Adaptively Learning Probabilistic Deterministic Automata from Data Streams

Borja Balle, Jorge Castro, and Ricard Gavaldà

LARCA research group, Universitat Politècnica de Catalunya,
08034 Barcelona, Spain

{bballe,castro,gavalda}@lsi.upc.edu

Submitted: December 5th, 2012
Revised: June 30th, 2013

Abstract

Markovian models with hidden state are widely-used formalisms for modeling sequential phenomena. Learnability of these models has been well studied when the sample is given in batch mode, and algorithms with PAC-like learning guarantees exist for specific classes of models such as Probabilistic Deterministic Finite Automata (PDFA). Here we focus on PDFA and give an algorithm for inferring models in this class in the restrictive *data stream* scenario: Unlike existing methods, our algorithm works incrementally and in one pass, uses memory sublinear in the stream length, and processes input items in amortized constant time. We also present extensions of the algorithm that 1) reduce to a minimum the need for guessing parameters of the target distribution and 2) are able to adapt to changes in the input distribution, relearning new models when needed. We provide rigorous PAC-like bounds for all of the above. Our algorithm makes a key usage of stream sketching techniques for reducing memory and processing time, and is modular in that it can use different tests for state equivalence and for change detection in the stream.

1 Introduction

Data streams are a widely accepted computational model for algorithmic problems that have to deal with vast amounts of data in real-time and where feasible solutions must use very little time and memory per example. Over the last ten years, the model has gained popularity among the Data Mining community, both as a source of challenging algorithmic problems and as a framework into which several emerging applications can be cast (Aggarwal, 2007; Gama, 2010). From these efforts, a rich suite of tools for data stream mining has emerged, solving difficult problems related to application domains like network traffic analysis, social web mining, and industrial monitoring.

Most algorithms in the streaming model fall into one of the following two classes: a class containing primitive building blocks, like change detectors and

sketching algorithms for computing statistical moments and frequent items; and a class containing full-featured data mining algorithms, like frequent itemsets miners, decision tree learners, and clustering algorithms. A generally valid rule is that primitives from the former class can be combined for building algorithms in the latter class. Still, most of the work described above assumes that data in the stream has *tabular* structure, i.e., stream elements are described by (attribute,value) pairs. Less studied are the cases where elements have other combinatorial structures (sequences, trees, graphs, etc.).

Here we focus on the sequence, or string, case. The grammatical inference community has produced remarkable algorithms for learning various classes of probabilistic finite state machines from sets of strings, presumably sampled from some stochastic generating phenomenon. *State-merging* algorithms for learning Probabilistic Deterministic Finite Automata (from now on, PDFA) have been proposed in the literature. Some of them are based on heuristics, while others come with theoretical guarantees, either of convergence in the limit, or in the PAC sense (Carrasco and Oncina, 1999; Ron et al, 1998; Clark and Thollard, 2004; Palmer and Goldberg, 2007; Guttman et al, 2005; Castro and Gavaldà, 2008). However, all of them are batch-oriented and require the full sample to be stored in memory and most of them perform several passes over the sample. Quite independently, the field known as Process Mining also attempts to build models (such as state-transition graphs and Petri nets) from process logs; its motivation comes more often from business process modeling or software or hardware system analysis, and emphasis is in typically in understandable outcome and modeling concurrency. Our approach remains closer to the grammatical inference one.

With the advent of the Web and other massively streaming environments, learning and analysis have to deal with high-speed, continuously arriving datasets, where the target to be modeled possibly changes or drifts over time. While the data stream computational model is a natural framework for these applications, adapting existing methods from either grammatical inference or process mining is far from obvious.

In this paper we present a new state-merging algorithm for PAC learning PDFA from a stream of strings. It uses little memory, constant processing time per item, and is able to detect changes in the distribution generating the stream and adapt the learning process accordingly. We will describe how to use it to design a complete learning system, with the two-level idea described above (primitives for sketching and change detection at the lower level, full learning algorithm at a higher level). Regarding the state-merging component, we make two main contributions. The first is the design of an efficient and adaptative scheduling policy to perform similarity tests, so that sound decisions are made as soon as enough examples are available. This behavior is essentially different from the PAC algorithms in (Clark and Thollard, 2004; Guttman et al, 2005; Palmer and Goldberg, 2007) which work by asking for a sample of a certain size upfront which is then used for learning the target. Thus, these algorithms always work with the worst-case sample size (over all possible distributions), while our algorithm is able to adapt to the complexity of the target and learn easy targets using less examples than predicted by the worst case analysis. Our algorithm resembles that of Castro and Gavaldà (2008) in this particular aspect, though there is still a significant difference: Their algorithm is adaptative in the sense that it takes a fixed sample and tries to make the best of it. In

contrast, having access to an unbounded stream of examples, adaptiveness in our algorithm comes from its ability to make probably correct decisions as soon as possible.

The second contribution from a state-merging perspective is use of sketching methods from the stream algorithmics literature to find frequent prefixes in streams of strings, which yield a PAC learning algorithm for PDFA using memory $O(1/\mu)$, in contrast with the usual $O(1/\mu^2)$ required by batch methods – here μ denotes the *distinguishability* of the target PDFA, a quantity which measures the difficulty of distinguishing the different states in a PDFA. In fact, the exact bound values we prove mostly follow from well-known bounds from statistics, and that have already been applied to state-merging methods. A main contribution is showing that these still hold, and can in fact be made tighter, when using sketches rather than full datasets.

The structure of the paper is as follows. Section 2 begins by introducing the notation we use throughout the paper and our main definitions. Then, in Section 2.5 we review previous work, and in Section 2.6 we explain our contributions in more detail. Section 3 describes the complete stream learning system arising from our methods and an illustrative scenario. Section 4 describes the basic state-merging algorithm for streams, with its analysis, and Section 5 describes two variants of the Space-Saving sketch central to having low memory use. In Section 6 we present the strategy for automatically finding correct parameters for the basic algorithm (number of states and distinguishability). Section 7 extends the algorithm to detect and handle changes in the stream. Finally, Section 8 presents some conclusions and outlines future work.

2 Our Results and Related Work

The following sections give necessary notation and formal definitions of PDFA, PAC learning, and the data stream computation model.

2.1 Notation

As customary, we use the notation $\widetilde{O}(f)$ as a variant of $O(f)$ that ignores poly-logarithmic factors, and the set of functions g such that $O(f) = O(g)$ is denoted with $\Theta(f)$. Unless otherwise stated we assume the unit-cost computation model, where (barring model abuses) e.g. an integer count can be stored in unit memory and operations on it take unit time. If necessary statements can be translated to the logarithmic model, where e.g. a counter with value t uses memory $O(\log t)$, or this factor is hidden within the $\widetilde{O}(\cdot)$ notation.

We denote by $\Sigma^\star$ the set of all strings over a finite alphabet Σ. Elements of $\Sigma^\star$ will be called strings or words. Given $x, y \in \Sigma^\star$ we will write xy to denote the concatenation of both strings. Concatenation is an associative operation. We use λ to denote the empty string, which satisfies $\lambda x = x\lambda = x$ for all $x \in \Sigma^\star$. The length of a string $x \in \Sigma^\star$ is denoted by $|x|$. The empty string is the only string with $|\lambda| = 0$. A *prefix* of a string $x \in \Sigma^\star$ is a string u such that there exists another string v such that $x = uv$; then string v is a *suffix* of x. Hence e.g. $u\Sigma^\star$ is the set of all strings having u as a prefix.

2.2 Learning Distributions in the PAC Framework

Several measures of divergence between probability distributions are considered. Let Σ be a finite alphabet and let D_1 and D_2 be distributions over $\Sigma^\star$. The total variation distance is $\mathrm{L}_1(D_1, D_2) = \sum_{x\in\Sigma^\star} |D_1(x) - D_2(x)|$. The supremum distance is $\mathrm{L}_\infty(D_1, D_2) = \max_{x\in\Sigma^\star} |D_1(x) - D_2(x)|$. Another distance between distributions over strings useful in the learning setting is the supremum over prefixes distance, or prefix-L_∞ distance: $\mathrm{L}^{\mathrm{p}}_\infty(D_1, D_2) = \max_{x\in\Sigma^\star} |D_1(x\Sigma^\star) - D_2(x\Sigma^\star)|$, where $D(x\Sigma^\star)$ denotes the probability under D of having x as a prefix.

It is obvious from this definition that $\mathrm{L}^{\mathrm{p}}_\infty$ is non-negative, symmetric, satisfies the triangle inequality, and is 0 when its two arguments are the same distribution; it is therefore a distance. There are examples of distributions whose $\mathrm{L}^{\mathrm{p}}_\infty$ is much larger than L_∞. On the other hand, Proposition A.7 in Appendix A.4 shows that, up to a factor that depends on the alphabet size, $\mathrm{L}^{\mathrm{p}}_\infty$ is always an upper bound for L_∞.

Now we introduce the PAC model for learning distributions. Let $\mathcal{D}$ be a class of distributions over some fixed set X. Assume $\mathcal{D}$ is equipped with some measure of *complexity* assigning a positive number $|D|$ to any $D \in \mathcal{D}$. We say that an algorithm A *PAC learns* a class of distributions $\mathcal{D}$ using $S(\cdot)$ examples and time $T(\cdot)$ if, for all $0 < \varepsilon, \delta < 1$ and $D \in \mathcal{D}$, with probability at least $1 - \delta$, the algorithm reads $S(1/\varepsilon, 1/\delta, |D|)$ examples drawn i.i.d. from D and after $T(1/\varepsilon, 1/\delta, |D|)$ steps outputs a hypothesis $\hat{D}$ such that $\mathrm{L}_1(D, \hat{D}) \leq \varepsilon$. The probability is over the sample used by A and any internal randomization. As usual, PAC learners are considered efficient if the functions $S(\cdot)$ and $T(\cdot)$ are polynomial in all of their parameters.

2.3 PDFA and State-Merging Algorithms

A Probabilistic Deterministic Finite Automaton (PDFA for short) T is a tuple $\langle Q, \Sigma, \tau, \gamma, \xi, q_0\rangle$ where Q is a finite set of states, Σ is an arbitrary finite alphabet, $\tau : Q \times \Sigma \longrightarrow Q$ is the deterministic transition function, $\gamma : Q \times (\Sigma \cup \{\xi\}) \longrightarrow [0, 1]$ defines the probability of emitting each symbol from each state – where we must have $\gamma(q, \sigma) = 0$ when $\sigma \in \Sigma$ and $\tau(q, \sigma)$ is not defined –, ξ is a special symbol not in Σ reserved to mark the end of a string, and $q_0 \in Q$ is the initial state.

It is required that $\sum_{\sigma\in\Sigma\cup\{\xi\}} \gamma(q, \sigma) = 1$ for every state q. Transition function τ is extended to $Q\times\Sigma^\star$ in the usual way. Also, the probability of generating a given string $x\xi$ from state q can be calculated recursively as follows: if x is the empty word λ the probability is $\gamma(q, \xi)$, otherwise x is a string $\sigma_0\sigma_1 \ldots \sigma_k$ with $k \geq 0$ and $\gamma(q, \sigma_0\sigma_1 \ldots \sigma_k\xi) = \gamma(q, \sigma_0)\gamma(\tau(q, \sigma_0), \sigma_1 \ldots \sigma_k\xi)$. It is well known that if every state in a PDFA has a non-zero probability path to a state with positive stopping probability, then every state in that PDFA defines a probability distribution; we assume this is true for all PDFA considered in this paper. For each state q a probability distribution D_q on $\Sigma^\star$ can be defined as follows: for each x, probability $D_q(x)$ is $\gamma(q, x\xi)$. The probability of generating a prefix x from a state q is $\gamma(q, x) = \sum_y D_q(xy) = D_q(x\Sigma^\star)$. The distribution defined by T is the one corresponding to its initial state, D_{q_0}. Very often we will identify a PDFA and the distribution it defines. The following parameter is used to measure the complexity of learning a particular PDFA.

Definition 2.1. *For a distance* dist *among distributions, we say distributions* D_1 *and* D_2 *are* μ-distinguishable *w.r.t.* dist *if* $\mu \leq \mathsf{dist}(D_1, D_2)$*. A PDFA* T *is* μ*-distinguishable when for each pair of states* q_1 *and* q_2 *their corresponding distributions* D_{q_1} *and* D_{q_2} *are* μ*-distinguishable. The* distinguishability *of a PDFA (w.r.t.* dist*) is defined as the supremum over all* μ *for which the PDFA is* μ*-distinguishable. Unless otherwise noted, we will use* $\mathsf{dist} = \mathrm{L}_\infty^{\mathrm{p}}$*, and occasionally use instead* $\mathsf{dist} = \mathrm{L}_\infty$*.*

State-merging algorithms form an important class of strategies of choice for the problem of inferring a regular language from samples. Basically, they try to discover the target automaton graph by successively applying tests in order to discover new states and merge them to previously existing ones according to some similarity criteria. In addition to empirical evidence showing a good performance, state-merging algorithms also have theoretical guarantees of learning in the limit the class of regular languages (see de la Higuera (2010)).

Clark and Thollard (2004) adapted the state-merging strategy to the setting of learning distributions generated by PDFA and showed PAC-learning results parametrized by the distinguishability of the target distribution. The distinguishability parameter can sometimes be exponentially small in the number of states in a PDFA. However, there exists reasonable evidence suggesting that polynomiality in the number of states alone may not be achievable (Kearns et al (1994); Terwijn (2002)); in particular, the problem is at least as hard as the *noisy parity learning* problem for which, despite remarkable effort, only exponential-time algorithms are known.

2.4 The Data Stream Framework

The *data stream* computation model has established itself in the last fifteen years for the design and analysis of algorithms on high-speed sequential data (Aggarwal, 2007). It is characterized by the following assumptions:

1. The input is a potentially infinite sequence of items $x_1, x_2, \ldots, x_t, \ldots$ from some (large) universe X.

2. Item x_t is available only at the tth time step and the algorithm has only that chance to process it, say by incorporating it to some summary or sketch; that is, only one pass over the data is allowed.

3. Items arrive at high-speed, so the processing time per item must be very low – ideally constant time, but most likely logarithmic in t and $|X|$.

4. The amount of memory used by algorithms (the size of the sketches alluded above) must be sublinear in the data seen so far at every moment; ideally, at time t memory must be polylogarithmic in t – for many problems, this is impossible and memory of the form t^c for constant $c < 1$ may be required. Logarithmic dependence on $|X|$ is also desirable.

5. Anytime answers are required, but approximate, probabilistic ones are often acceptable.

A large fraction of the data stream literature discusses algorithms working under worst-case assumptions on the input stream, e.g. compute the required

(approximate) answer at all times t for every possible values of $x_1, \ldots, x_t$ (Lin and Zhang, 2008; Muthukrishnan, 2005). For example, several sketches exist to compute an ε-approximation to the number of distinct items in memory $O(\log(t|X|)/\varepsilon)$ seen from the stream start until time t. In machine learning and data mining, this is often not the problem of interest: one is interested in modeling the current "state of the world" at all times, so the current items matter much more than those from the far past (Bifet, 2010; Gama, 2010). An approach is to assume that each item x_t is generated by some underlying distribution D_t over X, that varies over time, and the task is to track the distributions D_t (or its relevant information) from the observed items. Of course, this is only possible if these distributions do not change too wildly, e.g. if they remain unchanged for fairly long periods of time ("distribution shifts", "abrupt change"), or if they change only very slightly from t to $t+1$ ("distribution drift"). A common simplifying assumption (which, though questionable, we adopt here) is that successive items are generated independently, i.e. that x_t depends only on D_t and not on the outcomes x_{t-1}, x_{t-2}, etc.

In our case, the universe X will be the infinite set $\Sigma^\star$ of all string over a finite alphabet Σ. Intuitively, the role of $\log(|X|)$ will be replaced by a quantity such as the expected length of strings under the current distribution.

2.5 Related Work

The *distributional learning* problem is: Given a sample of sequences from some unknown distribution D, build a model that generates a distribution on sequences close to D. Here we concentrate on models which can be generically viewed as finite-state probabilistic automata, with transitions generating symbols and labeled by probabilities. When the transition diagram of the automaton is known or fixed, the problem amounts to assigning transition probabilities and is fairly straightforward. The problem is much harder, both in theory and in practice, when the transition diagram must be inferred from the sample as well. Several heuristics have been proposed, mostly by the Grammatical Inference community for these and related models such as Hidden Markov models (see (Dupont et al, 2005; Vidal et al, 2005a,b) and references therein).

On the theoretical side, building upon previous work by Ron et al (1998), Clark and Thollard (2004) gave an algorithm that provably learns the subclass of so-called Probabilistic Deterministic Finite Automata (PDFA). These are probabilistic automata whose transition diagram is deterministic (i.e. for each state q and letter σ there is at most one transition out of q with label σ and positive probability). These algorithms are based on the state-merging paradigm, and can be showed to learn in a PAC-like sense (see Section 2.2). The original algorithm by Clark and Thollard (2004) was successively refined in several works (Castro and Gavaldà, 2008; Palmer and Goldberg, 2007; Guttman et al, 2005; Balle et al, 2012b)[1]. Up to now, the best known algorithms can be shown to learn the correct structure when they receive a sample of size $\widetilde{O}(n^3/\varepsilon^2\mu^2)$,

[1]The algorithm in Clark and Thollard (2004) and several variants learn a hypothesis ε-close to the target in the Kullback-Leibler (KL) divergence. Unless noted otherwise, here we will consider learning PDFA with respect to the less demanding L_1 distance. In fact, Clark and Thollard (2004) learn under the KL divergence by first learning w.r.t. the L_1 distance, then *smoothing* the transition probabilities of the learned PDFA. This is also possible for our algorithm.

where n is the number of states on the target, μ is a measure of how *similar* the states in the target are, and ε is the usual accuracy parameter in the PAC setting. These algorithms work with memory on this same order and require a processing time of order $\widetilde{O}(n^4/\varepsilon^2\mu^2)$ to identify the structure (and additional examples to estimate the transition probabilities). Similar results with weaker guarantees and using very different methods have been given for other classes of models by Hsu et al (2009).

Unfortunately, known algorithms for learning PDFA (Carrasco and Oncina, 1999; Clark and Thollard, 2004; Hsu et al, 2009) are extremely far from the data stream paradigm. They all are batch oriented: they perform several passes over the sample; they need all the data upfront, rather than working incrementally; they store the whole sample in memory, using linear memory or more; and they cannot deal with data sources that evolve over time. Recently, algorithms for online induction of automata have been presented in Schmidt and Kramer (2012); Schmidt et al (2012); while online (i.e., non-batch) and able to handle drift, they seem to use memory linear in the sample size and do not come with PAC-like guarantees.

2.6 Our Contributions

Our first contribution is a new state-merging algorithm for learning PDFA. In sharp contrast with previous state-merging algorithms, our algorithm works in the demanding streaming setting: it processes examples in amortized constant time, uses memory that grows sublinearly with the length of the stream, and learns by making only one pass over the data. The algorithm uses an adaptative strategy to perform state similarity tests as soon as enough data is available, thus accelerating its convergence with respect to traditional approaches based on worst-case assumptions. Furthermore, it incorporates a variation of the well-known Space-Saving sketch (Metwally et al, 2005) for retrieving frequent prefixes in a stream of strings. Our state-merging algorithm uses memory of the order $\widetilde{O}(n|\Sigma|/\mu)$, where L is the expected length of strings in the stream. Note in particular the improvement in the exponent of $1/\mu$ over the batch setting, because in many cases this quantity can grow exponentially in the number of states, thus dominating the bounds above. Furthermore, we show that the *expected* number of examples read until the structure is identified is $\widetilde{O}(n^2|\Sigma|^2/\varepsilon\mu^2)$ (note that the bounds stated in the previous section are worst-case, with probability $1-\delta$).

Building on top of this algorithm, we present our second contribution: a search strategy for learning the target's parameters coupled with a change detector. This removes the highly impractical need to guess target parameters and revising them as the target changes. The parameter search strategy leverages memory usage in the state-merging algorithm by enforcing an invariant that depends on the two relevant parameters: number of states n and state distinguishability μ.

A delicate analysis of this invariant based on the properties of our state-merging algorithm yields sample and memory bounds for the parameter-free algorithm. These bounds reveal a natural trade-off between memory usage and number of samples needed to find the correct parameters in our search strategy, which can be adjusted by the user according to their time vs. memory priorities. For a particular choice of this parameter, we can show that this algorithm uses memory that grows like $\widetilde{O}(t^{3/4})$ with the number t of processed

examples, whether or not the target is generated by PDFA. If the target is one fixed PDFA, the expected number of examples the algorithm reads before converging is $\widetilde{O}(n^4|\Sigma|^2/\varepsilon\mu^2)$. Note that the memory usage grows sublinearly in the number of examples processed so far, while a factor n^2 is lost in expected convergence time with respect to the case where the true parameters of the target are known in advance.

In summary, we present what is, to our knowledge, the first algorithm for learning probabilistic automata with hidden states and meeting the restrictions imposed by the data streaming setting. Our algorithm has rigorous PAC guarantees, is capable of adapting to changes in the stream, uses at its core a new efficient and adaptative state-merging algorithm based on state-of-the-art sketching components, and is relatively simple to code and potentially scalable. While a strict implementation of the fully PAC algorithm is probably not practical, several relaxations are possible which may make it feasible in some applications.

3 A System for Continuous Learning

Before getting into the technicalities of our algorithm for learning PDFA from data streams, we begin with a general description of a complete learning system capable of adapting to changes and make predictions about future observations. In the following sections we will describe the components involved in this system in detail and prove rigorous time and memory bounds.

The goal of the system is to keep at all times a hypothesis – a PDFA in this case – that models as closely as possible the distribution of the strings currently being observed in the stream. For convenience, the hypothesis PDFA will be represented as two separate parts: the DFA containing the inferred transition structure, and tables of estimations for transition and stopping probabilities. Using this representation, the system is able to decouple the state-merging process that learns the transition structure of the hypothesis from the estimation procedure that computes transition and stopping probabilities. This decomposition is also useful in terms of change detection. Change in transition or stopping probabilities can be easily tracked with simple sliding window or decaying weights techniques. On the other hand, changes in the structure of a PDFA are much harder to detect, and modifying an existing structure to adapt to this sort of changes is a very challenging problem. Therefore, our system contains a block that continually estimates transition and stopping probabilities, and another block that detects changes in the underlying structure and triggers a procedure that learns a new transition structure from scratch. A final block that uses the current hypothesis to make predictions about the observations in the stream (more in general, decide actions on the basis of the current model) can also be integrated into the system. Since this block will depend on the particular application, it will not be discussed further here. The structure we just described is depicted in Figure 1.

The information flow in the system works as follows. The structure learning block – containing a state-merging algorithm and a parameter search block in charge of finding the correct n and μ for the target – is started and the system waits until it produces a first hypothesis DFA. This DFA is fed to the probability estimation block and the change detector. From now on, these two blocks run in parallel, as well as the learning block, which keeps learning new

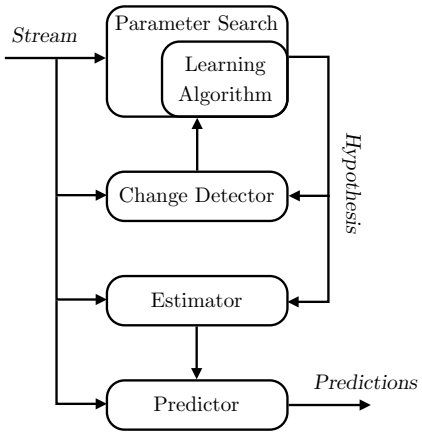


Figure 1: System for Continuous Learning from Data Streams

structures with more refined parameters. If at some point a change in the target structure is found, a signal is emitted and the learning block restarts the learning process.[2] In parallel, the estimator block keeps updating transition and stopping probabilities all the time. It may be the case that this latter parameter adaptation procedure is enough to track the current distribution. However, if the structure learning procedure produces a new DFA, transition probabilities are estimated for this new structure, which then takes the place of the current hypothesis. Thus, the system will recover much faster from a change that only affects transition probabilities, than from a change in the structure of the target.

3.1 Example: Application to Click Prediction in On-line Stores

Let us present a scenario that illustrates the functioning of such a learning system. Although not intended to guide our research in the rest of the paper, it may help understanding the challenges it faces and the interactions among its parts.

Consider an on-line store where customers navigate through a collection of linked pages to browse the store catalog and product pages, check current offers, update their profiles, edit their cart contents, proceed to checkout and payment, ask for invoices, etc. Web administrators would like to model customer behavior for a number of reasons, including: classifying customers according to their intention to purchase, predicting the next visited page for prefetching or caching, detecting when a customer is lost or not finding what s/he wants, reorganizing the pages for higher usability, etc. A natural and useful customer model for such tasks is one that models the distribution of "sessions", or user visits, the sequence of pages visited by the user – hence the set of pages forms a finite alphabet; we make this simplification for the rest of the discussion,

[2]Here one has a choice of keeping the current parameters in or restarting them to some initial values; previous knowledge on the changes the algorithm will be facing can help to make an informed decision in this point.

although page requests may have parameters, the user may click on a specific object of a page (e.g., a button), etc.

An easy way of defining a generative model in this case is to take the set of pages as a set of states, and connect pages i and j by a transition with the empirical probability that user moves to page j when in page i. In this case, the state is fully observable, and building such an automaton from a log of recorded sessions is trivial. This is in essence the model known as Customer Behavior Model Graphs (Menascé et al, 1999), often used in customer analytics and website optimization. A richer model should try to capture the non-observable user state, e.g., his/her mood, thought, or intention, beyond the page that s/he is currently looking at. In this case, a probabilistic state machine with hidden state is more adequate, if one accepts the hypothesis that a moderate number of states may suffice to explain a useful part of the users' behaviors. In this case, the structure of such a hidden machine is unknown as well and inferring it from a session log is the learning problem we have been discussing.

More importantly, user behavior may clearly evolve over time. Some of the changes may be attributed to the web itself: the administrator may change the connections among pages (e.g. to make some more prominent or accessible), or products may be added or removed from the catalog; these will tend to create sudden changes in the behavior, affecting often the structure of the inferred machine. Others may be due to the users' changing their behaviors, such as some products in this site's catalog becoming more or less fashionable or the competition changing their catalog or prices; these changes may preserve the structure or the machine or not, and be gradual or sudden. If the system is to keep an accurate and useful model of user behavior, it must be ready to detect all changes from the continuous clickstream arriving at the website and react or relearn appropriately.

4 State-merging in Data Streams

In this section we present an algorithm for learning distributions over strings from a data stream. The algorithm learns a PDFA by adaptively performing statistical tests in order to discover new states in the automata and merge similar states. We focus on the state-merging aspect of the algorithm, which is in charge of obtaining the transition structure between states. This stage requires the use of sketches to store samples of the distribution generated from each state in the PDFA, and a testing sub-routine to determine whether two states are equal or distinct based on the information contained in these sketches. We give two specific sketches and a test in Section 5 and Appendix A.3, respectively. Here we will just assume that components respecting certain bounds are used and state the properties of the algorithm under those assumptions. After finding the transition structure between states, a second stage in which transition and stopping probabilities are estimated takes place. For deterministic transition graphs the implementation of this stage is routine and will not be discussed here.

The algorithm will read successive strings over Σ from a data stream and, after some time, output a PDFA. Assuming all the strings are drawn according to a fixed distribution generated from a PDFA, we will show that then the output will be accurate with high probability.

We begin with an informal description of the algorithm, which is complemented by the pseudocode in Algorithm 1. The algorithm follows a structure similar to other state-merging algorithms, though here tests to determine similarity between states are performed adaptively as examples arrive.

Our algorithm requires some parameters as input: the usual accuracy ε and confidence δ, a finite alphabet Σ, a number of states n and a distinguishability μ. Some quantities defined in terms of these parameters that are used in the algorithm are given in Table 1. Specifically, quantities α_0 and α (respectively, β_0 and β) define milestones for similarity tests (insignificance tests, see below), θ is used to fix insignificance thresholds and $1-\delta_i$ values define confidence thresholds for the tests.

The algorithm, called `StreamPDFALearner`, reads data from a stream of strings over Σ. At all times it keeps a hypothesis represented by a DFA. States in the DFA are divided into three kinds: *safe*, *candidate*, and *insignificant* states, with a distinguished *initial* safe state denoted by q_λ. Candidate and insignificant states have no out-going transitions. To each string $w \in \Sigma^\star$ we may be able to associate a state by starting at q_λ and successively traversing the transitions labeled by the symbols of w in order. If all transitions are defined, the last state reached is denoted by q_w, otherwise q_w is undefined – note that it is possible that different strings w and w' represent the same state $q = q_w = q_{w'}$.

For each state q in its current DFA, `StreamPDFALearner` keeps a multiset S_q of strings. These multisets grow with the number of strings processed by the algorithm and are used to keep statistical information about a distribution D_q. In fact, since the algorithm only needs information from frequent prefixes in the multiset, it does not need to keep the full multiset in memory. Instead, it uses sketches to keep the relevant information for each state. We use $\hat{S}_q$ to denote the information contained in these sketches associated with state q, and $|\hat{S}_q|$ to denote the number of strings inserted into the sketch associated with state q.

Execution starts from a DFA consisting of a single safe state q_λ and several candidate states q_σ, one for each $\sigma \in \Sigma$. All states start with an empty sketch. Each element x_t in the stream is then processed in turn: for each prefix w of $x_t = wz$ that leads to a state q_w in the DFA, the corresponding suffix z is added to that state's sketch $\hat{S}_{q_w}$. During this process, similarity and insignificance tests are performed on candidate states following a certain schedule; the former are triggered by the sketch's size reaching a certain threshold, while the latter occur at fixed intervals after the state's creation. In particular, t_q^0 denotes the time state q was created, t_q^s is a threshold on the size $|\hat{S}_q|$ that will trigger the next round of similarity tests for q, and t_q^u is the time the next insignificance test will occur. Parameter i_q keeps track of the number of similarity tests performed for state q; this is used to adjust the confidence parameter in those tests.

Insignificance tests are used to check whether the probability that a string traverses an arc reaching a particular candidate state is below a certain threshold; it is known that these transitions can be safely ignored when learning a PDFA in the PAC setting (Clark and Thollard, 2004; Palmer and Goldberg, 2007). Similarity tests use statistical information provided by the candidate's sketch to determine whether it equals some already existing safe state or it is different from all safe states in the DFA. These tests can return three values: `equal`, `distinct`, and `unknown`. These answers are used to decide what to do with the candidate currently being examined.

A candidate state will exist until it is promoted to safe, merged to another safe state, or declared insignificant. When a candidate is merged to a safe state, the sketches associated with that candidate are discarded. The algorithm will end whenever there are no candidates left, or when the number of safe states surpasses the limit n given by the user.

An example execution of algorithm `StreamPDFALearner` is displayed in Figure 2. In this figure we see the evolution of the hypothesis graph, and the effect the operations *promote*, *merge*, and *declare insignificant* have on the hypothesis.

4.1 Analysis

Now we proceed to analyze the `StreamPDFALearner` algorithm. We will consider memory and computing time used by the algorithm, as well as accuracy of the hypothesis produced in the case when the stream is generated by a PDFA. Our analysis will be independent of the particular sketching methodology and similarity test used in the algorithm. In this respect, we will only require that the particular components used in `StreamPDFALearner` to that effect satisfy the following assumptions with respect to bounds M_{sketch}, T_{sketch}, T_{test}, N_{unknown}, and N_{equal} which are themselves functions of the problem parameters.

Assumption 1. Algorithms `Sketch` and `Test` algorithm used in `StreamPDFALearner` satisfy the following:

1. Each instance of `Sketch` uses memory at most M_{sketch}
2. A `Sketch.insert`(x) operation takes time T_{sketch}
3. Any call `Test`$(\hat{S}, \hat{S}', \delta)$ takes time at most T_{test}
4. There exists a N_{unknown} such that if $|\hat{S}|, |\hat{S}'| \geq N_{\text{unknown}}$, then a call `Test`$(\hat{S}, \hat{S}', \delta, \mu)$ will never return `unknown`
5. There exists a N_{equal} such that if a call `Test`$(\hat{S}, \hat{S}', \delta, \mu)$ returns `equal`, then necessarily $|\hat{S}| \geq N_{\text{equal}}$ or $|\hat{S}'| \geq N_{\text{equal}}$
6. When a call `Test`$(\hat{S}, \hat{S}', \delta)$ returns either `equal` or `distinct`, then the answer is correct with probability at least $1 - \delta$.

Our first result is about memory and number of examples used by the algorithm. We note that the result holds for any stream of strings for which `Sketch` and `Test` satisfy Assumptions 1, not only those generated by a PDFA.

Theorem 4.1. *The following hold for any call to* `StreamPDFALearner`$(n, \Sigma, \varepsilon, \delta, \mu)$*:*

1. *The algorithm uses memory* $O(n|\Sigma|M_{\text{sketch}})$
2. *The expected number of elements read from the stream is* $O(n^2|\Sigma|^2 N_{\text{unknown}}/\varepsilon)$
3. *Each item* x_t *in the stream is processed in* $O(|x_t|T_{\text{sketch}})$ *amortized time*
4. *If a merge occured, then at least* N_{equal} *elements were read from the stream*

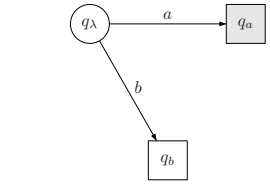
Algorithm 1: `StreamPDFALearner` procedure

Input: Parameters $n, \Sigma, \varepsilon, \delta, \mu$, `Sketch`, `Test`
Data: A stream of strings $x_1, x_2, \ldots \in \Sigma^\star$
Output: A hypothesis H

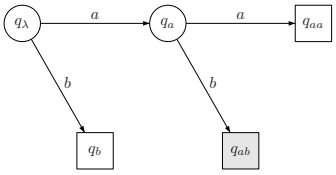
initialize H with safe q_λ and let $\hat{S}_{q_\lambda}$ be a new, empty `Sketch`;
foreach $\sigma \in \Sigma$ **do**
    add a candidate q_σ to H and let $\hat{S}_{q_\sigma}$ be a new, empty `Sketch`;
    $t^0_{q_\sigma} \leftarrow 0$, $t^s_{q_\sigma} \leftarrow \alpha_0$, $t^u_{q_\sigma} \leftarrow \beta_0$, $i_{q_\sigma} \leftarrow 1$;
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**
            $\hat{S}_{q_w}$.`insert`(z);
            if q_w *is a candidate* **and** $|\hat{S}_{q_w}| \geq t^s_{q_w}$ **then**
                foreach *safe* $q_{w'}$ *not marked as distinct from* q_w **do**
                    Call `Test`$(\hat{S}_{q_w}, \hat{S}_{q'_w}, \delta_{i_{q_w}}, \mu)$;
                    if `Test` *returned* `equal` **then** merge q_w to $q_{w'}$;
                    else if `Test` *returned* `distinct` **then** mark $q_{w'}$ as distinct from q_w;
                    else $t^s_{q_w} \leftarrow \alpha \cdot t^s_{q_w}$, $i_{q_w} \leftarrow i_{q_w} + 1$;
                if q_w *is marked distinct from all safes* **then**
                    promote q_w to safe;
                    foreach $\sigma \in \Sigma$ **do**
                        add a candidate $q_{w\sigma}$ to H and let $\hat{S}_{q_{w\sigma}}$ be a new, empty `Sketch`;
                        $t^0_{q_{w\sigma}} \leftarrow t$, $t^s_{q_{w\sigma}} \leftarrow \alpha_0$, $t^u_{q_{w\sigma}} \leftarrow t + \beta_0$, $i_{q_{w\sigma}} \leftarrow 1$;
    foreach *candidate* q **do**
        if $t \geq t^u_q$ **then**
            if $|\hat{S}_q| < \theta \cdot (t - t^0_q)$ **then** declare q insignificant;
            else $t^u_q \leftarrow t^u_q + \beta$;
    if H *has more than* n *safes* **or** *there are no candidates left* **then**
    return H;

$\alpha_0 = 128$	Milestones for significance testing
$\alpha = 2$	
$\beta_0 = (64n\|\Sigma\|/\varepsilon)\ln(2/\delta')$	Milestones for similarity testing
$\beta = (64n\|\Sigma\|/\varepsilon)\ln 2$	
$\theta = (3\varepsilon)/(8n\|\Sigma\|)$	Insignificance threshold
$\delta' = \delta/(2\|\Sigma\|n(n+2))$	Confidence parameters
$\delta_i = 6\delta'/\pi^2 i^2$	

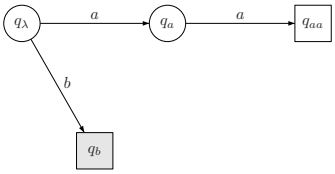
Table 1: Definitions used in Algorithm 1



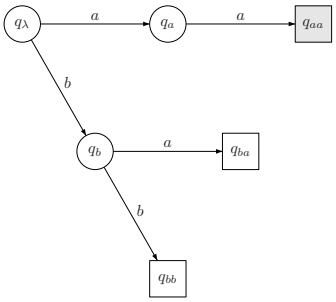
(a) q_a promoted to safe



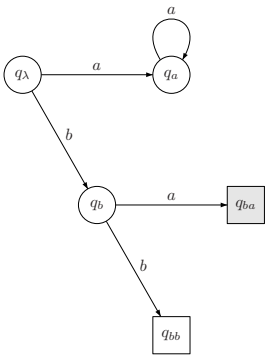
(b) q_{ab} declared insignificant



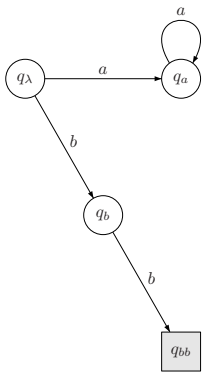
(c) q_b promoted to safe



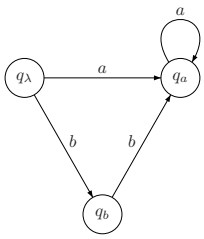
(d) q_{aa} merged to q_a



(e) q_{ba} declared insignificant



(f) q_{bb} merged to q_a



(g) Final hypothesis

Figure 2: Example run of `StreamPDFALearner` with $\Sigma = \{a, b\}$. Safe states are represented by circle nodes. Candidate states are represented with square nodes. States declared insignificant are not represented. Shaded nodes mark the candidate node on which some operation is performed at each step. Observe that after step (e) we have $q_a = q_{a^k}$ for every $k \geq 1$.

Proof. The memory bound follows from the fact that at any time there will be at most n safe states in the DFA, each with at most $|\Sigma|$ candidates attached, yielding a total of $n(|\Sigma|+1)$ states, each with an associated sketch using memory M_{sketch}. By assumption, a candidate state will be either promoted or merged after collecting N_{unknown} examples, provided that every safe state in the DFA has also collected N_{unknown} examples. Since this only matters for states with probability at least $\varepsilon/4n|\Sigma|$ because the rest of the states will be marked as insignificant (see Lemma 4.4), in expectation the algorithm will terminate after reading $O(n^2|\Sigma|^2 N_{\text{unknown}}/\varepsilon)$ examples from the stream. The time for processing each string depends only on its length: suffixes of string x_t will be inserted into at most $|x_t|+1$ states, at a cost T_{sketch} per insertion, yielding a total processing time of $O(|x_t|T_{\text{sketch}})$. It remains, though, to amortize the time used by the tests among all the examples processed. Any call to `Test` will take time at most T_{test}. Furthermore, for any candidate state, time between successive tests grows exponentially; that is, if t strings have been added to some $\hat{S}_q$, at most $O(\log t)$ tests on $\hat{S}_q$ have taken place due to the scheduling used. Thus, taking into account that each possible candidate may need to be compared to every safe state during each testing round, we obtain an expected amortized processing time per string of order $O(|x_t|T_{\text{sketch}} + n^2|\Sigma|T_{\text{test}}\log(t)/t)$. Finally, note that for a merge to occur necessarily some call to `Test` must return `equal`, which means that at least N_{equal} have been read from the stream.

We remark here that Item 3 above is a direct consequence of the scheduling policy used by `StreamPDFALearner` in order to perform similarity tests adaptatively. The relevant point is that the ratio between executed tests and processed examples is $O(\log(t)/t)$. In fact, by performing tests more often while keeping the tests/examples ratio tending to 0 as t grows, one could obtain an algorithm that converges slightly faster, but has a larger (though still constant with t) amortized processing time per item.

Our next theorem is a PAC-learning result. It says that if the stream is generated by a PDFA then the resulting hypothesis will have small error with high probability when transition probabilities are estimated with enough accuracy. Procedures to perform this estimation have been analyzed in detail in the literature. Furthermore, the adaptation to the streaming setting is straightforward. We use an analysis from (Palmer, 2008) in order to prove our theorem.

Theorem 4.2. *Suppose a stream generated from a PDFA D with n states and distinguishability μ is given to* `StreamPDFALearner`$(n', \Sigma, \varepsilon, \delta, \mu')$ *with $n' \geq n$ and $\mu' \leq \mu$. Let H denote the DFA returned by* `StreamPDFALearner` *and $\hat{D}_H$ a PDFA obtained from H by estimating its transition probabilities using $\widetilde{O}(n^4|\Sigma|^4/\varepsilon^3)$ examples. Then with probability at least $1-\delta$ we have $\mathrm{L}_1(D, \hat{D}_H) \leq \varepsilon$.*

The proof of Theorem 4.2 is similar in spirit to other in (Clark and Thollard, 2004; Palmer and Goldberg, 2007; Castro and Gavaldà, 2008; Balle et al, 2012b). Therefore, we only discuss in detail those lemmas involved in the proof which are significantly different from the batch setting. In particular, we focus on the effect of the adaptive test scheduling policy. The rest of the proof is quite standard: first show that the algorithm recovers a transition graph isomorphic to a subgraph of the target containing all relevant states and transitions, and then bound the overall error in terms of the error in transition probabilities. We

note that by using a slightly different notion of insignificant state and applying a smoothing operation after learning a PDFA, our algorithm could also learn PDFA under the more strict KL divergence.

The next two lemmas establish the correctness of the structure recovered: with high probability, merges and promotions are correct, and no significant candidate state are marked as insignificant.

Lemma 4.3. *With probability at least* $1 - n(n+1)|\Sigma|\delta'$*, all transitions between safe states are correct.*

Proof. We will inductively bound the error probability of a merge or promotion by assuming that all the previous ones were correct. If all merges and promotions performed so far are correct, there is a transition-preserving bijection between the safe states in H and a subset of states from target A_D; therefore, for each safe state q the distribution of the strings added to $\hat{S}$ is the same as the one in the corresponding state in the target. Note that this also holds before the very first merge or promotion.

First we bound the probability that the next merge is incorrect. Suppose `StreamPDFALearner` is testing a candidate q and a safe q' such that $D_q \neq D_{q'}$ and decides to merge them. This will only happen if, for some $i \geq 1$ a call $\texttt{Test}(\hat{S}_q, \hat{S}_{q'}, \delta_i)$ returns `equal`. By Assumption 1, for fixed i this happens with probability at most δ_i; hence, the probability of this happening for some i is at most $\sum_i \delta_i$, and this sum is bounded by δ' because of the fact that $\sum_{i\geq 1} 1/i^2 = \pi^2/6$. Since there are at most n safe states, the probability of next merge being incorrect is at most $n\delta'$.

Next we bound the probability that next promotion is incorrect. Suppose that we promote a candidate q to safe but there exists a safe q' with $D_q = D_{q'}$. In order to promote q to safe the algorithm needs to certify that q is distinct from q'. This will happen if a call $\texttt{Test}(\hat{S}_q, \hat{S}_{q'}, \delta_i)$ return `distinct` for some i. But again, this will happen with probability at most $\sum_i \delta_i \leq \delta'$.

Since a maximum of $n|\Sigma|$ candidates will be processed by the algorithm, the probability of an error in the structure is at most $n(n+1)|\Sigma|\delta'$.

Following Palmer and Goldberg (2007), we say a state in a PDFA is *insignificant* if a random string passes through that state with probability less than $\varepsilon/2n|\Sigma|$; the same applies to transitions. It can be proved that a subgraph from a PDFA that contains all its non-insignificant states and transitions fails to accept a set of strings accepted by the original PDFA of total probability at most $\varepsilon/4$.

Lemma 4.4. *With probability at least* $1 - n|\Sigma|\delta'$ *no significant candidate will be marked insignificant and all insignificant candidates with probability less than* $\varepsilon/4n|\Sigma|$ *will be marked as insignificant during its first insignificance test.*

Proof. First note that when insignificance test for a candidate state q is performed, it means that $T_j = (64n|\Sigma|/\varepsilon)\ln(2^j/\delta')$ examples have been processed since its creation, for some $j \geq 1$. Now suppose q is a non-insignificant candidate, i.e. it has probability more than $\varepsilon/2n|\Sigma|$. Then, by the Chernoff bounds, we have $|\hat{S}_q|/T_j < 3\varepsilon/8n|\Sigma|$ with probability at most $\delta'/2^j$. Thus, q will be marked as insignificant with probability at most δ'. On the other hand, if q has

probability less than $\varepsilon/4n|\Sigma|$, then $|\hat{S}_q|/T_1 > 3\varepsilon/8n|\Sigma|$ happens with probability at most $\delta'/2$. Since there will be at most $n|\Sigma|$ candidates, the statement follows by the union bound.

Though the algorithm would be equally correct if only a single insignificance test was performed for each candidate state, the scheme followed here ensures the algorithm will terminate even when the distribution generating the stream changes during the execution and some candidate that was significant w.r.t. the previous target is insignificant w.r.t. to the new one.

With the results proved so far we can see that, with probability at least $1 - \delta/2$, the set of strings in the support of D not accepted by H have probability at most $\varepsilon/4$ w.r.t. D_T. Together with the guarantees on the probability estimations of $\hat{D}_H$ provided by Palmer (2008), we can see that with probability at least $1-\delta$ we have $\mathrm{L}_1(D, \hat{D}_H) \leq \varepsilon$.

Structure inference and probability estimation are presented here as two different phases of the learning process for clarity and ease of exposition. However, probabilities could be incrementaly estimated during the structure inference phase by counting the number of times each arc is used by the examples we observe in the stream, provided a final probability estimation phase is run to ensure that probabilities estimated for the last added transitions are also correct.

5 Sketching Distributions over Strings

In this section we describe two sketches that can be used by our state-merging algorithm in data streams. The basic building block of both is the Space-Saving algorithm Metwally et al (2005). By using it directly, we provide an implementation of `StreamPDFALearner` that learns with respect to the L_∞-distinguishability of the target PDFA. By extending it to store information about *frequent prefixes*, and a more involved analysis, we obtain instead learning with respect to the $\mathrm{L}^{\mathrm{p}}_\infty$-distinguishability. This information will then be used to compute the statistics required by the similarity test described in Appendix A.3.

We begin by recalling the basic properties of the Space-Saving sketch introduced in Metwally et al (2005).

Given a number K, the Space-Saving sketch $\mathtt{SpSv}(K)$ is a data structure that uses memory $O(K)$ for monitoring up to K "popular" items from a potentially huge domain X. The set of monitored items may vary as new elements in the input stream are processed. In essence, it keeps K counters. Each counter c_i keeps an overestimate of the frequency $f(x_i)$ in the stream of a currently monitored item x_i. The number of stream items processed at any given time, denoted as m from now on, equals both $\sum_{i=1\ldots K} c_i$ and $\sum_{x\in X} f(x)$. Two operations are defined on the sketch: the *insertion* operation that adds a new item to the sketch, and the *retrieval* operation that returns a ordered list of pairs formed by items and estimations.

The *insertion* operation is straightforward. If the item x to be inserted is already being monitored as x_i, the corresponding counter c_i is incremented. Else, if there are less than K items being monitored, x is added with count 1. Otherwise, the monitored item x_M having the smallest associated c_M is replaced

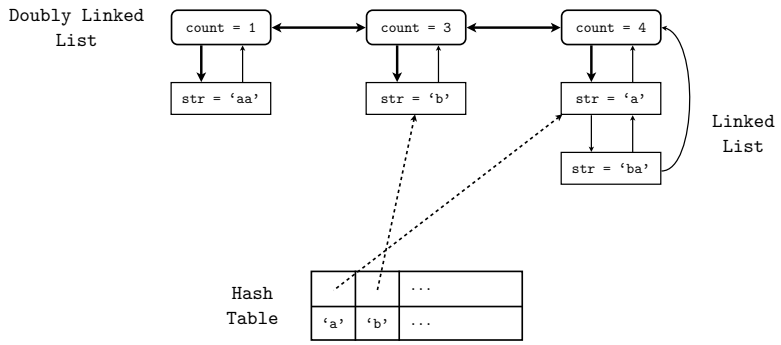


Figure 3: Conceptual representation of the Space-Saving data structure.

with x. The associated counter is incremented by 1, so in general it will now hold an overestimate of $f(x)$. This operation takes time $O(1)$.

The *retrieval* operation has a parameter $\varepsilon \geq 1/K$, takes time $O(1/\varepsilon)$ and returns at time t a set of at most K pairs of the form (x, c_x). The key properties of the sketch are: 1) This set is guaranteed to contain every x such that $f(x) \geq \varepsilon \cdot t$, and 2) For each (x, c_x) in the set, $0 \leq c_x - f(x) \leq t/K$. Claim 1) is shown as follows: Suppose x is not in the sketch at time t. If it was never in the sketch, then $f(x) = 0$ and we are done. Otherwise, suppose x was last removed from the sketch at time $t' < t$, and let c be its associated count at that moment. Since c was the smallest of K counts whose sum was t', we have $cK \leq t'$. Hence $f(x) \leq c \leq t'/K < t/K \leq \varepsilon \cdot t$. The proof of Claim 2) is similar.

An accurate description of a data structure with these requirements is given in Metwally et al (2005); we only outline it here. It consists of several linked list at two levels. At the first one, there is an ordered doubly linked list of buckets, each bucket being labeled by a distinct estimation value of monitored examples. At the second level, for each bucket there is a linked list representing monitored examples whose estimation corresponds to the bucket label. There are additional links leading from each represented example to its bucket and every bucket points to exactly one item among its child list. Finally, items are stored in a convenient structure, such as a hash table or associative memory, that guarantees constant access cost given the item. Figure 3 gives a simple example of this data structure.

Let us first analyze the direct use of `SpSv` to store information about the sample of strings reaching any given state. We first bound the error introduced by the sketch on the empirical L_∞ distance between the "exact" empirical distribution corresponding to a sample S and its sketched version, which we denote by $\hat{S}$. By the property of the sketch shown above, the following is easily seen to hold:

Lemma 5.1. *Let $S = (x_1, \ldots, x_m)$ be a sequence of examples and $\hat{S}$ a $\mathtt{SpSv}(K)$ sketch where each element of S has been inserted. Then $\mathrm{L}_\infty(S, \hat{S}) \leq 1/K$.*

Consider now an implementation of `StreamPDFALearner` that places a sketch as above with $K = 8/\mu$ in every safe or candidate state, and uses the `Test` described in Appendix A.3. By Lemma 5.1, $\mathrm{L}_\infty(S, \hat{S}) \leq \mu/8$ so the conditions of

Lemma A.6 in the appendix are satisfied. This pair of `Sketch` and `Test` satisfy all Assumptions 1 with $M_{\text{sketch}} = O(1/\mu)$, $T_{\text{sketch}} = O(1)$, $T_{\text{test}} = O(1/\mu)$, $N_{\text{unknown}} = \widetilde{O}(1/\mu^2)$, and $N_{\text{equal}} = \widetilde{O}(1/\mu^2)$. Applying Theorems 4.1 and 4.2 we have:

Corollary 5.2. *There are implementations of* `Test` *and* `Sketch` *such that* `StreamPDFALearner` *will PAC-learn streams generated by PDFA with n states and L_∞-distinguishability μ using memory $O(n|\Sigma|/\mu)$, reading $\widetilde{O}(n^2|\Sigma|^2/\varepsilon\mu^2 + n^4|\Sigma|^4/\varepsilon^3)$ examples in expectation, and processing each stream element x in time $O(|x|)$.*

Next, we propose a variant of the sketch useful for learning with respect to the $\mathrm{L}^{\mathrm{p}}_\infty$-distinguishability rather than the L_∞ one, whose analysis is substantially more involved. The sketch has to be modified to retrieve frequent prefixes from a stream of strings rather than the strings themselves. A first observation is that whenever we observe a string x of length $|x|$ in the stream, we should insert $|x| + 1$ prefixes to the sketch. This is another way of saying that under a distribution D over $\Sigma^\star$ events of the form $x\Sigma^\star$ and $y\Sigma^\star$ are not independent when x is a prefix of y. In fact, it can be shown that $\sum_x D(x\Sigma^\star) = L+1$, where $L = \sum_x |x|D(x)$ is the expected length of D (Clark and Thollard, 2004). In practice, a good estimate for L can be easily obtained from an initial fraction of the stream, so we assume it is known. It is easy to see that a Space-Saving sketch with $O(K\,L)$ counters can be used to retrieve prefixes with relative frequencies larger than some $\varepsilon \geq 1/K$ and approximating these frequencies with error at most $O(1/K)$. When computing relative frequencies, the absolute frequency obtained via a retrieval operation needs to be divided by the number of strings added to the sketch so far (instead of the number of prefixes).

We encapsulate all this behavior into a *Prefix-Space-Saving* sketch $\texttt{SpSv}^{\mathrm{p}}(K)$, which is basically a Space-Saving sketch with K counters where when one string is inserted, each proper prefix of the string is inserted into the sketch as well. A string x is processed in time $O(|x|)$. Such a sketch can be used to keep information about the frequent prefixes in a stream of strings, and the information in two Prefix-Space-Saving sketches corresponding to streams generated by different distributions can be used to approximate their $\mathrm{L}^{\mathrm{p}}_\infty$ distance.

We now analyze the error introduced by the sketch. As before, S and $\hat{S}$ denote the "exact" empirical distribution and its approximate version derived from the sketch. Fix $K > 0$. Given a sequence $S = (x_1, \ldots, x_m)$ of strings from $\Sigma^\star$, for each prefix $x \in \Sigma^\star$ we denote by $\hat{S}[x\Sigma^\star]$ the absolute frequency returned for prefix x by a Prefix-Space-Saving sketch $\texttt{SpSv}^{\mathrm{p}}(K)$ that received S as input; that is, $\hat{S}[x\Sigma^\star] = \hat{f}_x$ if the pair $(x, \hat{f}_x)$ was returned by a retrieval query with $\varepsilon = 1/K$, and $\hat{S}[x\Sigma^\star] = 0$ otherwise. Furthermore, $\hat{S}(x\Sigma^\star)$ denotes the relative frequency of the prefix x in $\hat{S}$: $\hat{S}(x\Sigma^\star) = \hat{S}[x\Sigma^\star]/m$. The following result analyzes the maximum of the differences $|\hat{S}(x\Sigma^\star) - S(x\Sigma^\star)|$.

Lemma 5.3. *Let $S = (x_1, \ldots, x_m)$ be a sequence of i.i.d. examples from a PDFA D and $\hat{S}$ a $\texttt{SpSv}^{\mathrm{p}}(K)$ sketch where each element of S has been inserted. Then for some c_D depending on D only and with probability at least $1 - \delta$ the following holds:*

$$\mathrm{L}^{\mathrm{p}}_\infty(S, \hat{S}) \leq \frac{L+1}{K} + \sqrt{\frac{32e^2}{mK^2c_D^2} \ln\left(\frac{1}{\delta}\right)} \ .$$

The proof is given in Appendix A.2. We will apply Lemma 5.3 to each state q of the target PDFA; let $c_D(q)$ be the constant c_D provided by the lemma for the distribution generated at q, and let κ be the smallest $c_D(q)$; one can view κ as property of the PDFA measuring some other form of its complexity. Similarly, let L_{max} be the largest among the expected lengths of strings generated from all states in the target PDFA; in particular $L \leq L_{max}$. Let now μ be a lower bound on the $\mathrm{L}_\infty^{\mathrm{p}}$ distinguishability of the target PDFA, and set

$$K = \frac{16}{\mu} \cdot \max\left\{ L_{max} + 1, \sqrt{\frac{32e^2}{\kappa^2} \ln\left(\frac{1}{\delta}\right)} \right\}.$$

The first argument of the max ensures that $(L+1)/K \leq \mu/16$, and the second argument in the max and the definition of κ and C_D ensure that $\sqrt{32e^2/(mK^2c_D^2)} \leq \mu/16$ as well. By Lemma 5.3 this ensures that for every state sketch we have $\mathrm{L}_\infty^{\mathrm{p}}(S, \hat{S}) \leq \mu/8$ with high probability for any sample size $m \geq 1$. We can apply then Lemma A.6 in the Appendix, and conclude that this pair of `Sketch` and `Test` satisfy all Assumptions 1 with $N_{\text{unknown}} = \widetilde{O}(1/\mu^2)$, $N_{\text{equal}} = \widetilde{O}(1/\mu^2)$, $M_{\text{sketch}} = T_{\text{test}} = \widetilde{O}(\max\{L_{max}, 1/\kappa\}/\mu)$, and $T_{\text{sketch}} = O(|x|)$. Applying Theorem 4.2 we have:

Corollary 5.4. *There are implementations of* `Test` *and* `Sketch` *such that* `StreamPDFALearner` *will PAC-learn streams generated by PDFA with n states and $\mathrm{L}_\infty^{\mathrm{p}}$-distinguishability μ using memory $\widetilde{O}((n|\Sigma|/\mu)\max\{L_{max}, 1/\kappa\})$, reading $\widetilde{O}(n^2|\Sigma|^2/\varepsilon\mu^2 + n^4|\Sigma|^4/\varepsilon^3)$ examples in expectation, and processing each stream element x in time $O(|x|^2)$.*

The algorithm above is supposed to receive as input the quantities L_{max} and κ, or at least upper bounds. To end this section, we note that we can get rid of L_{max} and κ in the corollary above at the price of introducing a dependence on ε in the amount of memory used by the algorithm. Indeed, consider the previous implementation of `StreamPDFALearner` and change it to use a sketch size that is provably sufficient for keeping accurate statistics for non-insignificant states, that is, those having probability larger than $\varepsilon/4n|\Sigma|$, although perhaps insufficient for insignificant states that generate long strings. Tests for insignificant states will (with high probability) result neither in merging nor in promotion, so `StreamPDFALearner` will not alter its behavior if sketch size is limited in this way for all states. Indeed, by Lemma A.2 in Appendix A.1, the constant c_D associated to each significant state is at least $\varepsilon/4n|\Sigma|L$, and therefore we derive a bound, independent of κ, on the sketch size sufficient for non-insignificant states, and the total memory used by this implementation of `StreamPDFALearner` is $\widetilde{O}(n^2|\Sigma|^2L/\mu\varepsilon)$.

6 A Strategy for Searching Parameters

Besides other parameters, a full implementation of `StreamPDFALearner`, `Sketch`, and `Test` as used in the previous section require a user to guess the number of states n and distinguishability μ of the target PDFA in order to learn it properly. These parameters are a priori hard to estimate from a sample of strings. And though in the batch setting a cross-validation-like strategy can be used to select these parameters in a principled way, the panorama in a data streams setting is far less encouraging. This is not only because storing a sample to

cross-validate the parameters clashes with the data streams paradigm, but also because when the target changes over time the algorithm needs to detect these changes and react accordingly. Here we focus on a fundamental part of this adaptive behavior: choosing the right n and μ for the current target.

We will give an algorithm capable of finding these parameters by just examining the output of previous calls to `StreamPDFALearner`. The algorithm has to deal with a trade-off between memory growth and time taken to find the correct number of states and distinguishability. This compromise is expressed by a pair of parameters given to the algorithm: $\rho > 1$ and $\phi > 0$. Here we assume that `StreamPDFALearner` receives as input just the current estimations for n and μ. Furthermore, we assume that there exists an unknown fixed PDFA with $n_\star$ states and distinguishability $\mu_\star$ which generates the strings in the stream. The rest of input parameters to `StreamPDFALearner` – Σ, ε, and δ – are considered fixed and ignored hereafter. Our goal is to identify as fast as possible (satisfying some memory constraints) parameters $n \geq n_\star$ and $\mu \leq \mu_\star$, which will allow `StreamPDFALearner` to learn the target accurately with high probability. For the sake of concreteness, and because they are satisfied by all the implementations of `Sketch` and `Test` we have considered, in addition to Assumption 1, we make the following assumption.

Assumption 2. Given a distinguishability parameter μ for the target PDFA, algorithms `Sketch` and `Test` satisfy Assumption 1 with $M_{\text{sketch}} = \Theta(1/\mu)$ and $N_{\text{equal}} = \Theta(1/\mu^2)$.

Our algorithm is called `ParameterSearch` and is described in Figure 2. It consists of an infinite loop where successive calls to `StreamPDFALearner` are performed, each with different parameters n and μ. `ParameterSearch` tries to find the correct target parameters using properties from successive hypothesis produced by `StreamPDFALearner` as a guide. Roughly, the algorithm increments the number of states n if more than n states were discovered in the last run, and decreases distinguishability μ otherwise. However, in order to control the amount of memory used by `ParameterSearch` distinguishability needs to be decreased sometimes even if the last hypothesis' size exceeded n. This is done by imposing as invariant to be maintained throughout the whole execution that $n \leq (1/\mu)^{2\phi}$. This invariant is key to proving the following results.

Theorem 6.1. *After each call to* `StreamPDFALearner` *where at least one merge happened, the memory used by* `ParameterSearch` *is* $O(t^{1/2+\phi})$*, where* t *denotes the number of examples read from the stream so far.*

Proof. First note that, by the choice of ρ', the invariant $n \leq (1/\mu)^{2\phi}$ is maintained throughout the execution of `ParameterSearch`. Therefore, at all times $n/\mu \leq (1/\mu)^{1+2\phi} \leq (1/\mu^2 + c)^{1/2+\phi}$ for any $c \geq 0$. Suppose that a sequence of $k \geq 1$ calls to `StreamPDFALearner` are made with parameters n_i, μ_i for $i \in [k]$. Write t_i for the number of elements read from the stream during the ith call, and $t = \sum_{i \leq k} t_i$ for the total number of elements read from the stream after the kth call. Now assume a merge occured in the process of learning the kth hypothesis, thus $t_k = \Omega(1/\mu_k^2)$ by Theorem 4.1 and Assumption 2. Therefore we have $t^{1/2+\phi} = (\sum_{i<k} t_i + \Omega(1/\mu_k^2))^{1/2+\phi} = \Omega(n_k/\mu_k)$. By Theorem 4.1 the memory in use after the kth call to `StreamPDFALearner` is $O(n_k M_{\text{sketch}}) = O(n_k/\mu_k)$.

Note the memory bound does not apply when `StreamPDFALearner` produces tree-shaped hypotheses because in that case the algorithm makes no merges.

However, if the target is a non-tree PDFA, then merges will always occur. On the other hand, if the target happens to be tree-shaped, our algorithm will learn it quickly (because no merges are needed). A stopping condition for this situation could be easily implemented, thus restricting the amount of memory used by the algorithm in this situation.

Next theorem quantifies the overhead that `ParameterSearch` pays for not knowing a priori the parameters of the target. This overhead depends on ρ and ϕ, and introduces a trade-off between memory usage and time until learning.

Theorem 6.2. *Assume $\phi < 1/2$. When the stream is generated by a PDFA with $n_\star$ states and distinguishability $\mu_\star$,* `ParameterSearch` *will find a correct hypothesis after making at most $O(\log_\rho(n_\star/\mu_\star^{2\phi}))$ calls to* `StreamPDFALearner` *and reading in expectation at most $\widetilde{O}(n_\star^{1/\phi}\rho^{2+1/\phi}/\mu_\star^2)$ elements from the stream.*

Proof. For convenience we assume that all $n_\star$ states in the target are important – the same argument works with minor modifications when $n_\star$ denotes the number of important states in the target. By Theorem 4.2 the hypothesis will be correct whenever the parameters supplied to `StreamPDFALearner` satisfy $n \geq n_\star$ and $\mu \leq \mu_\star$. If n_{k+1} and μ_{k+1} denote the parameters computed after the kth call to `StreamPDFALearner` we will show that $n_{k+1} \geq n_\star$ and $\mu_{k+1} \leq \mu_\star$ for some $k = O(\log_\rho(n_\star/\mu_\star^{2\phi}))$.

Given a set of k calls to `StreamPDFALearner` let us write $k = k_1 + k_2 + k_3$, where k_1, k_2 and k_3 respectively count the number of times n, μ, or n *and* μ are modified after a call to `StreamPDFALearner`. Now let $k_n = k_1 + k_3$, $k_\mu = k_2 + k_3$. Considering the first k calls to `StreamPDFALearner` in `ParameterSearch` one observes that $n_{k+1} = \rho^{1+k_n}$ and $1/\mu_{k+1} = \rho^{(1+k_\mu)/2\phi}$. Thus, from the invariant $n_{k+1} \leq (1/\mu_{k+1})^{2\phi}$ maintained by `ParameterSearch` (see the proof of Theorem 6.1), we see that $k_1 \leq k_2$ must necessarily hold.

Now assume k is the smallest integer such that $\mu_{k+1} \leq \mu_\star$. By definition of k we must have $\mu_{k+1} > \mu_\star/\rho'$, and $1/\mu_{k+1} = \rho^{(1+k_\mu)/2\phi}$, hence $k_\mu < \log_\rho(1/\mu_\star^{2\phi})$. Therefore we see that $k = k_1 + k_2 + k_3 \leq 2k_2 + k_3 \leq 2k_\mu < 2\log_\rho(1/\mu_\star^{2\phi})$. Next we observe that if $\mu \leq \mu_\star$ and n are fed to `StreamPDFALearner` then it will return a hypothesis with $|H| \geq n$ whenever $n < n_\star$. Thus, after the first k calls, n is incremented at each iteration. Therefore we must have at most $\log_\rho n_\star$ additional calls until $n \geq n_\star$. Together with the previous bound, we get a total of $O(\log_\rho(n_\star/\mu_\star^{2\phi}))$ calls to the learner.

It remains to bound the number of examples used in these calls. Note that once the correct μ is found, it will only be further decreased in order to maintain the invariant; hence, $1/\mu_{k+1} \leq \rho^{1/2\phi} \cdot \max\{1/\mu_\star, n_\star^{1/2\phi}\}$. Furthermore, if the correct μ is found before the correct n, the latter will never surpass $n_\star$ by more than ρ. However, it could happen that n grows more than really needed while $\mu > \mu_\star$; in this case the invariant will keep μ decreasing. Therefore, in the end $n_{k+1} \leq \rho \cdot \max\{n_\star, 1/\mu_\star^{2\phi}\}$. Note that since $\phi < 1/2$ we have $\max\{1/\mu_\star, n_\star^{1/2\phi}\} \cdot \max\{n_\star, 1/\mu_\star^{2\phi}\} = O(n_\star^{1/2\phi}/\mu_\star)$. Thus, by Theorem 4.1 we see that in expectation `ParameterSearch` will read at most $O(n_{k+1}^2 N_{\text{unknown}} \log_\rho(n_\star/\mu_\star^{2\phi})) = \widetilde{O}(n_\star^{1/\phi}\rho^{2+1/\phi}/\mu_\star^2)$ elements from the stream.

Note the trade-off in the choice of ϕ stressed by this result: small values guarantee little memory usage while potentially increasing the time until learn-

ing. A user should tune ϕ to meet the memory requirements of its particular system.

Algorithm 2: `ParameterSearch` algorithm

Input: Parameters ρ, ϕ
Data: A stream of strings $x_1, x_2, \ldots \in \Sigma^\star$

$\rho' \leftarrow \rho^{1/2\phi}$;
$n \leftarrow \rho$, $\mu \leftarrow 1/\rho'$;
while true do
  $H \leftarrow$ `StreamPDFALearner`(n, μ);
  if $|H| < n$ **then** $\mu \leftarrow \mu/\rho'$;
  else
    $n \leftarrow n \cdot \rho$;
    if $n > (1/\mu)^{2\phi}$ **then** $\mu \leftarrow \mu/\rho'$;

7 Detecting Changes in the Target

Now we describe a change detector `ChangeDetector` for the task of learning distributions from streams of strings using PDFA. Our detector receives as input a DFA H, a change threshold γ, and a confidence parameter δ. It then runs until a change relevant w.r.t. the structure of H is observed in the stream distribution. This DFA is *not* required to model the structure of current distribution, though the more accurate it is, the more sensitive to changes will the detector be. In the application we have in mind, the DFA will be a hypothesis produced by `StreamPDFALearner`.

We define first some new notation. Given a DFA H and a distribution D on $\Sigma^\star$, $D(H[q]\Sigma^\star)$ is the probability of all words visiting state q at least once. Given a sample from D, we denote by $\hat{D}(H[q]\Sigma^\star)$ the relative frequency of words passing through q in the sample. Furthermore, we denote by $\mathbf{D}$ the vector containing $D(H[q]\Sigma^\star)$ for all states q in H, and by $\hat{\mathbf{D}}$ the vector of empirical estimations.

Our detector will make successive estimations $\hat{\mathbf{D}}_0$, $\hat{\mathbf{D}}_1$, ... and decide that a change happened if some of these estimations differ too much. The rationale behind this approach to change detection is justified by the next lemma, showing that a non-negligible difference between $D(H[q]\Sigma^\star)$ and $D'(H[q]\Sigma^\star)$ implies a non-negligible distance between D and D'.

Lemma 7.1. *If for some state $q \in H$ we have $|D(H[q]\Sigma^\star) - D'(H[q]\Sigma^\star)| > \gamma$, then $\mathrm{L}_1(D, D') > \gamma$.*

Note that the converse is not true, that is there are arbitrarily different distributions that our test may confuse. An easy counterexample occurs for if H accepts $\Sigma^\star$ with one state; then the distance will be 0 for every D and D'. Other change detection mechanisms could be added to our scheme.

Proof. First note that $\mathrm{L}_1(D, D') \geq \sum_{x \in H[q], y \in \Sigma^\star} |D(xy) - D'(xy)|$. Then, by triangle inequality this is at least $|\sum_{x \in H[q], y \in \Sigma^\star} D(xy) - D'(xy)| = |D(H[q]\Sigma^\star) - D'(H[q]\Sigma^\star)| > \gamma$.

More precisely, our change detector `ChangeDetector` works as follows. It first reads $m = (8/\gamma^2)\ln(4|H|/\delta)$ examples from the stream and uses them to estimate $\hat{\mathbf{D}}_0$ in H. Then, for $i > 0$, it makes successive estimations $\hat{\mathbf{D}}_i$ using $m_i = (8/\gamma^2)\ln(2\pi^2 i^2|H|/3\delta)$ examples, until $\|\hat{\mathbf{D}}_0 - \hat{\mathbf{D}}_i\|_\infty > \gamma/2$, at which point a change is declared.

We will bound the probability of false positives and false negatives in `ChangeDetector`. For simplicity, we assume the elements on the stream are generated by a succession of distributions $D_0, D_1, \ldots$ with changes taking place only between successive estimations $\hat{\mathbf{D}}_i$ of state probabilities. The first lemma is about false positives.

Lemma 7.2. *If $D = D_i$ for all $i \geq 0$, with probability at least $1 - \delta$ no change will be detected.*

Proof. We will consider the case with a single state q; the general case follows from a simple union bound. Let $p = D(H[q]\Sigma^\star)$. We denote by $\hat{p}_0$ an estimation of p obtained with $(8/\gamma^2)\ln(4/\delta)$ examples, and by $\hat{p}_i$ an estimation from $(8/\gamma^2)\ln(2\pi^2 i^2/3\delta)$ examples. Recall that $p = \mathbb{E}[\hat{p}_0] = \mathbb{E}[\hat{p}_i]$. Now, change will be detected if for some $i > 0$ one gets $|\hat{p}_0 - \hat{p}_i| > \gamma/2$. If this happens, then necessarily either $|\hat{p}_0 - p| > \gamma/4$ or $|\hat{p}_i - p| > \gamma/4$ for that particular i. Thus, by Hoeffding's inequality, the probability of a false positive is at most $\mathbb{P}[|\hat{p}_0 - p| > \gamma/4] + \sum_{i>0} \mathbb{P}[|\hat{p}_i - p| > \gamma/4] \leq \delta$.

Next we consider the possiblity that a change occurs but is not detected.

Lemma 7.3. *If $D = D_i$ for all $i < k$ and $|D_{k-1}(H[q]\Sigma^\star) - D_k(H[q]\Sigma^\star)| > \gamma$ for some $q \in H$, then with probability at least $1 - \delta$ a change will be detected. Furthermore, if the change occurs at time t, then it is detected after reading at most $O(1/\gamma^2 \ln(\gamma t/\delta))$ examples more.*

Proof. As in Lemma 7.2, we prove it for the case with a single state. The same notation is also used, with $p_k = D_k(H[q]\Sigma^\star)$. The statement will not be satisfied if a change is detected before the kth estimation or no change is detected immediately after it. Let us assume that $|\hat{p}_0 - p| \leq \gamma/4$. Then, a false positive will occur if $|\hat{p}_0 - \hat{p}_i| > \gamma/2$ for some $i < k$, which by our assumption implies $|\hat{p}_i - p| > \gamma/4$. On the other hand, a false negative will occur if $|\hat{p}_0 - \hat{p}_k| \leq \gamma/2$. Since $|p - p_k| > \gamma$, our assumption implies that necessarily $|\hat{p}_k - p_k| > \gamma/4$. Therefore, the probability that the statement fails can be bounded by

$$\mathbb{P}[|\hat{p}_0 - p| > \gamma/4] + \sum_{0<i<k} \mathbb{P}[|\hat{p}_i - p| > \gamma/4] + \mathbb{P}[|\hat{p}_k - p_k| > \gamma/4] \ ,$$

which by Hoeffding's inequality is at most δ.

Now assume the change happened at time t. By Stirling's approximation we have $t = \Theta(\sum_{i\leq k}(1/\gamma^2)\ln(k^2/\delta)) = \Theta(k/\gamma^2 \ln k/\delta)$. Therefore $k = O(\gamma^2 t)$. If the change is detected at the end of the following window (which will happen with probability at least $1 - \delta$), then the response time is at most $O(1/\gamma^2 \ln(\gamma t/\delta))$.

8 Conclusions and Future Work

We have presented an algorithm that learns PDFA in the computationally strict data stream model, and is able to adapt to changes in the input distribution.

It has rigorous PAC-learning bounds on sample size required for convergence, both for learning its first hypothesis and for adapting after an abrupt change takes place. Furthermore, unlike other (batch) algorithms for the same task, it learns unknown target parameters (number of states and distinguishability) from the stream instead of requiring guesses from the user, and adapts to the complexity of the target so that it need not use the sample sizes stated by the worst-case bounds.

We are currently performing synthetic experiments on an initial implementation to investigate its efficiency bottlenecks. As future work, we would like to investigate whether the learning bounds can be tightened according to the observed experimental evidence, and further by relaxing the worst-case, overestimating bounds in the tests performed by the algorithm. The bootstrap-based state-similarity test outlined in Balle et al (2012a) seems very promising in this respect. It would also be interesting to parallelize the method so that it can scale to very high-speed data streams.

Acknowledgements

This work was partially supported by MICINN projects TIN2011-27479-C04-03 (BASMATI) and TIN-2007-66523 (FORMALISM), by SGR2009-1428 (LARCA), and by the EU PASCAL2 Network of Excellence (FP7-ICT-216886). B. Balle is supported by an FPU fellowship (AP2008-02064) from the Spanish Ministry of Education.

A preliminary version of this work was presented at the 11th Intl. Conf. on Grammatical Inference (Balle et al, 2012a). Here we provide missing proofs and discussions, and extend the results there to streams that evolve over time. On the other hand, Balle et al (2012a) outlined an efficient state-similarity test based on bootstrapping. Because it can be used independently of the specific PDFA learning method discussed here, and the full presentation and analysis are long, it will be published elsewhere.

References

Aggarwal C (ed) (2007) Data Streams – Models and Algorithms. Springer

Balle B, Castro J, Gavaldà R (2012a) Bootstrapping and learning pdfa in data streams. In: International colloquium on Grammatical Inference (ICGI)

Balle B, Castro J, Gavaldà R (2012b) Learning probabilistic automata: A study in state distinguishability. Theoretical Computer Science

Bifet A (2010) Adaptive Stream Mining: Pattern Learning and Mining from Evolving Data Streams. IOS Press - Frontiers of Artificial Intelligence Series and Applications

Bousquet O, Boucheron S, Lugosi G (2004) Introduction to statistical learning theory. Advanced Lectures on Machine Learning

Carrasco RC, Oncina J (1999) Learning deterministic regular grammars from stochastic samples in polynomial time. Informatique Théorique et Applications 33(1):1–20

Castro J, Gavaldà R (2008) Towards feasible PAC-learning of probabilistic deterministic finite automata. In: International colloquium on Grammatical Inference (ICGI)

Clark A, Thollard F (2004) PAC-learnability of probabilistic deterministic finite state automata. Journal of Machine Learning Research

Dupont P, Denis F, Esposito Y (2005) Links between probabilistic automata and hidden markov models: probability distributions, learning models and induction algorithms. Pattern Recognition

Gama J (2010) Knowledge Discovery from Data Streams. Taylor and Francis

Guttman O, Vishwanathan SVN, Williamson RC (2005) Learnability of probabilistic automata via oracles. In: Conference on Algorithmic Learning Theory (ALT)

de la Higuera C (2010) Grammatical Inference: Learning Automata and Grammars. Cambridge University Press

Hsu D, Kakade SM, Zhang T (2009) A spectral algorithm for learning hidden markov models. In: Conference on Learning Theory (COLT)

Kearns MJ, Mansour Y, Ron D, Rubinfeld R, Schapire RE, Sellie L (1994) On the learnability of discrete distributions. In: Symposium on Theory of Computation (STOC)

Lin X, Zhang Y (2008) Aggregate computation over data streams. In: Asian-Pacific Web Conference (APWeb)

Menascé DA, Almeida VAF, Fonseca R, Mendes MA (1999) A methodology for workload characterization of e-commerce sites. In: Proceedings of the 1st ACM conference on Electronic commerce, ACM, New York, NY, USA, EC '99, pp 119–128, DOI 10.1145/336992.337024, URL http://doi.acm.org/10.1145/336992.337024

Metwally A, Agrawal D, Abbadi A (2005) Efficient computation of frequent and top-k elements in data streams. In: International Conference on Database Theory (ICDT)

Muthukrishnan S (2005) Data streams: algorithms and applications. Foundations and Trends in Theoretical Computer Science

Palmer N, Goldberg PW (2007) PAC-learnability of probabilistic deterministic finite state automata in terms of variation distance. Theor Comput Sci

Palmer NJ (2008) Pattern classification via unsupervised learners. PhD thesis, University of Warwick

Ron D, Singer Y, Tishby N (1998) On the learnability and usage of acyclic probabilistic finite automata. Journal of Computing Systems Science

Schmidt J, Kramer S (2012) Online induction of probabilistic real time automata. Data Mining, IEEE International Conference on 0:625–634, DOI http://doi.ieeecomputersociety.org/10.1109/ICDM.2012.121

Schmidt J, Ansorge S, Kramer S (2012) Scalable induction of probabilistic real-time automata using maximum frequent pattern based clustering. In: Proceedings of the Twelfth SIAM International Conference on Data Mining, pp 272–283

Terwijn S (2002) On the learnability of hidden markov models. In: Intl. Conf. on Grammatical Inference (ICGI)

Vershynin R (2012) Introduction to the non-asymptotic analysis of random matrices. In: Eldar Y, Kutyniok G (eds) Compressed Sensing, Theory and Applications, CUP, chap 5

Vidal E, Thollard F, de la Higuera C, Casacuberta F, Carrasco RC (2005a) Probabilistic finite-state machines - part I. IEEE Transactions on Pattern Analysis and Machine Intelligence

Vidal E, Thollard F, de la Higuera C, Casacuberta F, Carrasco RC (2005b) Probabilistic finite-state machines - part II. IEEE Transactions on Pattern Analysis and Machine Intelligence

A Technical Results

A.1 Structural Results on PDFA

A real random variable X is *sub-exponential* if there exists a constant $c > 0$ such that $\mathbb{P}[|X| \geq t] \leq \exp(-ct)$ holds for all $t \geq 0$. The length of the strings generated by a PFA is always a sub-exponential random variable. Indeed, the following holds:

Lemma A.1. *For any PFA D it holds,*

1. *There exists a c_D such that $\mathbb{P}_{x\sim D}[|x| \geq t] \leq \exp(-c_D t)$ holds for all $t \geq 0$.*
2. *The expected length of the strings generated by D is at most $1/c_D$*

Proof. We show the first statement. Let n be the number of states of D and let q be any state. Starting from state q and before generating n new alphabet symbols, there is a probability $\rho > 0$ of emitting the final symbol ξ. Thus,

$$\mathbb{P}_{x\sim D}[|x| \geq t] \leq \mathbb{P}_{x\sim D}[|x| \geq \lfloor t/n \rfloor n] \leq (1-\rho)^{\lfloor t/n \rfloor} \leq \exp(-\rho \lfloor t/n \rfloor)$$

The last statement follows from the first one and from the expectation expression of a exponential distribution.

The following lemma gives an upper bound on the constant c_D in the previous one for states whose probability is not too small:

Lemma A.2. *Let q be a state of the target PDFA with $\Pr_D[x \text{ visits } q] \geq p$. The the constant $c_D(q)$ given by Lemma A.1 for the distribution generated from state q can be taken to be $c_D(q) = p/L$.*

Proof. Recall that $L = E_D[|x|]$, and let A be the event indicating that a string x generated from the target distribution D visits q. Then we have $E_{x\sim D}[\,|x|\,] \geq \Pr_D[A] \cdot E_{x\sim D}[\,|x| \mid A\,] \geq p \cdot E_{y\sim D(q)}[\,|y|\,]$. Hence, $E_{y\sim D(q)}[\,|y|\,] \leq L/p$. Now note that for every $\alpha > 1$ we must have $\Pr_{y\sim D(q)}[\,|y| \geq \alpha L/p\,] \leq 1/\alpha$; otherwise $E_{y\sim D(q)}[\,|y|\,] > L/p$.

As in Lemma A.1, let ρ be the probability that starting from state q and before generating n new symbols the machine stops. Then

$$\begin{aligned} E_{y\sim D(q)}[\,|y|\,] &= \sum_t \Pr_{y\sim D(q)}[\,|y| = t\,] \cdot t \geq \sum_{i\geq 0} \Pr_{y\sim D(q)}[\,|y| \in [in \ldots (i+1)n-1\,] \cdot i\,n \\ &\geq \sum_{i\geq 0}(1-\rho)^i \rho \cdot (in) = (\sum_{i\geq 0}(1-\rho)^i\, i\,\rho) \cdot n = (1-\rho)\, n/\rho. \end{aligned}$$

We must thus have $(1-\rho)\, n/\rho \leq E_{D(q)}[|y|] \leq L/p$, and therefore $\rho \geq n/(1 + L/p) \cong np/L$. The constant c_D can be taken, from the proof of Lemma A.1, to be $c_D = \rho/n$, therefore $c_D = p/L$ suffices.

A.2 Proof of Lemma 5.3

We use Lemma A.1 and the following theorem.

Theorem A.3 (Vershynin (2012))**.** *Let $X_1, \ldots, X_m$ be i.i.d. sub-exponential random variables and write $Z = \sum_{i=1}^m X_i$. Then, for every $t \geq 0$, we have*

$$\mathbb{P}[Z - \mathbb{E}[Z] \geq mt] \leq \exp\left(-\frac{m}{8e^2}\min\left\{\frac{t^2}{4c^2}, \frac{t}{2c}\right\}\right) \quad , \tag{1}$$

where c is the sub-exponential constant of variables X_i.

In the first place note that the number of elements inserted into the underlying space-saving sketch is $M = \sum_{i=1}^m (|x_i| + 1)$. This is a random variable whose expectation is $\mathbb{E}[M] = m(L+1)$. We claim that for any $x \in \Sigma^\star$ we have $|S(x\Sigma^\star) - \hat{S}(x\Sigma^\star)| \leq M/Km$. If $\hat{S}[x\Sigma^\star] = 0$, that means that $S[x\Sigma^\star] < M/K$, and therefore the bound holds. On the other hand, the guarantee on the sketch gives us $|S[x\Sigma^\star] - \hat{S}[x\Sigma^\star]|/m \leq M/Km$. Now, since $Z = M - m$ is the sum of m i.i.d. subexponential random variables by Lemma A.1, we can apply Theorem A.1 and obtain

$$\mathbb{P}[Z - \mathbb{E}[Z] \geq mt] \leq \exp\left(-\frac{m}{8e^2}\min\left\{\frac{t^2 c_D^2}{4}, \frac{t c_D}{2}\right\}\right) \quad . \tag{2}$$

The bound follows from choosing $t = \sqrt{(32e^2/mc_D^2)\ln(1/\delta)}$.

A.3 Similarity Tests

Suppose D and D' are two arbitrary distributions over $\Sigma^\star$. Let S be sample of m i.i.d. examples drawn from D and S' a sample of m' i.i.d. examples drawn from D'. For any event $A \subseteq \Sigma^\star$ we will use $S(A)$ to denote the *empirical probability of A under sample S*, which should in principle be an approximation to the probability $D(A)$. More specifically, if for any $x \in \Sigma^\star$ we let $S[x]$ denote the number of times that x appears in $S = (x^1, \ldots, x^m)$, then

$$S(A) = \frac{1}{m}\sum_{x\in A} S[x] = \frac{1}{m}\sum_{i=1}^m \mathbb{1}_{x_i \in A} \quad . \tag{3}$$

The sizes of S and S' will come into play in the statement of several results. The two following related quantities will be useful:

$$M = m + m', \qquad \text{and} \qquad M' = \frac{mm'}{(\sqrt{m} + \sqrt{m'})^2} \ . \tag{4}$$

The main tool we will use to build similarity tests are confidence intervals. Suppose dist is a distance measure between probability distributions and let $\mu_\star = \mathsf{dist}(D, D')$. Fix $0 < \delta < 1$ to be a confidence parameter. An *upper confidence limit* for $\mu_\star$ at confidence level δ computed from S and S' is a number $\hat{\mu}_U = \hat{\mu}_U(S, S', \delta)$ which for any two distributions D and D' satisfies $\mu_\star \leq \hat{\mu}_U$ with probability at least $1 - \delta$. Similarly, we define a *lower confidence limit* $\hat{\mu}_L$ satisfying $\hat{\mu}_L \leq \mu_\star$ with the same guarantees. Using these two quantities one can define a *confidence interval* $[\hat{\mu}_L, \hat{\mu}_U]$ which will contain $\mu_\star$ with probability at least $1 - 2\delta$.

Given a confidence interval $[\hat{\mu}_L, \hat{\mu}_U]$ for $\mu_\star$ and a distinguishability parameter μ it is simple to construct a similarity test as follows. If $\hat{\mu}_U < \mu$ then decide that $D = D'$ since we know that (with high probability) $\mu_\star < \mu$ which, by the promise on μ, implies $\mu_\star = 0$. If $\hat{\mu}_L > 0$ then decide that $D \neq D'$ since we then know that (with high probability) $\mu_\star > 0$. If none of the conditions above hold, then answer `unknown`. Let D and D' be two distributions over $\Sigma^\star$ with $\mu_\star = \mathrm{L}_\infty^{\mathrm{p}}(D, D')$. Suppose we have access to two i.i.d. samples S and S' drawn from D and D' respectively, where $m = |S|$ and $m' = |S'|$. We will use the empirical estimate $\hat{\mu} = \mathsf{dist}(S, S')$ to determine whether D and D' are equal or different. In particular, we will give a confidence interval for $\mu_\star$ centered around $\hat{\mu}$ for the cases $\mathsf{dist} = \mathrm{L}_\infty$ and $\mathsf{dist} = \mathrm{L}_\infty^{\mathrm{p}}$.

We use the well-known *Vapnik–Chernonenkis inequality*:

$$\mathbb{P}_{S \sim D^m}[\sup_{f \in \mathcal{F}} |\hat{\mathbb{E}}_S[f] - \mathbb{E}_D[f]| > t] \leq 4\Pi_{\mathcal{F}}(2m) \exp(-mt^2/8), \tag{5}$$

where $\Pi_{\mathcal{F}}$ is the *growth function* of class $\mathcal{F}$; see e.g. Bousquet et al (2004). Note that in the particular cases we are interested, the distance dist is defined as a supremum of some absolute diferences. For example, when $\mathsf{dist} = \mathrm{L}_\infty$ we have

$$\mathrm{L}_\infty(D, D') = \sup_{x \in \Sigma^\star} |D(x) - D'(x)| = \sup_{f \in \mathcal{F}_{\mathrm{L}_\infty}} |\mathbb{E}_D[f] - \mathbb{E}_{D'}[f]| \ , \tag{6}$$

where $\mathcal{F}_{\mathrm{L}_\infty} = \{\mathbb{1}_{\{x\}} \mid x \in \Sigma^\star\}$ is the set of indicator functions over all singletons of $\Sigma^\star$. In this case it is immediate to see that $\Pi_{\mathcal{F}_{\mathrm{L}_\infty}}(m) = m + 1$. In the case of $\mathrm{L}_\infty^{\mathrm{p}}$ distance we have $\mathcal{F}_{\mathrm{L}_\infty^{\mathrm{p}}} = \{\mathbb{1}_{x\Sigma^\star} \mid x \in \Sigma^\star\}$. A simple calculation shows that in this case $\Pi_{\mathcal{F}_{\mathrm{L}_\infty^{\mathrm{p}}}}(m) = 2m$. Thus, a direct application of Vapnik–Chervonenkis inequality can be used to proof the following results giving confidence limits for $\mu_\star$ of the form $\hat{\mu} \pm \Delta(\delta, m, m')$ in the cases $\mathsf{dist} = \mathrm{L}_\infty$ and $\mathsf{dist} = \mathrm{L}_\infty^{\mathrm{p}}$. For any $0 < \delta < 1$ let us define

$$\Delta(\delta, m, m') = \sqrt{\frac{8}{M'} \ln\left(\frac{16M}{\delta}\right)} \ . \tag{7}$$

Proposition A.4. *Suppose* $\mathsf{dist} = \mathrm{L}_\infty$ *or* $\mathsf{dist} = \mathrm{L}_\infty^{\mathrm{p}}$. *With probability at least* $1 - \delta$ *we have* $\mu_\star \leq \hat{\mu} + \Delta(\delta, m, m')$.

Proof. Let us write $\Delta = \Delta(\delta, m, m')$ for some fixed δ, m, and m'. The result follows from a standard application of the Vapnik–Chervonenkis inequality showing that $\mathbb{P}[\hat{\mu} < \mu_\star - \Delta] \leq \delta$. First note that by the triangle inequality we have $\mathsf{dist}(S, S') \geq \mathsf{dist}(D, D') - \mathsf{dist}(D, S) - \mathsf{dist}(D', S')$. Therefore, $\hat{\mu} < \mu_\star - \Delta$ implies $\mathsf{dist}(D, S) + \mathsf{dist}(D', S') > \Delta$. Now for any $0 < \gamma < 1$ we have

$$\begin{aligned}
&\mathbb{P}[\hat{\mu} < \mu_\star - \Delta] \leq \mathbb{P}[\mathsf{dist}(D, S) + \mathsf{dist}(D', S') > \Delta] \leq \\
&\mathbb{P}[\mathsf{dist}(D, S) > \gamma\Delta] + \mathbb{P}[\mathsf{dist}(D', S') > (1-\gamma)\Delta] \leq \\
&16me^{-m\gamma^2\Delta^2/8} + 16m'e^{-m'(1-\gamma)^2\Delta^2/8} = \\
&16Me^{-M'\Delta^2/8} = \delta \ ,
\end{aligned}$$

where we choose γ such that $m\gamma^2 = m'(1-\gamma)^2$.

Proposition A.5. *Suppose* $\mathsf{dist} = \mathrm{L}_\infty$ *or* $\mathsf{dist} = \mathrm{L}_\infty^{\mathrm{p}}$. *With probability at least* $1 - \delta$ *we have* $\mu_\star \geq \hat{\mu} - \Delta(\delta, m, m')$.

Proof. The argument is very similar to the one used in Proposition A.4. Write $\Delta = \Delta(\delta, m, m')$ for fixed parameters δ as well. We need to see that $\mathbb{P}[\mu_\star < \hat{\mu} - \Delta] \leq \delta$. Since $\mathsf{dist}(S, S') \leq \mathsf{dist}(D, D') + \mathsf{dist}(D, S) + \mathsf{dist}(D, D')$, then $\mu_\star < \hat{\mu} - \Delta$ implies $\mathsf{dist}(D, S) + \mathsf{dist}(D, D') > \Delta$. Thus, the conclusion follows from the same bound we used before.

Let us now describe a specific implementation of the `Test` used by the state-merging algorithm. It takes as parameters two sketches $\hat{S}_1$, $\hat{S}_2$, a lower bound μ on the dist-distinguishability μ_* of the target, (for some distance $\mathsf{dist} \in \{\mathrm{L}_\infty, \mathrm{L}_\infty^{\mathrm{p}}\}$) and a confidence parameter δ, and performs as follows:

1. let m, m' be the number of strings inserted in $\hat{S}_1$ and $\hat{S}_2$, resp.;
2. if $\Delta(\delta, m, m') > \mu/8$ then return `unknown`;
3. else, compute $d = \max_{f \in \mathcal{F}_{\mathsf{dist}}} |\hat{\mathbb{E}}_{S_1}[f] - \hat{\mathbb{E}}_{S_2}[f]|$;
4. if $d \leq \mu/2$ return `equal` else return `distinct`.

Lemma A.6. *Assume that* `Sketch` *is such that for every sample* $S = \{x_1, \ldots, x_m\}$, $\mathsf{dist}(S, \hat{S}) \leq \mu/8$, *where* $\hat{S}$ *is the sketched version of* S. *Then the* `Sketch` *and* `Test` *procedures above satisfy Assumptions 1.4, 1.5, 1.6, with both* N_{unknown} *and* N_{equal} *of the form* $\tilde{O}(1/\mu^2)$.

Proof. For assumption 1.4, note that the test never returns unknown when $\Delta(\delta, m, m') \leq \mu/8$, which is true when $m, m' \geq N_{\mathrm{unknown}} = \tilde{O}(1/\mu^2)$. For Assumption 1.5 follows similarly for $N_{\mathrm{equal}} = \tilde{O}(1/\mu^2)$ as well. For Assumption 1.6, suppose first that $\mu_* = 0$. By Proposition A.5, with probability $1 - \delta$

$$d = \mathsf{dist}(\hat{S}_1, \hat{S}_2) \leq \mathsf{dist}(\hat{S}_1, S_1) + \mathsf{dist}(S_1, S_2) + \mathsf{dist}(S_2, \hat{S}_2) \leq 2(\mu/8) + \mu/8 < \mu/2$$

so `Test` will return `equal`. If, on the other hand, $\mu_* \geq \mu$, using Proposition A.4 we can argue similarly that

$$d = \mathsf{dist}(\hat{S}_1, \hat{S}_2) \geq \mathsf{dist}(S_1, S_2) - \mathsf{dist}(\hat{S}_1, S_1) - \mathsf{dist}(S_2, \hat{S}_2) \geq (\mu - \mu/8) - 2\mu/8 > \mu/2,$$

and `Test` will return `distinct`. In both cases, the answer is correct with probability $1 - \delta$.

A.4 $\mathrm{L}_\infty^{\mathrm{p}}$ versus L_∞

It is easy to provide examples of distributions whose $\mathrm{L}_\infty^{\mathrm{p}}$ is much larger than L_∞ (Balle et al, 2012b). The next proposition shows that, up to a factor that depends on the alphabet size, $\mathrm{L}_\infty^{\mathrm{p}}$ is always an upper bound for L_∞.

Proposition A.7. *For any two distributions D and D' over $\Sigma^\star$ we have $\mathrm{L}_\infty(D, D') \leq (2|\Sigma| + 1)\mathrm{L}_\infty^{\mathrm{p}}(D, D')$.*

Proof. The inequality is obvious if $D = D'$. Thus, suppose $D \neq D'$ and let $x \in \Sigma^\star$ be such that $0 < \mathrm{L}_\infty(D, D') = |D(x) - D'(x)|$. Note that for any partition $x = u \cdot v$ we can write $D(x) = D^u(v)D(u\Sigma^\star)$. In particular, taking $v = \lambda$, the triangle inequality and ${D'}^x(\lambda) \leq 1$ yield the following:

$$\mathrm{L}_\infty(D, D') = |D(x) - D'(x)| \leq |D(x\Sigma^\star) - D'(x\Sigma^\star)| + D(x\Sigma^\star)|D^x(\lambda) - {D'}^x(\lambda)| \ .$$

Note that $|D(x\Sigma^\star) - D'(x\Sigma^\star)| \leq \mathrm{L}_\infty^{\mathrm{p}}(D, D')$. It remains to show that $D(x\Sigma^\star)|D^x(\lambda) - {D'}^x(\lambda)| \leq 2|\Sigma|\mathrm{L}_\infty^{\mathrm{p}}(D, D')$.

Observe the following, which is just a consequence of $D(\lambda) + \sum_\sigma D(\sigma\Sigma^\star) = 1$ for any distribution over $\Sigma^\star$:

$$D^x(\lambda) + \sum_\sigma D^x(\sigma\Sigma^\star) = {D'}^x(\lambda) + \sum_\sigma {D'}^x(\sigma\Sigma^\star) = 1 \ .$$

From these equations it is easy to show that there must exists a $\sigma \in \Sigma$ such that

$$|D^x(\lambda) - {D'}^x(\lambda)| \leq |\Sigma||D^x(\sigma\Sigma^\star) - {D'}^x(\sigma\Sigma^\star)| \ .$$

Therefore, using $D(x\Sigma^\star)D^x(\sigma\Sigma^\star) = D(x\sigma\Sigma^\star)$ we obtain

$$\begin{aligned} D(x\Sigma^\star)|D^x(\lambda) - {D'}^x(\lambda)| &\leq |\Sigma|D(x\Sigma^\star)|D^x(\sigma\Sigma^\star) - {D'}^x(\sigma\Sigma^\star)| \\ &\leq |\Sigma||D(x\sigma\Sigma^\star) - D'(x\sigma\Sigma^\star)| + |\Sigma|{D'}^x(\sigma\Sigma^\star)|D(x\Sigma^\star) - D'(x\Sigma^\star)| \\ &\leq 2|\Sigma|\mathrm{L}_\infty^{\mathrm{p}}(D, D') \ . \end{aligned}$$