PAC-Learning of Markov Models with Hidden State

Ricard GAVALDA, Philipp W. KELLER, Joelle PINEAU, Doina PRECUP

McGill University, School of Computer Science, 3480 University St., Montreal, QC, H3A2A7

Abstract. The standard approach for learning Markov Models with Hidden State uses the Expectation-Maximization framework. While this approach had a significant impact on several practical applications (e.g. speech recognition, biological sequence alignment) it has two major limitations: it requires a known model topology, and learning is only locally optimal. We propose a new PAC framework for learning both the topology and the parameters in partially observable Markov models. Our algorithm learns a Probabilistic Deterministic Finite Automata (PDFA) which approximates a Hidden Markov Model (HMM) up to some desired degree of accuracy. We discuss theoretical conditions under which the algorithm produces an optimal solution (in the PAC-sense) and demonstrate promising performance on simple dynamical systems.

1 Introduction

Hidden Markov Models (HMMs) are widely used tools for prediction under uncertainty. Successful applications of these technologies include speech recognition (Rabiner, 1989) and DNA sequence alignment (Durbin et al, 1998). In this paper, we address the issue of learning such models from data.

The standard approach at the moment is to estimate model parameters directly from trajectories of observations (or action-observation pairs) using Expectation-Maximization (EM) (Rabiner, 1989). This approach has proved successful in many applications, but it also has some significant drawbacks. First, it assumes a known set of "real" hidden states S. In many domains, in particular in physical systems, there is a natural state representation. For example, in speech recognition, the set of phonemes is the standard choice of state representation, and in computational biology, the type of the subsequence (e.g., gene or promoter) is a natural choice. However, there are many domains where the choice of states is not at all obvious. For example in dialogue modelling, the state representation must somehow capture the user's communication goals. Similarly, in medical diagnostic and adaptive treatment design, the state must capture complex information about the patient, his/her disease and treatment history. In these and similar cases, the state is best represented by summary statistics over the set of past observations. Some recent research has focused on modeling just the observed data (Jaeger et al, 2006, Rosencrantz et al, 2004, Singh et al, 2003). In this case, knowing or defining hidden states ahead of time is not necessary. The algorithm we propose in this paper has a similar goal, although the methodology is different. We build a learning algorithm for probabilistic models which can simultaneously estimate a good state topology and set of parameters.

The second drawback of EM is that it converges to a locally optimal solution, and there are no guarantees on the quality of the final solution. In some domains, this is very problematic. The algorithm we propose has PAC-style guarantees on the model learned, using a polynomial amount of data.

We use Probabilisitc Deterministic Finite Automata (PDFA), a standard tool in computational learning theory, as the basic representation to be learned. We show how PDFAs can approximate HMMs. Our algorithm is based on a state-splitting and merging technique, and is designed in such a way as to be able to provide PAC guarantees. We illustrate the algorithm on some example problems, and show promising empirical results.

2 Background

We address the problem of learning the structure and parameterization of a dynamical system, directly from observational data generated by this system. The data typically consists of a set of trajectories, $D = \{d^1, d^2, ..., d^n\}$, each containing a finite sequence of observations: $d = \sigma_0\sigma_1...\sigma_k$. Different models have been used to capture this type of data; in this paper, we focus on Hidden Markov Models and Probabilistic Finite Automata.

A *probabilistic deterministic finite automaton (PDFA)* is a tuple $\langle S, \Sigma, T, O, s_0\rangle$, where S is a finite set of states, Σ is a finite set of observations, $T : S \times \Sigma \rightarrow S$ is the transition function $O : S \times \Sigma \rightarrow [0,1]$ defines the probability of emitting each observation from each state, $O(s,\sigma) = P(\sigma_t = \sigma | s_t = s)$, and $s_0 \in S$ is the initial state. Note that the transition to a new state is deterministic once an observation has been selected: $T(s,\sigma)$ gives the next state s'. A special symbol is reserved to mark the end of a string; alternatively, one can interpret this as a stop state with no outgoing edges. A probabilistic nondeterministic finite automaton (PNFA) is defined similarly except the transition function is stochastic: $T : S \times \Sigma \times S \rightarrow [0,1]$, and $T(s,\sigma,s') = P(s_{t+1} = s' | s_t = s, \sigma_t = \sigma)$.

Given an observation trajectory $d = \sigma_0\sigma_1, ..., \sigma_k$ emitted by a known PDFA, the state at each time step can be tracked by starting from the initial state s_0 and following the labelled transitions according to d. Also, the probability of generating a given trajectory $d = \sigma_0\sigma_1, ..., \sigma_k$ from a state s can be calculated recursively as follows: $O(s, \sigma_0\sigma_1...\sigma_k) = O(s,\sigma_0)O(T(s,\sigma_0), \sigma_1...\sigma_k)$.

A *Hidden Markov Model* is a tuple $\langle S, \Sigma, T, O, b_0\rangle$, where S is a finite set of states, Σ is a finite set of observations, $T(s,s') = P(s_{t+1} = s' | s_t = s)$ defines the probability of transitioning between states, $O(s,\sigma) = P(\sigma_t = \sigma | s_t = s)$ defines the emission probability of each observation on each state, and $b_0(s) = P(s_0 = s)$ is the initial state distribution. Given an observation trajectory d emitted by a known HMM, the probability distribution over states at any time b_{t+1}, can be estimated recursively by Bayesian updating:

$$b_{t+1}(s) \propto \Sigma_{s' \in S} b_t(s') O(s', \sigma_t) T(s', s) \tag{1}$$

Several links have been established between HMMs and probabilistic automata; a comprehensive review is in (Dupont et al., 2005). From the point of view of this paper, it is most important to note that an HMM can be transformed into an equivalent PNFA with the same number of states. A PNFA can also be transformed into an HMM, but not necessarily with the same number of states. Any PDFA $M = \langle S, \Sigma, T, O, s_0\rangle$ can be converted to an equivalent HMM $M' = \langle S', \Sigma, T', O', b_0\rangle$. The states in S' correspond to pairs of states in S among which a transition is possible: $S' = \{(s_1, s_2) \in S \times S | \exists \sigma \in \Sigma \text{ s.t. } T(s,\sigma) = s'\}$. The probability distributions of the HMM are then constructed as

follows:

$$b_0((s_0,s')) = 1/|S|$$
$$O'((s,s'),\sigma) = \frac{O(s,\sigma)}{\sum_{\sigma'=\Sigma} O(s,\sigma')}$$
$$T'((s,s'),(s',s'')) = \sum_{\sigma\in\Sigma} O(s',\sigma)\delta(T(s',\sigma),s'')$$

where δ is an indicator function. All other parameters are 0. It is easy to show that M' defines a proper HMM, and that M and M' will generate the same probability distribution over observation trajectories. Unfortunately, the reverse is not true: there are finite HMMs that can only be converted into PDFAs of infinite size. However, we will now show that any HMM can be *approximated* with a finite PDFA up to any desired degree of precision.

3 Approximating HMMS with PDFAs

Recalling that every HMM is equivalent to a PNFA (Dupont et al, 2005), we show that every finite-size PNFA can be approximated by a finite-size PDFA.

Theorem 1. *Let N be a PNFA and L be the expected length of strings generated by N. Then there exists a PDFA of size at most L/ε^2 that generates a distribution over trajectories that is ε-close to the distribution generated by N, under the L_∞ distance.*

Proof. Let S be the set of strings having probability at least ε in N. Note that there are at most $1/\varepsilon$ such strings, i.e., finitely many. It is easy to build a finite, tree-like PDFA M with $|S|$ leaves that generates exactly the strings in S, each with the same probability as N, and no other string. Hence, the distributions of M and N are ε-close.

To explicitly bound the size of the tree, we observe that if $u \in S$, then necessarily $|u| \leq L/\varepsilon$. Let S_N be the random variable describing the string output by PNFA N. Then by Markov's inequality we have
$\varepsilon \leq \Pr[S_N = u] \leq \Pr[|S_N| \geq |u|] \leq E[|S_N|]/|u| \leq L/|u|$
which completes the proof.

This is a generic construction whose value is only to show that finite-size approximation of PNFA (and HMM) by PDFA is always possible. However, the machine we construct for the proof does not necessarily capture the internal structure of the PNFA/HMM. But the fact that PDFAs can be used to approximate HMMs suggests a new class of algorithms that could be used to learn HMMs. More precisely, one can think of trying to learn a PDFA that approximates an HMM. The size of the PDFA would depend on factors such as the desired degree of accuracy, and the amount of data available.

PDFAs and PNFAs have been studied extensively in computational learning theory, especially in the context of PAC-learning. In this context, the goal of learning is to find a model that approximates the true probability distribution over observation trajectories, $\mathcal{P}$. A learning algorithm will produce a model which generates a distribution over observation trajectories $\hat{\mathcal{P}}$. A model, or hypothesis, is called ε-good, if the distance between

$m(\mathcal{P}, \hat{\mathcal{P}}) < \varepsilon$, where m is a reasonable distance measure (e.g. L_∞ or the Kullback-Leibler divergence) and $\varepsilon > 0$ is the desired precision. Given observation trajectories that are drawn i.i.d. from the system, an error parameter $\varepsilon > 0$ and a confidence parameter $\delta \in (0,1)$, a PAC-learning algorithm must output an ε-good model with probability at least $1-\delta$. A class of machines is called efficiently PAC-learnable if there exists a PAC-learning algorithm whose time complexity is polynomial in $1/\varepsilon$, $1/\delta$ and the number of parameters of the target machine. A class of machines is polynomially PAC-learnable if the training sample (i.e. the number of trajectories needed) is polynomial in the same quantities.

Several PAC-style results have been established over the years on the topic of learning PDFAs and HMMs. (see Dupont et al, 2005 for a comprehensive discussion). Of particular relevance to our work is the result by Kearns et al. (1994) establishing that the class of all PDFAs is in fact *not* efficiently PAC-learnable. However Ron et al. (2005) argued that by restricting attention to the class of PDFAs that are acyclic and have a *distinguishability criterion* between states, PAC-learning is possible.

Definition 1. *Let m be a measure of the difference between two probability distributions. A PDFA has* ***distinguishability*** *μ if for any two states s and s', the probability distributions over observation trajectories starting at s and s', $\mathcal{P}_s$ and $\mathcal{P}_{s'}$, differ by at least μ: $m(\mathcal{P}_s, \mathcal{P}_{s'}) \geq \mu, \forall s, s' \in S$.*

Intuitively, this class of machines does not have states that are "too similar" in terms of the probability distribution of trajectories following them. More recently, Clark and Thollard (2004) provided an efficient PAC-learning algorithm for this subclass of PDFAs which requires an amount of data polynomial in the number of states in the target, the "distinguishability" of states and the expected length of strings generated from any state. In the next section, we build on their work to provide a learning algorithm for PDFAs/HMMs with PAC-style guarantees, then analyze this algorithm.

4 A PAC-learning algorithm

The algorithm builds a graph whose nodes intuitively represent postulated states of the target machine. We call these nodes "safe states". The algorithm also maintains a list of "candidate states" that will eventually be merged with existing safe states or promoted to be new safe states.

The algorithm uses both state splitting and state merging operations. We begin by assuming that the initial model is a trivial graph with a single safe state representing the initial state of the target machine. In the induction step, we refine the graph by adding a new safe state $s\sigma_i$, whenever the training data suggests that there is a sufficient difference between the probability distribution over the trajectories observed from $s\sigma_i$ and the distribution observed from any safe state s'. Similarly. if the distribution of trajectories observed from $s\sigma_i$ and an existing safe state s' are sufficiently similar, we merge (or identify) these states. The remainder of this section formalizes these basic ideas, including the precise criteria for creating new safe states and merging candidate states into existing safe states.

We assume that the set of possible observations Σ is known, and that we have a set of training trajectories D, with each trajectory being sampled i.i.d. from the correct target.

The algorithm assumes the following input parameters: δ, n, μ where δ is the desired confidence (as in standard PAC-learning YValiant1984), n is an upper bound on the number of states desired in the model, and μ is a lower bound on the distinguishability between any two states. We assume the L_∞ norm as the measure m (see Definition 1).

We begin by defining a single safe state $S = \{s_0\}$, labeled with a *null* observation. Then we consider a set of candidate states $s_0\sigma$ for every observation $\sigma \in \Sigma$. With each safe and candidate state, we associate a multiset, D_s and $D_{s\sigma}$ respectively, storing the suffixes of all training trajectories that pass through this state (or a sufficient statistic thereof).

For each given training trajectory $d = \sigma_0 \dots \sigma_{i-1}\sigma_i\sigma_{i+1} \dots \sigma_k$, we traverse the graph matching each observation σ_i to a state until either (1) all observations in d have been exhausted (in which case we discard d and proceed to the next training trajectory), or (2) a transition to a candidate state is reached. This occurs when all transitions up to σ_{i-1} are defined and lead to a safe state s, but there is no transition out of s with observation σ_i. In this case, we add the sub-trajectory $\{\sigma_{i+1} \dots \sigma_k\}$ to the multiset $D_{s\sigma_i}$.

The next step is to decide what to do about candidate state $s\sigma$. There are three possibilities: (1) **retain** it as a candidate state; (2) **merge** it with an existing state $s' \in S$; (3) **promote** it to be a new state $S = S \cup \{s\sigma\}$. This step is the core of the algorithm.

The decision of whether to merge, promote, or retain a candidate state depends on the content of its multiset $D_{s\sigma}$. To better explain this step, we introduce some notation, which applies both for safe and candidate states. We denote by $|D_s|$ the cardinality of D_s and by $D_s(d)$ the number of times trajectory d occurs in D_s. We denote by $|D_s(\sigma)|$ the number of trajectories starting with observation σ in multiset D_s. Note that $|D_s(d)|/|D_s|$ can be regarded as an empirical approximation of the probability that trajectory d will be observed starting from state s.

The decision of whether to retain a candidate is taken first. If the state is not retained, we then consider whether to promote it or merge it with an existing state. A candidate state $s\sigma$ is declared **large** when:

$$\text{(largeness condition) } |D_{s\sigma}| \geq \frac{3(1+\mu/4)}{(\mu/4)^2} \cdot \ln \frac{2}{\delta'} \tag{2}$$

where $\delta' = \frac{\delta\mu}{2(n|\Sigma|+2)}$. When a candidate state is declared large, it will not be retained. Intuitively, in this case there is enough information to promote or merge it correctly.

Suppose state $s\sigma$ has been declared large. If there exists some safe state s' such that for every trajectory d we have

$$\left| \frac{|D_{s\sigma}(d)|}{|D_{s\sigma}|} - \frac{|D_{s'}(d)|}{|D_{s'}|} \right| \leq \mu/2, \tag{3}$$

then we merge $s\sigma$ and s': we therefore remove $s\sigma$ as a candiate state, we create a transition from s to s' labelled with σ, and increase the counts of $|D_{s'}(d)|$ by those of $|D_{s\sigma}(d)|$.

If, on the contrary, for every s' there is a d such that

$$\left| \frac{|D_{s\sigma}(d)|}{|D_{s\sigma}|} - \frac{|D_{s'}(d)|}{|D_{s'}|} \right| > \mu/2$$

then we promote $s\sigma$ to be a new safe state; we add a transition from s to $s\sigma$ labelled with σ and add candidate states $s\sigma\sigma'$ for every observation $\sigma' \in \Sigma$. All trajectories in $D_{s\sigma}$ are distributed appropriately to these new candidate states, as if they had been observed from $s\sigma$.

The graph built as described above can easily be transformed into a PDFA. Every safe state becomes a state of the automaton. The set of observations Σ is the same. The observation probability function $O(s,\sigma)$ is calculated using the multiset statistics:

$$O(s,\sigma) = \frac{|D_s(\sigma)|}{\sum_{\sigma' \in \Sigma} |D_s(\sigma')|}(1 - (|\Sigma|+1)\gamma) + \gamma, \tag{4}$$

where $\gamma < \frac{1}{|\Sigma|+1}$ is a small smoothing probability (which can be set to 0 if smoothing is not desired).

The only real question left is what to do about the candidate states. Given a candidate state $s\sigma$, we look for the safe state s' that is most similar to it according to a chosen distance metric. E.g., assuming L_∞, we have $s' = \text{argmax}_{s' \in S}\left(\frac{|D_{s\sigma}(d)|}{|D_{s\sigma}|} - \frac{|D_{s'}(d)|}{|D_{s'}|}\right)$. We then add an edge from s to s' with label σ to the automaton M and calculate the observation probability as in Equation 4. Finally, the transition function is $T(s,\sigma) = s\sigma$.

Table 1. Learning Algorithm

M = PDFA-Learn $(\Sigma, D, \delta, n, \mu)$	
Initialize safe states $S = \{s_0\}$	INITIALIZING
$D_{s_0} = D$	
Initialize candidates $\bar{S} = \{s_0\sigma \mid \forall \sigma \in \Sigma\}$	
$D_{s_0\sigma} = \{\sigma_2 \ldots \sigma_k \mid \exists d \in D_{s_0}, d = \sigma\sigma_2 \ldots \sigma_k\}$	
While $\exists s\sigma \in \bar{S}$ which is large, as given by (2)	
Remove $s\sigma$ from $\bar{S}$	
If $\exists s' \in S$ such that $\forall d$ (3) is satisfied	MERGING
Add transition from s to s' labelled by σ	
$D_{s'} = D_{s'} \cup D_{s\sigma}$	
Else	PROMOTING
$s' = s\sigma$	
$S = S \cup \{s'\}$	
$D_{s'} = D_{s\sigma}$	
$\bar{S} = \bar{S} \cup \{s'\sigma' \mid \forall \sigma' \in \Sigma\}$	
$D_{s'\sigma} = \{\sigma_2 \ldots \sigma_k \mid \exists d \in D_{s'}, d = \sigma\sigma_2 \ldots \sigma_k\}$	
End if	
End while	
Construct the output graph.	

Table 1 summarizes the algorithm presented in this section. We note that as presented here, the algorithm works in batch mode. As such, there will necessarily be a point at which no candidate state meets the largeness condition, and the algorithm terminates. However, it is easy to imagine implementing this as an incremental algorithm, in which the graph is restructured after each trajectory is received. In this case, the

largeness condition will be checked every time a new trajectory is added to the multiset of a state. It is important to note that if the algorithm runs on-line, states can continue to become large as more data is gathered, and the machine will continue to grow. One possibility to stop this is to limit the number of acceptable states, using the parameter n. In Appendix A, we discuss a different, sufficient termination condition for this case. This is based on using the precision ε desired in the approximation of the trajectory distribution, and provides a strong improvement over the bounds of Clark & Thollard (2004) .

It is in general not necessary to recover a true HMM from the learned PDFA; we will consider the learned PDFA to be an approximation of the HMM, which can be used to compute (approximately) the probabilities of different trajectories. Not that an HMM can be recoverred followinng the steps outlines in Sec. 2. It should be noted that this output HMM may be of larger size than the target machine.

5 Analysis

A full analysis of the algorithm should show that 1) after seeing a certain number of examples, the graph under construction becomes isomorphic to that of the target machine, except for low-probability states, and that 2) in addition, after some more examples, the edge probabilities are close enough to the target ones that the distance in the probability distribution over trajectories is small. In this section we present a sketch of these proofs, highlighting the differences with results by Clark and Thollard (2004).

We first state how long it takes for a candidate state to become large. Observe that the more frequent a state is, the sooner it will be identified. In contrast, typical PAC approaches require a lower bound on the desired frequency, P, and run in time polynomial in $1/P$ even if most states have frequency much larger than P. No such parameter is required by our algorithm. This adaptive behavior shows good potential for the practicality of our approach.

Let $|\hat{D}_s|$ denote $E[|D_s|]$ and $|\hat{D}_s(d)|$ denote $E[|D_s(d)|]$.

Theorem 2. *(1) Let s be a candidate or safe node. At the time when s is declared large we have $||D_s| - |\hat{D}_s|| \leq |\hat{D}_s| \cdot (\mu/4)$ with probability $1-\delta'$. That is, $|D_s|$ is an approximation to $|\hat{D}_s|$ up to a multiplicative factor of $\mu/4$.*

(2) Let $s\sigma$ be a candidate node, and $p \cdot t$ be the expected value of $|\hat{D}_{s\sigma}|$ at time t Then $s\sigma$ is declared large at most

$$T = \frac{3(1+\mu/4)}{(1-\mu/4)(\mu/4)^2 p} \cdot \ln \frac{2}{\delta'}$$

steps after it was created, with probability at least $1-\delta'$.

Proof. The proof is technically similar to some used in (Lipton and Naughton, 1995) in the context of databases. For conciseness, let S_t denote $|D_s|$ at time t, $\mu' = \mu/4$ and B be the threshold of largeness, i.e.,

$$B = \frac{3(1+\mu')}{(\mu')^2} \cdot \ln \frac{2}{\delta'}.$$

Observe that part (1) of the theorem is falsified iff $(1-\mu')E[S_t] \leq S_t \leq (1+\mu')E[S_t]$ is false at the time t in which the node is declared large; note that t is a random variable. By definition, at the stopping time t we have $S_t = B$, and $E[S_t] = p \cdot t$. Hence, (1) is falsified if either $t \leq B/(1+\mu')p = (\text{def.})T^-$ or $t > B/(1-\mu')p = (\text{def.})T^+$, which is equivalent to saying $S_{T^-} \geq B$ or $S_{T^+} < B$. Thus, to show part (1) it suffices to show that the probability of each of these two events is at most $\delta'/2$. Note that T^+ is the T appearing in part (2) of the claim, so this also proves part (2). For the first one, observe that

$$\begin{aligned}\Pr[S_{T^-} \geq B] &= \Pr[S_{T^-} \geq (1+\mu')E[S_{T^-}]] \\ &\leq \exp\left(-\frac{1}{3}\mu'^2 E[S_{T^-}]\right) \leq \delta'/2,\end{aligned}$$

using (twice) that $B = (1+\mu')E[S_{T^-}]$, Chernoff bounds, and the definition of B. The proof that $\Pr[S_{T^+} < B] \leq \delta'/2$ is essentially identical.

Theorem 3. *The largeness condition in Equation (2) guarantees that, for any large state s,*

$$\forall d \quad \left|\frac{|D_s(d)|}{|D_s|} - \frac{|\hat{D}_s(d)|}{|\hat{D}_s|}\right| < \frac{\mu}{4} \tag{5}$$

with probability $1-\delta$.

The proof is essentially given in Section 6.1 of (Clark and Thollard, 2004).

From this claim, one can argue that the decisions to merge and promote candidate states are correct with high probability. Indeed, suppose that at any point we decide to merge $s\sigma$ with s'. This is because $s\sigma$ has become large and

$$\forall d, \left|\frac{|D_{s\sigma}(d)|}{|D_{s\sigma|}} - \frac{|D_{s'}(d)|}{|D_{s'}|}\right| \leq \mu/2.$$

Then by the claim and the triangle inequality we have

$$\left|\frac{\hat{D}_{s\sigma}(d)}{|\hat{D}_{s\sigma}|} - \frac{\hat{D}_{s'}(d)}{|\hat{D}_{s'}|}\right| < \mu.$$

Under the assumption that any two states in the target machine are μ-distinguishable, we conclude that $s\sigma$ and s' indeed reach the same state in the target machine.

Similarly, suppose that we decide to promote $s\sigma$ to be a new safe state. This is because for every s' there is some d such that

$$\left|\frac{|D_{s\sigma}(d)|}{|D_{s\sigma|}} - \frac{|D_{s'}(d)|}{|D_{s'}|}\right| > \mu/2.$$

Then by the claim and the triangle inequality we have

$$\left|\frac{\hat{D}_{s\sigma}(d)}{|\hat{D}_{s\sigma}|} - \frac{\hat{D}_{s'}(d)}{|\hat{D}_{s'}|}\right| > 0.$$

So, assuming μ-distinguishability, we know that $s\sigma$ reaches a state not reached by any safe s' in the target machine.

Finally with these claims one can make the following argument: suppose that every state in the target machine can be reached by some path containing only transitions of probability $\geq p$. Then, every candidate state will be either promoted or merged correctly in time T, where T is given by Theorem 2. Therefore, by time at most $n \cdot T$, no candidate states will be left, and the graph constructed will be isomorphic to the graph of the target machine.

In other words, if any candidate states remain after time $n \cdot T$, they should have probability less than p. Thus we can show that for sufficiently low p, these nonfrequent states can be ignored without introducing large errors.

6 Illustration

We consider a few examples to illustrate the empirical behaviour of the algorithm. Consider first a synthetic text generator with a simple alphabet $\Sigma = \{a, b, \#\}$, which is designed to generate only three words $d = \{abb, aaa, bba\}$ and where # indicates word termination. We can make a generative model for this text generator using an HMM as shown in Figure 1. All observations are deterministic, transitions are also deterministic, except from $s10$, and the initial state distribution is the same as transitions from $s10$.

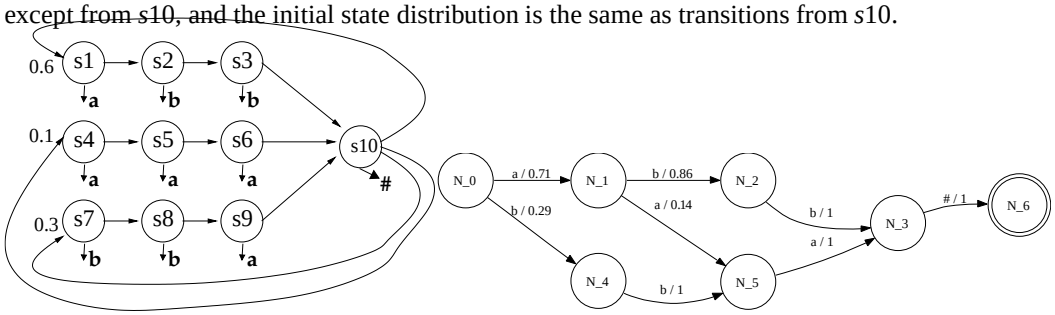

Fig. 1. A simple text-generation HMM (left) and the learned model (right)

We generate a number of trajectories from this HMM and apply the algorithm presented in Section 4 (using $\delta = 0.05$, $n = 8$, $\mu = 0.1$). Figure **??** shows the model that is learned. Nodes represent safe states. Edges are annotated by an observation and its probability (zero probability observations are not shown).

We now modify the HMM to produce noisy observations and repeat the experiment. We assume each state in Figure 1 generates the character shown with $P = 0.9$, and generates the other charather (i.e. "b" instead of "a" and vice-versa) with $P = 0.1$. In this case, as shown in Figure 2, our algorithm learns a slightly more complex model to account for the greater number of possible trajectories. It is easy to verify that the models shown in Figure **??** and 2 generate the observation strings with the same probability as the corresponding HMM.

Figure **??** shows the bound on the number of samples required as a function of the desired model precision. The increased data requirement with low ε values are natural,

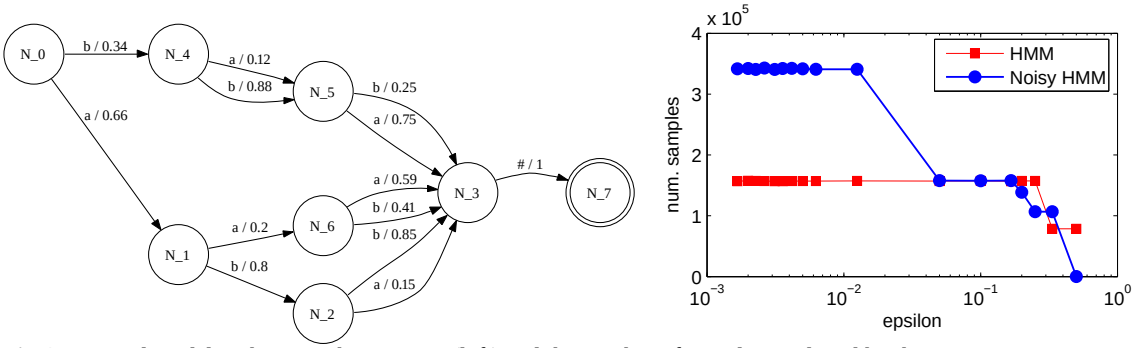

Fig. 2. Learned model with noisy observations (left) and the number of samples predicted by the PAC bounds for achieving the desired model precision (right). The noisy observation case is in blue.

since the size of the model must grow to achieve this increased precision. As expected, greater amounts of data are required when learning with noisy observations.

Next, we learn a model for a maze navigation domain called Cheese, illustrated in Figure 3. We modify the original problem slightly as follows. We assume the agent randomly starts in states $s5$ or $s7$. We assume a single action *float* which moves the agent to any adjacent cell with equal probability. The task resets whenever the agent gets to $s10$. Observations are generated deterministically and correspond to the number of walls in each state ("a"=1 wall, "b"=2 walls, "c"=3 walls), with the exception of $s10$ which produces a distinct terminal observation ("#").

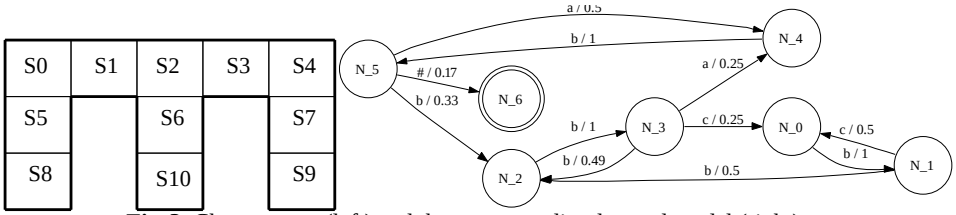

Fig. 3. Cheese maze (left) and the corresponding learned model (right)

Figure **??** shows the results of applying our learning algorithm. It is interesting to note the particular structure learned by our model. The states can be seen to represent core beliefs of the HMM after each *float* action (and before the observation is seen). For example, the states of the learned model in the figure represent the following: N_0: $Pr(s5) = Pr(s7) = 0.5$;
N_1: $Pr(s8) = Pr(s0) = Pr(s4) = Pr(s9) = 0.25$;
N_2: $Pr(s5) = Pr(s1) = Pr(s3) = Pr(s7) = 0.25$;
N_3: $Pr(s8) = Pr(s9) = 0.125$, $Pr(s0) = Pr(s4) = Pr(s2) = 0.25$;
N_4: $Pr(s1) = Pr(s3) = Pr(s6) = 0.333$;
N_5: $Pr(s0) = Pr(s4) = Pr(s10) = 0.0833$, $Pr(s2) = 0.5$;
N_6: end of trajectory.
This confirms that the graph learned is not arbitrary and has a nice structural interpretation.

7 Discussion and future work

There is considerable literature devoted to learning for all of the models introduced above. In HMMs, most of the existing work uses expectation maximization, like the

ones described in (Rabiner, 1989) . In these algorithms, the number of hidden states is assumed known. The algorithm starts with a guess about the parameters of the model and modifies this guess in such a way as to improve the likelihood of the observed data.

Several papers have looked at removing assumptions about the model topology. State splitting/merging approaches exist for learning PDFAs (Carrasco and Oncina, 1994; Ron et al, 2005; Thollard et al, 2000) and HMMs (Stolcke et al, 1992, Ostendorf et al, 1997). However the criterion for splitting/merging is typically heuristic or Bayesian in nature and does not provide correcteness guarantees.

More recent procedures rely on finding a minimal linear basis for the space of possible trajectories, by using techniques similar to singular value decomposition or principal component analysis (Jaeger et al, 2006,Singh et al, 2003, Rosencrantx et al, 2004). These procedures aim to find a globally or locally optimal solution in the sense of the L_2 norm. Usually, very large amounts of data are required for a good solution, and no PAC-style guarantees exist yet.

In contrast to existing work, we developed an algorithm that learns a PDFA that approximates an HMM. The algorithm addresses the problem of joint topology and parameter inference in Markov models with hidden state. We provided improved theoretical guarantees for PAC-learning of PDFAs from data, and described a natural extension to learning HMMs and POMDPs. This paper highlights important connections between the literature on learning automata and the problem of HMM and POMDP learning. Preliminary empirical results suggest that the algorithm learns correct models for simple HMMs. Further experiments will be conducted to better investigate generality and scalability of the approach.

References

Carrasco, R. and Oncina, J. "Learning stochastic regular grammars by means of a state merging method". LNAI 862. 1994.

Clark, A. and Thollard, F. "PAC-learnability of Probabilistic Deterministic Finite State Automata". Journal of Machine Learning Research, 5. 2004.

Dupont, P., Denis, F. and Esposito, Y. "Links between Probabilistic Automata and Hidden Markov Models". Pattern Recognition, 38 (9). 2005.

Durbin, R., Eddy, S.R., Krogh, A., Mitchison, G.J. "Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids". Cambridge University Press. 1998.

Jaeger, H., Zhao, M. and Kolling, A. "Efficient estimation of OOMs". In *Proceedings of NIPS*. 2005.

Kearns, M., Mansour, Y., Ron, D., Rubinfeld, R., Schapire, R. E. and Sellie, L. "On the learnability of discrete distributions". ACM Symposium on the Theory of Computing. 1995.

Lipton, R.J. and Naughton, J.F. "Query size estimation by adaptive sampling", J. Computer and System Sciences 51, 1995, 18–25.

Ostendorf, M. and Singer, H. "HMM topology design using maximum likelihood successive state splitting". Computer Speech and Language, 11. 1997.

Rabiner, L. R. "A tutorial on Hidden Markov Models and Selected Applications in Speech Recognition". Proceedings of the IEEE, 77(2). 1989.

Ron, D., Singer, Y. and Tishby, N. "On the learnability and usage of acyclic probabilistic finite automata". In *Proceedings of COLT*,1995.

Rosencrantz, M., Gordon, G. and Thrun, S. "Learning Low Dimensional Predictive Representations". In *Proceedings of ICML*, 2004.

Singh, S., Littman, M. L., Jong, N. K., Pardoe, D. and Stone, P. "Learning Predictive State Representations". In *Proceedings of ICML*, 2003.

Stolcke, A. and Omohundro, S. M. "Hidden Markov Model Induction by Bayesian Model Merging". In *Proceedings of NIPS*, 1993.

Thollard, F., Dupont, P. and Higuera, C. de la. "Probabilistic DFA Inference using Kullback-Leibler Divergence and Minimality". In *Proceedings of ICML*, 2000.

Valiant, L. "A theory of the learnable" Communications of the ACM, 27(11). 1984.

Appendix A: A termination condition for on-line learning

Suppose that the target model M^* would not only let us sample trajectories d, but also provide their true probability of occurring, $p_{M^*}(d)$. We will let the PDFA construction algorithm proceed until the distance between the target model M^* and the current model M is estimated to be less than a desired error parameter ε. Clearly, this step, and hence the running time of the algorithm, depend on the chosen notion of distance.

We implemented the following test, based on the L_∞ distance. For a suitably defined B, draw B trajectories from M^*, and obtain their probabilities, $p_{M^*}(d)$. For every trajectory d we compute its probability using the learned model so far, $p_M(d)$. If there is some d such that $|p_M(d) - p_{M^*}(d)| \geq \varepsilon$, we will consider that $L_\infty(M, M^*) \geq \varepsilon$ and let the learning continue. Otherwise we will consider that $L_\infty(M, M^*) \leq \varepsilon$ and terminate.

We set $B = \frac{3}{(\varepsilon/4)^2} \cdot \ln \frac{8}{\delta\varepsilon}$. We will now show that this test gives the correct answer whenever $L_\infty(M, M^*) \leq \varepsilon/2$ or $L_\infty(M, M^*) \geq 3\varepsilon/2$, i.e., when the L_∞ is at a certain distance from ε either way.

Claim. Let D_1 and D_2 be the two probability distributions to which the test is applied. With probability $1 - \delta$, if $L_\infty(D_1, D_2) \geq 3\varepsilon/2$ then the test above says "distance greater than ε", and if if $L_\infty(D_1, D_2) \leq \varepsilon/2$ it says 'distance less than ε"

Proof. Let $p_1(d)$ and $p_2(d)$ denote the probabilities of trajectory d as observed by the test on the B examples. One can show (as in Thm 3), that B is sufficiently large that
$\Pr[\exists d : |(p_1(d) - D_1(d)| \geq \varepsilon/4] \leq \delta/2$, and
$\Pr[\exists d : |(p_2(d) - D_2(d)| \geq \varepsilon/4] \leq \delta/2$
and the claim then holds by applying triangle inequality.

It can be furthermore shown that the distance between hypothesis and target machines will be below ε in a number of steps polynomial in the parameters: $1/\varepsilon$, $1/\mu$, $\ln(1/\delta)$, n, as well as the expected length of strings generated at any step, L. More precisely it is shown in YClark and Thollard2004 that a batch of examples

$$O\left(\frac{n^5 L^5}{\varepsilon^2} \cdot \max\left\{\frac{1}{\mu^2}, \frac{L^4}{\varepsilon^4}\right\} \cdot \ln\frac{1}{\delta}\right)$$

(ignoring some logarithmic factors) will suffice to bring even the KL-divergence between both machines below ε. This is clearly an impractical bound, even if their analysis could be tightened to some extent. The same analysis could be carried out for our algorithm. It is difficult to make formal claims about the extent our algorithm could be faster. We expect, however, that it might require much less data than suggested by the above bound.