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

Learning PDFA from Data Streams

Testing Similarity in Data Streams with the Bootstrap

Adapting to Changes in the Target

Conclusion
Outline

Learning PDFA from Data Streams

Testing Similarity in Data Streams with the Bootstrap

Adapting to Changes in the Target

Conclusion
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

<table>
<thead>
<tr>
<th>q</th>
<th>$p_q(a)$</th>
<th>$p_q(b)$</th>
<th>$p_q(\xi)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.3</td>
<td>0.7</td>
<td>0.0</td>
</tr>
<tr>
<td>2</td>
<td>0.5</td>
<td>0.5</td>
<td>0.0</td>
</tr>
<tr>
<td>3</td>
<td>0.8</td>
<td>0.0</td>
<td>0.2</td>
</tr>
</tbody>
</table>

Parameters

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

$$\mu = \min_{q \neq q'} \max_{x \in \Sigma^*} |D_q(x) - D_{q'}(x)|$$
State Merge/Split Algorithm


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 $q_\lambda$;

foreach $\sigma \in \Sigma$ do
  add a candidate $q_\sigma$ to $H$;
schedule insignificance and similarity tests for $q_\sigma$;

foreach string $x_t$ in the stream do
  foreach decomposition $x_t = wz$, with $w, z \in \Sigma^*$ 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($q_w, \delta$);
  
  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 $L_\infty(D, D') \geq \mu$
- In the second case, if $|D(x) - D'(x)| \geq \mu$ then either $D(x) \geq \mu$ or $D'(x) \geq \mu$
- 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 [Mettwally 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

Learning PDFA from Data Streams

Testing Similarity in Data Streams with the Bootstrap

Adapting to Changes in the Target

Conclusion
Testing Similarity between Probability Distributions

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

Statistical Test Based on Empirical $L_\infty$ (the “default”)

- Let $\mu_* = L_\infty(D, D')$ and compute $\hat{\mu} = L_\infty(S, S')$
- Compute $\Delta_l, \Delta_u$ such that $\hat{\mu} - \Delta_l \leq \mu_* \leq \hat{\mu} + \Delta_u$ 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 than deciding similarity

- When $D \neq D'$ will decide correctly w.h.p. when $|S|, |S'| \approx 1/\mu_*^2$
- 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 \( \mu_* \leq \hat{\mu} + \Delta_u \) 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 \( \hat{\mu} \)
Enter the Bootstrap

- In the test I just described there is another worst case assumption — the confidence interval $\mu_* \leq \hat{\mu} + \Delta_u$ 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 $\tilde{\mu}_1 \leq \ldots \leq \tilde{\mu}_r$
- Say that $\mu_* \leq \tilde{\mu}_{[(1-\delta)r]}$ with prob. $\geq 1 - \delta$
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 \( \mu_* \leq \tilde{\mu}_{[(1-\delta)r^2]} \) gives accurate and statistically 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

<table>
<thead>
<tr>
<th></th>
<th>Examples</th>
<th>Memory (MiB)</th>
<th>Time/item (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hoeffding</td>
<td>57617</td>
<td>6.1</td>
<td>0.05</td>
</tr>
<tr>
<td>Bootstrap</td>
<td>23844</td>
<td>53.7</td>
<td>1.2</td>
</tr>
</tbody>
</table>
Outline

Learning PDFA from Data Streams

Testing Similarity in Data Streams with the Bootstrap

Adapting to Changes in the Target

Conclusion
What if $n$ and $\mu$ are unknown (or change)?

Want to design strategy for fast and accurate parameter estimation

Parameter Search Algorithm

\[
\begin{align*}
    n &\leftarrow 2, \mu \leftarrow 1/8; \\
    \text{while true do} & \\
    & H \leftarrow \text{Learner}(n, \mu); \\
    & \text{if } |H| < n \text{ then } \mu \leftarrow \mu/8; \\
    & \text{else } n \leftarrow 2n; \\
    & \text{if } n > (1/\mu)^{1/3} \text{ then } \mu \leftarrow \mu/8;
\end{align*}
\]

Complexity Bounds

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

\[ S = \{abb, baab, bbaabb\} \]

<table>
<thead>
<tr>
<th>( q )</th>
<th>( p_q(a) )</th>
<th>( p_q(b) )</th>
<th>( p_q(\xi) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2/6</td>
<td>4/6</td>
<td>0/6</td>
</tr>
<tr>
<td>2</td>
<td>2/6</td>
<td>4/6</td>
<td>0/6</td>
</tr>
<tr>
<td>3</td>
<td>1/4</td>
<td>0/4</td>
<td>3/4</td>
</tr>
</tbody>
</table>

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

\[ S = \{abb, baab, bbaabb\} \]

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>( h_1 )</td>
<td>6/3</td>
<td>6/3</td>
<td>4/3</td>
</tr>
</tbody>
</table>

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

Learning PDFA from Data Streams

Testing Similarity in Data Streams with the Bootstrap

Adapting to Changes in the Target

Conclusion
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 opportunities
- 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

International Colloquium on Grammatical Inference
University of Maryland, September 2012

This work is partially supported by the PASCAL2 Network