PAC-Learning of Markov Models with Hidden State

Ricard Gavaldà [1] Philipp W. Keller [2]
Joelle Pineau [2] Doina Precup [2]

[1]Univ. Politècnica de Catalunya, Barcelona
[2]McGill University, Montréal

(work presented in ECML'06)

Outline

1. Introduction
2. HMM and PDFA
3. PAC-Learning PDFA
4. Our algorithm
5. Analysis
6. Experiments
7. Conclusions

Hidden Markov Models

- Hidden Markov Models (HMM) useful for prediction under uncertainty
- HMM generates probability distribution on sequences of observations (or action/observation pairs)
- Learning problem: Given sample of sequences of observations infer an HMM generating a similar distribution
- Standard approach: Expectation Maximization (EM) to approximate target's parameters [Rabiner89]
- Drawbacks:
  1. requires previous knowledge of state set - not always available
  2. converges to local minimum – how far from optimum?

Summary of Results

- We use Probabilistic Deterministic Finite Automata as approximations of HMM
- We give a learning algorithm for PDFA
  - that infers both state representations and parameters
  - has formal guarantees of performance – PAC-learning
- We test on (very small) simple dynamical systems - promising results

Previous work

Learning HMM without prior knowledge of states:

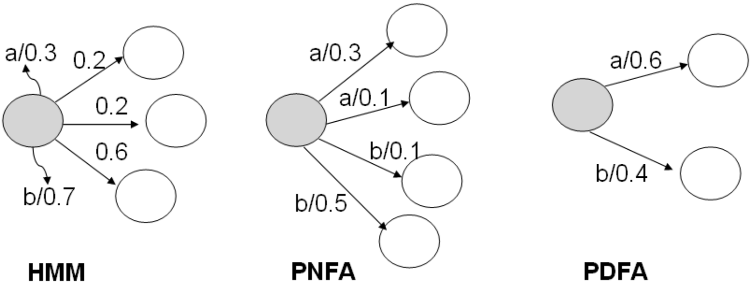
- Predictive State Representations [Jaeger et al 05, Rosencrantz et al 04, Singh et al 03].
  No formal guarantees, millions of examples.
- PAC-style: [Ron et al 95] [Clark & Tholard 04]: basis of our work
- [Holmes & Isbell 06]: similar to ours, deterministic systems

HMM, PNFA, PDFA

- Finite set of observations or letters
- Finite set of states
- Probabilities on transitions between states
- HMM: *States* emit observations, probabilistically
- PNFA, PDFA: *Transitions* emit observations, probabilistically

HMM, PNFA, PDFA

- N = Nondeterministic: Each (state,letter) leads to many states
- D = Deterministic: Fixing (state,observation) fixes next state



Relation between models

- HMM n states $\longrightarrow$ PNFA n states
- PNFA n states $\longrightarrow$ HMM n^2 states
- Some finite-size PNFA/HMM only have infinite-size PDFA
- *But:*

For every PNFA M and every ϵ there is a finite-size PDFA that approximates M within precision ϵ in L_∞ distance

Distribution distances

Definition

For two distributions D_1, D_2,

$$\begin{aligned} L_\infty(D_1, D_2) &= \max_x |D_1(x) - D_2(x)| \\ KLD(D_1 \| D_2) &= \sum_x D_1(x) \, \log \frac{D_1(x)}{D_2(x)} \end{aligned}$$

What do we mean by learning?

Definition

An algorithm PAC-learns PDFA if for every target PDFA M, every ϵ, every δ it produces a PDFA M' such that

$$\Pr[\, KLD(D(M) \| D(M')) \geq \epsilon \,] \leq \delta$$

in time $poly(size(M), 1/\epsilon, 1/\delta)$.

Unfortunately this is impossible [Kearns et al05]

What do we mean by learning?

- [Ron et al 96] Learning becomes possible by
  - restricting to *acyclic* PDFA and
  - considering *distinguishability* parameter μ
- [Clark&Thollard 04] Works for cyclic automata if we consider a new parameter L = bound on expected length of generated strings

  They learn in the KLD sense in time $\mathit{poly}(n, 1/\epsilon, \ln(1/\delta), 1/\mu, L)$

Distinguishability

Definition

- States q and q' are μ-distinguishable if

$$L_\infty(D(q), D(q')) \geq \mu,$$

  where $D(q)$ is the distribution of strings generated from q
- A PDFA is μ-distinguishable if every two states in it are μ-distinguishable

The C&T algorithm: promise and drawbacks

It provably PAC-learns. But:

- Asks for parameters ϵ, δ, ... and n, μ, L (guesswork)
- Requires full sample up-front:

  read parameters;
  compute $m = poly(\epsilon, \delta, n, \mu, L)$;
  get sample of size m;
  build pdfa from sample
- Always worst-case: as many samples as worst target PDFA!
- Polynomial is huge:
  for $n = L = 3$, $\epsilon = \delta = \mu = 0.1 \rightarrow m > 10^{20}$
- Analysis certainly not tight. Is this cost unavoidable?

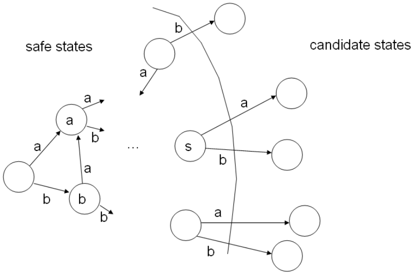
Our approach

Based on [C&T04], but:

- No need to give L and ϵ as parameters if m is fixed;
- Improved analysis:
  - separates time to get graph and time to tune parameters
  - time to get state graph independent of ϵ, L
  - this time smaller for “easier” graphs

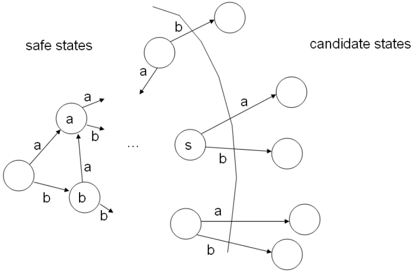
Data structures

- Graph with “safe” and “candidate” states
- Safe state s: represents state where string s ends
- Candidate state: pair (s, σ) where $next(s, \sigma)$ still unclear
- Invariant: all safe states are really distinct in target



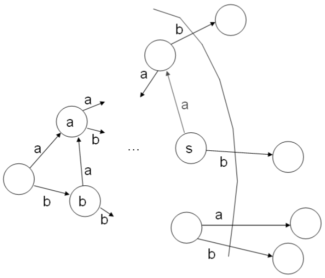
Growing the graph

- A candidate state can be *promoted* to safe or *merged* with an existing safe state
- Keep a multiset $D_{s,\sigma}$ for each candidate (s, σ)
- $D_{s,\sigma}$ sample of distribution from state reached by $s \cdot \sigma$



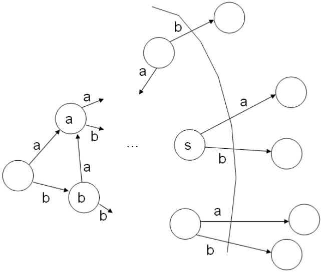
Growing the graph

- A candidate state can be *promoted* to safe or *merged* with an existing safe state
- Keep a multiset $D_{s,\sigma}$ for each candidate (s, σ)
- $D_{s,\sigma}$ sample of distribution from state reached by $s \cdot \sigma$



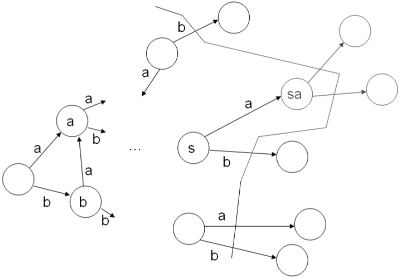
Growing the graph

- A candidate state can be *promoted* to safe or *merged* with an existing safe state
- Keep a multiset $D_{s,\sigma}$ for each candidate (s, σ)
- $D_{s,\sigma}$ sample of distribution from state reached by $s \cdot \sigma$



Growing the graph

- A candidate state can be *promoted* to safe or *merged* with an existing safe state
- Keep a multiset $D_{s,\sigma}$ for each candidate (s,σ)
- $D_{s,\sigma}$ sample of distribution from state reached by $s \cdot \sigma$



The algorithm

1. define safe initial state, labelled with empty string;
2. define candidate states out of initial state, one per letter;
3. **while** there are samples left **do**
4.   run next sample through current graph;
5.   **if** it ends in a candidate state (s, σ) **then**
6.   let w be the unprocessed part of sample;
7.   store w in $D_{s,\sigma}$;
8.   if $D_{s,\sigma}$ large enough, either merge or promote (s, σ);
9.   **endif**
10. **endwhile**
11. build PDFA from current graph

Merging and promoting states

Largeness condition: $D_{s,\sigma}$ has size at least

$$T = \frac{c}{\mu^2} \cdot \ln \frac{n|\Sigma|}{\delta}$$

Assuming μ-distinguishable target, we can then decide reliably:

- if distributions observed at (s, σ) and *some* safe state s' are $\mu/2$-close $\rightarrow$ identify (s, σ) and s', i.e., set $next(s, \sigma) = s'$
- else, (s, σ) is $\mu/2$-far from *all* safe states $\rightarrow$ promote (s, σ) to safe state labelled $s\sigma$, create new candidate states
- rerun strings in $D_{s,\sigma}$ from merged/promoted state

Building the PDFA from the graph

- Identify each remaining candidate states with a closest safe state;
- Compute transition probabilities in obvious way:

$$\Pr[\, s \xrightarrow{\sigma} s' \,] = \frac{\text{\#samples using } (s \xrightarrow{\sigma} s')}{\text{\#samples passing through } s}$$

(maybe with some smoothing)

Main claim 1: time to learn topology

Lemma

Suppose a target state q is reachable by a path of length ℓ all whose edges have absolute probability $\geq p$. Then q has a corresponding safe state in the graph by time at most

$$\frac{\ell}{p} \cdot O(T) = O\left(\frac{\ell}{\mu^2 p} \cdot \ln \frac{n|\Sigma|}{\delta}\right)$$

- Time depends on unknown ℓ and p: easier states are found faster
- No dependence on ϵ; on L, indirectly via p

Main claim 2: time to learn parameters

Lemma

Suppose the built graph is isomorphic to target graph; if we see

$$poly(n, 1/\epsilon, \ln(1/\delta), 1/\mu, L)$$

additional samples, the PDFA obtained from the graph satisfies the PAC-learning criterion

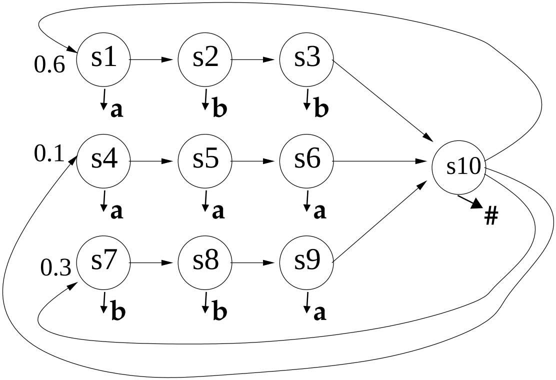
[proof basically as in Clark&Thollard04]

Wrap-up

- Lemma 1 states time to identify non-negligible states
- Lemma 2 states time to approximate transition probabilities
- Together, we recover [Clark&Thollard04] PAC-guarantees
- But with less parameters, faster in non-worst-case situations

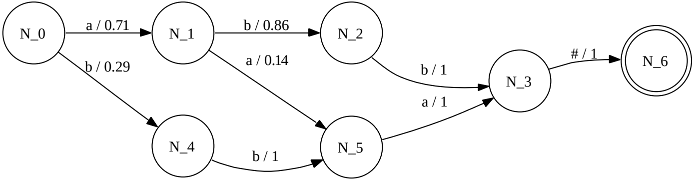
Simple text generation

- alphabet = $\{a, b, \#\}$, $\#$ as word separator
- HMM generates only $\{abb, aaa, bba\}$



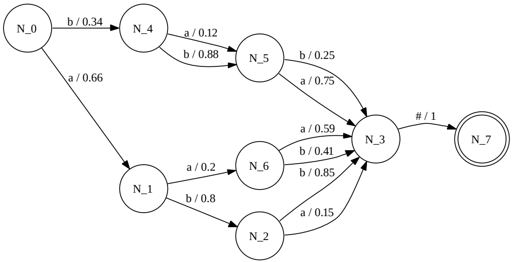
Simple text generation

- alphabet = $\{a, b, \#\}$, $\#$ as word separator
- HMM generates only $\{abb, aaa, bba\}$

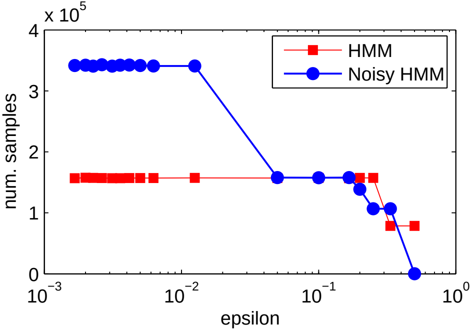


Simple text generation

- alphabet = $\{a, b, \#\}$, $\#$ as word separator
- HMM generates only $\{abb, aaa, bba\}$
- Noisy: flip letter with probability 0.1

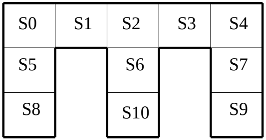


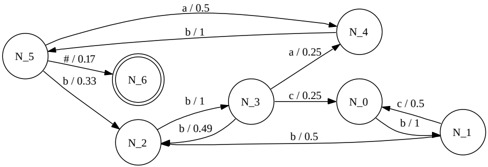
Samples to achieve desired prediction



Cheese maze experiment

- observations: a = 1 wall; b = 2 walls; c = 3 walls
- move to random neighbor
- task resets whenever we reach s10
- each state of learned PDFA has natural interpretation
- e.g. $N_5 =$ "We're at S5 or S7, prob. 0.5 each"





Conclusions

- A PAC-learning algorithm for learning HMM as PDFA
- Learns state structure as well as transition probabilities
- # samples order of 10^5 where theory said $> 10^{20}$

Future work:

- Extend to distances other than L_∞
- No need to input μ
- Reduce number of samples (by tighter analysis)
- [Denis et al 06] PAC-learn full class of PNFA. Practical?