all the non-simplified $\alpha_t(j, d)$ and $\beta_t(j, d)$ to evaluate the $\eta(j, d)$ and $\xi_t(i, j)$. The other computations will all be less costly.

- ***M-step***

  We have K means, covariances and states prior to compute, and for each $i \in \{1..K\}$:

  - *$\widehat{\mu}_i^{new}$ takes $O(Tr)$,*
  - *$\widehat{\Sigma}_i^{new}$ takes $O(Tr^2)$, in dimension r,*

[11] The number of E-M steps is typically linked with the aimed precision, but we do not discuss about these details here.

– $\widehat{\pi}_i^{new}$ *uses the filtering given by* α-β *but is in* $O(1)$ *after computing it (so for the entire complexity it yields* $O(TKD_{\max})$*),*

For the transition matrix, we have K^2 *terms to compute. Each* $\widehat{A}_{i,j}^{new}$ *takes* $O(T)$ *for the numerator, and* $O(TK)$ *for the denominator without any optimization. But when we computes these* $\xi_t(i,j)$*, by dynamic programming we can keep an additional vector (of size* $O(K)$*) for the normalization factor* $\sum_j \sum_t \xi_t(i,j)$*, and so only* $O(T)$*. Finally, updating the transitions takes* $O(TK^2)$ *(and does not depend on* $D_{\max}$*, this is logical because the E-step already computed the* $\xi_t(i,j)$*).*

And finally, for the durations, we have $D_{\max}K$ *terms to compute. Each* $\widehat{D}_{j,d}^{new}$ *takes a priori a time* $O(D_{\max})$ *after having computed the* $\eta(j,d)$*. Each* $\eta(j,d)$ *requires to compute a big sum on* $t = 1 \dots T-d-1$*, so about* $O((T-d)d)$ *(after having all these probabilities, thanks to the E-step). Hence, computing all the* $\eta(j,d)$ *takes a total time of* $\sum_{j=1..K} \sum_{d=1..D_{\max}} O((T-d)d) = O(KTD_{\max}^2)$ *(because* $(T-d)d \leq TD_{\max}$*).*
But we can again be clever, and by dynamic programming keep an additional vector (of size $O(D_{\max})$*) for the normalization factor* $\sum_d \eta(j,d)$ *while computing the* $\eta(j,d)$*; and so computing one* $\widehat{D}_{j,d}^{new}$ *after having all the* $\eta(j,d)$ *only takes a constant time. Finally updating the durations takes* $O(TKD_{\max}^2)$*.*

Finally, when we sum-up all these computations, we end up with $O(TK^2D_{\max}^2)$ *for each of the* m *steps.*

As for the constant in the O*, the operations we perform here are most costly than simple sums and products as it was for* α-β*. If our data are in* $\mathbb{R}^r$*, then we have an additional* r^2 *in this* O *(as for almost all data analysis algorithms). For typical music application, the dimension should not be seen as a parameter of the complexity, but* T, K *and* $D_{\max}$ *should.*

Acknowledgments

We would first like to thank Guillaume Obozinski our project advisor, as he replied quickly to our queries and provided useful direction of research before and during the poster presentation. Thanks also to both professors Simon Lacoste-Julien and Francis Bach for the Probabilistic Graphical Model course at the MVA Master program (Cachan, France) in Fall 2015.
Finally, thanks to our comrade Gabriel Huang (student at ECP) who also worked on the project on HSMM, but he preferred to focus on applications to music processing, and we had a few interesting discussions about HSMM.

Personal feelings about the project

We enjoyed working on a project which was close to the last homework, and we liked the different aspects we touched with this project: algorithms, complexity, implementation, simulation, illustration, probabilistic models, inference etc. The possible applications of HSMM, especially the one that learns to play the left-hand of a 2-hand piano track is very impressive (and might help one-handed music players!).

References

[BBC15] Alberto Bietti, Francis Bach, and Arshia Cont (2015). *An online EM algorithm in Hidden (Semi-)Markov Models for audio segmentation and clustering.* In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference*, pages 1881–1885.

[Bis06] Christopher M Bishop (2006). *Pattern Recognition And Machine Learning*. Springer.

[BP66] Leonard E Baum and Ted Petrie (1966). *Statistical inference for probabilistic functions of finite state Markov chains. The Annals of Mathematical Statistics*, pages 1554–1563.

[CC14] Philippe Cuvillier and Arshia Cont (2014). *Coherent time modeling of Semi-Markov Models with application to real-time audio-to-score alignment.* In *Machine Learning for Signal Processing (MLSP), 2014 IEEE International Workshop*, pages 1–6. IEEE.

[Gué03] Yann Guédon (2003). *Estimating Hidden Semi-markov chains from discrete sequences. Journal of Computational and Graphical Statistics*, 12(3):604–639.

[Jc15] Matthew J. Johnson and GitHub contributors (December 2015). *Bayesian inference in HSMMs and HMMs in Python [pyhsmm] (GitHub repository).* URL `https://github.com/mattjj/pyhsmm`, online, accessed 28.12.2015.

[Mur02] Kevin P. Murphy (2002). *Hidden Semi-Markov Models (HSMMs).* URL `http://www.cs.ubc.ca/~murphyk/mypapers.html`, Unpublished notes.

[NCC+15] Eita Nakamura, Philippe Cuvillier, Arshia Cont, Nobutaka Ono, and Shigeki Sagayama (2015). *Autoregressive Hidden Semi-markov Model of symbolic music performance for score following.* In *16th International Society*

for Music Information Retrieval Conference (ISMIR). Malaga, Spain.

[OH+11] Jared O'Connell, Søren Højsgaard, *et al.* (2011). *Hidden semi markov models for multiple observation sequences: The mhsmm package for R. Journal of Statistical Software*, 39(4):1–22. URL `https://cran.r-project.org/web/packages/mhsmm/`.

[Yu10] Shun-Zheng Yu (2010). *Hidden semi-Markov Models. Artificial Intelligence*, 174(2):215–243.

(*Note:* a more detailed bibliography is available on-line.)
