Bootstrapping and Learning PDFA in Data Streams

Borja Balle, Jorge Castro, Ricard Gavaldà

LARCA. Laboratory for Relational Algorithmics, Complexity and Learning

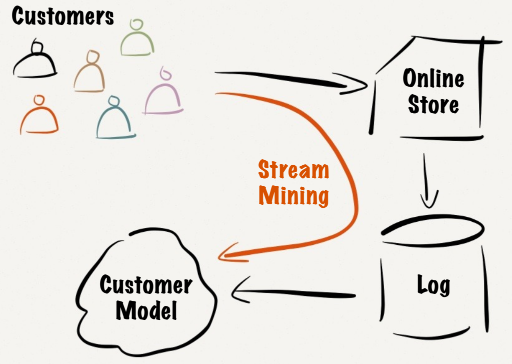
UNIVERSITAT POLITÈCNICA DE CATALUNYA

International Colloquium on Grammatical Inference

University of Maryland, September 2012

This work is partially supported by the PASCAL2 Network

Example Application: Web User Modeling



"Wish List"

- Process examples *as fast as they arrive* (10^5 per sec. or more)
- Use *small amount of memory* (must fit into machine's main memory)
- Detect *changes* in customer behavior and *adapt* the model accordingly

Other Applications: Process Mining, Biological Models (DNA and aminoacid sequences)

Outline

Outline

The Data Streams Algorithmic Model

An algorithm receives an infinite stream $x_1, x_2, \ldots, x_t, \ldots$ from some domain X and must:

- Make only one pass over the data and process each item in time $O(1)$
- At every time t use sublinear memory (e.g. $O(\log t)$, $O(\sqrt{t})$)
- Adapt to possible "changes" in the data

It is a theoretically challenging model useful for applications:

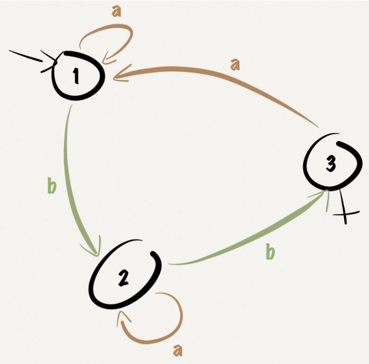
- Originated in the algorithmics community
- Realistic for Data Mining and Machine Learning tasks in real-time
- Feasible way to deal with Big Data problems

When studying learning problems with streaming data:

- In the worst case setting it resembles Gold's model (with algorithmic constraints)
- But we consider a PAC-style scenario where:
  - x_t are all independent and generated from a distribution D_t
  - the sequence of distributions $D_1, D_2, \ldots, D_t, \ldots$ either *changes very slowly* or presents only *abrupt changes but very rarely*

Hypothesis Class: PDFA

Probabilistic Deterministic Finite Automata = DFA + Probabilities



Transition/Stop probabilities

q	$p_q(a)$	$p_q(b)$	$p_q(\xi)$
1	0.3	0.7	0.0
2	0.5	0.5	0.0
3	0.8	0.0	0.2

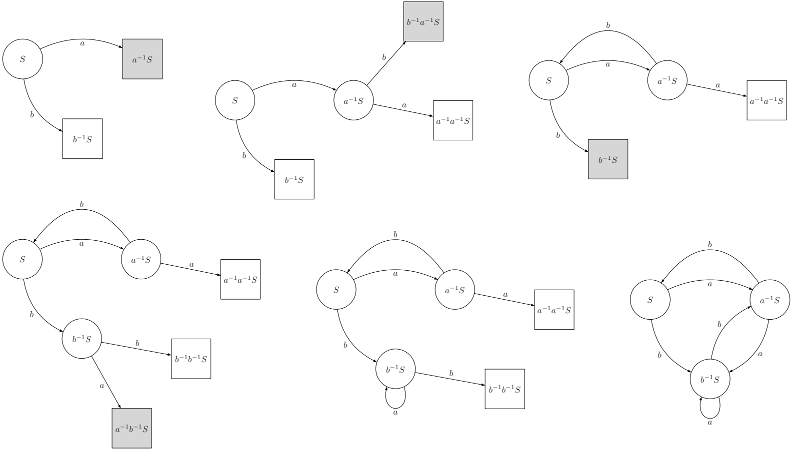
Parameters

- n (states)
- $|\Sigma|$ (alphabet)
- L (expected length)
- μ (distinguishability, L_∞)

$\mu = \min_{q \neq q'} \max_{x \in \Sigma^\star} |D_q(x) - D_{q'}(x)|$

State Merge/Split Algorithm

Usual approach to PDFA learning [Carrasco–Oncina '94, Ron et al. '98, Clark–Thollard '04, Palmer–Goldberg '05, Castro–Gavaldà '08, etc.]

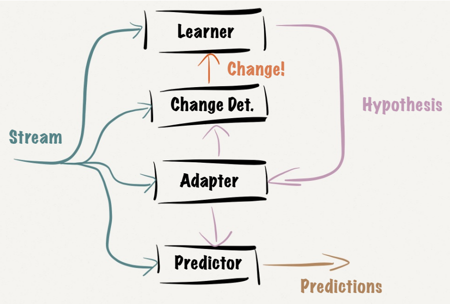


Statistical tests

$$S \not\approx a^{-1}S$$
$$S \approx b^{-1}a^{-1}S$$
$$S \not\approx b^{-1}S$$
$$a^{-1}S \not\approx b^{-1}S$$
$$b^{-1}S \approx a^{-1}a^{-1}S$$
$$b^{-1}S \approx b^{-1}b^{-1}S$$

Description of the Algorithm

System Architecture



Learner Module

initialize H with safe q_λ;
foreach $\sigma \in \Sigma$ **do**
    add a candidate q_σ to H;
    schedule insignificance and similiarity tests for q_σ;
foreach *string* x_t *in the stream* **do**
    foreach *decomposition* $x_t = wz$, *with* $w, z \in \Sigma^\star$ **do**
        if q_w *is defined* **then**
            add z to $\hat{S}_w$;
            if q_w *is a candidate* **and** $|\hat{S}_w|$ *is large enough* **then** call SimilarityTest(q_w, δ);
    foreach *candidate* q_w **do**
        if *it is time to test insignificance of* q_w **then**
            if $|\hat{S}_w|$ *is too small* **then** declare q_w insignificant;
            else schedule another insignificance test for q_w;
    if H *has more than* n *safes* **or** *there are no candidates left* **then**
    return H;

Sample Sketches for Similarity Testing

Note: Instead of keeping a sample S_w for each state q_w, the algorithm keeps a *sketch* $\hat{S}_w$ of each sample

A sketch using memory $O(1/\mu)$ should be enough:

- Given samples S, S' from distributions D, D'
- Algorithm wants to test $L_\infty(D, D') = 0$ or $L_\infty(D, D') \geqslant \mu$
- In the second case, if $|D(x) - D'(x)| \geqslant \mu$ then either $D(x) \geqslant \mu$ or $D'(x) \geqslant \mu$
- It is enough to find all strings with $D(x), D'(x) = \Omega(\mu)$, of which there are $O(1/\mu)$

In our algorithm, each sketch uses a **SpaceSaving** data structure [Metwally et al. '05]:

- Uses memory $O(1/\mu)$
- Finds every string whose probability is $\Omega(\mu)$ (frequent strings)
- And approximates their probability with enough accuracy
- Easier to implement than sketches based on hash functions

Properties of the Algorithm

Streaming-specific features

- Adaptive test scheduling (decide as soon as possible)
- Similarity test based on Vapnik–Chervonenkis bound (slow similarity detection)
- Use bootstrapped confidence intervals in tests (faster convergence)

Complexity Bounds (with any reasonable test)

- Time per example $O(L)$ (expected, amortized)
- The learner reads $O(n^2|\Sigma|^2/\epsilon\mu^2)$ examples (in expectation)
- Memory usage is $O(n|\Sigma|L/\mu)$ (roughly $O(\sqrt{t})$)

Outline

Testing Similarity between Probability Distributions

Goal: decide if $L_\infty(D, D') = 0$ or $L_\infty(D, D') \geqslant \mu$ from samples S, S'

Statistical Test Based on Empirical L_∞ (the "default")

- Let $\mu_\star = L_\infty(D, D')$ and compute $\hat{\mu} = L_\infty(S, S')$
- Compute Δ_l, Δ_u such that $\hat{\mu} - \Delta_l \leqslant \mu_\star \leqslant \hat{\mu} + \Delta_u$ holds w.h.p.
- If $\hat{\mu} - \Delta_l > 0$ decide $D \neq D'$
- If $\hat{\mu} + \Delta_u < \mu$ decide $D = D'$
- Else, wait for more examples

Problem: asymmetry — deciding *dissimilarity* is easier that deciding *similarity*

- When $D \neq D'$ will decide correctly w.h.p. when $|S|, |S'| \approx 1/\mu_\star^2$
- When $D = D'$ will decide correctly w.h.p. when $|S|, |S'| \approx 1/\mu^2$

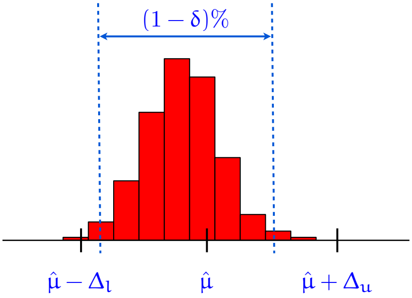
In the later we are always competing against the *worst case* $L_\infty(D, D') = \mu$

Enter the Bootstrap

- In the test I just described there is another *worst case* assumption — the confidence interval $\mu_\star \leqslant \hat{\mu} + \Delta_u$ must hold *for any D and D'*
- *But* it may be the case that for some D, certifying that $S, S' \sim D$ come from the same distribution is *easier*
- The *bootstrap* is widely used in statistics for computing *distribution dependent* confidence intervals (among many other things)

Basic Idea

- Suppose we have r different samples $S_{(1)}, \ldots, S_{(r)} \sim D$
- Compute distances $\hat{\mu}_i = \mathrm{L}_\infty(S_{(i)}, S'_{(i)})$
- Use them to compute a histogram of the distribution of $\hat{\mu}$



Enter the Bootstrap

- In the test I just described there is another *worst case* assumption — the confidence interval $\mu_\star \leqslant \hat{\mu} + \Delta_u$ must hold *for any D and D'*
- *But* it may be the case that for some D, certifying that $S, S' \sim D$ come from the same distribution is *easier*
- The *bootstrap* is widely used in statistics for computing *distribution dependent* confidence intervals (among many other things)

Basic Idea

- Suppose we have r different samples $S_{(1)}, \ldots, S_{(r)} \sim D$
- Compute distances $\hat{\mu}_i = L_\infty(S_{(i)}, S'_{(i)})$
- Use them to compute a histogram of the distribution of $\hat{\mu}$

Bootstrapped Confidence Intervals

- Given a sample S, obtain other samples $\tilde{S}_{(i)}$ by sampling from S uniformly with replacement
- Sort estimates increasingly $\tilde{\mu}_1 \leqslant \ldots \leqslant \tilde{\mu}_r$
- Say that $\mu_\star \leqslant \tilde{\mu}_{\lceil(1-\delta)r\rceil}$ with prob. $\geqslant 1-\delta$

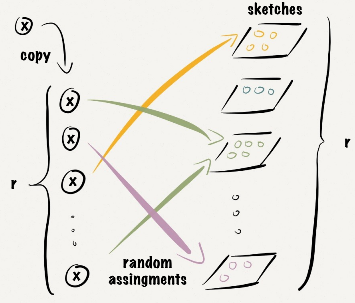
Bootstrapped Confidence Intervals in Data Streams

Question: Do you need to *store the full sample* to do bootstrap resampling?

Answer: No, if you can *test from sketched data*

The Bootstrap Sketch

- Keep r copies of the sketch you use for testing (e.g. **SpaceSaving**)
- For each item x_t in the stream, randomly insert r copies of x_t into the r sketches
- Comparing each pair $\tilde{S}_{(i)}, \tilde{S}'_{(j)}$ can obtain r^2 approximations $\tilde{\mu}_{i,j}$
- Choosing r involves a trade-off between accuracy and memory



In theory can prove bound (asymptotically) comparable to Vapnik–Chervonenkis

In practice assuming $\mu_\star \leqslant \tilde{\mu}_{\lceil(1-\delta)r^2\rceil}$ gives accurate and statisically efficient similarity test

Experimental Results for Learner

- Prototype written in C++ and Boost, run in this laptop
- Evaluated with Reber Grammar (typical Grammatical Inference benchmark)
  - $|\Sigma| = 5$, $n = 6$, $\mu = 0.2$, $L \approx 8$
- Compared VC and Bootstrap ($r = 10$) based tests

	Examples	Memory (MiB)	Time/item (ms)
Hoeffding	57617	6.1	0.05
Bootstrap	23844	53.7	1.2

Outline

What if n and μ are unknown (or change)?

Want to design strategy for *fast and accurate* parameter estimation

Parameter Search Algorithm

$n \leftarrow 2$, $\mu \leftarrow 1/8$;
while true do
  $H \leftarrow \texttt{Learner}(n, \mu)$;
  if $|H| < n$ **then** $\mu \leftarrow \mu/8$;
  else $n \leftarrow 2n$;
  if $n > (1/\mu)^{1/3}$ **then** $\mu \leftarrow \mu/8$;

Complexity Bounds

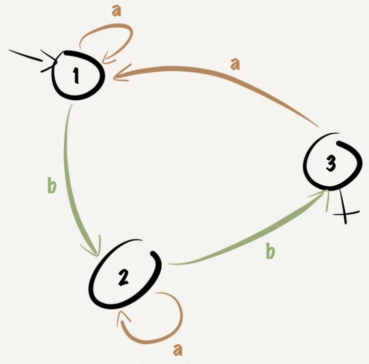
- Needs only $O(\log(n_\star/\mu_\star^{1/3}))$ calls to **Learner**
- In expectation will read $O(n_\star^6 |\Sigma|^2/\varepsilon\mu_\star^2)$ elements
- Memory usage grows like $O(t^{2/3})$

Note: can tweak parameters to trade-off convergence speed and memory usage

Adapting the Hypothesis to Changes

Adapter block — Once the structure is known...

- Estimating probabilities is easy
- Estimations can be adapted to changes (e.g. moving average)



Transition/Stop probabilities

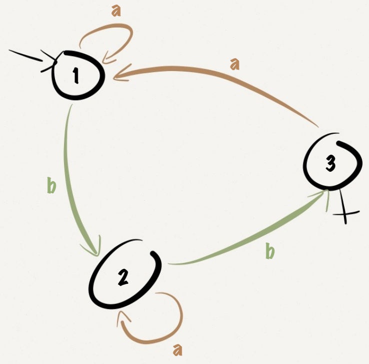
$S = \{abb, baab, bbaabb\}$

q	$p_q(a)$	$p_q(b)$	$p_q(\xi)$
1	2/6	4/6	0/6
2	2/6	4/6	0/6
3	1/4	0/4	3/4

But, sometimes the current structure is not good anymore

Detecting Structural Changes

Idea: "change" is *difficult to define* in general, focus on changes explained in terms of *structure*



- Given a *PDFA*, compute the expected number of times each state is visited when generating a string
- Given a *sample*, compute the average number of times strings hit any state
- If there is a *significant difference*, conclude the structure has changed

$$S = \{abb, baab, bbaabb\}$$

h_1	h_2	h_3
6/3	6/3	4/3

- Restart structure learning when a change is detected
- Adapting probabilities may be enough, but re-learning does no damage

Outline

Conclusion

Summary of Contributions

- Adaptation of state-merging paradigm to streaming data
- Fast convergence achieved by:
  - adaptive test scheduling
  - better similarity testing
  - efficient parameter search
- Use of sketching algorithms for implementing the bootstrap and reducing memory usage

Future Work

- Deploy real system and exploit parallelization oportunities
- Develop further similarity tests based on the bootstrap
- Adapt other GI algorithms to the data streams framework

Bootstrapping and Learning PDFA in Data Streams

Borja Balle, Jorge Castro, Ricard Gavaldà

LARCA. Laboratory for Relational Algorithmics, Complexity and Learning
UNIVERSITAT POLITÈCNICA DE CATALUNYA

International Colloquium on Grammatical Inference
University of Maryland, September 2012

This work is partially supported by the PASCAL2 Network