Learning probability distributions generated by finite-state machines

Jorge Castro and Ricard Gavaldà

Abstract We review methods for inference of probability distributions generated by probabilistic automata and related models for sequence generation. We focus on methods that can be proved to learn in the inference in the limit and PAC formal models. The methods we review are state merging and state splitting methods for probabilistic deterministic automata and the recently developed spectral method for nondeterministic probabilistic automata. In both cases, we derive them from a high-level algorithm described in terms of the Hankel matrix of the distribution to be learned, given as an oracle, and then describing how to adapt that algorithm to account for the error introduced by a finite sample.

1 Introduction

Finite state machines in their many variants are acceptedly one of the most useful and used modeling formalisms for sequential processes. One of the reasons is their versatility: They may be deterministic, nondeterministic, or probabilistic, they may have observable or hidden states, and they may be acceptors, transducers, or generators. Additionally, many algorithmic problems (determinization, minimization, equivalence, set-theoretic or linear-algebraic operations, etc.) are often computationally feasible for these models.

Learning from samples or observations of their behavior is one of the most important associated problems, both theoretically and practically, since good methods for the task offer a competitive alternative to expensive modeling by experts. It has been intensely studied in various communities, and particularly in the grammatical inference one. Here we concentrate on learning probabilistic finite automata that generate probabilistic distributions over strings, where more precisely the task is to

Jorge Castro and Ricard Gavaldà
LARCA Research Group, Universitat Politècnica de Catalunya - BarcelonaTech, Barcelona, Spain, e-mail: {castro,gavalda}@lsi.upc.edu

come up with a device generating a similar distribution. We focus on two formal models of learning (the identification in the limit paradigm and the PAC learning models) rather than on heuristics, practical issues, and applications.

The main goal of the chapter is to survey known results and to connect the research on merging/splitting methods, mainly by the grammatical inference community, with the recently proposed methods collectively known as "spectral methods". The latter have emerged from several communities who present them in very different lights, for example as instances of the method of moments, of principal component analysis methods, of tensor-based subspace learning, etc. Our goal is to present it as emerging as a relatively natural extension of the work on automata induction that starts with Angluin's $L^\star$ method [3] for DFA and continues with the work by Beimel *et al.* [11] on learning multiplicity automata from queries. The Hankel matrix representation of functions and the generalization from probabilistic automata to weighted automata are essential ideas here.

The chapter is organized as follows. Section 2 presents the preliminaries on languages, probability, finite-state machines, and learning models.

Section 3 surveys the existing results on identification-in-the-limit and PAC learning. We discuss the evidence pointing to the hardness of PAC learning probabilistic automata when the only measures of complexity of the target machine are the number of states and alphabet size. We then indicate how PAC learning becomes feasible if other measures of complexity are taken into account.

Section 4 introduces the two main notions on which we base the rest of our exposition: the Hankel matrix of a function from strings to real values, and weighted automata, which generalize DFA and probabilistic automata. We present three results that link automata size and properties of the Hankel matrix: the well-known Myhil-Nerode theorem for DFA; a theorem originally due to Schützenberger and rediscovered several times linking weighted automata size and rank (in the linear algebraic sense) of the Hankel matrix; and a similar characterization of the size of deterministic weighted automata in terms of the Hankel matrix, which we have not seen stated so far - although it may be known.

Section 5 presents a high-level algorithm for learning probabilistic deterministic finite automata using the Hankel matrix which distills the reasoning behind many of the state merging/splitting methods described in the literature. We then present (variants of) the ALERGIA method [13, 14] and of the method by Clark and Thollard [16] building on this formulation; additionally, we describe them using a recently introduced notion of statistical query learning for distributions, which we believe makes for a clearer presentation.

Section 6, in analogy with the previous one, presents a high-level algorithm for learning weighted automata using the Hankel matrix. We then derive a formulation of the spectral method as a way of dealing with the effect of finite size samples in that high-level algorithm. We also discuss a few optimizations and extensions in recent woks in Section 6.3.

Finally, Section 7 mentions a few open questions for further research. In an Appendix we describe the well-known Baum-Welch heuristics for learning HMM.

Even though it does not fit into the formal models we discuss, the comparison with the other methods presented is interesting.

Since the literature in this topic is large, we have surely omitted many relevant references, either involuntarily or because they were out of our focus (rigorous results in formal models of learning). For example, recent works using Bayesian approaches to learning finite automata have not been covered because, while promising, seem still far from providing formal guarantees. We have also omitted most references to probability smoothing, a most essential ingredient in any implementation of such methods; information on smoothing for automata inference can be found in [16, 23, 36, 42].

The reader is referred to the surveys by Vidal *et al.* [45, 46], Dupont *et al.* [24], and the book by de la Higuera [18] for background on the models and results by the grammatical inference community. A good source of information about spectral learning of automata is the thesis of B. Balle [17] and the paper [9]. The 2012 and 2013 editions of the NIPS conference hosted workshops dedicated to spectral learning, including but more general than automata learning.

2 Preliminaries

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. We use λ to denote the empty string which satisfies $\lambda x = x\lambda = x$ for all $x \in \Sigma^\star$. The length of $x \in \Sigma^\star$ is denoted by $|x|$. The empty string is the only string with $|\lambda| = 0$. We denote by Σ^t the set of strings of length t. A *prefix* of a string $x \in \Sigma^\star$ is a string u such that there exists another string v such that $x = uv$. String v is a *suffix* of x. Hence e.g. $u\Sigma^\star$ is the set of all strings having u as a prefix. A subset X of $\Sigma^\star$ is prefix-free whenever for all $x \in X$, if y is a prefix of x and $y \in X$ then $y = x$.

Several measures of divergence between probability distributions are considered. Let D^1 and D^2 be distributions over $\Sigma^\star$. The *Kullback–Leibler (*KL*) divergence* or *relative entropy* is defined as

$$\mathrm{KL}(D^1 \| D^2) = \sum_{x \in \Sigma^\star} D^1(x) \log \frac{D^1(x)}{D^2(x)} \ ,$$

where the logarithm is taken to base 2 and by definition $\log(0/0) = 0$.

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 $\mathrm{L}_\infty(D^1, D^2) = \max_{x\in\Sigma^\star} |D^1(x) - D^2(x)|$. While KL is neither symmetric nor satisfies the triangle inequality, measures L_1 and L_∞ are true distances. We recall Pinsker's inequality, $\mathrm{L}_1 \leq \sqrt{2\mathrm{KL}}$, bounding the total variation distance in terms of the relative entropy. Thus, as L_1 obviously upperbounds L_∞, the relative entropy is, up to a factor, the most sensitive divergence measure among the ones considered here to distribution perturbations, and convergence criteria based on the KL value are most demanding.

Frequently, machine descriptions are provided in terms of vectors and matrices of real numbers and computations are defined by matrix products. We use square brackets to denote an specific component of a vector or matrix. For instance, component j of vector α is $\alpha[j]$. Row x of a matrix T is denoted by $T[x,:]$ and column y is $T[:,y]$. Vectors are always assumed to be columns. If α is a vector, a row vector α^T is obtained by transposing α.

2.1 Learning Distributions in the PAC Framework

We introduce the PAC model for learning distributions, an adaptation of Valiant's PAC model for concept (function) learning [44]. Let $\mathscr{D}$ be a class of distributions over some fixed set X. Assume $\mathscr{D}$ is equipped with some measure of *complexity* assigning a positive number $|D|$ to any $D \in \mathscr{D}$. We say that an algorithm *A PAC learns* a class of distributions $\mathscr{D}$ using $S(\cdot)$ examples and time $T(\cdot)$ if, for all $0 < \varepsilon, \delta < 1$ and $D \in \mathscr{D}$, 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 $\widehat{D}$ such that $L_1(D, \widehat{D}) \leq \varepsilon$. 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.

Sometimes we will consider the relative entropy instead of the variation distance as a measure of divergence, which creates a different learning problem. Which specific measure we are considering in each PAC result will be clear from the context. Although the majority of PAC statements in the chapter are provided for L_1 all of them can be also shown for the KL divergence measure, at the cost of longer proofs. As the main proof ideas are the same for both measures, we have chosen mainly L_1 versions for simplicity.

Concerning the measure of complexity of distributions, standard practice is to fix a formalism for representing distributions, such as finite-state machines, and then consider the smallest, in some sense, representation of a given distribution in that formalism (usually exactly, but possibly approximately). In the case of finite-state machines, a seemingly reasonable measure of "size" is the number states times the number of alphabet symbols, as that roughly measures the size of the transition table, hence of the automaton description. A first objection is that transition tables contain numbers, so this notion does not match well the more customary notion of bit-length complexity. One could refine the measure by assuming rational probabilities (who have reasonable bit-length complexity measures) or by truncating the probabilities to a number of digits that does not distort the probability distribution by more than about ε. We will see however that number of states and symbols alone do not fully determine the learning complexity of probabilistic finite-state machines, and that the bit-length of the probabilities seems irrelevant. This will motivate the (non-standard) introduction of further complexity parameters, some defined in terms of the finite-state machine, and some in terms of the distribution itself.

2.2 Identification in the Limit Paradigm

An alternative framework for learning distributions is the identification in the limit paradigm, originally introduced by Gold [27] for the setting of language learning. Later, the model was adapted in [19] to the learning distributions scenario. Basically, the model demands that with probability 1, given an infinite sample from the target, the learning algorithm with input the first m examples of the sample exactly identifies the target distribution when m is large enough. We consider here a slightly weaker definition, mainly because we consider state machines defined on real numbers instead of rational ones as in [19].

As before, let $\mathscr{D}$ be a class of distributions. We say that an algorithm A *identifies in the limit* a class of distributions $\mathscr{D}$ if for all $0 < \varepsilon < 1$ and $D \in \mathscr{D}$, given an infinite sample $x_1, x_2, \ldots$ of D

1. With probability 1, there exists $m_0(\varepsilon)$ such that for all $m \geq m_0(\varepsilon)$ algorithm A with input $x_1, \ldots, x_m$ outputs a hypothesis $\widehat{D}$ such that $\mathrm{L}_1(D, \widehat{D}) \leq \varepsilon$.
2. A runs in polynomial time in its input size.

It is easy to check that any PAC learning algorithm also achieves identification in the limit: Assume that identification in the limit does not occur. We have that with probability $\delta > 0$, there are arbitrarily large values m such that A with input $x_1, \ldots x_m$ outputs a hypothesis $\widehat{D}$ such that $\mathrm{L}_1(D, \widehat{D}) > \varepsilon$. This implies that PAC learning does not hold.

2.3 Probabilistic Automata

A *probabilistic finite automaton* (pfa) of size n is a tuple $\langle Q, \Sigma, \tau, \alpha_0, \alpha_\infty \rangle$ where Q is a set of n states, Σ is a finite alphabet, $\tau : Q \times \Sigma \times Q \to [0,1]$ is the transition probability function and α_0 and α_∞ are respectively $[0,1]^n$ vectors of initial and final probabilities. We require that $\sum_{i \in Q} \alpha_0[i] = 1$ and, for every state i, $\alpha_\infty[i] + \sum_{a \in \Sigma, j \in Q} \tau(i,a,j) = 1$. States i such that $\alpha_0[i] > 0$ ($\alpha_\infty[i] > 0$) are initial (final) sates. To each pfa D corresponds an underlying non-deterministic finite automaton (nfa) called the *support* of D, defined as $\langle Q, \Sigma, \delta, Q_I, Q_F \rangle$ where Q_I and Q_F are respectively the set of initial and final states, and transition function δ is defined as $\delta(i,a) = \{ j | \tau(i,a,j) > 0 \}$.

The transition probability function of a pfa D can be extended to strings in $\Sigma^\star$. Given $x \in \Sigma^\star$, we define $\tau(i, xa, j) = \sum_{k \in Q} \tau(i,x,k)\tau(k,a,j)$ and $\tau(i, \lambda, j) = 1$ if $i = j$ and 0 otherwise. A probability mass can be assigned to words in Σ defining:

$$D(x) = \sum_{i \in Q_I, j \in Q_F} \alpha_0[i] \tau(i,x,j) \alpha_\infty[j].$$

Defining transition matrices T_a for each $a \in \Sigma$ as $T_a[i,j] = \tau(i,a,j)$ and provided $x = x_1 \ldots x_m$ where $x_t \in \Sigma$ for $t = 1 \ldots m$, a more convenient expression in terms of matrix products for the probability mass of x is

$$D(x) = \alpha_0^T T_{x_1} \cdots T_{x_m} \alpha_\infty$$

that we write in short by $\alpha_0^T T_x \alpha_\infty$. Note that transition matrices are square $|Q| \times |Q|$ and define the transition function. So, an alternative tuple description for a pfa D in terms of transition matrices is $\langle \Sigma, \{T_a\}_{a\in\Sigma}, \alpha_0, \alpha_\infty \rangle$.

The probability mass induced by D defines a semi-distribution in $\Sigma^\star$: it satisfies $0 \leq \sum_{x\in\Sigma^\star} D(x) \leq 1$. Whenever from every state i there is non-zero probability of reaching a final state, D defines a true probability distribution, i.e. $\sum_{x\in\Sigma^\star} D(x) = 1$. In the rest of the chapter we will consider only pfas defining true probability functions. More generally, every state i of a pfa D defines a probability distribution $D_i(x) = \gamma_i^T T_x \alpha_\infty$ where γ_i is the i-indicator vector $\gamma_i[j] = 1$ if $i = j$ and 0 otherwise. We have $D(x) = \sum_{i\in Q} \alpha_0[i] D_i[x]$.

2.4 Probabilistic Deterministic Automata and Distinguishability

A *probabilistic deterministic finite automaton* (pdfa for short) is a pfa whose support is a deterministic finite automaton (dfa). We note that for a pdfa we can assume without loss of generality — in short, w.l.o.g.— that the initial probability vector α_0^T is (1,0,0,…,0) and each row of each transition matrix T_a has at most one non-zero component.

It is known [24] that pfa cannot exactly compute all probability distributions over $\Sigma^\star$, and that there are pfa computing distributions that cannot be exactly computed by any pdfa. That is, pfa are strictly more expressive than pdfa.

The following parameter will be useful to measure the complexity of learning a particular pdfa. It appears in [16] as defined here, although a similar idea is implicit in [36].

Definition 1. We say that distributions D^1 and D^2 are *μ-distinguishable* if $\mu \leq \mathrm{L}_\infty(D^1, D^2)$. A pdfa D is μ-distinguishable when for each pair of states i and j their corresponding distributions D_i and D_j are μ-distinguishable. The *distinguishability* of a pdfa is defined as the supremum over all μ for which the pdfa is μ-distinguishable.

The distinguishability parameter can sometimes be exponentially small in the number of states. There exists reasonable evidence suggesting that polynomially learnability of pdfa in the number of states alone may not be achievable [30, 41]. However, PAC results have been obtained [16] when the inverse of distinguishability of the target is also considered, as we will see. We will also discuss parameters that play similar roles in pfa learning.

2.5 Hidden Markov Models

Hidden Markov models HMM are representations of choice for Markovian stochastic processes whose latent states are non-observable but their effects are. Visible effects are the observations arising from the process.

There are several formal definitions for HMM in the literature, but all of them are equivalent up to a polynomially transformation [24]. Often, observations in HMM are associated to states rather than transitions but for simplicity, we choose a definition closest to that of pfa. An HMM is a tuple $\langle Q, \Sigma, \tau, \alpha_0 \rangle$ where the four parameters have the same meaning that in the pfa definition. For any state i, it always holds $\sum_{a\in\Sigma, j\in Q} \tau(i,a,j) = 1$. Thus, one can see a HMM as a pfa having no stopping probabilities, i.e. as an infinite duration process. Transition probability matrices $\{T_a\}_{a\in\Sigma}$ are defined as in the pfa case, i. e. $T_a[i,j] = \tau(i,a,j)$.

An HMM defines, for each integer $t \geq 0$, a probability distribution on Σ^t. The probability mass assigned to string $x = x_1 \ldots x_t$ is $\alpha_0^T T_{x_1} \cdots T_{x_t} \mathbf{1}$ where $\mathbf{1}$ is the all-ones vector. This is the probability that the machine generates an infinite string of observations whose length-t prefix is x.

2.6 Weighted Automata

Weighted automata (wa) are the most general class of finite state machines we consider. They encompass all the models we have introduced before. A weighted automaton T over Σ with n states is a tuple $\langle \Sigma, \{T_a\}_{a\in\Sigma}, \alpha_0, \alpha_\infty \rangle$ where $T_a \in \mathbb{R}^{n\times n}$ and α_0 and α_∞ are vectors in $\mathbb{R}^n$. To each weighted automaton T corresponds a real-valued function defined on $\Sigma^\star$. Given $x = x_1 \ldots x_m \in \Sigma^\star$:

$$f_T(x) = \alpha_0^T T_{x_1} \cdots T_{x_m} \alpha_\infty = \alpha_0^T T_x \alpha_\infty.$$

We note that wa can be defined over semirings other than the real numbers, but whether wa are learnable (in any particular learning model) strongly depends on the semiring. For example, the high-level algorithms for real-valued wa that we present in Sections 5.1 and 6.1 work in fact on any field, including finite fields. On the other hand, wa over the boolean semiring $(\vee, \wedge, 0, 1)$ are at least as expressive as DNF formulas, whose learnability in the PAC [44] or query [3] concept learning models are major open problems in computational learning theory.

A weighted automaton is deterministic (in short, a dwa) when rows of its transition matrices have at most one nonzero value and its initial vector α_0^T is $(1,0,\ldots,0)$.

3 A Panoramic View of Known Results

Around 1970, Baum and Welch described a practical heuristic for learning Hidden Markov Models generating infinite sequences; their properties were studied in [10], where it was shown to perform hill-climbing with respect to maximum likelihood. As such, it cannot be shown to learn in either of the two models we have presented (identification in the limit and PAC) but, even today, it is possibly the most used method in practice. Although it is described in many sources, we state it in our formalism in the Appendix for completeness and for comparison to the other methods we review.

Rudich [37] was, to our knowledge, the first to prove that Hidden Markov Models generating infinite sequences are identifiable in the limit. His method seems far from the PAC criterion in the sense that not only there are no bounds on the the error on a finite sample, but even processing each observation involves cycling over all exponentially many possible state structures, hence is very inefficient.

A decade later, Carrasco and Oncina [13, 14] described ALERGIA, a practical method for learning pdfa generating distributions on $\Sigma^\star$. We will describe a slight variant of ALERGIA which we name the Red-Blue algorithm in Section 5.2.1. ALERGIA has been highly influential in the grammatical inference community. Some of the (many) works that build on the ideas of ALERGIA to learn pdfa are Section [20, 43, 31]. All these algorithms are efficient in the sense that they work in time polynomial in the size of the sample, although no bounds are given on the number of samples required to reach convergence up to some ε. Apparently independently, [39] proposes the Causal State Splitting Reconstruction algorithm (CSSR) for inferring the structure of *deterministic* Hidden Markov Models. Although the underlying ideas are similar, their work differs in that the input consists of a single biinfinite string, not a sample of finite strings.

Still in the paradigm of identification in the limit, we mention the work by Denis and Esposito [21] who learn Residual Automata, a class strictly more powerful than pdfa but less than general pfa.

If we move to the PAC paradigm, the first result to mention on learning pfa is that of Abe and Warmuth [1], who show that they are PAC learnable in polynomial space. The method essentially iterates over all (exponentially many) state structures with a hypothesized number of states n, fitting probabilities in each according to the sample, and producing the one which assigns maximum likelihood to the sample. It can be shown that, for a sample of size polynomial in n, $|\Sigma|$, $1/\varepsilon$, and $\log(1/\delta)$, this hypothesis satisfies the PAC criterion of learning. The difficulty of learning pfa is thus computation, not information.

The same paper [1] shows that it is NP-complete to PAC learn pfa when the size of the alphabet Σ is unbounded, which in effect implies that all methods will require time exponential in $|\Sigma|$. Kearns *et al.* [30] showed that the problem may be hard even for 2-letter alphabets. They show that learning pdfa generalizes the *noisy parity learning problem,* which has received a fair amount of attention and for which all algorithms known so far take exponential time. Furthermore, only a small set of simple probabilities are required in the reduction (say, $\{i/8 \mid i = 0 \dots 8\}$). Terwijn

[41] showed that indeed the problems of learning HMM and acyclic pdfa are hard under plausible cryptographic assumptions.

The previous results paint a rather discouraging panorama with respect to PAC learning of pfa and even pdfa. But note that, critically, these negative results all imply the hardness when the complexity of the target distribution (hence, the polynomiality of the PAC model) is defined to be number of states times alphabet size for the smallest machine generating it. A line of research starting in [36] aims at giving positive results by using other measures of complexity, in particular taking other parameters of the distribution that may not even be directly observable from a generating machine. Alternatively, one can view this line of research as designing sensible algorithms for learning pdfa/pfa, then analyzing their running time in search of the relevant features of the target distribution, instead of deciding, *a priori*, what the bounds on the running time should look like.

To be precise, Ron *et al.* [36] gave an algorithm for learning acyclic pdfa that can be shown to PAC learn with respect to the KL-divergence in time and sample size polynomial in the inverse of the distinguishability of the target machine, besides the usual parameters.

Later, Clark and Thollard [16] extended the result to cyclic automata; they introduce an additional dependence on the expected length of the strings in the distribution, L. We will describe the Clark-Thollard algorithm in detail in Section 5.2.2, under the name of Safe-Candidate algorithm.

Extensions and variants of the Clark-Thollard algorithm include the following. Palmer and Goldberg [35] show that the dependence on L can be removed if learning only w.r.t. the L_1 distance is required. In another direction, Guttman *et al.* [28] show that PAC learning is still possible in terms of L_2-distinguishability, which is more restrictive that the L_∞-distinguishability we use here. The variations presented in [26] and [15], while retaining the PAC guarantees, aim at being more efficient in practice. In [5] the algorithm is extended to machines whose transitions have random durations, determined by associated probability distribution that must be learned too. In [7], the algorithm is transported to the so-called *data stream* paradigm, where data (strings) arrive in sequence and the algorithm is required to use sublinear memory and low time per item.

In substantial breakthroughs, Mossel and Roch [33], Denis *et al.* [22], Hsu *et al.* [29], and Bailly *et al.* [4] gave algorithms having formal proofs of PAC learning the full class of pfa. The sample size and running times of the algorithms depend polynomially in the inverse of some quantity of a spectral flavor associated to the Hankel matrix of the target distribution. This is, for example, the determinant in [33] and the inverse of its n-th singular value in [29]. Denis *et al.* [22] do not explicitly state a PAC learning result, but in our opinion they refrain from doing so only because they lack a proper name for the feature of the distribution that determines their algorithms' running time.

4 The Hankel Matrix Approach

In this section we review the notion of Hankel matrix and weighted automata, which will be the main tools for describing generic methods for learning pdfa and pfa.

The *Hankel matrix* [25] H_f of a function $f : \Sigma^\star \to \mathbb{R}$ is a matrix representation of f in $\mathbb{R}^{\Sigma^\star \times \Sigma^\star}$ defined as:

$$H_f[x,y] = f(xy), \forall x,y \in \Sigma^\star.$$

Despite the fact that the Hankel matrix H_f is a very redundant representation —value $f(z)$ is represented $|z|+1$ times—, the computability of f by finite state machines is determined by algebraic properties of H_f.

Assume function f is a language, i.e. $f : \Sigma^\star \to \{0,1\}$. For each $x \in \Sigma$ we consider the subset $x^{-1}f = \{y \in \Sigma^\star | f(xy) = 1\}$. The well-known Myhill-Nerode theorem claims that f is a regular language if and only if there are only a finite number of different sets $x^{-1}f$ when x varies on $\Sigma^\star$. The number of different sets $x^{-1}f$ also determines the minimum dfa size required for computing f. Thus, translating this theorem in terms of the Hankel matrix, we have:

Theorem 1. *Function $f : \Sigma^\star \to \{0,1\}$ is regular if and only if the Hankel matrix H_f has a finite number n of different rows. Moreover, the minimum size of a dfa computing f is n.*

A similar characterization can be shown for functions on $\Sigma^\star$. Given a function $f : \Sigma^\star \to \mathbb{R}$, an equivalence relation on $\Sigma^\star$ can be defined as follows. Words x and y are related —written $x \sim_f y$— iff their corresponding rows in the Hankel matrix H_f are the same up to an nonzero scalar factor, i.e. $H_f[x,:] = c_{x,y} H_f[y,:]$ for some scalar $c_{x,y} \neq 0$. Let $\Sigma^\star / \sim_f$ be the quotient set. When matrix H_f has an identically zero row, this set has a class —the zero class— representing all zero rows. We propose the following theorem that characterizes the computability of f by deterministic weighted automata in terms of the cardinality of the quotient set.

Theorem 2. *Let $f : \Sigma^\star \to \mathbb{R}$ be any function. There is a dwa computing f if and only if the cardinality of $\Sigma^\star / \sim_f$ is finite. Moreover, the minimum size of a dwa computing f is the number n of nonzero classes in $\Sigma^\star / \sim_f$.*

Proof. We shorten $\sim_f$ to $\sim$ in this proof. We prove first the if part. Assume that words x and y are $\sim$-related and let $c_{x,y}$ be a scalar such that $H_f[x,:] = c_{x,y} H_f[y,:]$. It is easy to see that

1. the value of $c_{x,y}$ is uniquely determined except when x and y are in the zero class, in which case we define $c_{x,y}$ as 0, and
2. for any strings x_1, x_2, x_3, u, v, if $x_1 u \sim x_2$ and $x_2 v \sim x_3$, then $x_1 uv \sim x_3$ and $c_{x_1 uv, x_3} = c_{x_1 u, x_2} c_{x_2 v, x_3}$.

The last property uses the redundancy of the Hankel matrix, namely, for every z,

$$\begin{aligned} H_f[x_1uv,z] &= f(x_1u\cdot vz) = c_{x_1u,x_2}f(x_2\cdot vz) = c_{x_1u,x_2}f(x_2v\cdot z) \\ &= c_{x_1u,x_2}c_{x_2v,x_3}f(x_3\cdot z) = c_{x_1u,x_2}c_{x_2v,x_3}H_f[x_3,z] \end{aligned}$$

from where, by definition of $\sim$, it is $x_1uv \sim x_3$ with scalar $c_{x_1uv,x_3} = c_{x_1u,x_2}c_{x_2v,x_3}$.

Let $x^1,\ldots x^n$ be representatives of the n nonzero classes of $\Sigma^\star/\sim$. W.l.o.g we assume $n \geq 1$ —otherwise f is the null function — and $x^1 = \lambda$. We define for each $a \in \Sigma$ the transition matrix $T_a \in \mathbb{R}^{n\times n}$ as $T_a[i,j] = c_{x^ia,x^j}$ if $x^ia \sim x^j$ and 0 otherwise. It is immediate to see that every row of T_a has at most one nonzero value.

Let γ_i be the i-indicator vector, with $\gamma_i[j] = 1$ if $i = j$ and 0 otherwise. We show by induction on $|w|$ that for any word w we have

$$\gamma_i^T T_w = \begin{cases} \mathbf{0} & \text{if } x^iw \text{ is in the zero class} \\ c_{x^iw,x^j}\,\gamma_j^T & \text{for the } j \text{ such that } x^iw \sim x^j \text{ otherwise.} \end{cases}$$

The equality is obvious for $|w| = 0$, since T_w is the identity. Let $w = au$ for some alphabet symbol a. We consider two cases. In the first one we assume x^ia belongs to a nonzero class represented by x^k. Observe that by definition of T_a we have $\gamma_i^T T_a = c_{x^ia,x^k}\gamma_k^T$. Assuming x^ku belongs to a nonzero class represented by x^j, we have

$$\gamma_i^T T_{au} = \gamma_i^T T_a T_u = c_{x^ia,x^k}\gamma_k^T T_u = c_{x^ia,x^k}\,c_{x^ku,x^j}\gamma_j^T\,.$$

where the last equality follows from the induction hypothesis. By property (2) above, $x^iau \sim x^j$ and $c_{x^iau,x^j} = c_{x^ia,x^k}\,c_{x^ku,x^j}$ as required. If x^ku is in the zero class, by transitivity x^iau also belongs to the zero class. By the induction hypothesis

$$\gamma_i^T T_{au} = \gamma_i^T T_a T_u = c_{x^ia,x^k}\gamma_k^T T_u = c_{x^ia,x^k}\,\mathbf{0} = \mathbf{0}.$$

as required. This concludes the inductive claim in the first case. For the second one, note that $\gamma_i^T T_a$ is the zero vector and the claim is obvious.

Consider the dwa T with matrices T_a defined above, initial vector $\alpha_0 = \gamma_1$ and $\alpha_\infty^T = (f(x^1),\ldots,f(x^n))$. Then for any string w, if $w \sim x^k$,

$$T(w) = \alpha_0^T\,T_w\,\alpha_\infty = \gamma_1^T\,T_w\,\alpha_\infty = c_{\lambda w,x^k}\,\gamma_k^T\,\alpha_\infty = c_{w,x^k}\,f(x^k),$$

and then we have

$$f(w) = H_f[w,\lambda] = c_{w,x^k}H_f[x^k,\lambda] = c_{w,x^k}f(x^k) = T(w).$$

We consider now the only if part. Let $\langle\{T_a\}_{a\in\Sigma},\alpha_0,\alpha_\infty\rangle$ be a dwa of size n computing a function f. We say that a matrix is deterministic when all rows have at most one nonzero value. We note that the product of deterministic matrices is also deterministic. So, for any string x, matrix T_x is deterministic and $\alpha_0^T\,T_x$ must be either $c_x\gamma_j^T$ for some nonzero scalar c_x and integer $j \in \{1\ldots n\}$ or the zero vector. Let h be an integer function on $\Sigma^\star$ such that $h(x) = j$ when $\alpha_0^T\,T_x = c_x\gamma_j^T$ for some nonzero scalar c_x and $h(x) = 0$ when $\alpha_0\,T_x$ is the zero vector. Note that the cardinality of the range of h is at most $n+1$. We show that if $h(x) = h(y)$ then rows x and y

of the Hankel matrix H_f are the same up to a nonzero factor. The result is obvious when $h(x) = h(y) = 0$. Assume $h(x) = h(y) = j > 0$. For any string z,

$$\begin{aligned} H_f[x,z] &= \alpha_0^T T_x T_z \alpha_\infty = c_x \gamma_j^T T_z \alpha_\infty = (c_x/c_y) c_y \gamma_j^T T_z \alpha_\infty \\ &= (c_x/c_y) \alpha_0^T T_y T_z \alpha_\infty = (c_x/c_y) H_f[y,z]. \end{aligned}$$

Thus, up to nonzero scalar factors, the Hankel matrix of f has at most n nonzero rows. □

As an example, consider $\Sigma = \{a\}$ and function f defined on $\Sigma^\star$ having values $f(\lambda) = 0$ and $f(a^k) = 1/2^k$ when $k \geq 1$. It is easy to check that, up to nonzero scalar factors, the Hankel matrix H_f has only two different rows, $(0, 1/2, 1/4, \ldots)$ and $(1/2, 1/4, 1/8, \ldots)$ corresponding, respectively, to words λ and a. Following the proof of Theorem 2 the dwa defined by $\alpha_0^T = (1,0)$, $\alpha_\infty^T = (0, 1/2)$ and matrix T_a having rows $(0,1)$ and $(0,1/2)$ computes function f.

Finally, the full class of weighted automata can be also characterized by an algebraic parameter, in this case the number of linearly independent rows —i.e. the rank— of the Hankel matrix. The characterization of weighted automata in terms of the rank has been shown by several authors [11, 12, 25, 38]. We follow the exposition in [11].

Theorem 3. *Let $f : \Sigma^\star \to \mathbb{R}$ be a function with Hankel matrix H_f. Function f can be computed by a weighted automaton if and only if the rank n of H_f is finite. Moreover, the minimum size of a wa computing f is n.*

Proof. Only if. Let $\langle \{T_a\}_{a \in \Sigma}, \alpha_0, \alpha_\infty \rangle$ be a weighted automaton of size n computing f. We define backward an forward matrices $B \in \mathbb{R}^{\infty \times n}$ and $F \in \mathbb{R}^{n \times \infty}$ with rows, respectively columns, indexed by words in $\Sigma^\star$ as:

$$\begin{aligned} B[x,:] &= \alpha_0^T T_x, \\ F[:,y] &= T_y \alpha_\infty. \end{aligned}$$

From the fact that $H_f[x,y] = \alpha_0 T_x T_y \alpha_\infty = B[x,:]F[:y]$ we conclude that $H_f = BF$ and therefore $\text{rank}(H_f) \leq \text{rank}(F) \leq n$.

For the if part let $x^1, \ldots, x^n$ be words in $\Sigma^\star$ indexing n linearly independent rows of H_f. W.l.o.g. we can assume $x^1 = \lambda$, as otherwise f is the null function. We consider the weighted automaton of size n defined by vectors $\alpha_0^T = (1, 0, \ldots, 0)$, $\alpha_\infty^T = (f(x^1), \ldots, f(x^n))$ and transition matrices $T_a \in \mathbb{R}^{n \times n}$ for $a \in \Sigma$ with values $T_a[i,j] = a_j^i$ satisfying:

$$H_f[x^i a, :] = \sum_j a_j^i H_f[x^j, :].$$

These values exist because by hypothesis, H_f is a rank n matrix and strings x^j for $j = 1, \ldots, n$ are indexes of n linear independent rows.

We show that $f(x^i v) = \gamma_i^T T_v \alpha_\infty$ for $i = 1, \ldots, n$, where γ_i is the i-indicator vector. Once the proof is completed, the if part of the theorem follows from considering the equality for $i = 1$. We induct on the length of v. When $|v| = 0$ the equality is immediate. Assume $v = au$ for some $a \in \Sigma$. We have:

$$f(x^i v) = f(x^i au) = H_f[x^i a, u] = \sum_j a^i_j H_f[x^j, u] = \sum_j a^i_j f(x^j u).$$

By induction hypothesis, $f(x^j u) = \gamma_j^T T_u \alpha_\infty$. Thus,

$$f(x^i v) = \sum_j a^i_j(\gamma_j^T T_u \alpha_\infty) = (\sum_j a^i_j \gamma_j^T) T_u \alpha_\infty = (\gamma_i^T T_a) T_u \alpha_\infty = \gamma_i^T T_v \alpha_\infty. \quad \square$$

We remark that Theorems 2 and 3 work on any field, not just the real numbers. We would like to see a derivation of Theorem 2 as a consequence of Theorem 3 plus the assumption of determinism, instead of having to reprove it from scratch.

5 Learning PDFA

5.1 An Oracle Algorithm for Learning DWA

In this subsection we present a high-level algorithm for learning Deterministic Weighted Automaton assuming that we have *oracle* access to the function $f : \Sigma^\star \to \mathbb{R}$. This model differs from the one considered so far for probabilistic automata in that we receive the exact probabilities of strings instead of estimating them from a sample, and that learning must be exact and not approximate. Furthermore, this model allows discussing functions other than probability distributions, for which estimating function values from samples does not even make sense, and even functions mapping strings to finite fields. At the end of the subsection, we will discuss how to turn this high-level algorithm into a sample-based one for the case of pfa, and give specific (and more efficient) implementations in the following subsections.

The high-level algorithm is given in Figure 1. To clarify the implementation of line 3, initially set $X' = \{\lambda\}$ and repeatedly pick any $z \in X - X'$ such that $H[z,:]$ differs from all $H[x,:]$ with $x \in X'$, while such a z exists, and add it to X'. It is clear that the final X' is as desired. It is also clear that, excluding the time required to answer oracle queries, the algorithm runs polynomial time in $|\Sigma|$ and the sum of the lengths of strings in X and Y.

Following the proof of Theorem 2, it is now easy to prove that the algorithm is correct if it is given "right" sets X and Y.

Theorem 4. *If the cardinality of X' is at least the number of nonzero rows of H_f up to nonzero scalar factors, then the dwa generated by the algorithm computes f.*

We do not discuss here the question of how to find suitable sets X, Y that provide a large enough X' in the oracle model. We instead move to the problem of translating this algorithm to the setting in which f is a probability distribution and all the information about f is given by some sample S, a multiset of strings.

1. Choose any two finite sets $X, Y \subseteq \Sigma^\star$ with $\lambda \in X \cap Y$, in some way not specified by the algorithm;
2. Build the submatrix $H = H_f[X \cup X\Sigma, Y]$ of H_f by asking oracle queries on f;
3. Find a minimal $X' \subseteq X$ such that $\lambda \in X'$ and for every $z \in X$, $H[z,:]$ is a multiple of $H[x,:]$ for $x \in X'$;
4. Build a dwa from H, X, Y, X' as follows. Say $X' = \{x^1 = \lambda, x^2, \ldots, x^n\}$, then

$$\begin{aligned} \alpha_0^T &= (1,0,\ldots,0) \\ \alpha_\infty^T &= (f(x^1),\ldots,f(x^n)) \\ T_a[i,j] &= v \text{ if } H[x^i a,:] = vH[x^j,:], \text{ and 0 otherwise;} \end{aligned}$$

Fig. 1 Learning dwa with an oracle

The obvious choice is to set $X = \mathit{prefixes}(S)$, $Y = \mathit{suffixes}(S)$, and then create an approximation $\hat{H}$ of H by $\hat{H}[x,y]$ = empirical probability of xy in S. Note that although X and Y are sets, the approximation $\hat{H}[x,y]$ is still computed taking into account the possible repetitions of xy in the multiset S. Now, the question "is $H[x,:]$ a multiple of $H[z,:]$?" becomes a statistical question on $\hat{H}$ for a finite S. If the answer is "no", it will become clear as S grows in size, but if the answer is "yes" no finite amount of information will prove it conclusively. Let the algorithm use a statistical test that will return the right yes/no answer with probability tending to 1 as the sample size tends to infinity. (Different statistical tests have been used in the literature for specific instantiations of this general algorithm). For sufficiently large samples, representatives of all equivalence classes of rows will appear in the sample and furthermore the test will correctly answer all the finitely many row equivalences. It is now easy to argue that the algorithm above endowed with such a statistical test will identify the target probability distribution in the limit.

In the PAC paradigm, however, tests should be sufficiently reliable for samples of polynomial size *and* be computationally efficient. Unfortunately, such tests may not exist if the target machine complexity is defined as number of states times size of the alphabet. This is precisely what the negative results in [1, 30, 41] formalize. It may become possible if one restricts the class of target machines in some way, such as requiring a minimum L_∞-distinguishability among states; alternatively, letting the inverse of the distinguishability be also a parameter of complexity. This will be the case for e.g. the method by Clark and Thollard [16].

Most methods for learning pdfa described in the literature conceptually rely on the dwa construction described above, with a number of refinements towards practicality. For example, they tend to identify states not in one shot from the matrix H, but instead by iteratively splitting or merging simpler states. This has the advantage that once several strings (rows) have been merged, their statistical evidence accumulates, allowing for sounder decisions from then on. Additionally, they are able to come up with a pdfa when it is guaranteed that the target is a pdfa - while our construction here may return dwa which are not pdfa. Finally, they differ in the precise statistical tests for state identification and in the smoothing policies to account for unseen mass, both of which may make a large difference in practice.

5.2 State-merging learning algorithms

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.

We show below two adaptations of the state-merging strategy to pdfa learning, the Red-Blue [14] and the Safe-Candidate [16] algorithms. Both infer gradually elements of the target graph, i.e. states and transitions, and estimate their corresponding probabilities. The Red-Blue algorithm starts by inferring a prefix tree acceptor and then merges equivalent states. The Safe-Candidate algorithm alternates between inferring and merging new elements. While the first one achieves learning in the limit, the second one has PAC guarantees whenever a polynomial dependence on an additional parameter —the target distinguishability— is accepted.

The usual description of these algorithms considers the learning from examples paradigm. In contrast, following [6, 17], the exposition below assumes a query framework: algorithms get information on the target by asking queries of possibly different kinds instead of analyzing a sample. Proceeding in this way, many details concerning probability approximation issues can be abstracted, and a more compact, clear and elegant presentation can be provided.

The query model we use allows two type of queries, so-called respectively *statistical* and L_∞-*queries*, that can be solved, with high probability, by drawing a sample of the target distribution. Thus, learning algorithms in this query setting can be easily moved to standard learning from examples.

Given a distribution D on $\Sigma^\star$ a *statistical query for D* is a tuple (X,α) where X is an efficiently decidable subset of $\Sigma^\star$ and $0<\alpha<1$ is some *tolerance* parameter. The query $\mathsf{SQ}^D(X,\alpha)$ returns an α-approximation $\hat{p}$ of $D(X)$ such that $|\hat{p}-D(X)|<\alpha$.

Let X be a prefix-free subset of $\Sigma^\star$. Function $D^X(y)=\frac{D(Xy)}{D(X\Sigma^\star)}$ defines a probability distribution on $\Sigma^\star$ that corresponds to the distribution over suffixes conditioned on having a prefix in X. An L_∞-query is a tuple (X,Y,α,β) where X and Y are efficiently decidable, disjoint and prefix-free subsets of $\Sigma^\star$, and $0<\alpha,\beta<1$ are, respectively, the *tolerance* and *threshold* of the query. Query $\mathsf{DIFF}^D_\infty(X,Y,\alpha,\beta)$ is answered by the oracle according to the following rules:

1. If either $D(X\Sigma^\star)<\beta$ or $D(Y\Sigma^\star)<\beta$ it answers '?'.
2. If both $D(X\Sigma^\star)>3\beta$ and $D(Y\Sigma^\star)>3\beta$, it answers with some α-approximation $\hat{\mu}$ of $\mathrm{L}_\infty(D^X,D^Y)$.
3. Otherwise, the oracle may either answer '?' or give an α-approximation of $\mathrm{L}_\infty(D^X,D^Y)$.

Both query types can be easily simulated with high probability with a sample of distribution D. The following result holds.

Proposition 1. *For any probability distribution D, a statistical query* $\mathsf{SQ}^D(X,\alpha)$ *can be simulated using* $O(\alpha^{-2}\log(1/\delta))$ *examples from D. A* $\mathsf{DIFF}_\infty^D(X,Y,\alpha,\beta)$ *query can be solved using* $\tilde{O}(\alpha^{-2}\beta^{-2}\log(1/\delta))$ *examples. In both cases, the error probability is less than* δ*.*

5.2.1 The Red-Blue algorithm

State merging algorithms were initially proposed for the regular language inference problem. Gold [27] had shown that regular languages are identifiable in the limit from positive and negative data, but consistency with the input sample was not guaranteed when the sample does not contain some crucial information —the so called *characteristic set*—. Oncina and García [34] proposed a state merging algorithm for inferring regular languages that overcomes this drawback. The algorithm always returns a dfa that is data consistent and, provided a complete presentation is given, achieves identification in the limit.

Carrasco and Oncina in [14] adapted the dfa state merging learning algorithm to the stochastic setting. The new algorithm, so-called ALERGIA, was shown to identify in the limit any pdfa provided stochastic examples of the target are given. We present here a version of ALERGIA in the query model we have just introduce that considers statistical and L_∞-queries. We rename this version as the Red-Blue algorithm.

Given an integer m as input, Red-Blue starts by building a prefix tree acceptor representing significant prefixes by calling the Pta function on parameters m and λ, see Figures 2 and 3. On these values, Pta returns a prefix tree dfa whose leaves at level j correspond to prefixes of probability at least $2^j/m$. Thus, the resulting tree dfa has depth at most $\lceil \log m \rceil$. In order to decide whether a prefix has enough probability to be represented, Pta makes statistical queries.

Once the the tree acceptor is built, Red-Blue colors the initial state as red and its direct descendants as blue, leaving other states uncolored, and starts merging compatible states. Two states are considered merge-compatible when a call to the DIFF_∞^D oracle returns a numerical value not exceeding $1/m$, meaning that there is not strong evidence that they define different distributions. Specifically, we define for each state q a prefix free subset $H[q]$ consisting of words x such that $\tau(q_0,x) = q$ and for any prefix y of x it holds $x = y$ or $\tau(q_0,y) \neq q$. Asking the query $\mathsf{DIFF}_\infty^D(H[q_1],H[q_2],\alpha,\alpha)$ we get information about how different the distributions defined by states q_1 and q_2 are.

The merging flow proceeds as follows. For an arbitrarily chosen blue state, either there is a red state that is merge-compatible with it or no red state is. In the first case, both states are merged and the result colored red. This may introduce nondeterministic choices, which are eliminated by further merging in the merge procedure. Note that every merge always involves a blue state. On the other hand, if there is no merge-compatible red state for this blue state, it is promoted to red and new blue

```
input: integer $m$
output: pdfa $H$
algorithm Red-Blue
    $H \leftarrow \mathrm{Pta}(m,\lambda)$
    Red $\leftarrow \{q_\lambda\}$;
    Blue $\leftarrow \{\tau(q_\lambda,\sigma),\sigma \in \Sigma\}$
    Let $\alpha \leftarrow 1/m$
    while Blue $\neq \emptyset$
        pick some state $b \in$ Blue
        if there is $r \in$ Red with $\mathsf{DIFF}^D_\infty(H[r],H[b],\alpha,\alpha) \leq \alpha$
            $H \leftarrow \mathrm{Merge}(r,b,H)$
        else Red $\leftarrow$ Red $\cup \{b\}$
        Blue $\leftarrow$ Blue $- \{b\} \cup \{\tau(b,\sigma) | \sigma \in \Sigma$ and $\tau(b,\sigma)$ is uncolored$\}$
    for $q \in H$
        $\gamma(q,\xi) \leftarrow \mathsf{SQ}^D(H[q],\alpha)/\mathsf{SQ}^D(H[q]\Sigma^\star,\alpha)$
        for $\sigma \in \Sigma$ such that $\tau(q,\sigma)$ is defined
            $\gamma(q,\sigma) \leftarrow \mathsf{SQ}^D(H[q]\sigma\Sigma^\star,\alpha)/\mathsf{SQ}^D(H[q]\Sigma^\star,\alpha)$
return $H$
```

Fig. 2 Red-Blue algorithm

```
input: integer $m$ and string $w \in \Sigma^\star$
output: tree dfa $H$
algorithm Pta
    Set a initial state $q_w$ of $H$
    Let $\alpha \leftarrow 1/m$
    for each $\sigma \in \Sigma$
        if $\mathsf{SQ}^D(w\sigma\Sigma^\star,\alpha) > 3\alpha$
            $H_\sigma \leftarrow \mathrm{Pta}(\lceil m/2 \rceil, w\sigma)$
            Set a new transition $\tau(q_w,\sigma) = q_{w\sigma}$
return $H$
```

Fig. 3 Pta algorithm

states are generated from it. When the set of blue states is empty, Red-Blue stops merging and a pdfa is returned by setting transition probabilities of H according to prefix probability approximations obtained by issuing statistical queries.

Figures 2 and 4 show the Red-Blue and the merge algorithms. Let H be the resulting automaton after some iterations of Red-Blue. Red states of H and transitions between them represent the the part of the graph we trust as correct, based on processed information. An invariant of the algorithm is that non-red states are always roots of trees in H and blue states are always direct successors of a red state.

```
input: dfa $H$ and states $q$ and $q'$ of $H$
output: dfa
algorithm Merge
    Replace each occurrence $q'$ in the description of $H$ by $q$
    while $H$ contains a nondeterministic transition
        Let $p$ and $p'$ be target states of a nondeterministic choice
        Merge$(p, p', H)$
    return $H$
```

Fig. 4 Merge function

Assuming unit time for oracle calls, the complexity of Red-Blue is $O(m^2)$. Moreover, if m is large enough that every state or transition of D has a counterpart in the tree dfa returned by the Pta algorithm and such that $2/m$ is less than the distinguishability of the target machine, the algorithm on input m learns a pdfa hypothesis whose graph agrees with the target graph. As probability estimations of states and transitions will improve as m is larger, we have:

Theorem 5. *The Red-Blue algorithm learns every pdfa in the identification in the limit paradigm.*

This result appears (for the equivalent ALERGIA algorithm) in [13, 14]. But from this point it is easy to argue that Red-Blue also learns in the PAC model if the complexity of the target pdfa is taken to number of states times alphabet size times the inverse of the distinguishability.

5.2.2 The Safe-Candidate algorithm

We describe a variant of the algorithm in [16], which was the first one to claim any formal PAC guarantee. Our version fits the query framework in [6, 17], which makes the exposition easier and simplifies correctness arguments.

The main differences of the presentation below with respect to the description in [16] are two. First, as said, the algorithm gets information on the target distribution asking statistical and L_∞-queries defined above instead of analyzing a sample. Second, it guarantees a good L_1 approximation, a weaker requirement than the good relative entropy approximation guaranteed in [16]. The latter choice avoids some rather subtle points in [16] by the fact that L_1 is a true distance unlike KL.

The Safe-Candidate algorithm works in two stages. In the first one, it builds a transition graph that is isomorphic to the target subgraph formed by important elements —these are sates and transitions whose probability of being visited while generating a random string is above a threshold defined by the input parameters—. The construction uses both statistical and L_∞-queries. The second stage consists of converting the learned graph into a pdfa by estimating the transition and stopping

```
input: $n, \mu, \Sigma, \varepsilon$, oracles $\mathsf{DIFF}_\infty^D$ and $\mathsf{SQ}^D$
output: A graph $H$
algorithm Safe-Candidate
$\alpha \leftarrow \mu/2$; $\beta \leftarrow \varepsilon/24n|\Sigma|$
initialize the graph $H$ with a safe $q_\lambda$ and candidates $q_\lambda^\sigma$ for $\sigma \in \Sigma$
while $H$ has candidates
    choose a candidate $q$ maximizing $\mathsf{SQ}^D(H[q]\Sigma^\star, \beta)$
    foreach safe $q'$
        make a call to the oracle $\mathsf{DIFF}_\infty^D(H[q], H[q'], \alpha, \beta)$
        if the answer is '?'
            remove $q$ from the candidate list
            break
        if the answer $\hat{\mu} < \mu/2$
            merge $q$ and $q'$
            remove $q$ from the candidate list
            break
    if $q$ is still a candidate
        promote $q$ to safe
        add candidates $q^\sigma$ for each $\sigma \in \Sigma$
```

Fig. 5 Safe-Candidate graph construction

probabilities corresponding to each state. This is similar to the estimation step in the Red-Blue algorithm performed once the set of blue states is empty, see Fig. 2. In this stage, only statistical queries are necessary.

Figure 5 shows the code for the graph construction stage. The algorithm keeps two set on nodes, the set of safe nodes and the set of candidates. Safe nodes and transitions between them are known to be correct. Initially, there is only a safe state q_λ corresponding to the empty word and one candidate q_λ^σ for each $\sigma \in \Sigma$. Each candidate represents a still unknown transition in the graph H. In each iteration, statistical queries are performed to choose the most informative candidate q and, after that, L_∞-queries are issued in order to decide if the candidate is either insignificant at all, it is an already known safe node q' or it is a new safe one. In the latter case, when a candidate is promoted to safe, new candidates are considered representing undefined transitions leaving the new safe node. The algorithm finishes when there are no candidates left. An inductive reasoning proves that the resulting graph is isomorphic to the target subgraph containing significant states and transitions.

The following theorem summarizes the performance of Safe-Candidate.

Theorem 6. *Let M denote an n-state pdfa computing a distribution D over $\Sigma^\star$, L denote the expected length of D, and π be the smallest nonzero stopping probability in M. Then the execution of Safe-Candidate on a sample from D satisfies:*

1. *It runs in time $\mathsf{poly}(n, |\Sigma|, \log(1/\pi))$.*

2. *It asks* $O(n^2|\Sigma|^2\log(n/\pi))$ *statistical queries with tolerance* $\tilde{\Omega}(\varepsilon^3\pi^2/n^3|\Sigma|^3L)$.
3. *It asks* $O(n^2|\Sigma|)$ L_∞*-queries with tolerance* $\Omega(\mu)$ *and threshold* $\Omega(\varepsilon/n|\Sigma|)$.
4. *It returns a pdfa H such that* $\mathrm{L}_1(D,H)\leq\varepsilon$.

A short and complete proof of this theorem is in [17]. A similar theorem without any dependence on L and π is shown in [35] but the proof is more complex. The proof in [16] that shows PAC learnability under the KL-divergence measure is much longer.

6 Learning PFA

In this section we discuss algorithms having formal guarantees of learning the whole class of pfa. Similarly to the PDFA case, we first give an algorithm that has access to the target function as oracle, and can exactly learn the class of Weighted Automata when the right set of prefixes and suffixes is provided. Then, we specialize the algorithm to pfa and the case in which the input is a randomly drawn finite sample. We discuss the solutions given by Denis *et al.* [22] on the one hand and, on the other, by Mossel and Roch [33], Hsu *et al.* [29], and Bailly *et al.* [4]. The latter leads to the spectral method, which we expose in more detail following mostly the presentation in [9]. We finally mention a few of the most recent works extending the spectral method in several directions.

6.1 An Oracle Algorithm for Learning WA

The algorithm in Figure 6 encapsulates the main idea for learning general Weighted Automata in the oracle model. It resembles the algorithm given by Beimel *et. al* [11] that learned instead from Evaluation and Equivalence queries.

There are two nondeterministic steps in this algorithm: One is the choice of sets X and Y; like for pdfa, we do not discuss how to choose them in the oracle model. The other one is the choice of a factoring QR of H', which represents the choice of an arbitrary basis for the rows of the Hankel matrix. The construction of the wa in the proof of Theorem 3 is the special case in which $R=H'$ and Q the identity, but it is easy to see that the correctness of the special case implies the correctness of the general QR case. Indeed, the value of the automaton for any word w is

$$\begin{aligned}&\alpha_0^T T_w \alpha_\infty\\ &=(H'[\lambda,:]R^{-1})(Q^{-1}H'_{w_1}R^{-1})(Q^{-1}H'_{w_2}R^{-1})\dots(Q^{-1}H'_{w_m}R^{-1})Q^{-1}H'[:,\lambda]\\ &=H'[\lambda,:]H'^{-1}H'_{w_1}H'^{-1}H'_{w_2}\dots H'_{w_m}H'^{-1}H'[:,\lambda]\end{aligned}$$

which is the value computed for $R=H'$ and Q the identity. The reason for the more general presentation will be clear later. It is also clear that the algorithm runs in time

1. Choose any two finite sets $X,Y \subseteq \Sigma^\star$ with $\lambda \in X \cap Y$, in some way not specified by the algorithm;
2. Build the submatrix $H = H_f[X \cup X\Sigma, Y]$ of H_f by asking oracle queries to f;
3. Find two minimal subsets $X' \subseteq X$, $Y' \subseteq Y$ such that $\lambda \in X'$ and $H' = H[X', Y']$ has the same rank as H; note that $|X'| = |Y'|$, say n, and H' has full rank;
4. Let $Q, R \in \mathbb{R}^{n\times n}$ be any two matrices factoring H', i.e., $H' = QR$;
5. For each symbol a, let $H'_a = H_f[X'a, Y'] = H_f[X', aY']$;
6. Build a wa from Q, R, and $\{T_a\}_a$ as follows.

$$\begin{aligned}\alpha_0^T &= H'[\lambda,:]R^{-1}\\ \alpha_\infty &= Q^{-1}H'[:,\lambda]\\ T_a &= Q^{-1}H'_aR^{-1}\end{aligned}$$

Fig. 6 Learning wa with an oracle

polynomial in Σ and the sums of lengths of strings in $X \cup Y$. Following the proof of Theorem 3 we have:

Theorem 7. *Let f be computed by some wa. If the cardinality of X' is at least the rank of H_f then the wa built in this way computes f.*

Proof. The algorithm defines matrices T_a by $H'_a = T_aH'$; they are uniquely defined as H' has full rank. Furthermore, since the rank of H' is that of the whole Hankel matrix H of f, these matrices must also satisfy $H_a = T_aH$. Therefore, the automaton constructed by the algorithm is exactly the one built in the proof of Theorem 3, which by the theorem computes f. □

Consider now the adaptation of the algorithm to the case in which f is a probability distribution and we are given a finite sample S. As in the pdfa case, we take $X = \mathit{prefixes}(S)$, $Y = \mathit{suffixes}(S)$, and then create an approximation $\hat{H}$ of H by $\hat{H}[x,y]$ = empirical probability of xy in S. We know that the unperturbed H has rank at most n, the number of states of the smallest wa for f but, because rank is so fragile under perturbations, $\hat{H}$ will probably have maximal rank, even if $|S|$ is large.

The solution taken by [22] can be intuitively described as follows: Compute a subset $X' \subseteq X$ such that $|X'| = n$ and every row of $\hat{H}[X,:]$ is "close to" a linear combination of rows indexed by X'. For sufficiently small choice of "close to" (depending on the distribution), X' will be as in the algorithm above, and lead to a correct solution.

The solution taken by e.g., [33, 29, 4], which leads to the spectral method, is less combinatorial and more algebraic, less local and more global, and can be phrased as follows: Let us instead find a matrix H' that 1) is easy to compute 2) has the same dimensions as H, but rank at most n, and 3) is "as close as possible" to $\hat{H}$ under some metric, with this rank constraint. One particular way of computing such H' (among, perhaps, other possibilities) is the spectral method described next.

6.2 The Spectral Method

An important tool for the spectral method is the *Singular Value Decomposition* theorem; see e.g. [40].

Theorem 8. *(SVD theorem) Let $H \in \mathbb{R}^{p\times q}$. There are matrices $U \in \mathbb{R}^{p\times p}$, $D \in \mathbb{R}^{p\times q}$ and $V \in \mathbb{R}^{q\times q}$ such that:*

- *$H = UDV^T$*
- *U and V are orthonormal: $U^TU = I \in \mathbb{R}^{p\times p}$ and $V^TV = I \in \mathbb{R}^{q\times q}$*
- *D is a diagonal matrix of non-negative real numbers.*

The diagonal values of D, denoted $\sigma_1, \sigma_2, \ldots$, are the *singular values* of H, and column vectors of U are its *left singular vectors*. It follows that $\text{rank}(A) = \text{rank}(D)$ is the number of non-zero singular values. W.l.o.g. by rearranging rows and columns, the diagonal values in D are nondecreasing, i.e. $\sigma_1 \geq \sigma_2 \geq \ldots$. The SVD decomposition can be computed in time $O(pq^2)$.

The Frobenius norm of a matrix H is

$$\|H\|_F = \left(\sum_{i,j} H[i,j]^2\right)^{1/2}.$$

As this norm is invariant by unitary products, the square of the Frobenius norm of H is the sum of the squares of its singular values. It follows that from the singular value decomposition $H = UDV$ of H, we can compute a low rank approximation: Fix $n \leq \text{rank}(H)$, and define H'_n as

$$H'_n = \left(\begin{array}{c|c|c|c} u_1 & \ldots & u_n & \mathbf{0} \end{array}\right) \left(\begin{array}{ccc|c} \sigma_1 & & & \\ & \ddots & & \\ & & \sigma_n & \\ \hline & & & \mathbf{0} \end{array}\right) \left(\begin{array}{c} v_1 \\ \hline \vdots \\ \hline v_n \\ \hline \mathbf{0} \end{array}\right).$$

The following is the crucial fact:

Fact. H'_n has rank n and minimizes $\|H - G\|_F$ among all rank-n matrices G.

Now we would like to use H'_n to find a full rank submatrix H' of H, since H'_n will not in general have full rank and cannot be inverted. Alternatively, there is a notion of *pseudoinverse* matrix that satisfies what we need for the algorithm, and is easily computable from the SVD decomposition.

The *Moore-Penrose pseudoinverse* of A, denoted A^+, admits many different definitions. The most algorithmic one is perhaps:

- If $A \in \mathbb{R}^{p\times q}$ is a diagonal matrix, $A^+ \in \mathbb{R}^{q\times p}$ is formed by transposing A, and taking the inverse of each non-zero element.

```
1. get $n$ and sample $S$;
2. $X = \mathit{preffixes}(S)$; $Y = \mathit{suffixes}(S)$;
3. define $H[X,Y] \in \mathbb{R}^{p\times q}$ and set $H[x,y]$ = empirical probability of $xy$;
4. define $H_a[X,Y] \in \mathbb{R}^{p\times q}$ and set $H_a[x,y]$ = empirical probability of $xay$;
5. Let $QR$ be a rank-$n$ factorization of $H$, that is:
    • $Q \in \mathbb{R}^{p\times n}$ and $R \in \mathbb{R}^{n\times q}$, both having rank $n$,
    • $H = QR$,
   for instance, take $Q$ to be the first $n$ left singular vectors of $H$;
6. output the WA $M$ such that
    $\alpha_0^T = H[\lambda,:]R^+, \quad \alpha_\infty = Q^+H[:,\lambda], \quad T_a = Q^+H_aR^+$
```

Fig. 7 Spectral learning of pfa

- In the general case, if $A = UDV^T$ then $A^+ = VD^+U^T$.

Some of its many interesting properties are:

- In general, $AA^+ \neq I$ and $A^+A \neq I$.
- But if A is invertible, $A^+ = A^{-1}$.
- If columns of A are independent, $A^+A = I \in \mathbb{R}^{q\times q}$.
- If rows of A are independent, $AA^+ = I \in \mathbb{R}^{p\times p}$.

With this artillery in place, the spectral method for learning probability distributions generated by wa is described in Figure 7. The following PAC result was shown in [29] and reformulated in [9].

Theorem 9. *[29, 9] Let S be a sample of a probability distribution D such that H_D has rank n. Let σ_n be the nth largest singular value of H_D, and M the wa produced by the algorithm in Figure 7 on input S. There is a polynomial p such that if $|S| \geq p(n,|\Sigma|,1/\sigma_n,1/\varepsilon,\log(1/\delta))$, with probability at least $1-\delta$:*

$$\sum_{|x|=t} |D(x) - M(x)| < \varepsilon.$$

Observe that $\sigma_n \neq 0$ if and only if $\mathrm{rank}(H_P) \geq n$, so $1/\sigma_n$ makes sense as a complexity parameter. Observe also that the output of the algorithm is not necessarily a pfa, even under the assumption that P is a probability distribution. It may assign negative values to some strings, and not add up to exactly 1. However, by the theorem, such oddities tend to disappear as the sample size grows. Converting a wa known to compute (or approximate) a probability distribution to a pfa is, in general, uncomputable. The problem is discussed in detail in [22]. Both [33] and [2] give partial solutions by considering somewhat restricted machine models.

6.3 Variations and Implementation of the Spectral Method

Several variations of the spectral method have been introduced in order to improve its sample-efficiency or to extend it to wider settings. We point out below a couple of recent proposals.

Let D be a distribution on $\Sigma^\star$. Derived from D, we define function D_p assigning to each string x the probability of the set $x\Sigma^\star$, i.e. $D_p(x) = \sum_y D(xy)$. Similarly, we also consider function D_s that on input x evaluates to the expected number of times x appears in a random string w. It turns out that if any of these three functions has wa, then all three functions have wa. Moreover, wa descriptions can be obtained from original wa parameters of one of them. The following lemma is shown in [17]:

Lemma 1. *Let $\langle \{T_\sigma\}_{\sigma\in\Sigma^\star}, \alpha_0, \alpha_\infty\rangle$ be a wa and define $S = \sum_\sigma T_\sigma$, $\tilde{\alpha}_0 = \alpha_0^T (I-S)^{-1}$ and $\tilde{\alpha}_\infty = (I-S)^{-1}\alpha_\infty$. Then the following are equivalent*

1. *$\langle \{T_\sigma\}_{\sigma\in\Sigma^\star}, \alpha_0, \alpha_\infty\rangle$ computes D.*
2. *$\langle \{T_\sigma\}_{\sigma\in\Sigma^\star}, \alpha_0, \tilde{\alpha}_\infty\rangle$ computes D_p.*
3. *$\langle \{T_\sigma\}_{\sigma\in\Sigma^\star}, \tilde{\alpha}_0, \tilde{\alpha}_\infty\rangle$ computes D_s .*

Thus, besides using statistics of full strings in order to approximate the Hankel matrix from a sample we can also try to use statistics from prefixes and substrings in order to learn functions D_p and D_s. Experimentally, this yields more sample-efficient algorithms.

The use of statistics on prefixes instead of full strings in the spectral method was already proposed by Hsu et al. [29]. Later, Luque *et al.* [32] take advantage of substring statistics when applying the spectral method to learn non-deterministic split head-automata grammars, a hidden-state formalism for dependency parsing.

Recently, the spectral method in combination with matrix completion techniques has been also applied to a more general learning setting [8]. Here, the learning problem is to infer a weighted automaton from a sample of labeled examples but, in contrast with the standard paradigm, the sample is provided according to an arbitrary unknown distribution. Note that, for this type of learning problem, it is not guaranteed that a full approximation of a convenient Hankel submatrix can be achieved. This is because the input sample can lack of information for many submatrix entries and now it can not be assumed that the wa function value must be close to 0 there, as one can in the standard probability setting. In [8], matrix completion techniques are used to fill the gaps in the Hankel submatrix and then the spectral method is used to infer the target wa. They prove formal learning guarantees for some of the completion techniques under mild conditions on the distribution.

7 Future Work

Let us mention a few open questions or future lines of research. Concerning the spectral method, it is clearly in an early stage and further extensions and applications will keep appearing. Making it generally practical and competitive is certainly of interest. Most spectral methods produce weighted automata with negative transition values when learning pfa; this may somewhat hinder their application in contexts where interpretability of the learned model is important.

At a more theoretical level, we would like to find some geometric or algebraic interpretation of pdfa distinguishability; this might explain its role in pdfa learning as a particular case of the spectral values that come up in learning pfa.

As mentioned, it is known that pdfa cannot exactly compute all distributions computed by pfa [22]. But, to our knowledge, it is not known whether pdfa can reasonably approximate distributions computed by pfa, say with a number of states polynomial in $1/\varepsilon$ for the desired approximation ε in the L_1 distance (a rather direct result for L_∞ is given in [26], which extends to every L_p for $p > 1$). If such approximability is true, then it may be possible to transfer PAC learnability results for pdfa to pfa with a polynomial overhead.

Acknowledgements

This work is partially supported by MICINN projects TIN2011-27479-C04-03 (BASMATI) and TIN-2007-66523 (FORMALISM), by SGR2009-1428 (LARCA). We thank the chairs of ICGI 2012 for the invitation to present a preliminary version of this work as tutorial. We particularly thank the reviewer of this version for a thorough and useful work.

References

1. Naoki Abe and Manfred K. Warmuth. On the computational complexity of approximating distributions by probabilistic automata. *Machine Learning*, 9:205–260, 1992.
2. Animashree Anandkumar, Daniel Hsu, and Sham M. Kakade. A method of moments for mixture models and hidden Markov models. In *25th Annual Conference on Learning Theory (COLT); Journal of Machine Learning Research - Proceedings Track - COLT 2012*, volume 23, pages 33.1–33.34, 2012.
3. Dana Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1987.
4. Raphaël Bailly, François Denis, and Liva Ralaivola. Grammatical inference as a principal component analysis problem. In *26th Intl. Conf. on Machine Learning (ICML)*, page 5. ACM, 2009.
5. Borja Balle, Jorge Castro, and Ricard Gavaldà. Learning pdfa with asynchronous transitions. In *10th Intl. Coll. on Grammatical Inference (ICGI)*, volume 6339 of *Lecture Notes in Computer Science*, pages 271–275. Springer, 2010.
6. Borja Balle, Jorge Castro, and Ricard Gavaldà. A lower bound for learning distributions generated by probabilistic automata. In *21st Intl. Conf. on Algorithmic Learning Theory (ALT)*, volume 6331 of *Lecture Notes in Computer Science*, pages 179–193. Springer, 2010.
7. Borja Balle, Jorge Castro, and Ricard Gavaldà. Bootstrapping and learning pdfa in data streams. In *11th Intl. Conf. on Grammatical Inference (ICGI)*, volume 21 of *JMLR Workshop and Conf. Proceedings*, pages 34–48, 2012.

8. Borja Balle and Mehryar Mohri. Spectral learning of general weighted automata via constrained matrix completion. In *Neural Information Processing Systems (NIPS)*, 2012.
9. Borja Balle, Ariadna Quattoni, and Xavier Carreras. Local loss optimization in operator models: A new insight into spectral learning. In *29th Intl. Conf. on Machine Learning (ICML)*, 2012.
10. L. E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *Ann. Math. Statist.*, 41(1):164–171, 1970.
11. Amos Beimel, Francesco Bergadano, Nader H. Bshouty, Eyal Kushilevitz, and Stefano Varricchio. Learning functions represented as multiplicity automata. *J. ACM*, 47(3):506–530, 2000.
12. J.W. Carlyle and A. Paz. Realization by stochastic finite automaton. *J. Comput. Syst. Sci.*, 5:26–40, 1971.
13. Rafael C. Carrasco and José Oncina. Learning stochastic regular grammars by means of a state merging method. In *2nd Intl. Coll. on Grammatical Inference (ICGI)*, volume 862 of *Lecture Notes in Computer Science*, pages 139–152. Springer, 1994.
14. Rafael C. Carrasco and José Oncina. Learning deterministic regular grammars from stochastic samples in polynomial time. *Informatique Théorique et Applications*, 33(1):1–20, 1999.
15. Jorge Castro and Ricard Gavaldà. Towards feasible PAC-learning of probabilistic deterministic finite automata. In *9th Intl. Coll. on Grammatical Inference (ICGI)*, volume 5278 of *Lecture Notes in Computer Science*, pages 163–174. Springer, 2008.
16. Alexander Clark and Franck Thollard. PAC-learnability of probabilistic deterministic finite state automata. *Journal of Machine Learning Research*, 5:473–497, 2004.
17. Borja de Balle Pigem. *Learning Finite-State Machines. Algorithmic and Statistical Aspects*. PhD thesis, Universitat Politècnica de Catalunya, 2013.
18. Colin de la Higuera. *Grammatical Inference Learning Automata and Grammars*. Cambridge University Press, 2010.
19. Colin de la Higuera and José Oncina. Learning stochastic finite automata. In *7th Intl. Coll. on Grammatical Inference (ICGI)*, volume 3264 of *Lecture Notes in Computer Science*, pages 175–186. Springer, 2004.
20. Colin de la Higuera and Franck Thollard. Identification in the limit with probability one of stochastic deterministic finite automata. In *5th Intl. Coll. on Grammatical Inference (ICGI)*, volume 1891 of *Lecture Notes in Computer Science*, pages 141–156. Springer, 2000.
21. François Denis and Yann Esposito. Learning classes of probabilistic automata. In *17th Annual Conference on Learning Theory (COLT)*, volume 3120 of *Lecture Notes in Computer Science*, pages 124–139. Springer, 2004.
22. François Denis, Yann Esposito, and Amaury Habrard. Learning rational stochastic languages. In *19th Annual Conference on Learning Theory (COLT)*, volume 4005 of *Lecture Notes in Computer Science*, pages 274–288. Springer, 2006.
23. Pierre Dupont and Juan-Carlos Amengual. Smoothing probabilistic automata: An error-correcting approach. In *5th Intl. Coll. on Grammatical Inference (ICGI 2000)*, volume 1891 of *Lecture Notes in Computer Science*, pages 51–64. Springer, 2000.
24. Pierre Dupont, François Denis, and Yann Esposito. Links between probabilistic automata and hidden Markov models: probability distributions, learning models and induction algorithms. *Pattern Recognition*, 38(9):1349–1371, 2005.
25. M. Fliess. Matrices de Hankel. *J. Math. Pures Appl.*, 53:197–222, 1974. (Erratum in vol. 54, 1975.).
26. Ricard Gavaldà, Philipp W. Keller, Joelle Pineau, and Doina Precup. PAC-learning of Markov models with hidden state. In *17th European Conf. on Machine Learning (ECML)*, volume 4212 of *Lecture Notes in Computer Science*, pages 150–161. Springer, 2006.
27. E. Mark Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.
28. Omri Guttman, S. V. N. Vishwanathan, and Robert C. Williamson. Learnability of probabilistic automata via oracles. In *16th Intl. Conf. on Algorithmic Learning Theory (ALT)*, pages 171–182, 2005.

29. Daniel Hsu, Sham M. Kakade, and Tong Zhang. A spectral algorithm for learning hidden Markov models. In *22nd Annual Conference on Learning Theory (COLT)*. ACM, 2009.
30. Michael J. Kearns, Yishay Mansour, Dana Ron, Ronitt Rubinfeld, Robert E. Schapire, and Linda Sellie. On the learnability of discrete distributions. In *26th Annual ACM Symp. on Theory of Computing (STOC)*, pages 273–282. ACM, 1994.
31. Christopher Kermorvant and Pierre Dupont. Stochastic grammatical inference with multinomial tests. In *6th Intl. Coll. on Grammatical Inference (ICGI)*, volume 2484 of *Lecture Notes in Computer Science*, pages 149–160. Springer, 2002.
32. Franco M. Luque, Ariadna Quattoni, Borja Balle, and Xavier Carreras. Spectral learning for non-deterministic dependency parsing. In *13th Conf. of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 409–419. The Association for Computer Linguistics, 2012.
33. Elchanan Mossel and Sébastien Roch. Learning nonsingular phylogenies and hidden Markov models. In *37th Annual ACM Symp. on Theory of Computing (STOC)*, pages 366–375. ACM, 2005.
34. José Oncina and Pedro García. Identifying regular languages in polynomial. In *Advances in Structural and Syntactic Pattern Recognition*, pages 99–108. World Scientific, 1992.
35. Nick Palmer and Paul W. Goldberg. PAC-learnability of probabilistic deterministic finite state automata in terms of variation distance. *Theor. Comput. Sci.*, 387(1):18–31, 2007.
36. Dana Ron, Yoram Singer, and Naftali Tishby. The power of amnesia: Learning probabilistic automata with variable memory length. *Machine Learning*, 25(2-3):117–149, 1996.
37. Steven Rudich. Inferring the structure of a Markov chain from its output. In *26th Annual Symp. on Foundations of Computer Science (FOCS)*, pages 321–326, 1985.
38. Marcel P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4:245–270, 1961.
39. Cosma Rohilla Shalizi and Kristina Lisa Shalizi. Blind construction of optimal nonlinear recursive predictors for discrete sequences. In *20th Conf. on Uncertainty in Artificial Intelligence (UAI)*, pages 504–511, 2004.
40. Gilbert Strang. *Introduction to Linear Algebra, 4th edition*. Wellesley-Cambridge Press and SIAM, 2009.
41. Sebastiaan Terwijn. On the learnability of hidden Markov models. In *6th Intl. Coll. on Grammatical Inference (ICGI)*, volume 2484 of *Lecture Notes in Computer Science*, pages 261–268. Springer, 2002.
42. Franck Thollard. Improving probabilistic grammatical inference core algorithms with post-processing techniques. In *18th Intl. Conf. on Machine Learning (ICML 2001)*, pages 561–568, 2001.
43. Franck Thollard, Pierre Dupont, and Colin de la Higuera. Probabilistic dfa inference using Kullback-Leibler divergence and minimality. In *17th Intl. Conf. on Machine Learning (ICML)*, pages 975–982. Morgan Kaufmann, 2000.
44. Leslie G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, 1984.
45. Enrique Vidal, Franck Thollard, Colin de la Higuera, Francisco Casacuberta, and Rafael C. Carrasco. Probabilistic finite-state machines - part i. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(7):1013–1025, 2005.
46. Enrique Vidal, Franck Thollard, Colin de la Higuera, Francisco Casacuberta, and Rafael C. Carrasco. Probabilistic finite-state machines - part ii. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(7):1026–1039, 2005.
47. Lloyd R. Welch. Hidden Markov Models and the Baum-Welch Algorithm. *IEEE Information Theory Society Newsletter*, 53(4), 2003.

Appendix: The Baum-Welch Method

The Baum-Welch algorithm [10] is one of the most popular methods to infer a hidden Markov model from observed data. Despite the fact that there are no bounds on convergence time and that it may get trapped in local optima, it is intuitively attractive since it has a clear focus — maximizing the observed data likelihood — and performs a simple step-by-step hill climbing progress to this goal. However, we think that as new techniques based on spectral methods progress and gain popularity, Baum-Welch may lose its status as first option. Theoretically, spectral methods will obtain global optima, come with performance guarantees in time and accuracy, and tend to work faster at least on large samples.

The Baum-Welch algorithm starts by guessing some hidden Markov model; this is frequently done using problem-specific heuristics. Provided with a sequence of data observations $x = x_1 \dots x_m$, the algorithm continues by iterating a process where parameters of the last built model H are updated according to a posteriori state and transition probability values. Iteration finishes either when a parameter convergence criterion is achieved or there is a loss of prediction accuracy. The learning algorithm is shown in Figure 8. Specifically, let $H = \langle \{T_a\}_{a\in\Sigma}, \alpha_0 \rangle$ the last hypothesis built. Backward and forward probability vectors for $t = 0 \dots m$ are, respectively:

$$\begin{aligned} \beta_t^T &= \alpha_0^T T_{x_1} \cdots T_{x_t}, \\ \phi_t &= T_{x_{t+1}} \cdots T_{x_m} \mathbf{1}. \end{aligned}$$

Component j of the backward vector β_t is the probability of generating prefix $x_1 \dots x_t$ and being in state j just after emitting x_t, i.e. $\beta_t[j] = \Pr[x_1 \dots x_t \wedge S(t) = j]$. On the other hand, component j of the forward vector ϕ_t is the probability of emitting suffix $x_{t+1} \dots x_m$ from state j, $\phi_t[j] = \Pr[x_{t+1} \dots x_m | S(t) = j]$. Let $S(t)$ denote the state at time t and $O(t)$ the observation at time (whose realization is thus x_t). Given data $x = x_1 \dots x_m$, the a posteriori state visit probability of state j is:

$$\alpha_0'[j] = \frac{1}{m+1} \sum_{t=0}^{m} \Pr[S(t) = j | x] = \frac{1}{m+1} \sum_{t=0}^{m} \frac{\beta_t[j]\phi_t[j]}{\beta_t^T \phi_t} = \frac{\sum_{t=0}^{m} \beta_t[j]\phi_t[j]}{(m+1)(\alpha_0 T_x \mathbf{1})}. \quad (1)$$

Similarly, the a posteriori probability of transition a from state i to j is:

$$T_a'[i,j] = \frac{\sum_{t|x_t=a} \xi_t^{i,j}}{\sum_{t=1}^{m} \xi_t^{i,j}} \quad (2)$$

where

$$\xi_t^{i,j} = \Pr[S(t-1) = i \wedge S(t) = j \wedge O(t) = x_t | x] = \frac{\beta_{t-1}[i] T_{x_t}[i,j] \phi_t[j]}{\alpha_0 T_x \mathbf{1}}.$$

Thus, $\xi_t^{i,j}$ denotes the probability that provided observed data x, state i is reached just after processing length $t-1$ prefix of x and x_t moves from state i to state j.

```
input: data observation $x = x_1 \dots x_m$ and some initial HMM guess $H = \langle \{T_a\}_{a\in\Sigma}, \alpha_0 \rangle$
output: updated $H$, locally maximizing sample likelihood
algorithm Baum-Welch
repeat
    compute backward $\beta_t$ and forward $\phi_t$ probability vectors for $t = 0 \dots m$
    compute a posteriori state visit and transition probabilities $\alpha'_0$ and $\{T'_a\}_{a\in\Sigma}$
    $H \leftarrow \langle \{T'_a\}_{a\in\Sigma}, \alpha'_0 \rangle$
until stopping condition
```

Fig. 8 The Baum-Welch algorithm

The following theorem shows that the iterated updating procedure in the Baum-Welch algorithm either increases the sample likelihood or, at local maxima, keeps it unchanged. We follow the presentation in [47].

Theorem 10. *[10, 47] Let $x = x_1 \dots x_m$ be a sample and let $H = \langle \alpha_0, \{T_a\}_{a\in\Sigma} \rangle$ and $H' = \langle \alpha'_0, \{T'_a\}_{a\in\Sigma} \rangle$ be the hidden Markov models defined above. Then, $H'(x) \geq H(x)$ with equality at local maxima.*

Proof. (sketch) Let $s = s_0 \dots s_m$ be a sequence of states in machine H and consider conditional distributions $H(s|x)$ and $H'(s|x)$. Starting from the KL divergence formula and expanding conditional probabilities, it is easy to derive the relations

$$\begin{aligned} 0 &\leq \mathrm{KL}(H(\cdot|x), H'(\cdot|x)) \\ &= \sum_s H(s|x) \log \frac{H(s|x)}{H'(s|x)} = \log \frac{H'(x)}{H(x)} + \sum_s \frac{H(x \wedge s)}{H(x)} \log \frac{H(x \wedge s)}{H'(x \wedge s)}. \end{aligned}$$

Defining function Q as

$$Q(x, H, H') \doteq \sum_s H(x \wedge s) \log H'(x \wedge s),$$

the last inequality can be rearranged to show that

$$\frac{Q(x,H,H') - Q(x,H,H)}{H(x)} \leq \log \frac{H'(x)}{H(x)}.$$

Thus, $H'(x) > H(x)$ when $Q(x,H,H') > Q(x,H,H)$. We obtain a hill climbing procedure by finding H' maximizing the $Q(x,H,\cdot)$ function. Using Lagrange's method to find critical points of Q subject to stochastic constraints (H' must define a probability function) results in values for α'_0 and $\{T'_a\}_{a\in\Sigma}$ defining H' as the ones displayed in Equations (1) and (2). □