Towards Feasible PAC-Learning of Probabilistic Deterministic Finite Automata

Jorge Castro Ricard Gavaldà

LARCA Research Group, Departament LSI
Univ. Politècnica de Catalunya, Barcelona

ICGI'08, september 2008

PFA and PDFA

- Finite alphabet, finite set of states
- PFA, Probabilistic Finite State Automata:
  Each state has a probability distribution on transitions out it
- PDFA, Probablistic Deterministic Finite Automata:
  One transition per pair (state,letter)

- Every PFA M defines a probability distribution on strings $D(M)$, a.k.a. a stochastic language

Learning PDFA

- Many algorithms to learn PDFA, either heuristically or provably in the limit
- [Clark-Thollard 04] An algorithm that provably learns in a PAC-like framework from polynomial-size samples
- Followup papers, slightly different frameworks:
  - [Palmer-Goldberg 05, Guttman et al 05, G-Keller-Pineau-Precup 06]
- Sample sizes are polynomial, but huge for practical parameters

Our contribution

- A variation of the Clark-Thollard algorithm for learning PDFA
  - that has formal guarantees of performance: PAC-learning w.r.t. KL-divergence
  - does not require unknown parameters as input
- Potentially much more efficient:
  - Finer notion of *state distinguishability*
  - More efficient test to decide state merging/splitting
  - Adapts to complexity of target: faster on simpler problems
- Promising results on simple dynamical systems, and on a large weblog dataset

PAC-learning PDFA

- Let d be a measure of divergence among distributions
- Popular choice for d: Kullback-Leibler divergence

- Learning algorithm can sample $D(M)$ for unknown target PDFA M

Definition

An algorithm PAC-learns PDFA w.r.t. d if for every target PDFA M, every ϵ, every δ it produces a PDFA M' such that

$$\Pr[\, d(D(M), D(M')) \geq \epsilon\,] \leq \delta.$$

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

Previous Results

- PAC-learning PDFA this way may be impossible [Kearns et al 95]
- [Ron et al 96] Learning becomes possible by
  - considering *acyclic* PDFA
  - introducing a distinguishability parameter μ
    = bound on how similar two states can be
- [Clark-Thollard 04]
  - Extends to cyclic PDFA considering parameter L
    = bound on expected length of generated strings.
  - Provably PAC-learns w.r.t. Kullback-Leibler divergence

The C&T algorithm: promise and drawbacks

- It provably PAC-learns with sample size

$$poly(|\Sigma|, n, \ln\frac{1}{\delta}, \frac{1}{\epsilon}, \frac{1}{\mu}, L)$$

But

- Requires full sample up-front: Always worst-case sample size
- Polynomial is huge: for $n = 3$, $\epsilon = \delta = \mu = 0.1 \rightarrow m > 10^{24}$
- Parameters n, L, μ are user-entered – upper bounds, guesswork

Distinguishability

For a state q, $D_q =$ distribution on strings generated starting at q

L_∞-distinguishability

$$L_\infty\text{-dist}(q, q') = \max_{x \in \Sigma^\star} |D_q(x) - D_{q'}(x)|$$

$$L_\infty\text{-dist}(M) = \min_{q,q'} L_\infty\text{-dist}(q, q')$$

Distinguishability

For a state q, D_q = distribution on strings generated starting at q

L_∞-distinguishability

$$L_\infty\text{-dist}(q, q') = \max_{x \in \Sigma^\star} |D_q(x) - D_{q'}(x)|$$

$$L_\infty\text{-dist}(M) = \min_{q,q'} L_\infty\text{-dist}(q, q')$$

Prefix L_∞-distinguishability

prefL_∞-distinguishability

$$\text{pref}L_\infty\text{-dist}(q,q') = \max_{x\in\Sigma^\star} |D_q(x\Sigma^\star) - D_{q'}(x\Sigma^\star)|$$

$$\text{pref}L_\infty\text{-dist}(M) = \min_{q,q'} \max\{L_\infty\text{-dist}(q,q'), \text{pref}L_\infty\text{-dist}(q,q')\}$$

Obviously for every M

$$\text{pref}L_\infty\text{-dist}(M) \geq L_\infty\text{-dist}(M)$$

Prefix L_∞-distinguishability

prefL_∞-distinguishability

$$\text{pref}L_\infty\text{-dist}(q, q') = \max_{x \in \Sigma^\star} |D_q(x\Sigma^\star) - D_{q'}(x\Sigma^\star)|$$

$$\text{pref}L_\infty\text{-dist}(M) = \min_{q,q'} \max\{L_\infty\text{-dist}(q, q'), \text{pref}L_\infty\text{-dist}(q, q')\}$$

Obviously for every M

$$\text{pref}L_\infty\text{-dist}(M) \geq L_\infty\text{-dist}(M)$$

Data Structures

- Algorithm keeps a graph with “safe” and “candidate” states
- Safe state *s*: represents state where string *s* ends
- Invariant: Graph of safe states isomorphic to a subgraph of target

- Candidate state: pair (s, σ) where $\mathit{next}(s, \sigma)$ still unclear
- For each candidate (s, σ), keep multiset $B_{(s,\sigma)}$, sample of $D_{(s,\sigma)}$
- Eventually, all candidate states are promoted to safe states or merged with existing safe states

The Clark-Thollard algorithm

1. input $|\Sigma|$, n, δ, ϵ, μ, L

// Assumption:

// target is $\mu \geq$ distinguishability, $n \geq$ #states, $L \geq$expected length

2. compute $m = poly(|\Sigma|, n, \ln \frac{1}{\delta}, \frac{1}{\epsilon}, \frac{1}{\mu}, L)$
3. ask for sample S of size m
4. work on S, using again n, ϵ, μ, L
5. produce pdfa

Theorem

PAC-learning w.r.t. KL-divergence occurs

The Clark-Thollard algorithm

1. input $|\Sigma|$, n, δ, ϵ, μ, L
// Assumption:
// target is $\mu \geq$ distinguishability, $n \geq$ #states, $L \geq$expected length
2. compute $m = poly(|\Sigma|, n, \ln\frac{1}{\delta}, \frac{1}{\epsilon}, \frac{1}{\mu}, L)$
3. ask for sample S of size m
4. work on S, using again n, ϵ, μ, L
5. produce pdfa

Theorem

PAC-learning w.r.t. KL-divergence occurs

Our algorithm

1. input $|\Sigma|$, δ, available sample S
2. work on S
3. produce pdfa

Theorem

If $|S| \geq poly(|\Sigma|, n, \ln\frac{1}{\delta}, \frac{1}{\epsilon}, \frac{1}{\mu}, L)$, *then*

PAC-learning w.r.t. KL-divergence occurs

(n =#target states,
$\mu =$ prefL_∞-dist(target),
L =expected-length(target))

Our algorithm

1. input $|\Sigma|$, δ, available sample S
2. work on S
3. produce pdfa

Theorem

If $|S| \geq poly(|\Sigma|, n, \ln \frac{1}{\delta}, \frac{1}{\epsilon}, \frac{1}{\mu}, L)$, *then*

PAC-learning w.r.t. KL-divergence occurs

($n =$#target states,
$\mu =$ prefL_∞-dist(target),
$L =$expected-length(target))

Our algorithm, more precisely

1. input $|\Sigma|$, δ, available sample
2. define initial safe state, labelled with empty string
3. define candidate states out of initial state, one per letter
4. **while** there are candidate states left **do**
5. process the whole sample, growing sets $B_{(s,\sigma)}$
6. choose candidate state (s,σ) with largest set $B_{(s,\sigma)}$
7. either merge or promote (s,σ)
8. **endwhile**
9. build PDFA from current graph
10. set transition probabilities & smooth out

Processing the sample

1. **foreach** string x in sample
2. **if** x ends in a candidate state (s, σ) **then**
3. let w be the unprocessed part of x
4. store w in $B_{s,\sigma}$
5. **endif**
6. **end foreach**

Criterion for merging or promoting

1. Let (s, σ) be chosen candidate state
2. **foreach** safe s' **do**
3. run statistical test for distinct distributions of $B_{(s,\sigma)}$ and $B_{s'}$
4. **if** all tests passed
5. // w.h.p. (s, σ) is distinct from all existing states
6. promote (s, σ) as a new safe state
6. **else**
7. // some test failed: (s, σ) similar to an existing safe state s'
8. identify (merge) (s, σ) with s'

Criterion for merging or promoting

Criterion in [CT04]:

- Always makes correct decision (w.h.p. $\geq 1 - \delta$)
- But decides only if $B(s, \sigma)$ large enough
- Based on μ: “large enough” always worst case

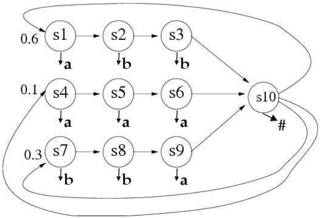
Our criterion:

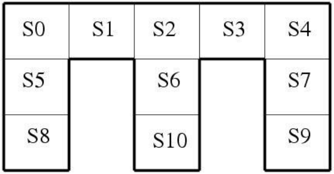
- Always decides after whole sample processed
- Decision may be wrong if sample is too small!
- But is correct if sample is large enough (w.r.t. *these* states)
- No knowledge of μ
- Plus finer math to avoid excess resampling

Implementation

- Goal: Sanity check
- 100% PAC algorithm for graph identification: no cutting corners!
- No smoothing in second phase yet: learn w.r.t. L_1 rather than KL
- Slow; optimizations in progress, but keep it PAC

Simple dynamical processes



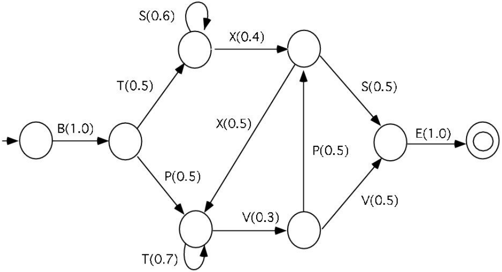


From [G et al, ecml06], another implementation of Clark-Thollard:

- HMM generating $\{abb, aaa, bba\}$
- Cheese maze HMM: state = position in maze; observed = #walls
- Implementation described there required $\geq 10^5$ samples to identify structure

Simple dynamical processes

Reber grammar [Carrasco-Oncina 99]:



Simple dynamical processes

- Three 10-state machines, alphabet size 2 or 3
- Graph is correctly identified by our algorithm with 200-500 samples
- Comparable sample size reported for heuristic (non PAC-guaranteed) methods

A large dataset

- Log from an ecommerce website selling flights, hotels, car rental, show tickets...
- 91 distinct “pages”, 120,000 user sessions, average length 12 clicks
- definitely NOT generated by a PDFA

- Our algorithm produces a nontrivial 50-60-state PDFA
- L_1 distance to dataset ≈ 0.44 – baseline is ≈ 0.39

A large dataset

- Log from an ecommerce website selling flights, hotels, car rental, show tickets. . .
- 91 distinct “pages”, 120,000 user sessions, average length 12 clicks
- definitely NOT generated by a PDFA

- Our algorithm produces a nontrivial 50-60-state PDFA
- L_1 distance to dataset ≈ 0.44 – baseline is ≈ 0.39

Conclusions

- An algorithm for learning PDFA with PAC guarantees
- # samples order of $200 - 1000$ where theory predicts 10^{20}

Future work:

- Extend to distances other than L_∞
- Other notions of distinguishability?
- [Denis et al 06] PAC-learn full class of PNFA. Practical?

Conclusions

- An algorithm for learning PDFA with PAC guarantees
- # samples order of $200 - 1000$ where theory predicts 10^{20}

Future work:

- Extend to distances other than L_∞
- Other notions of distinguishability?
- [Denis et al 06] PAC-learn full class of PNFA. Practical?